springer tracts in advanced robotics 56

Martin Buehler
Karl Iagnemma
Sanjiv Singh
(Eds.)

The DARPA Urban Challenge

Autonomous Vehicles in City Traffic

With Forewords by
Anthony J. "Tony" Tether
and Norm Whitaker

Springer Tracts in Advanced Robotics

Volume 56

Martin Buehler, Karl Iagnemma,
Sanjiv Singh (Eds.)

The DARPA Urban Challenge

Autonomous Vehicles in City Traffic

Springer

Professor Bruno Siciliano, Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, Via Claudio 21, 80125 Napoli, Italy, E-mail: siciliano@unina.it

Professor Oussama Khatib, Artificial Intelligence Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305-9010, USA, E-mail: khatib@cs.stanford.edu

Professor Frans Groen, Department of Computer Science, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands, E-mail: groen@science.uva.nl

Editors

Dr. Martin Buehler
iRobot Corporation
8 Crosby Drive, M/S 8-1
Bedford, MA 01730
USA
E-mail: mbuehler@irobot.com

Prof. Sanjiv Singh
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
USA
E-mail: ssingh@ri.cmu.edu

Dr. Karl Iagnemma
Department of Mechanical Engineering
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
USA
E-mail: kdi@mit.edu

ISBN 978-3-642-03990-4 e-ISBN 978-3-642-03991-1

DOI 10.1007/978-3-642-03991-1

Springer Tracts in Advanced Robotics ISSN 1610-7438

Library of Congress Control Number: 2009934347

Typeset & Cover Design: Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed in acid-free paper

5 4 3 2 1 0

springer.com

STAR (Springer Tracts in Advanced Robotics) has been promoted under the auspices of EURON (European Robotics Research Network)

Foreword

By the dawn of the new millennium, robotics has undergone a major transformation in scope and dimensions. This expansion has been brought about by the maturity of the field and the advances in its related technologies. From a largely dominant industrial focus, robotics has been rapidly expanding into the challenges of the human world. The new generation of robots is expected to safely and dependably co-habitat with humans in homes, workplaces, and communities, providing support in services, entertainment, education, healthcare, manufacturing, and assistance.

Beyond its impact on physical robots, the body of knowledge robotics has produced is revealing a much wider range of applications reaching across diverse research areas and scientific disciplines, such as: biomechanics, haptics, neurosciences, virtual simulation, animation, surgery, and sensor networks among others. In return, the challenges of the new emerging areas are proving an abundant source of stimulation and insights for the field of robotics. It is indeed at the intersection of disciplines that the most striking advances happen.

The goal of the series of *Springer Tracts in Advanced Robotics (STAR)* is to bring, in a timely fashion, the latest advances and developments in robotics on the basis of their significance and quality. It is our hope that the wider dissemination of research developments will stimulate more exchanges and collaborations among the research community and contribute to further advancement of this rapidly growing field.

The volume edited by Martin Buehler, Karl Iagnemma and Sanjiv Singh presents a unique and complete collection of the scientific results by the finalist teams which took part into the Urban Challenge in November 2007 in the mock city environment of the George Air Force base in Victorville, California. The book is the companion of the previous book by the same editors which was devoted to the Grand Challenge in the Nevada desert of October 2005, the second in the series sponsored by DARPA.

The Urban Challenge demonstrated how the fast growing progress toward the development of new perception, control, and motion planning techniques allow intelligent autonomous vehicles not only to travel significant distances in off-road terrain, but also to operate in urban scenarios. Beyond the value for future military applications motivating DARPA to sponsor the race, the expected impact

in the commercial sector for automotive manufacturers is equally if not more important: autonomous sensing and control constitute key technologies to vehicles of the future that might help save thousands of lives now lost in traffic accidents!

Like in the case of the previous volume, the original papers were earlier published in three special issues of the Journal of Field Robotics. Our series is very proud to reprise them and again offer archival publication as a special STAR volume!

Naples, Italy
July 2009

Bruno Siciliano
STAR Editor

Foreword

It might have been the first robotic demolition derby.

Imagine a large field of vehicles without drivers traversing 60 miles in live traffic, operating entirely without human guidance. A complex course including intersections, traffic circles, and parking lots, defined by just kilobytes of data. Vehicles several meters wide traveling down lanes only slightly wider, using localization systems with an accuracy of several meters. Humans in vehicles facing full-size unmanned vehicles at approach speeds up to 60 miles per hour. Even today, this does not sound like a recipe for success.

The Urban Challenge, conducted in November, 2007, began with the vision of orderly robotic traffic –busy city streets with a mix of human and robotic drivers. It is clear that the future use of autonomous vehicles would be severely limited unless operation were demonstrably safe amidst moving traffic. A conclusive demonstration would be impossible for many other organizations, but this is precisely the type of risk that the Defense Advanced Research Projects Agency (DARPA) was created to tackle.

In the face of such long odds, the Agency's ace card is its ultra-resourceful contractor community. It was this community of participants, who deciphered the rules, husbanded resources, and invented the way to a successful conclusion. Their technical results are set down in this special edition, but read between the lines to appreciate the magnitude of the task and the inherent risk undertaken.

The successful program outcome is really a tribute to this entire community, from the top teams who conducted tutorials in the pit area, to the intrepid groups of undergraduates who dared to compete on a shoestring. This technical community is the group that both taught one another and competed with one another to create the excellence that will be remembered as the Urban Challenge.

In the end, when the last paper is written and the best ideas are carried forward into subsequent developments, what remains is the inspiration of a group of researchers who proved to themselves and to the world what was possible with too little funds, not enough time, in pursuit of a clear and focused goal.

Congratulations to them all.

Norm Whitaker
DARPA Urban Challenge Program Manager

Foreword

The 3rd DARPA Grand Challenge also known as the "Urban Challenge" almost didn't happen. The previous challenges ended so successfully that I didn't see a point to go onto another one. DARPA's mission is to show that something can be done and to transition it on to other agencies and organizations for the development while we go back to determine what new technology needs to be brought forth.

But there were strong arguments to carry on; the major point was that we didn't prove it could be done in traffic when there were both moving robotic vehicles and moving vehicles driven by people. I agreed and the Urban Challenge was born.

We decided on George Air Force base in Victorville California, which was no longer in use but still had a housing development with streets, stop signs, etc. We also decided that the evaluation would not be strictly based on speed getting through a course but that the vehicles had to obey California driving laws. In fact we decided to use the California driving test evaluation criteria as a way to distinguish between vehicles that could go fast and those that also had at least the intelligence to pass the test.

This meant that we were going to have to have people out on the track writing traffic tickets which increased greatly the danger of the event.

We had 20+ vehicles make it into the qualifying runs. They had to go through a difficult test. Not only did they have to be good technically but they also had to prove that they were safe. The safety concern culled the number of vehicles who were going to be allowed into the finals down to eleven.

I worried about what would happen the first time robotic vehicles met other robotic vehicles with no possible human intervention. This was something we couldn't test in the qualifying runs and was a major unknown.

The nightmare happened within minutes of the start when four robotics vehicles came to a 4-way stop at the same time. I held my breath as this event unfolded.

It turned out that there wasn't a problem. California rules state that the vehicle which arrives before yours at the stop sign has the right of way. The robotic vehicles knew the arrival order and therefore knew their turn. The robots performed perfectly. In fact, we were having trouble with the vehicles driven by humans who tended to somewhat disobey the California rules.

It was a spectacular finish. We had 6 out of the 11 starters finish and gave away all the 1st, 2nd, and 3rd prizes. I am sure this book will go into greater depth on the details.

The response from the US and the world was fantastic. We had done what we wanted to do and showed that robotic vehicles could perform, even when mixed in with each other and people driven vehicles, in a very realistic environment.

The Urban Challenge showed that a new important military capability was possible and convoys, for example, might someday not need people drivers. But as important, we had impacted the lives of tens of thousands of people who might never have gotten involved in science and engineering if it had not been for the Challenge series and learned how much fun it was.

The Challenge series may not have been the most important capability that was developed during my 8 years as DARPA Director but it was high on the list and is undoubtedly the most publicly known development since the internet. I am sure that this book will give you much more insight and details in what happened and I know you will enjoy reading it even if you were not there in person.

Anthony J. "Tony" Tether
DARPA Director, 2001-2009

Preface

The Defense Advanced Research Projects Agency (DARPA) Urban Challenge (DUC) was held on November 3, 2007, at the now-closed George Air Force Base in Victorville, California, in the United States. The DUC was the third in a series of DARPA-sponsored competitions for autonomous vehicles. Whereas the previous two "Grand Challenges" (held in 2003 and 2005) were intended to demonstrate that autonomous ground vehicles could travel significant distances in off-road terrain, the DUC was designed to foster innovation in autonomous vehicle operation in busy urban environments. Competitors developed full-scale (i.e., passenger vehicle–sized) autonomous vehicles to navigate through a mock city environment, executing simulated military supply missions while merging into moving traffic, navigating traffic circles, negotiating busy intersections, and avoiding obstacles. Race rules required that the 96 km (60 mile) course be completed in less than 6 hours. The rules also required that competitors obey all traffic regulations while avoiding other competitors and humandriven "traffic vehicles."

The winner of the race—and of a $2 million dollar first prize—was a modified Chevy Tahoe named "Boss" developed by Tartan Racing, a team led by Carnegie Mellon University. The second-place finisher, and recipient of a $1 million prize, was Stanford Racing Team's "Junior," a Volkswagen Passat. In third place was team VictorTango from Virginia Tech, winning a $500,000 prize with a Ford Escape hybrid dubbed "Odin." Vehicles from MIT, Cornell, and the University of Pennsylvania/Lehigh also successfully completed the course. It should be noted that these 6 teams were winnowed down from an initial pool of 89 teams that were initially accepted for participation in the DUC. Three months before the race, a panel from DARPA selected 35 teams to participate in the National Qualifying Event (NQE), which was held one week before the final race. Field trials at the NQE narrowed the field down to the 11 teams that competed on race day.

It can be argued that the greatest achievement of the Urban Challenge was the production of important new research in perception, control, and motion planning for intelligent autonomous vehicles operating in urban scenarios. Another long-term result of the DUC is the undeniable shift in public perception that robotic systems are now able to successfully manage the complexities of an urban environment. Although the race's mock city environment simplified some of the

challenges present in a real urban environment (e.g., there were no pedestrians or traffic signals), the race left no doubt in the minds of most observers that the development of vehicles that can "drive themselves" in real-world settings is now inevitable.

Although DARPA's direct motivation for sponsoring the race was to foster technology for future military applications, a nearer term impact may lie in the commercial sector. Automotive manufacturers view autonomous sensing and control technologies as keys to vehicles of the future that will save thousands of lives now lost in traffic accidents. Manufacturers of mining and agricultural equipment are also interested in creating a next generation of vehicles that will reduce the need for human control in dirty and dangerous applications. Clearly, if the technology displayed at the DUC can be made inexpensive and robust enough for use in the commercial sector, the effect of the Urban Challenge on society will be substantial and long lasting. For this, the robotics community is beholden to DARPA for providing both critical resources and a well-designed evaluation process for the competition.

This book presents 13 papers describing all of the vehicles that competed as finalists in the DUC. These papers initially appeared in three special issues of the *Journal of Field Robotics*, in August, September, and October 2008. They document the mechanical, algorithmic, and sensory solutions developed by the various teams. All papers were subjected to the normal *Journal of Field Robotics* peer review process. Also included in this volume is a new picture gallery of the finalist robots, with a description of their individual race results, and forewards by Norm Whitaker, the DARPA program manager who oversaw the Urban Challenge contest, and Tony Tether, who served as DARPA's director from 2001-2009.

The first paper, Tartan Racing's "Autonomous Driving in Urban Environments: Boss and the Urban Challenge" by Urmson et al., is a comprehensive description of Boss. The paper describes Boss's mechanical and software systems, including its motion planning, perception, mission planning, and tactical behavior algorithms. The software infrastructure is also detailed. Testing, performance in the NQE, and race performance are also documented. Boss averaged 22.5 km/h during the race and completed the course with a winning time of 4 hours and 10 minutes. A companion paper, "Motion Planning in Urban Environments," by Ferguson et al., offers more detail about Boss' planning system, which combines a model-predictive trajectory generation algorithm for computing dynamically feasible actions with two higher-level planners for generating long-range plans in both on-road and unstructured regions of the environment.

The next paper, "Junior: The Stanford Entry in the Urban Challenge" by Montemerlo et al., focuses on Stanford's software and describes how Junior made its driving decisions through a distributed software pipeline that integrated perception, planning, and control. The paper illustrates the development of a robust system for urban in-traffic autonomous navigation, based on the integration of recent innovations in probabilistic localization, mapping, tracking, global and local planning, and a finite state machine for making the robot robust to unexpected situations. Also presented are new developments in obstacle/curb detection, vehicle tracking, motion planning, and behavioral hierarchies that

address a broad range of traffic situations. The paper concludes with an analysis of notable race events. Junior averaged 22.1 km/h during the race and completed the course with a second-place time of 4 hours and 29 minutes.

Team VictorTango's entry into the DUC is described in the paper "Odin: Team VictorTango's Entry in the DARPA Urban Challenge" by Bacha et al. An overview of the vehicle platform and system architecture is provided, along with a description of the perception and planning systems. A description of Odin's performance in the NQE and race is also provided, including an analysis of various issues faced by the vehicle during testing and competition. Odin averaged just under 21 km/h in the race and completed the course in third place with a time of 4 hours and 36 minutes.

The paper, "A Perception-Driven Autonomous Urban Vehicle," from the MIT team, describes the architecture and implementation of a vehicle designed to handle the DARPA Urban Challenge requirements of perceiving and navigating a road network with segments defined by sparse waypoints. The vehicle implementation includes a large suite of heterogeneous sensors with significant communications and computation bandwidth to capture and process high-resolution, high-rate sensor data. The output of the perception system is fed into a kinodynamic motion planning algorithm that enables driving in lanes, three-point turns, parking, and maneuvering through obstacle fields. The intention was to develop a platform for research in autonomous driving in GPS-denied and highly dynamic environments with poor a priori information. Team MIT's entry successfully completed the course, finishing in fourth place.

"Little Ben: The Ben Franklin Racing Team's Entry in the 2007 DARPA Urban Challenge" by Bohren et al. details the sensing, planning, navigation, and actuation systems for "Little Ben," a modified Toyota Prius hybrid. The paper describes methods for integrating sensor information into a dynamic map that can robustly handle GPS dropouts and errors. A planning algorithm is presented that consists of a high-level mission planner and low-level trajectory planner. A method for cost-based actuator level control is also described. Little Ben was one of the six vehicles that successfully completed the Urban Challenge.

The paper "Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment" by Miller et al. describes Team Cornell's entry into the DUC, detailing the design and software of the autonomous vehicle Skynet. The article describes Skynet's custom actuation and power distribution system, tightly coupled attitude and position estimator, novel obstacle detection and tracking system, system for augmenting position estimates with vision-based detection algorithms, path planner based on physical vehicle constraints and a nonlinear optimization routine, and a state-based reasoning agent for obeying traffic laws. The successful performance of Skynet at the NQE and final race are also described.

"A Practical Approach to Robotic Design for the DARPA Urban Challenge" by Patz et al. describes the journey of TeamUCF and their "Knight Rider" during the Urban Challenge. Three of the only five core team members had participated in the 2005 Grand Challenge. This team's success is all the more impressive when considering its small size and budget. Sensor data were fused from a Doppler

radar and multiple SICK laser scanners. Two of those scanners rotated to provide 3-D image processing with both range and intensity data. This "world view" was processed by a context-based reasoning control system to yield tactical mission commands, which were forwarded to traditional PID control loops.

The next paper, "Team AnnieWAY's Autonomous System for the DARPA Urban Challenge 2007," describes Team AnnieWay's minimalistic approach that relied primarily on a multibeam Velodyne laser scanner mounted on the rooftop of their VW Passat, and just one computer. The laser scanner's range data provided 3D scene geometry information, and the reflectivity data allowed robust lane marker detection. Mission and maneuver selection was conducted via a hierarchical state machine. The reactive part of the system used a precomputed set of motion primitives that vary with the speed of the vehicle and that are described in the subsequent paper, "Driving with Tentacles: Integral Structures for Sensing and Motion" by von Hundelshausen et al. Here, motion primitives (tentacles) that Team AnnieWAY used for both perception and motion execution are described. In contrast to other methods, the algorithm uses a vehicle-centered occupancy grid to avoid obstacles. The approach is very efficient, because the relationship between tentacles and the grid is static. Even though this approach is not based on vehicle dynamics, the resulting path errors are shown to be bounded to obstacle-free areas.

"Caroline: An Autonomously Driving Vehicle for Urban Environments," describes the architecture of a system comprising eight main modules: sensor data acquisition, sensor data fusion, image processing, digital map, artificial intelligence, vehicle path planning and low-level control, supervisory watchdog and online-diagnosis, and telemetry and data storage for offline analysis. Detailed analysis of the vehicle's performance provides interesting insights into the challenges of autonomous urban driving systems. The paper concludes with a description of the events that led up to the collision with MIT's Talus, and the resulting elimination of Caroline.

The paper, "The MIT–Cornell Collision and Why It Happened," is an in-depth analysis into the collision between the MIT and the Cornell vehicles partway into the competition. This collaborative study, conducted jointly by MIT and Cornell, traces the sequence of events that preceded the collision and examines its root causes. A summary of robot–robot interactions during the race is presented. The logs from both vehicles are used to show the gulf between robot and human-driver behavior at close vehicle proximities. The paper ends with proposed approaches that could address the issues found to be at fault.

The paper, "A Perspective on Emerging Automotive Safety Applications, Derived from Lessons Learned through Participation in the DARPA Grand Challenges," is a description of the entry led by Ford Motor Company. The article provides a motivation for robotics research as a means to achieve greater safety for passenger vehicles, with an analysis that suggests that human drivers are four to six times as competent as today's autonomous vehicles. The article examines the design of the Ford team's autonomous system and accompanying sensor suite. It presents a detailed analysis of vehicle performance during trials and the competition and concludes with lessons learned.

The final paper, "TerraMax: Team Oshkosh Urban Robot," describes an entry

that was distinguished by its use of a 12-ton medium tactical vehicle replacement (MTVR), which provides the majority of the logistics support for the Marine Corps. Sensing was primarily done using passive computer vision augmented by laser scanning. The article provides a description of the system and an analysis of the performance during the competition, and during a run conducted afterward on the same course.

We hope that the papers collected here will be of interest to both roboticists and a wider audience of readers who are interested in learning about the state of the art in autonomous vehicle technology. The sensors, algorithms, and architectures described in these issues will no doubt soon be seen on highways, construction sites, and factory floors. Readers of this book might also be interested in a companion volume, The 2005 DARPA Grand Challenge: The Great Robot Race (Springer, 2007), which describes the technological innovation behind robots that raced in the 2005 DARPA Grand Challenge.

Finally, we would like to express our gratitude to the many individuals who served as reviewers of these papers, often through several iterations, and contributed to their high quality.

Martin Buehler
Karl Iagnemma
Sanjiv Singh

Acknowledgements to Reviewers

The editors would like to express their appreciation to the following professionals who generously contributed their time and expertise to provide reviews for the manuscripts in the DARPA Urban Challenge Special Issues of the *Journal of Field Robotics*:

Barrett, David
BenAmar, Faiz
Berkemeier, Matthew
Broggi, Alberto
Corke, Peter
Cousins, Steve
Crane, Carl
Daily, Robert
Digney, Bruce
Durrant-Whyte, Hugh
Ferguson, Dave
Feron, Eric
Foessel, Alex
Frazzoli, Emilio
Golconda, Suresh
Hall, Ernie
Halme, Aarne
Hamner, Bradley
How, Jonathan
Howard, Andrew
Howard, Thomas
Hundelshausen, Felix
Jones, Randy
Kalik, Steven
Kammel, Soeren
Klarquist, William
Kluge, Karl
Kuwata, Yoshi
Lee, Daniel
Leedy, Brett
Leonard, John
Maimone, Mark
Matthies, Larry
Minor, Mark
Montemerlo, Mike
Nebot, Eduardo
Newman, Paul
Papelis, Yiannis
Pillat, Remo
Pradalier, Cedric
Rasmussen, Christopher
Roberts, Jonathan
Rumpe, Bernhard
Rybski, Paul
Simmons, Reid
Spenko, Matthew
Stroupe, Ashley
Teller, Seth
Trepagnier, Paul
Vandapel, Nicolas
Urmson, Chris
Wang, Chieh-Chih (Bob)
Wooden, David
Yamauchi, Brian
Yoder, JD
Yoshida, Kazuya
Zlot, Robert

Picture Gallery of Finalists

Vehicle Name	Team Name	Result
Boss	Tartan Racing Team	Finished first in 250 minutes, 20 seconds.

Source: Tartan Racing Team

Vehicle Name	Team Name	Result
Junior	Stanford Racing Team	Finished second in 269 minutes, 28 seconds.

Source: DARPA

Vehicle Name	Team Name	Result
Odin	Victor Tango	Finished third in 276 minutes, 38 seconds.

Source: Victor Tango

Vehicle Name	Team Name	Result
Talus	MIT	Finished fourth.

Source: DARPA

Vehicle Name	Team Name	Result
Little Ben	Ben Franklin Racing Team	One of the six vehicles that finished the course (fifth or sixth place).

Source: DARPA

Vehicle Name	Team Name	Result
Skynet	Team Cornell	One of the six vehicles that finished the course (fifth or sixth place).

Source: DARPA

Vehicle Name	Team Name	Result
KnightRider	Team UCF	Drove for two hours until a GPS data failure occurred.

Source: DARPA

Vehicle Name	Team Name	Result
AnnieWay	Team AnnieWay	AnnieWay stopped due to a software exception that occurred while switching from road planning to zone navigation.

Source: DARPA

Vehicle Name	Team Name	Result
Caroline	CarOLO	Caroline drove 16.4 km (10.2 miles) and was retired shortly after a collision with MIT's TALOS.

Source: DARPA

Vehicle Name	Team Name	Result
XAV-250	Intelligent Vehicle Systems	While waiting at a stop sign, a false positive (sensing a curb where there was none) together with a large time-out value caused an excessive wait which disqualified the vehicle.

Source: IVS

Vehicle Name	Team Name	Result
TerraMax	Team Oshkosh	TerraMax completed the first four sub-missions in mission 1 before being stopped after a failure in the parking lot due to a software bug.

Source: Oshkosh

Contents

Autonomous Driving in Urban Environments: Boss and the Urban Challenge

Chris Urmson[1,*], Joshua Anhalt[1], Drew Bagnell[1], Christopher Baker[1], Robert Bittner[1], M.N. Clark[1], John Dolan[1], Dave Duggins[1], Tugrul Galatali[1], Chris Geyer[1], Michele Gittleman[1], Sam Harbaugh[1], Martial Hebert[1], Thomas M. Howard[1], Sascha Kolski[1], Alonzo Kelly[1], Maxim Likhachev[1], Matt McNaughton[1], Nick Miller[1], Kevin Peterson[1], Brian Pilnick[1], Raj Rajkumar[1], Paul Rybski[1], Bryan Salesky[1], Young-Woo Seo[1], Sanjiv Singh[1], Jarrod Snider[1], Anthony Stentz[1], William "Red" Whittaker[1], Ziv Wolkowicki[1], Jason Ziglar[1], Hong Bae[2], Thomas Brown[2], Daniel Demitrish[2], Bakhtiar Litkouhi[2], Jim Nickolaou[2], Varsha Sadekar[2], Wende Zhang[2], Joshua Struble[3], Michael Taylor[3], Michael Darms[4], and Dave Ferguson[5]

[1] Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
curmson@ri.cmu.edu
[2] General Motors Research and Development
Warren, Michigan
[3] Caterpillar Inc.
Peoria, Illinois 61656
[4] Continental AG
Auburn Hills, Michigan 48326
[5] Intel Research
Pittsburgh, Pennsylvania 15213

Abstract. Boss is an autonomous vehicle that uses on-board sensors (GPS, lasers, radars, and cameras) to track other vehicles, detect static obstacles and localize itself relative to a road model. A three-layer planning system combines mission, behavioral and motion planning to drive in urban environments. The mission planning layer considers which street to take to achieve a mission goal. The behavioral layer determines when to change lanes, precedence at intersections and performs error recovery maneuvers. The motion planning layer selects actions to avoid obstacles while making progress towards local goals.

The system was developed from the ground up to address the requirements of the DARPA Urban Challenge using a spiral system development process with a heavy emphasis on regular, regressive system testing. During the National Qualification Event and the 85km Urban Challenge Final Event Boss demonstrated some of its capabilities, qualifying first and winning the challenge.

1 Introduction

In 2003 the Defense Advanced Research Projects Agency (DARPA) announced the first Grand Challenge. The goal was to develop autonomous vehicles capable

* Corresponding author.

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 1–59.
springerlink.com

of navigating desert trails and roads at high speeds. The competition was generated as a response to a congressional mandate that a third of US military ground vehicles be unmanned by 2015. While there had been a series of ground vehicle research programs, the consensus was that existing research programs would be unable to deliver the technology necessary to meet this goal (Committee on Army Unmanned Ground Vehicle Technology, 2002). DARPA decided to rally the field to meet this need.

The first Grand Challenge was held in March 2004. Though no vehicle was able to complete the challenge, a vehicle named Sandstorm went the furthest and the farthest, setting a new benchmark for autonomous capability and provided a template on how to win the challenge (Urmson et al., 2004). The next year, 5 vehicles were able to complete a similar challenge, with Stanley (Thrun et al., 2006) finishing minutes ahead of Sandstorm and H1ghlander (Urmson et al, 2006) to complete the 244km race in a little under 7 hours.

After the success of the Grand Challenges, DARPA organized a third event: the Urban Challenge. The challenge, announced in April 2006, called for autonomous vehicles to drive 97 km through an urban environment interacting with other moving vehicles and obeying the California Driver Handbook. Interest in the event was immense, with 89 teams from around the world registering interest in competing. The teams were a mix of industry and academics, all with enthusiasm for advancing autonomous vehicle capabilities.

To compete in the challenge, teams had to pass a series of tests. The first was to provide a credible technical paper describing how they would implement a safe and capable autonomous vehicle. Based on these papers, fifty-three teams were given the opportunity to demonstrate firsthand for DARPA their ability to navigate simple urban driving scenarios including passing stopped cars and

Fig. 1. Boss, the autonomous Chevy Tahoe that won the 2007 DARPA Urban Challenge.

interacting appropriately at intersections. After these events, the field was further narrowed to thirty-six teams who were invited to participate in the National Qualification Event (NQE) in Victorville, California. Of these teams, only eleven would qualify for the Urban Challenge Final event (UCFE).

1.1 Overview

This article describes the algorithms and mechanism that makeup Boss (see Figure 1), an autonomous vehicle capable of driving safely in traffic at speeds up to 48 kph. Boss is named after Charles "Boss" Kettering, a luminary figure in the automotive industry, with inventions as wide-ranging as the all-electric starter for the automobile, the coolant Freon, and the premature-infant incubator. Boss was developed by the Tartan Racing Team, which was composed of students, staff and researchers from several organizations including Carnegie Mellon University, General Motors, Caterpillar, Continental, and Intel. This article begins by describing the autonomous vehicle and sensors before moving on to a discussion of the algorithms and approaches that enabled it to drive autonomously.

The motion planning sub-system (described in section 3) consists of two planners, each capable of avoiding static and dynamic obstacles while achieving a desired goal. Two broad scenarios are considered: structured driving (road following) and unstructured driving (maneuvering in parking lots). For structured driving, a local planner generates trajectories to avoid obstacles while remaining in its lane. For unstructured driving, such as entering/exiting a parking lot, a planner with a four-dimensional search space (position, orientation, direction of travel) is used. Regardless of which planner is currently active, the result is a trajectory that, when executed by the vehicle controller, will safely drive toward a goal.

The perception sub-system (described in section 4) processes and fuses data from Boss's multiple sensors to provide a composite model of the world to the rest of the system. The model consists of three main parts: a static obstacle map, a list of the moving vehicles in the world, and the location of Boss relative to the road.

The mission planner (described in section 5) computes the cost of all possible routes to the next mission checkpoint given knowledge of the road network. The mission planner reasons about the optimal path to a particular checkpoint much like a human would plan a route from their current position to a destination, such as a grocery store or gas station. The mission planner compares routes based on knowledge of road blockages, the maximum legal speed limit, and the nominal time required to make one maneuver versus another. For example, a route that allows a higher overall speed, but incorporates a U-turn, may actually be slower than a route with a lower overall speed but that does not require a U-turn.

The behavioral system (described in section 6) formulates a problem definition for the motion planner to solve based on the strategic information provided by the mission planner. The behavioral sub-system makes tactical decisions to execute the mission plan and handles error recovery when there are problems. The behavioral system is roughly divided into three sub-components: *Lane Driving*, *Intersection Handling*, and *Goal Selection*. The roles of the first two sub-components are self-explanatory. *Goal Selection* is responsible for distributing

execution tasks to the other behavioral components or the motion layer, and for selecting actions to handle error recovery.

The software infrastructure and tools that enable the other sub-systems are described in section 7. The software infrastructure provides the foundation upon which the algorithms are implemented. Additionally, the infrastructure provides a toolbox of components for online data logging, offline data log playback, and visualization utilities that aid developers in building and troubleshooting the system. A run-time execution framework is provided that wraps around algorithms and provides inter-process communication, access to configurable parameters, a common clock, and a host of other utilities.

Testing and performance in the NQE and UCFE are described in sections 8 and 9. During the development of Boss, the team put a significant emphasis on evaluating performance and finding weaknesses to ensure the vehicle would be ready for the Urban Challenge. During the qualifiers and final challenge, Boss performed well, but made a few mistakes. Despite these mistakes and a very capable field of competitors, Boss qualified for the final event and won the Urban Challenge.

2 Boss

Boss is a 2007 Chevrolet Tahoe modified for autonomous driving. Modifications were driven by the need to provide computer control and also to support safe and efficient testing of algorithms. Thus, modifications can be classified into two categories: those for automating the vehicle and those that made testing either safer or easier. A commercial off-the-shelf drive-by-wire system was integrated into Boss with electric motors to turn the steering column, depress the brake pedal, and shift the transmission. The third-row seats and cargo area were replaced with electronics racks, the steering was modified to remove excess compliance, and the brakes were replaced to allow faster braking and reduce heating.

Boss maintains normal human driving controls (steering wheel, brake and gas pedals) so that a safety driver can quickly and easily take control during testing. Boss has its original seats in addition to a custom center console with power and network outlets which enable developers to power laptops and other accessories, supporting longer and more productive testing. A welded tube roll cage was also installed to protect human occupants in the event of a collision or roll-over during testing. For unmanned operation a safety radio is used to engage autonomous driving, pause or disable the vehicle.

Boss has two independent power busses. The stock Tahoe power bus remains intact with its 12VDC battery and harnesses but with an upgraded high-output alternator. An auxiliary 24VDC power system provides power for the autonomy hardware. The auxiliary system consists of a belt-driven alternator which charges a 24VDC battery pack which is inverted to supply a 120VAC bus. Shore power, in the form of battery chargers, enables Boss to remain fully powered when in the

shop with the engine off. Thermal control is maintained using the stock vehicle air conditioning system.

For computation, Boss uses a CompactPCI chassis with ten 2.16GHz Core2Duo processors, each with 2GB of memory and a pair of gigabit Ethernet ports. Each computer boots off of a 4GB flash drive, reducing the likelihood of a disk failure. Two of the machines also mount 500GB hard drives for data logging. Each computer is also time-synchronized through a custom pulse-per-second adaptor board.

Boss uses a combination of sensors to provide the redundancy and coverage necessary to navigate safely in an urban environment. Active sensing is used predominantly, as can be seen in Table 1. The decision to emphasize active sensing was primarily due to the team's skills and the belief that in the Urban Challenge direct measurement of range and target velocity was more important than getting richer, but more difficult to interpret, data from a vision system. The configuration of sensors on Boss is illustrated in Figure 2. One of the novel aspects of this sensor configuration is the pair of pointable sensor pods located above the driver and front passenger doors. Each pod contains an ARS 300 Radar and ISF 172 LIDAR. By pointing these pods, Boss can adjust its field of regard to cover crossroads that may not otherwise be observed by a fixed sensor configuration.

Table 1. A description of the sensors incorporated into Boss.

Sensor	Characteristics
Applanix POS-LV 220/420 GPS / IMU (APLX)	• sub-meter accuracy with Omnistar VBS corrections • tightly coupled inertial/GPS bridges GPS-outages
SICK LMS 291-S05/S14 LIDAR (LMS)	• 180º / 90º x 0.9º FOV with 1º / 0.5º angular resolution • 80m maximum range
Velodyne HDL-64 LIDAR (HDL)	• 360º x 26º FOV with 0.1º angular resolution • 70m maximum range
Continental ISF 172 LIDAR (ISF)	• 12º x 3.2º FOV • 150m maximum range
IBEO Alasca XT LIDAR (XT)	• 240º x 3.2º FOV • 300m maximum range
Continental ARS 300 Radar (ARS)	• 60º / 17º x 3.2º FOV • 60m / 200m maximum range
Point Grey Firefly (PGF)	• High dynamic range camera • 45 º FOV

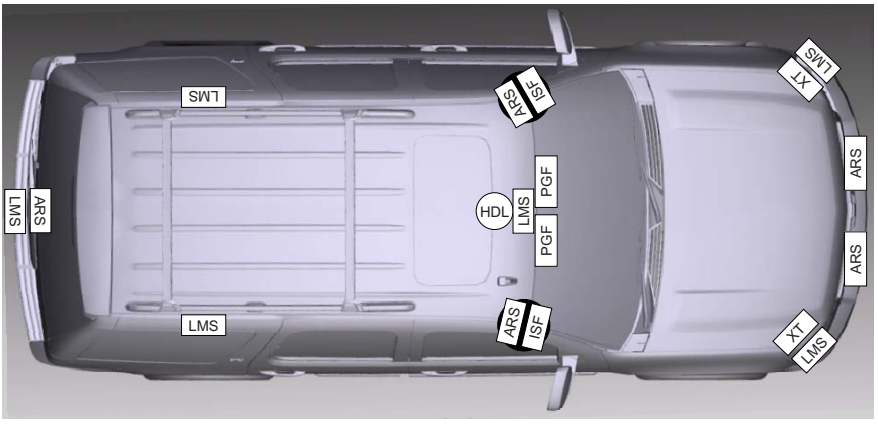


Fig. 2. The mounting location of sensors on the vehicle, refer to Table 1 for abbreviations used in this figure.

3 Motion Planning

The motion planning layer is responsible for executing the current motion goal issued from the behaviors layer. This goal may be a location within a road lane when performing nominal on-road driving, a location within a zone when traversing through a zone, or any location in the environment when performing error recovery. The motion planner constrains itself based on the context of the goal to abide by the rules of the road.

In all cases, the motion planner creates a path towards the desired goal, then tracks this path by generating a set of candidate trajectories that follow the path to varying degrees and selecting from this set the best trajectory according to an evaluation function. This evaluation function differs depending on the context, but includes consideration of static and dynamic obstacles, curbs, speed, curvature, and deviation from the path. The selected trajectory can then be directly executed by the vehicle. For more details on all aspects of the motion planner, see (Ferguson et al., 2008)

3.1 Trajectory Generation

A model-predictive trajectory generator originally presented in (Howard and Kelly, 2007) is responsible for generating dynamically feasible actions from an initial state (**x**) to a desired terminal state. In general, this algorithm can be applied to solve the problem of generating a set of parameterized controls (**u**(**p**,**x**)) that satisfy state constraints (**C**(**x**)) whose dynamics can be expressed in the form of a set of differential equations (**f**).

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x},\mathbf{u}\left(\mathbf{p},\mathbf{x}\right)\right) \tag{1}$$

To navigate urban environments, position and heading terminal state constraints are typically required to properly orient a vehicle along the road. The constraint equation ($\mathbf{x}_C$) is the difference between the target terminal state constraints and the integral of the model dynamics.

$$\mathbf{x}_C = \begin{bmatrix} x_C & y_C & \theta_C \end{bmatrix}^T \tag{2}$$

$$\mathbf{C}(\mathbf{x}) - \mathbf{x}_C - \int_0^{t_f} \dot{\mathbf{x}}(\mathbf{x},\mathbf{p})\, dt = 0 \tag{3}$$

The fidelity of the vehicle model directly correlates to the effectiveness of a model-predictive planning approach. The vehicle model describes the mapping from control inputs to state response (changes in position, orientation, velocity, etc...). Selecting an appropriate parameterization of controls is important because it defines the space over which the optimization is performed to satisfy the boundary state constraints.

The vehicle model used for Boss combines a curvature limit (the minimum turning radius), a curvature rate limit (a function of the maximum speed at which the steering wheel can be turned), maximum acceleration and deceleration, and a model of the control input latency. This model is then simulated using a fixed timestep Euler integration to evaluate the constraint equation.

The control inputs are described by two parameterized functions: a time-based linear velocity function (v_{cmd}) and an arc-length-based curvature function (κ_{cmd}):

$$\mathbf{u}(\mathbf{p},\mathbf{x}) = \begin{bmatrix} v_{cmd}(\mathbf{p},\mathbf{t}) + \kappa_{cmd}(\mathbf{p},\mathbf{s}) \end{bmatrix}^T \tag{4}$$

The linear velocity profile takes the form of a constant profile, linear profile, linear ramp profile, or a trapezoidal profile (Figure 3). The local motion planner selects the appropriate parameterization for particular applications (such as parking and distance keeping).

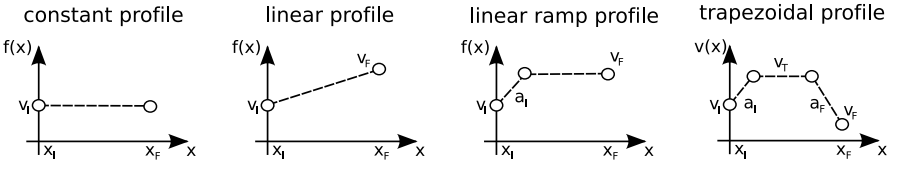


Fig. 3. Velocity profiles used by the trajectory generator.

The response to the curvature command function by the vehicle model defines the shape of the trajectory. The profile consists of three dependent parameters (κ_0, κ_1, and κ_2) and the trajectory length (s_f). A second-order spline profile was chosen because it contains enough degrees of freedom (4) to satisfy the boundary state constraints (3). The initial spline knot point (κ_0) is fixed during the optimization

process to a value that generates a *smooth* or *sharp* trajectory and will be discussed later.

$$p_{free} = \begin{bmatrix} \kappa_1 & \kappa_2 & s_f \end{bmatrix}^T \tag{5}$$

As described, the system retains three parameterized freedoms: two curvature command spline knot points (κ_1, κ_2) and the trajectory length (s). The duality of the trajectory length (s_f) and time (t_f) can be resolved by estimating the time that it takes to drive the entire distance through the linear velocity profile. Time was used for the independent variable for the linear velocity command function because of the simplicity of computing profiles defined by accelerations (linear ramp and trapezoidal profiles). Arc-length was used for the curvature command function because the trajectory shape is less dependent to the speed at which they are executed.

Given the three free parameters and the three constraints in our system, we can use various optimization techniques to solve for the parameter values that minimize our constraint equation. An initial estimate of the parameter values is defined using a pre-computed approximate mapping from state space to parameter space in a lookup table. The parameter estimates are iteratively modified by linearizing and inverting the differential equations describing the equations of motion. A correction factor is generated by taking the product of the inverted Jacobian and the boundary state constraint error. The Jacobian is model-invariant because it is determined numerically through central differences of simulated vehicle actions.

$$\mathbf{x}_F(\mathbf{p},\mathbf{x}) = \int_0^{tf} \dot{\mathbf{x}}(\mathbf{x},\mathbf{p})\, dt \tag{6}$$

$$\mathbf{C}(\mathbf{x},\mathbf{p}) = \mathbf{x}_C - \mathbf{x}_F(\mathbf{p},\mathbf{x}) \tag{7}$$

$$\Delta\mathbf{p} = -\left[\frac{\delta\mathbf{C}(\mathbf{x},\mathbf{p})}{\delta p}\right]^{-1} \mathbf{C}(\mathbf{x},\mathbf{p}) \tag{8}$$

The control parameters are modified until the residual of the boundary state constraints is within acceptable bounds or until the optimization diverges. If the boundary state constraints are infeasible to reach given a particular parameterization (e.g. inside the minimum turning radius) the optimization is expected to diverge. The resulting trajectory is returned as the best estimate and is evaluated by the motion planner.

3.2 On-Road Navigation

During on-road navigation, the motion goal from the behavioral system is a location within a road lane. The motion planner then attempts to generate a trajectory that moves the vehicle towards this goal location in the desired lane. To do this, it first constructs a curve along the centerline of the desired lane. This

represents the nominal path that the center of the vehicle should follow. This curve is then transformed into a path in rear-axle coordinates to be tracked by the motion planner.

To robustly follow the desired lane and to avoid static and dynamic obstacles, the motion planner generates trajectories to a set of local goals derived from the centerline path. The local goals are placed at a fixed longitudinal distance down the centerline path, but vary in lateral offset from the path to provide several options for the planner. The trajectory generation algorithm is used to compute dynamically feasible trajectories to these local goals. For each goal, two trajectories are generated: a smooth trajectory and a sharp trajectory. The smooth trajectory has the initial curvature parameter fixed to the curvature of the forwards-predicted vehicle state. The sharp trajectory has the initial curvature parameter set to an offset value from the forwards-predicted vehicle state to produce a sharp initial action. The velocity profile used for each of these trajectories is computed based on several factors, including: the maximum velocity bound given from the behavioral sub-system, the speed limit of the current road segment, the maximum velocity feasible given the curvature of the centerline path, and the desired velocity at the goal (e.g. zero if it is a stop line).

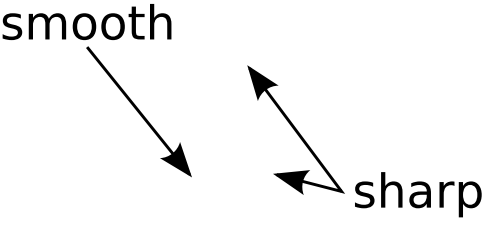


Fig. 4. Smooth and sharp trajectories. The trajectory sets are generated to the same endpoints but differ in their initial commanded curvature.

Figure 4 provides an example of smooth and sharp trajectories (light and dark), generated to the same goal poses. The smooth trajectories exhibit continuous curvature control throughout; the sharp trajectories begin with a discontinuous jump in curvature control, resulting in a sharp response from the vehicle.

The resulting trajectories are then evaluated against their proximity to static and dynamic obstacles in the environment, as well as their distance from the centerline path, their smoothness, and various other metrics. The best trajectory according to these metrics is selected and executed by the vehicle. Because the trajectory generator computes the feasibility of each trajectory using an accurate vehicle model, the selected trajectory can be directly executed by the vehicle controller.

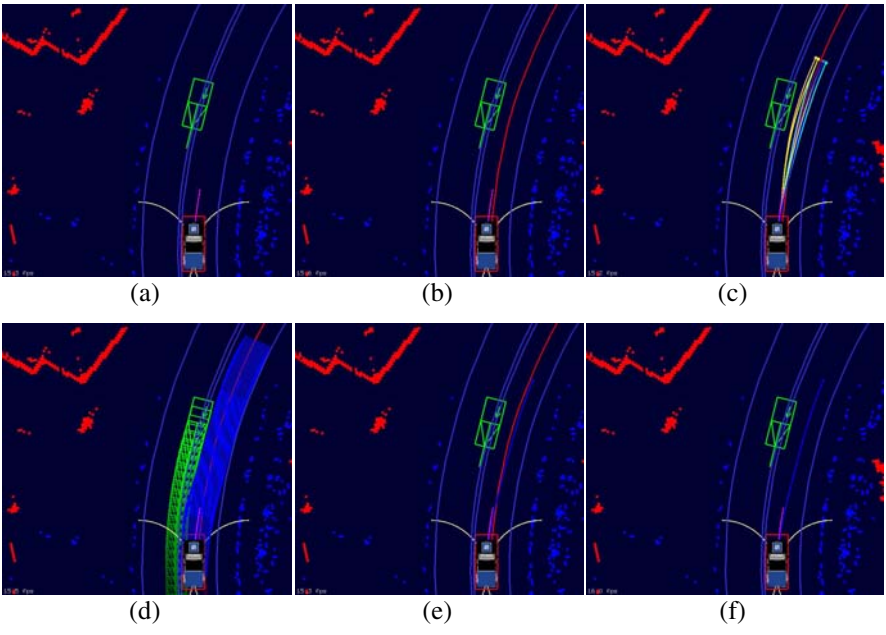


Fig. 5. A single timeframe following a road lane from the DARPA Urban Challenge. Shown is the centerline path extracted from the lane (b), the trajectories generated to track this path (c), and the evaluation of one of these trajectories against both static and dynamic obstacles (d &e).

Figure 5 provides an example of the local planner following a road lane. Figure 5 (a) shows the vehicle navigating down a two-lane road (lane boundaries shown in blue, current curvature of the vehicle shown in pink, minimum turning radius arcs shown in white) with a vehicle in the oncoming lane. Figure 5 (b) shows the extracted centerline path from the desired lane (in red). Figure 5 (c) shows a set of trajectories generated by the vehicle given its current state and the centerline path and lane boundaries. From this set of trajectories, a single trajectory is selected for execution, as discussed above. Figure 5 (d) shows the evaluation of one of these trajectories against both static and dynamic obstacles in the environment, and Figure 5(f) shows this trajectory being selected for execution by the vehicle.

3.3 Zone Navigation

During zone navigation, the motion goal from behaviors is a pose within a zone (such as a parking spot). The motion planner attempts to generate a trajectory that moves the vehicle towards this goal pose. However, driving in unstructured environments, such as zones, significantly differs from driving on roads. As mentioned in the previous section, when traveling on roads the desired lane implicitly provides a preferred path for the vehicle (the centerline of the lane).

In zones there are no driving lanes and thus the movement of the vehicle is far less constrained.

To efficiently plan a smooth path to a distant goal pose in a zone, we use a lattice planner that searches over vehicle position (x, y), orientation (θ), and speed (v). The set of possible local maneuvers considered for each (x, y, θ, v) state in the planner's search space is constructed offline using the same vehicle model as used in trajectory generation, so that it can be accurately executed by the vehicle. This planner searches in a backwards direction, from the goal pose out into the zone, and generates a path consisting of a sequence of feasible high-fidelity maneuvers that are collision-free with respect to the static obstacles observed in the environment. This path is also biased away from undesirable areas within the environment, such as curbs and locations in the vicinity of dynamic obstacles.

To efficiently generate complex plans over large, obstacle-laden environments, the planner relies on an anytime, replanning search algorithm known as Anytime D* (Likhachev et al., 2005). Anytime D* quickly generates an initial, suboptimal plan for the vehicle and then improves the quality of this solution while deliberation time allows. At any point in time, Anytime D* provides a provable upper bound on the sub-optimality of the plan. When new information concerning the environment is received (for instance, a new static or dynamic obstacle is observed), Anytime D* is able to efficiently repair its existing solution to account for the new information. This repair process is expedited by performing the search in a backwards direction, because in such a scenario, updated information in the vicinity of the vehicle affects a smaller portion of the search space so that less repair is required.

To scale to very large zones (up to 0.5 km by 0.5 km), the planner uses a multi-resolution search and action space. In the vicinity of the goal and vehicle, where very complex maneuvering may be required, the search considers states of the vehicles with 32 uniformly spaced orientations. In the areas that are not in the vicinity of the goal or a vehicle, the search considers only the states of the vehicle with 16 uniformly spaced orientations. It also uses a sparse set of actions that allow the vehicle to transition in between these states. Because coarse- and dense-resolution variants both share the same dimensionality and, in particular, have 16 orientations in common, they seamlessly interface with each other and the resulting solution paths overlapping both coarse and dense areas of the space are smooth and feasible.

To ensure that a path is available for the vehicle as soon as it enters a zone, the lattice planner begins planning for the first goal pose within the zone while the vehicle is still approaching the zone. By planning a path from the entry point of the zone in advance, the vehicle can seamlessly transition into the zone without needing to stop, even for very large and complex zones. In a similar vein, when the vehicle is in a zone traveling towards a parking spot, we have a second lattice planner computing a path from that spot to the next desired location (e.g. the next parking spot to reach or an exit of the zone). When the vehicle reaches its intended parking spot, the vehicle then immediately follows the path from this second planner, again eliminating any time spent waiting for a plan to be generated.

The resulting plan is then tracked by the local planner in a similar manner to the paths extracted from road lanes. The motion planner generates a set of trajectories that attempt to follow the plan while also allowing for local maneuverability. However, in contrast to when following lane paths, the trajectories generated to follow the zone path all attempt to terminate on the path. Each trajectory is in fact a concatenation of two short trajectories, with the first of the two short trajectories ending at an offset position from the path and the second ending back on the path. By having all concatenated trajectories return to the path, we significantly reduce the risk of having the vehicle move itself into a state that is difficult to leave.

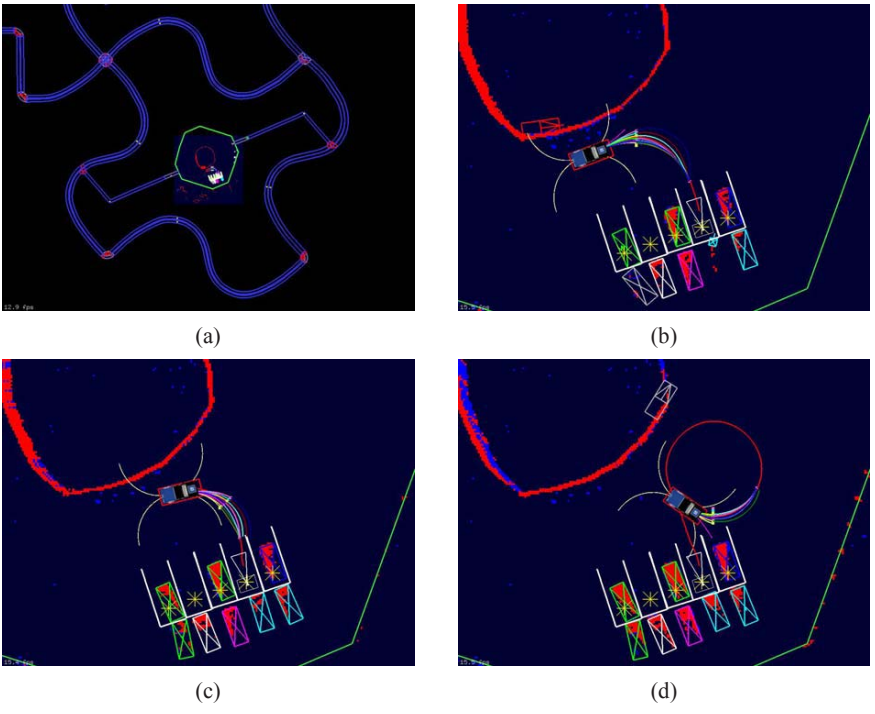


Fig. 6. Replanning when new information is received. As Boss navigates towards its desired parking spot (lattice path shown in red, trajectories to track path in various colors), it observes more of one of the adjacent vehicles and replans a path that brings it smoothly into the spot.

Figure 6 illustrates the tracking of the lattice plan and the replanning capability of the lattice planner. These images were taken from a parking task performed during the National Qualification Event (the top-left image shows the zone in green and the neighboring roads in blue). The top-right image shows the initial path planned for the vehicle to enter the parking spot indicated by the white triangle. Several of the other spots were occupied by other vehicles (shown as rectangles of varying colors), with detected obstacles shown as red areas. The

trajectories generated to follow the path are shown emanating from our vehicle (notice how each trajectory consists of two sections, with the first leaving the path and the second returning to the path). As the vehicle gets closer to its intended spot, it observes more of the vehicle parked in the right-most parking spot (bottom-left image). At this point, it realizes its current path is infeasible and replans a new path that has the vehicle perform a loop and pull in smoothly. This path was favored in terms of time over stopping and backing up to re-position.

The lattice planner is flexible enough to be used in a large variety of cases that can occur during on-road and zone navigation. In particular, it is used during error recovery when navigating congested intersections, to perform difficult U-turns, and to get the vehicle back on track after emergency defensive driving maneuvers. In such cases, the behaviors layer issues a goal pose (or set of poses) to the motion planner and indicates that it is in an error recovery mode. The motion planner then uses the lattice planner to generate a path to the set of goals, with the lattice planner determining during its planning which goal is easiest to reach. In these error recovery scenarios the lattice planner is biased to avoid areas that could result in unsafe behavior (such as oncoming lanes when on roads).

4 Perception

The perception system is responsible for providing a model of the world to the behavioral and motion planning sub-systems. The model includes the moving vehicles (represented as a list of tracked objects), static obstacles (represented in a regular grid), and localizing the vehicle relative to, and estimating the shape of, the roads it is driving on.

4.1 Moving Obstacle Detection and Tracking

The moving obstacle detection and tracking subsystem provides a list of object hypotheses and their characteristics to the behavioral and motion planning sub-systems. The following design principles guided the implementation:

- No information about driving context is used inside the tracking algorithm.
- No explicit vehicle classification is performed. The tracking system only provides information about the movement state of object hypotheses.
- Information about the existence of objects is based on sensor information only. It is possible for some objects to be predicted, but only for short time intervals, as a compensation for known sensor parameters. Detection drop outs caused by noise, occlusions and other artifacts must be handled elsewhere.
- Object identifiers are not guaranteed to be stable. A new identifier does not necessarily mean that it is a new object.
- Well-defined and distinct tracking models are used to maximize the use of information provided by heterogeneous sensors.
- Motion prediction exploits known road geometry when possible.
- Sensor-specific algorithms are encapsulated in sensor-specific modules.

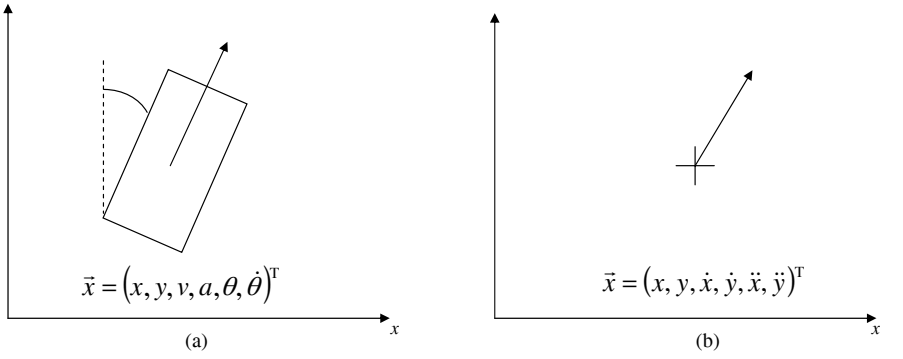


Fig. 7. The two models used by the tracking system are a reduced bicycle model with a fixed shape (a) and a Point Model without shape information (b).

Figure 7 shows the two tracking models used to describe object hypotheses. The box model represents a vehicle by using a simplified bicycle model (Kaempchen et al., 2004) with a fixed length and width. The point model provides no estimate of extent of the obstacle and assumes a constant acceleration model (Darms et al., 2008a) with adaptive noise dependent on the length and direction of the velocity vector. Providing two potential tracking models enables the system to represent the best model of tracked objects supported by the data. The system is able to switch between these models as appropriate.

The system classifies object hypotheses as either *Moving* or *Not Moving* and either *Observed Moving* or *Not Observed Moving*, so that each hypothesis can be in one of four possible states. The *Moving* flag is set if the object currently has a velocity which is significantly different from zero. The *Observed Moving* flag is set once the object has been moving for a significant amount of time (on the order of 0.4s) and is not cleared until the vehicle has been stopped for some larger significant amount of time (on the order of 10s). The four states act as a well-defined interface to the other software modules, enabling classes of tracked objects to be ignored in specific contexts (e.g. *Not Observed Moving* object hypotheses that are not fully on a road can be ignored for distance-keeping purposes, as they likely represent vehicles parked at the side of the road or other static obstacles (Darms et al., 2008b)).

Figure 8 illustrates the architecture of the tracking system. It is divided into two layers, a *Sensor Layer* and a *Fusion Layer* (Darms & Winner, 2005). For each sensor type (e.g. radar, scanning laser, etc.) a specialized sensor layer is implemented. For each physical sensor on the robot a corresponding sensor layer instance runs on the system. The architecture enables new sensor types to be added to the system with minimal changes to the fusion layer and other sensor modules, such as new physical sensors, can be added without any modifications to source code. The following paragraphs describe the path from sensor raw data to a list of object hypotheses.

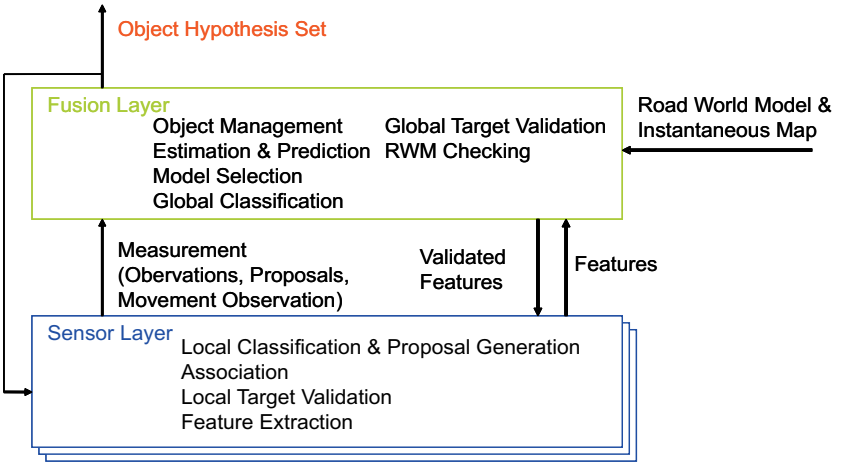


Fig. 8. The moving obstacle detection and tracking system architecture

Each time a sensor receives new raw data, its corresponding sensor layer instance requests a prediction of the current set of object hypotheses from the fusion layer. Features are extracted out of the measured raw data with the goal of finding all vehicles around the robot (e.g. edges from laser scanner data (MacLachlan, 2005)). Artifacts caused by ground detections or vegetation, for example, are suppressed by validating features in two steps. In the first step validation is performed with sensor-specific algorithms, e.g. using the velocity measurements inside a radar module to distinguish a static ground return from a moving vehicle. The second step is performed via a general validation interface. The validation performed inside the fusion layer uses only non-sensor-specific information. It performs checks against the road geometry and against an instantaneous obstacle map, which holds untracked 3D information about any obstacles in the near range. The result is a list of validated features which potentially originate from vehicles.

The validated features are associated with the predicted object hypotheses using a sensor-type-specific association algorithm. Afterwards, for each extracted feature (associated or not), multiple possible interpretations as a box or point model are generated using a sensor-type-specific heuristic, which takes the sensor characteristics into account (e.g. resolution, field of view, detection probabilities). The compatibility of each generated interpretation with its associated prediction is computed. If an interpretation differs significantly, or if the feature could not be associated, the sensor module initializes a new object hypothesis. In case of an associated feature, a new hypothesis can replace the current model hypothesis (box or point model). Note that for each feature, multiple new hypotheses can be generated. A set of new object hypotheses is called a proposal.

For each associated feature the interpretation which best fits the prediction is used to generate an observation. An observation holds all of the data necessary to update the state estimation for the associated object hypothesis in the fusion layer. If no interpretation is compatible, then no observation is generated and only the

proposal exists. As additional information for each extracted feature becomes available, the sensor module can also provide a movement observation. The movement observation tells the fusion layer whether an object is currently moving or not. This information is only based on sensor raw data (e.g. via an evaluation of the velocity measurement inside the radar module).

The proposals, observations and movement observations are used inside the fusion layer to update the object hypotheses list and the estimated object states. First the best tracking model (box or point) is selected with a voting algorithm. The decision is based on the number and type of proposals provided from the different sensors (Darms et al., 2008c). For objects which are located on roads, the road shape is used to bias the decision.

Once the best model is determined, the state estimate is either updated with the observation provided by the sensor layer or the model for the object hypothesis is switched to the best alternative. For unassociated features, the best model out of the proposal is added to the current list of object hypotheses. With the update process complete, object hypotheses which have not been observed for a certain amount of time are removed from the list.

Finally, a classification of the movement state for each object hypothesis is carried out. It is based on the movement observations from the sensors and a statistical test based on the estimated state variables. The movement observations from sensors are prioritized over the statistical test, and movement observations which classify an object as not moving overrule movement observations which classify an object as moving (Darms et al., 2008d).

The result is an updated list of object hypotheses which are accompanied by the classification of the movement state. For objects which are classified as *Moving* and *Observed Moving*, a prediction of the state variables is made. The prediction is based on logical constraints for objects which are located on the road. At every point where a driver has a choice to change lanes (e.g. at intersections), multiple hypotheses are generated. In zones (parking lots, for example) the prediction is solely based on the estimated states of the hypothesis (see Figure 9).

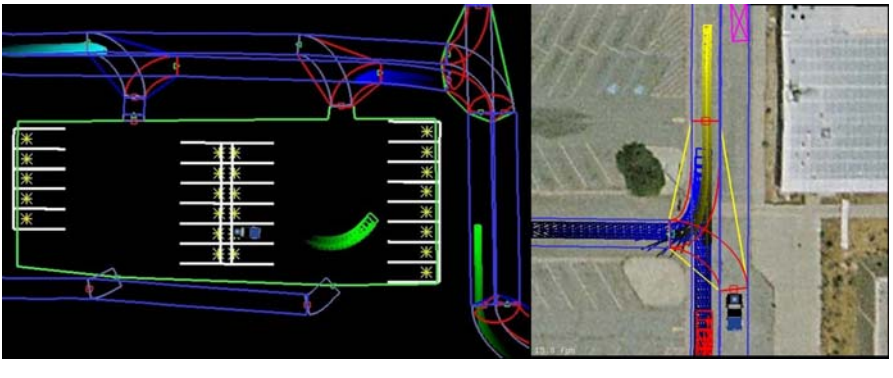


Fig. 9. The moving obstacle detection system predicts the motion of tracked vehicles. In parking lots (left) predictions are generated by extrapolating the tracking filter. For roads (right) vehicles are predicted to move along lanes.

4.2 Static Obstacle Detection and Mapping

The static obstacle mapping system combines data from the numerous scanning lasers on the vehicle to generate both instantaneous and temporally filtered obstacle maps. The instantaneous obstacle map is used in the validation of moving obstacle hypotheses. The temporally filtered maps are processed to remove moving obstacles and are filtered to reduce the number of spurious obstacles appearing in the maps. While there were several algorithms used to generate obstacle maps, only the curb detection algorithm is presented here.

Curb Detection and Mapping

Geometric features (curbs, berms and bushes) provide one source of information for determining road shape in urban and off-road environments. Dense lidar data provides sufficient information to generate accurate, long-range detection of these relevant geometric features. Algorithms to detect these features must be robust to the variation in features found across the many variants of curbs, berms, ditches, embankments, etc. The curb detection algorithm presented here exploits the Haar wavelet to deal with this variety.

To detect curbs we exploit two principle insights into the LIDAR data to simplify detection. First, the road surface is assumed to be relatively flat and slow changing, with road edges defined by observable changes in geometry, specifically in height. This simplification means the primary feature of a road edge reduces to changes in the height of the ground surface. Second, each LIDAR scan is processed independently, as opposed to building a three dimensional point cloud. This simplifies the algorithm to consider input data along a single dimension. The curb detection algorithm consists of three main steps: preprocessing, wavelet-based feature extraction, and post-processing.

The preprocessing stage provides two important features: mitigation of false positives due to occlusions and sparse data, and formatting the data for feature extraction. False geometric cues can result from striking both foreground and background objects, or due to missing data in a scan. Foreground objects are typically detected as obstacles (e.g. cones, telephone poles) and do not denote road edges. In order to handle these problems, points are initially clustered by distance between consecutive points. After clustering, small groups of points are removed from the scan. A second pass labels the points as dense or sparse based on the distances between them. The dense points are then linearly resampled in order to produce an input sequence of 2^n heights.

The Wavelet-based feature extraction step analyzes height data through a discrete wavelet transform using the Haar wavelet (Daubechies, 1992). The Haar wavelet is defined by the mother wavelet and scaling function:

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \le t < \frac{1}{2}, \\ -1 & \text{if } \frac{1}{2} < t < 1, \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$\varphi\left(2^j t - i\right) = \begin{cases} 1 \; if \; 0 \le t < 1 \\ 0 \; \text{otherwise} \end{cases} \quad j > 0 \wedge 0 \le i \le 2^j - 1 \tag{10}$$

The Haar transform results in a sequence of coefficients representing the scaled average slopes of the input signal within varying sampling windows (Shih & Tseng, 2005). Since each sampling window is half the size of the previous window, these windows successively subdivide the signal into higher resolution slopes or detail levels.

The feature extraction step (see Figure 10) takes the Haar coefficients, y, and considers them by window sizes, going from largest to smallest window. The algorithm classifies points as road points (class 1) or non-road points, and works as follows:

1. Collect coefficients for the current detail level, i.
2. Label each coefficient with the label of the coefficient at detail level $i - 1$ which represents the same portion of the signal.
3. Calculate $\hat{y}_{road}$ using these labels.
4. Re-label coefficients by absolute distance from $\hat{y}_{road}$, where the distance threshold for detail level i is given as d_i. In other words, points are labeled by the function:

$$class\left(y[n], i\right) = \begin{cases} 1 \text{ if} \left| y[n] - \hat{y}_{road} \right| \ge d_i \\ 0 \text{ otherwise} \end{cases} \tag{11}$$

5. Continue to detail level $i + 1$.

Post-processing applies a few extra heuristics to eliminate false positives and detect some additional non-road points. Using the dense/sparse labeling from pre-processing, non-road labels in sparse sections are moved from the sparse points to the neighboring dense point closest to the vehicle. Since all LIDARs on the vehicle look downwards, the closer point corresponds to the higher surface (e.g. berm, wall) creating the geometric cue. Afterwards, sparse points are removed from the classification list. The resulting list represents the locations of the likely road and surrounding geometric cues. Figure 11 illustrates the performance of the algorithm in a typical on road scene from the Urban Challenge.

4.3 Roadmap Localization

Boss is capable of either estimating road geometry or localizing itself relative to roads with known geometry. Most urban roads change shape infrequently, and most urban driving can be thought of as responding to local disturbances within the constraints of a fixed road network. Given that the shape and location of paved roads changes infrequently, our approach was to localize relative to paved roads and estimate the shape of dirt roads, which change geometry more frequently. This approach has two main advantages:

- it exploits a priori knowledge to eliminate the necessity of estimating road shape in most cases;
- it enables the road shape estimation problem to emphasize geometric cues such as berms and bushes, which are common in environments with dirt roads, and easier to detect at long range than lane markings.

This approach led to two independent algorithms, one to provide a smooth pose relative to a road network, and one to estimate the shape of dirt roads. The two algorithms are never operated simultaneously, thus avoiding complex interactions between them. Both the localization and road shape estimation algorithms were heavily tested and proved effective. Despite confidence in the road shape estimation system, it was not enabled during the Urban Challenge competition. Based on the waypoint density and aerial imagery of the UCFE course, the team determined that there was not a need to estimate road shape. A description of the road shape estimation approach is provided in section 4.4 for completeness since it enables Boss to drive on general roads and was ready to be used in the event, if necessary.

Localization Inputs

The localization process can be thought of as transforming the pose provided by a GPS-based pose estimation system into a smooth coordinate frame registered to a

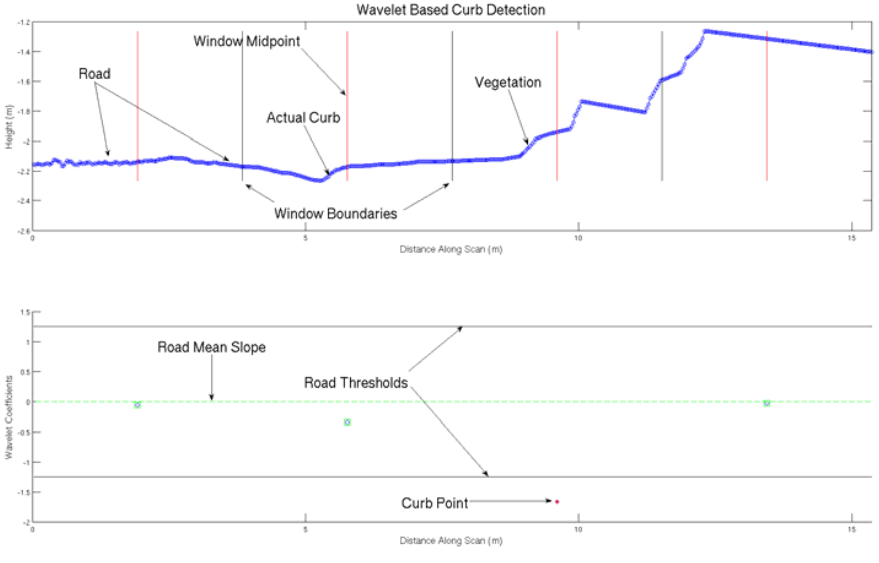
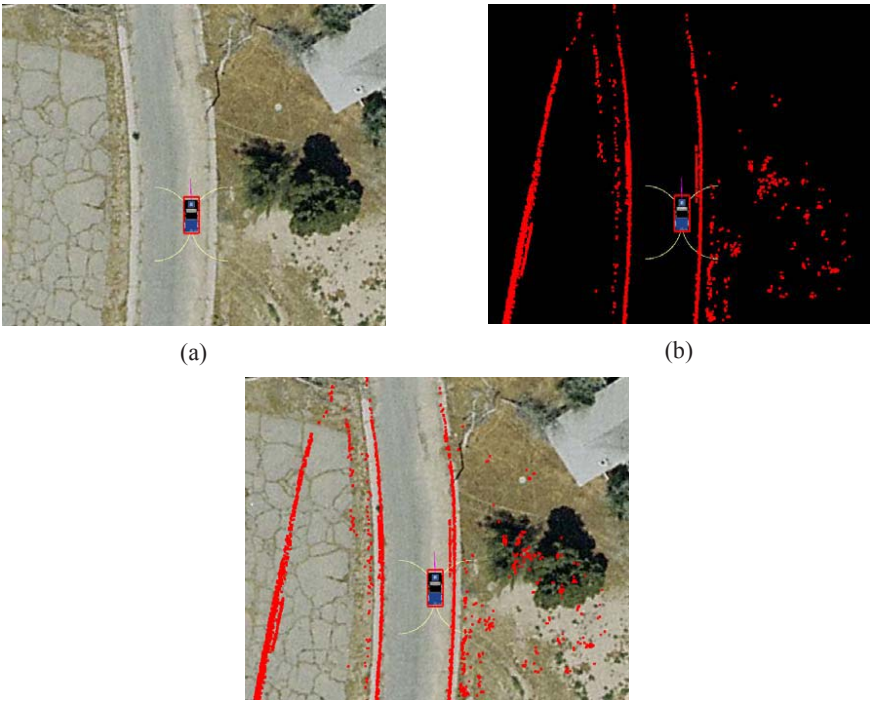


Fig. 10. Single frame of the feature extraction. The top frame contains the original height signal in blue. The black vertical lines show the sample windows from a single detail level of the transform. The red vertical lines define the midpoint of each window. The bottom frame show the wavelet coefficients for each window, labeled as road (green) or non-road (red).



(a) (b)

(c)

Fig. 11. Overhead view of a road section from the Final Event course (a). Red points show non-road points (b). Overlay of non-road points on imagery (c).

Table 2. Error characteristics of the Applanix POS-LV, as reported by Applannix.

	Error with GPS & differential corrections	Error after 1km of travel without GPS
Planar Position (m)	0.3	0.88
Heading (°)	0.05	0.07

road network. To do this it combines data from a commercially available position estimation system and measurements of road lane markers with an annotated road map.

The initial global position estimate is received from a device (POS-LV) developed by the Applanix Corporation. This system fuses GPS, inertial and wheel encoder data to provide a 100Hz position estimate that is robust to GPS dropout. Table 2 describes the nominal performance of this system with and without GPS. The POS-LV is configured to slightly outperform the nominal performance specifications through the use of a combination of Omnistar Virtual

Base Station and High Precision *s*ervices. By incorporating the High Precision data, nominal performance is improved to a 0.1m planar expected positioning error. While a positioning accuracy of 0.1m sounds sufficient to blindly localize within a lane, these correction signals are frequently disrupted by even small amounts of overhead vegetation. Once disrupted, this signal's reacquisition takes approximately a half hour. Thus, relying on these corrections is not viable for urban driving. Furthermore, lane geometries may not be known to meter accuracies a priori. It is critically important to be localized correctly relative to the lane boundaries, since crossing over the lane center could have disastrous consequences.

To detect lane boundaries, down-looking Sick LMS lasers are used to detect the painted lane markers on roads. Lane markers are generally brighter than the surrounding road material and are detected by convolving the intensities across a line scan with a slope function. Peaks and troughs in the response represent the edges of potential lane marker boundaries. To reduce false positives, only appropriately spaced pairs of peaks and troughs are considered lane markers. Candidate markers are then further filtered based on their brightness relative to their support region. The result is a set of potential lane marker positions.

The road map used for localization encodes both correct local geometry and information about the presence or absence of lane markings. While it is possible of road geometry to be incorrect globally, the local geometry is important to the estimation scheme, as will be described below. If the road geometry is not well known, the map must indicate this. When the vehicle traverses parts of the map with poor geometry, the road shape estimation algorithms operate and the road map localization algorithms are disabled.

Position Filtering

To transform the measurements provided by the POS-LV to a smooth, road-network- registered frame, we consider three potential sources of position error:

1. *Position Jumps*- despite the availability of inertial information, the POS-LV will occasionally generate position jumps.
2. *Position Drift*- the correction signals, variation in satellite constellation and ionospheric disturbances cause slowly varying changes to the position reported by the POS-LV.
3. *Road model errors*- our approach to creating road maps is to manually extract road shapes from aerial imagery. Modern aerial imagery can provide quarter-meter or better image resolution, but global registration is generally only good to a meter or worse. Distortion in the imagery generally has a low spatial frequency, so that the local shape of the road is accurate, but the apparent global position may be inaccurate.

These three error sources are grouped into two classes; discontinuous errors (such as jumps) and continuous errors (drift and model errors). With every new state measurement, the change in position ($\Delta\mathbf{x}$) is checked for validity based on

measured wheel speed (v), anticipated percentage velocity error (ζ), allowed position jitter (ε), travel direction (θ), and allowable travel direction error (τ):

$$reject = |\Delta \mathbf{x}| > v\left(1+\zeta\right)\Delta t + \varepsilon \quad \vee \quad \left(\left(|\Delta \mathbf{x}| > \varepsilon\right) \wedge \frac{\Delta \mathbf{x}}{|\Delta \mathbf{x}|} \cdot \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} > \tau\right) \tag{12}$$

In the above equation, the first term ensures that the reported motion of the vehicle is not significantly greater than the anticipated motion given the vehicle wheel speed. The second term ensures that for any significant motion, the reported motion is in approximately the same direction as the vehicle is pointed (which is expected for the speeds and conditions of the Urban Challenge). If $\Delta \mathbf{x}$ is rejected, a predicted motion is calculated based on heading and wheel speed. The residual between the prediction and the measured change in position is accumulated in a running sum which is subtracted from position estimates reported by the POS-LV. In practice, values of ζ=0.05, ε=0.02 and τ=cos(30°) produce good performance.

Correcting for the continuous class of errors is how localization to the road model is performed. The localization process accumulates lane marker points (LMP) generated by the laser lane marker detection algorithms. After a short travel distance, 1m during the Urban Challenge, each LMP is associated with a lane boundary described in the road model. The distance of the corrected global position (p) for each LMP from the lane center is calculated and the projection point onto the lane center is noted (p_c). Half of the lane width is subtracted from this distance, resulting in the magnitude of the local error estimate between the lane boundary position and the model of the lane boundary position. This process is repeated for each LMP, resulting in an error estimate:

$$e_{LMP} = \frac{1}{n_{LMP}} \sum_{1}^{n_{LMP}} \left(\left| \left(p^i - p_c^i\right) \right| - \frac{w_l^i}{2} \right) \cdot \left(\frac{p^i - p_c^i}{|p^i - p_c^i|} \right) \tag{13}$$

This represents the error in the current filtered/localized position estimate, thus the e_{LMP} represents how much error there is between the current combination of the existing error estimate and position. In practice, we further gate the error estimates, discarding any larger than some predetermined maximum error threshold (3m during the Urban Challenge). Over time, error estimates are accumulated through a recursive filter:

$$e_{cur} = e_{prev} + \alpha e_{LMP} \tag{14}$$

This approach generates a smooth, road-network-referenced position estimate, even in situations where GPS quality is insufficient to otherwise localize within a lane. This solution proved effective. During pre-challenge testing, we performed several tests with GPS signals denied (through the placement of aluminum caps over the GPS antennas). In one representative test, the vehicle was able to

maintain position within a lane, while traveling over 5.7km without GPS. During this test, the difference error in the POS-LV position reached up to 2.5m, more than enough to put the vehicle either off the road, or in another lane if not compensated for.

4.4 Road Shape Estimation

In order to robustly drive on roads where the geometry is not known a priori, the road shape estimator measures the curvature, position, and heading of roads near the vehicle. The estimator fuses inputs from a variety of LIDAR sensors, and cameras to composite a model of the road. The estimator is initialized using available prior road shape data, and generates a best-guess road location between designated sparse points where a road may twist and turn. The road shape is represented as the Taylor expansion of a clothoid with an offset normal to the direction of travel of the vehicle. This approximation is generated at 10 Hz.

Sensor Inputs

Three primary features were used to determine road location.

Curbs represent the edge of the road and are detected using the Haar wavelet (see Curb Detection and Mapping section). When curbs are detected, the estimator attempts to align the edge of the parametric model with the detections.

Obstacles represent areas where the road is unlikely to exist and are detected using the obstacle detection system. The estimator is less likely to pick a road location where obstacle density is high.

State Vector

To represent the parameters of a road, the following model is used:

$$s(t) = \left(x(t), y(t), \varphi(t), C_0(t), C_1(t), W(t)\right) \tag{15}$$

where $(x(t),y(t),\varphi(t))$ represent the origin and orientation of the base of the curve, $C_0(t)$ is the curvature of the road, $C_1(t)$ is the rate of curvature, and $W(t)$ is the road width. A Taylor series representation of a clothoid is used to generate the actual curve. This is represented as:

$$y(x) = \tan\left(\varphi(t)\right) x + C_0 \frac{t}{2} x^2 + C_1 \frac{t}{6} x^3 \tag{16}$$

Particle Filter

The road estimator uses an SIR (sample importance resample) (Duda & Hart, 1972) filter populated by 500 particles. Each particle is an instantiation of the state vector. During the sampling phase, each particle is propagated forward according to the following set of equations where **ds** represents the relative distance that the robot traveled from one iteration of the algorithm to the next.

$$y = y + \varphi ds + \frac{(ds)^2}{2} C_0 + \frac{(ds)^3}{6} C_1 \tag{17}$$

$$\varphi = \varphi + C_0 ds + \frac{(ds)^2}{2} C_1 - d\varphi \tag{18}$$

$$C_0 = C_0 + C_1 ds \tag{19}$$

$$C_1 = (0.99) C_1 \tag{20}$$

The final C_1 term represents the assumption that the curvature of a road will always tend to head towards zero, which helps to straighten out the particle over time. After the deterministic update, the particle filter adds random Gaussian noise to each of the dimensions of the particle in an effort to help explore sudden changes in the upcoming road curvature that are not modeled by the curve parameters. In addition to Gaussian noise, several more directed searches are performed, where the width of the road can randomly increase or decrease itself by a fixed amount. Empirically, this represents the case where a road suddenly becomes wider because a turn lane or a shoulder has suddenly appeared.

Sensor Data Processing

Because particle filtering requires the evaluation of a huge number of hypotheses (greater than ten thousand hypotheses per second in this case), it is desirable to be able to evaluate road likelihoods very quickly. Evaluations are typically distributed over a large proportion of the area around the vehicle, and occur at a high rate. Therefore, the likelihood evaluations were designed to have low computational cost, on average requiring one lookup per sample point along the road shape.

The likelihood function for the filter is represented as a log-linear cost function:

$$L = \frac{1}{Z} e^{-C(shape,data)} \tag{21}$$

In the above equation, Z is a normalization constant that forces the sum of the likelihoods over all road shapes to be one and C is a cost function that specifies the empirical "cost" of a road shape as a function of the available sensor data. The cost function is the sum of several terms represented by three sub-classes of cost function: distances, counts, and blockages.

The filter evaluates the number of obstacles, N_O, and number of curb points, N_C, encountered inside the road shape; the distance of the edge of the road to the detected curb points, D_C; the distance between the observed lane markers and the model's lane markers, D_L; and the presence of blockage across the road, B. In order to scale the cost function, counts and distances are normalized. The resulting cost function is:

$$C = \sum_{i=0}^{N} \left(\frac{N_O^i}{\sigma_O}\right)^2 + \left(\frac{N_C^i}{\sigma_C}\right)^2 + \left(\frac{D_C}{\sigma_C}\right)^2 + \left(\frac{D_L}{\sigma_L}\right)^2 \quad (22)$$

Fast Convolutions and Distance Transforms

In order to exactly compute the cost function, we need to convolve each road shape with the cost map to sum the detection counts and obstacle costs. This requires tens of thousands of convolutions per second for each count. While fast methods exist to exactly compute simple shapes (Viola & Jones, 2001), the shapes we wish to convolve are too complicated for these approaches. Instead, we approximate the road shape as a set of overlapping discs centered on the road shape (see Figure 12).

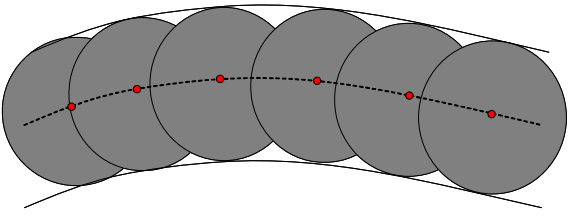


Fig. 12. Example illustrating how road shape is approximated by a series of disks.

The disks have a diameter equal to the width of the road and are spaced at 1.5 meter samplings. While this approach tends to over-count, we have found that it is adequate for the purposes of tracking the road, and is more than fast enough for our purposes.

In order to allow the width of the road to vary, we compute convolutions for different-width discs ranging from 2.2 meters to 15.2 meters sampled at half-meter spacing. Intermediate widths are interpolated.

Each width requires one convolution with a kernel size that varies linearly with the width of the road. Computing these convolutions for each frame is not possible, so the convolutions are computed iteratively. In the case of curb detections, curb points arrive and are tested against a binary map which indicates whether a curb point near the new detection has already been considered. If the location has not been considered, then the point is added to the convolution result by adding a disc at each radius to the map stack. In the case of an obstacle map, when a new map arrives, a difference map is computed between the current map and the previous convolution indicator. New obstacle detections are added into the convolution result as in the case of the point detection, while obstacles that have vanished are removed. The result of the convolutions is a set of cost maps that represent the road configuration space for each potential road width.

To evaluate the distance components of the cost function, we employ a distance transform (Huttenlocker & Felzenswalb, 2004). The distances from the nearest curb location or lane marker location to a given sample location is built into a

distance map. The distance map can then be examined at sample points and evaluated like the cost counts. Summing the overall cost function results in a minimum located at the true location of the road.

A snapshot of the overall system performance is illustrated in Figure 13. The example shows on an off-road stretch where some geometric features were visible in terms of berms and shrubbery as obstacles. The top of the figure shows the output from two cameras mounted on the top of Boss. The particle filter stretches forward from the vehicle and the road is represented as 3 lines.

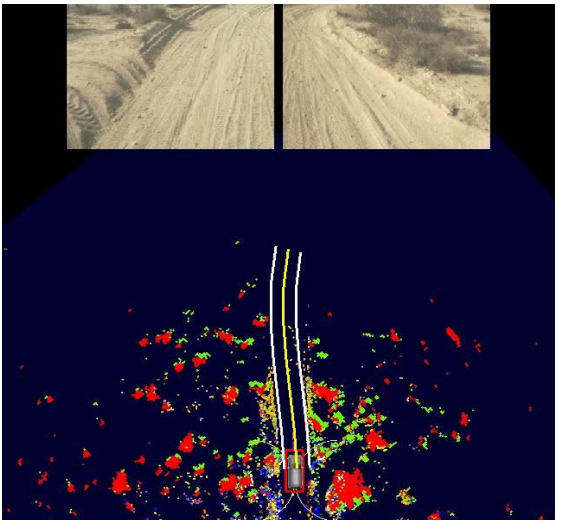


Fig. 13. Example showing the road shape estimate (parallel curves) for an off-road scene. Obstacles and berms are illustrated by colored pixels.

5 Mission Planning

To generate mission plans, the data provided in the road network definition file (RNDF) is used to create a graph that encodes the connectivity of the environment. Each waypoint in the RNDF becomes a node in this graph, and directional edges (representing lanes) are inserted between waypoints and all other waypoints they can reach. For instance, a waypoint defining a stop line at an intersection will have edges connecting it to all waypoints leaving the intersection that can be legally driven to. These edges are also assigned costs based on a combination of several factors, including expected time to traverse the edge, distance of the edge, and complexity of the corresponding area of the environment. The resultant cost graph is the baseline for travel road and lane decisions by the behavioral sub-system.

A value function is computed over this graph, providing the path from each waypoint to the current goal (e.g. the first checkpoint in a mission). In addition to providing the executive more information to reason about, computing a value

function is useful because it allows the navigation system to react appropriately if an execution error occurs (e.g. if the vehicle drives through an intersection rather than turning, the new best path to take is instantly available).

As the vehicle navigates through the environment, the mission planner updates its graph to incorporate newly-observed information such as road blockages. Each time a change is observed, the mission planner re-generates a new policy. Because the size of the graph is relatively small, this replanning can be performed quickly, allowing for near-immediate response to detected changes.

To correctly react to these occurrences, the robot must be able to detect when a road is impassable and plan another route to its goal, no matter where the goal is. Particularly difficult cases include planning to a goal immediately on the other side of a newly discovered blockage and behaving reasonably when a one-way street becomes blocked. Road conditions are fluid and highly variable and road blockages may not be permanent. Thus, a robot should eventually revisit the site of a previously encountered blockage to see if it has been cleared away. In fact, the robot *must* revisit a blockage if all other paths to a goal have also been found to be blocked, hoping to discover that a road blockage has been cleared.

5.1 Detecting Blockages

To determine there is a blockage Boss can either directly detect the blockage or infer it by a failure to navigate a lane. To directly detect a blockage, the road in front of Boss is checked against a static obstacle map to see if lethal obstacles completely cross the road. Noise in the obstacle map is suppressed by ignoring apparent blockages that have been observed for less than a time constant (nominally five seconds). Blockages are considered to no longer exist if the obstacle map shows a sufficiently wide corridor through the location where a blockage previously existed.

The direct detection algorithm generates obstacles using an efficient but optimistic algorithm; thus, there are configurations of obstacles that effectively block the road but are not directly detected as a road blockage. In these conditions, the On-Road navigation algorithm may report the road is blocked, inducing a *Virtual Blockage*. Since the behavior generation module works by picking the lowest cost path to its goal, this is an elegant way for it to stimulate itself to choose an alternate route. Since virtual blockages induced by the behavior generation module are not created due to something explicitly observed, they cannot be removed by observation; thus, they are removed each time the vehicle achieves a checkpoint. While heuristic, this approach works well in practice, as these blockages are often constructed due to odd geometries of temporary obstacles. By waiting until the current checkpoint is complete, this approach ensures the vehicle will wait until its current mission is complete before revisiting a location. If the only path to goal is through a virtual blockage that cannot be detected as cleared, and the situation that caused the blockage to be declared has been resolved, then forward progress will still occur, since the virtual blockages decay in the same manner that explicitly observed blockages do.

5.2 Blockages

Once a blockage has been detected, the extent along affected lanes that the blockage occupies is determined. Locations before and after the blockage are identified where U-turn maneuvers can be performed. At these locations, road model elements representing legal places to make a U-turn are added. Simultaneously, the corresponding traversal costs for the U-turn maneuvers are set to a low values, the costs for crossing the blockages are increased by a large amount, and the navigation policy is recomputed. Since Boss follows the policy, the high costs levied on traversing a blockage cause the robot to choose an alternate path. If Boss later detects that the blockage is gone, the traversal costs for elements crossing the blockage are restored to their defaults, and the traversal costs for the added U-turn elements are effectively removed from the graph.

Revisiting of previously detected blockages is implemented by gradually reducing the traversal cost applied to road elements crossing a blockage. If the cost eventually drops below a predefined threshold, the blockage is treated as if it were observed to be gone. The U-turn traversal costs are not concomitantly increased; instead, they are changed all at once when the blockage is observed or assumed to be gone. Decreasing the cross-blockage traversal costs encourages the robot to return to check whether a blockage is removed, while *not* increasing the U-turn traversal costs encourages the robot to continue to plan to traverse the U-turn if it is beneficial to do so.

The cost (*c)* increment added by a blockage is decayed exponentially:

$$c = p2^{-\frac{a}{h}} \tag{23}$$

where a is the time since the blockage was last observed, h is a half-life parameter, and p the starting cost penalty increment for blockages. To illustrate, if the blockage is new, we have $a=0$ and $c=p$. If the blockage was last observed h time units in the past, we have $a=h$ and $c=p/2$. The cost continues to decay exponentially as the blockage ages.

An exponential decay rate mitigates the problem of a single blockage being interpreted as multiple blockages (due to incomplete perception), causing a cost of np where n is the number of blockages. Under this condition, a linear decay rate would cause an unacceptably long delay before revisiting a blockage.

A weakness of this blockage handling approach is that it is possible to waste time making multiple visits to a blockage that never gets removed. A simple solution of incrementing h for the blockage after each new visit would make the traversal costs decay more slowly each time the obstacle is observed.

On one-way roads, U-turn lanes are not created in response to road blockages. However, traversal costs across the blockage are increased, decreasing the likelihood of reusing the road. To respond to one-way road blockages, the zone navigation planner is invoked as an error recovery mode, as discussed in section 6.3.

6 Behavioral Reasoning

The Behavioral architecture is responsible for executing the policy generated by the Mission Planner; making lane-change, precedence, and safety decisions

respectively on roads, at intersections, and at yields; and responding to and recovering from anomalous situations.

The Behavioral architecture is based on the concept of identifying a set of driving contexts, each of which requires the vehicle to focus on a reduced set of environmental features. At the highest level of this design, the three contexts are road, intersection, and zone, and their corresponding behaviors are respectively *Lane Driving*, *Intersection Handling*, and *Achieving a Zone Pose*. The achieving a zone pose behavior is meant for unstructured or unconstrained environments, including parking lots and jammed intersections. In practice this behavior's function is performed by the zone planner. Figure 14 shows a diagram of Behavioral sub-system architecture with the sub-behaviors corresponding to the high-level behaviors, along with two sub-behaviors making up the auxiliary Goal Selection behavior, which plays a crucial role not only in standard operation, but also in error recovery. The function of each of these sub-components is described in Table 3.

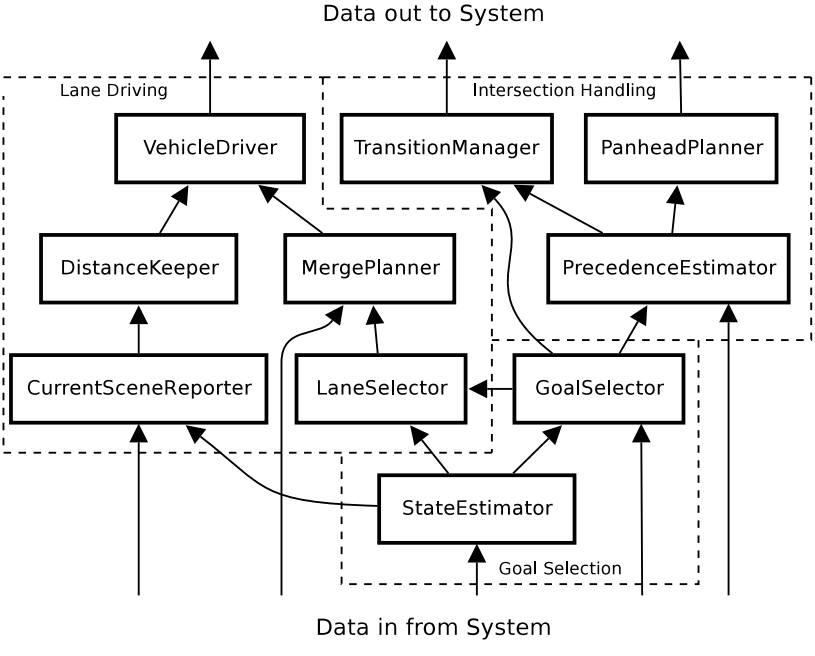


Fig. 14. High-Level Behaviors Architecture.

6.1 Intersections and Yielding

The *Precedence Estimator* is most directly responsible for the system's adherence to the Urban Challenge rules (DARPA 2007) including obeying precedence, not entering an intersection when another vehicle is in it, and being able to merge into

Table 3. Components of the Behavioral sub-system.

Goal Selection Components	Drive Down Road	Handle Intersection
***State Estimator*:** combines the vehicle's position with the world model to produce a discrete and semantically rich representation of the vehicle's logical position with the RNDF. ***Goal Selector*:** uses the current logical location as reported by *State Estimator* to generate the next series of local goals for execution by the Motion Planner, these will either be lane goals or zone goals.	***Lane Selector*:** uses the surrounding traffic conditions to determine the optimal lane to be in at any instant and executes a merge into that lane if it is feasible. ***Merge Planner*:** determines the feasibility of a merge into a lane proposed by *Lane Selector*. ***Current Scene Reporter*:** The Current Scene Reporter distills the list of known vehicles and discrete obstacles into a few discrete data elements, most notably the distance to and velocity of the nearest vehicle in front of Boss in the current lane. ***Distance Keeper*:** uses the surrounding traffic conditions to determine the necessary in-lane vehicle safety gaps and govern the vehicle's speed accordingly. ***VehicleDriver*:** combines the outputs of *Distance Keeper* and *Lane Selector* with its own internal rules to generate a so-called "Motion Parameters" message, which governs details such as the vehicle's speed, acceleration and desired tracking lane.	***Precedence Estimator*:** uses the list of known other vehicles and their state information to determine precedence at an intersection. ***Pan-head Planner*:** aims the pan-head sensors to gain the most relevant information for intersection precedence decisions. ***Transition Manager*:** manages the discrete-goal interface between the Behavioral Executive and the Motion Planner, using the goals from *Goal Selector* and the gating function from *Precedence Estimator* to determine when to transmit the next sequence of goals.

and across moving traffic. To follow the rules, the precedence estimator combines data from the rest of the system to determine if it is clear to go. This state is used as a gate condition in the *Transition Manager* and triggers the issuance of the motion goal to proceed through the intersection.

The precedence estimator uses a combination of the road model and moving obstacle information to determine if it is clear to go. The road model provides static information about an intersection, including the set of lane exit waypoints that compose the intersection and the geometry of their associated lanes. The moving obstacle set provides dynamic information about the location, size and speed of estimated nearby vehicles. Given the problem of spatial uncertainty, false positives and false negatives must be accounted for in the precedence estimation system.

The road model provides important data, including:

- The current intersection of interest, which is maintained in the world model as a group of exit waypoints, some subset of which will also be stop lines;
- A virtual lane representing the action the system will take at that intersection;
- A set of yield lanes[1] for that virtual lane; and
- Geometry and speed limits for those lanes and any necessary predecessor lanes.

These data are known in advance of arrival at the intersection, are of high accuracy, and are completely static. Thus, the Precedence estimator can use them to preprocess an intersection's geometry.

The moving obstacle set is received periodically and represents the location, size and speed of all detected vehicles around the robot. In contrast to the information gleaned from the road model, these data are highly dynamic, displaying several properties which must be accounted for in the precedence estimation system: tracked vehicles can flicker in and out of existence for short durations of time; sensing and modeling uncertainties can affect the estimated shape, position and velocity of a vehicle; and, the process of determining moving obstacles from sensor data may represent a vehicle as a small collection of moving obstacles. Among other things, this negates the usefulness of attempting to track specific vehicles through an intersection and requires an intersection-centric (as opposed to vehicle-centric) precedence estimation algorithm.

Intersection-Centric Precedence Estimation

Within the road model, an intersection is defined as a group of lane exit-waypoints. That is, an intersection must contain one or more lane exit-waypoints,

[1] Yield lanes are lanes of moving traffic for which a vehicle must wait for a clear opportunity to execute the associated maneuver.

and each lane exit-waypoint will be a part of exactly one intersection. The next intersection is thus determined as the intersection containing the next lane exit-waypoint that will be encountered. This excludes exits that Boss will cross but not stop at (e.g. crossing the top of a tee-intersection that has only one stop sign).

Precedence between any two exit-waypoints is determined first by whether the exit-waypoints are stop lines. Non-stop exit-waypoints automatically have precedence over exit-waypoints that have stop lines. Among stop line exit-waypoints, precedence is determined by arrival times, where earlier arrivals have precedence over later arrivals.

The robust computation of arrival time is critical to the correct operation of the precedence estimator. Given the dynamic and noisy nature of the moving obstacle set, the algorithm uses a purely geometric and instantaneous notion of waypoint occupancy for computing arrival times. An exit waypoint is considered to be occupied when any vehicle's estimated front bumper is inside or intersects a small polygon around the waypoint, called its *Occupancy Polygon*. Boss's front bumper is added to the pool of estimated front bumpers and is treated no differently for the purposes of precedence estimation.

Occupancy polygons are constructed for each exit-waypoint and for the whole intersection. The occupancy polygons for each exit waypoint are used to determine precedence, where the occupancy polygon constructed for the intersection is used to determine whether the intersection is clear of other traffic. The size of the polygon for an exit waypoint determines several factors:

- The point at which a vehicle gains its precedence ordering, which is actually some length along the lane backward from the stopline;
- The system's robustness to spatial noise, where larger polygons are generally more robust than smaller ones at retaining the precedence order;
- The system's ability to discriminate two cars moving through the intersection in sequence, where larger polygons are more likely to treat two discrete cars as one for the purposes of precedence ordering.

Figure 15 shows a typical exit occupancy polygon extending three meters back along the lane from the stop line and with one meter of padding on all sides. This is the configuration that was used on race day.

The estimated front bumper of a vehicle must be inside the occupancy polygon as shown in Figure 15 to be considered to be an occupant of that polygon.

A given occupancy polygon maintains its associated exit waypoint, its occupancy state and two pieces of temporal data:

1. The time of first occupancy, which is used to determine precedence ordering; and
2. The time of most recent (last) occupancy, which is used to implement a temporal hysteresis around when the polygon becomes unoccupied.

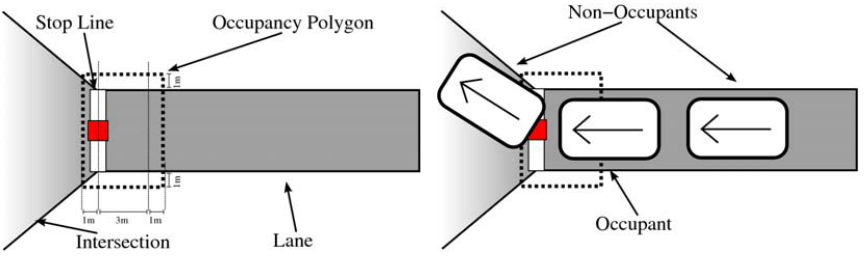


Fig. 15. Typical Exit Occupancy Polygon and examples of vehicles at an exit.

To account for (nearly) simultaneous arrival, arrival times are biased for the sake of precedence estimation by some small time factor that is a function of their position relative to Boss's exit-waypoint. Exit waypoints that are *to the right* receive a negative bias, and are thus treated as having arrived slightly earlier than in actuality, encoding an implicit yield-to-right rule. Similarly, exit waypoints that are *to the left* receive a positive bias, seeming to have arrived later and thus causing the system to take precedence from the left (empirically, 0.5s worked well for this value). The result is considered to be the exit waypoint's *Modified Arrival Time*.

With these data available, the determination of precedence order becomes a matter of sorting the occupied polygons in ascending order by their modified arrival time. The resulting list is a direct representation of the estimated precedence ordering, and when the front of that list represents Boss's target exit waypoint, Boss is considered to have precedence at that intersection.

Yielding

Beyond interacting with stopped traffic, the precedence estimator is also responsible for merging into or across moving traffic from a stop. To support this, the system maintains a *Next Intersection Goal*, which is invariably a *Virtual Lane* that connects the target exit waypoint to some other waypoint, nominally in another lane or in a parking or obstacle zone. That virtual lane has an associated set of *Yield Lanes*, which are other lanes that must be considered for moving traffic in order to take the next intersection action. The yield lanes are defined as the real lanes which overlap a virtual lane. Overlapped virtual lanes are not considered since they must already have a clearly established precedence order via stop lines. Intersections that fail this requirement (e.g. an intersection with four yield signs) are considered ill-formed and are not guaranteed to be handled correctly. Thus, yield cases are only considered for merges into or across real lanes. Figure 16 shows an example tee intersection, highlighting the next intersection goal and the associated yield lanes.

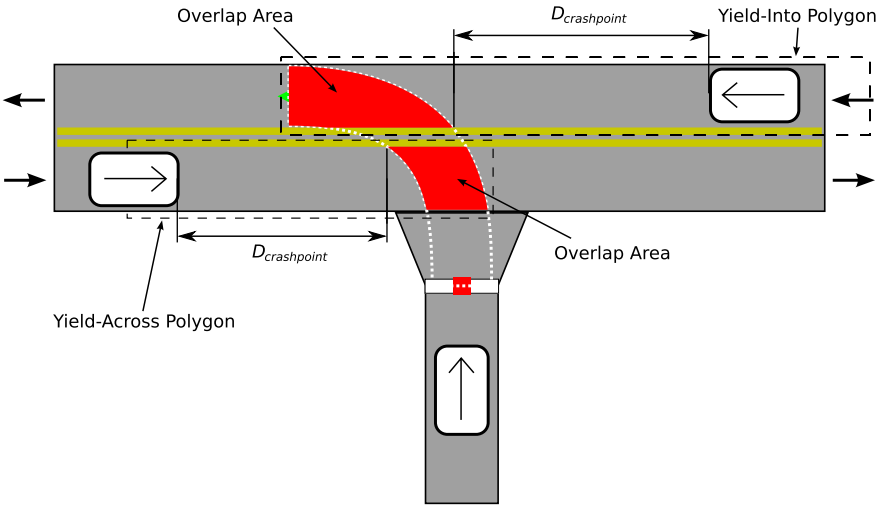


Fig. 16. Typical Tee-intersection with yield lanes.

First, temporal requirements are derived for the next intersection goal as follows:

1. T_{action} is computed as the time to traverse the intersection and get into the target lane using conservative accelerations from a starting speed of zero.
2. $T_{accelarate}$ is computed as the time for accelerating from zero up to speed in the destination lane using the same conservative acceleration.
3. T_{delay} is estimated as the maximum system delay.
4. $T_{spacing}$ is defined as the minimum required temporal spacing between vehicles, where one second approximates a vehicle-length per 10mph.

Using these values, a required temporal window, $T_{required}$, is computed for each yield lane as:

$$T_{required} = T_{action} + T_{delay} + T_{spacing} \tag{24}$$

for lanes that are crossed by the next intersection action. In the case of merging into a lane, the required window is extended to include the acceleration time, if necessary, as:

$$T_{required} = \max(T_{action}, T_{accelerate}) + T_{delay} + T_{spacing} \tag{25}$$

This temporal window is then used to construct a polygon similar to an exit occupancy polygon backward along the road network for a distance of:

$$\ell_{yield\,polygon} = v_{maxlane} T_{required} + d_{safety} \tag{26}$$

These yield polygons, shown in Figure 16, are used as a first pass for determining cars that are relevant to the yield window computations.

Any reported vehicle that is inside or overlaps the yield polygon is considered in the determination of the available yield window. Yield polygons are also provided to a *Panhead Planner*, which performs coverage optimization to point long-range sensors along the yield lanes and thus at oncoming traffic, increasing the probability of detection for these vehicles at long range.

For each such vehicle in a yield lane, a time of arrival is estimated at the near edge of the overlap area, called the *Crash Point* and illustrated in Figure 16 as follows:

1. Compute a worst-case speed $v_{obstacle}$ along the yield lane by projecting the reported velocity vector, plus one standard deviation, onto the yield lane.
2. Compute d_{crash} as the length along the road network from that projected point to the leading edge of the overlap area.
3. Compute an estimated time of arrival as:

$$T_{arrival} = \frac{d_{crash}}{v_{obstacle}} \tag{27}$$

4. Retain the minimum $T_{arrival}$ as $T_{current}$ over all relevant vehicles per yield lane.

The yield window for the overall intersection action is considered to be instantaneously open when $T_{current} > T_{required}$ for all yield lanes. In order to account for the possibility of tracked vehicles being lost momentarily, as in the exit waypoint precedence determination, this notion of instantaneous clearance is protected by a one-second hysteresis. That is, all yield windows must be continuously open for at least one second before yield clearance is passed to the rest of the system.

Gridlock Management

With exit precedence and yield clearance in place, the third and final element of intersection handling is the detection and prevention of gridlock situations. Gridlock is determined simply as a vehicle (or other obstacle) blocking the path of travel immediately after the next intersection goal such that the completion of the next intersection goal is not immediately feasible (i.e., a situation that would cause Boss to become stopped in an intersection).

Gridlock management comes into effect once the system determines that Boss has precedence at the current intersection and begins with a 15-second timeout to give the problematic vehicle an opportunity to clear. If still gridlocked after 15 seconds, the current intersection action is marked as locally high-cost and the mission planner is allowed to determine if an alternate path to goal exists. If so, Boss will re-route along that alternate path; otherwise, the system jumps into error recovery for intersection goals, using the generalized pose planner to find a way

around the presumed-dead vehicle. This is discussed in greater detail in the section describing error recovery.

6.2 Distance Keeping and Merge Planning

The distance-keeping behavior aims simultaneously to zero the difference between Boss' velocity and that of the vehicle in front of Boss, and the difference between the desired and actual inter-vehicle gaps. The commanded velocity is:

$$v_{cmd} = K_{gap}\left(d_{actual} - d_{desired}\right) \tag{28}$$

where v_{target} is the target-vehicle velocity and K_{gap} is the gap gain. The desired gap is:

$$d_{desired} = \max\left(\frac{\ell_{vehicle}}{10} v_{actual}, d_{mingap}\right) \tag{29}$$

where $\ell_{vehicle}$ term represents the one-vehicle-length-per-ten-mph minimum-separation requirement, and d_{mingap} is the absolute minimum gap requirement. When Boss's velocity exceeds the target vehicle's, its deceleration is set to a single configurable default value; when Boss's velocity is less than the target vehicle's, for safety and smoothness, Boss's commanded acceleration is made proportional to the difference between the commanded and actual velocities and capped at maximum and minimum values (a_{max} =4.0 and a_{min}=1.0 m/sec^2 on race day) according to:

$$a_{cmd} = a_{\min} + K_{acc}\, v_{vmd}\left(a_{\max} - a_{\min}\right) \tag{30}$$

Merge Planning

The merge, or lane-change, planner determines the feasibility of changing lanes. It is relevant not only on a unidirectional multi-lane road, but also on a bidirectional two-lane road in order to handle passing a stopped vehicle after coming to a stop. Feasibility is based on the ability to maintain proper spacing with surrounding vehicles and to reach a checkpoint in the lane to merge into (the "merge-to" lane) while meeting a velocity constraint at the checkpoint. The two-lane unidirectional case is depicted in Figure 17. The merge planner performs the following steps:

1. Check whether it is possible to reach the checkpoint in the merge-to lane from the initial position given velocity and acceleration constraints and the "merge distance", i.e., the distance required for Boss to move from its current lane into an adjacent lane. For simplicity's sake, the merge distance was made a constant parameter whose setting based on experimentation was 12m on race day.
2. Determine the merge-by distance, i.e., the allowable distance in the current lane in order to complete the merge. The merge-by distance in the case of a moving obstacle in front of Boss is:

$$d_{obst} = \frac{v_0 d_{initial}}{v_0 - v_1} \tag{31}$$

where $d_{initial}$ is the initial distance to the moving obstacle. Note that this reduces to $d_{initial}$ if the obstacle is still, i.e. if v_1=0.

3. For each of the obstacles in the merge-to lane, determine whether a front-merge (overtaking the obstacle and merging into its lane in front of it with proper spacing) is feasible and whether a back-merge (dropping behind the obstacle and merging behind it with proper spacing) is feasible.

   For either a front- or back-merge, first determine whether proper spacing is already met. For a front merge, this means:

$$x_0 - \ell_{vehicle} - x_1 \geq \max\left(\frac{v_1 \ell_{vehicle}}{10}, d_{mingap}\right) \tag{32}$$

   For a back merge:

$$x_1 - \ell_{vehicle} - x_0 \geq \max\left(\frac{v_0 \ell_{vehicle}}{10}, d_{mingap}\right) \tag{33}$$

   If proper spacing is met:

   Check whether the other vehicle's velocity can be matched by acceleration or deceleration after the merge without proper spacing being violated. If so, the merge is so far feasible; if not, the merge is infeasible.

   Otherwise:

   Determine the acceleration profile to accelerate or decelerate respectively to, and remain at, either the maximum or minimum speed until proper spacing is reached.

4. Check whether it is possible to reach and meet the velocity constraint at the checkpoint in the merge-to lane starting from the merge point, i.e., the position and velocity reached in the previous step after proper spacing has been met. If so, the merge is feasible.
5. Repeat the above steps for all n obstacles in the merge-to lane. There are n+1 "slots" into which a merge can take place, one each at the front and rear of the line of obstacles, the rest in between obstacles. The feasibility of the front and rear slots is associated with a single obstacle and therefore already determined by the foregoing. A "between" slot is feasible if the following criteria are met: 1) the slot's front-obstacle back-merge and rear-obstacle front-merge are feasible; 2) the gap between obstacles is large enough for Boss plus proper spacing in front and rear; 3) the front obstacle's velocity is greater than or equal to the rear obstacle's velocity, so the gap is not closing; 4) the merge-between point will be reached before the checkpoint.

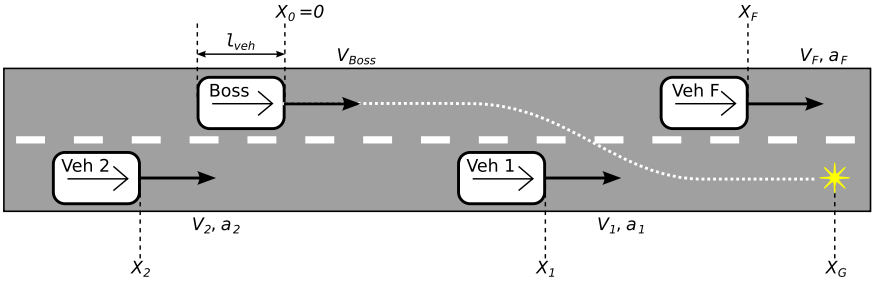


Fig. 17. Two-Lane Merging

Boss determines whether a merge is feasible in all slots in the merge-to lane (there are three in the example: in front of vehicle 1, between vehicles 1 and 2, and behind vehicle 2), and targets the appropriate feasible merge slot depending on the situation. For a multi-lane unidirectional road, this is generally the foremost feasible slot.

Once feasibility for all slots is determined, appropriate logic is applied to determine which slot to merge into depending on the situation. For a multi-lane unidirectional road, Boss seeks the foremost feasible slot in order to make the best time. For a two-lane bidirectional road, Boss seeks the closest feasible slot in order to remain in the wrong-direction lane for the shortest time possible.

Determination of the merge-by distance is the smallest of the distance to the: 1) next motion goal (i.e., checkpoint); 2) end of the current lane; 3) closest road blockage in the current lane; 4) projected position of the closest moving obstacle in the current lane.

6.3 Error Recovery

One of the most important aspects of the behavioral reasoning system is its responsibility to detect and address errors from the motion planner and other aberrant situations. To be effective, the recovery system should:

- Be able to generate a non-repeating and novel sequence of recovery goals in the face of repeated failures ad-infinitum.
- Be able to generate different sets of recovery goals to handle different contexts.
- Be implemented with minimal complexity so as to produce as few undesirable behaviors as possible.

To reduce interface complexity, the goal selection system follows the state graph shown in Figure 18.

Each edge represents the successful completion (Success) or the failed termination (Failure) of the current motion goal. Goal failure can be either directly reported by the motion planner, or triggered internally by progress monitoring that

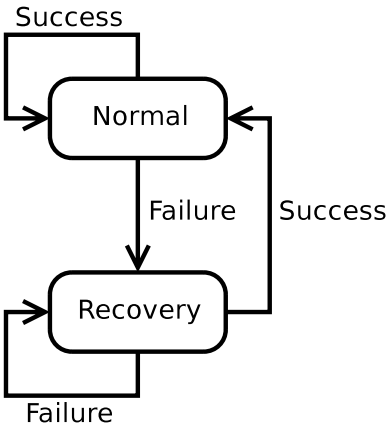


Fig. 18. Goal selection state graph.

declares failure if sufficient progress is not made within some time. All edge transitions trigger the selection of a new goal and modify the *Recovery Level*:

- **Success** resets recovery level to zero but caches the previous recovery level.
- **Failure** sets the recovery level to one greater than the maximum of the cached and current recovery level.

The recovery level along with the type and parameters of the original failed goal are the primary influences on the recovery goal algorithms. The general form of the recovery goals is that an increasing recovery level results in higher risk attempts to recover, meaning actions that are generally farther away from the current position and/or the original goal. This process ensures that Boss tries low-risk and easy to execute maneuvers initially while still considering more drastic measures when necessary. If the recovery process chooses an obviously-infeasible goal, the motion planner will signal failure immediately and the recovery level will increment. Otherwise, to complement explicit failure reports from the motion planner, forward progress is monitored such that the robot must move a minimum distance toward the goal over some span of time. If it does not, the current goal is treated as a failure, and the recovery level is incremented. The progress threshold and time span were determined largely by experimentation and set to 10m and 90s on race day. Generally speaking, these represent the largest realistic delay the system was expected to encounter during operation.

In general, the successful completion of a recovery goal sets the system back to normal operation. This eliminates the possibility of complex multi-maneuver recovery schemes, at the benefit of simplifying the recovery state tracking. In situations where the vehicle oscillates between recovery and normal operation, the recovery system maintains sufficient state to increase the complexity of recovery maneuvers.

On-Road Failures

The most commonly encountered recovery situation occurs when the on-road planner generates an error while driving down a lane. Any number of stimuli can trigger this behavior, including.

- Small or transient obstacles, e.g. traffic cones that do not block the entire lane but are sufficient to prevent the planner from finding a safe path through them.
- Larger obstacles such as road barrels, K-rails or other cars that are detected too late for normal distance keeping to bring the system to a graceful stop,
- Low-hanging canopy, which is generally detected late and often requires additional caution and careful planning, and
- Lanes whose shape is kinematically infeasible.

The algorithm for lane recovery goal selection, called *Shimmy*, and illustrated in Figure 19, selects an initial set of goals forward along the lane with the distance forward described by:

$$d_{shimmy} = d_{initial} + Rd_{incremental} \tag{34}$$

Empirically, $d_{initial}$=20m and $d_{incremental}$=10m worked well. The 20m initial distance was empirically found to give the planner sufficient room to get past stopped cars in front of the vehicle.

These forward goals (Goals 1,2,3 in Figure 19) are selected out to some maximum distance, roughly 40*m* and corresponding to our high-fidelity sensor-range, after which a goal is selected 10*m* behind the vehicle (Goal 4) with the intent of backing up and getting a different perspective on the immediate impediment.

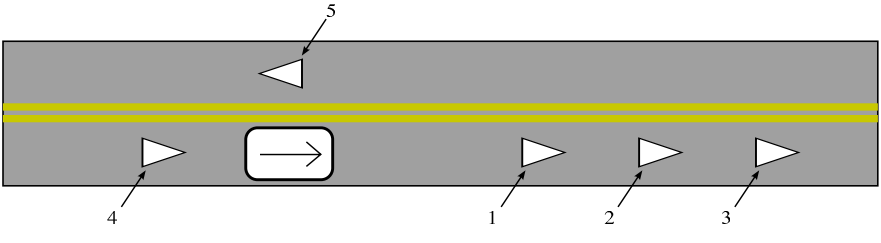


Fig. 19. Example Shimmy error recovery goal sequence.

After backing up, the sequence of forward goals is allowed to repeat once more with slight (less than 5m) alterations after which continued failure causes one of two things to happen:

1. If a lane is available in the opposing direction, mark the segment as locally blocked and issue a U-turn (Goal 5). This is supplemental to the external treatment of total segment blockages discussed in section 5.2 , and presumes that a U-turn has not yet been detected and generated.

2. If there are no lanes available in the opposing direction (i.e., Boss is stuck on a one-way road), then the goal selection process is allowed to continue forward infinitely beyond the 40*m* limit with the additional effect of removing an implicit *stay near the lane* constraint that is associated with all previous recovery goals. Removing this constraint gives the pose planner complete freedom to wander the world arbitrarily in an attempt to achieve some forward goal.

Intersection Failures

Error cases in intersections are perhaps the most difficult to recover from. These errors happen when an attempt to traverse an intersection is not possible due to obstacles and/or kinematic constraints, or else as part of the gridlock resolution system. A simplified example sequence from this recovery algorithm, called *Jimmy*, is show in Figure 20.

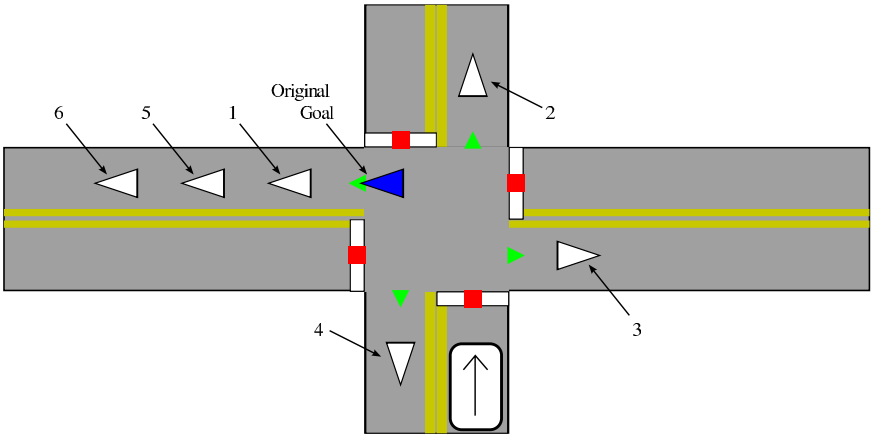


Fig. 20. Example Jimmy recovery goal sequence.

The first step in the Jimmy algorithm is to try the original failed goal over again as a pose goal, instead of a road driving goal (e.g. Goal), giving the motion planner the whole intersection as its workspace instead of just the smooth path between the intersection entry and exit. This generally and quickly recovers from small or spurious failures in intersections as well as compensating for intersections with turns that are tighter than the vehicle can make in a single motion. Should that first recovery goal fail, its associated entry waypoint is marked as blocked, and the system is allowed an opportunity to plan an alternate mission plan to the goal. If there is an alternate path, that alternate intersection goal (e.g. Goals 2,3) is selected as the next normal goal and the cycle is allowed to continue. If that alternate goal fails, it is also marked as blocked, and the system is allowed to replan, and so forth until all alternate routes to goal are exhausted, whereupon the system will select unconstrained pose goals (i.e. pose goals that can drive outside

of the intersection and roads) incrementally further away from the intersection along the original failed goal.

Zone Failures

The case where specifics do matter, however, is the third recovery scenario, failures in zones. The pose planner that executes zone goals is general and powerful enough to find a path to any specific pose if such a path exists, so a failure to do so implies one of the following:

1. The path to the goal has been transiently blocked by another vehicle. While DARPA guaranteed that a parking spot in a zone will be free, they made no such guarantees about traffic accumulations at zone exits, or about the number of vehicles between the current position and the target parking spot. In either case, a retry of the same or a selection of a nearby similar goal should afford the transient blockage time to pass.
2. Due to some sensor artifact, the system believes there is no viable path to goal. In this case, selecting nearby goals that offer different perspectives on the original goal area may relieve the situation.
3. The path to the goal is actually blocked.

The goal selection algorithm for failed zone goals, called *Shake*, selects goals in a regular, triangular pattern facing the original goal as shown in Figure 21.

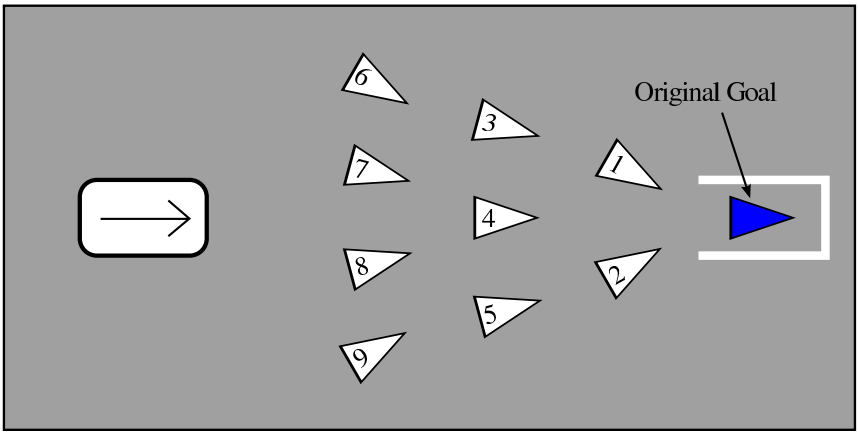


Fig. 21. Example Shake recovery goal sequence.

On successful completion of any one these goals, the original goal is re-attempted. If the original goal fails again, the Shake pattern picks up where it left off. If this continues through the entire pattern, the next set of actions is determined by the original goal. Parking spot goals were guaranteed to be empty, so the pattern is repeated with a small incremental angular offset ad-infinitum. For zone exit waypoint goals, the exit is marked as blocked and the system attempts to

re-route through alternate exits similar to the alternate path selection in the Jimmy algorithm. In the case of no other exits, or no other path to goal, the system issues completely unconstrained goals to logical successors of the zone exit waypoint in a last-ditch effort to escape the zone.

If these goals continue to fail, then farther successors are selected in a semi-random breadth-first search along the road network in a general last-ditch recovery algorithm called *Bake*. Increasing values of recovery level call out farther paths in the search algorithm. The goals selected in this manner are characterized by being completely unconstrained, loosely specified goals that are meant to be invoked when each of the other recovery goal selection schemes have been exhausted. In addition to Shake goals at a zone exit, these are selected for Shimmy goals that run off the end of the lane and similarly for Jimmy goals when all other attempts to get out of an intersection have failed.

Through these four recovery algorithms (Shimmy, Jimmy, Shake and Bake) many forward paths are explored from any single location, leaving only the possibility that the system is stuck due to a local sensor artifact or some other strange local minima that requires a small local adjustment. To address this possibility, a completely separate recovery mechanism runs in parallel to the rest of the goal monitoring and recovery system with a very simple rule: if the system has not moved at least one meter in the last five minutes, override the current goal with a randomized local goal. When that goal is completed, pretend a completely fresh wakeup at that location, possibly clearing accumulated state, and try again.

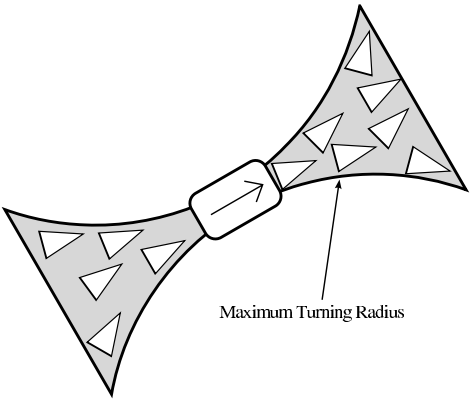


Fig. 22. Example Wiggle recovery goals.

The goals selected by this algorithm, called *Wiggle*, are illustrated in Figure 22. The goals are pseudo-random, approximately kinematically feasible and can be either in front of or behind the vehicle's current pose. The algorithm is, however, biased somewhat forward of the robot's position so there is statistically net-forward motion if the robot is forced to choose these goals repeatedly over time in a behavior similar to the "Wander" behavior described by 0.

The composite recovery system provides a necessary line of defense against failures in the planning and perceptive systems. This robustness was a key element of winning the Urban Challenge.

7 Software Infrastructure

The software infrastructure is a tool box that provides the basic tools required to build a robotic platform. The infrastructure takes the form of common libraries that provide fundamental capability, such as inter-process communication, common data types, robotic math routines, data log/playback, and much more. Additionally, the infrastructure reinforces a standard mechanism for processes in the system to exchange, log, replay, and visualize data across any interface in the system, thereby reducing the time to develop and test new modules.

The following is a list of the tools provided by the infrastructure:

Communications Library – Abstracts around basic inter-process communication over UNIX Domain Sockets, TCP/IP, or UDP; supports the Boost::Serialization library to easily marshal data structures across a communications link, then unmarshal on the receiving side. A key feature is anonymous publish/subscribe, which disconnects a consumer of data from having to know who is actually providing that data, enabling the easy interchange of components during testing and development.

Interfaces Library – Each interface between two processes in the system is added to this library. Each interface fits into a plug-in framework so that a task, depending on its communications configuration, can dynamically load the interfaces required at run-time. For example, this enables a perception task to abstract the notion of a LIDAR source and at run-time be configured to use any LIDAR source, effectively decoupling the algorithmic logic from the nuts-and-bolts of sensor interfacing. Furthermore, interfaces can be built on top of other interfaces in order to produce composite information from multiple sources of data. For example, a pointed LIDAR interface combines a LIDAR interface, and a pose interface.

Configuration Library – Parses configuration files written in the Ruby scripting language in order to configure various aspects of the system at run-time. Each individual task can add parameters specific to its operation, in addition to common parameters like those that affect the loggers' verbosity, and the configuration of communications interfaces. The Ruby scripting language is used in order to provide a more familiar syntax, ease of detecting errors in syntax or malformed scripts, and to give the user several options for calculating and deriving configuration parameters.

Task Library – Abstracts around the system's *main()* function, provides an event loop that is triggered at specified frequencies, and automatically establishes communication with other tasks in the system.

Debug Logger – Provides a mechanism for applications to send debug messages of varying priority to the console, operator control station, log file, etc., depending on a threshold that varies verbosity according to a priority threshold parameter.

Log/Playback – The data log utility provides a generic way to log any *interface* in the system. Every data structure that is transmitted through the inter-process communication system can inherently be captured using this utility. The logged data are saved to a Berkeley database file along with a timestamp. The playback utility can read a Berkeley database file, seek to a particular time within the file, and transmit the messages stored in the file across the inter-process communication system. Since the inter-process communication system uses an anonymous publish/subscribe scheme, the consuming processes will receive the played-back messages, without realizing that data are not coming from a live sensor. This feature is useful in order to replay incidents that occurred on the vehicle for off-line analysis.

Fig. 23. The Tartan Racing Operator Control Station (TROCS) is an extensible GUI that enables developers to both monitor telemetry from Boss while it is driving and replay data offline for algorithm analysis.

Tartan Racing Operator Control Station (TROCS) – A graphical interface based on QT that provides an operator, engineer, or tester a convenient tool for starting and stopping the software, viewing status/health information, and debugging the various tasks that are executing. Each developer can develop custom widgets that plug into TROCS in order to display information for debugging and/or monitoring purposes.

8 Testing

Testing was a central theme of the research program that developed Boss. Over the 16 months of development, Boss performed more than 3000km of autonomous

driving. The team used time on two test vehicles, a simulation and data replay tool, and multiple test sites in three states to debug, test and evaluate the system.

Testing and development were closely intertwined, following the cyclic process illustrated in Figure 24. Before algorithm development began, the team assembled requirements that defined the capabilities Boss would need to be able to complete the challenge. Requirements drove the development of algorithms and selection of components. Algorithms and approaches were tested offline either in simulation or by using data replay. Once an algorithm became sufficiently mature, it would move to on-vehicle testing, where system and environmental interactions could be fully evaluated. These tests would often uncover algorithmic problems or implementation bugs that were not obvious during offline testing, often resulting in numerous cycles of rework and further offline debugging. Once the developers were satisfied that the algorithm worked, testing of the algorithm was added to the formal, regularly scheduled system test. Independent testing of algorithms would often uncover new deficiencies, requiring some rework. In other cases, testing and post-analysis would cause the team to modify the requirements driving the development, limiting or extending scope as appropriate. Eventually, the technology would be deemed accepted, and ready for the Urban Challenge.

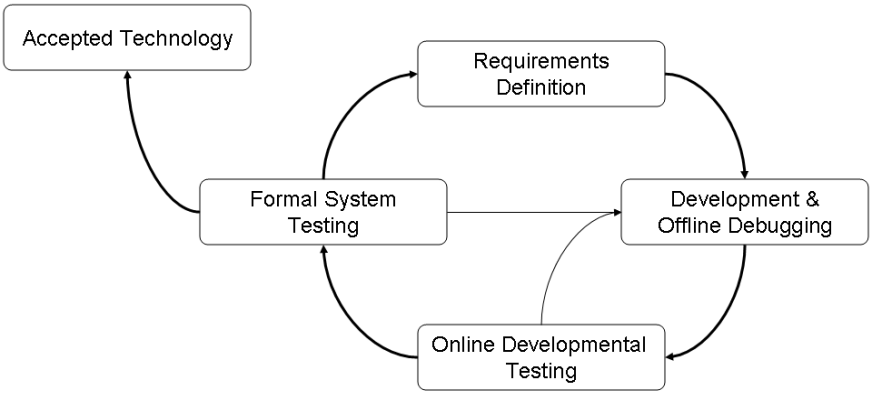


Fig. 24. The requirements and testing process used in the development of Boss.

8.1 System Testing

Regressive system testing was the cornerstone of the development process. Following the cyclic development process, each researcher was free to implement and test as independently of the overall system as possible, but to judge overall capability, and to verify that component changes did not degrade overall system performance, the team performed regressive system testing.

System testing was performed with a frequency proportionate to system readiness. Between February and October 2007, the team performed 65 days of system testing. Formal testing time was adjusted over the course of the program

to ensure relevance. As an example, during February the team performed less than 16 kilometers of system testing. In contrast, during the first three weeks of October, the team performed over 1500 kilometers of autonomous testing.

In general, the team tested once a week, but leading up to major milestones (the midterm site visit and national qualification event) the team moved to daily regressive testing. During regressive testing, the team would evaluate Boss's performance against a standard set of plays (or scenarios) described in a master playbook. The playbook captures over 250 different driving events that are important to evaluate. Figure 25 illustrates what a page from the playbook looks like. Each play is annotated with priority (ranked 1 to 3), how thoroughly it has been tested, how it relates to requirements, and a description of how Boss should behave when encountering this scenario. The 250 plays cover the mundane (correctly stopping at a stop sign) to the challenging (successfully navigating a jammed intersection), enabling the team to have confidence that even the most coupled software changes were not damaging overall system performance.

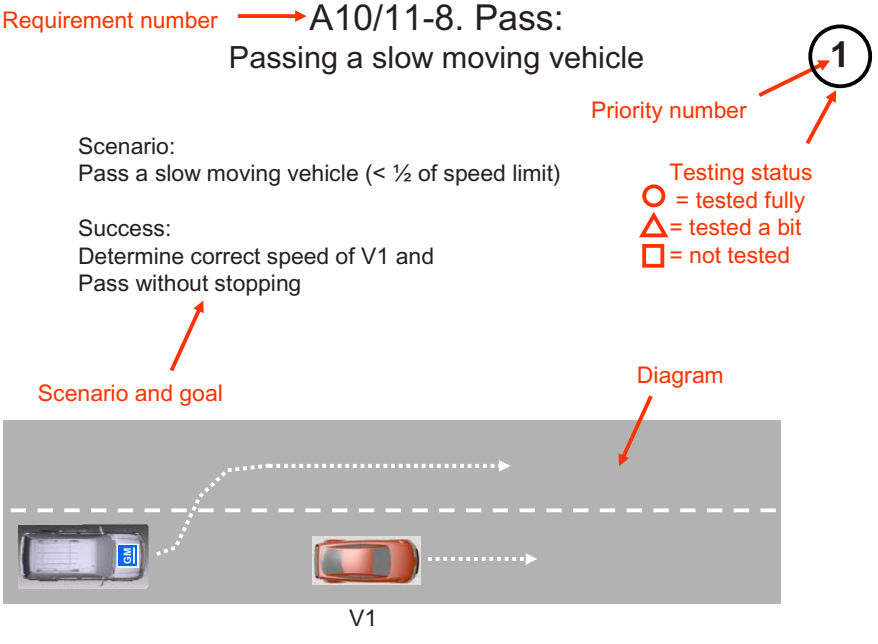


Fig. 25. A representative page from the testing playbook.

Feedback from system tests was rapidly passed to the development team through a *Hot Wash* after each system test, and a test report was published within 48 hours. Whereas the hot wash was delivered by the test software operator, who conveyed first-hand experience of code performance, the test report provided a more black-box understanding of how well the system met its mission requirements. Software bugs discovered through this and other testing were

electronically tracked, and formal review occurred weekly. The test report included a *Gap Analysis* showing the requirements that remained to be verified under system testing. This gap analysis was an essential measure of the team's readiness to compete at the Urban Challenge.

In addition to regressive testing, the team performed periodic endurance tests designed to confirm that Boss could safely operate for at least 6 hours or 60 miles (the stated length of the Urban Challenge). This was the acid test of performance, allowing the team to catch intermittent and subtle software and mechanical defects by increasing time on the vehicle. One of the most elusive problems discovered by this testing process was an electrical shorting problem that was the result of a 2mm gash in a signal line on the base vehicle Tahoe bus. The problem caused Boss to lose all automotive electrical power, killing the vehicle. Had the team not performed endurance testing it is plausible this defect would have never been encountered before the UCFE, and would have caused Boss to fail.

9 Performance at the National Qualification Event and Urban Challenge Final Event

The National Qualification and Urban Challenge Final events were held at the former George Air Force Base (see Figure 26), which provided a variety of roads, intersections and parking lots to test the vehicles. The NQE allowed DARPA to assess the capability and safety of each of the competitors. The teams were evaluated on three courses. Area A required the autonomous vehicles to merge into and turn across dense moving traffic. Vehicles had to judge the size of gaps between moving vehicles, assess safety and then maneuver without excessive delay. For many of the vehicles, this was the most difficult challenge, as it involved significant reasoning about moving obstacles. Area B was a relatively long road course that wound through a neighborhood. Vehicles were challenged to find their way through the road network while avoiding parked cars, construction areas and other road obstacles, but did not encounter moving traffic. Area C was a relatively short course, but required autonomous vehicles to demonstrate correct behavior with traffic at four-way intersections and to demonstrate rerouting around an unexpectedly blocked road.

For the final event, the field was reduced to eleven teams (see Figure 27). Traffic on the course was provided by not only the eleven qualifying vehicles, but fifty human-driven vehicles operated by DARPA. While primarily on-road, the course also included a pair of relatively short dirt roads, one of which ramped its way down a 50m elevation change.

Despite the testing by each of the teams and a rigorous qualification process, the challenge proved to be just that. Of the eleven teams that entered, six were able to complete the 85 kilometer course, three of them without human intervention. Boss finished the challenge approximately nineteen minutes faster than the second-place vehicle, Junior (from Stanford University) and twenty-six minutes ahead of the third-place vehicle, Odin (from Virginia Tech). The vehicles from Cornell, MIT and the University of Pennsylvania rounded out the finishers.

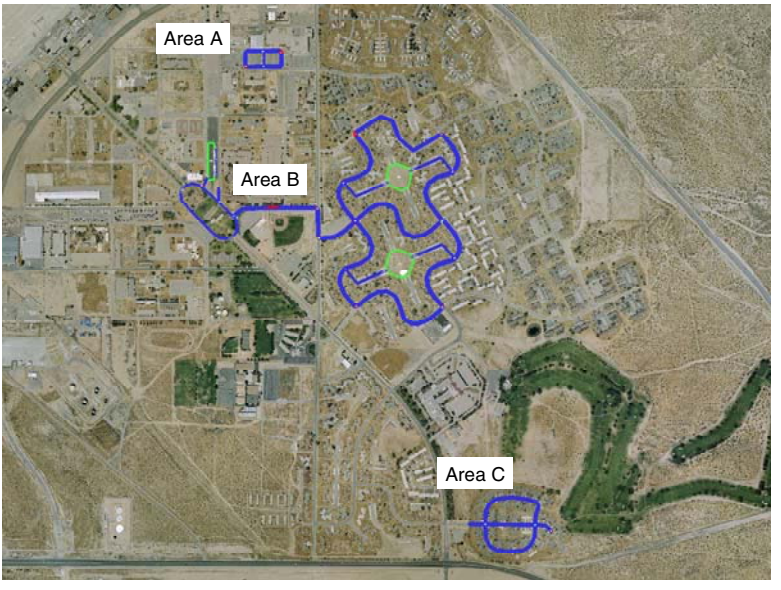


Fig. 26. The National Qualification Event took place in three areas, each emphasizing different skills. Area A tested merging with moving traffic, Area B tested navigation, while Area C tested re-routing and intersection skills.

Fig. 27. Of the eleven vehicles that qualified for the Urban Challenge Final event, three completed the challenge without human intervention. Three additional teams finished the course with minor interventions.

Overall, the vehicles that competed in the challenge performed admirably, with only one vehicle-to-vehicle collision which occurred at very low speeds and resulted in no damage. While the vehicles drove well, none of them was perfect. Amongst the foibles: Boss twice incorrectly determined that it needed to make a U-turn, resulting in its driving an unnecessary two miles; Junior had a minor bug that caused it to repeatedly loop twice through one section of the course; and Odin incurred a significant GPS error that caused it to drive partially off the road for part of the challenge. Despite these glitches, these vehicles represent a new state-of-the-art for urban driving.

The following sections describe a few incidents during the qualifications and final event where Boss encountered some difficulties.

9.1 National Qualification Event Analysis

Boss performed well at each component of the National Qualification event, consistently demonstrating good driving skills and overall system robustness. Through the NQE testing, Boss demonstrated three significant bugs, none of which were mission-ending.

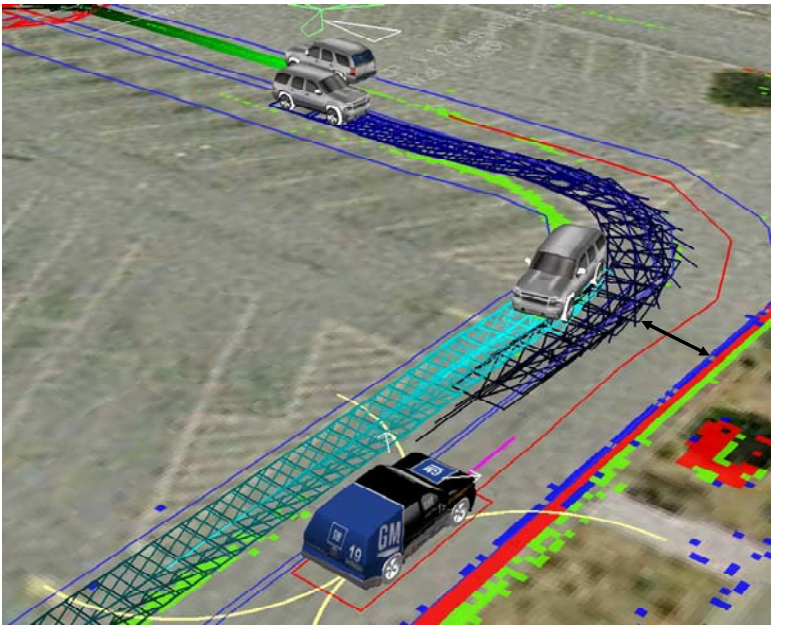


Fig. 28. Data replay shows how the incorrectly extrapolated path of a vehicle (dark blue) and the wall (red pixels) create space that Boss believes is too narrow to drive through (indicated by the arrow).

The first significant incident occurred in area A, the course that tested a robot's ability to merge with traffic. During Boss's first run on this course, it approached a corner and stopped for about 20 seconds before continuing, crossing the center line for some time before settling back into the correct lane and continuing. This incident had two principal causes: narrow lanes and incorrect lane geometry. In preparing the course, DARPA arranged large concrete "Jersey" barriers immediately adjacent to the lane edges to prevent vehicles from leaving the course and injuring spectators. This left little room for navigational error. When configuring Boss for the run, the team adjusted the geometry defined in the RNDF, with the intention of representing the road shape as accurately as possible

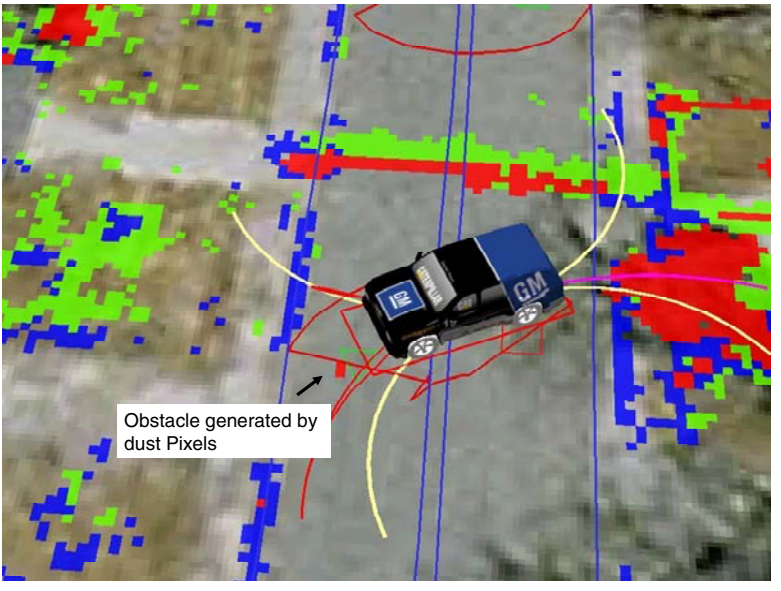


Fig. 29. The false obstacles generated by dust and the bush behind Boss prevented Boss from initially completing its U-turn.

given the available overhead imagery. During this process, the team incorrectly defined the shape of the inner lanes. While this shape was unimportant for Boss's localization and navigation, it was used to predict the motion of the other vehicles on the course. The incorrect geometry caused Boss to predict that the other vehicles were coming into its lane (see Figure 28). It thus stopped and an error recovery mode (Shimmy) kicked in. After shimmying down the lane, Boss reverted to normal driving and completed the test.

The second significant incident occurred during testing in Area C. This course tested the autonomous vehicle's ability to handle traffic at intersections and replan for blocked roads. For most of this course, Boss drove well and cleanly, correctly making its way through intersections and replanning when it encountered a blocked road. The second time Boss encountered a blocked road it began to U-turn, getting two of its wheels up on a curb. As it backed back down off the curb, it caused a cloud of dust to rise and then stopped and appeared stuck, turning its wheels backwards and forwards for about a minute. After the minute, Boss started moving again and completed the course without further incident.

Post-test data analysis revealed that Boss perceived the dust cloud as an obstacle. In addition, an overhanging tree branch behind Boss caused it to believe there was insufficient room to back up. Normally, when the dust settled, Boss would have perceived that there was no longer an obstacle in front of it and continued driving, but in this case the dust rose very close to Boss, on the boundary of its blind spot (see Figure 29). The obstacle detection algorithms treat this area specially and do not clear obstacles within this zone. Eventually Boss wiggled enough to verify that the cell where it had previously seen the dust was no

longer occupied. Once again, Boss's consistent attempts to replan paid off, this time indirectly. After this test, the obstacle detection algorithms were modified to give no special treatment to obstacles within the blind spot.

The third significant incident during qualifications occurred in area B, the course designed to test navigation and driving skills. During the test, Boss came up behind a pair of cars parked along the edge of the road. The cars were not sufficiently in the road to be considered stopped vehicles by the perception system, so it fell to the motion planning system to avoid them. Due to pessimistic parameter settings, the motion planner believed there was insufficient room to navigate around the obstacles without leaving its lane, invoking the behavioral error recovery system. Due to a bug in the error handling system, the goal requested by the behavioral engine was located approximately 30m behind Boss's current position, causing it to back up. During the backup maneuver DARPA paused Boss. This cleared the error recovery stack, and upon restart, Boss continued along the lane normally until it encountered the same vehicles, at which point it invoked the recovery system again, but due to the pause clearing state in the behavioral system, the new recovery goal was in a correct, down-road location.

Despite these foibles, after completion of the qualification events Boss was ranked as the top performer and was given pole position for the final event.

9.2 Final Event Analysis

Immediately before Boss was to begin the final event, the team noticed that Boss's GPS receivers were not receiving GPS signals. Through equipment tests and observation of the surroundings, it was determined that the most likely cause of this problem was jamming from a Jumbotron (a large television commonly used at major sporting events) newly positioned near the start area. After a quick conference with the DARPA officials, the Jumbotron was shut down and Boss's GPS receivers were restarted and ready to go. Boss smoothly departed the launch area to commence its first of three missions for the day.

Once launched, Boss contended with ten other autonomous vehicles and approximately fifty other human driven vehicles. During the event Boss performed well, but did have a few occurrences of unusual behavior.

The first incident occurred on a relatively bumpy transition from a dirt road to a paved road. Instead of stopping and then continuing normally, Boss stopped for a prolonged period of time and then hesitantly turned onto the paved road. Boss's hesitation was due to a calibration error in the Velodyne LIDAR, used for static obstacle detection. In this case, the ground slope combined with this miscalibration was sufficient for the laser to momentarily detect the ground, calling it an obstacle (see Figure 30). Boss then entered an error recovery mode, invoking the zone planner to get to the road. With each movement, the false obstacles in front of Boss would change position, causing replanning, and thus hesitant behavior. Once Boss made it off the curb and onto the road, the false obstacles cleared and Boss was able to continue driving normally.

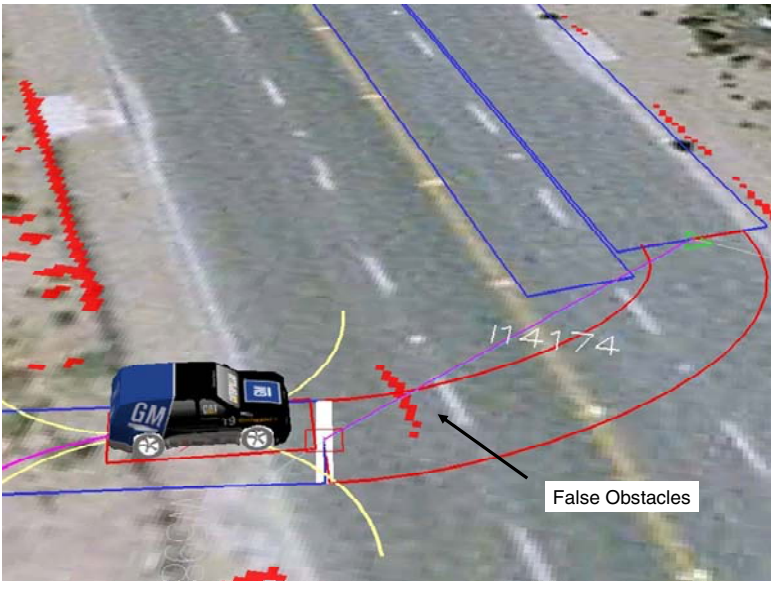


Fig. 30. False obstacles that caused Boss to stutter when leaving a dirt road.

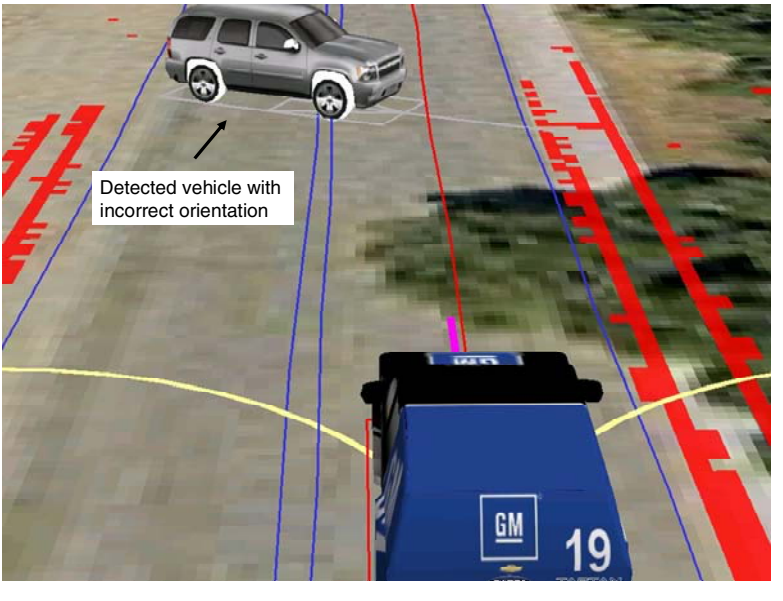


Fig. 31. This incorrect estimate of an oncoming vehicles orientation caused Boss to swerve and almost caused Boss to become irrevocably stuck.

The next incident occurred when Boss swerved abruptly while driving past an oncoming robot. The oncoming vehicle was partially in Boss's lane, but not sufficiently that Boss needed to maneuver to avoid it. The oncoming vehicle partially occluded its chase vehicle and momentarily caused Boss's perception system to estimate the vehicle with an incorrect orientation such that it was perceived to be entering Boss's travel lane (see Figure 31). At this point, Boss swerved and braked to try and avoid the perceived oncoming vehicle, moving it very close to the Jersey barrier wall. While this was happening, DARPA ordered a full course pause due to activity elsewhere on the course. This interrupted Boss mid-motion, causing it to be unable to finish its maneuver. Upon awakening from the pause, Boss felt it was too close to the wall to maneuver safely. After a minute or so of near-stationary wheel-turning, its pose estimate shifted laterally by about 2cm, just enough that it believed it had enough room to avoid the near wall. With a safe path ahead, Boss resumed driving and continued along its route.

Later in the first mission Boss was forced to queue behind a vehicle already waiting at a stop line. Boss began queuing properly, but when the lead vehicle pulled forward, Boss did not. After an excessive delay, Boss performed a U-turn and drove away from the intersection, taking an alternative route to its next checkpoint.

Analysis revealed a minor bug in the planning system, which did not correctly update the location of moving vehicles. For efficiency, the zone planner checks goal locations against the location of obstacles before attempting to generate a path. This check can be performed efficiently, allowing the planning system to report that a goal is unreachable in much less time than it takes to perform an exhaustive search. A defect in this implementation caused the planning system to not correctly update the list of moving obstacles prior to performing this check. Thus, after the planner's first attempt to plan a path forward failed, every future attempt to plan to the same location failed, since the planner erroneously believed that a stopped vehicle was in front of it. The behavioral error recovery system eventually selected a U-turn goal, and Boss backed up and went on its way. This re-route caused Boss to drive an additional 2.7 km, but it was still able to complete the mission.

Despite these incidents, Boss was still able to finish the challenge in four hours, ten minutes and twenty seconds, roughly nineteen minutes faster than the second-place competitor. Boss averaged 22.5kph during the challenge (while enabled), and 24.5kph when moving. Through offline simulation we estimated that the maximum average speed Boss could have obtained over the course was 26.2kph. Figure 32 shows the distribution of Boss's moving speeds during the challenge. The large peak at 9-10mps is due to the course speed limits. This spike implies that Boss was limited by these speed limits, not its capability.

The roadmap localization system played an important role during the challenge. For a majority of the challenge, the error estimate in the roadmap localization system was less than 0.5m, but there were over 16 minutes of time that the error was greater than 0.5m, with a peak error of 2.5m. Had the road map localization system not been active, it is likely that Boss would have been either off the road or in a wrong lane for a significant amount of time.

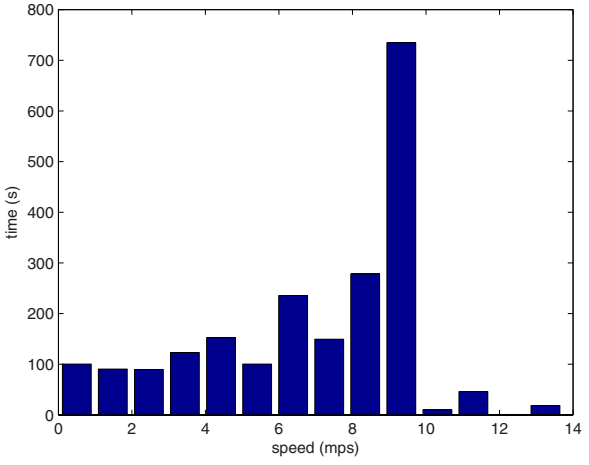


Fig. 32. Boss averaged 22.5kph during the challenge, this figure shows a distribution of the vehicle's speed while moving.

In general, Boss drove well, completing a large majority of the course with skill and precision. As demonstrated in this brief analysis, one of Boss's strengths was its robustness and ability to recover from unexpected error cases autonomously.

10 Lessons Learned

Through the development of Boss and competition in the Urban Challenge, the team learned several valuable lessons:

Available off-the-shelf sensors are insufficient for urban driving- There is currently no single sensor capable of providing environmental data to sufficient range, and with sufficient coverage to support autonomous urban driving. The Velodyne sensor used on Boss (and other Urban Challenge vehicles) comes close, but has insufficient angular resolution at long ranges and is unwieldy for commercial automotive applications.

Road shape estimation may be replaced by estimating position relative to the road- In urban environments, the shape of roads changes infrequently. There may be local anomalies (e.g. a stopped car or construction), but in general, a prior model of road shape can be used for on-road navigation. Several Urban Challenge teams took this approach, including our team, and demonstrated that it was feasible on a small to medium scale. While this may not be a viable approach for all roads, it has proven a viable method for reducing complexity in common urban driving scenarios. The next step will be to apply the same approach on a large or national scale and automate the detection of when the road has changed shape from the expected.

Human-level urban driving will require a rich representation- The representation used by Boss consists of lanes and their interconnections, a regular map containing large obstacles and curbs, a regular map containing occlusions, and a list of rectangles (vehicles) and their predicted motions. Boss has a very primitive notion of what is and is not a vehicle: if it is observed to move within some small time window and it is in a lane or parking lot, then it is a vehicle, otherwise it is not. Time and location are thus the only elements that Boss uses to classify an object as a vehicle. This can cause unwanted behavior; for example, Boss will wait equally long behind a stopped car (appearing reasonable) and a barrel (appearing unreasonable), while trying to differentiate between them. A richer representation including more semantic information will enable future autonomous vehicles to behave more intelligently.

Validation and verification of urban driving systems is an unsolved problem- The authors are unaware of any formal methods that would allow definitive statements about the completeness or correctness of a vehicle interacting with a static environment, much less a dynamic one. While sub-systems that do not interact directly with the outside world can be proven correct and complete (e.g. the planning algorithm), verifying a system that interacts with the world (e.g. sensors/ world model building) is as of yet impossible.

Our approach of generating an ad hoc, but large, set of test scenarios performed relatively well for the Urban Challenge, but as the level of reliability and robustness approaches that needed for autonomous vehicles to reach the market place, this testing process will likely be insufficient. The real limitation of these tests, is that it is too easy to "teach to the test" and develop systems that are able to reliably complete these tests but are not robust to a varied world. To reduce this problem, we incorporated *free-for-all* testing in our test process, which allowed traffic to engage Boss in a variety of normal, but unscripted, ways. While this can increase robustness, it can in no way guarantee that the system is correct.

Sliding Autonomy will reduce complexity of autonomous vehicles- In building a system that was able to recover from a variety of failure cases, we introduced significant system complexity. In general, Boss was able to recover from many failure modes, but took considerable time to do so. If, instead of attempting an autonomous recovery, the vehicle were to request assistance from a human controller, much of the system complexity would be reduced and the time taken to recover from faults would decrease dramatically. The critical balance here is to ensure the vehicle is sufficiently capable that it does not request help so frequently that the benefits of autonomy are lost. As an example, if Boss was allowed to ask for help during the four-hour Urban Challenge, there were three occasions where it might have requested assistance. Human intervention at these times would likely have reduced Boss's overall mission time by approximately fifteen minutes.

Driving is a social activity- Human driving is a social activity consisting of many subtle and some not-so-subtle cues. Drivers will indicate their willingness for other vehicles to change lanes by varying their speed, and the gap between themselves and another vehicle, by small amounts. At other times it is necessary to interpret hand gestures and eye contact in situations when the normal rules of

the road are violated, or need to be violated for traffic to flow smoothly and efficiently. For autonomous vehicles to seamlessly integrate into our society, they would need to be able to interpret these gestures.

Despite this, it may be possible to deploy autonomous vehicles which are unaware of the subtler social cues. During our testing and from anecdotal reports during the final event, it became clear that human drivers were able to quickly adapt and infer (perhaps incorrectly) the reasoning within the autonomy system. Perhaps it will be sufficient and easier to assume that we humans will adapt to robotic conventions of driving rather than the other way around.

11 Conclusions

The Urban Challenge was a tremendously exciting program to take part in. The aggressive technology development timeline, international competition, and compelling motivations fostered an environment that brought out a tremendous level of creativity and effort from all those involved. This research effort generated many innovations:

- a coupled moving obstacle and static obstacle detection and tracking system;
- a road navigation system that combines road localization and road shape estimation to drive both on roads where there is and is not a priori road geometry is available;
- a mixed-mode planning system that is able to both efficiently navigate on roads and safely maneuver through open areas and parking lots;
- a behavioral engine that is capable of both following the rules of the road and violating them when necessary;
- a development and testing methodology that enables rapid development and testing of highly capable autonomous vehicles.

While this article outlines the algorithms and technology that made Boss capable of meeting the challenge, there is much left to do. Urban environments are considerably more complicated than what the vehicles faced in the Urban Challenge; pedestrians, traffic lights, varied weather, and dense traffic all contribute to this complexity.

As the field advances to address these problems, we will be faced with secondary problems, such as how do we test these systems and how will society accept them? While defense needs may provide the momentum necessary to drive these promising technologies, we must work hard to ensure our work is relevant and beneficial to a broader society. While these challenges loom large, it is clear that there is a bright and non-too-distant future for autonomous vehicles.

Acknowledgments

This work would not have been possible without the dedicated efforts of the Tartan Racing team and the generous support of our sponsors, including General

Motors, Caterpillar, and Continental. This work was further supported by DARPA under contract HR0011-06-C-0142.

References

Brooks, R.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation 2(1), 14–23 (1986)

Committee on Army Unmanned Ground Vehicle Technology and the National Research Council. Technology Development for Army Unmanned Ground Vehicles. Washington, D.C (2002)

Darms, M., Winner, H.: A modular system architecture for sensor data processing of ADAS applications. In: Proceedings of IEEE Intelligent Vehicles Symposium, Las Vegas, USA, pp. 729–734 (2005)

Darms, M.: Eine Basis-Systemarchitektur zur Sensordatenfusion von Umfeldsensoren Fahrerassistenzsysteme, Fortschrittb. VDI: R12, Nr. 653 (2007)

Darms, M., Rybski, P., Urmson, C.: An Adaptive Model Switching Approach for a Multisensor Tracking System used for Autonomous Driving in an Urban Environment. Steuerung und Regelung von Fahrzeugen und Motoren – AUTOREG 2008, Baden Baden (February 2008)

Darms, M., Baker, C., Rybksi, P., Urmson, C.: Vehicle Detection and Tracking for the Urban Challenge- The Approach taken by Tartan Racing. In: Maurer, M., Stiller, C. (eds.) 5. Workshop Fahrerassistenzsysteme FAS 2008, April 2008, FMRT, Karlsruhe (2008)

Darms, M., Rybski, P., Urmson, C.: Classification and Tracking of Dynamic Objects with Multiple Sensors for Autonomous Driving in Urban Environments. In: Proceedings of the IEEE Intelligent Vehicles Symposium. Eindhoven, NL (June 2008)

DARPA Urban Challenge (2007), `http://www.darpa.mil/grandchallenge/index.asp`

Daubechies, I.: Ten lectures on wavelets. Society for Industrial and Applied. Mathematics (1992)

Duda, R.O., Hart, P.E.: Use of the Hough Transformation to Detect Lines and Curves in Pictures. Comm. ACM 15, 11–15 (1972)

Ferguson, D., Howard, T., Likhachev, M.: Motion Planning in Urban Environments (2008) (in preparation)

Howard, T.M., Kelly, A.: Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots. International Journal of Robotics Research 26(2), 141–166 (2007)

Huttenlocker, D., Felzenswalb, P.: Distance Transforms of Sampled Functions, Cornell Computing and Information Science Technical Report TR2004-1963 (2004)

Kaempchen, N., Weiss, K., Schaefer, M., Dietmayer, K.C.J., et al.: IMM object tracking for high dynamic driving maneuvers. In: IEEE Intelligent Vehicles Symposium 2004, Parma, Italy, pp. 825–830 (June 2004)

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., Thrun, S.: Anytime Dynamic A*: An Anytime, Replanning Algorithm. In: International Conference on Automated Planning and Scheduling, ICAPS (2005)

MacLachlan, R.: Tracking Moving Objects From a Moving Vehicle Using a Laser Scanner. tech. report CMU-RI-TR-05-07, Carnegie Mellon University (June 2005)

Shih, M.-Y., Tseng, D.-C.: A wavelet-based multi-resolution edge detection and tracking. Image and Vision Computing 23(4), 441–451 (2005)

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley, the robot that won the DARPA Grand Challenge. Journal of Field Robotics 23(9), 661–692 (2006)

Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J.P., Gowdy, J., Gutierrez, A., Harbaugh, S., Johnson-Roberson, M., Kato, H., Koon, P.L., Peterson, K., Smith, B.K., Spiker, S., Tryzelaar, E., Whittaker, W.L.: High Speed Navigation of Unrehearsed Terrain: Red Team Technology for Grand Challenge, Tech. report CMU-RI-TR-04-37, Robotics Institute, Carnegie Mellon University (2004)

Urmson, C., Anhalt, J., Bartz, D., Clark, M., Galatali, T., Gutierrez, A., Harbaugh, S., Johnston, J., Kato, H., Koon, P.L., Messner, W., Miller, N., Mosher, A., Peterson, K., Ragusa, C., Ray, D., Smith, B.K., Snider, J.M., Spiker, S., Struble, J.C., Ziglar, J., Whittaker, W.L.: A Robust Approach to High-Speed Navigation for Unrehearsed Desert Terrain. Journal of Field Robotics 23(8), 467–508 (2006)

Viola, P., Jones, M.: Robust real-time object detection. In: The 2nd International Workshop on Statistical and Computational Theories of Vision-Modeling, Learning, Computing, and Sampling (2001)

Motion Planning in Urban Environments

Dave Ferguson[1], Thomas M. Howard[2], and Maxim Likhachev[3]

[1] Intel Research Pittsburgh
Pittsburgh, PA 15213, USA
dave.ferguson@intel.com
[2] Carnegie Mellon University
Pittsburgh, PA 15213, USA
thoward@ri.cmu.edu
[3] University of Pennsylvania
Philadelphia, PA, USA
maximl@seas.upenn.edu

Abstract. We present the motion planning framework for an autonomous vehicle navigating through urban environments. Such environments present a number of motion planning challenges, including ultra-reliability, high-speed operation, complex inter-vehicle interaction, parking in large unstructured lots, and constrained maneuvers. Our approach combines a model-predictive trajectory generation algorithm for computing dynamically-feasible actions with two higher-level planners for generating long range plans in both on-road and unstructured areas of the environment. In the first part of this article, we describe the underlying trajectory generator and the on-road planning component of this system. We then describe the unstructured planning component of this system used for navigating through parking lots and recovering from anomalous on-road scenarios. Throughout, we provide examples and results from "Boss", an autonomous SUV that has driven itself over 3000 kilometers and competed in, and won, the DARPA Urban Challenge.

1 Introduction

Autonomous passenger vehicles present an incredible opportunity for the field of robotics and society at large. Such technology could drastically improve safety on roads, provide independence to millions of people unable to drive because of age or ability, revolutionize the transportation industry, and reduce the danger associated with military convoy operations. However, developing robotic systems that are sophisticated enough and reliable enough to operate in everyday driving scenarios is tough. As a result, up until very recently, autonomous vehicle technology has been limited to either off-road, unstructured environments where complex interaction with other vehicles is non-existent (Stentz and Hebert, 1995; Kelly, 1995; Singh et al., 2000; JFR, 2006a; JFR, 2006b; Carsten et al., 2007), or very simple on-road maneuvers such as highway-based lane following (Thorpe et al., 1997).

The DARPA Urban Challenge competition was designed to extend this technology as far as possible towards the goal of unrestricted on-road driving. The event consisted of an autonomous vehicle race through an urban environment containing single and multi-lane roads, traffic circles and intersections, open areas and unpaved sections, road blockages, and complex parking tasks. Successful vehicles had to

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 61–89.
springerlink.com

travel roughly 60 miles, all in the presence of other human-driven and autonomous vehicles, and all while abiding by speed limits and California driving rules.

This challenge required significant advances over the state of the art in autonomous vehicle technology. In this paper, we describe the motion planning system developed for Carnegie Mellon University's winning entry into the Urban Challenge, "Boss". This system enabled Boss to travel extremely quickly through the urban environment to complete its missions; interact safely and intelligently with obstacles and other vehicles on roads, at intersections, and in parking lots; and perform sophisticated maneuvers to solve complex parking tasks.

We first introduce very briefly the software architecture used by Boss and the role of motion planning within that architecture. We then describe the trajectory generation algorithm used to generate every move of the vehicle. In Section 5, we discuss the motion planning framework used when navigating on roads.

In Section 6, we discuss the framework used when navigating through unstructured areas or performing complex maneuvers. We then provide results and discussion from hundreds of hours and thousands of miles of testing, and describe related work in both on-road and unstructured planning.

2 System Architecture

Boss' software system is decomposed into four major blocks (see Figure 1) and is described in detail in (Urmson et al., 2008). The **Perception** component fuses and processes data from Boss' sensors to provide key environmental information, including:

- **Vehicle State**, globally-referenced position, attitude and speed for Boss;
- **Road World Model**, globally-referenced geometric information about the roads, parking zones, and intersections in the world;
- **Moving Obstacle Set**, an estimation of other vehicles in the vicinity of Boss;
- **Static Obstacle Map**, a 2D grid representation of free, dangerous, and lethal space in the world; and
- **Road Blockages**, an estimation of clearly impassable road sections.

The **Mission Planning** component computes the fastest route through the road network to reach the next checkpoint in the mission, based on knowledge of road blockages, speed limits, and the nominal time required to make special maneuvers such as lane changes or u-turns.

The **Behavioral Executive** combines the strategic global information provided by Mission Planning with local traffic and obstacle information provided by Perception and generates a sequence of local tasks for the Motion Planner. It is responsible for the system's adherence to various rules of the road, especially those concerning structured interactions with other traffic and road blockages, and for detection of and recovery from anomalous situations. The local tasks it feeds to the Motion Planner take the form of discrete motion goals, such as driving along a road lane to a specific point or maneuvering to a specific pose or parking spot. The issuance of these goals is predicated on traffic concerns such as precedence among vehicles stopped at an

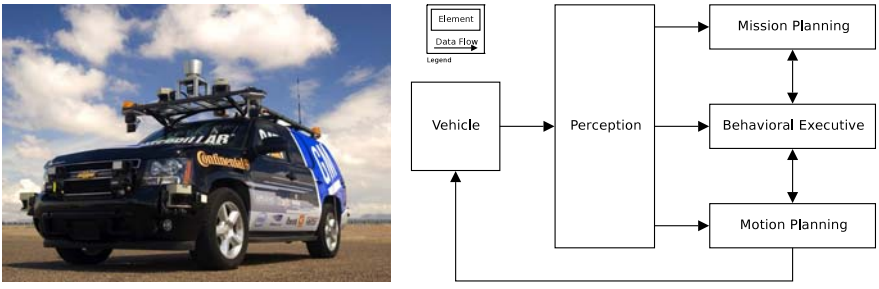


Fig. 1. "Boss": Carnegie Mellon's winning entry in the Urban Challenge, along with its software system architecture.

intersection. In the case of driving along a road, desired lane and speed commands are given to the Motion Planner to implement behaviors such as distance keeping, passing maneuvers, and queueing in stop-and-go traffic.

The **Motion Planning** component takes the motion goal from the Behavioral Executive and generates and executes a trajectory that will safely drive Boss towards this goal, as described in the following section. Two broad contexts for motion planning exist: on-road driving and unstructured driving.

3 Motion Planning

The motion planning layer is responsible for executing the current motion goal issued from the Behavioral Executive. This goal may be a location within a road lane when performing nominal on-road driving, a location within a parking lot or obstacle field when traversing through one of these areas, or any location in the environment when performing error recovery. The motion planner constrains itself based on the context of the goal and the environment to abide by the rules of the road.

Figure 2 provides a basic illustration of the nature of the goals provided by the Behavioral Executive. During nominal on-road driving, the goal entails a desired lane and a desired position within that lane (typically a stop-line at the end of the lane). In such cases, the motion planner invokes a high-speed on-road planner to generate a path that tracks the desired lane. During unstructured driving, such as when navigating through parking lots, the goal consists of a desired pose of the vehicle in the world. In these cases, the motion planner invokes a 4D lattice planner that generates a global path to the desired pose. These unstructured motion goals are also used when the vehicle encounters an anomalous situation during on-road driving and needs to perform a complex maneuver (such as when an intersection is partially blocked and cannot be traversed in the desired lane).

As well as issuing motion goals, the Behavioral Executive is constantly providing desired maximum speed and acceleration/deceleration commands to the motion planner. It is through this interface that the Behavioral Executive is able to control the vehicle's forward progress in distance keeping and intersection precedence scenarios. When the vehicle is not constrained by such scenarios, the motion planner

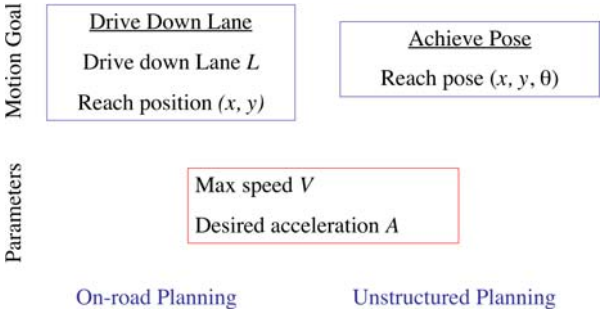


Fig. 2. Motion goals provided by the Behavioral Executive to the Motion Planner. Also shown are the frequently updated speed and desired acceleration commands.

computes desired speeds and accelerations based on the constraints of the environment itself (e.g. road curvature and speed limits).

Given a motion goal, the motion planner creates a path towards the desired goal then tracks this path by generating a set of candidate trajectories that follow the path to varying degrees and selecting from this set the best trajectory according to an evaluation function. Each of these candidate trajectories is computed using a trajectory generation algorithm described in the next section. As mentioned above, the nature of the path being followed generated differs based on the context of the motion goal and the environment. In addition, the evaluation function differs depending on the context but always includes consideration of static and dynamic obstacles, curbs, speed, curvature, and deviation from the path. The selected trajectory is then directly executed by the vehicle. This process is repeated at 10 Hz by the motion planner.

4 Trajectory Generation

Each candidate trajectory is computed using a model-predictive trajectory generator from (Howard and Kelly, 2007) that produces dynamically feasible actions between initial and desired vehicle states. In general, this algorithm can be used to solve the problem of generating a set of parameterized controls ($\mathbf{u}(\mathbf{p}, \mathbf{x})$) that satisfy a set of state constraints whose dynamics can be expressed by a set of differential equations:

$$\mathbf{x} = \begin{bmatrix} x \; y \; \theta \; \kappa \; v \; \ldots \end{bmatrix}^T \tag{1}$$

$$\dot{\mathbf{x}}(\mathbf{x}, \mathbf{p}) = f(\mathbf{x}, \mathbf{u}(\mathbf{p}, \mathbf{x})), \tag{2}$$

where $\mathbf{x}$ is the vehicle state (with position (x, y), heading (θ), curvature (κ), and velocity (v), as well as other state parameters such as commanded velocity, etc) and $\mathbf{p}$ is the set of parameters for which we are solving. The derivative of vehicle state $\dot{\mathbf{x}}$ is a function of both the parameterized control input $\mathbf{u}(\mathbf{p}, \mathbf{x})$ and the vehicle state $\mathbf{x}$ because the vehicle's response to a particular control input is state dependent. In

this section, we describe the application of this general algorithm to our domain, specifically addressing the state constraints, vehicle model, control parameterization, initialization function, and trajectory optimization approaches used.

4.1 State Constraints

For navigating both on-road and unstructured areas of urban environments we generated trajectories that satisfied both target two-dimensional position (x, y) and heading (θ) constraints. We defined the constraint equation formula $(\mathbf{C}(\mathbf{x}, \mathbf{p}))$ as the difference between these target boundary state constraints (denoted $\mathbf{x_C}$) and the integral of the model dynamics (the endpoint of the computed vehicle trajectory):

$$\mathbf{x_C} = \begin{bmatrix} x_C & y_C & \theta_C \end{bmatrix}^T \tag{3}$$

$$\mathbf{x_F}(\mathbf{p}, \mathbf{x}) = \mathbf{x_I} + \int_0^{t_f} \dot{\mathbf{x}}(\mathbf{x}, \mathbf{p}) dt \tag{4}$$

$$\mathbf{C}(\mathbf{x}, \mathbf{p}) = \mathbf{x_C} - \mathbf{x_F}(\mathbf{p}, \mathbf{x}) \tag{5}$$

The constrained trajectory generation algorithm determines the control parameters $\mathbf{p}$ that drive Equation 5 to zero. This results in a trajectory from an initial state $\mathbf{x_I}$ to a terminal vehicle state $\mathbf{x_F}$ that is as close as possible to the desired terminal state $\mathbf{x_C}$.

4.2 Vehicle Modeling

The development of a high fidelity vehicle dynamics model is important for the accurate prediction of vehicle motion and thus for the generation of accurate trajectories using our constraint-based approach. Our vehicle model consists of a set of parameterized functions that were fit to data extracted from human-driven performance runs in the vehicle. The key parameters in our model are the controller delay, the curvature limit (the minimum turning radius), the curvature rate limit (a function of the maximum speed at which the steering wheel can be turned), and the maximum acceleration and deceleration of the vehicle. The controller delay accurately predicts the difference in time between a command from software and the corresponding initial response from hardware and is an important consideration when navigating at high speeds. The curvature, rate of curvature, acceleration and deceleration limits were essential for accurately predicting the response of the vehicle over entire trajectories. This model is then simulated using a fixed-timestep Euler integration to predict the vehicle's motion. Appendix A provides details of the model used for Boss.

4.3 Controls Parameterization

For Ackermann steered vehicles, it is advantageous to parameterize the curvature function in terms of arclength $(\kappa(\mathbf{p}, s))$. This is because, for similar trajectories, the

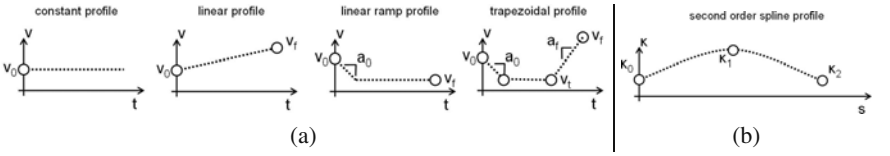


Fig. 3. Velocity and curvature profiles. (a) Several different linear velocity profiles were applied in this system, each with their own parameterization and application. Each parameterization contains some subset of velocity and acceleration knot points (v_0, v_t, v_f, a_0, and a_f) and the length of the path, measured in time (t_0, t_f). (b) The curvature profile includes four possible degrees of freedom: the three spline knot points (κ_0, κ_1, and κ_2) and the length of the path (s_f).

solution values for the curvature profile parameters are less dependent on the speed at which the trajectory is executed. For Boss, we parameterize the vehicle controls with a time-based linear velocity function ($v(\mathbf{p}, t)$) and an arclength-based curvature function ($\kappa(\mathbf{p}, s)$):

$$\mathbf{u}(\mathbf{p}, \mathbf{x}) = \left[v(\mathbf{p}, t) \ \kappa(\mathbf{p}, s) \right]^T \tag{6}$$

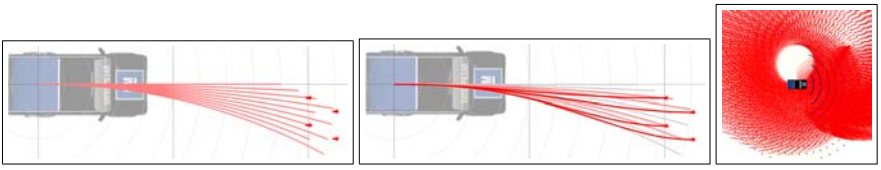
We allow the linear velocity profile to take the form of a constant profile, linear profile, linear ramp profile, or a trapezoidal profile (see Figure 3(a)). The motion planner selects the appropriate profile based on the driving mode and context (e.g. maintaining a constant velocity for distance keeping or slowing down for an upcoming intersection). Each of these profiles consists of a set of dependent parameters (v_0, v_t, v_f, a_0, and a_f) and the time to complete the profile (t_0, t_f), all of which become members of the parameter set $\mathbf{p}$. Since all of the dependent profile parameters are typically known, no optimization is done on the shape of each of these profiles.

The curvature profile defines the shape of the trajectory and is the primary profile over which optimization is performed. Our profile consists of three independent curvature knot point parameters (κ_0, κ_1, and κ_2) and the trajectory length (s_f) (see Figure 3(b)). In general, it is important to limit the degrees of freedom in the system to minimize the presence of local optima and to improve the runtime performance of the algorithm (which is approximately linear with the number of free parameters in the system) but maintain enough flexibility to satisfy all of the boundary state constraints. We chose a second order spline profile because it contains 4 degrees of freedom, enough to satisfy the 3 boundary state constraints. We further fix the initial command knot point κ_0 during the optimization process to the curvature at the initial state $\mathbf{x_I}$ to provide smooth controls[1].

With the linear velocity profile's dependent parameters being fully defined and the initial spline parameter of the curvature profile fixed, we are left with a system with three parameterized freedoms: the latter two curvature spline knot points and the trajectory length:

$$\mathbf{p_{free}} = \left[\kappa_1 \ \kappa_2 \ s_f \right]^T \tag{7}$$

[1] However, this can also be fixed to a different value to produce sharp trajectories, as described in Section 5.2.



(a) (b) (c)

Fig. 4. Offline lookup table generation. (a) Some sampled trajectories (in red) and some table endpoints (red arrows) that we wish to generate trajectories to for storage in the lookup table. (b) The closest sampled trajectories to the desired table endpoints are selected and then optimized to reach the desired endpoints. The parameter sets corresponding to these optimized trajectories are then stored in the lookup table. (c) The set of all sampled trajectories (red) and table endpoints (blue) for a single initial vehicle state. In this case the initial vehicle curvature is not zero so the set of trajectories is not symmetric about the vehicle.

The duality of the trajectory length (s_f) and time (t_f) can be resolved by estimating the time that it takes to drive the entire distance through the linear velocity profile. Arclength was used for the independent parameter for the curvature profiles because the shape of these profiles is somewhat independent of the speed at which they are traveled. This allows solutions with similar parameters for varying linear velocity profiles.

4.4 Initialization Function

Given the three free parameters and the three constraints in our system, we can use various optimization or root finding techniques to solve for the parameter values that minimize our constraint equation. However, for efficiency it is beneficial to precompute offline an approximate mapping from relative state constraint space to parameter space to seed the constraint optimization process. This mapping can drastically speed up the algorithm by placing the initial guess of the control parameters close to the desired solution, reducing the number of online optimization steps required to reach the solution (within a desired precision). Given the high number of state variables and the fact that the system dynamics cannot be integrated in closed form, it is infeasible to precompute the entire mapping of state space to input space for any nontrivial system, such as the Boss vehicle model. We instead generate an approximation of this mapping through a five-dimensional lookup table with varying relative initial and terminal position ($\triangle x, \triangle y$), relative heading ($\triangle\theta$), initial curvature (κ_i), and constant velocities (v). Because this is only an approximation some optimization is usually required, however the initial seed from the lookup table significantly reduces the number of optimization iterations required from an arbitrary set of parameters.

Figure 4 provides an illustration of the lookup table generation process. First, the five dimensions of interest are discretized into a 5D table. Next, uniform sampling is used to sample from the set of all possible parameter values and the table positions each of these sample trajectories terminate in are recorded. The 5D table is then

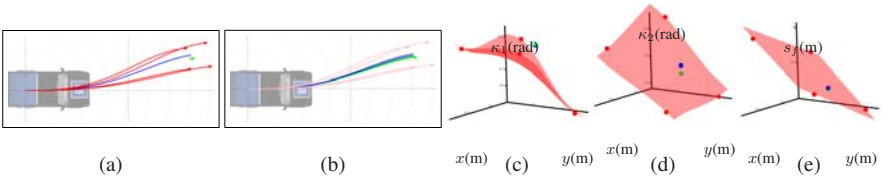


Fig. 5. Online Trajectory Generation. (a) Given an initial state and desired terminal state (relative terminal state shown in green), we find the closest elements of the lookup table (in red) and interpolate between the control parameters associated with these elements (interpolation of the free parameters is shown in graphs (c) through (e)) to come up with an initial approximation of the free parameters (resulting corresponding trajectory shown in blue). (b) This approximate trajectory is then optimized by modifying the free parameters based on the endpoint error, resulting in a sequence of trajectories that get closer to the desired terminal state. When the endpoint error is within an acceptable bound, the most recent parameter set is returned (trajectory shown in green). The interpolation over the free parameters κ_1, κ_2, and s_f is shown by the three graphs (c) through (e) (interpolated solutions shown in blue, final optimized solutions shown in green).

stepped through and for each position in the table the sample parameter values that came closest to this position are found. This parameter set is then optimized (using the optimization technique presented in the next section) to accurately match the table position, and then the resulting parameter set is stored in the corresponding index of the 5D table.

4.5 Trajectory Optimization

Given a set of parameters $\mathbf{p}$ that provide an approximate solution, it is then necessary to optimize these parameters to reduce the endpoint error and 'snap' the corresponding trajectory to the desired terminal state[2]. To do this, we linearize and invert our system of equations to produce a correction factor for the free control parameters based on the product of the inverted Jacobian and the current boundary state error. The Jacobian is model-invariant since it is determined numerically through central differences of simulated vehicle actions.

$$\Delta\mathbf{p} = -\left[\frac{\delta\mathbf{C}(\mathbf{x},\mathbf{p})}{\delta\mathbf{p}}\right]^{-1}\mathbf{C}(\mathbf{x},\mathbf{p}) \tag{8}$$

The control parameters are modified until the residual of the boundary state constraints is within an acceptable bound or until the optimization process diverges. In situations where boundary states are unachievable due to vehicle limitations, the optimization process predictably diverges as the partial derivatives in the Jacobian approach zero. The optimization history is then searched for the best candidate action (the most agressive action that gets closest to the state constraints) and this candidate is accepted or rejected basde on the magnitude of its error.

[2] Depending on the desired terminal state accuracy, it is sometimes possible to use the approximate parameters from the lookup table without any further optimization.

Figure 5 illustrates the online trajectory generation approach in action. Given a desired terminal state, we first look up from our table the closest terminal and initial states and their associated free parameter sets. We then interpolate between these closest parameter sets in 5D to produce our initial approximation of the parameter set to reach our desired terminal state. Figure 5(a) shows the lookup and interpolation steps, with the resulting parameter set values and corresponding trajectory. Figure 5(c) through (e) show the interpolation process for the free parameters[3]. Next, we evaluate the endpoint error of the resulting trajectory and we use this error to modify our parameter values to get closer to our desired terminal state, using the optimization approach just described. We repeat this optimization step until our endpoint error is within an allowed bound of the desired state (see Figure 5(b), and the resulting parameters and trajectory are stored and evaluated by the motion planner.

5 On-Road Planning

5.1 Path Extraction

During on-road navigation, the motion goal from the Behavioral Executive is a location within a road lane. The motion planner then attempts to generate a trajectory that moves the vehicle towards this goal location in the desired lane. To do this, it first constructs a curve along the centerline of the desired lane, representing the nominal path that the center of the vehicle should follow. This curve is then transformed into a path in rear-axle coordinates to be tracked by the motion planner.

5.2 Trajectory Generation

To robustly follow the desired lane and to avoid static and dynamic obstacles, the motion planner generates trajectories to a set of local goals derived from the centerline path. Each of these trajectories originates from the predicted state that the vehicle will reach by the time the trajectories will be executed. To calculate this state, forwards-prediction using an accurate vehicle model (the same model used in the trajectory generation phase) is performed using the trajectories selected for execution in previous planning episodes. This forwards-prediction accounts for both the high-level delays (the time required to plan) and the low-level delays (the time required to execute a command).

The goals are placed at a fixed longitudinal distance down the centerline path (based on the speed of the vehicle) but vary in lateral offset from the path to provide several options for the planner. The trajectory generation algorithm described above is used to compute dynamically feasible trajectories to these local goals. For each goal, two trajectories are generated: a smooth trajectory and a sharp trajectory. The

[3] Note that, for illustration purposes, only a subset of the parameter sets used for interpolation are shown in these figures (the full interpolation is in 5D not 2D) and this is why the interpolated result does not lie within the convex hull of the four sample points shown.

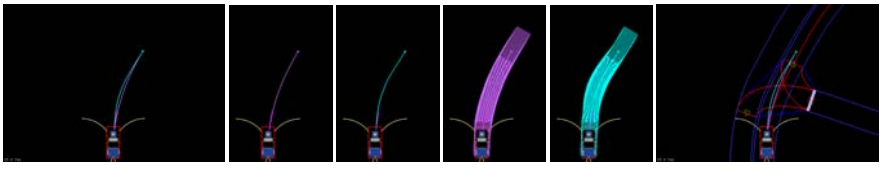


Fig. 6. Smooth and sharp trajectories

smooth trajectory has the initial curvature parameter κ_0 fixed to the curvature of the forwards-predicted vehicle state. The sharp trajectory has this parameter set to an offset value from the forwards-predicted vehicle state curvature to produce a sharp initial action. These sharp trajectories are useful for providing quick responses to suddenly appearing obstacles or dangerous actions of other vehicles.

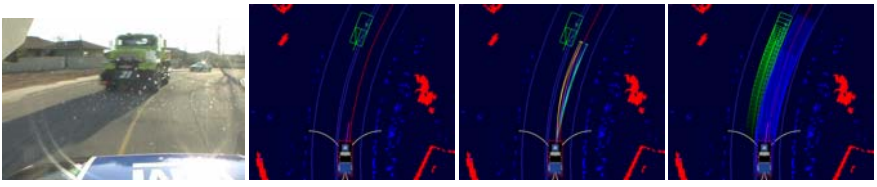
Figure 6 provides an example of smooth and sharp trajectories. The left-most image shows two trajectories (cyan and purple) generated to the same goal pose. The purple (smooth) trajectory exhibits continuous curvature control throughout; the cyan (sharp) trajectory begins with a discontinuous jump in commanded curvature, resulting in a sharp response from the vehicle. In these images, the initial curvature of the vehicle is shown by the short pink arc. The four center images show the individual sharp and smooth trajectories, along with the convolution of the vehicle along these trajectories. The right-most image illustrates how these trajectories are generated in practice for following a road lane.

5.3 Trajectory Velocity Profiles

The velocity profile used for each of the generated trajectories is selected from the set introduced in Section 4.3 based on several factors, including: the maximum speed given from the Behavioral Executive based on safe following distance to the lead vehicle, the speed limit of the current road segment, the maximum velocity feasible given the curvature of the centerline path, and the desired velocity at the local goal (e.g. if it is a stopline).

In general, profiles are chosen that maximize the speed of the vehicle at all times. Thus, typically a linear ramp profile is used, with a ramp velocity equal to the maximum speed possible and a linear component corresponding to the maximum acceleration possible. If the vehicle is slowly approaching a stop-line (or stopped short of the stop-line), a trapezoidal profile is employed so that the vehicle can both reach the stop-line quickly and come smoothly to a stop.

Multiple velocity profiles are considered for a particular trajectory when the initial profile results in a large endpoint error. This can occur when the rate of curvature required to reach the desired endpoint is not possible given the velocity imposed by the initial profile. In such cases, additional profiles with less aggressive speeds and accelerations are generated until either a valid trajectory is found or a maximum number have been evaluated (in our case, three per initial trajectory).



(a) (b) (c) (d)

Fig. 7. Following a road lane. These images show a single timeframe from the Urban Challenge.

5.4 Trajectory Evaluation

The resulting set of trajectories are then evaluated against their proximity to static and dynamic obstacles in the environment, as well as their distance from the centerline path, their smoothness, their endpoint error, and their speed. The best trajectory according to these metrics is selected and executed by the vehicle. Because the trajectory generator computes the feasibility of each trajectory using an accurate vehicle model, the selected trajectory can be directly executed by a vehicle controller.

One of the challenges of navigating in urban environments is avoiding other moving vehicles. To do this robustly and efficiently, we predict the future behavior of these vehicles and collision-check our candidate trajectories in state-time space against these predictions. We do this collision checking efficiently by using a hierarchical algorithm that performs a series of intersection tests between bounding regions for our vehicle and each other vehicle, with each test successively more accurate (and more computationally expensive). See (Ferguson et al., 2008) for more details on the algorithms used for prediction and efficient collision checking.

Figure 7 provides an example of the local planner following a road lane. Figure 7(b) shows the vehicle navigating down a two-lane road (detected obstacles and curbs shown as red and blue pixels, lane boundaries shown in blue, centerline of lane in red, current curvature of the vehicle shown in pink, minimum turning radius arcs shown in white) with a vehicle in the oncoming lane (in green). Figure 7(c) shows a set of trajectories generated by the vehicle given its current state and the centerline path and lane boundaries. From this set of trajectories, a single trajectory is selected for execution, as discussed above. Figure 7(d) shows the evaluation of one of these trajectories against both static and dynamic obstacles in the environment.

5.5 Lane Changing

As well as driving down the current lane, it is often necessary or desired in urban environments to perform lane changes. This may be to pass a slow or stalled vehicle in the current lane or move into an adjacent lane to prepare for an upcoming turn.

In our system, lane changes are commanded by the Behavioral Executive and implemented by the motion planner in a similar way to normal lane driving: a set of trajectories are generated along the centerline of the desired lane. However, because it is not always possible to perform the lane change immediately, an additional

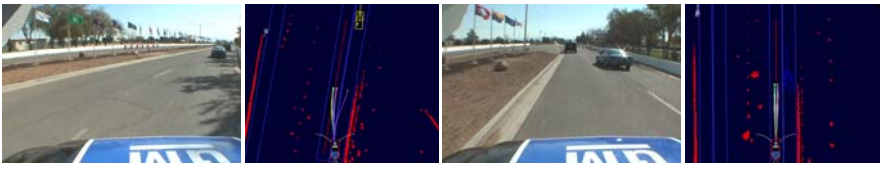
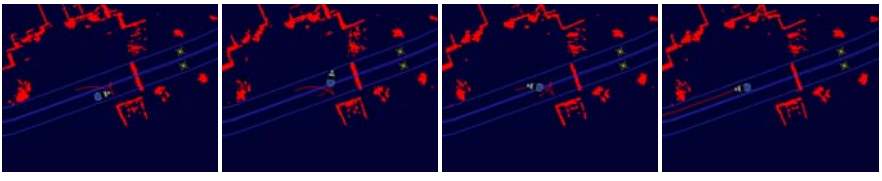


Fig. 8. Performing a lane change reliably and safely. Here, Boss changed lanes because another robot's chase vehicle was traveling too slowly in its original lane.



(a) (b) (c) (d)

Fig. 9. Performing a U-turn when encountering a road blockage. In this case the road was too narrow to perform a single forwards action to turn around and a three-point turn was required. (a) Initial plan generated to reverse the direction of the vehicle. (b, c) Tracking the plan. (d) Reverting back to lane driving after the vehicle has completely turned around and is in the correct lane.

trajectory is generated along the current lane in case none of the desired lane trajectories are feasible. Also, to ensure smooth lane changes, no sharp trajectories are generated in the direction of the current lane. Figure 8 provides an example lane change performed during the Urban Challenge to pass a chase vehicle.

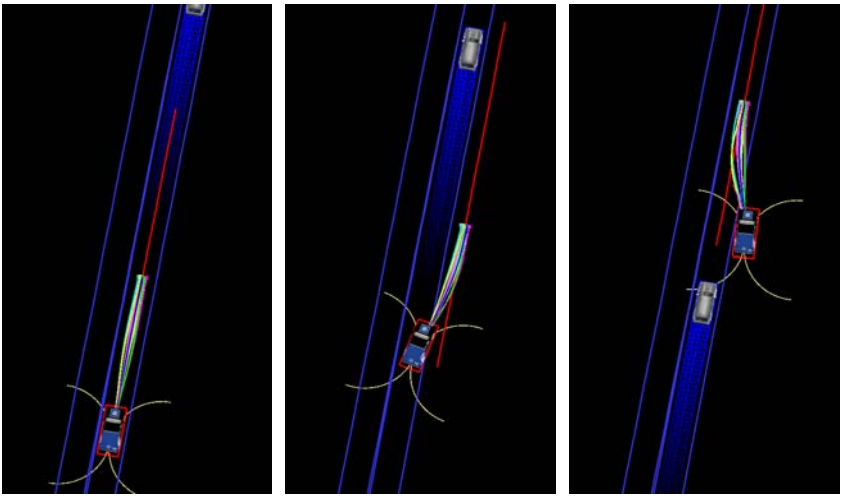
5.6 U-Turns

If the current road segment is blocked the vehicle must be able to turn around and find another route to its destination. In this scenario, Boss uses information about the dimensions of the road to generate a smooth path that turns the vehicle around. Depending on how constrained the road is, this path may consist of a single forwards segment (e.g. a traditional U-turn) or a three-point turn[4]. This path is then tracked in a similar fashion to the lane centerline paths, using a series of trajectories with varying offsets. Figure 9 provides an example three-point turn performed during one of the qualification events at the Urban Challenge.

5.7 Defensive Driving

One of the advanced requirements of the Urban Challenge was the ability to react safely to aberrant behavior of other vehicles. In particular, if another vehicle was

[4] If the road is extremely narrow then even a three-point turn may not be possible. In such cases, a more powerful lattice planner is invoked, as described in Section 7.2.



(a) (b) (c)

Fig. 10. Defensive Driving on roads. (a) Boss initially plans down its lane while the oncoming vehicle is far away. (b) When the oncoming vehicle is detected as dangerous, Boss generates a set of trajectories off the right side of the road. (c) After the oncoming vehicle has passed, Boss plans back onto the road and continues.

detected traveling the wrong direction in Boss' lane, it was the responsibility of Boss to pull off the road in a defensive driving maneuver to avoid a collision with the vehicle. To implement this behavior, the Behavioral Executive closely monitors other vehicles and if one is detected traveling towards Boss in its lane, the motion planner is instructed to move Boss off the right side of the road and come to a stop. This is performed in a similar fashion to a lane change to a hallucinated lane off the road but with a heavily reduced velocity so that Boss does not leave the road traveling too quickly and then comes to a stop once it is completely off the road. After the vehicle has passed, Boss then plans back onto the road and continues (see Figure 10).

5.8 Error Detection and Recovery

A central focus of our system-level approach was the detection of and recovery from anomalous situations. In lane driving contexts, such situations usually presented themselves through the motion planner being unable to generate any feasible trajectories to track the desired lane (for instance, if the desired lane is partially blocked and the on-road motion planner cannot plan a path through the blockage). In such cases, the Behavioral Executive issues a motion goal that invokes the more powerful, yet more computationally expensive, 4D lattice motion planner. If this goal is achieved, the system resumes with lane driving. If the motion planner is unable to reach this goal, the Behavioral Executive continues to generate new goals for

the lattice planner until one is satisfied. We provide more details on how the lattice planner interacts with these goals in the latter part of this article, and more details on the error detection and recovery process can be found in (Baker et al., 2008).

6 Unstructured Planning

During unstructured navigation, the motion goal from the Behavioral Executive is a pose (or set of poses) in the environment. The motion planner attempts to generate a trajectory that moves the vehicle towards this goal pose. However, driving in unstructured environments significantly differs from driving on roads. As mentioned in Section 5.1, when traveling on roads the desired lane implicitly provides a preferred path for the vehicle (the centerline of the lane). In unstructured environments there are no driving lanes and thus the movement of the vehicle is far less constrained.

To efficiently plan a smooth path to a distant goal pose, we use a lattice planner that searches over vehicle position (x, y), orientation (θ), and velocity (v). The set of possible local maneuvers considered for each (x, y, θ, v) state in the planner's search space are constructed offline using the same vehicle model as used in trajectory generation, so that they can be accurately executed by the vehicle. This planner searches in a backwards direction out from the goal pose(s) and generates a path consisting of a sequence of feasible high-fidelity maneuvers that are collision-free with respect to the static obstacles observed in the environment. This path is also biased away from undesirable areas within the environment such as curbs and locations in the vicinity of dynamic obstacles.

This global high-fidelity path is then tracked by a local planner that operates similarly to the on-road lane tracker, by generating a set of candidate trajectories that follow the path while allowing for some flexibility in local maneuvering. However, the nature of the trajectories generated in unstructured environments is slightly different. In the following sections, we describe in more detail the elements of our approach and how it exploits the context of its instantiation to adapt its behavior based on the situation (e.g. parking lot driving vs off-road error recovery).

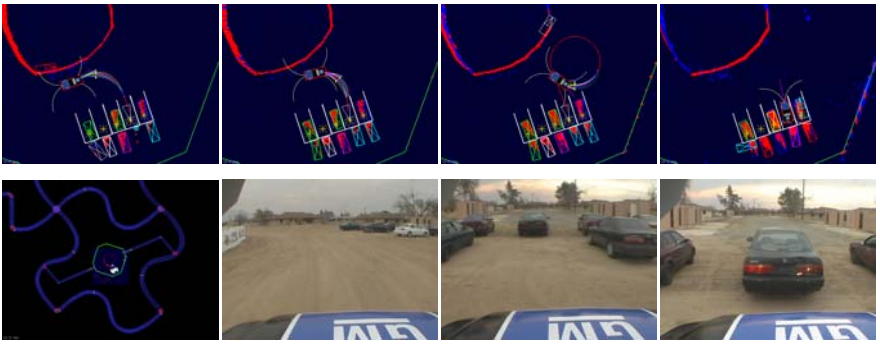


Fig. 11. Replanning when new information is received.

6.1 Planning Complex Maneuvers

To efficiently generate complex plans over large, obstacle-laden environments, the planner relies on an anytime, replanning search algorithm known as Anytime Dynamic A* (Anytime D*), developed by Likhachev et al. (Likhachev et al., 2005). Anytime D* quickly generates an initial, suboptimal plan for the vehicle and then improves the quality of this solution while deliberation time allows. The algorithm is also able to provide control over the suboptimality bound of the solution at all times during planning. Figure 19 shows an initial, suboptimal path converging over time to the optimal solution.

When new information concerning the environment is received (for instance, a new static or dynamic obstacle is observed), Anytime D* is able to efficiently repair its existing solution to account for the new information. This repair process is expedited by performing the search in a backwards direction, as in such a scenario updated information in the vicinity of the vehicle affects a smaller portion of the search space and so Anytime D* is able to reuse a large portion of its previously-constructed search tree in re-computing a new path. Figure 11 illustrates this replanning capability. These images were taken from a parking task performed during the National Qualification Event (the bottom-left image shows the parking lot in green and the neighboring roads in blue). The top-left image shows the initial path planned for the vehicle to enter the parking spot indicated by the white triangle. Several of the other spots were occupied by other vehicles (shown as rectangles of varying colors), with detected obstacles shown as red areas. The trajectories generated to follow the path are shown emanating from our vehicle (discussed later). As the vehicle gets closer to its intended spot, it observes a little more of the vehicle parked in the right-most parking spot (top, second-from-left image). At this point, it realizes its current path is infeasible and replans a new path that has the vehicle perform a loop and pull in smoothly. This path was favored in terms of time over stopping and backing up to re-position.

To further improve efficiency, the lattice planner uses a multi-resolution state and action space. In the vicinity of the goal and vehicle, where very complex maneuvering may be required, a dense set of actions and a fine-grained discretization of orientation are used during the search. In other areas, a coarser set of actions and discretization of orientation are employed. However, these coarse and dense resolution areas both share the same dimensionality and seamlessly interface with each other, so that resulting solution paths overlapping both coarse and dense areas of the space are smooth and feasible. Figure 12 illustrates how the dense and coarse action and state spaces differ.

The effectiveness of the Anytime D* algorithm is highly dependent on its use of an informed heuristic to focus its search. An accurate heuristic can reduce the time and memory required to generate a solution by orders of magnitude, while a poor heuristic can diminish the benefits of the algorithm. It is thus important to devote careful consideration to the heuristic used for a given search space.

Since in our setup Anytime D* searches backwards, the heuristic value of a state estimates the cost of a path from the robot pose to that state. Anytime D* requires these values to be admissible (not to overestimate the actual path cost) and

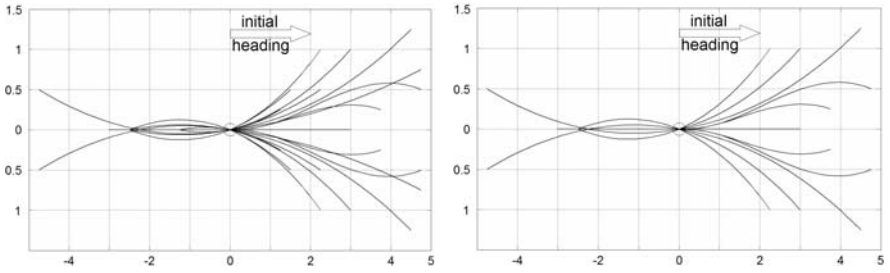
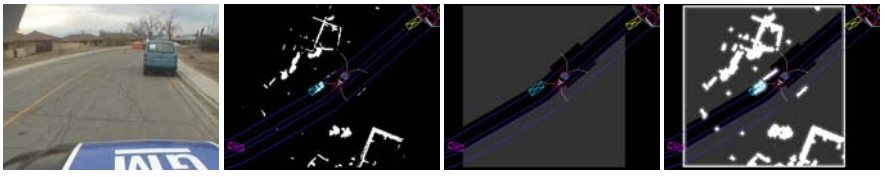


Fig. 12. Dense and coarse resolution action spaces. The coarse action space contains many fewer actions (24 versus 36 in the dense action space) with transitions only to states with a coarse-resolution heading discretization (in our case, 16 headings versus 32 in the dense-resolution discretization). In both cases the discretization in position is 0.25 meters and the axes in both diagrams are in meters.

consistent (Pearl, 1984). For any state (x, y, θ, v), the heuristic we use is the maximum of two values. The first value is the cost of an optimal path from the robot pose to (x, y, θ, v) assuming a completely empty environment. These values are precomputed offline and stored in a heuristic lookup table (Knepper and Kelly, 2006). This is a very well informed heuristic function when operating in sparse environments and is guaranteed to be admissible. The second value is the cost of a 2D path from the robot (x_r, y_r) coordinates to (x, y) given the actual environment. These values are computed online by a 2D grid-based Dijkstra's search. This second heuristic function is very useful when operating in obstacle-laden environments. By taking the maximum of these two heuristic values we are able to incorporate both the constraints of the vehicle and the constraints imposed by the obstacles in the environment. The result is a very well-informed heuristic function that can speed up the search by an order of magnitude relative to either of the component heuristics alone. For more details concerning the benefit of this combined heuristic function and other optimizations implemented in our lattice planner, including its multi-resolution search space and how it efficiently performs convolutions and replanning, see (Likhachev and Ferguson, 2008) and (Ferguson and Likhachev, 2008).

6.1.1 Incorporating Environmental Constraints

In addition to the geometric obstacle information provided by perception, we incorporate context-specific constraints on the movement of the vehicle by creating an additional cost map known as a constrained map. This 2D grid-based cost map encodes the relative desirability of different areas of the environment based on the road structure in the vicinity and, if available, prior terrain information. This constrained cost map is then combined with the static map from perception to create the final combined cost map to be used by the lattice planner. Specifically, for each cell (i, j) in the combined cost map $\mathcal{C}$, the value of $\mathcal{C}(i, j)$ is computed as the maximum of $\mathcal{EPC}(i, j)$ and $\mathcal{CO}(i, j)$, where $\mathcal{EPC}(i, j)$ is the static map value at (i, j) and $\mathcal{CO}(i, j)$ is the constrained cost map value at (i, j).



(a) (b) (c) (d)

Fig. 13. A snapshot from a qualification run during the Urban Challenge, showing (b) the obstacle map from perception (obstacles in white), (c) the constrained cost map based on the road structure (lighter areas are more costly), and (c) the resulting combined cost map used by the planner.

For instance, when invoking the lattice planner to plan a maneuver around a parked car or jammed intersection, the constrained cost map is used to specify that staying within the desired road lane is preferable to traveling in an oncoming lane, and similarly that driving off-road to navigate through a cluttered intersection is dangerous. To do this, undesirable areas of the environment based on the road structure are assigned high costs in the constrained cost map. These can be both soft constraints (undesirable but allowed areas), which correspond to high costs, and hard constraints (forbidden areas), which correspond to infinite costs. Figure 13 shows the constrained cost map generated for an on-road maneuver, along with the expanded perception cost map and the resulting combined cost map used by the planner.

6.1.2 Incorporating Dynamic Obstacles

The combined cost map of the planner is also used to represent dynamic obstacles in the environment so that these can be avoided by the planner. The perception system of Boss represents static and dynamic obstacles independently, which allows the motion planner to treat each type of obstacle differently. The lattice planner adapts the dynamic obstacle avoidance behavior of the vehicle based on its current proximity to each dynamic obstacle. If the vehicle is close to a particular dynamic obstacle, that obstacle and a short-term prediction of its future trajectory is encoded into the combined cost map as a hard constraint so that it is strictly avoided. For every dynamic obstacle, both near and far, the planner encodes a varying high-cost region around the obstacle to provide a safe clearance. Although these high-cost regions are not hard constraints, they result in the vehicle avoiding the vicinity of the dynamic obstacles if at all possible. Further, the generality of this approach allows us to influence the behavior of our vehicle based on the specific behavior of the dynamic obstacles. For instance, we offset the high-cost region based on the relative position of the dynamic obstacle and our vehicle so that we will favor moving to the right, resulting in yielding behavior in unstructured environments quite similar to how humans react in these scenarios. Figure 14 provides an example scenario involving a dynamic obstacle along with the corresponding cost map generated.

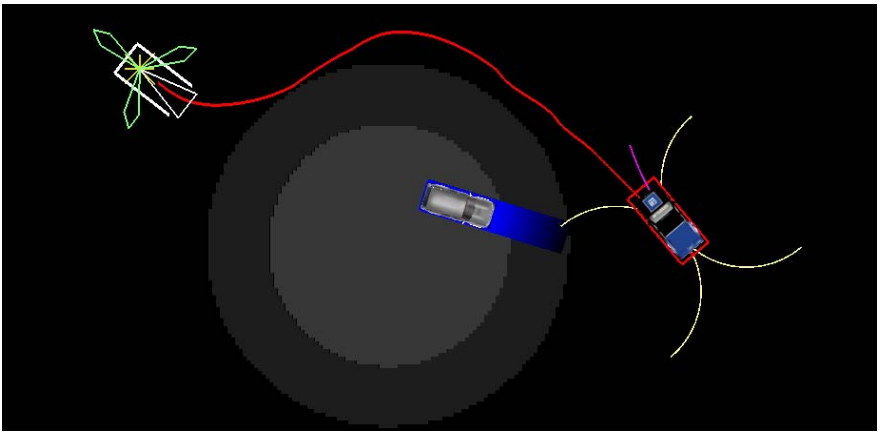
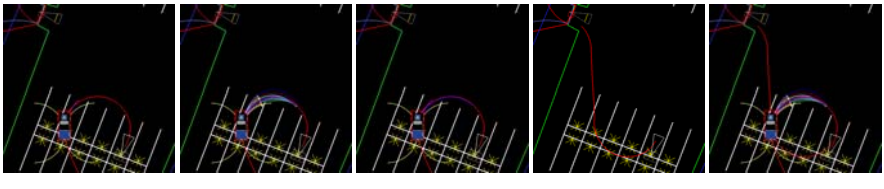


Fig. 14. Biasing the cost map for the lattice planner so that the vehicle keeps away from dynamic obstacles. Notice that the high-cost region around the dynamic obstacle is offset to the left so that Boss will prefer moving to the right of the vehicle.

7 Tracking Complex Paths

The resulting lattice plan is then tracked in a similar manner to the paths extracted from road lanes: the motion planner generates a set of trajectories that attempt to follow the plan while also allowing for local maneuverability. However, in contrast to when following lane paths, the trajectories generated to follow the lattice path all attempt to terminate on the path. Each trajectory is in fact a concatenation of two short trajectories, with the first of the two short trajectories ending at an offset position from the path and the second ending back on the path. By having all concatenated trajectories return to the path we significantly reduce the risk of having the vehicle move itself into a state that is difficult to leave.

As mentioned earlier, the motion planner generates trajectories at a fixed 10Hz during operation. The lattice planner also nominally runs at 10Hz. However, in very



(a) (b) (c) (d) (e)

Fig. 15. Following a lattice plan to a parking spot. Here, one lattice planner is updating the path to the spot while another is simultaneously pre-planning a path out of the spot. The goals are represented by the white (current goal) and grey (next goal) triangles.

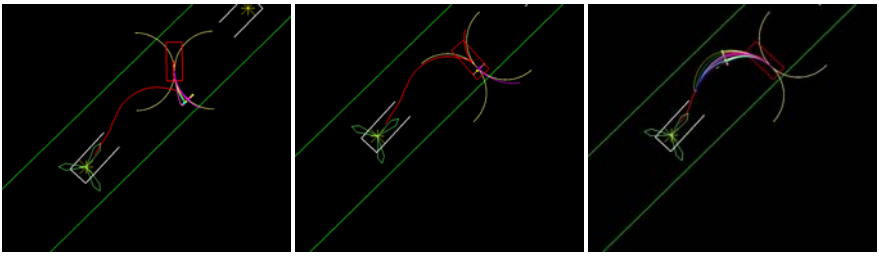


Fig. 16. Reversing during path tracking. The goal is represented by the green star inside the white parking spot.

difficult planning scenarios the lattice planner may take longer (up to a couple seconds) to generate its initial solution and it is for this reason that preplanning is performed whenever possible (as will be discussed later). The motion planner continues to track the current lattice path until it is updated by the lattice planner.

Figure 15 provides an example of the local planner following a lattice plan to a specified parking spot. Figure 15(a) shows the lattice plan generated for the vehicle (in red) towards the desired parking spot (desired pose of the vehicle shown as the white triangle). Figure 15(b) shows the set of trajectories generated by the vehicle to track this plan, and Figure 15(c) shows the best trajectory selected by the vehicle to follow the path.

Both forwards and reverse trajectories are generated as appropriate based on the velocity of the lattice path being tracked. When the path contains an upcoming velocity switching point, or cusp point, the local planner generates trajectories that bring the vehicle to a stop at the cusp point. Figure 16 shows reverse trajectories generated to a cusp point in the lattice path.

As mentioned above, one of the desired capabilities of our vehicle was to be able to exhibit human-like yielding behavior in parking lots, to allow for safe, natural interaction with other vehicles. Through our biased cost function, the lattice planner typically generates paths through parking lots that keep to the right of other vehicles. However, it is possible that another vehicle may be quickly heading directly towards Boss, requiring evasive action similar to the on-road defensive driving maneuvers discussed in Section 5.7. In such a case, Boss' local planner detects that it is unable to continue along its current course without colliding with the other vehicle and it then generates a set of trajectories that are offset to the right of the path. The intended behavior here is for each vehicle to move to the right to avoid a collision. Figure 17 provides an example of this behavior in a large parking lot.

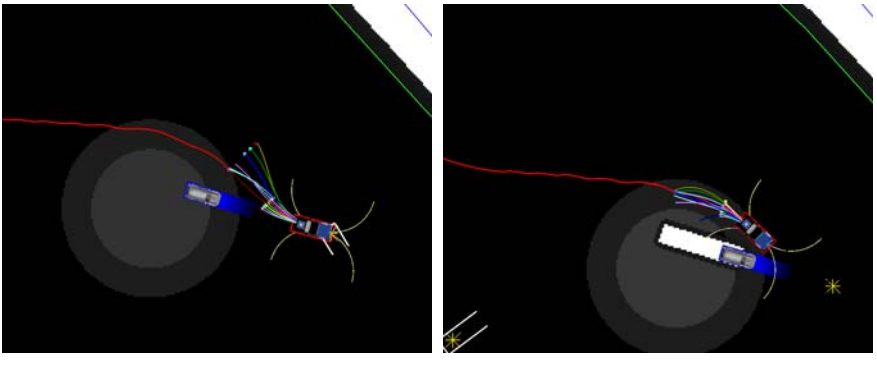
In addition to the general optimizations always employed by the lattice planner, there are several context-specific reasoning steps performed in different urban driving scenarios for which the lattice planner is invoked. In the following two sections we describe different methods used to provide optimized, intelligent behavior in parking lots and on-road error recovery scenarios.

7.1 Planning in Parking Lots

Because the location of parking lots is known a priori, this information can be exploited by the motion planning system to improve the efficiency and behavior of the lattice planner within these areas. First, we can use the extents of the parking lot to constrain the vehicle through the constrained cost map. To do this, we use the a priori specified extents of the parking lot to set all cells outside the lot (and not part of entry or exit lanes) in the constrained cost map to be hard constraints. This constrains the vehicle to operate only inside the lot. We also include a high cost buffer around the perimeter of the parking lot to bias the vehicle away from the boundaries of the lot.

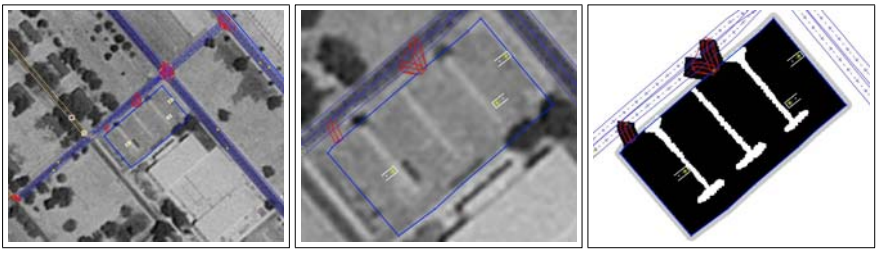
When prior terrain information exists such as overhead imagery, this information can also be incorporated into the constrained cost map to help provide global guidance for the vehicle. For instance, this information can be used to detect features such as curbs or trees in parking lots that should be avoided, so that these features can be used in planning before they are detected by onboard perception. Figure 18(a,b) shows overhead imagery of a parking lot area used to encode curb islands into a constrained cost map for the parking lot, and Figure 18(c) shows the corresponding constrained cost map. This constrained cost map is then stored offline and loaded by the planner online when it begins planning paths through the parking lot. By storing the constrained cost maps for parking lots offline we significantly reduce online processing as generating the constrained cost maps for large, complex parking lots can take several seconds.

Further, because the parking lot goals are also known in advance of entering the parking lot (e.g. the vehicle knows which parking spot it is intending on reaching), the lattice planner can pre-plan to the first goal pose within the parking lot while the vehicle is still approaching the lot. By planning a path from the entry point of the parking lot in advance, the vehicle can seamlessly transition into the lot without needing to stop, even for very large and complex lots.



(a) (b)

Fig. 17. Defensive driving when in unstructured environments. (a) Trajectories thrown to right of path (other vehicle should likewise go to right). (b) New path planned from new position.



(a) (b) (c)

Fig. 18. Generating constrained cost maps offline for Castle Commerce Center, CA. (a) Overhead imagery showing testing area with road network overlaid. (b) Parking lot area (boundary in blue) with overhead imagery showing curb islands. (c) Resulting constrained cost map incorporating boundaries, entry and exit lanes, and curb islands (the lighter the color, the higher the cost).

Figure 19 illustrates the pre-planning used by the lattice planner. The left-most image shows our vehicle approaching a parking lot (boundary shown in green), with its intended parking spot indicated by the white triangle (and multi-colored set of goal poses). While the vehicle is still outside the lot, it begins planning a path from one of the entries to the desired spot, and the path converges to the optimal solution well before the vehicle enters the lot.

In a similar vein, when the vehicle is in a lot traveling towards a parking spot, we use a second lattice planner to simultaneously plan a path from that spot to the next desired location (e.g. the next parking spot to reach or an exit of the lot). When the vehicle reaches its intended parking spot, it then immediately follows the path from this second planner, again eliminating any time spent waiting for a plan to be generated.

Figure 15 provides an example of the use of multiple concurrent lattice planners. Figure 15(a) shows the lattice plan generated towards the desired parking spot. Figure 15(d) shows the path simultaneously being planned out of this spot to the exit of the parking lot, and Figure 15(e) shows the paths from both planners at the same time.

7.2 Planning in Error Recovery Scenarios

The lattice planner is flexible enough to be used in a large variety of cases that can occur during on-road and unstructured navigation. In particular, it is used during error recovery when navigating congested lanes or intersections and to perform difficult U-turns. In such cases, the nominal on-road motion planner determines that it is unable to generate any feasible trajectory and reports its failure to the Behavioral Executive, which in turn issues an unstructured goal pose (or set of poses) to the motion planner and indicates that it is in an error recovery mode. The motion planner then uses the lattice planner to generate a path to the set of goals, with the lattice planner determining during its planning which goal is easiest to reach. In these error recovery scenarios the lattice planner is biased to avoid areas that could result

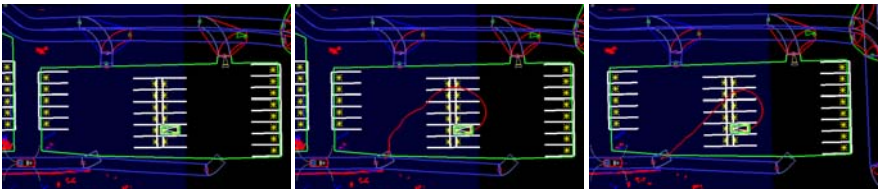
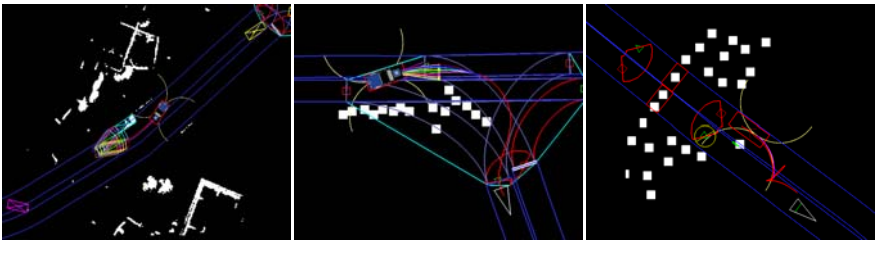


Fig. 19. Pre-planning a path into a parking spot and improving this path in an anytime fashion. A set of goal poses are generated that satisfy the parking spot and an initial path is planned while the vehicle is still outside the parking lot. This path is improved as the vehicle approaches, converging to the optimal solution shown in the right image.



(a) (b) (c)

Fig. 20. Error Recovery examples. (a) Planning around a stalled vehicle in the road (obstacles in white, set of pose goals given to planner shown as various colored rectangles, and resulting path shown in red). Corresponds to scenario from Figure 13. (b) Planning through a partially blocked intersection. (c) Performing a complex U-turn in a cluttered area (Boss' 3D model has been removed to see the lattice plan underneath the vehicle).

in unsafe behavior (such as oncoming lanes when on roads) through increasing the cost of undesirable areas in the constrained cost map (see Figure 13).

The ability to cope with anomalous situations was a key focus of Boss' software system and the lattice planner was used as a powerful tool to maneuver the vehicle to arbitrary locations in the environment. Figure 20 provides several error recovery examples involving the lattice planner.

The lattice planner is also invoked when the road cannot be detected with certainty due to the absence of markers (e.g. an unpaved road with berms). In this case, the lattice planner is used to bias the movement of the vehicle to stay within any detected geometric extents of the road.

8 Results and Discussion

Our motion planning system was developed over the course of more than a year and tested over thousands of kilometers of autonomous operation in three different extensive testing sites, as well as the Urban Challenge event itself. Through this extensive testing, all components were hardened and the planners were incrementally improved upon.

A key factor in our system-level design was that Boss should never give up. As such, the lattice planner was designed to be general enough and powerful enough to plan in extremely difficult scenarios. In addition, the Behavioral Executive was designed to issue an infinite sequence of pose goals to the motion planner should it continue to fail to generate plans (see (Baker et al., 2008) for details of this error recovery framework). And in the final Urban Challenge event, as anticipated, this error recovery played a large part in Boss' successful completion of the course. Over the course of the three final event missions, there were 17 different instances where the lattice planner was invoked due to an encountered anomalous situation (some of these included getting cut off at intersections, coming across other vehicles blocking the lane, and perception occasionally observing heavy dust clouds as static obstacles).

Incorporation of an accurate vehicle model was also an important design choice for Boss' motion planning system. This allowed Boss to push the limits of acceleration and speed and travel as fast as possible along the road network, confident in its execution. Combining this model with an efficient on-road planner that could safely handle the high-speeds involved was also central to Boss' on-road performance.

In addition, the efficiency and path quality of the lattice planner enabled Boss to also travel smoothly and quickly through parking lot areas, without ever needing to pause to generate a plan. As well as generating smooth paths in (x, y, θ), by also considering the velocity dimension v the lattice planner was able to explicitly reason about the time required to change direction of travel and was thus able to generate very fast paths even when complex maneuvers were required. Overall, the focus on execution speed and smoothness strongly contributed to Boss finishing the four-hour race 19 minutes and 8 seconds faster than its nearest competitor (DARPA, 2008).

One of the important lessons learned during the development of this system was that it is often extremely beneficial to exploit prior, offline processing to provide efficient online planning performance. We used this idea in several places, from the generation of lookup tables for the trajectory generator and lattice planner heuristic function to the precomputing of constrained cost maps for parking lots. This prior processing saved us considerably at run-time. Further, even when faced with calculations that cannot be precomputed offline, such as planning paths through novel environments, it can often pay to begin planning before a plan is required. This concept was the basis for our preplanning on approach to parking lots and our concurrent planning to both current and future goals, and it enabled us to produce high-quality solutions without needing to wait for these solutions to be generated.

Finally, although simplicity was central to our high-level system development and significant effort was put into making the interfacing between processes as lightweight as possible, we found that in regards to motion planning, although simple, approximate planning algorithms can work well in most cases, generality and completeness when needed are priceless. Using a high-fidelity, high-dimensional lattice planner for unstructured planning problems proved time and time again to be the right choice for our system.

9 Prior Work

Existing research on motion planning for autonomous outdoor vehicles can be roughly broken into two classes: motion planning for autonomous vehicles following roads and motion planning for autonomous vehicles navigating unstructured environments including off-road areas and parking lots. A key difference between road following and navigating unstructured environments is that in the former case a global plan is already encoded by the road (lane) itself, and therefore the planner only needs to generate short-term motion trajectories that follow the road, while in the latter case no such global plan is provided.

9.1 On-Road Planning

A vast amount of research has been conducted in the area of road following. Some of the best known and fully-operational systems include the CMU NavLab project (Thorpe et al., 1988), the INRIA autonomous car project (Baber et al., 2005) in France, the VaMoRs (Dickmanns et al., 1993) and VITA projects (Ulmer, 1992) in Germany, and the Personal Vehicle System (PVS) project (Hattori et al., 1992) in Japan. Approaches to road following in these and other systems varies drastically. For example, one of the road following algorithms used by NavLab vehicles is ALVINN (Pomerleau, 1991) which learns the mapping from road images onto control commands using Neural Network by observing how the car is driven manually for several minutes. In the VITA project, on the other hand, the planner was used to track the lane while regulating the velocity of the vehicle in response to the curvature of the road and the distance to nearby vehicles and obstacles. Stanford University's entry in the second DARPA Grand Challenge also exhibited lane following behavior through evaluating a set of candidate trajectories that tracked the desired path (Thrun et al., 2006). Our lane planning approach is closely related to theirs, however to generate their candidate trajectories they sample the control space around a base trajectory (e.g. the trajectory leading down the center of the lane), while we sample the state space along the road lane. Some significant advantages of using a state space approach include the ability to finely control position and heading at the terminal state of each trajectory (which we can align with the road shape), the ability to impose the requirement that each trajectory terminates at exactly the same distance along the path, allowing for fairer evaluation of candidate actions, and the simplification of generating complex maneuvers such as U-turns and lane changes.

However, several of these existing approaches have been shown to be very effective in road following in normal conditions. A major strength of our approach, on the other hand, is that it can handle difficult scenarios, such as when a road is partially blocked (e.g., by an obstacle, a stalled car, a slow-moving car or a car driving in an opposite direction but moving out of the bounds of its own lane). Our system can handle these scenarios robustly and at high speeds.

9.2 Unstructured Planning

Roboticists have concentrated on the problem of mobile robot navigation in unstructured environments for several decades. Early approaches concentrated on performing local planning, where very short term reasoning is performed to generate the next action for the vehicle (Khatib, 1986; Simmons, 1996; Fox et al., 1997). A major limitation of these purely local approaches was their capacity to get the vehicle stuck in local minima en route to the goal (for instance, cul-de-sacs). To improve upon this limitation, algorithms were developed that incorporated global as well as local information (Thrun et al., 1998; Brock and Khatib, 1999; Kelly, 1995; Philippsen and Siegwart, 2003). Subsequent approaches have focused on improving the local planning component of these approaches by using more sophisticated local action sets that better follow the global value function (Thrun et al., 2006; Howard and Kelly, 2007), and by generating sequences of actions to perform more complex local maneuvers (Stachniss and Burgard, 2002; Urmson et al., 2006; Braid et al., 2006). In parallel, researchers have concentrated on improving the quality of global planning, so that a global path can be easily tracked by the vehicle (LaValle and Kuffner, 2001; Song and Amato, 2001; Likhachev et al., 2003; Likhachev et al., 2005; Pivtoraiko and Kelly, 2005; Knepper and Kelly, 2006). However, the computational expense of generating complex global plans over large distances has remained very challenging, and the approaches to date have been restricted to either small distances, fairly simple environments, or highly suboptimal solutions. Our lattice-based global planner is able to efficiently generate feasible global paths over much larger distances than previously possible, while providing suboptimality bounds on the quality of the solutions and anytime improvement of the solutions generated.

10 Conclusions

We have presented the motion planning framework for an autonomous vehicle navigating through urban environments. Our approach combines a high-fidelity trajectory generation algorithm for computing dynamically-feasible actions with an efficient lane-based planner, for on-road planning, and a 4D lattice planner, for unstructured planning. It has been implemented on an autonomous vehicle that has traveled over 3000 autonomous kilometers and we have presented sample illustrations and results from the Urban Challenge, which it won in November 2007.

Acknowledgements

This work would not have been possible without the dedicated efforts of the Tartan Racing team and the generous support of our sponsors including General Motors, Caterpillar, and Continental. This work was further supported by DARPA under contract HR0011-06-C-0142.

References

Special Issue on the DARPA Grand Challenge, Part 1. Journal of Field Robotics 23(8) (2006a)

Special Issue on the DARPA Grand Challenge, Part 2. Journal of Field Robotics 23(9) (2006b)

Baber, J., Kolodko, J., Noel, T., Parent, M., Vlacic, L.: Cooperative autonomous driving: intelligent vehicles sharing city roads. IEEE Robotics and Automation Magazine 12(1), 44–49 (2005)

Baker, C., Ferguson, D., Dolan, J.: Robust mission execution for autonomous urban driving. In: Proceedings of the International Conference on Intelligent Autonomous Systems, IAS (2008)

Braid, D., Broggi, A., Schmiedel, G.: The TerraMax autonomous vehicle. Journal of Field Robotics 23(9), 693–708 (2006)

Brock, O., Khatib, O.: High-speed navigation using the global dynamic window approach. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA (1999)

Carsten, J., Rankin, A., Ferguson, D., Stentz, A.: Global path planning on-board the Mars Exploration Rovers. In: Proceedings of the IEEE Aerospace Conference (2007)

DARPA Urban Challenge Official Results (2008), http://www.darpa.mil/GRANDCHALLENGE/mediafaq.asp

Dickmanns, E.D., Behringer, R., Brudigam, C., Dickmanns, D., Thomanek, F., Holt, V.: All-transputer visual autobahn-autopilot/copilot. In: Proceedings of the 4th Int. Conference on Computer Vision ICCV, pp. 608–615 (1993)

Ferguson, D., Darms, M., Urmson, C., Kolski, S.: Detection, Prediction, and Avoidance of Dynamic Obstacles in Urban Environments. In: Proceedings of the IEEE Intelligent Vehicles Symposium, IV (2008)

Ferguson, D., Likhachev, M.: Efficiently using cost maps for planning complex maneuvers. In: Proceedings of the Workshop on Planning with Cost Maps, IEEE International Conference on Robotics and Automation (2008)

Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics and Automation 4(1) (1997)

Hattori, A., Hosaka, A., Taniguchi, M., Nakano, E.: Driving control system for an autonomous vehicle using multiple observed point information. In: Proceedings of Intelligent Vehicle Symposium (1992)

Howard, T., Kelly, A.: Optimal rough terrain trajectory generation for wheeled mobile robots. International Journal of Robotics Research 26(2), 141–166 (2007)

Kelly, A.: An Intelligent Predictive Control Approach to the High Speed Cross Country Autonomous Navigation Problem. PhD thesis, Carnegie Mellon University (1995)

Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research 5(1), 90–98 (1986)

Knepper, R., Kelly, A.: High performance state lattice planning using heuristic look-up tables. In: Proceedings of the IEEE International Conference on Intelligent Robots and Systems, IROS (2006)

LaValle, S., Kuffner, J.: Rapidly-exploring Random Trees: Progress and prospects. Algorithmic and Computational Robotics: New Directions, pp. 293–308 (2001)

Likhachev, M., Ferguson, D.: Planning Dynamically Feasible Long Range Maneuvers for Autonomous Vehicles. In: Proceedings of Robotics: Science and Systems, RSS (2008)

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., Thrun, S.: Anytime Dynamic A*: An Anytime, Replanning Algorithm. In: Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS (2005)

Likhachev, M., Gordon, G., Thrun, S.: ARA*: Anytime A* with provable bounds on sub-optimality. In: Advances in Neural Information Processing Systems. MIT Press, Cambridge (2003)

Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading (1984)

Philippsen, R., Siegwart, R.: Smooth and efficient obstacle avoidance for a tour guide robot. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA (2003)

Pivtoraiko, M., Kelly, A.: Constrained motion planning in discrete state spaces. In: Proceedings of the International Conference on Advanced Robotics, FSR (2005)

Pomerleau, D.: Efficient training of artificial neural networks for autonomous navigation. Neural Computation 3(1), 88–97 (1991)

Simmons, R.: The curvature velocity method for local obstacle avoidance. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA (1996)

Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., Schwehr, K.: Recent progress in local and global traversability for planetary rovers. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA (2000)

Song, G., Amato, N.: Randomized motion planning for car-like robots with C-PRM. In: Proceedings of the IEEE International Conference on Intelligent Robots and Systems, IROS (2001)

Stachniss, C., Burgard, W.: An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In: Proceedings of the IEEE International Conference on Intelligent Robots and Systems, IROS (2002)

Stentz, A., Hebert, M.: A complete navigation system for goal acquisition in unknown environments. Autonomous Robots 2(2), 127–145 (1995)

Thorpe, C., Hebert, M., Kanade, T., Shafer, S.: Vision and navigation for the Carnegie-Mellon Navlab. IEEE Transactions on Pattern Analysis and Machine Intelligence 10(3), 362–373 (1988)

Thorpe, C., Jochem, T., Pomerleau, D.: The 1997 automated highway demonstration. In: Proceedings of the International Symposium on Robotics Research, ISRR (1997)

Thrun, S., et al.: Map learning and high-speed navigation in RHINO. In: Kortenkamp, D., Bonasso, R.P., Murphy, R. (eds.) AI-based Mobile Robots: Case Studies of Successful Robot Systems, MIT Press, Cambridge (1998)

Thrun, S., et al.: Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics 23(9), 661–692 (2006)

Ulmer, B.: VITA - an autonomous road vehicle (arv) for collision avoidance in traffic. In: Proceedings of Intelligent Vehicle Symposium, pp. 36–41 (1992)

Urmson, C., et al.: A robust approach to high-speed navigation for unrehearsed desert terrain. Journal of Field Robotics 23(8), 467–508 (2006)

Urmson, C., et al.: Autonomous driving in urban environments: Boss and the Urban Challenge. Journal of Field Robotics 25(8), 425–466 (2008)

A Vehicle Model

Boss' vehicle model predicts the resulting vehicle state $\mathbf{x}_{t+\Delta t}$ after applying a parameterized set of controls $\mathbf{u}(\mathbf{p},\mathbf{x})$ to an initial vehicle state $\mathbf{x}_t$. It does this by forwards-simulating the movement of the vehicle given the commanded controls. However, it also constrains these controls based on the physical constraints of the vehicle and safety bounds. Algorithms 1 through 3 provide pseudocode of this process (the main vehicle model function is *MotionModel* in Algorithm 3). Values for the parameters used in each function are defined in Table 1.

Algorithm 1. SpeedControlLogic($\mathbf{x}_{t+\Delta t}$)

Input: $\mathbf{x}_{t+\Delta t}$
Output: $\mathbf{x}_{t+\Delta t}$

$|v|_{cmd} \leftarrow |\,[v_{t+\Delta t}]_{cmd}\,|$; // calculate speed

$[|v|_{cmd}]_{max} \leftarrow max\left[|v|_{scl}, \left[\frac{\kappa_{t+\Delta t}-a}{b}\right]\right]$; // compute safe speed

$[\kappa]_{max,scl} \leftarrow min\left[[\kappa]_{max}, a+b\,|v|_{cmd}\right]$; // compute safe curvature

if $\left[|\kappa_{t+\Delta t}| \geq [\kappa]_{max,scl}\right]$ **then**

  $|v|_{cmd} \leftarrow |safety_{factor} \cdot [|v|_{cmd}]_{max}\,|$; // check for safe speed

$[v_{t+\Delta t}]_{cmd} \leftarrow |v|_{cmd}\frac{[v_t]_{cmd}}{|[v_t]_{cmd}|}$; // update velocity command

return $\mathbf{x}_{t+\Delta t}$;

Algorithm 2. DynamicsResponse($\mathbf{x}_t$,$\mathbf{x}_{t+\Delta t}$, Δt)

Input: $\mathbf{x}_t$,$\mathbf{x}_{t+\Delta t}$, Δt
Output: $\mathbf{x}_{t+\Delta t}$

$\left[\frac{d\kappa}{dt}\right]_{cmd} \leftarrow \frac{[\kappa_{t+\Delta t}]_{cmd}-\kappa_t}{dt}$; // compute curvature rate command

$\left[\frac{d\kappa}{dt}\right]_{cmd} \leftarrow min\left[\frac{d\kappa}{dt}, \left[\frac{d\kappa}{dt}\right]_{max}\right]$; // upper bound curvature rate

$\left[\frac{d\kappa}{dt}\right]_{cmd} \leftarrow max\left[\frac{d\kappa}{dt}, \left[\frac{d\kappa}{dt}\right]_{min}\right]$; // lower bound curvature rate

$\mathbf{x}_{t+\Delta t} \leftarrow$ SpeedControlLogic $[\mathbf{x}_{t+\Delta t}]$; // speed control logic

$\kappa_{t+\Delta t} \leftarrow \kappa_t + \left[\frac{d\kappa}{dt}\right]_{cmd}\Delta t$; // compute curvature at time $t+\Delta t$

$\kappa_{t+\Delta t} \leftarrow min\left[\kappa_{t+\Delta t}, \kappa_{max}\right]$; // upper bound curvature

$\kappa_{t+\Delta t} \leftarrow max\left[\kappa_{t+\Delta t}, \kappa_{min}\right]$; // lower bound curvature

$\left[\frac{dv}{dt}\right]_{cmd} \leftarrow \frac{[v_{t+\Delta t}]_{cmd}-vt}{dt}$; // compute acceleration command

$\left[\frac{dv}{dt}\right]_{cmd} \leftarrow min\left[\left[\frac{dv}{dt}\right]_{cmd}, \left[\frac{dv}{dt}\right]_{max}\right]$; // upper bound acceleration

$\left[\frac{dv}{dt}\right]_{cmd} \leftarrow max\left[\left[\frac{dv}{dt}\right]_{cmd}, \left[\frac{dv}{dt}\right]_{min}\right]$; // lower bound acceleration

$v_{t+\Delta t} \leftarrow v_t + \left[\frac{dv}{dt}\right]_{cmd}\Delta t$; // compute velocity at time $t+\Delta t$

return $\mathbf{x}_{t+\Delta t}$;

Algorithm 3. MotionModel($\mathbf{x}_t$, $\mathbf{u}(\mathbf{p},\mathbf{x})$, Δt)

Input: $\mathbf{x}_t$, $\mathbf{u}(\mathbf{p},\mathbf{x})$, Δt
Output: $\mathbf{x}_{t+\Delta t}$

$x_{t+\Delta t} \leftarrow x_t + v_t \cos[\theta_t]\,\Delta t$; // compute change in 2D x-position
$y_{t+\Delta t} \leftarrow y_y + v_t \sin[\theta_t]\,\Delta t$; // compute change in 2D y-position
$\theta_{t+\Delta t} \leftarrow \theta_t + v_t \kappa_t \Delta t$; // compute change in 2D orientation
$[\kappa_{t+\Delta t}]_{cmd} \leftarrow \mathbf{u}[\mathbf{p}, s]$; // get curvature command
$[v_{t+\Delta t}]_{cmd} \leftarrow \mathbf{u}[\mathbf{p}, t - t_{delay}]$; // get velocity command
$[a_{t+\Delta t}]_{cmd} \leftarrow \mathbf{u}[\mathbf{p}, t - t_{delay}]$; // get acceleration command
$\mathbf{x}_{t+\Delta t} \leftarrow$ DynamicsResponse($\mathbf{x}_t$,$\mathbf{x}_{t+\Delta t}$, Δt) ; // estimate response
return $\mathbf{x}_{t+\Delta t}$;

Table 1. Parameters used in vehicle model

Description	Parameter	Raceday Values
maximum curvature	$[\kappa]_{max}$	$0.1900 rad$
minimum curvature	$[\kappa]_{min}$	$-0.1900 rad$
maximum rate of curvature	$\left[\frac{d\kappa}{dt}\right]_{max}$	$0.1021 \frac{rad}{sec}$
minimum rate of curvature	$\left[\frac{d\kappa}{dt}\right]_{min}$	$-0.1021 \frac{rad}{sec}$
maximum acceleration	$\left[\frac{dv}{dt}\right]_{max}$	$2.000 \frac{m}{sec}$
maximum deceleration	$\left[\frac{dv}{dt}\right]_{min}$	$-6.000 \frac{m}{sec}$
control latency	t_{delay}	$0.0800 sec$
speed control logic “a” coefficient	a_{scl}	0.1681
speed control logic “b” coefficient	b_{scl}	-0.0049
speed control logic threshold	$\lvert v\rvert_{scl}$	$4.000 \frac{m}{sec}$
max curvature for speed	$[\kappa v]_{max}$	$0.1485 rad$
speed control logic safety factor	$safety_{factor}$	1.000

Junior: The Stanford Entry in the Urban Challenge

Michael Montemerlo[1], Jan Becker[4], Suhrid Bhat[2], Hendrik Dahlkamp[1], Dmitri Dolgov[1], Scott Ettinger[3], Dirk Haehnel[1], Tim Hilden[2], Gabe Hoffmann[1], Burkhard Huhnke[2], Doug Johnston[1], Stefan Klumpp[2], Dirk Langer[2], Anthony Levandowski[1], Jesse Levinson[1], Julien Marcil[2], David Orenstein[1], Johannes Paefgen[1], Isaac Penny[1], Anna Petrovskaya[1], Mike Pflueger[2], Ganymed Stanek[2], David Stavens[1], Antone Vogt[1], and Sebastian Thrun[1]

[1] Stanford Artificial Intelligence Lab, Stanford University, Stanford CS 94305
[2] Electronics Research Lab, Volkswagen of America, 4009 Miranda Avenue, Palo Alto, CA 94304
[3] Intel Research, 2200 Mission College Blvd., Santa Clara, CA 95052
[4] Robert Bosch LLC, Research and Technology Center, 4009 Miranda Avenue, Palo Alto, CA 94304

Abstract. This article presents the architecture of Junior, a robotic vehicle capable of navigating urban environments autonomously. In doing so, the vehicle is able to select its own routes, perceive and interact with other traffic, and execute various urban driving skills including lane changes, U-turns, parking, and merging into moving traffic. The vehicle successfully finished and won second place in the DARPA Urban Challenge, a robot competition organized by the U.S. Government.

1 Introduction

The vision of self-driving cars promises to bring fundamental change to one of the most essential aspects of our daily lives. In the U.S. alone, traffic accidents cause the loss of over 40,000 people annually, and a substantial fraction of the world's energy is used for personal car-based transportation [U.S. Department of Transportation, 2005]. A safe, self-driving car would fundamentally improve the safety and comfort of the driving population, while reducing the environmental impact of the automobile.

In 2003, the Defense Advanced Research Projects Agency (DARPA) initiated a series of competitions aimed at the rapid technological advancement of autonomous vehicle control. The first such event, the "DARPA Grand Challenge," led to the development of vehicles that could confidently follow a desert trail at average velocities nearing 20mph [Buehler et al., 2006]. In October 2005, Stanford's robot "Stanley" won this challenge and became the first robot to finish the 131-mile long course [Montemerlo et al., 2006]. The "DARPA Urban Challenge," which took place on November 3, 2007, brought about vehicles that could navigate in traffic in a mock urban environment.

The rules of the DARPA Urban Challenge were complex [DARPA, 2007]. Vehicles were provided with a digital street map of the environment, in the form of a *Road Network Description File*, or RNDF. The RNDF contained geometric

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 91–123.
springerlink.com

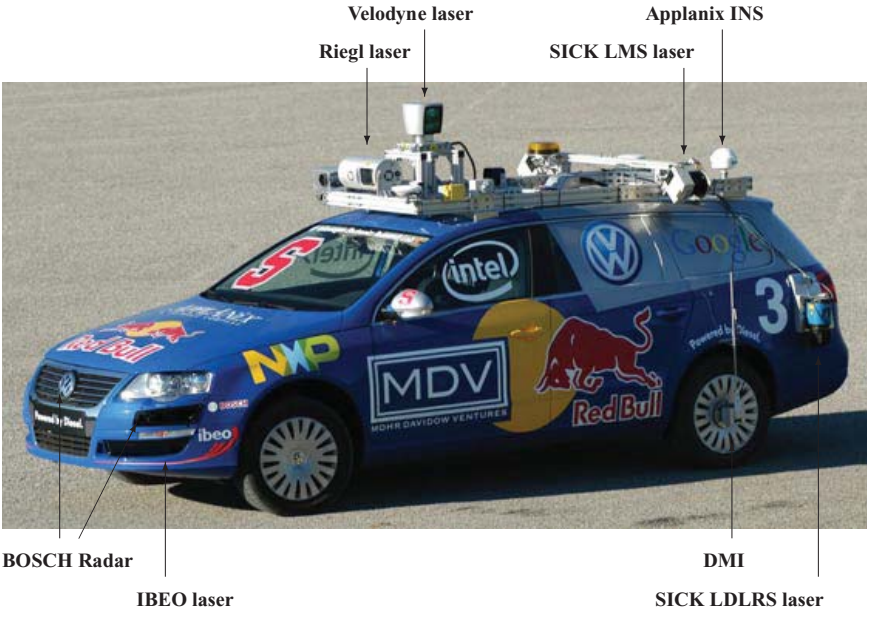


Fig. 1. Junior, our entry in the DARPA Urban Challenge. Junior is equipped with five different laser measurement systems, a multi-radar assembly, and a multi-signal inertial navigation system, as shown in this figure.

information on lanes, lane markings, stop signs, parking lots, and special checkpoints. Teams were also provided with a high-resolution aerial image of the area, enabling them to manually enhance the RNDF before the event. During the Urban Challenge event, vehicles were given multiple missions, defined as sequences of checkpoints. Multiple robotic vehicles carried out missions in the same environment at the same time, possibly with different speed limits. When encountering another vehicle, each robot had to obey traffic rules. Maneuvers that were specifically required for the Urban Challenge included: passing parked or slow-moving vehicles, precedence handling at intersections with multiple stop signs, merging into fast-moving traffic, left turns across oncoming traffic, parking in a parking lot, and the execution of U-turns in situations where a road is completely blocked. Vehicle speeds were generally limited to 30mph, with lower speed limits in many places. DARPA admitted eleven vehicles to the final event, of which the present vehicle was one.

"Junior," the robot shown in Figure 1, is a modified 2006 Volkswagen Passat Wagon, equipped with five laser rangefinders (manufactured by IBEO, Riegl, SICK, and Velodyne), an Applanix GPS-aided inertial navigation system, five BOSCH radars, two Intel quad core computer systems, and a custom drive-by-wire interface developed by Volkswagen's Electronic Research Lab. The vehicle has an obstacle detection range of up to 120 meters, and reaches a maximum velocity of 30mph, the maximum speed limit according to the Urban Challenge rules. Junior made its

Fig. 2. All computing and power equipment is placed in the trunk of the vehicle. Two Intel quad core computers (bottom right) run the bulk of all vehicle software. Other modules in the trunk rack include a power server for selectively powering individual vehicle components, and various modules concerned with drive-by-wire and GPS navigation. A 6 DOF inertial measurement unit is also mounted in the trunk of the vehicle, near the rear axle.

driving decisions through a distributed software pipeline that integrates perception, planning, and control. This software is the focus of the present article.

Junior was developed by a team of researchers from Stanford University, Volkswagen, and its affiliated corporate sponsors: Applanix, Google, Intel, Mohr Davidow Ventures, NXP, and Red Bull. This team was mostly comprised of the original Stanford Racing Team, which developed the winning entry "Stanley" in the 2005 DARPA Grand Challenge [Montemerlo et al., 2006]. In the Urban Challenge, Junior placed second, behind a vehicle from Carnegie Mellon University, and ahead of the third-place winner from Virginia Tech.

2 Vehicle

Junior is a modified 2006 Passat wagon, equipped with a 4-cylinder turbo diesel injection engine. The 140 hp vehicle is equipped with a limited-torque steering motor, an electronic brake booster, electronic throttle, gear shifter, parking brake, and turn signals. A custom interface board provides computer control over each of these vehicle elements. The engine provides electric power to Junior's computing system through a high-current prototype alternator, supported by a battery-backed electronically controlled power system. For development purposes, the cabin is equipped with switches that enable a human driver to engage various electronic interface components at will. For example, a human developer may choose the computer to control the steering wheel and turn signals, while retaining manual control over the throttle and the vehicle brakes. These controls were primarily for testing purposes; during the actual competition, no humans were allowed inside the vehicles.

For inertial navigation, an Applanix POS LV 420 system provides real-time integration of multiple dual-frequency GPS receivers which includes a GPS Azimuth

Heading measurement subsystem, a high-performance inertial measurement unit, wheel odometry via a distance measurement unit (DMI), and the Omnistar satellite-based Virtual Base Station service. The real-time position and orientation errors of this system were typically below 100 cm and 0.1 degrees, respectively.

Two side-facing SICK LMS 291-S14 sensors and a forward-pointed RIEGL LMS-Q120 laser sensor provide measurements of the adjacent 3-D road structure and infrared reflectivity measurements of the road surface for lane marking detection and precision vehicle localization.

For obstacle and moving vehicle detection, a Velodyne HDL-64E is mounted on the roof of the vehicle. The Velodyne, which incorporates 64 laser diodes and spins at up to 15 Hz, generates dense range data covering a 360 horizontal field-of-view and a 30 degree vertical field-of-view. The Velodyne is supplemented by two SICK LDLRS sensors mounted at the rear of the vehicle, and two IBEO ALASCA XT lidars mounted in the front bumper. Five BOSCH Long Range Radars (LRR2) mounted around the front grill provide additional information about moving vehicles.

Junior's computer system consists of two Intel quad core servers. Both computers run Linux, and they communicate over a gigabit ethernet link.

3 Software Architecture

Junior's software architecture is designed as a data driven pipeline in which individual modules process information asynchronously. This same software architecture was employed successfully by Junior's predecessor Stanley in the 2005 challenge [Montemerlo et al., 2006]. Each module communicates with other modules via an anonymous publish/subscribe message passing protocol, based on the Inter Process Communication Toolkit (IPC) [Simmons and Apfelbaum, 1998].

Modules subscribe to message streams from other modules, which are then sent asynchronously. The result of the computation of a module may then be published to other modules. In this way, each module is processing data at all times, acting as a pipeline. The time delay between entry of sensor data into the pipeline to the effect on the vehicle's actuators is approximately 300ms. The software is roughly organized into five groups of modules.

- **sensor interfaces** – The sensor interfaces manage communication with the vehicle and individual sensors, and make resulting sensor data available to the rest of the software modules.
- **perception modules** – The perception modules segment the environment data into moving vehicles and static obstacles. They also provide precision localization of the vehicle relative to the digital map of the environment.
- **navigation modules** – The navigation modules determine the behavior of the vehicle. The navigation group consists of a number of motion planners, plus a hierarchical finite state machine for invoking different robot behaviors and preventing deadlocks.

Table 1. Table of processes running during the Urban Challenge.

Process name	Computer	Description
PROCESS-CONTROL	1	starts and restarts processes, adds process control via IPC
APPLANIX	1	Applanix interface (via IPC).
LDLRS1 & LDLRS2	1	SICK LDLRS laser interface (via IPC).
IBEO	1	IBEO laser interface (via IPC).
SICK1 & SICK2	1	SICK LMS laser interfaces (via IPC).
RIEGL	1	Riegl laser interface (via IPC).
VELODYNE	1	Velodyne laser interface (via IPC and shared memory). This module also projects the 3d points using Applanix pose information.
CAN	1	CAN bus interface
RADAR1 - RADAR5	1	Radar interfaces (via IPC).
PERCEPTION	1	IPC/Shared Memory interface of Velodyne data, obstacle detection, dynamic tracking and scan differencing
RNDF_LOCALIZE	1	1D localization using RNDF
HEALTHMON	1	logs computer health information (temperature, processes, CPU and memory usage)
PROCESS-CONTROL	2	start/restarts processes and adds process control over IPC
CENTRAL	2	IPC-server
PARAM_SERVER	2	central server for all parameters
ESTOP	2	IPC/serial interface to DARPA E-stop
HEALTHMON	2	monitors the health of all modules
POWER	2	IPC/serial interface to power-server (relay card)
PASSAT	2	IPC/serial interface to vehicle interface board
CONTROLLER	2	vehicle motion controller
PLANNER	2	path planner and hybrid A* planner

- **drive-by-wire interface** – Controls are passed back to the vehicle through the drive-by-wire interface. This module enables software control of the throttle, brake, steering, gear shifting, turn signals, and emergency brake.
- **global services** – A number of system level modules provide logging, time stamping, message passing support, and watchdog functions to keep the software running reliably.

Table 1 lists the actual processes running on the robot's computers during the race event, and Figure 3 shows a overview of the data flow between modules.

4 Environment Perception

Junior's perceptual routines address a wide variety of obstacle detection and tracking problems. Figure 4a shows a scan from the primary obstacle detection sensor, the Velodyne. Scans from the IBEO lasers, shown in Figure 4b, and LDLRS lasers are used to supplement the Velodyne data in blind spots. A radar system complements the laser system as an early warning system for moving objects in intersections.

4.1 Laser Obstacle Detection

In urban environments, the vehicle encounters a wide variety of static and moving obstacles. Obstacles as small as a curb may trip a fast-moving vehicle, so detecting small objects is of great importance. Overhangs and trees may look like large

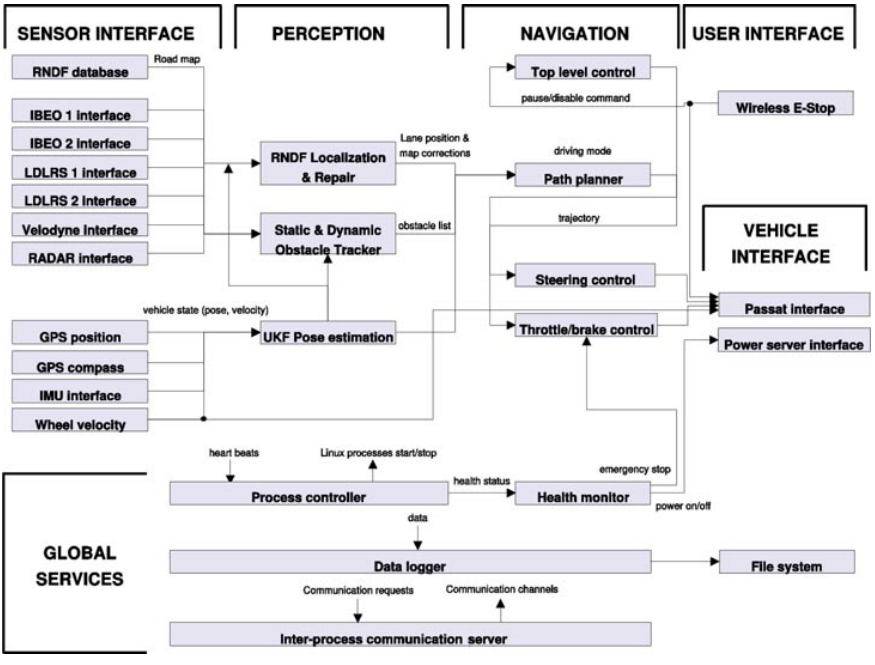
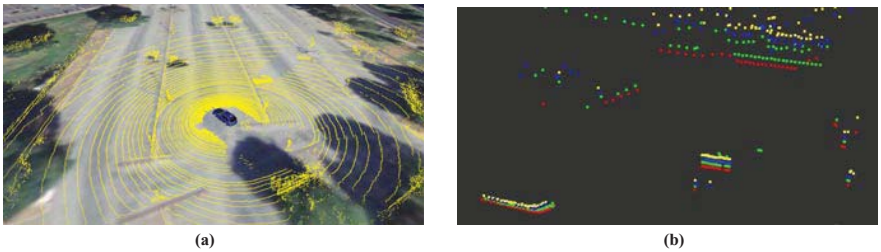


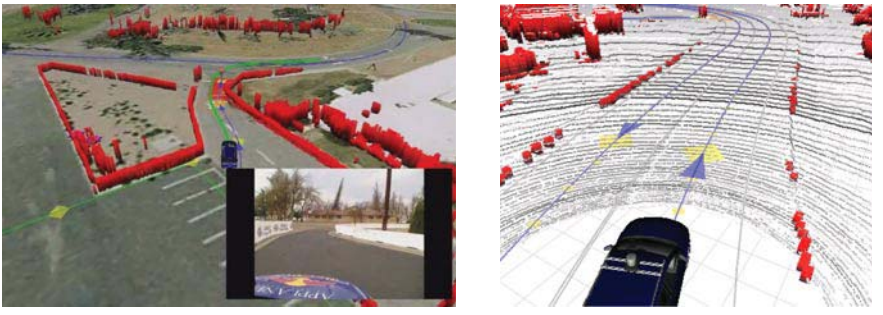
Fig. 3. Flow diagram of the Junior Software.

obstacles at a distance, but traveling underneath is often possible. Thus, obstacle detection must consider the 3-D geometry of the world. Figure 5 depicts a typical output of the obstacle detection routine in an urban environment. Each red object corresponds to an obstacle. Towards the bottom right, a camera image is shown for reference.



(a) (b)

Fig. 4. (a) The Velodyne contains 64 laser sensors and rotates at 10 Hz. It is able to see objects and terrain out to 60 meters in every direction. (b) The IBEO sensor possesses four scan lines which are primarily parallel to the ground. The IBEO is capable of detecting large vertical obstacles, such as cars and signposts.



(a) (b)

Fig. 5. Obstacles detected by the vehicle are overlayed over aerial imagery (left) and Velodyne data (right). In the example on the right, the curbs along both sides of the road are detected.

The robot's primary sensor for obstacle detection is the Velodyne laser. A simple algorithm for detecting obstacles in Velodyne scans would be to find points with similar x-y coordinates whose vertical displacement exceeds a given threshold. Indeed, this algorithm can be used to detect large obstacles such as pedestrians, signposts, and cars. However, range and calibration error are high enough with this sensor that the displacement threshold cannot be set low enough in practice to detect curb-sized objects without substantial numbers of false positives.

An alternative to comparing vertical displacements is to compare the range returned by two adjacent beams, where "adjacency" is measured in terms of the pointing angle of the beams. Each of the 64 lasers has a fixed pitch angle relative to the vehicle frame, and thus would sweep out a circle of a fixed radius on a flat ground plane as the sensor rotates. Sloped terrain locally compresses these rings, causing the distance between adjacent rings to be smaller than the inter-ring distance on flat terrain. In the extreme case, a vertical obstacle causes adjacent beams to return nearly equal ranges. Because the individual beams strike the ground at such shallow angles, the distance between rings is a much more sensitive measurement of terrain slope than vertical displacement. By finding points that generate inter-ring distances that differ from the expected distance by more than a given threshold, even obstacles that are not apparent to the vertical thresholding algorithm can be reliably detected.

In addition to terrain slope, rolling and pitching of the vehicle will cause the rings traced out by the individual lasers to compress and expand. If this is not taken into account, rolling to the left can cause otherwise flat terrain to the left of the vehicle to be detected incorrectly as an obstacle. This problem can be remedied by making the expected distance to the next ring a function of range, rather than the index of the particular laser. Thus as the vehicle rolls to the left, the expected range difference for a specific beam decreases as the ring moves closer to the vehicle. Implemented in this way, small obstacles can be reliably detected even as the sensor rolls and pitches.

Two more issues must be addressed when performing obstacle detection in urban terrain. First, trees and other objects frequently overhang safe driving surfaces and

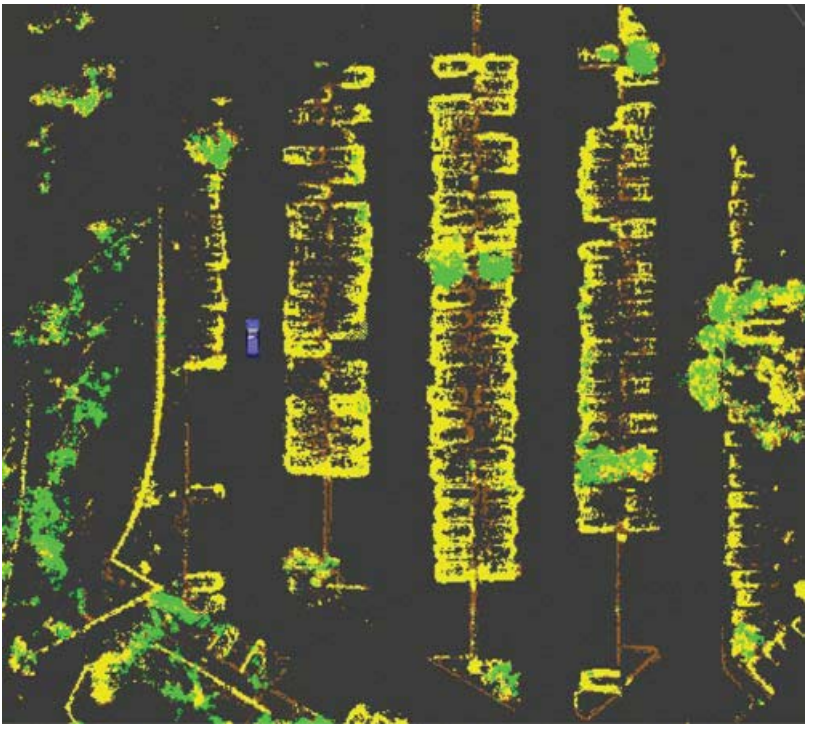
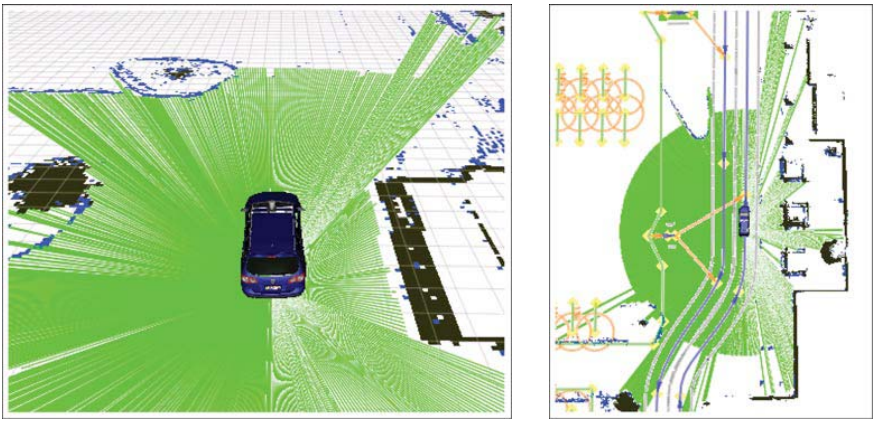


Fig. 6. A map of a parking lot. Obstacles colored in yellow are tall obstacles, brown obstacles are curbs, and green obstacles are overhanging objects (e.g. tree branches) that are of no relevance to ground navigation.

should not be detected as obstacles. Overhanging objects are filtered out by comparing their height with a simple ground model. Points that fall in a particular x-y grid cell that exceed the height of the lowest detected point in the same cell by more than a given threshold (the height of the vehicle plus a safety buffer), are ignored as overhanging obstacles.

Second, the Velodyne sensor possesses a "blind spot" behind the vehicle. This is the result of the sensor's geometry and mounting location. Further, it also cannot detect small obstacles such as curbs in the immediate vicinity of the robot due to self-occlusion. Here the IBEO and SICK LDLRS sensors are used to supplement the Velodyne data. Because both of these sensors are essentially 2-D, ground readings cannot be distinguished from vertical obstacles, and hence obstacles can only be found at very short range (where ground measurements are unlikely). Whenever either of these sensors detects an object within a close range (15 meters for the LDLRS and 5 meters for the IBEO), the measurement is flagged as an obstacle. This combination between short-range sensing in 2-D and longer range sensing using the 3-D sensor provides high reliability. We note that a 5 meter cut-off for the IBEO sensor may seem overly pessimistic, as this laser is designed for long range detection



(a) (b)

Fig. 7. Examples of free space analysis for Velodyne scans. The green lines represent the area surrounding the robot that is observed to be empty. This evidence is incorporated into the static map, shown in black and blue.

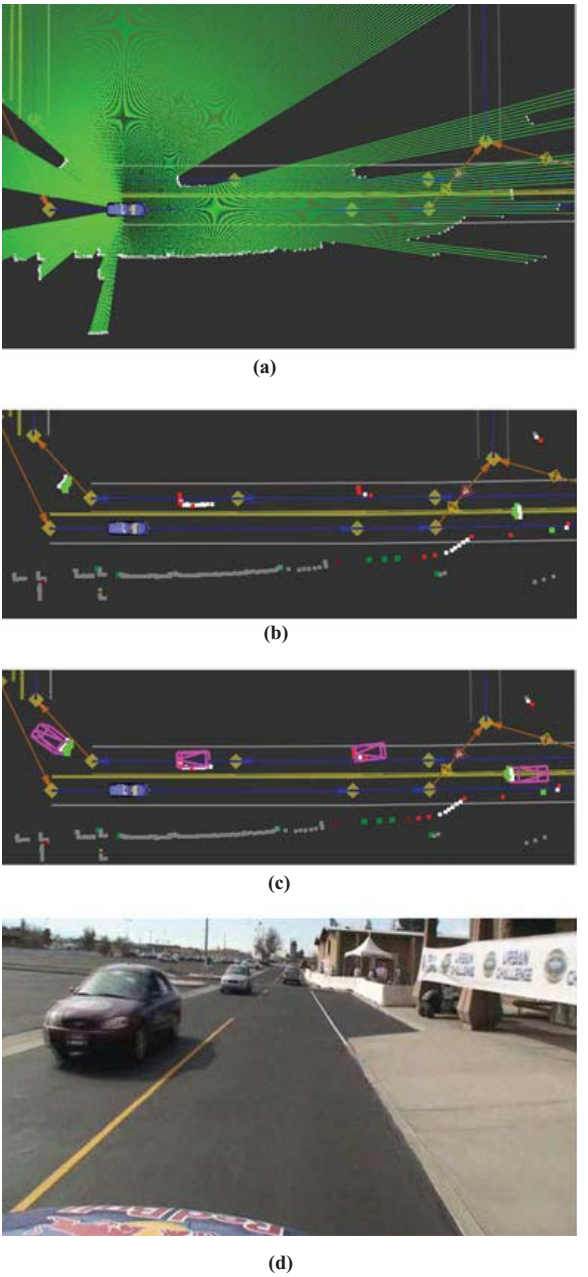
(100 meters and more). However, the sensor presents a large number of false positive detections on non-flat terrain, such as dirt roads.

Our obstacle detection method worked exceptionally well. In the Urban Challenge, we know of no instance in which our robot Junior collided with an obstacle. In particular, Junior never ran over a curb. We also found that the number of false positives was remarkably small, and false positives did not measurably impact the vehicle performance. In this sense, static obstacle detection worked flawlessly.

4.2 Static Mapping

In many situations, multiple measurements have to be integrated over time even for static environment mapping. Such is the case, for example, in parking lots, where occlusion or range limitations may make it impossible to see all relevant obstacles at all times. Integrating multiple measurements is also necessary to cope with certain blind spots in the near range of the vehicle. In particular, curbs are only detectable beyond a certain minimum range with a Velodyne laser. To alleviate these problems, Junior caches sensor measurement into local maps. Figure 6 shows such a local map, constructed from many sensor measurements over time. Different colors indicate different obstacle types on a parking lot. The exact map update rule relies on the standard Bayesian framework for evidence accumulation [Moravec, 1988]. This safeguards the robot against spurious obstacles that only show up in a small number of measurements.

A key downside of accumulating static data over time into a map arises from objects that move. For example, a passage may be blocked for a while, and then become drivable again. To accommodate such situations, the software performs a local visibility calculation. In each polar direction away from the robot, the grid



(a)

(b)

(c)

(d)

Fig. 8. (a) Synthetic 2-D scan derived from Velodyne data. (b) Scan differencing provides areas in which change has occurred, colored here in green and red. (c) Tracks of other vehicles. (d) The corresponding camera image.

cells between the robot and the nearest detected object are observed to be free. Beyond the first detected obstacle, of course, it is impossible to say whether the absence of further obstacles is due to occlusion. Hence, no map updating takes place beyond this range. This mechanism may still lead to an overly conservative map, but empirically works well for navigating cluttered spaces such as parking lots. Figure 7 illustrates the region in which free space is detected in a Velodyne sensor scan.

4.3 Dynamic Object Detection and Tracking

A key challenge in successful urban driving pertains to other moving traffic. The present software provides a reliable method for moving object detection and prediction based on particle filters.

Moving object detection is performed on a synthetic 2-D scan of the environment. This scan is synthesized from the various laser sensors by extracting the range to the nearest detected obstacle along an evenly spaced array of synthetic range sensors. The use of such a synthetic scan comes with several advantages over the raw sensor data. First, its compactness allows for efficient computation. Second, the method is applicable to any of the three obstacle-detecting range sensors (Velodyne, IBEO, and SICK LDLRS), and any combination thereof. The latter property stems from the fact that any of those laser measurements can be mapped easily into a synthetic 2-D range scan, rendering the scan representation relatively sensor-independent. This synergy thus provides our robot with a unified method for finding, tracking, and predicting moving objects. Figure 8a shows such a synthetic scan.

The moving object tracker then proceeds in two stages. First, it identifies *areas of change*. For that, it compares two synthetic scans acquired over a brief time interval. If an obstacle in one of the scans falls into the free space of the respective other scan, this obstacle is a witness of motion. Figure 8b shows such a situation. The red color of a scan corresponds to an obstacle that is new, and the green color marks the absence of a previously seen obstacle.

When such witnesses are found, the tracker initializes a set of particles as possible object hypotheses. These particles implement rectangular objects of different dimensions, and at slightly different velocities and locations. A particle filter algorithm is then used to track such moving objects over time. Typically, within three sightings of a moving object, the filter latches on and reliably tracks the moving object.

Figure 8c depicts the resulting tracks; a camera image of the same scene is shown in Figure 8d. The tracker estimates the location, the yaw, the velocity, and the size of the object.

5 Precision Localization

One of the key perceptual routines in Junior's software pertains to localization. As noted, the robot is given a digital map of the road network in form of an RNDF. While the RNDF is specified in GPS coordinates, the GPS-based inertial position computed by the Applanix system is generally not able to recover the coordinates of

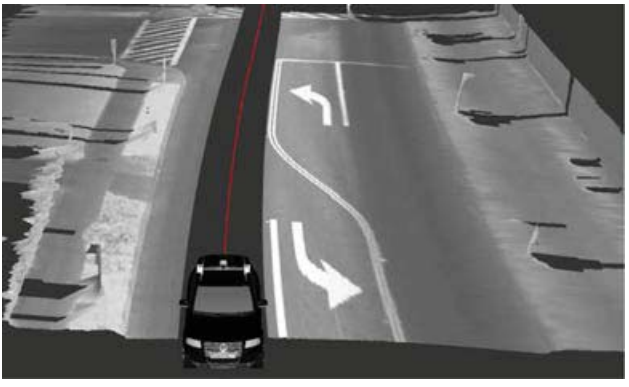


Fig. 9. The side lasers provide intensity information that is matched probabilistically with the RNDF for precision localization.

the vehicle with sufficient accuracy to perform reliable lane keeping without sensor feedback. Further, the RNDF is itself inaccurate, adding further errors if the vehicle were to blindly follow the road using the RNDF and Applanix pose estimates. Junior therefore estimates a local alignment between the RNDF and its present position using local sensor measurements. In other words, Junior continuously localizes itself relative to the RNDF.

This fine-grained localization uses two types of information: road reflectivity and curb-like obstacles. The reflectivity is sensed using the RIEGL LMS-Q120 and the SICK LMS sensors, both of which are pointed towards the ground. Fig. 9 shows the reflectivity information obtained through the sideways mounted SICK sensors, and integrated over time. This diagram illustrates the varying infrared reflectivity of the lane markings.

The filter for localization is a 1-D histogram filter which estimates the vehicle's lateral offset relative to the RNDF. This filter estimates the posterior distribution of any lateral offset based on the reflectivity and the sighted curbs along the road. It "rewards," in a probabilistic fashion, offsets for which lane-marker-like reflectivity patterns align with the lane markers or the road side in the RNDF. The filter "penalizes" offsets for which an observed curb would reach into the driving corridor of the RNDF. As a result, at any point in time the vehicle estimates a fine-grained offset to the measured location by the GPS-based INS system.

Figure 10 illustrates localization relative to the RNDF in a test run. Here the green curves depicts the likely locations of lane markers in both lasers, and the yellow curve depicts the posterior distribution in the lateral direction. This specific posterior deviates from the Applanix estimate by about 80 cm, which, if not accounted for, would make Junior's wheels drive on the center line. In the Urban Challenge Event, localization offsets of 1 meter or more were common. Without this localization step, Junior would have frequently crossed the center line unintentionally, or possibly hit a curb.

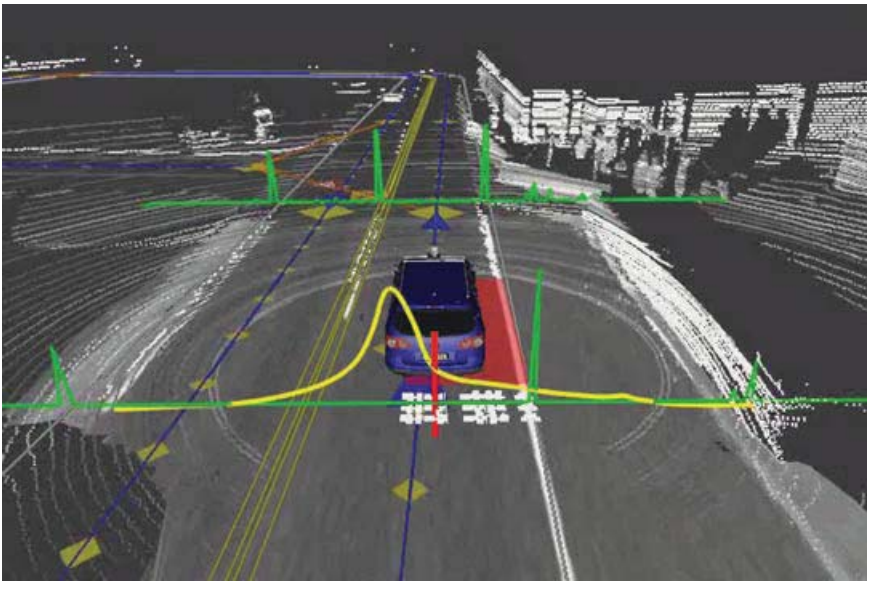


Fig. 10. Typical localization result: The red bar illustrates the Applanix localization, whereas the yellow curve measures the posterior over the lateral position of the vehicle. The green line depicts the response from the lane line detector. In this case, the error is approximately 80 cm.

Finally, Figure 11 shows a distribution of lateral offset corrections that were applied during the Urban Challenge.

5.1 Smooth Coordinates

When integrating multiple sensor measurements over time, it may be tempting to use the INS pose estimates (the output of the Applanix) to calculate the relative offset between different measurements. However, in any precision INS system, the estimated position frequently "jumps" in response to GPS measurements. This is because INS systems provide the *most likely* position at the present time. As new GPS information arrives, it is possible that the most likely position changes by an amount inconsistent with the vehicle motion. The problem, then, is that when such a revision occurs, past INS measurements have to be corrected as well, to yield a consistent map. Such a problem is known in the estimation literature as (backwards) smoothing [Jazwinsky, 1970].

To alleviate this problem, Junior maintains an internal *smooth* coordinate system that is robust to such jumps. In the smooth coordinate system, the robot position is defined as the sum of all incremental velocity updates:

$$\bar{x} = x_0 + \sum_t \Delta t \cdot \dot{x}_t$$

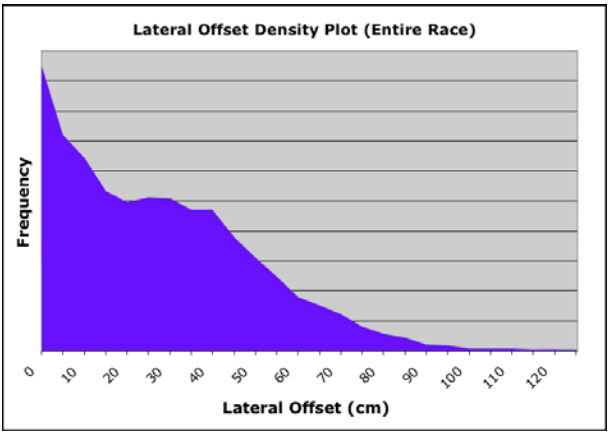


Fig. 11. Histogram of average localization corrections during the race. At times the lateral correction exceeds one meter.

where x_0 is the first INS coordinate, and $\dot{x}_t$ are the velocity estimates of the INS. In this internal coordinate system, sudden INS position jumps have no effect, and the sensor data are always locally consistent. Vehicle velocity estimates from the pose estimation system tend to be much more stable than the position estimates, even when GPS is intermittent or unavailable. X and Y velocities are particularly resistant to jumps because they are partially observed by wheel odometry.

This "trick" of smooth coordinates makes it possible to maintain locally consistent maps even when GPS shifts occur. We note, however, that the smooth coordinate system may cause inconsistencies in mapping data over long time periods, hence can only be applied to local mapping problems. This is not a problem for the present application, as the robot only maintains local maps for navigation.

In the software implementation, the mapping between raw (global) and smooth (local) coordinates only requires that one maintain the sum of all estimation shifts, which is initialized by zero. This correction term is then recursively updated by adding mismatches between actual INS coordinates and the velocity-based value.

6 Navigation

6.1 Global Path Planning

The first step of navigation pertains to global path planning. The global path planner is activated for each new checkpoint; it also is activated when a permanent road blockage leads to a change of the topology of the road network. However, instead of planning one specific path to the next checkpoint, the global path planner plans paths from every location in the map to the next checkpoint. As a result, the vehicle may depart from the optimal path and select a different one without losing direction as to where to move.

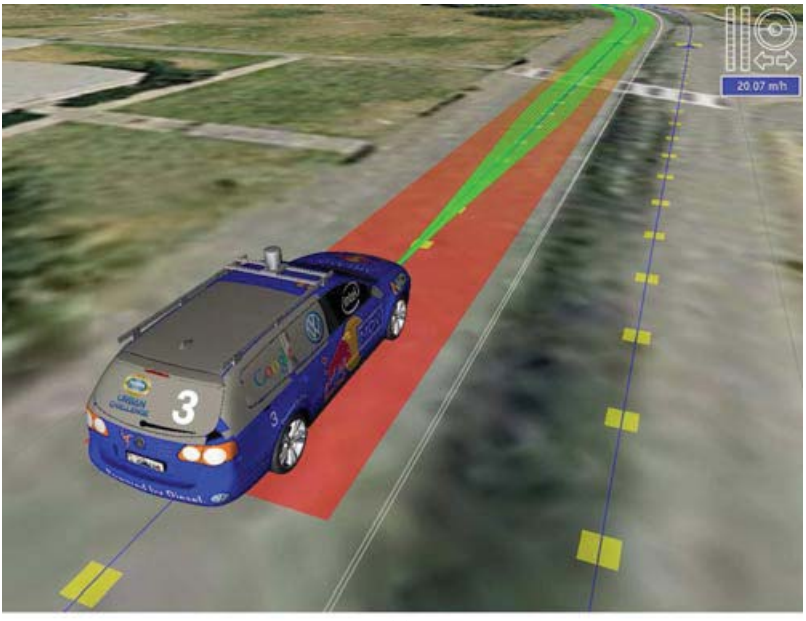
Fig. 12. Global planning: Dynamic programming propagates values through a crude discrete version of the environment map. The color of the RNDF is representative of the cost to move to the goal from each position in the graph. Low costs are green and high costs are red.

Junior's global path planner is an instance of *dynamic programming*, or DP [Howard, 1960]. The DP algorithm recursively computes for each cell in a discrete version of the RNDF the *cumulative costs* of moving from each such location to the goal point. The recursive update equation for the cost is standard in the DP literature. Let $V(x)$ be the cost of a discrete location in the RNDF, with $V(\text{goal}) = 0$. Then the following recursive equation defines the backup and, implicitly, the cumulative cost function V:

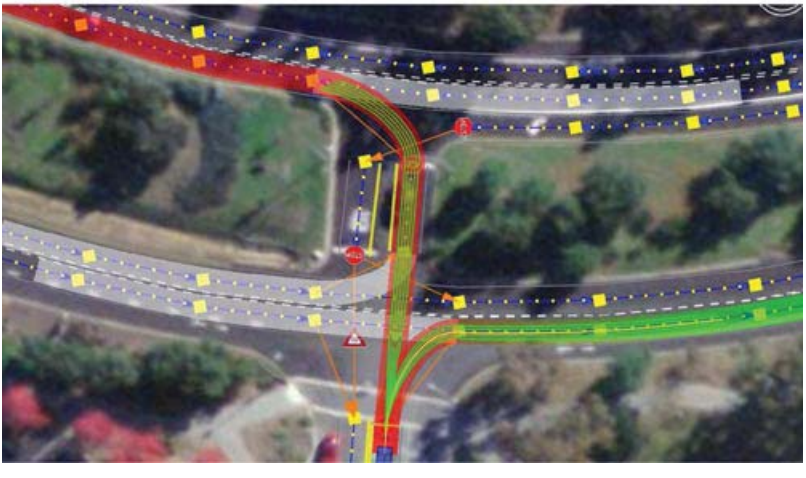
$$V(x) \longleftarrow \min_u \; c(x,u) + \sum_y p(y \mid x,u)\; V(y)$$

Here u is an action, e.g., drive along a specific road segment. In most cases, there is only one admissible action. At intersections, however, there are choices (go straight, turn left, ...). Multi-lane roads offer the choice of lane changes. For these cases the maximization over the control choice u in the expression above will provide multiple terms, the minimization of which leads to the fastest expected path.

In practice, not all action choices are always successful. For example, a shift from a left to a right lane only "succeeds" if there is no vehicle in the right lane; otherwise the vehicle cannot shift lanes. This is accommodated in the use of the transition probability $p(y \mid x,u)$. Junior, for example, might assess the success



(a)



(b)

Fig. 13. Planner roll-outs in an urban setting with multiple discrete choices. (a) For each principle path, the planner rolls out trajectories that undergo lateral shifts. (b) A driving situation with two discrete plan choices, turn right or drive straight through the intersetion. The paths are colored according to the DP value function, with red being high cost and green being low cost.

probability of a lane shift at any given discrete location as low as 10%. The benefit of this probabilistic view of decision making is that it penalizes plans that delay lane changes to the very last moment. In fact, Junior tends to execute lane shifts at the earliest possibility, and it trades off speed gains with the probability (and the cost) of failure when passing a slow moving vehicle at locations where a subsequent right turn is required (which may only be admissible when in the right lane).

A key ingredient in the recursive equation above is the cost $c(x, u)$. In most cases, the cost is simply the time it takes to move between adjacent cells in the discrete version of the RNDF. In this way, the speed limits are factored into the optimal path calculation, and the vehicle selects the path that in expectation minimizes arrival time. Certain maneuvers, such as left turns across traffic, are "penalized" by an additional time penalty to account for the risk that the robot takes when making such a choice. In this way, the cost function c implements a careful balance between navigation time and risk. So in some cases, Junior engages on a slight detour so as to avoid a risky left turn, or a risky merge. The additional costs of maneuvers can either be set by hand (as they were for the Urban Challenge) or learned from simulation data in representative environments.

Figure 12 shows a propagated cumulative cost function. Here the cumulative cost is indicated by the color of the path. This global function is brought to bear to assess the "goodness" of each location beyond the immediate sensor reach of the vehicle.

6.2 RNDF Road Navigation

The actual vehicle navigation is handled differently for common road navigation and the free-style navigation necessary for parking lots.

Figure 13 visualizes a typical situation. For each principal path, the planner rolls out a trajectory that is parallel to the smoothed center of the lane. This smoothed lane center is directly computed from the RNDF. However, the planner also rolls out trajectories that undergo lateral shifts. Each of those trajectories is the result of an internal vehicle simulation with different steering parameters. The score of a trajectory considers the time it will take to follow this path (which may be infinite if a path is blocked by an obstacle), plus the cumulative cost computed by the global path planner, for the final point along the trajectory. The planner then selects the trajectory which minimizes this total cost value. In doing so, the robot combines optimal route selection with dynamic nudging around local obstacles.

Figure 14 illustrates this decision process in a situation where a slow-moving vehicle blocks the right lane. Even though lane changes come with a small penalty cost, the time savings due to faster travel on the left lane result in a lane change. The planner then steers the robot back into the right lane when the passing maneuver is complete.

We find that this path planner works well in well-defined traffic situations. It results in smooth motion along unobstructed roads, and in smooth and well-defined passing maneuvers. The planner also enables Junior to avoid small obstacles that might extend into a lane, such as parked cars on the side. However, it is unable to handle blocked roads or intersections, and it also is unable to navigate parking lots.

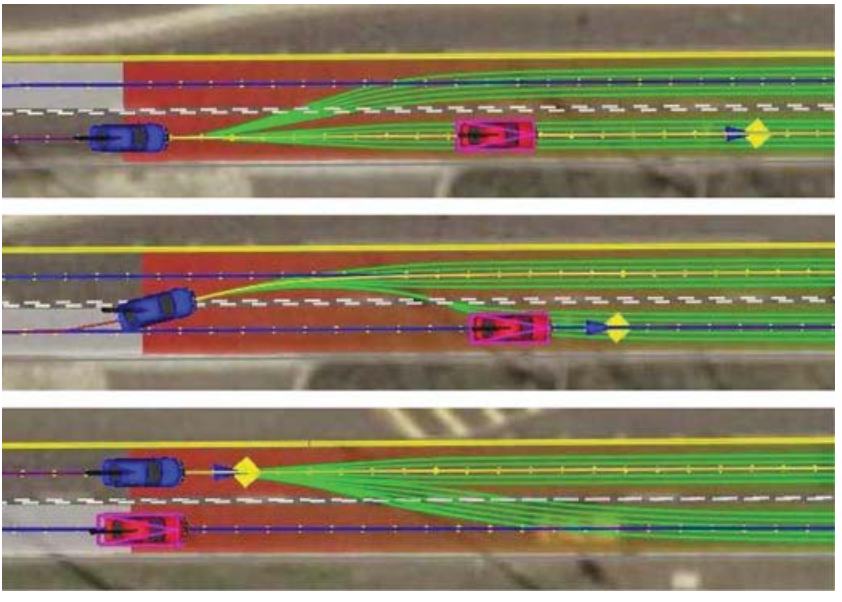


Fig. 14. A passing maneuver. The additional cost of being in a slightly sub-optimal lane is overwhelmed by the cost of driving behind a slow driver, causing Junior to change lanes and pass.

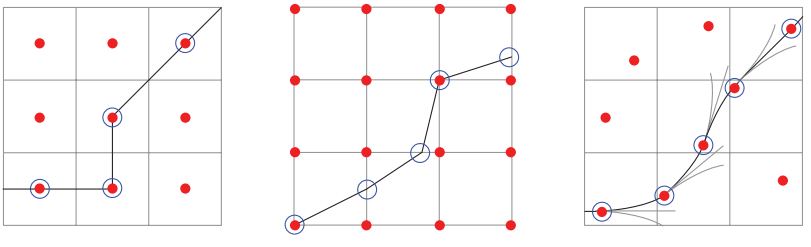


Fig. 15. Graphical comparison of search algorithms. Left: A* associates costs with centers of cells and only visits states that correspond to grid-cell centers. Center: Field D* [Ferguson and Stentz, 2005] associates costs with cell corners and allows arbitrary linear paths from cell to cell. Right: Hybrid A* associates a continuous state with each cell and the score of the cell is the cost of its associated continuous state.

6.3 Free-Form Navigation

For free-form navigation in parking lots, the robot utilizes a second planner, which can generate arbitrary trajectories irrespective of a specific road structure. This planner requires a goal coordinate and a map. It identifies a near-cost optimal path to the goal should such a path exist.

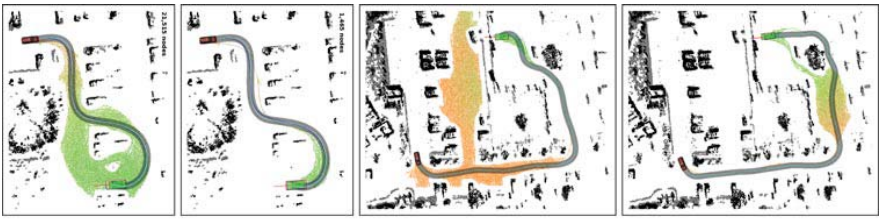


Fig. 16. Hybrid-state A* heuristics. (a) Euclidean distance in 2-D expands 21,515 nodes. (b) The non-holonomic-without-obstacles heuristic is a significant improvement, as it expands $1,465$ nodes, but as shown in (c), it can lead to wasteful exploration of dead-ends in more complex settings (68,730 nodes). (d) This is rectified by using the latter in conjunction with the holonomic-with-obstacles heuristic (10,588 nodes).

This free-form planner is a modified version of A*, which we call *hybrid A**. In the present application, hybrid A* represents the vehicle state in a 4-D discrete grid. Two of those dimensions represent the x-y-location of the vehicle center in smooth map coordinates; a third the vehicle heading direction θ, and a forth dimension pertains the direction of motion, either forward or reverse.

Fig. 17. Path smoothing with Conjugate Gradient. This smoother uses a vehicle model to guarantee that the resulting paths are attainable. The Hybrid A* path is shown in black. The smoothed path is shown in blue (front axle) and cyan (rear axle). The optimized path is much smoother than the Hybrid A* path, and can thus be driven faster.

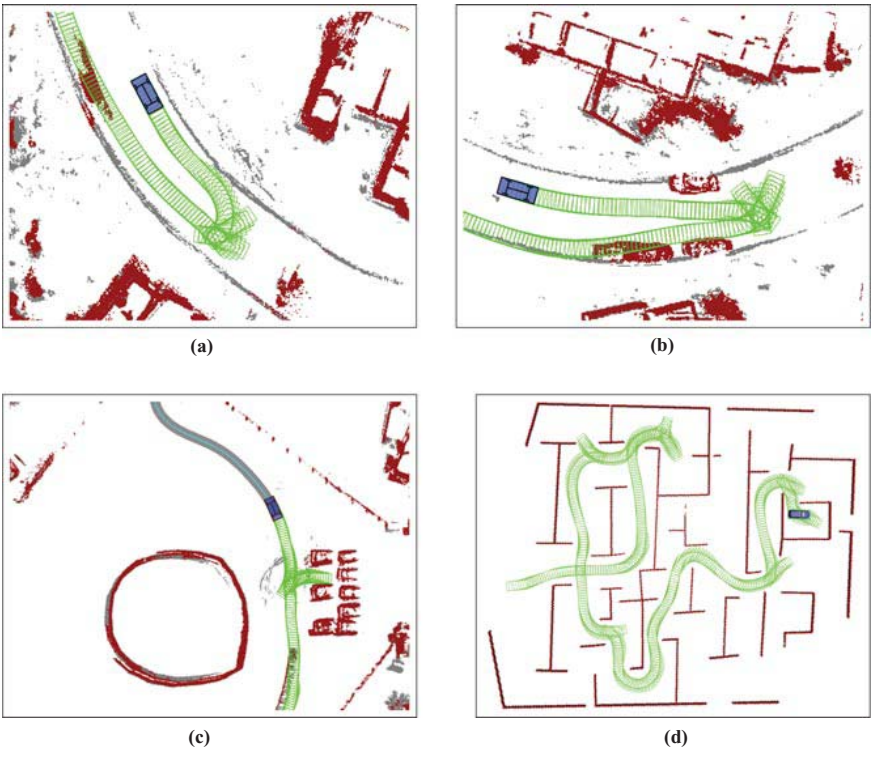


Fig. 18. Examples of trajectories generated by Junior's hybrid A* planner. Trajectories in (a)–(c) were driven by Junior in the DARPA Urban challenge: (a),(b) show U-turns on blocked roads, (c) shows a parking task. The path in (d) was generated in simulation for a more complex maze-like environment. Note that in all cases the robot had to replan in response to obstacles being detected by its sensors. In particular, this explains the sub-optimality of the trajectory in (d).

One problem with regular (non-hybrid) A* is that the resulting discrete plan cannot be executed by a vehicle, simply because the world is continuous, whereas A* states are discrete. To remedy this problem, hybrid A* assigns to each discrete cell in A* a continuous vehicle coordinate. This continuous coordinate is such that it can be realized by the actual robot.

To see how this works, let $\langle x, y, \theta \rangle$ be the present coordinates of the robot, and suppose those coordinates lie in cell c_i in the discrete A* state representation. Then, by definition, the continuous coordinates associated with cell c_i are $x_i = x$, $y_i = y$, and $\theta_i = \theta$. Now predict the (continuous) vehicle state after applying a control u for a given amount of time. Suppose the prediction is $\langle x', y', \theta' \rangle$, and assume this prediction falls into a different cell, denoted c_j. Then, if this is the first time c_j has been expanded, this cell will be assigned the associated continuous coordinates $x_j = x'$, $y_j = y'$, and $\theta_j = \theta'$. The result of this assignment is that there exists

an actual control u in which the continuous coordinates associated with cell c_j can actually be attained—a guarantee which is not available for conventional A*. The hybrid A* algorithm then applies the same logic for future cell expansions, using $\langle x_j, y_j, \theta_j \rangle$ whenever making a prediction that starts in cell c_j. We note that hybrid A* is guaranteed to yield realizable paths, but it is not complete. That is, it may fail to find a path. The coarser the discretization, the more often hybrid A* will fail to find a path.

Figure 15 compares hybrid A* to regular A* and Field D* [Ferguson and Stentz, 2005], an alternative algorithm that also considers the continuous nature of the underlying state space. A path found by plain A* cannot easily be executed; and even the much smoother Field D* path possesses kinks that a vehicle cannot execute. By virtue of associating continuous coordinates with each grid cell in Hybrid A*, our approach results in a path that is executable.

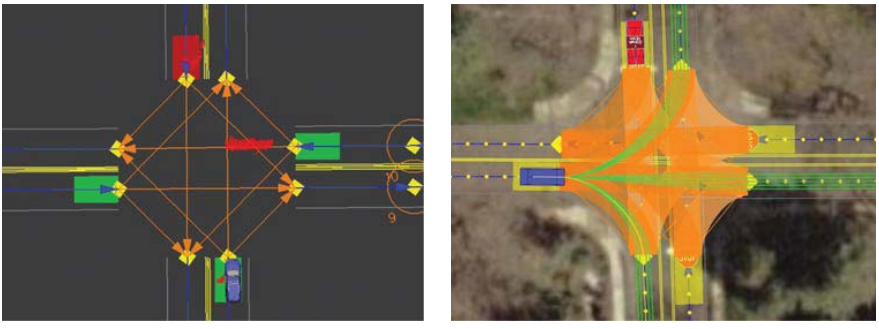
The cost function in A* follows the idea of execution time. Our implementation assigns a slightly higher cost to reverse driving to encourage the vehicle to drive "normally." Further, a change of direction induces an additional cost to account for the time it takes to execute such a maneuver. Finally, we add a pseudo-cost that relates to the distance to nearby obstacles so as to encourage the vehicle to stay clear of obstacles.

Our search algorithm is guided by two heuristics, called the *non-holonomic-without-obstacles heuristic* and the *holonomic-with-obstacles heuristic*. As the name suggests, the first heuristic ignores obstacles but takes into account the non-holonomic nature of the car. This heuristic, which can be completely precomputed for the entire 4D space (vehicle location, and orientation, and direction of motion), helps in the end-game by approaching the goal with the desired heading. The second heuristic is a dual of the first in that it ignores the non-holonomic nature of the car, but computes the shortest distance to the goal. It is calculated online by performing dynamic programming in 2-D (ignoring vehicle orientation and motion direction). Both heuristics are admissible, so the maximum of the two can be used.

Figure 16a illustrates A* planning using the commonly used Euclidean distance heuristic. As shown in Figure 16b, the non-holonomic-without-obstacles heuristic is significantly more efficient than Euclidean distance, since it takes into account vehicle orientation. However, as shown in Figure 16c, this heuristic alone fails in situations with U-shaped dead ends. By adding the holonomic-with-obstacles heuristic, the resulting planner is highly efficient, as illustrated in Figure 16d.

While hybrid A* paths are realizable by the vehicle, the small number of discrete actions available to the planner often lead to trajectories with rapid changes in steering angles, which may still lead to trajectories that require excessive steering. In a final post-processing stage, the path is further smoothed by a Conjugate Gradient smoother that optimizes similar criteria as hybrid A*. This smoother modifies controls and moves waypoints locally. In the optimization, we also optimize for minimal steering wheel motion and minimum curvature. Figure 17 shows the result of smoothing.

The hybrid A* planner is used for parking lots and also for certain traffic maneuvers, such as U-turns. Figure 18 shows examples from the Urban Challenge and the



(a) (b)

Fig. 19. Critical zones: (a) At this four-way stop sign, busy critical zones are colored in red, whereas critical zones without vehicles are shown in green. In this image, a vehicle can be seen driving through the intersection from the right. (b) Critical zones for merging into an intersection.

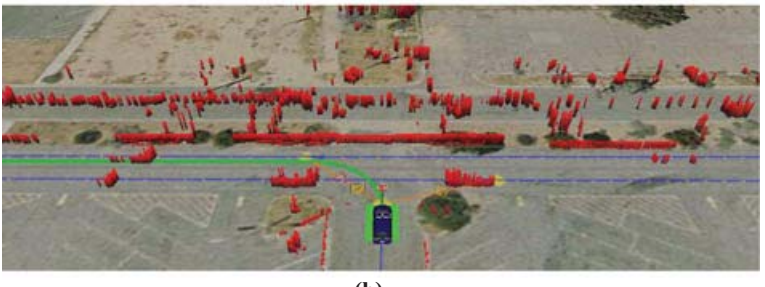
associated National Qualification Event. Shown there are two successful U-turns and one parking maneuver. The example in Figure 18d is based on a simulation of a more complex parking lot. The apparent suboptimality of the path is the result of the fact that the robot "discovers" the map as it explores the environment, forcing it into multiple backups as a previously believed free path is found to be occupied. All of those runs involve repetitive executions of the hybrid A* algorithm, which take place while the vehicle is in motion. When executed on a single core of Junior's computers, planning from scratch requires up to 100 milliseconds; in the Urban Challenge, planning was substantially faster because of the lack of obstacles in parking lots.

6.4 Intersections and Merges

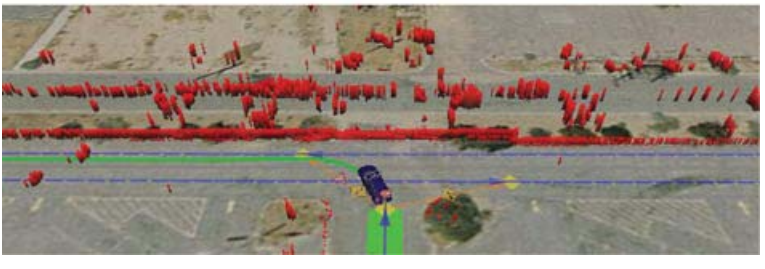
Intersections are places that require discrete choices not covered by the basic navigation modules. For example, at multi-way intersections with stop signs, vehicles may only proceed through the intersection in the order of their arrival.

Junior keeps track of specific "critical zones" at intersections. For multi-way intersections with stop signs, such critical zones correspond to regions near each stop sign. If such a zone is occupied by a vehicle at the time the robot arrives, Junior waits until this zone has cleared (or a timeout has occurred). Intersection critical zones are shown in Figure 19. In merging, the critical zones correspond to segments of roads where Junior may have to give precedence to moving traffic. If an object is found in such a zone, Junior uses its radars and its vehicle tracker to determine the velocity of moving objects. Based on the velocity and proximity, a threshold test then marks the zone in question as busy, which then results in Junior waiting at a merge point. The calculation of critical zones is somewhat involved. However, all computations are performed automatically based on the RNDF, and ahead of the actual vehicle operation.

(a)



(b)



(c)

Fig. 20. Merging into dense traffic during the qualification events at the Urban Challenge. (a) Photo of merging test; (b)-(c) The merging process.

Figure 20 visualizes a merging process during the qualification event to the Urban Challenge. This test involves merging into a busy lane with 4 human-driven vehicles, and across another lane with 7 human-driven cars. The robot waits until none of the critical zones are busy, and then pulls into the moving traffic. In this example, the vehicle was able to pull safely into 8 second gaps in two-way traffic.

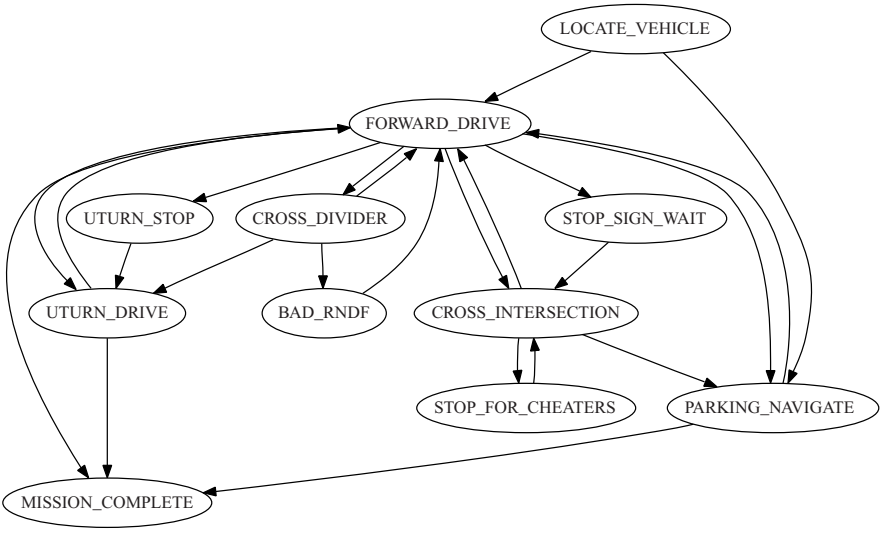
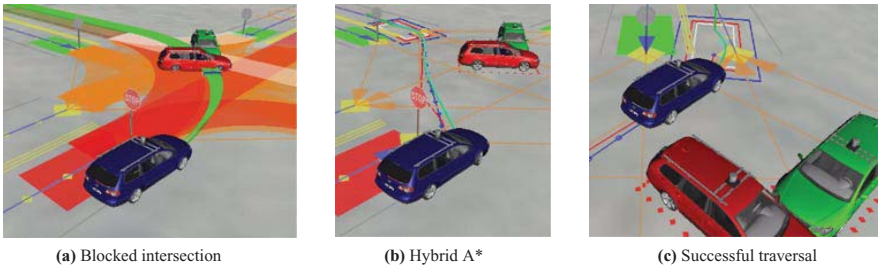


Fig. 21. Finite State Machine that governs the robot's behavior.



(a) Blocked intersection **(b)** Hybrid A* **(c)** Successful traversal

Fig. 22. Navigating a simulated traffic jam: After a timeout period, the robot resorts to hybrid A* to find a feasible path across the intersection.

6.5 Behavior Hierarchy

An essential aspect of the control software is logic that prevents the robot from getting stuck. Junior's *stuckness detector* is triggered in two ways: through timeouts when the vehicle is waiting for an impasse to clear, and through the repeated traversal of a location in the map—which may indicate that the vehicle is looping indefinitely.

Figure 21 shows the finite state machine (FSM) that is used to switch between different driving states, and that invokes exceptions to overcome stuckness. This FSM possesses 13 states (of which 11 are shown; 2 are omitted for clarity). The individual states in this FSM correspond to the following conditions:

- LOCATE_VEHICLE: This is the initial state of the vehicle. Before it can start driving, the robot estimates its initial position on the RNDF, and starts road driving or parking lot navigation, whichever is appropriate.
- FORWARD_DRIVE: This state corresponds to forward driving, lane keeping and obstacle avoidance. When not in a parking lot, this is the preferred navigation state.
- STOP_SIGN_WAIT: This state is invoked when the robot waits at at a stop sign to handle intersection precedence.
- CROSS_INTERSECTION: Here the robot waits if it is safe to cross an intersection (e.g., during merging), or until the intersection is clear (if it is an all-way stop intersection). The state also handles driving until Junior has exited the intersection.
- STOP_FOR_CHEATERS: This state enables Junior to wait for another car moving out of turn at a four way intersection.
- UTURN_DRIVE: This state is invoked for a U-turn.
- UTURN_STOP: Same as UTURN_DRIVE, but here the robot is stopping in preparation for a U-turn.
- CROSS_DIVIDER: This state enables Junior to cross the yellow line (after stopping and waiting for oncoming traffic) in order to avoid a partial road blockage.
- PARKING_NAVIGATE: Normal parking lot driving.
- TRAFFIC_JAM: In this sate, the robot uses the general-purpose hybrid A* planner to get around a road blockage. The planner aims to achieve any road point 20 meters away on the current robot trajectory. Use of the general-purpose planner allows the robot to engage in unrestricted motion and disregard certain traffic rules.
- ESCAPE: This state is the same as TRAFFIC_JAM, only more extreme. Here the robot aims for any waypoint on any base trajectory more than 20 meters away. This state enables the robot to choose a suboptimal route at an intersection in order to extract itself out of a jam.
- BAD_RNDF: In this state, the robot uses the hybrid A* planner to navigate a road that does not match the RNDF. It triggers on one lane, one way roads if CROSS_DIVIDER fails.
- MISSION_COMPLETE: This state is set when race is over.

For simplicity, Figure 21 omits ESCAPE and TRAFFIC_JAM. Nearly all states have transitions to ESCAPE and TRAFFIC_JAM.

At the top level, the FSM transitions between the normal driving states, such as lane keeping and parking lot navigation. Transitions to lower driving levels (exceptions) are initiated by the stuckness detectors. Most of those transition invoke a "wait period" before the corresponding exception behavior is invoked. The FSM returns to normal behavior after the successful execution of a robotic behavior.

The FSM makes the robot robust to a number of contingencies. For example:

- For a blocked lane, the vehicle considers crossing into the opposite lane. If the opposite lane is also blocked, a U-turn is initiated, the internal RNDF is modified

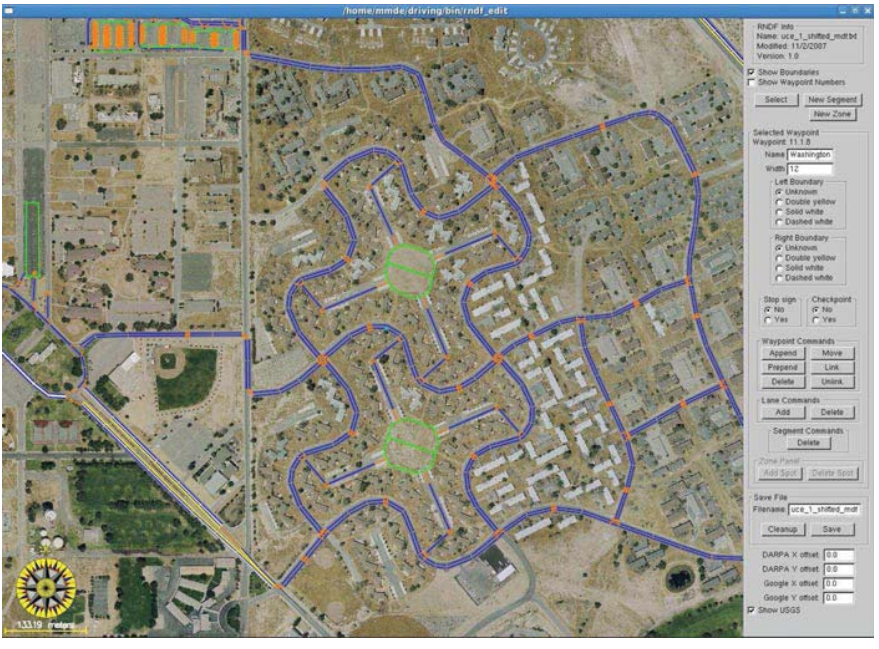
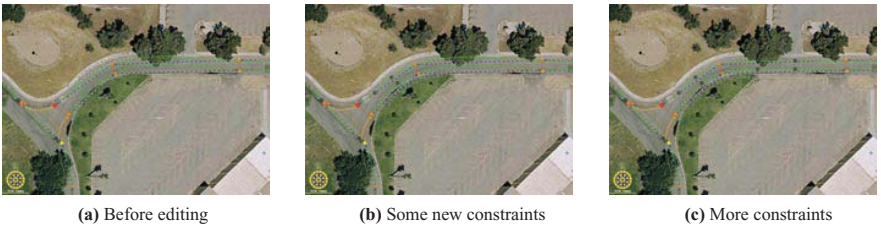


Fig. 23. RNDF editor tool.



(a) Before editing **(b)** Some new constraints **(c)** More constraints

Fig. 24. Example: Effect of adding and moving waypoints in the RNDF. Here the corridor is slightly altered to better match the aerial image. The RNDF editor permits for such alterations in an interactive manner, and displays the results on the base trajectory without any delay.

accordingly, and dynamic programming is run to regenerate the RNDF value function.

- Failure to traverse a blocked intersection is resolved by invoking the hybrid A* algorithm, to find a path to the nearest reachable exit of the intersection; see Figure 22 for an example.
- Failure to navigate a blocked one-way road results in using hybrid A* to the next GPS waypoint. This feature enables vehicles to navigate RNDFs with sparse GPS waypoints.

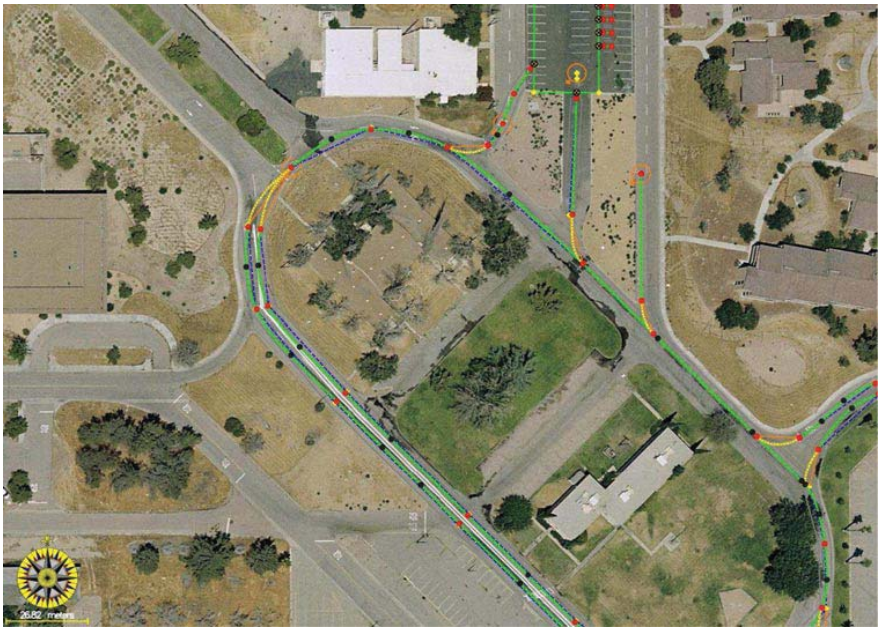


Fig. 25. The SRNDF creator produces a smooth base trajectory automatically by minimizing a set of nonlinear quadratic constraints. The original RNDF is shown in blue. The smooth SRNDF is shown in green.

- Repeated looping while attempting to reach a checkpoint results in the checkpoint being skipped, so as to not jeopardize the overall mission. This behavior avoids infinite looping if a checkpoint is unreachable.
- Failure to find a path in a parking lot with hybrid A* makes the robot temporarily erase its map. Such failures may be the result of treating as static objects that since moved away – which cannot be excluded.
- In nearly all situations, failure to make progress for extended periods of time ultimately leads to the use of hybrid A* to find a path to a nearby GPS waypoint. When this rare behavior is invoked, the robot does not obey traffic rules any longer.

In the Urban Challenge event, the robot almost never entered any of the exception states. This is largely because the race organizers repeatedly paused the robot when it was facing traffic jams. However, extensive experiments prior to the Urban Challenge showed that it was quite difficult to make the robot fail to achieve its mission, provided that the mission remained achievable.

6.6 Manual RNDF Adjustment

Ahead of the Urban Challenge event, DARPA provided teams not just with an RNDF, but also with a high-resolution aerial image of the site. While the RNDF

was produced by careful ground-based GPS measurements along the course, the aerial image was purchased from a commercial vendor and acquired by aircraft.

To maximize the accuracy of the RNDF, the team manually adjusted and augmented the DARPA-provided RNDF. Figure 23 shows a screen shot of the editor. This tool enables an editor to move, add, and delete waypoints. The RNDF editor program is fast enough to incorporate new waypoints in real time (10Hz).

The editing required three hours of a person's time. In an initial phase, waypoints were shifted manually, and roughly 400 new way points were added manually to the 629 lane waypoints in the RNDF. Those additions increased the spatial coherence of the RNDF and the aerial image. Figure 24 shows a situation in which the addition of such additional waypoint constraints leads to substantial improvements of the RNDF.

To avoid sharp turns at the transition of linear road segments, the tool provides an automated RNDF smoothing algorithm. This algorithm upsamples the RNDF at one meter intervals, and sets those as to maximize the smoothness of the resulting path. The optimization of these additional points combines a least squares distance measure with a smoothness measure. The resulting "smooth RNDF," or SRNDF, is then used instead of the original RNDF for localization and navigation. Figure 25 compares the RNDF and the SRNDF for a small fraction of the course.

7 The Urban Challenge

7.1 Results

The Urban Challenge took place Nov. 3, 2007, in Victorville, CA. Figure 26 shows images of the start and the finish of the Urban Challenge. Our robot Junior never hit an obstacle, and according to DARPA, it broke no traffic rule. A careful analysis of the race logs and official DARPA documentation revealed two situations (described below) in which Junior behaved suboptimally. However, all of those events were deemed rule conforming by the race organizers. Overall, Junior's localization and

Fig. 26. The start and the finish of the Urban Challenge. Junior arrives at the finish line.

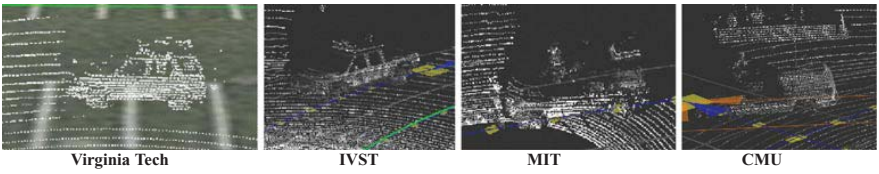


Fig. 27. Scans of other robots encountered in the race.

road following behaviors were essentially flawless. The robot never came close to hitting a curb or crossing into opposing traffic.

The event was organized in three missions, which differed in length and complexity. Our robot accomplished all three missions in 4 hours 5 minutes, and 6 seconds of run time. During this time, the robot traveled a total of 55.96 miles, or 90.068 km. Its average speed while in run mode was thus 13.7 mph. This is slower than the average speed in the 2005 Grand Challenge [Montemerlo et al., 2006, Urmson et al., 2004], but most of the slowdown was caused by speed limits, traffic regulations (e.g., stop signs), and other traffic. The total time from the start to the final arrival was 5 hours, 23 minutes, and 2 seconds, which includes all pause times. Thus, Junior was paused for a total of 1 hour, 17 minutes and 56 seconds. None of those pauses were caused by Junior, or requested by our team. An estimated 26 minutes and 27 seconds were "local" pauses, in which Junior was paused by the organizers because other vehicles were stuck. Our robot was paused six times because other robots encountered problems on the off-road section, or were involved in an accident. The longest local pause (10 min, 15 sec) occurred when Junior had to wait behind a two-robot accident. Because of DARPA's decision to pause robots, Junior could not exercise its hybrid A* planner in these situations. DARPA determined Junior's adjusted total time to be 4 hours, 29 minutes, and 28 seconds. Junior was judged to be the second fastest finishing robot in this event.

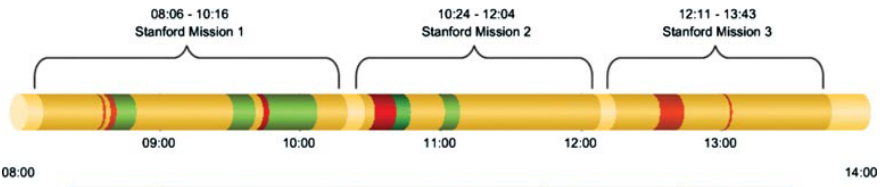
7.2 Notable Race Events

Figure 27 shows scans of other robots encountered in the race. Overall, DARPA officials estimate that Junior faced approximately 200 other vehicles during the race. The large number of robot-robot encounters was a unique feature of the Urban Challenge.

There were several notable encounters during the race in which Junior exhibited particularly intelligent driving behavior, as well as two incidents where Junior made clearly suboptimal decisions (neither of which violated any traffic rules).

Hybrid A* on the Dirt Road

While the majority of the course was paved, urban terrain, the robots were required to traverse a short off-road section connecting the urban road network to a 30mph highway section. The off-road terrain was graded dirt path with a non-trivial elevation change, reminiscent of the 2005 DARPA Grand Challenge course. This section caused problems for several of the robots in the competition. Junior traveled down



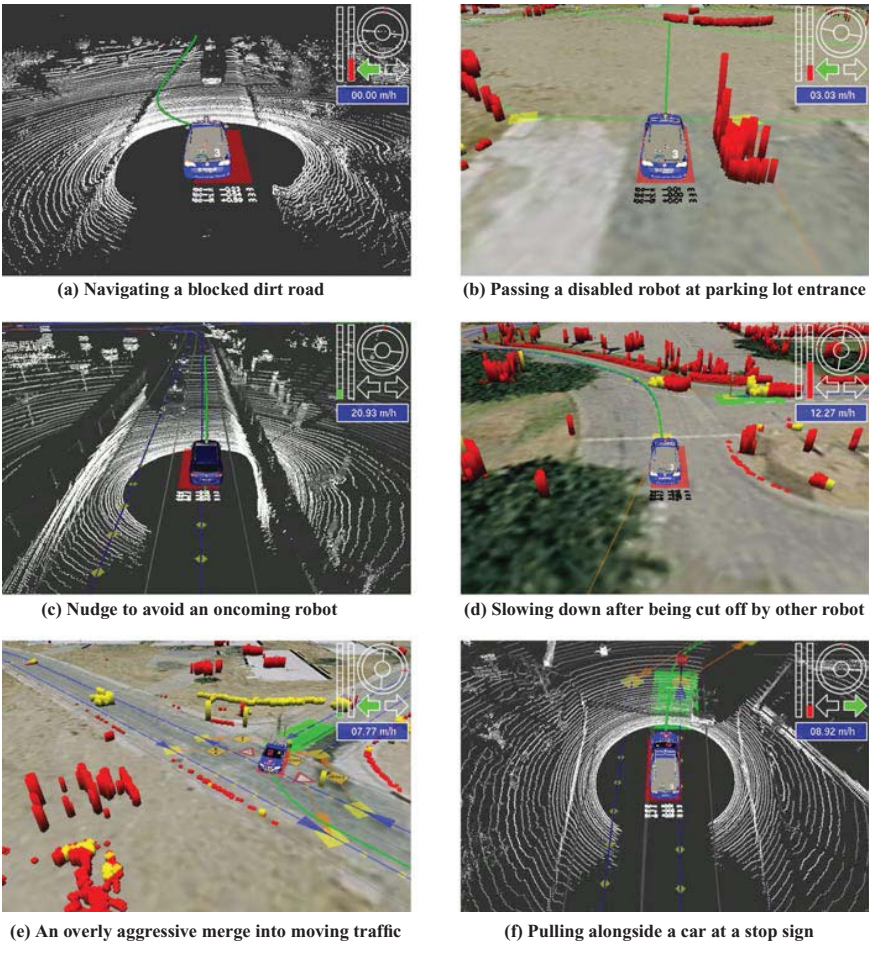
Pause Type	Why	When	Duration
Local	Little Ben: Problems on dirt road	08:32:23 - 08:33:52	0:01:08
Local	Little Ben: Did not leave the dirt road	08:35:28 - 08:37:58	0:02:30
All	Little Ben: Rescue team repositioned the car on Phantom East	08:37:58 - 08:46:30	0:08:32
All	AnnieWay, IVS, Terramax: car got removed	09:27:52 - 09:37:20	0:09:28
Local	Little Ben: Accident with UCF	09:40:42 – 09:43:12	0:02:30
All	Little Ben: Rescue Team	09:43:12 - 10:03:50	0:20:38
Local	Cornell: Stopped without reason, moved on, stopped, ...	10:28:26 - 10:37:30	0:09:04
All	UCF: car got removed	10:37:30 – 10:43:44	0:06:14
All	Carolo: Accident with MIT, car got removed	10:58:11 - 11:04:46	0:06:35
Local	Cornell: Accident with MIT	12:30:11 - 12:40:27	0:10:15
Local	Little Ben: using same parking area	13:00:00 - 13:01:00	0:01:00

Fig. 28. Junior mission times during the Urban Challenge. Times marked green correspond to local pauses, and times in red to all-pauses, in which all vehicles were paused.

the dirt road during the first mission, immediately behind another robot and its chase car. While Junior had no difficulty following the dirt road, the robot in front of Junior stopped three times for extended periods of time. In response to the first stop, Junior also stopped and waited behind the robot and its chase car. After seeing no movement for a period of time, Junior activated several of its recovery behaviors. First, Junior considered CROSS_DIVIDER, a preset passing maneuver to the left of the two stopped cars. There was not sufficient space to fit between the cars and the berm on the side of the road, so Junior then switched to the BAD_RNDF behavior, in which the Hybrid A* planner is used to plan an arbitrary path to the next DARPA waypoint. Unfortunately, there was not enough space to get around the cars even with the general path planner. Junior repeatedly repositioned himself on the road in an attempt to find a free path to the next waypoint, until the cars started moving again. Junior repeated this behavior when the preceding robot stopped a second time, but was paused by DARPA until the first robot recovered. Figure 29a shows data and a CROSS_DIVIDER path around the preceding vehicle on the dirt road.

Passing Disabled Robot

The course included several free-form navigation zones where the robots were required to navigate around arbitrary obstacles and park in parking spots. As Junior approached one of these zones during the first mission, it encountered another robot which had become disabled at the entrance to the zone. Junior queued up behind the robot, waiting for it to enter the zone. After the robot did not move for a given amount of time, Junior passed it slowly on the left using the CROSS_DIVIDER behavior. Once Junior had cleared the disabled vehicle, the Hybrid A* planner was



(a) Navigating a blocked dirt road

(b) Passing a disabled robot at parking lot entrance

(c) Nudge to avoid an oncoming robot

(d) Slowing down after being cut off by other robot

(e) An overly aggressive merge into moving traffic

(f) Pulling alongside a car at a stop sign

Fig. 29. Key moments in the Urban Challenge race.

enabled to navigate successfully through the zone. Figure 29b shows this passing maneuver.

Avoiding Opposing Traffic

During the first mission, Junior was traveling down a two-way road and encountered another robot in the opposing lane of traffic. The other robot was driving such that its left wheels were approximately one foot over the yellow line, protruding into oncoming traffic. Junior sensed the oncoming vehicle and quickly nudged the right side of its lane, where it then passed at full speed without incident. This situation is depicted in Figure 29c.

Reacting to an Aggressive Merge
During the third mission, Junior was traveling around a large traffic circle which featured prominently in the competition. Another robot was stopped at a stop sign waiting to enter the traffic circle. The other robot pulled out aggressively in front of Junior, who was traveling approximately 15mph at the time. Junior braked hard to slow down for the other robot, and continued with its mission. Figure 29d depicts the situation during this merge.

Junior Merges Aggressively
Junior merged into moving traffic successfully on numerous occasions during the race. On one occasion during the first mission, however, Junior turned left from a stop sign in front of a robot that was moving at 20mph with an uncomfortably small gap. Data from this merge is shown in Figure 29e. The merge was aggressive enough that the chase car drivers paused the other vehicle. Later analysis revealed that Junior saw the oncoming vehicle, yet believed there was a sufficient distance to merge safely. Our team had previously lowered merging distance thresholds to compensate for overly conservative behavior during the qualification event. In retrospect, these thresholds were set too low for higher speed merging situations. While this merge was definitely suboptimal behavior, it was later judged not be a violation of the rules by DARPA.

Pulling Alongside a Waiting Car
During the second mission, Junior pulled up behind a robot waiting at a stop sign. The lane was quite wide, and the other robot was offset towards the right side of the lane. Junior, on the other hand, was traveling down the left side of the lane. When pulling forward, Junior did not register the other car as being inside the lane of travel, and thus began to pull alongside of the car waiting at the stop sign. As Junior tried to pass, the other car pulled forward from the stop sign and left the area. This incident highlights how difficult it can be for a robot to distinguish between a car stopped at a stop sign and a car parked on the side of the road. See Figure 29f.

8 Discussion

This paper described a robot designed for urban driving. Stanford's robot Junior integrates a number of recent innovations in mobile robotics, such as probabilistic localization, mapping, tracking, global and local planning, and an FSM for making the robot robust to unexpected situations. The results of the Urban Challenge, along with prior experiments carried out by the research team, suggest that the robot is capable of navigating in other robotic and human traffic. The robot successfully demonstrated merging, intersection handling, parking lot navigation, lane changes, and autonomous U-turns.

The approach presented here features a number of innovations, which are well-grounded in past research on autonomous driving and mobile robotics. These innovations include the obstacle/curb detection method, the vehicle tracker, the various motion planners, and the behavioral hierarchy that addresses a broad range of traffic

situations. Together, these methods provide for a robust system for urban in-traffic autonomous navigation.

Still, a number of advances are required for truly autonomous urban driving. The present robot is unable to handle traffic lights. No experiments have been performed with a more diverse set of traffic participants, such as bicycles and pedestrians. Finally, DARPA frequently paused robots in the Urban Challenge to clear up traffic jams. In real urban traffic, such interventions are not realistic. It is unclear if the present robot (or other robots in this event!) would have acted sensibly in lasting traffic congestion.

References

Buehler et al., 2006. Buehler, M., Iagnemma, K., Singh, S. (eds.): The 2005 DARPA Grand Challenge: The Great Robot Race. Springer, Berlin (2006)

DARPA, 2007. DARPA, Urban challenge rules (2007), http://www.darpa.mil/grandchallenge/rules.asp (revision october 27, 2007)

Ferguson and Stentz, 2005. Ferguson, D., Stentz, A.: Field D*: An interpolation-based path planner and replanner. In: Proceedings of the 12th International Symposium of Robotics Research (ISRR 2005), San Francisco, CA. Springer, Heidelberg (2005)

Howard, 1960. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press and Wiley (1960)

Jazwinsky, 1970. Jazwinsky, A.: Stochastic Processes and Filtering Theory. Academic, New York (1970)

Montemerlo et al., 2006. Montemerlo, M., Thrun, S., Dahlkamp, H., Stavens, D., Strohband, S.: Winning the DARPA Grand Challenge with an AI robot. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Boston, MA. AAAI, Menlo Park (2006)

Moravec, 1988. Moravec, H.P.: Sensor fusion in certainty grids for mobile robots. AI Magazine 9(2), 61–74 (1988)

Simmons and Apfelbaum, 1998. Simmons, R., Apfelbaum, D.: A task description language for robot control. In: Proceedings of the Conference on Intelligent Robots and Systems (IROS), Victoria, CA (1998)

Urmson et al., 2004. Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J., Gowdy, J., Gutierrez, A., Harbaugh, S., Johnson-Roberson, M., Kato, H., Koon, P., Peterson, K., Smith, B., Spiker, S., Tryzelaar, E., Whittaker, W.: High speed navigation of unrehearsed terrain: Red Team technology for the Grand Challenge, Technical Report CMU-RI-TR-04-37, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (2004)

U.S. Department of Transportation, 2005. U.S. Department of Transportation, B. o. T. S. (2005); Transportation statistics annual report

Odin: Team VictorTango's Entry in the DARPA Urban Challenge

Charles Reinholtz, Dennis Hong[2], Al Wicks[2],
Andrew Bacha[3], Cheryl Bauman[3], Ruel Faruque[3], Michael Fleming[3],
Chris Terwelp[3], Thomas Alberi[4], David Anderson[4], Stephen Cacciola[4],
Patrick Currier[4], Aaron Dalton[4], Jesse Farmer[4], Jesse Hurdus[4], Shawn Kimmel[4],
Peter King[4], Andrew Taylor[4], David Van Covern[4], and Mike Webster[4]

[1] Department of Mechanical Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114
charles.reinholtz@erau.edu
[2] Department of Mechanical Engineering
Virginia Tech
Blacksburg, VA 24060
[3] TORC Technologies, LLC
Blacksburg, VA 24060
[4] Unmanned Systems Group
Virginia Tech
Blacksburg, VA 24060

Abstract. The DARPA Urban Challenge required robotic vehicles to travel over 90km through an urban environment without human intervention and included situations such as stop intersections, traffic merges, parking, and road blocks. Team VictorTango separated the problem into three parts: base vehicle, perception, and planning. A Ford Escape outfitted with a custom drive-by-wire system and computers formed the basis for Odin. Perception used laser scanners, GPS, and a priori knowledge to identify obstacles, cars, and roads. Planning relied on a hybrid deliberative/reactive architecture to analyze the situation, select the appropriate behavior, and plan a safe path. All vehicle modules communicated using the JAUS standard. The performance of these components in the Urban Challenge is discussed and successes noted. The result of VictorTango's work was successful completion of the Urban Challenge and a third place finish.

1 Introduction

On November 3rd, 2007, DARPA hosted the Urban Challenge, an autonomous ground vehicle competition in an urban environment. To meet this challenge, Virginia Tech and TORC Technologies formed team VictorTango, a collaborative effort between academia and industry. The team includes 46 undergraduate students, 8 graduate students, 4 faculty members, 5 full time TORC employees and industry partners, including Ford Motor Co. and Caterpillar, Inc. Together

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 125–162.
springerlink.com

team VictorTango and its partners developed Odin, a 2005 Ford Hybrid Escape modified for autonomous operation.

In the weeks prior to competition, 35 teams prepared for the National Qualifying Event (NQE). Vehicles had to navigate various courses, merge with traffic, navigate cluttered roads and zones, park in full parking lots, and detect road blocks. After a rigorous qualifying event, only 11 teams were deemed ready by DARPA to line up in the start chutes of the final Urban Challenge Event (UCE). The vehicles had to navigate similar situations to those they encountered during the NQE. However, each vehicle also had to share the road with the other 10 autonomous vehicles, 10 chase vehicles, and 50 human-driven traffic vehicles. Six of the eleven vehicles finished the race. This paper provides a summary of the approach, final configurations, successes, and incidents of the third place team, VictorTango.

1.1 VictorTango Overview

Team VictorTango divided the problem posed by the Urban Challenge into three major parts: base vehicle platform, perception, and planning. Each of these sections was then subdivided into distinct components for parallel development. Team members were able to split up the required tasks, execute and debug them individually, and provide finished components for full system testing. This modular approach provided the rapid development time needed to complete a project of such magnitude in only 14 months. This section provides a description of the components that constitute the team's approach.

1.2 Base Vehicle Platform

Team VictorTango's entry in the Urban Challenge is a modified 2005 Hybrid Ford Escape named Odin, shown in Figure 1. This base vehicle platform meets the DARPA requirement of a midsize commercial automobile with a proven safety record. The use of the hybrid-electric Ford Escape provides numerous advantages in the areas of on-board power generation, reliability, safety and autonomous operation. As required by DARPA, the drive-by-wire conversion does not bypass any of the OEM safety systems. Since the stock steering, shifting and throttle systems on the Hybrid Escape are already drive-by-wire, these systems can be controlled electronically by emulating the command signals, eliminating the complexity and failure potential associated with the addition of external actuators. The stock hybrid power system is able to provide sufficient power for sensors and computers without the need for a separate generator.

Odin's main computing is supplied by a pair of Hewlett-Packard servers each of which are equipped with two quad-core processors. One of the servers runs Microsoft Windows XP and is dedicated to sensor processing. Windows was selected since some of the sensor processing software uses National Instruments' LabVIEW Vision development module, requiring Windows. The other server runs Linux and is further subdivided into four virtual machines for process load balancing and isolation. The Linux system, selected for its configurability and

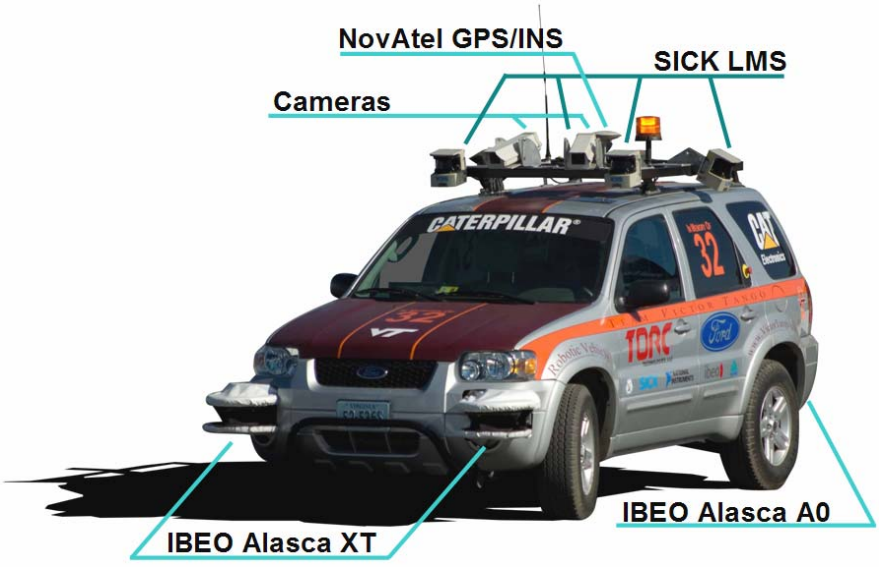


Fig. 1. External view of Odin with sensors labeled

stability, runs all of the decision making and planning modules. The vehicle hardware is controlled by a National Instruments CompactRIO unit, which contains a real-time capable OS and an FPGA. The primary communications backbone is provided by a gigabit Ethernet network.

1.3 Perception

To fulfill the behavioral requirements of the Urban Challenge, Odin must first be able to adequately localize its position and perceive the surrounding environment. Since there may be sparse waypoints in an RNDF and areas of poor GPS coverage, the surrounding road coverage and legal lanes of travel must also be sensed. Finally, Odin must be able to perceive all obstacles in its path and appropriately classify obstacles as vehicles.

For each perception requirement, multiple sensors are desirable to achieve the highest levels of fidelity and reliability. To allow for maximum flexibility in sensor fusion, the planning software does not use any raw sensor data; rather it uses a set of sensor-independent perception messages. The perception components and the resulting messages are shown in Figure 2. The Localization component determines the vehicle position and orientation in the world. The Road Detection component determines a road coverage map as well as the position of each lane in nearby segments. The Object Classification component detects obstacles and classifies them as either static or dynamic. A dynamic obstacle is any obstacle that is capable of movement, so a stopped vehicle would be classified as a dynamic obstacle with zero forward velocity.

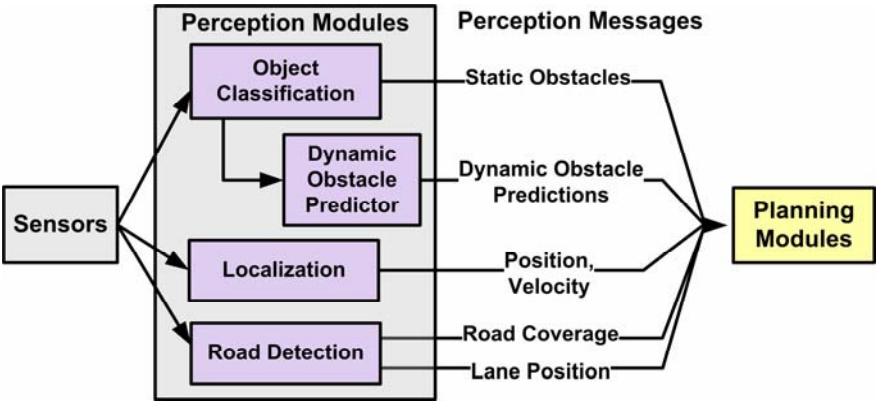


Fig. 2. Perception structure overview

1.4 Planning

The planning software on Odin uses a Hybrid Deliberative-Reactive model dividing upper level decisions and lower level reactions into separate components. These components run concurrently at independent rates, allowing the vehicle to react to emergency situations without needing to re-plan an entire route. Splitting the decision making into separate components also allows each system to be tested independently and fosters parallel development, which is especially attractive given the short development timeline of the DARPA Urban Challenge.

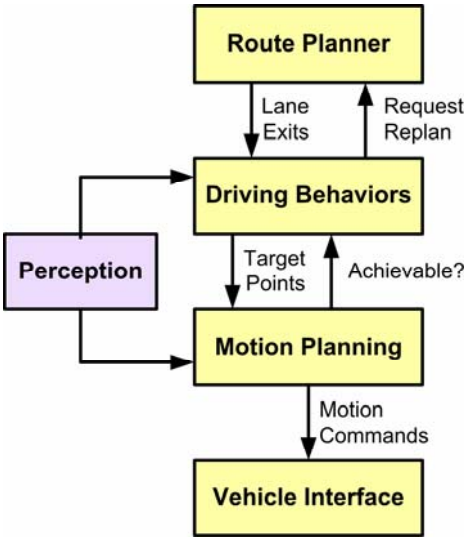


Fig. 3. Planning structure overview

The Route Planner component is the coarsest level of planning and is responsible for determining which road segments and zones the vehicle should use to travel to all checkpoints. The Driving Behaviors component is responsible for obeying the rules of the road and guiding the vehicle along the planned route. The lowest level of the planning process is the Motion Planning component, which determines the path and speed of Odin. Motion commands are then passed to the Vehicle Interface to be translated into actuator control signals. An overview of the planning process is shown in Figure 3.

2 Technical Approach

This section presents an overview of the major design choices made in the development of Odin, focusing on perception and planning systems. In each of these sections, an overview of the system function is given as well as the design of key elements.

2.1 System Architecture and Communications

While previous Grand Challenges could be solved using a purely reactive software architecture, the complex nature of the Urban Challenge necessitates a hybrid solution. In addition to the simpler goal-seeking behavior required in the previous challenges, Urban Challenge vehicles must maintain knowledge of intent, precedence, and timing. With many concurrent perception and planning tasks of varying complexity, priority, and computation time, parallelism is preferred to a single monolithic Sense-Plan-Act structure (Murphy, 2000). In addition, the complexity of the Urban Challenge problem necessitates a well-defined software architecture that is modular, clearly segmented, robust, safe, and simple.

VictorTango's software structure employs a novel Hybrid Deliberative-Reactive paradigm. Odin's perception, planning, and acting occur at several levels and in parallel tasks, acting on the most recent information received from other modules. With traditional Hybrid architectures, deliberative components are usually kept at a high level, while the more reactive, behavior-based, components are used at a low-level for direct actuator control (Konolige, 1998 and Rosenblatt, 1995). With the rapid growth of computing technology, however, there has been a re-emergence of deliberative methods for low-level motion planning (Urmson, 2006, and Thrun, 2006). Search-based approaches provide the important traits of predictability and optimality, which are useful from an engineering point of view (Russel, 2003). VictorTango's system architecture therefore exhibits a *deliberative-reactive-deliberative* progression. As a result, the scope of a behavioral control component can be moved from low-level reflexes to higher-level decision making for solving complex, temporal problems. An overview of the hybrid mixture of deliberative planning, reactive navigation, and concurrent sensor processing is shown in Figure 4. Each of the modules is further detailed in the following sections.

SAE AS-4 JAUS (Joint Architecture for Unmanned Systems) was implemented for communications, enabling automated dynamic configuration and enhancing the future reusability and commercialization potential of DUC software. Each software module is implemented as a JAUS component with all interactions to and from other modules occurring via JAUS messages. As such, each software module operates as a standalone component that can be run on any one of the computing nodes. Since dynamic configuration and data subscription is handled via JAUS, the system is highly reconfigurable, modular, expandable, and reusable beyond the Urban Challenge. An additional benefit of employing a communications standard toolkit was the easy integration of logging and simulation (both discussed further in section 6).

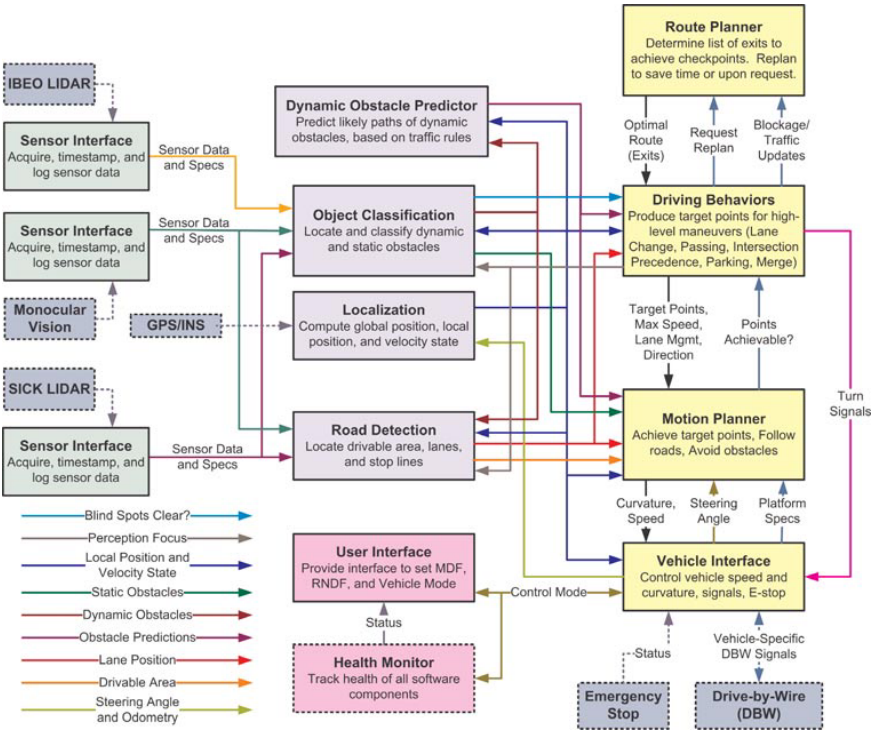


Fig. 4. System Architecture for Odin, omitting Health Monitor connections for clarity

2.2 Perception

Perception is defined to include all aspects of the design necessary to sense the environment and the vehicle's position in it. Each perception module transforms raw data collected from multiple sensors into information useful to the decision making software.

2.2.1 Sensor Layout

The sensor coverage for Odin is shown in Figure 5. The largest portion of Odin's detection coverage is provided by a coordinated pair of IBEO Alasca XT Fusion laser rangefinders. This system comprises two 4-plane, multi-return rangefinders and a single external control unit (ECU) that covers a 260 degree field of view as shown in Figure 5. The system has an advertised range of almost 200 meters, although the effective range to reliably identify most objects has been shown in testing to be closer to 70 meters. A single IBEO Alasca A0 unit with a field of view of 150-degrees is used to detect approaching vehicles behind Odin and navigate in reverse. The Alasca A0 is an earlier generation Alasca sensor than the XT, and testing has shown a lower range of approximately 50 meters for reliable object classification.

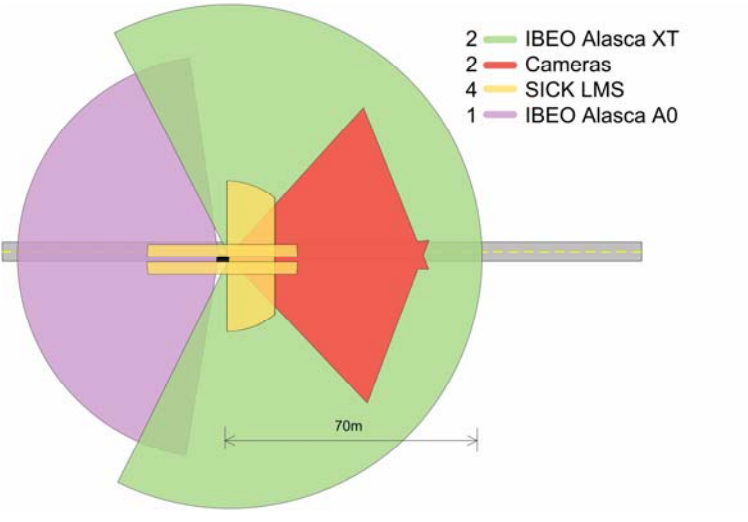


Fig. 5. Odin's sensor coverage. The colored areas indicate the maximum range of the sensor or the point at which the sensors scanning plane intersects the ground. Odin is facing to the right in this figure.

For short range road detection and obstacle detection, two additional SICK LMS 291 laser rangefinders are angled downward on the front corners of the roof rack. These sensors are able to detect negative obstacles and smaller obstacles that may be underneath the IBEO XT vertical field of view. Two side-mounted SICK LMS 291 single plane rangefinders are used to cover the side blind spots of the vehicle and ensure 360-degree coverage. The side mounted SICK LMS sensors are primarily utilized during passing maneuvers.

Two IEEE 1394 color cameras were intended to supplement the IBEO obstacle classification software and perform road detection, but were not used in the final competition configuration. In combination, the cameras cover a 90-degree horizontal field of view in front of Odin, and each transmit raw 1024 by 768 images at 15 frames per second.

2.2.2 Road Detection

The Road Detection software component provides information about nearby roads and zones in the form of lanes (Report Lane Position) and overall drivable area (Drivable Area Coverage). Report Lane Position describes the available lanes of travel, and is used for decision making, vehicle navigation, and dynamic obstacle predictions. Drivable Area Coverage defines all areas available for Odin to drive, which is applied as a road filter for detected objects, and is used to assist with zone navigation. These two outputs are generated from three different sources: the RNDF, vision data, and SICK LIDAR data. The RNDF is used to define all lanes and drivable areas within a certain range of the vehicle. The sensor data is then used to better define roads when the waypoints are sparse or GPS coverage is poor. Both SICK LIDAR and vision processing can be manually enabled or disabled if not needed due to the configuration of the current course.

RNDF Processing

The basis for the Road Detection module is the Route Network Definition File (RNDF) supplied by DARPA. The specified lanes and exit-entrance pairs in the file are preprocessed to automatically to create continuous paths for Odin. Cubic spline interpolations produce a piecewise continuous curve that passes through all waypoints in each lane. This interpolation uses a cubic function, the waypoint positions, and a desired heading to ensure a smooth transition between adjoining pieces of the lane (Eren, 1999). The cubic function used to define the spline interpolation is:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$
$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

where $x(u)$ and $y(u)$ are the point position at u which is incremented from zero to one to generate the spline interpolation between two points. The eight unknowns of these two equations (a_x, b_x, c_x, d_x, a_y, b_y, c_y, d_y) can be determined using the eight boundary conditions set by the waypoint positions and desired headings:

$$p(0) = p_k$$
$$p(1) = p_{k+1}$$
$$p'(0) = \tfrac{1}{2} c(p_{k+1} - p_{k-1})$$
$$p'(1) = \tfrac{1}{2} c(p_{k+2} - p_k)$$

where p_{k-1}, p_k, p_{k+1}, and p_{k+2} represent the waypoint positions (x and y), and c is the desired curvature control. In matrix form, this set of equations for a spline interpolation between two points is as follows:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \\ a_y \\ b_y \\ c_y \\ d_y \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ x_{k+1} \\ y_{k+1} \\ \frac{1}{2}c(x_{k+1} - x_{k-1}) \\ \frac{1}{2}c(y_{k+1} - y_{k-1}) \\ \frac{1}{2}c(x_{k+2} - x_k) \\ \frac{1}{2}c(y_{k+2} - y_k) \end{bmatrix}$$

After solving for the unknowns, u is incremented at the desired resolution, or number of points, to create the interpolation between the two waypoints. The curvature control variable can be adjusted to increase or decrease the curvature between the points. This splining is also used to generate the desired path (termed a "lane branch") through intersections for every exit-entrance pair. Branches extend a lane from an exit point to the entrance point of a connecting lane, allowing lane maintenance to guide a vehicle through an intersection. Figure 6 shows an example spline interpolation for a 90° right turn (e.g. a typical right turn at a four-way intersection). The plot on the left shows an ideal spline for this intersection while the plot on the right shows the effect of a lower curvature control value. Four waypoints, shown as round points, are required for a cubic spline interpolation. The linear connection between these points is shown as the dashed line for visual reference. The cubic spline interpolation with a resolution of five points is the solid line with square points.

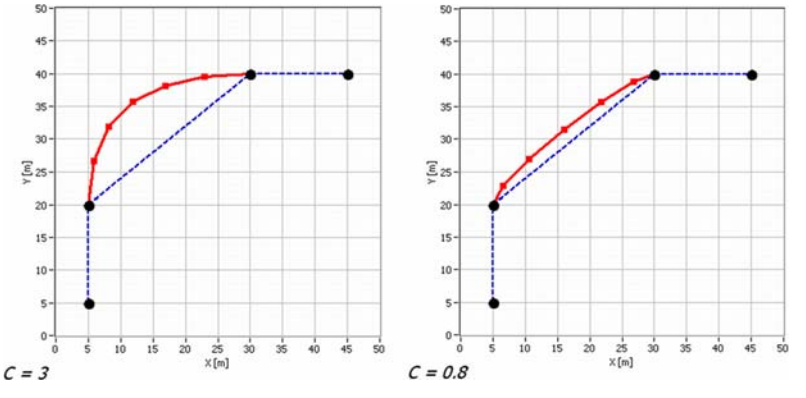


Fig. 6. Example cubic spline interpolation for a 90° right turn with an ideal spline (left). The effect of a lower curvature control value is also plotted (right). Waypoints are the round points. Spline interpolation is the solid line with square points. For visual reference, linear connections are shown by the dashed line.

All splining for the RNDF is preprocessed when an RNDF is loaded. As Odin drives the RNDF, nearby lanes and intersections are extracted and assembled into the Report Lane Position output. This output creates the possible paths for Odin and other sensed vehicles. The Report Lane Position software was developed to automatically generate these cubic spline interpolations for an entire RNDF. This is achieved in two steps using the geometry between waypoints. First, it is determined whether or not spline interpolation is necessary. In other words, a series of straight waypoints or a straight through intersection does not require cubic splining, but traveling around a traffic circle would. Next, the curvature control is selected for the locations that required splining using the distance between waypoints, the previous and future desired headings, and knowledge gained from cubic spline data analysis. The automatic spline generator was originally designed to guide Odin through intersections as a human driver would, and therefore performed close to flawlessly in these more open navigation situations. Lane splines, on the other hand, required much more precision to follow the course of the road. The splining process always provided smooth paths for the vehicle, but lacked the realization of the actual geometry of the roads and intersections.

A solution to this problem was to compare the cubic spline output to geo-referenced aerial imagery, which was guaranteed to be supplied by DARPA for the competition. Using this information, the spline curvatures could be manually adjusted to help ensure the splined lane positions match the physical lanes in the road network. Therefore, the RNDF processing software was modified to accept manual overrides to the generated splines, and save a configuration file if changes were required. Figure 7 shows an example of comparing the splined Report Lane Position output (bright overlays) to the actual road (outlined by the beige curbs) in the geo-referenced aerial imagery of Victorville.

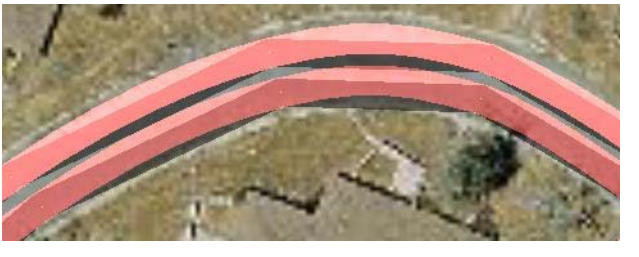


Fig. 7. Example of the splined lane positions (bright overlays) output matching the actual roads in the aerial imagery. The road displayed is segment 11 of the UCE RNDF.

RNDF based Drivable Area Coverage uses a combination of the splined Report Lane Position output and the zones from the RNDF. This is also preprocessed to create the Drivable Area Coverage for the entire course. During operation, the drivable area within range of Odin is extracted and output as a drivable area map. This drivable area map is a binary image that marks all nearby areas in which Odin or another vehicle can be expected to operate.

Sensor Data

Odin also uses LIDAR data to supplement the RNDF generated lane and road positions. The two forward-looking SICK LIDAR identify the road by looking for rapid changes in range that result from discontinuities in a flat road surface, such as those caused by curbs, potholes, and obstacles. The SICK LIDAR are positioned at different vertical angles to allow the algorithm to analyze multiple returns on the same features.

Lane positions can be predicted by fitting probable road boundary locations through a second-order least squares regression analysis. Given the locations classified as potential curb sites by the LIDAR, the curb points are defined as the points that follow the previous estimate inside an allowable scatter. This scatter is determined by the standard deviation of the previous estimate and is weighted to allow more points closer to Odin and to select the closest curb boundary detected. Figure 8a shows logged data indicating all potential curb points, the subset used in the regression, and resulting boundary curves. Lane position can then be determined by referencing the expected number of lanes in the RNDF. The RNDF-based Report Lane Position output can be augmented with these sensed lanes. Figure 8b shows an aggregation of the curb points previously used in the regression traveling down a two lane road. The curb points follow the shape of the of the road, but as shown in Figure 8a, detection only reaches 10-15 meters in front of the vehicle, requiring software to slow the speed of the vehicle to maintain safe operation.

In addition to LIDAR, computer vision approaches were also attempted for detecting lanes as well as improving the drivable area coverage map. The visual lane detection software uses image intensity to distinguish bright lane boundaries

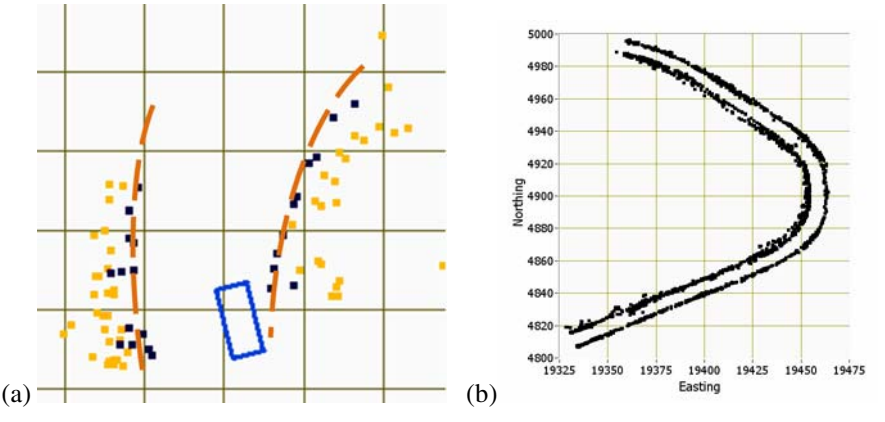


Fig. 8. LIDAR based road detection results: (a) A single frame of potential curb points in world frame with points used in the regression darkened. The regression output curve is the dashed line. Grid spacing is 5 meters, and Odin is represented by the rectangle. (b) An aggregation of all curb points previously used in the lane boundary regression along a single road, plotted in UTM coordinates.

from the darker surrounding areas of the road surface. Edge detection is applied to the results of the intensity operation, separating out the lines and edges of the road. The position of each lane is found by fitting the strongest lines on the road to a curve through a Hough transform (Duda, 1972). Vision was also used to improve the drivable area coverage map in zones by finding low-profile islands usually found in parking lots. These islands are often a different color than the surrounding area, and therefore vision processing is ideal for detecting them. The algorithm thresholds the entire image according to the absolute color of an area directly in front of Odin, which is assumed to be drivable. Significant color changes in this control area are ignored for a short period of time to improve this assumption. The detected features are then subtracted from the Drivable Area Coverage map generated from the RNDF. Both of these vision algorithms were not used in the final competition due to a lack of sufficient testing and development as further discussed in section 3.

2.2.3 Object Classification

The accurate identification and classification of objects is one of the most fundamental and difficult requirements of the Urban Challenge. The vision system and the laser rangefinders each have advantages and disadvantages for classification. The IBEO rangefinders can determine the location of an object to sub-meter accuracy, but they have poor classification capabilities. Vision-based methods can result in accurate classification, but they are computationally intensive and have a limited horizontal field of view and range.

The classification module splits all objects into one of two categories: static objects that will not be in motion, and dynamic objects that are in motion or could be in motion. Dynamic objects on the course are expected to be either manned or unmanned vehicles. The core of the classification module, shown in Figure 9, is the IBEO laser rangefinders. While visual methods of detection were examined, they were determined to be too computationally intensive to return accurate information about nearby objects, especially at higher vehicle speeds. This problem is intensified due to the fact that multiple cameras are needed to cover a full 360-degree sweep around Odin. The A0 and XT Fusion rangefinders cover almost the entire area around Odin, and objects can be detected and segmented using software available on the IBEO ECUs (Fuerstenberg, Dietmayer, Lages, 2003), (Fuerstenberg, Linzmeier, Dietmayer, 2003). These ECU objects serve as the basis for the module. However small deviations in the laser pulse's reflection point and time of flight often causes variations in the results of the built-in segmentation routines. To account for these variations a filter is applied that requires each object to have been detected and tracked for a short but continuous period of time before the object is considered valid. The positions of these valid objects are then checked against the Drivable Area Coverage map; anything not on or close to drivable area is considered inconsequential and is not examined.

Once these detected objects are sufficiently filtered through their position and time of detection, their characteristics are passed to a classification center for verification. Through testing, the IBEOs have proven to be accurate in

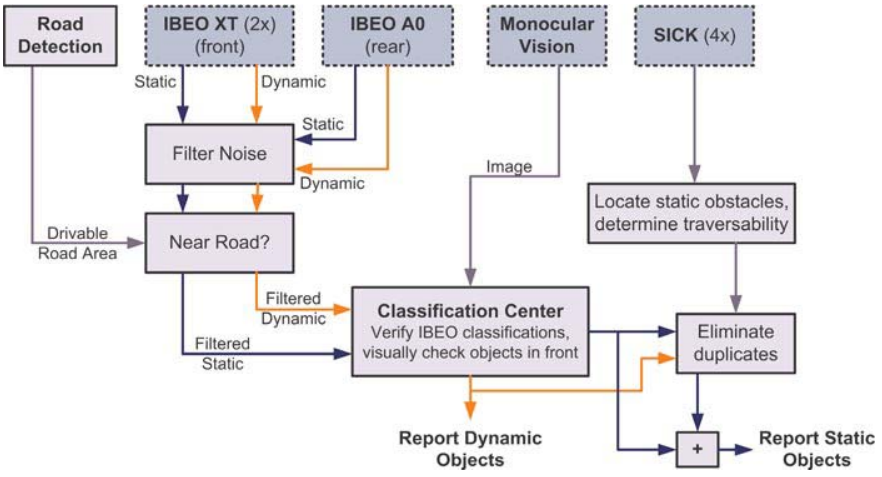


Fig. 9. The Object Classification module localizes and classifies all perceived objects

determining a moving object's velocity, and it is assumed that all large moving objects are vehicles. It is also important for the system to detect stationary vehicles in front of Odin for situations such as intersection queuing and precedence. The initial software design included verification of object classification using monocular image processing. The real-world locations of objects obtained from the IBEOs are transformed into regions of interest within the image, which are searched for features common to cars such as tail lights and tires (Cacciola, 2007). By restricting processing to these regions, high resolution imagery could be used without the usual processing penalty. This feature was effective at correcting groups of static obstacles being incorrectly classified as a dynamic obstacle. However there was an inadequate amount of test time logged to verify that certain critical failure modes, such as the vision system incorrectly identifying a dynamic object as static or the correct handling of complete vision outages due to poor lighting conditions, would not occur. Therefore, the vision portion of the classification module was not used in the final competition.

2.2.4 SICK LIDAR Based Detection

The four SICK LMS-291 units on Odin were used for close-range object detection. The two side-mounted SICK LIDAR are devoted to blind spot checking. Figure 10 shows a history of LIDAR objects after being transformed into vehicle frame. If the areas adjacent to the vehicle sides have a return that is above a height threshold, then the blind spot is reported as not clear.

The two front-mounted SICK LIDAR are used to detect objects that are within close range of the vehicle but outside the IBEO's vertical field of view. The returns of these downward pointed laser rangefinders are segmented based on range discontinuities and classified as road or an obstacle based on a height threshold. This information is compared over multiple scans to capture the overall

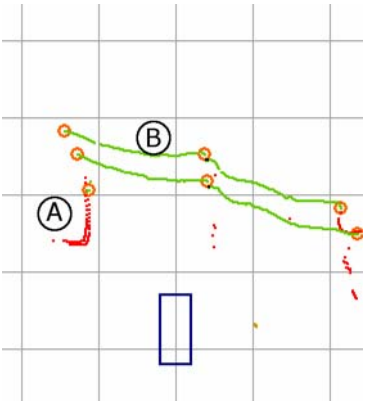


Fig. 10. SICK LMS scan data transformed into vehicle frame. A (the outline of a car) has been classified as an obstacle, B has been classified as drivable road, the circles are potential curb sites. Grid spacing is 5 meters, and Odin is represented by the rectangle.

behavior of an object through time, and is used to reclassify the object if necessary. Figure 10 shows an example of classifications derived from a SICK scan cycle. This illustration also shows the potential curbs marked as circles. These points are determined after the drivable area is distinguished, and defined as points where the drivable area ends or had a sharp step in the profile.

2.2.5 Dynamic Obstacle Predictor

Once an object has been classified as a vehicle, it is monitored by the Dynamic Obstacle Predictor, which predicts likely paths for each vehicle based on road data and the motion history of the object. If there is no available lane data, such as in zones or if a dynamic obstacle doesn't appear to be following a lane, the Dynamic Obstacle Predictor simply continues the current motion into the future. These predictions are used by Driving Behaviors for traffic interaction at intersections (such as merges) and Motion Planning for obstacle avoidance.

2.2.6 Localization

Odin has been equipped with a Novatel Propak LB+ system that provides a filtered GPS/INS solution. In addition, wheel speed and steering angle measurements are available from the vehicle interface. An Extended Kalman Filter (EKF) has been developed using standard methodology (Kelly, 1994) that combines these measurements with the goal of improving the Novatel solution during GPS outages. In addition, this filter provides a means for incorporating other absolute position measurements. The best case accuracy of the position provided by this localization solution is 10 cm CEP.

A separate local position solution is also tracked in Odin's localization software. This solution provides a smooth and continuous position estimate that places Odin in an arbitrary local frame. The goal of this position solution is to provide perception modules with a position frame to place obstacles that is free of

discontinuities caused by GPS position jumps. This is accomplished by calculating the local position using only odometry. This position typically drifts by 2.6% of the total distance traveled by the vehicle; however, this error accumulation is small enough that it does not cause significant position error within the range of the vehicle's perception sensors.

2.3 Planning

Decision making for Odin is handled by a suite of modules cooperating in a hybrid deliberative-reactive-deliberative fashion. Each of the major components is presented in sequence, from the top down.

2.3.1 Route Planning

The Route Planner component is the coarsest level of decision planning on Odin as it only determines which road segments should be traveled to complete a mission. It uses a-priori information such as the road network and speed limits specified by the RNDF and MDF respectively, as well as blockage information gathered during mission runs. After processing, the Route Planner outputs a series of waypoint exits to achieve each checkpoint in the mission.

By only considering exit waypoints, it is easy to formulate the Route Planner as a graph search problem. The Route Planner on Odin implements the A* graph search method (Hart, 1968) using a time-based heuristic to plan the roads traveled. While the A* search algorithm guarantees an optimal solution, it depends on the validity of the data used in the search. The time estimate used during the search assumes that the vehicle is able to travel at the specified segment speed limits, and it uses predefined estimates of the time for typical events, such as the time taken when traversing a stop line intersection, performing a U-turn, or entering a zone.

2.3.2 Driving Behaviors

Driving Behaviors is responsible for producing three outputs. First, Driving Behaviors must produce a behavior profile command which defines short term goals for Motion Planning in both roads and zones. Second, in the event of a road-block, a new set of directions must be requested from the Route Planner. Finally the turn signals and horn must be controlled to appropriately signal the intent of the vehicle.

The behavior profile sent to Motion Planning comprises six target points, a desired maximum speed, travel lane, and direction (Forward, Reverse, and Don't Care). Each target point contains a waypoint location in UTM coordinates, the lane, and lane branch (for intersections). Target points can also contain optional fields such as a stop flag and a desired heading. Lastly, the behavior profile also contains zone and safety area flags to enable different behaviors in Motion Planning.

Action-Selection Mechanism

Driving Behaviors must coordinate the completion of sophisticated tasks in a dynamic, partially observable and unpredictable environment. The higher level

decision making being performed in Driving Behaviors must be able to handle multiple goals of continually changing importance, noisy and incomplete perception data, and non-instantaneous control. To do this, a Behavior-Based Paradigm was implemented. The unpredictable nature of an urban environment calls for a robust, vertical decomposition of behaviors. Other advantages of using a Behavior-Based Paradigm include modularity, the ability to test incrementally, and graceful degradation (Murphy, 2000). For instance, if a behavior responsible for handling complex traffic situations malfunctions, simpler behaviors should still be operable, allowing Odin to continue on missions with slightly less functionality.

As with any Behavior-Based architecture, implementation details are extremely important and can lead to drastically different emergent behaviors. Coordination becomes paramount since no centralized planning modules are used and control is shared amongst a variety of perception-action units, or behaviors. In the Urban Challenge environment, the problem of action selection in the case for conflicting desires is of particular interest. For example the desire to drive in the right lane due to an upcoming right turn must supersede the desire to drive in the left lane due to a slow moving vehicle. Furthermore, due to the strict rules of urban driving, certain maneuvers must be explicitly guaranteed by the programmer. To address this problem, a method of behavior-selection is needed such that Driving Behaviors will actively determine and run the most appropriate behaviors given the current situation.

An *arbitration* method of action selection (Pirjanian, 1999) is used for the Driving Behaviors module. In the above example of choosing the appropriate lane to drive in, driving with two wheels in each lane is not an acceptable solution. Therefore, a modified Winner-Takes-All (Maes, 1989) mechanism was chosen. To address the situational awareness problem, a system of hierarchical finite state machines is used. Such a system allows Driving Behaviors to distinguish between intersection, parking lot, and normal road scenarios (Hurdus, 2008). The implementation of finite state machines also provides resilience to perception noise and by using a hierarchical system, concurrency is easily produced. The overall architecture of Driving Behaviors is shown in Figure 11. A finite state machine is used to classify the situation, and each individual behavior can be viewed as a lower-level, nested state machine. The chosen Action Selection Mechanism operates within the Behavior Integrator. This approach is considered a *modified* Winner-Takes-All approach because all behavior outputs are broken down into one of several categories, including, but not limited to, Target Point Drivers, Speed Drivers, and Lane Drivers.

Passing and Blocked Roads

When driving down a normal section of road, (i.e. not in a safety zone approaching an intersection, not in an intersection polygon, and not in a zone)

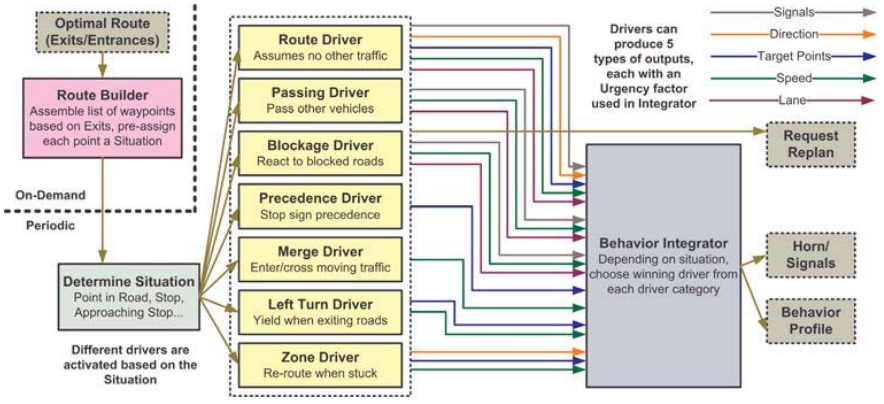


Fig. 11. Flow diagram of the Behavior-Based, Winner-Takes-All Driving Behaviors implementation. Behavior Integrator ensures there is one winner from each driver category.

Odin runs three behaviors, the Route Driver, the Passing Driver, and the Blockage Driver. The Route Driver is responsible for driving the route as close as possible to the route originally provided by the Route Planner. If no obstacles or traffic are ever encountered, then the Route Driver will maintain control of the vehicle throughout all segments of the RNDF. When entering a new segment, for example, the Route Driver will immediately attempt to move Odin to the correct lane for the next exit.

The Passing Driver is concerned with getting around slow moving or disabled vehicles. It is therefore responsible for monitoring other vehicles in the near vicinity, deciding if a pass is necessary, and executing this pass in a safe and legal manner. Awareness of the roads is necessary as the Passing Driver must distinguish between passing in an oncoming lane and passing in a forward lane, and subsequently check the appropriate areas for traffic. The Passing Driver does not maintain knowledge of the overall route, so it is the responsibility of the Route Driver to overrule the Passing Driver if a pass is initiated too close to an exit or intersection.

Finally, the Blockage Driver maintains a current list of available lanes. If static obstacles in the road or a disabled vehicle cause a lane to be blocked, the Blockage Driver removes this lane from the available list. If all RNDF defined lanes are removed from the list and at least one of these lanes is an oncoming lane, then the Blockage Driver commands a dynamic replan. When this is necessary, the Route Planner is first updated with the appropriate blockage information and all behaviors are reset while a new route is generated.

The interaction of these three drivers was sufficient for giving Odin the ability to pass disabled and slow-moving vehicles in the presence of oncoming and forward traffic, pass static obstacles blocking a lane, pass over all checkpoints, be in the correct lane for any exit, and initiate dynamic replans when necessary.

Intersections

To handle intersections, Odin uses three drivers (Precedence, Merge, and Left Turn) in the Approaching Stop, Stop, Approaching Exit, and Exit situations. Of special note is that all three drivers operate by monitoring areas where vehicles (or their predictions) may be, rather than tracking vehicles by ID. While this decision was initially made to deal with object tracking issues in early iterations of the perception software, it turned out to greatly enhance the overall robustness of the intersection behavior.

The Precedence Driver activates when Odin stops at junctions with more than one stop sign. This driver overrides the Route Driver by maintaining the current stop target point with a high urgency until it is Odin's turn or a traffic jam has been detected and handled. The Merge Driver activates at intersections where Odin must enter or cross lanes of moving traffic, controlling the speed by monitoring areas of the merge lane or cross lanes for vehicles or vehicle predictions. To handle intersections with both moving traffic and other stop signs, the Merge Driver cannot adjust the speed until the Precedence Driver has indicated it is Odin's turn or a traffic jam has been detected.

For turns off a main road (a case where the RNDF does not explicitly state right of way via stop points), the Left Turn Driver activates when Odin's desired lane branch at an upcoming exit waypoint crosses over oncoming traffic lanes. In this case, the Left Turn Driver overrides the Route Driver by setting the stop flag for the exit target point. Once the exit waypoint has been achieved, the driver controls the desired speed in the behavior profile while monitoring the cross-lanes for a sufficient gap to safely achieve the left turn.

Parking Lot Navigation

In zones, the role of Driving Behaviors is to provide general zone traversal guidance by controlling the behavior profile target points. Initially, VictorTango planned to fully automate the first stage of this process – determining the accepted travel patterns through a parking lot (along the parking rows). However, zones containing only a subset of the actual parking spots (i.e. one spot per row) make it difficult to automatically identify parking rows based on the RNDF alone. Since this could very well be the case in the final event, a tool was designed to manually place "control points" in the zone, in the form of a directionally-connected graph.

In the route building stage of Driving Behaviors, Odin performs a guided Dijkstra search to select control points for navigating toward the parking spot and reversing out of the spot. The Route Driver then guides Odin along this pre-planned path. If the path is blocked, the Zone Driver can disconnect a segment of the graph and choose a different set of control points. The parking maneuver is signaled to Motion Planning by enabling the stop flag and providing a desired heading on the parking checkpoint. To reverse out of the spot, the direction is constrained to be only in reverse, and a target point is placed in order to position Odin for the next parking spot or zone exit.

Driving Behaviors is not responsible for any object avoidance or traffic behavior in zones. Motion Planning handles tasks such as steering to the right for oncoming dynamic objects, performing the parking maneuver, and checking for safe space to reverse out of the spot. This was primarily due to the largely unknown environment in a zone, and the desire to keep Driving Behaviors independent of the exact size and mobility constraints of the vehicle.

2.3.3 Motion Planning

Motion Planning is responsible for planning the speed and path of the vehicle. Motion Planning receives behavior profiles, defined in section 2.3.2, from Driving Behaviors and plans a series of motion commands called a motion profile. The motion profile is a platform independent series of commands that include a desired curvature, curvature rate of change, desired velocity, maximum acceleration and time duration of each command. The motion profile consists of the entire path planned by motion planning and typically contains 2-3 seconds of commands. A platform independent motion profile message allows re-use of communication messages across base platforms, and allows vehicle specific control loops to take place in the Vehicle Interface. However, Motion Planning still requires a basic model of the specific operating platform. In addition to commanding the Vehicle Interface, Motion Planning can also provide feedback to Driving Behaviors about whether the currently commanded behavior profile is achievable. This feedback is critical when detecting a blocked lane or even an entirely blocked roadway.

Motion Planning is structured into two main components consisting of a Speed Limiter and a Trajectory Search as shown in Figure 12. The Speed Limiter commands a maximum speed based on traffic in the future path of Odin and upcoming stop commands. Dynamic Obstacle predictions are analyzed to follow slower moving traffic or to stop behind a disabled vehicle leaving enough room to pass. The Speed Limiter sends Trajectory Search this maximum speed and an obstacle ID if the speed is limited by a dynamic obstacle. The speed limiter is disabled when traveling through zones, leaving Trajectory Search to handle all dynamic obstacles.

The core of Motion Planning is the Trajectory Search module that simulates future motion to determine a safe and fast path through the sensed environment. Three main steps happen in Trajectory Search: (1) a cost map is created using lane and obstacle data, (2) a series of actions is chosen to reach a goal state, and (3) the series of actions is processed into a feasible motion profile. The Trajectory Search plans with the assumption of an 80ms processing time. If run time exceeds 450ms or all possible actions are exhausted, the goal criteria are declared unachievable. Driving Behaviors is notified about the unachievable goal and a stop is commanded, with urgency depending on the proximity of obstacles.

Trajectory Search uses a fixed grid size cost map to represent the environment when solving for a motion path. The range and resolution of the cost map is configurable, and the final configuration used a resolution of $20cm^2$ per cell, extending 30m in front of the vehicle, and 15m behind and to the sides of the vehicle. It is important to note that Driving Behaviors and the Speed Limiter

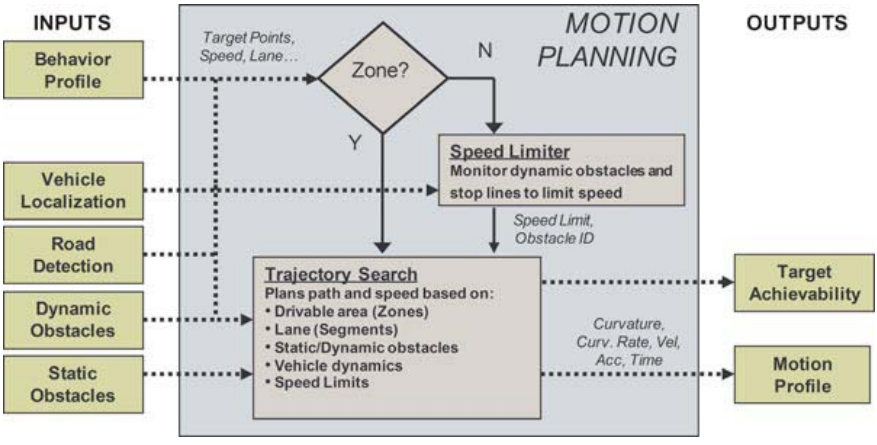


Fig. 12. Software flow diagram of the Motion Planning component

software did not use this cost map, allowing these modules to incorporate dynamic obstacles beyond this range. The cost map stores costs as 8-bit values allowing obstacles to be stored as highest cost at the obstacle and reduced cost around the obstacle. Figure 13a shows an example cost map with costs added from obstacles and lane boundaries. Dynamic obstacles are expanded to account for future motion. However, this treatment of dynamic obstacles is very limiting, and responding to dynamic obstacles is mainly the job of the Speed Limiter. If the Speed Limiter is reacting to a dynamic obstacle, the ID is passed to Trajectory Search and the obstacle is omitted from the cost map. Omitting these dynamic obstacles prevents Trajectory Search from deciding a slower moving vehicle is blocking a lane.

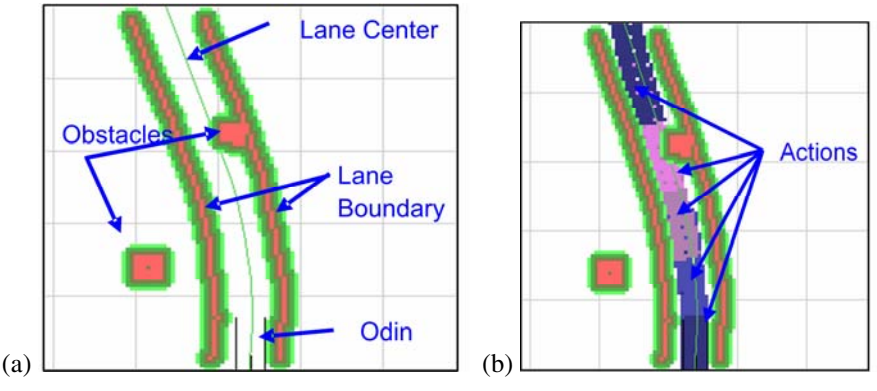


Fig. 13. (a) Trajectory Search cost map with labeled features. (b) Trajectory Search cost map with planned solution representing each action in a different color.

Once a cost map is created, Trajectory Search then produces a set of goal criteria using data from the behavior profile such as: desired lane, zone, desired gear and heading criteria. Goal criteria may be as simple as driving a set distance down a lane or more specific such as achieving a desired position with a desired heading. The search process starts with the current vehicle state and uses an A* search to evaluate sequences of possible future actions. Each action consists of a forward velocity and steering rate over a period of time. These actions are evaluated by checking the predicted path of the vehicle due to the action against the cost map as well as other costs such as distance from the lane center and time. The search speed is improved by only using a finite set of possible actions that have pre-computed motion simulation results (Lacaze, 1998). Figure 13b shows a planned path with each action having a different color.

Odin uses a pre-computed results set with an initial steering angle varied at 0.25 degree increments, commanded steering rates varied from 0 to 18 degrees/sec at 6 degree/second increments and commanded velocities ranging from 2 to 12.5 m/s at 3.5 m/s increments. The pre-computed results contained information including which grid cells will be occupied as well as final state information. A separate set of coarser results is used in situations where the vehicle is allowed to travel in reverse. When creating a sequence of actions, the pre-computed occupancy data is translated and rotated based on the previous ending state conditions. When the search algorithm runs, actions that are dynamically unsafe or have a commanded velocity too far from the previous velocity are filtered out. After the search is complete, the list of actions is converted to a drivable series of motion commands. This last step selects accelerations, and accounts for decelerating in advance for future slower speeds. Steering rates are also scaled to match the planned path if the vehicle is travelling at a different speed than originally planned. The Trajectory Search solves the goal using the fastest possible paths (usually resulting in smoother paths), and these speeds are often reduced due to MDF speed limits or due to the Speed Limiter.

While traveling in segments, Trajectory Search chooses goals that travel down a lane. In contrast, zone traversal is guided by target points along with goal criteria and search heuristics to produce different behaviors. The behaviors include forming artificial lanes to connect the points, a recovery behavior allowing the vehicle to reverse and seek the center of the fake lane, and a wandering behavior that uses no lane structure at all. These behaviors are activated by a simple state machine that chooses the next behavior if the current behavior was resulting in unachievable goals. Development time was reduced and reliability was improved by using the same Trajectory Search algorithm in zones as well as segments. While the overall behavior of the vehicle could be adjusted by changing the goals and heuristics, the core functionality of planning fast collision-free motion remained constant.

2.3.4 Vehicle Interface

The main role of the Vehicle Interface component is to interpret the generic motion profile messages from Motion Planning, defined in section 2.3.3, and output vehicle-specific throttle, brake, steering, and shifting signals. By accepting

platform independent motion commands and handling speed control and steering control at a low level, any updates to the vehicle-specific hardware or software can be made transparent to the higher level decision making components. Additionally, the Vehicle Interface can actuate other vehicle systems such as lights, turn signals, and the horn.

Closed loop speed control is provided by a map-linearized PID controller. The controller, as shown in Figure 14, takes the output of the PID, band limits it to control the maximum acceleration, and inputs it to a map lookup function to produce a throttle or brake command. Terrain compensation is provided by a proportional controller that estimates the longitudinal acceleration on the vehicle and additively enhances the map input (Currier, 2008).

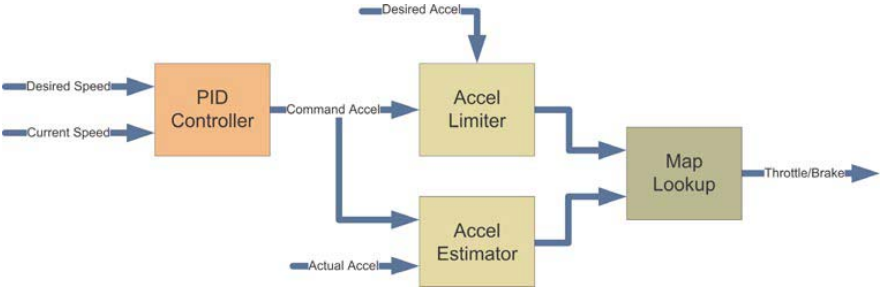


Fig. 14. Block diagram of speed controller showing PID controller, acceleration controller and map lookup linearization function.

Steering control relies on a standard bicycle model to estimate the curvature response of the vehicle (Milliken, 1995). This model can be shown to produce estimates accurate enough for autonomous driving in the operational conditions found in the Urban Challenge (Currier, 2008). The steering angle and rate calculated by the bicycle model is tracked by a rate controlled PID loop to produce the desired vehicle path.

3 Final Software Configuration

As the Urban Challenge approached, decisions had to be made for the final software configuration to be used in the event to ensure adequate testing time. These choices ranged from setting maximum/minimum parameters to disabling software components. This section explains test procedures and rationale for the final adjustments made to Odin's software configuration.

3.1 Motion Planning Parameters

For all NQE runs as well as the final UCE, the maximum speed that Odin would drive was limited to 10 m/s (22 mph). This allowed Odin to drive at the maximum speed specified for all segments during NQE, and the maximum speed for all but

two segments on the UCE. The limiting factor in determining a safe maximum speed was the distance that obstacles could be reliably detected. During testing, the vehicle could smoothly drive roads at speeds of 13 m/s (29 mph), but would occasionally stop very close to obstacles such as a disabled vehicle or roadblock. This was due to hills or varying terrain causing obstacles to be out of the vertical sensor field of view until Odin was closer to the obstacle.

3.2 Sparse Road Detection

Prior to competition, members of the team tested the road detection suite in a wide variety of sparse waypoint scenarios. Odin was tested on various road compositions including dirt, gravel, and asphalt roads. Lane definitions were also varied. These tests were on well lined and poorly lined roads, as well roads with curbs, drop offs, and ditches. Laser rangefinder based methods of lane detection yielded reasonably robust results, but due to the possible variety of road surfaces and markings, vision based methods were much less robust. The primary challenge for lane detection was defining assumptions. The algorithm did not have enough engineering time to handle all possible roads. Further, the sparse waypoint example in the sample RNDF indicated that Odin would be required to negotiate intersections in sparse scenarios, which through analysis became very demanding on the algorithms.

As a result, the team felt comfortable that if sparse waypoints would be required, this software could be turned on; but it was deemed that splining waypoints would be the method of choice if the final event RNDF permitted. Hence, before arriving in California, all code associated with vision based lane detection was disabled, and an implementation of the software that allowed the team to selectively disable/enable the laser rangefinder based lane detection was put in place. This allowed the team to turn on laser rangefinder based lane detection where certain conditions are met. Upon receipt of each RNDF, the team would examine all points in the simulator visualization to determine if the segments required the sparse waypoint algorithm to be enabled for that RNDF. Due to the relative density of the waypoints in all events there was never a requirement for this algorithm to be enabled.

3.3 Vision Drivable Area Coverage

As previously defined in section 2.2.2, the RNDF defines the drivable areas while sensor data is used to verify the RNDF information and subtract out areas that contained obstacles, such as landscaped islands in parking lots. After extensive testing in various environments and lighting conditions, the vision Drivable Area Coverage was not reliable enough to be trusted. It would occasionally produce false positives that eliminated drivable areas directly in front of the vehicle when no obstacles were present. Also, there were no un-drivable areas in zones present during the NQE or UCE courses. The team decided that the laser rangefinders and Object Classification were able to detect most static obstacles that would get in the way of Odin. Therefore, vision Drivable Area Coverage was disabled to prevent

the vehicle from stopping and getting stuck in the event of a false positive produced by the vision processing.

4 National Qualifying Event

The team spent many long days and nights testing and preparing for the Urban Challenge NQE and UCE. This section presents and analyzes Odin's performance in the National Qualifying Event.

4.1 NQE A – Traffic

This NQE course challenged a vehicle's ability to drive in heavy traffic while balancing safety and aggressiveness. The first vehicle to perform at NQE course A, Odin performed well, executing merges and left turns with only a few minor errors. Odin also had no trouble with the k-rails surrounding the course that caused many other teams problems when merging into traffic. The mistakes Odin made were subsequently fixed and could be attributed to the following: needing to adjust intersection commitment threshold, an IBEO region of interest bug, and false object classifications.

To prevent Odin from slamming on the brakes and blocking an intersection, Odin will commit to traversing an intersection once it has passed a calculated threshold. This threshold is based on being able to stop without protruding into traffic lanes. However, several times in Odin's first run on NQE Area A, Odin commanded a merge in the situation show in Figure 15. About 350 milliseconds later, the previously occluded vehicle (4) suddenly appeared to sensors at a range that would normally prevent Odin from commanding a merge. However, since the stop line was very close to the cross-lane, Odin had already passed the commit point and disabled the cancellation mechanism. For the second NQE run, the team adjusted the commit point, allowing merges to be cancelled closer to the road. On the second course A run, Odin performed satisfactorily with the occluded cars by properly canceling the merge when the occluded vehicle was in view.

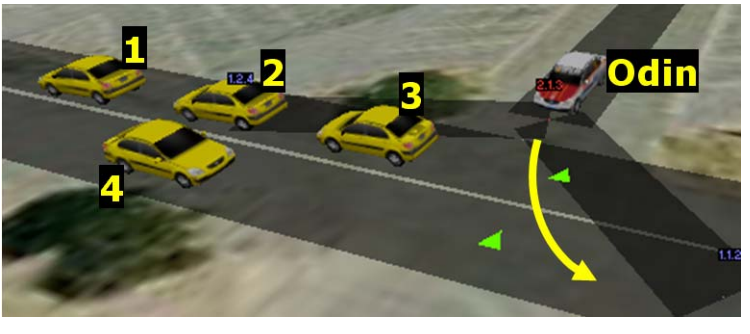


Fig. 15. A merge in NQE course A with an occluded vehicle (vehicle 4)

A bug in the IBEO factory software also resulted in Odin cutting off traffic vehicles. The IBEO sensors allow a region of interest to be defined to reduce the number of output objects, which is important given that the IBEO software has an internal limit of 64 objects. This region is in the shape of a trapezoid defined by its upper-left and lower-right corners. Objects filtered out by the region of interest can still count towards the limit of 64, which could cause objects far away from the vehicle to prevent closer objects from being returned. This behavior can be prevented by enabling an option to only process scan returns into objects within the region. However, if this option is enabled, the region is incorrectly defined as the thin rectangle within the trapezoid, causing all of the pre-calculated regions to be smaller than intended. Enabling this option was a configuration change enabled shortly before the challenge, and the issue wasn't discovered until after the second NQE course run. Afterwards, the pre-calculated regions were redefined to ensure that vehicles were always seen in time for the behavior software to correctly interact with them.

The most serious incident on NQE course A occurred during the first attempt when Odin completed a couple of laps, then came to a stop when making the left turn off the main loop. Odin unfortunately remained stopped and did not proceed through large gaps in the traffic. The judges let the team reposition Odin, and Odin was able to continue completing laps without getting stuck again. After examining logs, it was found that the retro-reflective lane markings were appearing as obstacles to the IBEO LIDAR. These sections were about the same length as a typical car and were classified as a stopped dynamic obstacle. The false objects would intermittently appear, only causing Odin to be stuck once. The software already had safeguards against waiting for stationary vehicles, but these were ineffective due to the object flickering in and out, resetting timers used to track stationary vheicles. To ensure this would not occur in the final race, the planning software was made more robust against flickering objects. These changes prevented Odin from being stuck on multiple occasions on the second NQE course A run, but did cause unnecessary delay when making a left turn. This problem was mainly a result of only depending on a single sensor modality, LIDAR, for obstacle detection and classification. Additional sensing methods, such as vision or radar, could help reduce the number of these false positives by providing unique information about surrounding objects detected through LIDAR.

4.2 NQE B – Navigation and Parking

NQE course B was used to test road and zone navigation, parking, and obstacle avoidance. The vehicles were given missions that took them around the large course, through parking areas, and down streets with parked cars and other blockages. Odin accepted the challenge at top speed and set himself up to complete with a competitive time. However, during the first run, Odin had minor parking confusion and got stuck in the gauntlet area resulting in insufficient time to complete the mission. Odin experienced a significant GPS pop on the second run of NQE course B that causing the vehicle to jump a curb. After restarting back on the road, Odin finished the course without any problems.

During the first run of NQE B, Odin approached the desired space in the row of parking spots, shown in Figure 16, and stopped for a moment. Instead of pulling into the spot, Odin reversed, drove forward in a loop around the entire row of cars, and then pulled right into the spot at a crooked angle. This surprising maneuver was caused by an incorrect assumption regarding the amount of space that would be available in front of the vehicle after a parking maneuver. Motion Planning could not find a clear path due to the car parked in front of the spot. After Odin tried to reposition itself without any progress, another behavior took control that tries to navigate around obstructions, causing Odin to circle the lot. When Odin returned to the parking spot, the software was able to find a parking solution with a clear path by parking at an angle. The parking software was improved between runs to more accurately check for a clear path that did not extend past the parking space. It is interesting to note that with a lower level software failure, Odin was still able to park, but with reduced performance.

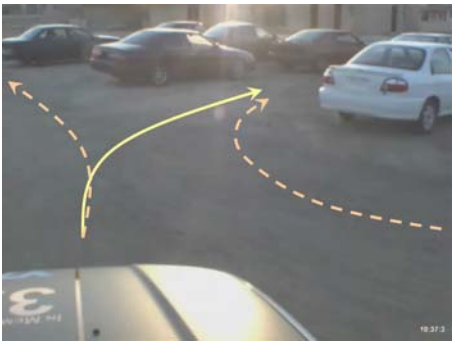


Fig. 16. During NQE B, Odin did not pull into the parking spot by the direct path (solid line). Instead, Odin pulled to the left (dashed line), circling the parked cars and eventually entered the space.

During the first run of NQE course B, Odin became stuck in the area known as the ‘Gauntlet’, characterized by vehicles parked along both sides of the road. This length of road was further complicated with cones and traffic barrels marking construction hazards along the centerline of the road. A simulator screenshot of what was seen in the Gauntlet is shown in Figure 17a. Odin tried to change lanes to pass the disabled vehicle, but was unable to execute this command because the blind spot was reported as not clear. Reviewing the logged data also revealed that Odin had difficulty traveling where both lanes of travel were significantly blocked. As seen in Figure 17a, both lanes are mostly blocked, but there is a large space in between the lanes. To solve the first problem, the lane change driver was changed to be less cautious about obstacles in a blind spot when lanes had opposite traffic directions. To allow Odin to use the entire road more effectively, motion planning would still have a strong desire to stay in the commanded lane during a lane change, but was changed to not be constrained to remain entirely in the commanded lane.

(a) (b)

Fig. 17. Gauntlet scenario from NQE area B seen in (a) simulation replay and (b) logged video

Odin experienced localization failures on the second attempt at NQE course B. As Odin exited the traffic circle onto Sabre St, the position solution provided by the Novatel system suddenly jumped almost 10 meters to the southeast. Since the road estimate was derived completely from GPS, this new vehicle position caused Odin to think it had left the lane. Motion Planning attempted to move Odin back into the desired lane of travel, but could not due to a virtual boundary placed around the lane. As a result, Odin drove to the end of Sabre Rd, thinking that it was driving off of the road. At the end of Sabre Rd, Odin reached a virtual boundary caused by the intersection of two segments. Having nowhere else to go, Odin turned right and drove onto the dirt on the side of the road and was paused. Within milliseconds of the pause, the localization solution reported by the Novatel system corrected itself. Such a large jump in position was never encountered in testing, and was not repeated in any NQE course or the final event.

4.3 NQE C – Intersections and Road Blocks

The final NQE course was used to test intersection precedence and dynamic replanning due to road blocks. Odin performed perfectly at all intersections yielding to those who had right-of-way and taking the appropriate turn, shown in Figure 18a. Replanning due to a road blockage was also a success. An interesting challenge presented in NQE C was a road blockage that was repeatedly approached by the vehicles during a mission. When Odin detected a blockage requiring a route replan, the Route Planner would add the blockage and a pair of u-turn connections to the waypoint on both sides of the blockage to its internal road map. Allowing a U-turn on the opposite side of a blockage provided an adequate solution for NQE C; however, it can introduce a problem in a dynamic environment where a blockage may not be permanent, allowing a vehicle to plan a U-turn in the middle of a clear road.

Odin did have trouble detecting the stop sign blockage shown in Figure 18b. The perception module had never been presented with an obstacle that was not attached to the ground within the road area. The laser rangefinder sensor suite on

(a) (b)

Fig. 18. NQE course C contained (a) stop sign intersections and (b) road blockages

Odin had a narrow vertical field of view which caused difficulty perceiving this blockage. The IBEO sensors were barely able to detect the bottom of the stop signs attached to the gate while the vehicle was in motion. Once the signs were detected and the vehicle brakes were applied, the downward pitch of the vehicle caused the IBEOs to lose sight of the gate. The vehicle was then commanded to accelerate back up to speed, at which point the gate was seen again and the cycle repeated. The resulting behavior was that Odin crept forward towards the gate until the signs were detected by the close-range downward looking SICK rangefinders. At times the SICKs were barely detecting the stop signs, resulting in the processed stop sign object flickering in and out of existence. Because of this, it took the behavior module a significant amount of time to initiate a re-plan around the blockage. To resolve this issue the behavior software was modified to re-plan not only after constantly seeing a blockage for a constant period of time, but also after a blockage is seen in an area for a cumulative period of time.

4.4 Practice and Preparation

This section explains the VictorTango practice and preparation routine during the Urban Challenge events that helped make Odin successful.

4.4.1 Practice Areas

Team VictorTango always tried to maximize use of each practice timeslot. The team developed a number of RNDFs that replicated sections of the NQE RNDF for each of the practice areas. A detailed test plan was created for each practice. If problems arose, the goal was not to debug software, but to gather as much test data as possible for further analysis. This proved to be an extremely effective way to test given the short amount of time allowed for each practice block.

During the practice sessions a fair amount of dust collected on the lenses of the IBEO laser rangefinders. Although the team was allowed to clean sensors between missions, Odin's performance could have suffered until the vehicle returned due to this layer of dust. The IBEO sensors are capable of reading up to four returns per laser pulse in order to handle cases where small particles such as dust or precipitation cause the light reflection. After looking at the scan profiles for the

sensors while covered in dust, it was confirmed that a majority of the primary scan returns were contacting the dust resting on the lens. Even with the primary returns blocked, the sensors were still able to perceive all objects due to the multi-return feature. While running with dust on the lens is not ideal, Odin is able to continue without any reduction in perception capability until the lens can be cleaned.

4.4.2 Simulation

Simulation was a tool heavily used by team VictorTango during NQE and UCE preparation (the role of simulation in the entire software development process is discussed in section 6.2.2). In preparation for NQE and UCE, team VictorTango used an interactive simulator to load NQE & UCE RNDFs and dynamically add traffic vehicles or static obstacles. The team validated that Odin would be able to drive all areas of the RNDF before ever running Odin on the course and possibly wasting NQE run. For example, the planning software had trouble with the short road segments (less than 2 meters long) connecting the parking zones to the surrounding road segment. Software modifications to handle this previously untested RNDF style were validated in simulation.

If a failure occurred during an NQE run, a simulation scene could be created to match the environment in which the failure occurred. As software developers fixed problems, the test team had the manpower to run simulations in 5 parallel groups on laptop computers. This gave the software developers the luxury of concentrating on remaining software bugs while the test team exhaustively checked new software in a full range of tests ensuring there were no unintended side effects.

5 Urban Challenge Event

The most obvious accomplishment of the VictorTango team is that Odin finished the Urban Challenge competitively. This section highlights the successes and analyzes the incidents of Odin's performance in the Urban Challenge race.

5.1 Performance Overview

First out of the gate, Odin left the start zone and drove away at top speed to complete his first mission. Not knowing what to expect, the team eagerly looked on with people stationed at each of the viewing locations of the UCE course. Odin performed superbly, finishing in third place with no accidents or major errors. Odin's chase vehicle driver informed the team that Odin was a predictable and safe driver throughout the challenge. As the team watched the in car video, they saw that Odin navigated the roads and zones smoothly, made smart choices at intersections, and obeyed the rules of the road. The UCE did not demand as much obstacle avoidance and intersection navigation as the NQE. However, the endurance element and unpredictable nature of robot-on-robot interactions made it just as challenging.

5.2 Perception

This section presents the perception issues Odin faced during the Urban Challenge. Perception is defined to include all aspects of the design necessary to sense the environment and transform the raw data into information useful to the decision making software.

5.2.1 Localization Pops

During the UCE, Odin experienced a localization failure much like the one encountered during the NQE area B second attempt. Unlike the localization error during NQE, the position this time jumped to the north, causing Odin to compensate by driving onto the curb to the south. Fortunately, the position jump was not as severe. After a brief pause the localization solution returned to the correct position and Odin returned to the road and continued through the course.

Just as in the NQE, the data needed to diagnose the true cause of this error was not being logged. Although code changes could have been made to add this data to the localization logs, the team decided that this was an unnecessary code change and that there was not sufficient time to test for unexpected errors before the final competition.

5.2.2 IBEO Reset

A known problem with the IBEO sensors was that occasionally the internal ECU factory software would freeze, resulting in no scan or object data being transmitted to the classification software module. No specific cause could be identified for this problem; however it often occurred after the ECU software had been running for over four hours. To remedy this situation, a health monitoring routine had been built into the sensor interface code. When the interface fails to receive sensor data for a full second, it can stop Odin by reporting an error to the Health Monitoring module. If the connection is not restored within five seconds the power is cycled to the IBEO ECUs to reset the software. This reset takes approximately 90 seconds, and the vehicle will remain in a paused software state until sensor data is restored. During the 3rd mission of the UCE, this ECU software freeze occurred as Odin approached one of the four-way intersections. The software correctly identified this failure and cycled power to the sensors. After the reset time elapsed, Odin was able successfully proceed through the course without any human intervention.

5.2.3 Phantom Object

Early in the race, Odin travelled down the dirt road on the south east section of the RNDF. Without incident, Odin traversed the road up until the road began bending left before entering Phantom East. Odin slowed to a stop prior to making this last turn and waited for close to a minute. The forward spacing enforcer was keeping Odin from going further because Object Detection was reporting a dynamic obstacle ahead in the lane. Due to the method in which dynamic obstacles are calculated, the berm to Odin's left had a shape and height that appeared to be a vehicle. Had sparse lane detection been enabled for the race, Odin would have corrected the actual location of the road and Object Classification's road filter

would have surely eliminated the berm as a possible vehicle. However, since this was not in place, Odin was doomed to wait indefinitely. The vehicle was never classified as disabled, because the RNDF prohibited passing in a one-way road. Otherwise, Motion Planning would have easily navigated Odin beyond the phantom object. Fortunately, due to a small amount of GPS drift to the south the forward spacing enforcer allowed Odin to pass due to an acceptable clearance between Odin, the berm to the right, and the phantom object still in view. The conditions that allowed Odin to pass were similar to conditions experienced in the gauntlet with cars parked on the side of the road. In the end, had localization data been more accurate, Odin may have waited forever.

5.2.4 Road Detection

The cubic spline interpolation of the RNDF provided Odin with smooth paths to drive that followed the curvature of the roads exceptionally well. Driving directly to waypoints was not sufficient for navigating the Victorville RNDFs. There were only two instances during the Urban Challenge events where Odin drove over the curbs due to splining issues rather than localization errors. One example is in lane 8.1 which was the entry lane to the parking zones in the UCE RNDF. Figure 19 displays the Report Lane Position output (overlaid on upper lane). Based on the image, the spline follows the roads. However, the curb in the right turn of this s-curve was clipped every time Odin drove down this lane. This issue was due to tight geometry of the lane, the curvature control limitations of the chosen implementation of cubic splines, and human error in checking the RNDF preprocessing.

Fig. 19. Report Lane Position overlay for lane 8.1 of the UCE RNDF where Odin clipped a curb

5.3 Driving Behaviors

During the UCE, Driving Behaviors performed well, with no major issues. Below are detailed some of the interesting situations Odin encountered in the UCE.

5.3.1 Intersections

Odin handled intersections well in the UCE, according to logs, team observation, and the driver of Odin's chase vehicle. During several merge scenarios, Odin encountered traffic cars or other robots; however in less than 5% of 4-way stop scenarios did Odin have to respect precedence for cars arriving before him. It is

interesting to note that on several occasions, Odin observed traffic vehicles and chase cars roll through stop signs without ever stopping. On one occasion, after properly yielding precedence to Little Ben from U-Penn, Odin safely yielded to Little Ben's chase car, which proceeded out of turn at a stop sign despite arriving at the intersection after Odin.

5.3.2 Passing and Blocked Roads

At no time during the UCE did Odin encounter a disabled vehicle outside of a safety area or blocked road requiring a replan.

5.3.3 Parking Lot Navigation

Odin performed extremely well in the zone navigation/parking portions of the UCE; however the missions were very easy in comparison to pre-challenge testing performed by the team. While each of the three UCE missions contained only one parking spot (and the same one each time), Odin was prepared for much more complicated parking lot navigation. The team had anticipated missions with multiple consecutive parking checkpoints in the same zone but in different rows, requiring intelligent navigation from spot to spot, travelling down the accepted patterns (parking rows). A special strategy was even implemented to navigate parking lots with diagonally oriented spots, travelling down the rows in the correct direction.

While Odin was over-prepared for more complex parking lots, dynamic obstacle avoidance in zones was weak however, having seen far less testing. For this reason, the Route Planner gave zones a higher time penalty.

5.4 Motion Planning

One of Odin's key strengths in performance was smooth motion planning that maintained the maximum speed of a segment or a set global maximum of 10 m/s. The motion planning used on Odin was flexible enough to be used for all situations of the challenge such as road driving, parking and obstacle avoidance. By handing the problem of motion planning in a general sense, goals and weightings could be adjusted for new situations, but the core motion planning would ensure that no obstacles would be hit.

Reviewing race logs, there was one situation where more robust motion planning would have been required during the UCE. This occurred when Odin was traveling east on Montana and was cut-off by Stanford's Junior taking a left off of Virginia. Since Odin had the right-of-way, the motion planning algorithm would not slow down until Junior had mostly entered and been classified in Odin's lane. This situation was tested prior to competition, and Odin would have likely stopped without hitting Junior, but it would have been an abrupt braking maneuver. However, before the speed limiter of Motion Planning engaged, the safety driver paused Odin to prevent what was perceived as an imminent collision. More reliable classification of vehicle orientation and speed would allow motion planning to consider a wider range of traffic obstacles and apply brakes earlier in a similar situation.

6 Overall Successes

All of the teams in the Urban Challenge produced amazing results considering the scope of the project and short timeline. This section provides examples of the characteristics and tools of team VictorTango that increased productivity to accomplish the required tasks to successfully complete the Urban Challenge.

6.1 Base Vehicle Design

The Ford Escape hybrid platform used for Odin proved to be an excellent selection. The vehicle was large enough to accommodate all of the required equipment, but was not so large that tight courses posed maneuvering difficulties. The seamless integration of the drive-by-wire system proved highly reliable and was capable of responding quickly to motion planning commands. The power available from the hybrid system enabled the vehicle to easily power all systems and to run for more than 18 hours continuously. Good design combined with attention to detail in execution produced an autonomous vehicle that offered great performance and suffered very few hardware failures, enabling the team to focus testing time on software development.

6.2 Software Development

This section provides the features and tools used by the team for rapid software development, testing and debugging, and error handling. These attributes helped Odin emerge as a successful competitor in the Urban Challenge.

6.2.1 Software Architecture

During the initial planning and design phases of the project, team VictorTango spent a significant amount of time breaking down the Urban Challenge problem into specific areas. The resulting subset of problems then became the guiding force in the overall software architecture design. This yielded an extremely modular architecture and also ensured that a clear approach to all the main problems was addressed early on. Evidence of this foresight is the fact that the software architecture is almost exactly the same as it was originally conceived in early January of 2007. Slight modifications were made to some of the messages between software components, but in general, the original approach proved to be very successful.

Another benefit to the modular architecture was that it suited team VictorTango's structure very well. The size and scope of each software component could be developed by one or two principal software developers. Along with open communication channels and strong documentation, the team was able to tackle all of the Urban Challenge sub-problems in a methodical, efficient manner. Furthermore, by avoiding any sort of large, global solver, or all-encompassing artificial intelligence, team VictorTango's approach provided more flexibilty. The team was not pigeon-holed into any one approach and could find the best solution for a variety of problems. Finally, the modular architecture

prevented setbacks and unforeseen challenges from having debilitating effects as they might with less diverse systems.

Finally, the decision to implement the Joint Architecture for Unmanned Systems (JAUS) for inter-process communications in Odin offered several key advantages. Primarily, JAUS provided a structure for laying out software modules in a peer-to-peer, modular, automatically reconfigurable, and reusable fashion. It provided a framework for inter-process communication inside and across computing nodes, and a set of messages and rules for dynamic configuration, message routing, and data serving. Finally, the implementation of JAUS ensured that software developed under the Urban Challenge is reusable in future robotics projects, a critical step in accelerating the progress of unmanned systems.

6.2.2 Simulation

A key element of the software development was a custom developed simulation environment, shown in Figure 20 presenting Odin with a simulated road blockage. The simulator was used in all phases of software development, including: initial development, software validation, hardware-in-the-loop simulation, and even as a data visualization tool during vehicle runs. An easy to use simulator allowed software developers to obtain instant feedback on any software changes made as well as allowing other members of the team to stress test new software before deploying it to the vehicle.

Testing also involved more stringent validation milestones. Typically the first milestone for a new behavior involved a software validation. Software validations consisted of running software components on the final computer hardware with all perception messages being supplied by the simulator and all motion commands being sent to the simulator. These validations were run by the test team which would provide a set of different scenarios saved as separate scene files. After a successful software validation, some behaviors required hardware-in-the-loop simulation. In these simulations, the software was running on Odin in a test course and the software was configured to send motion commands to Odin as well as the simulator. The simulator would produce obstacle messages, allowing Odin to be tested against virtual obstacles and traffic eliminating any chance of a real collision during early testing. Lastly, in final validation, the simulator was used as a visualization tool during real-world testing.

6.2.3 Data Log Replay

Instrumental in diagnosing and addressing failures was a data logging and replay system integrated at the communications level. Called Déjà Vu, the system amounted to logging all input JAUS messages between modules for later playback in near real-time. During diagnostics, the component under analysis operated just as it did on the vehicle, only with the messages replayed from the Déjà Vu files. Additional features allowed logged data and Odin's position to be visualized in the simulator's 3D view as well.

Déjà Vu was critical in solving the problems encountered during NQE. In a typical scenario, the logged messages were played into the software as it ran in

Fig. 20. Screenshot of simulated Odin encountering a roadblock

source code form, allowing diagnostic breakpoints and probes to be used during playback. Once the problem had been diagnosed and a solution implemented, the new software was verified against the same logged data to verify the software made the intended decision.

Finally, by integrating Déjà Vu logging directly into the TORC JAUS Toolkit, it remained independent of the primary software module's functionality. As such, Déjà Vu is immediately reusable tool for future use in other projects.

7 Conclusions

Team VictorTango successfully completed the DARPA Urban Challenge final event, finishing 3rd, shown in Figure 21. During the competition, Odin was able to drive several hours without human intervention, negotiating stop sign intersections, merging into and across traffic, parking, and maintaining road speeds. Heightening the challenge was a very aggressive development timeline and a loosely defined problem allowing for many unknown situations. These factors made development efficiency as well as testing key components for success.

The aspect of the challenge that gave team VictorTango the most difficulty was environmental sensing. Unreliable sensing at longer distances was a major factor in limiting the vehicle maximum speed to 10 m/s. This speed limit and especially delays due to falsely perceived obstacles added significant time during the final event. The vertical field of view of sensors on the market today is a major limiting factor, especially in laser rangefinders. New technology such as the IBEO Alasca XT sensors and the Velodyne laser rangefinder are beginning to address these issues, but are relatively new products that are still undergoing significant design modifications. The use of prototype products in a timeline as short as the Urban Challenge introduces risk, as functions may be unreliable or settings may change.

Fig. 21. Odin crosses the finish line.

When evaluating the performance and design of vehicles participating in the DARPA Urban Challenge, it is important to consider the short development timeline of 18 months. Due to funding and organization, team VictorTango had closer to 14 months of actual development time. For example, road detection algorithms using vision were developed, and good results were achieved in certain conditions, but the team felt the software was not mature enough to handle all the possible cases within the scope of the urban challenge rules. A LIDAR road detection algorithm gave more consistent results over a wider variety of terrain, but had limited range requiring a reduction in travel speed. These results are more of a product of the development time and number of team members available to work on road sensing, rather than the limitations of the technology itself. With the short timeline, the team chose to use roads defined entirely defined by GPS, causing failures on at least 3 occasions during the race and NQE.

The Urban Challenge event demonstrated to the world that robot vehicles could interact with other robots and even humans in a complex environment. Team VictorTango has already received feedback from industry as well as military groups wanting to apply the technology developed in the Urban Challenge to their fields immediately.

Acknowledgements

This work was supported and made possible by DARPA track A funding and by the generous support of Caterpillar, Inc and Ford Motor Co. We would also like to thank National Instruments, NovaAtel, Omnistar, Black Box Corporation, Tripp Lite, Ibeo, Kairos Autonomi and Ultramotion for sponsorship or other support.

References

Avila-Garcıa, O., Hafner, E., Canamero, L.: Relating Behavior Selection Architectures to Environmental Complexity. In: Proc. Seventh Intl. Conf. on Simulation of Adaptive Behavior, MIT Press, Cambridge (2002)

Cacciola, S.J.: Fusion of Laser Range-Finding and Computer Vision Data for Traffic Detection by Autonomous Vehicles. Master's Thesis. Virginia Tech., Blacksburg, VA (2007)

Currier, P.N.: Development of an Automotive Ground Vehicle Platform for Autonomous Urban Operations. Master's Thesis. Virginia Tech, Blacksburg, VA (2008)

Duda, R.O., Hart, P.E.: Use of the Hough Transform to Detect Lines and Curves in Pictures. Commun. ACM 15(1), 11–15 (1972)

Eren, H., Fung, C.C., Evans, J.: Implementation of the Spline Method for Mobile Robot Path Control. In: Piuri, V., Savino, M. (eds.) Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference, vol. 2, pp. 739–744. IEEE, Venice (1999)

Fuerstenberg, K.C., Dietmayer, K.C.J., Lages, U.: Laserscanner Innovations for Detection of Obstacles and Road. In: Proceedings of 7th International Conference on Advanced Microsystems for Automotive Applications, Berlin, Germany (2003)

Fuerstenberg, K.C., Linzmeier, D.T., Dietmayer, K.C.J.: Pedestrian Recognition and Tracking of Vehicles using a Vehicle Based Multilater Laserscanner. In: Proceedings of 10th World Congress on Intelligent Transport Systems, Madrid, Spain (2003)

Hart, P.E., Nilsson, N.J., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC 4(2), 100–107 (1968)

Hurdus, J.G.: A Portable Approach to High-Level Behavioral Programming for Complex Autonomous Robot Applications. Master's Thesis. Virginia Tech, Blacksburg, VA (2008)

Kelly, A.J.: A 3D State Space Formulation of a Navigation Kalman Filter for Autonomous Vehicles. CMU Robotics Institute Technical Report CMU-RI-TR-94-19 (1994)

Konolige, K., Myers, K.: The Saphira Architecture for Autonomous Mobile Robots. In: Kortenkamp, D., Bonasson, R., Murphy, R. (eds.) Artificial Intelligence and Mobile Robots. MIT Press, Cambridge (1998)

Lacaze, A., Moscovitz, Y., DeClaris, N., Murphy, K. (1998). Path Planning for Autonomous Vehicles Driving Over Rough Terrain. In: Proceedings of the ISIC/CIRA/ISAS Conference. Gaithersburg, MD, September 14-17 (1998)

Maes, P.: How To Do the Right Thing.Technical Report NE 43–836, AI Laboratory. MIT, Cambridge (1989)

Milliken, W.F., Milliken, D.L.: Race Car Vehicle Dynamics. SAE International, Warrendale, PA (1995)

Murphy, R.R.: Introduction to AI Robotics. MIT Press, Cambridge (2000)

Pirjanian, P.: Multiple Objective Behavior-Based Control. Robotics and Autonomous Systems 31(1), 53–60 (2000)

Pirjanian, P.: Behavior Coordination Mechanisms – State-of-the-Art. Tech Report IRIS-99-375, Institute for Robotics and Intelligent Systems, University of Southern California, Los Angeles, California (1999)

Rosenblatt, J.: DAMN: A Distributed Architecture for Mobile Navigation. In: AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents, Stanford, CA. AAAI Press, Menlo Park (1995)

Russel, S., Norvig, P.: Artificial Intelligence – A Modern Approach. Pearson Education, Inc., Upper Saddle River (2003)

Thrun, S., Montemerlo, M., et al.: Stanley: The robot that won the DARPA Grand Challenge: Research Articles. Journal of Field Robotics 23(9), 661–692 (2006)

Urmson, C., et al.: A Robust Approach to High-Speed Navigation for Unrehearsed Desert Terrain. Journal of Field Robotics 23(8), 467 (2006)

A Perception-Driven Autonomous Urban Vehicle

John Leonard[1], Jonathan How[1], Seth Teller[1], Mitch Berger[1], Stefan Campbell[1], Gaston Fiore[1], Luke Fletcher[1], Emilio Frazzoli[1], Albert Huang[1], Sertac Karaman[1], Olivier Koch[1], Yoshiaki Kuwata[1], David Moore[1], Edwin Olson[1], Steve Peters[1], Justin Teo[1], Robert Truax[1], Matthew Walter[1], David Barrett[2], Alexander Epstein[2], Keoni Maheloni[2], Katy Moyer[2], Troy Jones[3], Ryan Buckley[3], Matthew Antone[4], Robert Galejs[5], Siddhartha Krishnamurthy[5], and Jonathan Williams[5]

[1] MIT, Cambridge, MA 02139
jleonard@mit.edu
[2] Franklin W. Olin College
Needham, MA 02492
david.barrett@olin.edu
[3] Draper Laboratory
Cambridge, MA 02139
tbjones@draper.com
[4] BAE Systems Advanced Information Technologies
Burlington, MA 01803
matthew.antone@baesystems.com
[5] MIT Lincoln Laboratory
Lexington, MA 02420
galejs@ll.mit.edu

Abstract. This paper describes the architecture and implementation of an autonomous passenger vehicle designed to navigate using locally perceived information in preference to potentially inaccurate or incomplete map data. The vehicle architecture was designed to handle the original DARPA Urban Challenge requirements of perceiving and navigating a road network with segments defined by sparse waypoints. The vehicle implementation includes many heterogeneous sensors with significant communications and computation bandwidth to capture and process high-resolution, high-rate sensor data. The output of the comprehensive environmental sensing subsystem is fed into a kino-dynamic motion planning algorithm to generate all vehicle motion. The requirements of driving in lanes, three-point turns, parking, and maneuvering through obstacle fields are all generated with a unified planner. A key aspect of the planner is its use of closed-loop simulation in a Rapidly-exploring Randomized Trees (RRT) algorithm, which can randomly explore the space while efficiently generating smooth trajectories in a dynamic and uncertain environment. The overall system was realized through the creation of a powerful new suite of software tools for message-passing, logging, and visualization. These innovations provide a strong platform for future research in autonomous driving in GPS-denied and highly dynamic environments with poor *a priori* information.

1 Introduction

In November 2007 the Defense Advanced Research Projects Agency (DARPA) conducted the DARPA Urban Challenge Event (UCE), which was the third in a series of competitions designed to accelerate research and development of full-sized

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 163–230.
springerlink.com

Fig. 1. Talos in action at the National Qualifying Event.

autonomous road vehicles for the Defense forces. The competitive approach has been very successful in porting a substantial amount of research (and researchers) from the mobile robotics and related disciplines into autonomous road vehicle research (DARPA, 2007). The UCE introduced an urban scenario and traffic interactions into the competition. The short aim of the competition was to develop an autonomous vehicle capable of passing the California driver's test (DARPA, 2007). The 2007 challenge was the first in which automated vehicles were required to obey traffic laws including lane keeping, intersection precedence, passing, merging and maneuvering with other traffic.

The contest was held on a closed course within the decommissioned George Air force base. The course was predominantly the street network of the residential zone of the former Air force base with several graded dirt roads added for the contest. Although all autonomous vehicles were on the course at the same time, giving the competition the appearance of a conventional race, each vehicles was assigned individual missions. These missions were designed by DARPA to require each team to complete 60 miles within 6 hours to finish the race. In this race against time, penalties for erroneous or dangerous behavior were converted into time penalties. DARPA provided all teams with a single Route Network Definition File (RNDF) 24 hours before the race. The RNDF was very similar to a digital street map used by an in-car GPS navigation system. The file defined the road positions, number of lanes, intersections, and even parking space locations in GPS coordinates. On the day of the race each team was provided with a second unique file called a Mission Definition File (MDF). This file consisted solely of a list of checkpoints (or locations) within the RNDF which the vehicle was required to cross. Each vehicle competing in the UCE was required to complete three missions, defined by three separate MDFs.

Team MIT developed an urban vehicle architecture for the UCE. The vehicle (shown in action in Figure 1) was designed to use locally perceived information

in preference to potentially inaccurate map data to navigate a road network while obeying the road rules. Three of the key novel features of our system are: (1) a perception-based navigation strategy; (2) a unified planning and control architecture, and (3) a powerful new software infrastructure. Our system was designed to handle the original race description of perceiving and navigating a road network with a sparse description, enabling us to complete national qualifying event (NQE) Area B without augmenting the RNDF. Our vehicle utilized a powerful and general-purpose Rapidly-exploring Randomized Tree (RRT)-based planning algorithm, achieving the requirements of driving in lanes, executing three-point turns, parking, and maneuvering through obstacle fields with a single, unified approach. The system was realized through the creation of a powerful new suite of software tools for autonomous vehicle research, which our team has made available to the research community. These innovations provide a strong platform for future research in autonomous driving in GPS-denied and highly dynamic environments with poor *a priori* information. Team MIT was one of thirty-five teams that participated in the DARPA Urban Challenge NQE, and was one of eleven teams to qualify for the UCE based on our performance in the NQE. The vehicle was one of six to complete the race, finishing in fourth place.

This paper reviews the design and performance of Talos, the MIT autonomous vehicle. Section 2 summarizes the system architecture. Section 3 describes the design of the race vehicle and software infrastructure. Sections 4 and 5 explain the set of key algorithms developed for environmental perception and motion planning for the Challenge. Section 6 describes the performance of the integrated system in the qualifier and race events. Section 7 reflects on how the perception-driven approach fared by highlighting some successes and lessons learned. Section 8 provides details on the public release of our team's data logs, interprocess communications and image acquisition libraries, and visualization software. Finally, Section 9 concludes the paper.

2 Architecture

Our overall system architecture (Figure 2) includes the following subsystems:

- The **Road Paint Detector** uses two different image-processing techniques to fit splines to lane markings from the camera data.
- The **Lane Tracker** reconciles digital map (RNDF) data with lanes detected by vision and lidar to localize the vehicle in the road network.
- The **Obstacle Detector** uses Sick and Velodyne lidar to identify stationary and moving obstacles.
- The low-lying **Hazard Detector** uses downward looking lidar data to assess the drivability of the road ahead and to detect curb cuts.
- The **Fast Vehicle** detector uses millimeter wave radar to detect fast approaching vehicles in the medium to long range.
- The **Positioning** module estimates the vehicle position in two reference frames. The local frame is an integration of odometry and Inertial Measurement Unit

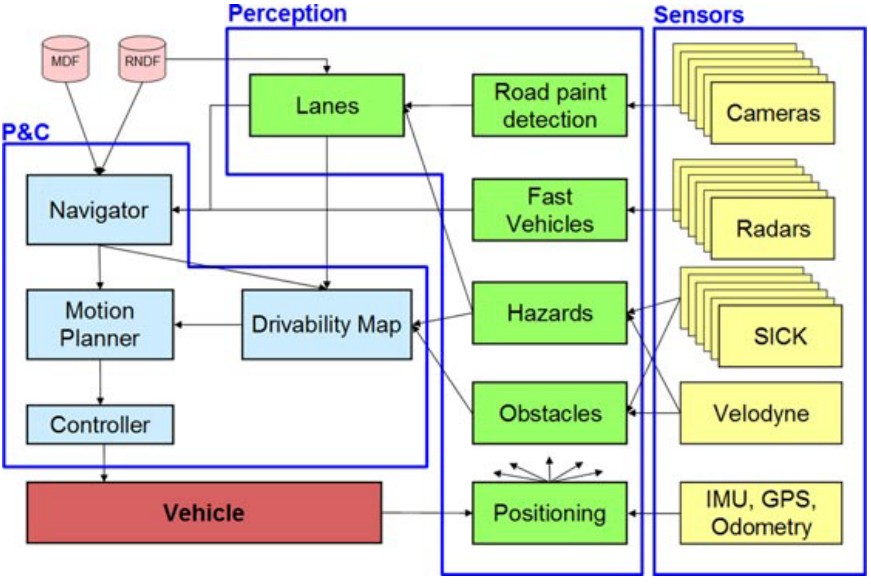


Fig. 2. System Architecture.

(IMU) measurements to estimate the vehicle's egomotion through the local environment. The global coordinate transformation estimates the correspondence between the local frame and the GPS coordinate frame. GPS outages and odometry drift will vary this transformation. Almost every module listens to the Positioning module for egomotion correction or path planning.

- The **Navigator** tracks the mission state and develops a high-level plan to accomplish the mission based on the RNDF and MDF. The output of the robust minimum-time optimization is a short-term goal location provided to the *Motion Planner*. As progress is made the short-term goal is moved, like a carrot in front of a donkey, to the achieve the mission.
- The **Drivability Map** provides an efficient interface to perceptual data, answering queries from the *Motion Planner* about the validity of potential motion paths. The *Drivability Map* is constructed using perceptual data filtered by the current constraints specified by the *Navigator*.
- The **Motion Planner** identifies, then optimizes, a kino-dynamically feasible vehicle trajectory that moves towards the goal point selected by the *Navigator* using the constraints given by the situational awareness embedded in the *Drivability Map*. Uncertainty in local situational awareness is handled through rapid replanning and constraint tightening. The *Motion Planner* also explicitly accounts for vehicle safety, even with moving obstacles. The output is a desired vehicle trajectory, specified as an ordered list of waypoints (position, velocity, headings) that are provided to the low-level motion *Controller*.

- The **Controller** executes the low-level motion control necessary to track the desired paths and velocity profiles issued by the *Motion Planner*.

These modules are supported by a powerful and flexible software architecture based on a new lightweight UDP message passing system (described in Section 3.3). This new architecture facilitates efficient communication between a suite of asynchronous software modules operating on the vehicle's distributed computer system. The system has enabled the rapid creation of a substantial code base, currently approximately 140,000 source lines of code, that incorporates sophisticated capabilities, such as data logging, replay, and 3-D visualization of experimental data.

3 Infrastructure Design

Achieving an autonomous urban driving capability is a difficult multi-dimensional problem. A key element of the difficulty is that significant *uncertainty* occurs at multiple levels: in the *environment*, in *sensing*, and in *actuation*. Any successful strategy for meeting this challenge must address all of these sources of uncertainty. Moreover, it must do so in a way that is *scalable* to spatially extended environments, and efficient enough for *real-time* implementation on a rapidly moving vehicle.

The difficulty in developing a solution that can rise to these intellectual challenges is compounded by the many unknowns in the system design process. Despite DARPA's best efforts to define the rules for the UCE in detail well in advance of the race, there was huge potential variation in the difficulty of the final event. It was difficult at the start of the project to conduct a single analysis of the system that could be translated to one static set of system requirements (for example, to predict how different sensor suites would perform in actual race conditions). For this reason, Team MIT chose to follow a spiral design strategy, developing a flexible vehicle design and creating a system architecture that could respond to an evolution of the system requirements over time, with frequent testing and incremental addition of new capabilities as they become available.

Testing "early and often" was a strong recommendation of successful participants in the 2005 Grand Challenge (Thrun et al., 2006; Urmson et al., 2006; Trepagnier et al., 2006). As newcomers to the Grand Challenge, it was imperative for our team to obtain an autonomous vehicle as quickly as possible. Hence, we chose to build a prototype vehicle very early in the program, while concurrently undertaking the more detailed design of our final race vehicle. As we gained experience from continuous testing with the prototype, the lessons learned were incorporated into the overall architecture and our final race vehicle.

The spiral design strategy has manifested itself in many ways – most dramatically in our decision to build two (different) autonomous vehicles. We acquired our prototype vehicle, a Ford Escape, at the outset of the project, to permit early autonomous testing with a minimal sensor suite. Over time we increased the frequency of tests, added more sensors, and brought more software capabilities online to meet a larger set of requirements. In parallel with this, we procured and fabricated our race vehicle Talos, a Land Rover LR3. Our modular and flexible software architecture was

designed to enable a rapid transition from one vehicle to the other. Once the final race vehicle became available, all work on the prototype vehicle was discontinued, as we followed the adage to "build one system to throw it away".

3.1 Design Considerations

We employed several key principles in designing our system.

Use of many sensors. We chose to use a large number of low-cost, unactuated sensors, rather than to rely exclusively on a small number of more expensive, high-performance sensors. This choice produced the following benefits:

- By avoiding any single point of sensor failure, the system is more robust. It can tolerate loss of a small number of sensors through physical damage, optical obscuration, or software failure. Eschewing actuation also simplified the mechanical, electrical and software systems.
- Since each of our many sensors can be positioned at an extreme point on the car, more of the car's field of view (FOV) can be observed. A single sensor, by contrast, would have a more limited FOV due to unavoidable occlusion by the vehicle itself. Deploying many single sensors also gave us increased flexibility as designers. Most points in the car's surroundings are observed by at least one of each of the three exteroceptive sensor types. Finally, our multi-sensor strategy also permits more effective distribution of I/O and CPU bandwidth across multiple processors.

Minimal reliance on GPS. We observed from our own prior research, and other teams' prior Grand Challenge efforts, that GPS cannot be relied upon for high-accuracy localization at all times. That fact, along with the observation that humans do not need GPS to drive well, led us to develop a navigation and perception strategy that uses GPS only when absolutely necessary, i.e., to determine the general direction to the next waypoint, and to make forward progress in the (rare) case when road markings and boundaries are undetectable. One key outcome of this design choice is our "local frame" situational awareness, described more fully in Section 4.1.

Fine-grained, high-bandwidth CPU, I/O and network resources. Given the short time (18 months, from May 2006 to November 2007) available for system development, our main goal was simply to get a first pass at every required module working, and working solidly, in time to qualify. Thus we knew at the outset that we could not afford to invest effort in premature optimization, i.e., performance profiling, memory tuning, etc. This led us to the conclusion that we should have many CPUs, and that we should lightly load each machine's CPU and physical memory (say, at half capacity) to avoid non-linear systems effects such as process or memory thrashing. A similar consideration led us to use a fast network interconnect, to avoid operating regimes in which network contention became non-negligible. The downside of our choice of many machines was a high power budget, which required an external generator on the car. This added mechanical and electrical complexity to the system, but the computational flexibility that was gained justified this effort.

Asynchronous sensor publish and update; minimal sensor fusion. Our vehicle has sensors of six different types (odometry, inertial, GPS, lidar, radar, vision), each type generating data at a different rate. Our architecture dedicates a software driver to each individual sensor. Each driver performs minimal processing, then publishes the sensor data on a shared network. A "drivability map" API (described more fully below) performs minimal sensor fusion, simply by depositing interpreted sensor returns into the map on an "as-sensed" (just in time) basis.

"Bullet proof" low-level control. To ensure that the vehicle was always able to make progress, we designed the low-level control using very simple, well proven algorithms that involved no adaptation or mode switching. These control add-ons might give better performance, but they are difficult to verify and validate. The difficulty being that a failure in this low-level control system would be critical and it is important that the motion planner always be able to predict the state of the controller/vehicle with a high degree of confidence.

Strong reliance on simulation. While field testing is paramount, it is time consuming and not always possible. Thus we developed multiple simulations that interfaced directly with the vehicle code that could be used to perform extensive testing of the software and algorithms prior to testing them on-site.

3.2 Race Vehicle Configuration

The decision to use two different types of cars (the Ford Escape and Land Rover LR3) entailed some risk, but given the time and budgetary constraints, this approach had significant benefits. The spiral design approach enabled our team to move quickly up the learning curve and accomplish many of our "milestone 2" site visit requirements before mechanical fabrication of the race vehicle was complete.

Size, power and computation were key elements in the design of the vehicle. For tasks such as parking and the execution of U-turns, a small vehicle with a tight turning radius was ideal. Given the difficulty of the urban driving task, and our desire to use many inexpensive sensors, Team MIT chose a large and powerful computer

(a) (b)

Fig. 3. Developed vehicles. (a) Ford Escape rapid prototype. (b) Talos, our Land Rover LR3 race vehicle featuring five cameras, 15 radars 12 Sick lidars and a Velodyne lidar.

system. As mentioned above, this led our power requirements to exceed the capabilities of aftermarket alternator solutions for our class of vehicles, necessitating the use of a generator.

Our initial design aim to use many inexpensive sensors was modified substantially midway through the project when resources became available to purchase a Velodyne HDL-64 3D lidar. The Velodyne played a central role for the tasks of vehicle and hazard detection in our final configuration.

The Land Rover LR3 provided a maneuverable and robust platform for our race vehicle. We chose this vehicle for its excellent maneuverability and small turning radius and large payload capacity. Custom front and roof fixtures were fitted, permitting sensor positions to be tuned during system development. Wherever possible the fixtures were engineered to protect the sensors from collisions.

The stock vehicle was integrated with the following additional components:

- Electronic Mobility Controls (EMC) drive-by-wire system (AEVIT)
- Honda EVD6010 internal power generator
- 2 Acumentrics uninterruptible power supplies
- Quanta blade server computer system (the unbranded equivalent of Fujitsu Primergy BX600)
- Applanix POS-LV 220 GPS/INS
- Velodyne HDL-64 lidar
- 12 Sick lidars
- 5 Point Grey Firefly MV Cameras
- 15 Delphi Radars

The first step in building the LR3 race vehicle was adapting it for computer-driven control. This task was outsourced to Electronic Mobility Controls in Baton Rouge, Louisiana. They installed computer-controlled servos on the gear shift, steering column, and a single servo for throttle and brake actuation. Their system was designed for physically disabled drivers, but was adaptable for our needs. It also provided a proven and safe method for switching from normal human-driven control to autonomous control.

Safety of the human passengers was a primary design consideration in integrating the equipment into the LR3. The third row of seats in the LR3 was removed, and the entire back end was sectioned off from the main passenger cabin by an aluminum and Plexiglas wall. This created a rear “equipment bay” which held the computer system, the power generator, and all of the power supplies, interconnects, and network hardware for the car. The LR3 was also outfitted with equipment and generator bay temperature readouts, a smoke detector, and a passenger cabin carbon monoxide detector.

The chief consumer of electrical power was the Quanta blade server. The server required 240V as opposed to the standard 120V and could consume up to 4000Watts, dictating many of the power and cooling design decisions. Primary power for the system came from an internally mounted Honda 6000 Watt R/V generator. It draws fuel directly from the LR3 tank and produces 120 and 240VAC at 60 Hz. The generator was installed in a sealed aluminum enclosure inside the equipment bay; cooling

air is drawn from outside, and exhaust gases leave through an additional muffler under the rear of the LR3.

The 240VAC power is fed to twin Acumentrics rugged UPS 2500 units which provide backup power to the computer and sensor systems. The remaining generator power is allocated to the equipment bay air conditioning (provided by a roof-mounted R/V air conditioner) and non-critical items such as back-seat power outlets for passenger laptops.

3.2.1 Sensor Configuration

As mentioned, our architecture is based on the use of many different sensors, based on multiple sensing modalities. We positioned and oriented the sensors so that most points in the vehicle's surroundings would be observed by at least one sensor of each type: lidar, radar, and vision. This redundant coverage gave robustness against both type-specific sensor failure (e.g., difficulty with vision due to low sun angle) or individual sensor failure (e.g., due to wiring damage).

We selected the sensors with several specific tasks in mind. A combination of "skirt" (horizontal Sick) 2-D lidars mounted at a variety of heights, combined with the output from a Velodyne 3-D lidar, performs close-range obstacle detection. "Pushbroom" (downward-canted Sick) lidars and the Velodyne data detect drivable surfaces. Out past the lidar range, millimeter wave radar detects fast approaching vehicles. High-rate forward video, with an additional rear video channel for higher-confidence lane detection, performs lane detection.

Ethernet interfaces were used to deliver sensor data to the computers for most devices. Where possible, sensors were connected as ethernet devices. In contrast to many commonly used standards such as RS-232, RS-422, serial, CAN, USB or Firewire, ethernet offers, in one standard: electrical isolation, RF noise immunity, reasonable physical connector quality, variable data rates, data multiplexing, scalability, low latencies and large data volumes.

The principal sensor for obstacle detection is the Velodyne HDL-64, which was mounted on a raised platform on the roof. High sensor placement was necessary to raise the field of view above the Sick lidar units and the air conditioner. The velodyne is a 3D laser scanner comprised of 64 lasers mounted on a spinning head. It produces approximately a million range samples per second, performing a full 360 degree sweep at 15Hz.

The Sick lidar units (all model LMS 291-S05) served as the near-field detection system for obstacles and the road surface. On the roof rack there are five units angled down viewing the ground ahead of the vehicle, while the remaining seven are mounted lower around the sides of the vehicle and project outwards parallel to the ground plane.

Each Sick sensor generates an interlaced scan of 180 planar points at a rate of 75Hz. Each of the Sick lidar units has a serial data connection which is read by a MOXA NPort-6650 serial device server. This unit, mounted in the equipment rack above the computers, takes up to 16 serial data inputs and outputs TCP/IP link.

The Applanix POS-LV navigation solution was used to for world-relative position and orientation estimation of the vehicle. The Applanix system combines differential GPS, a one degree of drift per hour rated IMU and a wheel encoder to estimate the vehicle's position, orientation, velocity and rotation rates. The position information was used to determine the relative position of RNDF GPS waypoints to the vehicle. The orientation and rate information were used to estimate the vehicle's local motion over time. The Applanix device is interfaced via a TCP/IP link.

Delphi's millimeter wave OEM automotive Adaptive Cruise Control radars were used for long-range vehicle tracking. The narrow field of view of these radars (around 18°) required a tiling of 15 radars to achieve the desired 240° field of view. The radars require a dedicated CAN bus interface each. To support 15 CAN bus networks we used 8 internally developed CAN to ethernet adaptors (EthCANs). Each adaptor could support two CAN buses.

Five Point Grey Firefly MV color cameras were used on the vehicle, providing close to a 360° field of view. Each camera was operated at 22.8 Hz and produced Bayer-tiled images at a resolution of 752x480. This amounted to 39 MB/s of image data, or 2.4 GB/min. To support multiple parallel image processing algorithms, camera data was JPEG-compressed and then re-transmitted over UDP multicast to other computers (see Section 3.3.1). This allowed multiple image processing and logging algorithms to operate on the camera data in parallel with minimal latency.

The primary purpose of the cameras was to detect road paint, which was then used to estimate and track lanes of travel. While it is not immediately obvious that rearward-facing cameras are useful for this goal, the low curvature of typical urban roads means that observing lanes behind the vehicle greatly improves forward estimates.

The vehicle state was monitored by listening to the vehicle CAN bus. Wheel speeds, engine RPM, steering wheel position and gear selection were monitored using a CAN to Ethernet adaptor (EthCAN).

3.2.2 Autonomous Driving Unit

The final link between the computers and the vehicle was the Autonomous Driving Unit (ADU). In principle, it was an interface to the drive-by-wire system that we purchased from EMC. In practice, it also served a critical safety role.

The ADU was a very simple piece of hardware running a real-time operating system, executing the control commands passed to it by the non-real-time computer cluster. The ADU incorporated a watchdog timer that would cause the vehicle to automatically enter PAUSE state if the computer generated either invalid commands or if the computer stopped sending commands entirely.

The ADU also implemented the interface to the buttons and displays in the cabin, and the DARPA-provide E-Stop system. The various states of the vehicle (PAUSE, RUN, STANDBY, E-STOP) were managed in a state-machine within the ADU.

3.3 Software Infrastructure

We developed a powerful and flexible software architecture based on a new lightweight UDP message passing system. Our system facilitates efficient communication between a suite of asynchronous software modules operating on the vehicle's distributed computer system. This architecture has enabled the rapid creation of a substantial code base that incorporates data logging, replay, and 3-D visualization of all experimental data, coupled with a powerful simulation environment.

3.3.1 Lightweight Communications and Marshalling

Given our emphasis on perception, existing interprocess communications infrastructures such as CARMEN (Thrun et al., 2006) or MOOS (Newman, 2003) were not sufficient for our needs. We required a low-latency, high-throughput communications framework that scales to many senders and receivers. After our initial assessment of existing technologies, we designed and implemented an interprocess communications system that we call *Lightweight Communications and Marshaling* (LCM).

LCM is a minimalist system for message passing and data marshaling, targeted at real-time systems where latency is critical. It provides a publish/subscribe message passing model and an XDR-style message specification language with bindings for applications in C, Java, and Python. Messages are passed via UDP multicast on a switched local area network. Using UDP multicast has the benefit that it is highly scalable; transmitting a message to a thousand subscribers uses no more network bandwidth than does transmitting it to one subscriber.

We maintained two physically separate networks for different types of traffic. The majority of our software modules communicated via LCM on our primary network, which sustained approximately 8 MB/s of data throughout the final race. The secondary network carried our full resolution camera data, and sustained approximately 20 MB/s of data throughout the final race.

While there certainly was some risk in creating an entirely new interprocess communications infrastructure, the decision was consistent with our team's overall philosophy to treat the DARPA Urban Challenge first and foremost as a *research* project. The decision to develop LCM helped to create a strong sense of ownership amongst the key software developers on the team. The investment in time required to write, test, and verify the correct operation of the LCM system paid off for itself many times over, by enabling a much faster development cycle than could have been achieved with existing interprocess communication systems. LCM is now freely available as a tool for widespread use by the robotics community.

The design of LCM, makes it very easy to create logfiles of all messages transmitted during a specific window of time. The logging application simply subscribes to every available message channel. As messages are received, they are timestamped and written to disk.

To support rapid data analysis and algorithmic development, we developed a log playback tool that reads a logfile and retransmits the messages in the logfile back over the network. Data can be played back at various speeds, for skimming or careful analysis. Our development cycle frequently involved collecting extended datasets

with our vehicle and then returning to our offices to analyze data and develop algorithms. To streamline the process of analyzing logfiles, we implemented a user interface in our log playback tool that supported a number of features, such as randomly seeking to user-specified points in the logfile, selecting sections of the logfile to repeatedly playback, extracting portions of a logfile to a separate smaller logfile and the selected playback of message channels.

3.3.2 Visualization

The importance of visualizing sensory data as well as the intermediate and final stages of computation for any algorithm cannot be overstated. While a human observer does not always know exactly what to expect from sensors or our algorithms, it is often easy for a human observer to spot when something is wrong. We adopted a mantra of "Visualize Everything" and developed a visualization tool called the *viewer*. Virtually every software module transmitted data that could be visualized in our viewer, from GPS pose and wheel angles to candidate motion plans and tracked vehicles. The viewer quickly became our primary means of interpreting and understanding the state of the vehicle and the software systems.

Debugging a system is much easier if data can be readily visualized. The LCGL library was a simple set of routines that allowed any section of code in any process on any machine to include in-place OpenGL operations; instead of being rendered, these operations were recorded and sent across the LCM network (hence LCGL), where they could be rendered by the viewer.

LCGL reduced the amount of effort to create a visualization to nearly zero, with the consequence that nearly all of our modules have useful debugging visualizations that can be toggled on and off from the viewer.

3.3.3 Process Manager and Mission Manager

The distributed nature of our computing architecture necessitated the design and implementation of a process management system, which we called *procman*. This provided basic failure recovery mechanisms such as restarting failed or crashed processes, restarting processes that have consumed too much system memory, and monitoring the processor load on each of our servers.

To accomplish this task, each server ran an instance of a *procman deputy*, and the operating console ran the only instance of a *procman sheriff*. As their names suggest, the user issues process management commands via the sheriff, which then relays commands to the deputies. Each deputy is then responsible for managing the processes on its server independent of the sheriff and other deputies. Thus, if the sheriff dies or otherwise loses communication with its deputies, the deputies continue enforcing their last received orders.

Messages passed between sheriffs and deputies are stateless, and thus it is possible to restart the sheriff or migrate it across servers without interrupting the deputies.

The mission manager interface provided a minimalist user interface for loading, launching, and aborting missions. This user interface was designed to minimize the potential for human error during the high-stress scenarios typical on qualifying runs

and race day. It did so by running various "sanity checks" on the human-specified input, displaying only information of mission-level relevance, and providing a minimal set of intuitive and obvious controls. Using this interface, we routinely averaged well under one minute from the time we received the MDF from DARPA officials to having our vehicle in pause mode and ready to run.

4 Perception Algorithms

Team MIT implemented a sensor rich design for the Talos vehicle. This section describes the algorithms used to process the sensor data. Specifically the Local Frame, Obstacle Detector, Hazard Detector and Lane Tracking modules.

4.1 The Local Frame

The *Local Frame* is a smoothly varying coordinate frame into which sensor information is projected. We do not rely directly on the GPS position output from the Applanix because it is subject to sudden position discontinuities upon entering or leaving areas with poor GPS coverage. We integrate the velocity estimates from the Applanix to get position in the local frame.

The local frame is a Euclidean coordinate system with arbitrary origin. It has the desirable property that the vehicle always moves smoothly through this coordinate system—in other words, it is very accurate over short time scales but may drift relative to itself over longer time scales. This property makes it ideal for registering the sensor data for the vehicle's immediate environment. An estimate of the coordinate transformation between the local frame and the GPS reference frame is updated continuously. This transformation is only needed when projecting a GPS feature, such as an RNDF waypoint, into the local frame. All other navigation and perceptual reasoning is performed directly in the local frame.

A single process is responsible for maintaining and broadcasting the vehicle's pose in the local frame (position, velocity, acceleration, orientation, and turning rates) as well as the most recent local-to-GPS transformation. These messages are transmitted at 100Hz.

4.2 Obstacle Detector

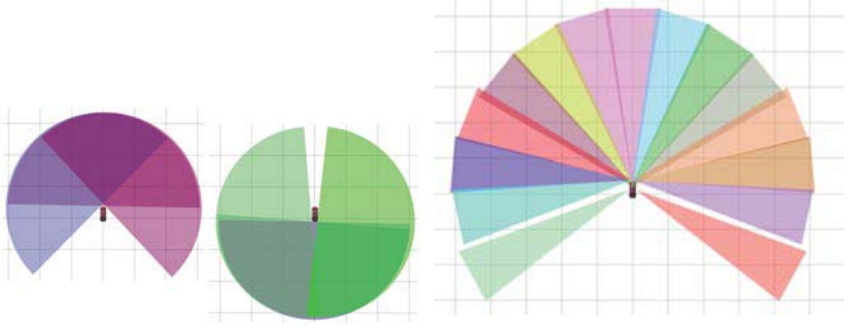
The system's large number of sensors provided a comprehensive field-of-view and provided redundancy both within and across sensor modalities. Lidars provided near-field obstacle detection (Section 4.2.1), while radars provided awareness of moving vehicles in the far field (Section 4.2.7).

Much previous work in automotive vehicle tracking has used computer vision for detecting other cars and other moving objects, such as pedestrians. Of the work in the vision literature devoted to tracking vehicles, the techniques developed by Stein and collaborators (Stein et al., 2000; Stein et al., 2003) are notable because this work provided the basis for the development of a commercial product – the

Mobileye automotive visual tracking system. We evaluated the Mobileye system; it performed well for tracking vehicles at front and rear aspects during highway driving, but did not provide a solution that was general enough for the high-curvature roads, myriad aspect angles and cluttered situations encountered in the Urban Challenge. An earlier effort at vision-based obstacle detection (but not velocity estimation) employed custom hardware (Bertozzi, 1998).

A notable detection and tracking system for urban traffic from lidar data was developed by Wang *et al.*, who incorporated dynamic object tracking in a 3D SLAM system (Wang, 2004). Data association and tracking of moving objects was a prefilter for SLAM processing, thereby reducing the effects of moving objects in corrupting the map that was being built. Object tracking algorithms used the interacting multiple model (IMM) (Blom and Bar-Shalom, 1988) for probabilistic data association. A related project addressed the tracking of pedestrians and other moving objects to develop a collision warning system for city bus drivers (Thorpe et al., 2005).

Each UCE team required a method for detecting and tracking other vehicles. The techniques of the Stanford Racing Team (Stanford Racing Team, 2007) and the Tartan Racing Team (Tartan Racing Team, 2007) provide alternative examples of successful approaches. Tartan Racing's vehicle tracker built on the algorithm of Mertz *et al.* (Mertz et al., 2005), which fits lines to lidar returns and estimates convex corners from the laser data to detect vehicles. The Stanford team's object tracker has similarities to the Team MIT approach. It is based first on filtering out vertical obstacles and ground plane measurements, as well as returns from areas outside of the Route Network Definition File. The remaining returns are fit to 2-D rectangles using particle filters, and velocities are estimated for moving objects (Stanford Racing Team, 2007). Unique aspects of our approach are the concurrent processing of lidar and radar data and a novel multi-sensor calibration technique.



(a) (b)

Fig. 4. Sensor fields of view (20m grid size). (a) Our vehicle used seven horizontally-mounted 180° planar lidars with overlapping fields of view. The 3 lidars at the front and the 4 lidars at the back are drawn separately so that the overlap can be more easily seen. The ground plane and false positives are rejected using consensus between lidars. (b) Fifteen 18° radars yield a wide field of view.

Our obstacle detection system combines data from 7 planar lidars oriented in a horizontal configuration, a roof-mounted 3D lidar unit, and 15 automotive radars. The planar lidars were Sick units returning 180 points at one degree spacing, with scans produced at 75Hz. We used Sick's "interlaced" mode, in which every scan is offset 0.25 degree from the previous scan; this increased the sensitivity of the system to small obstacles. For its larger field-of-view and longer range, we used the Velodyne "High-Definition" lidar, which contains 64 lasers arranged vertically. The whole unit spins, yielding a 360-degree scan at 15Hz.

Our Delphi ACC3 radar units are unique among our sensors in that they are already deployed on mass-market automobiles to support so-called "adaptive cruise control" at highway speeds. Since each radar has a narrow 18° field of view, we arranged fifteen of them in an overlapping, tiled configuration in order to achieve a 256° field-of-view.

The planar lidar and radar fields of view are shown in Figure 4. The 360° field of view of the Velodyne is a ring around the vehicle stretching from 5 to 60m. A wide field of view may be achieved either through the use of many sensors (as we did) or by physically actuating a smaller number of sensors. Actuated sensors add complexity (namely, the actuators, their control circuitry, and their feedback sensors), create an additional control problem (which way should the sensors be pointed?), and ultimately produce less data for a given expenditure of engineering effort. For these reasons, we chose to use many fixed sensors rather than fewer mechanically actuated sensors.

The obstacle tracking system was decoupled into two largely independent subsystems: one using lidar data, the other using radar. Each subsystem was tuned individually for a low false-positive rate; the output of the high-level system was the union of the subsystems' output. Our simple data fusion scheme allowed each subsystem to be developed in a decoupled and parallel fashion, and made it easy to add or remove a subsystem with a predictable performance impact. From a reliability perspective, this strategy could prevent a fault in one subsystem from affecting another.

4.2.1 Lidar-Based Obstacle Detection

Our lidar obstacle tracking system combined data from 12 planar lidars (Figure 5) and the Velodyne lidar. The Velodyne point cloud was dramatically more dense than all of the planar lidar data combined (Figure 6), but including planar lidars brought three significant advantages. First, it was impossible to mount the Velodyne device so that it had no blind spots (note the large empty area immediately around the vehicle): the planar lidars fill in these blind spots. Second, the planar lidars provided a measure of fault tolerance, allowing our system to continue to operate if the Velodyne failed. Since the Velodyne was a new and experimental sensor with which we had little experience, this was a serious concern. The faster update rate of the planar lidars (75Hz versus the Velodyne's 15Hz) also makes data association of fast-moving obstacles easier.

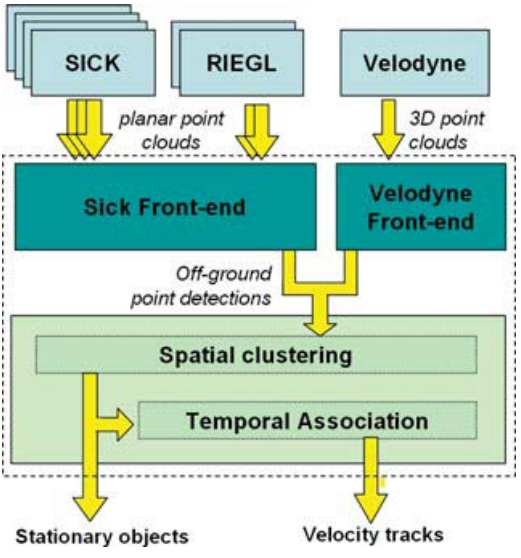


Fig. 5. Lidar subsystem block diagram. Lidar returns are first classified as "obstacle", "ground", or "outlier". Obstacle returns are clustered and tracked.

Each lidar produces a stream of range and angle tuples; this data is projected into the local coordinate system using the vehicle's position in the local coordinate system (continuously updated as the vehicle moves) and the sensor's position in the vehicle's coordinate system (determined off-line).The result is a stream of 3D points in the local coordinate frame, where all subsequent sensor fusion takes place.

The lidar returns often contain observations of the ground and of obstacles. (We define the ground to be any surface that is locally traversable by our vehicle.) The first phase of our data processing is to classify each return as either "ground", "obstacle", or "outlier". This processing is performed by a "front-end" module. The planar lidars all share a single front-end module whereas the Velodyne has its own

Fig. 6. Raw data. Left: camera view of an urban scene with oncoming traffic. Middle: corresponding horizontal planar lidar data ("pushbroom" lidars not shown for clarity). Right: Velodyne data.

specialized front-end module. In either case, their task is the same: to output a stream of points thought to correspond only to obstacles (removing ground and outliers).

4.2.2 Planar Lidar Front-End

A single planar lidar cannot reliably differentiate between obstacles and non-flat terrain (see Figure 7). However, with more than one planar lidar, an appreciable change in z (a reliable signature of an obstacle) can be measured.

This strategy requires that any potential obstacle be observable by multiple planar lidars, and that the lidars observe the object at different heights. Our vehicle has many planar lidars, with overlapping fields of view but different mounting heights, to ensure that we can observe nearby objects more than once (see Figure 4). This redundancy conveys an additional advantage: many real-world surfaces are highly reflective and cannot be reliably seen by Sick sensors. Even at a distance of under 2m, a dark-colored shiny surface (like the wheel well of a car) can scatter enough incident laser energy to prevent the lidar from producing a valid range estimate. With multiple lasers, at different heights, we increase the likelihood that the sensor will return at least some valid range samples from any given object. This approach also increases the system's fault tolerance.

Before classifying returns, we de-glitch the raw range returns. Any returns that are farther than 1m away from any other return are discarded; this is effective at removing single-point outliers.

The front-end algorithm detects returns that are near each other (in the vehicle's XY plane). If two nearby returns arise from different sensors, we know that there is an obstacle at the corresponding (x, y) location. To implement this algorithm, we allocate a two-dimensional grid at 25cm resolution representing an area of 200×200m centered around the vehicle. Each grid cell has a linked list of all lidar returns that have recently landed in that cell, along with the sensor ID and timestamp of each return. Whenever a new return is added to a cell, the list is searched: if one of the previous returns is close enough and was generated by a different sensor, then

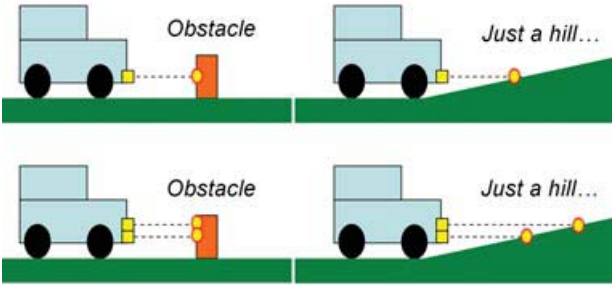


Fig. 7. Obstacle or hill? With a single planar lidar, obstacles cannot be reliably discriminated from traversable (but hilly) terrain. Multiple planar lidars allow appreciable changes in z to be measured, resolving the ambiguity.

both returns are passed to the obstacle tracker. As this search proceeds, returns older than 33ms are discarded.

One difficulty we encountered in developing the planar lidar subsystem is that it is impossible to mount two lidars so that they are exactly parallel. Even small alignment errors are quickly magnified at long ranges, with the result that the actual change in z is not equal to the difference in sensor mounting height. Convergent sensors pose the greatest problem: they can potentially sense the same object *at the same height*, causing a false positive. Even if the degree of convergence can be precisely measured (so that false positives are eliminated), the result is a blind spot. Our solution was to mount the sensors in slightly *divergent* sets: this reduces our sensitivity to small obstacles at long ranges (since we can detect only larger-than-desired changes in z), but eliminates false positives and blind spots.

4.2.3 Velodyne Front-End

As with the planar lidar data, we needed to label each Velodyne range sample as belonging to either the ground or an obstacle. The high density of Velodyne data enabled us to implement a more sophisticated obstacle-ground classifier than for the planar lidars. Our strategy was to identify points in the point cloud that are likely to be on the ground, then fit a non-parametric ground model through those ground points. Other points in the cloud that are far enough above the ground model (and satisfy other criteria designed to reject outliers) are output as obstacle detections.

Although outlier returns with the planar lidars are relatively rare, Velodyne data contains a significant number of outlier returns, making outlier rejection a more substantial challenge. These outliers include ranges that are both too short and too long, and are often influenced by the environment. Retro-reflectors wreak havoc with the Velodyne, creating a cloud of erroneous returns all around the reflector. The sensor also exhibits systematic errors: observing high-intensity surfaces (such as road paint) causes the range measurements to be consistently too short. The result is that brightly painted areas can appear as curb-height surfaces. The Velodyne contains 64 individual lasers, each of which varies from the others in sensitivity and range offset; this variation introduces additional noise.

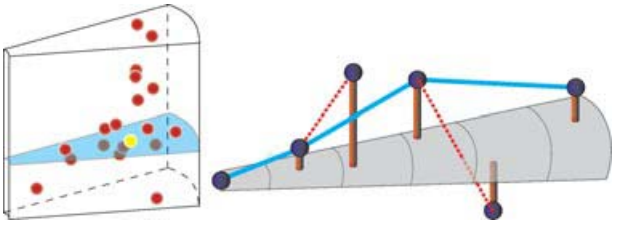


Fig. 8. Ground candidates and interpolation. Velodyne returns are recorded in a polar grid (left: single cell is shown). The lowest 20% (in z height) are rejected as possible outliers; the next lowest return is a ground candidate. A ground model is linearly interpolated through ground candidates (right), subject to a maximum slope constraint.

Our ground estimation algorithm estimates the terrain profile from a sequence of "candidate" points that locally appear to form the ground. The system generates ground candidate points by dividing the area around the vehicle into a polar grid. Each cell of the grid collects all Velodyne hits landing within that cell during four degrees of sensor rotation and three meters of range. If a particular cell has more than a threshold number of returns (nominally 30), then that cell will produce a candidate ground point. Due to the noise in the Velodyne, the candidate point is not the lowest point; instead, the lowest 20% of points (as measured by z) are discarded before the next lowest point is accepted as a candidate point.

While candidate points often represent the true ground, it is possible for elevated surfaces (such as car roofs) to generate candidates. Thus the system filters candidate points further by subjecting them to a maximum ground-slope constraint. We assume that navigable terrain never exceeds a slope of 0.2 (roughly 11 degrees). Beginning at our own vehicle's wheels (which, we hope, are on the ground) we process candidate points in order of increasing distance from the vehicle, rejecting those points that would imply a ground slope in excess of the threshold (Figure 8). The resulting ground model is a polyline (between accepted ground points) for each radial sector (Figure 9).

Explicit ground tracking serves not only as a means of identifying obstacle points, but improves the performance of the system over a naive $z = 0$ ground plane model in two complementary ways. First, knowing where the ground is allows the height of a particular obstacle to be estimated more precisely; this in turn allows the obstacle height threshold to be set more aggressively, detecting more actual obstacles with fewer false positives. Second, a ground estimate allows the height above the ground of each return to be computed: obstacles under which the vehicle will safely pass (such as overpasses and tree canopies) can thus be rejected.

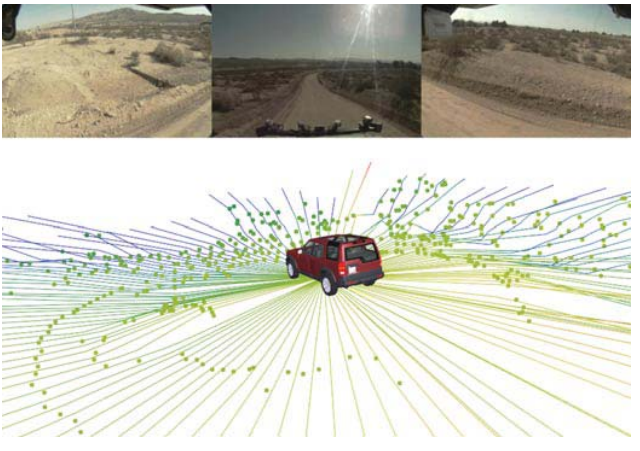


Fig. 9. Ground model example. On hilly terrain, the terrain deviates significantly from a plane, but is tracked fairly well by the ground model.

Given a ground estimate, one could naively classify lidar returns as "obstacles" if they are a threshold above the ground. However, this strategy is not sufficiently robust to outliers. Individual lasers tend to generate consecutive sequences of outliers: for robustness, it was necessary to require multiple lasers to agree on the presence of an obstacle.

The laser-to-laser calibration noise floor tends to lie just under 15cm: constantly changing intrinsic variations across lasers makes it impossible to reliably measure, across lasers, height changes smaller than this. Thus the overlying algorithm cannot reliably detect obstacles shorter than about 15cm.

For each polar cell, we tally the number of returns generated by each laser that is above the ground by an "evidence" threshold (nominally 15cm). Then, we consider each return again: those returns that are above the ground plane by a slightly larger threshold (25cm) and are supported by enough evidence are labelled as obstacles. The evidence criteria can be satisfied in two ways: by three lasers each with at least three returns, or by five lasers with one hit. This mix increases sensitivity over any single criterion, while still providing robustness to erroneous data from any single laser.

The difference between the "evidence" threshold (15cm) and "obstacle" threshold (25cm) is designed to increase the sensitivity of the obstacle detector to low-lying obstacles. If we used the evidence threshold alone (15cm), we would have too many false positives since that threshold is near the noise floor. Conversely, using the 25cm threshold alone would require obstacles to be significantly *taller* than 25cm, since we must require multiple lasers to agree and each laser has a different pitch angle. Combining these two thresholds increases the sensitivity without significantly affecting the false positive rate.

All of the algorithms used on the Velodyne operate on a single sector of data, rather than waiting for a whole scan. If whole scans were used, the motion of the vehicle would inevitably create a seam or gap in the scan. Sector-wise processing also reduces the latency of the system: obstacle detections can be passed to the obstacle tracker every 3ms (the delay between the first and last laser to scan at a particular bearing), rather than every 66ms (the rotational period of the sensor). During the saved 63ms, a car travelling at 15m/s would travel almost a meter. Every bit of latency that can be saved increases the safety of the system by providing earlier warning of danger.

4.2.4 Clustering

The Velodyne alone produces up to a million hits per second; tracking individual hits over time is computationally prohibitive and unnecessary. Our first step was in data reduction: reducing the large number of hits to a much smaller number of "chunks." A chunk is simply a record of multiple, spatially close range samples. The chunks also serve as the mechanism for fusion of planar lidar and Velodyne data: obstacle detections from both front ends are used to create and update chunks.

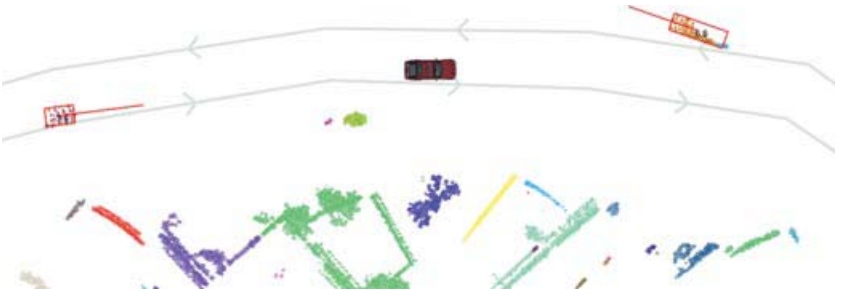


Fig. 10. Lidar obstacle detections. Our vehicle is in the center; nearby (irregular) walls are shown, clustered according to physical proximity to each other. Two other cars are visible: an oncoming car ahead and to the left, and another vehicle following us (a chase car). The red boxes and lines indicated estimated velocities. The long lines with arrows indicated the nominal travel lanes – they are included to aid interpretation, but were not used by the tracker.

One obvious implementation of chunking could be through a grid map, by tallying hits within each cell. However, such a representation is subject to significant quantization effects when objects lie near cell boundaries. This is especially problematic when using a coarse spatial resolution.

Instead, we used a representation in which individual chunks of bounded size could be centered arbitrarily. This permitted us to use a coarse spatial decimation (reducing our memory and computational requirements) while avoiding the quantization effects of a grid-based representation. In addition, we recorded the actual extent of the chunk: the chunks have a *maximum* size, but not a minimum size. This allows us to approximate the shape and extent of obstacles much more accurately than would a grid-map method. This floating "chunk" representation yields a better approximation of an obstacle's boundary without the costs associated with a fine-resolution gridmap.

Chunks are indexed using a two-dimensional look-up table with about 1m resolution. Finding the chunk nearest a point p involves searching through all the grid cells that could contain a chunk that contains p. But since the size of a chunk is bounded, the number of grid cells and chunks is also bounded. Consequently, lookups remain an $O(1)$ operation.

For every obstacle detection produced by a front-end, the closest chunk is found by searching the two-dimensional lookup table. If the point lies within the closest chunk, or the chunk can be enlarged to contain the point without exceeding the maximum chunk dimension (35cm), the chunk is appropriately enlarged and our work is done. Otherwise, a new chunk is created; initially it will contain only the new point and will thus have zero size.

Periodically, every chunk is re-examined. If a new point has not been assigned to the chunk within the last 250ms, the chunk expires and is removed from the system.

Clustering Chunks Into Groups

A physical object is typically represented by more than one chunk. In order to compute the velocity of obstacles, we must know which chunks correspond to the same physical objects. To do this, we clustered chunks into groups; any two chunks within 25cm of one another were grouped together as the same physical object. This clustering operation is outlined in Algorithm 1.

Algorithm 1. Chunk Clustering

```
1: Create a graph G with a vertex for each chunk and no edges
2: for all c ∈ chunks do
3:     for all chunks d within ε of c do
4:         Add an edge between c and d
5:     end for
6: end for
7: Output connected components of G.
```

This algorithm requires a relatively small amount of CPU time. The time required to search within a fixed radius of a particular chunk is in fact $O(1)$, since there is a constant bound on the number of chunks that can simultaneously exist within that radius, and these chunks can be found in $O(1)$ time by iterating over the two-dimensional lookup table that stores all chunks. The cost of merging subgraphs, implemented by the Union-Find algorithm (Rivest and Leiserson, 1990), has a complexity of less than $O(\log N)$. In aggregate, the total complexity is less than $O(N \log N)$.

4.2.5 Tracking

The goal of clustering chunks into groups is to identify connected components so that we can track them over time. The clustering operation described above is repeated at a rate of 15Hz. Note that chunks are persistent: a given chunk will be assigned to multiple groups, one at each time step.

At each time step, the new groups are associated with a group from the previous time step. This is done via a voting scheme; the new group that overlaps (in terms of the number of chunks) the most with an old group is associated with the old group. This algorithm yields a fluid estimate of which objects are connected to each other: it is not necessary to explicitly handle groups that appear to merge or split.

The bounding boxes for two associated groups (separated in time) are compared, yielding a velocity estimate. These instantaneous velocity estimates tend to be noisy: our view of obstacles tends to change over time due to occlusion and scene geometry, with corresponding changes in the apparent size of obstacles.

Obstacle velocities are filtered over time *in the chunks*. Suppose that two sets of chunks are associated with each other, yielding a velocity estimate. That velocity estimate is then used to update the constituent chunks' velocity estimates. Each

chunk's velocity estimate is maintained with a trivial Kalman filter, with each observation having equal weight.

Storing velocities in the chunks conveys a significant advantage over maintaining separate "tracks": if the segmentation of a scene changes, resulting in more or fewer tracks, the new groups will inherit reasonable velocities due to their constituent chunks. Since the segmentation is fairly volatile due to occlusion and changing scene geometry, maintaining velocities in the chunks provides greater continuity than would result from frequently creating new tracks.

Finally, we output obstacle detections using the current group segmentation, with each group reported as having a velocity equal to the weighted average of its constituent chunks. (The weights are simply the confidence of each individual chunk's velocity estimate.)

A core strength of our system is its ability to produce velocity estimates for rapidly moving objects with very low latency. This was a design goal, since fast moving objects represent the most acute safety hazard.

The corresponding weakness of our system is in estimating the velocity of slow-moving obstacles. Accurately measuring small velocities requires careful tracking of an object over relatively long periods of time. Our system averages instantaneous velocity measurements, but these instantaneous velocity measurements are contaminated by noise that can easily swamp small velocities. In practice, we found that the system could reliably track objects moving faster than 3m/s. The motion planner avoids "close calls" with all obstacles, keeping the vehicle away from them. Improving tracking of slow-moving obstacles remains a goal for future work.

Another challenge is the "aperture" problem, in which a portion of a static obstacle is sensed through a small gap. The motion of our own vehicle can make it appear that an obstacle is moving on the other side of the aperture. While apertures could be detected and explicitly filtered, the resulting phantom obstacles tend to have velocities parallel to our own vehicle and thus do not significantly affect motion planning.

Use of a Prior

Our system operates *without* a prior on the location of the road. Prior information on the road could be profitably used to eliminate false positives (by assuming that moving cars must be on the road, for example), but we chose not to use a prior for two reasons. Critically, we wanted our system to be robust to moving objects *anywhere*, including those that might be pulling out of a driveway, or jaywalking pedestrians. Second, we wanted to be able to test our detector in a wide variety of environments without having to first generate the corresponding metadata.

4.2.6 Lidar Tracking Results

The algorithm performed with high reliability, correctly detecting obstacles including a thin metallic gate that errantly closed across our path.

In addition to filling in blind spots (to the Velodyne) immediately around the vehicle, the Sick lidars reinforced the obstacle tracking performance. In order to quantitatively measure the effectiveness of the planar lidars (as a set) versus the

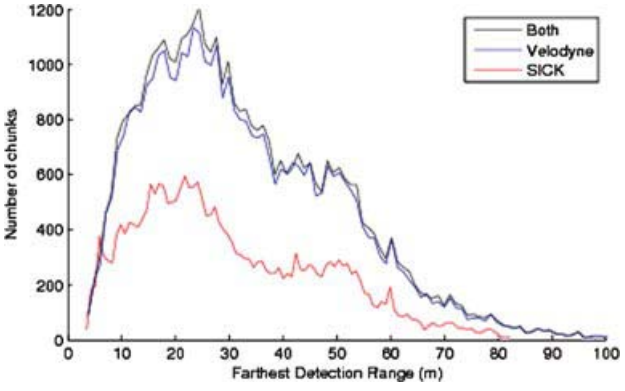


Fig. 11. Detection range by sensor. For each of 40,000 chunks, the earliest detection of the chunk was collected for each modality (Velodyne and Sick). The Velodyne's performance was substantially better than that of the Sick's, which observed fewer objects.

Velodyne, we tabulated the maximum range at which each subsystem first observed an obstacle (specifically, a chunk). We consider only chunks that were, at one point in time, the closest to the vehicle along a particular bearing; the Velodyne senses many obstacles farther away, but in general, it is the closest obstacle that is most important. Statistics gathered over the lifetimes of 40,000 chunks (see Figure 11) indicate that:

- The Velodyne tracked 95.6% of all the obstacles that appeared in the system; the Sicks alone tracked 61.0% of obstacles.
- The union of the two subsystems yielded a minor, but measurable, improvement with 96.7% of all obstacles tracked.
- Of those objects tracked by both the Velodyne and the Sick, the Velodyne detected the object at a longer range: 1.2m on average.

In complex environments, like the one used in this data set, the ground is often non-flat. As a result, planar lidars often find themselves observing sky or dirt. While we can reject the dirt as an obstacle (due to our use of multiple lidars), we cannot see the obstacles that might exist nearby. The Velodyne, with its large vertical field of view, is largely immune to this problem: we attribute the Velodyne subsystem's superior performance to this difference. The Velodyne could also see over and sometimes through other obstacles (i.e., foliage), which would allow it to detect obstacles earlier.

One advantage of the Sicks was that their higher rotational rate (75Hz versus the Velodyne's 15Hz) which makes data association easier for fast-moving obstacles. If another vehicle is moving at 15m/s, the velodyne will observe a 1m displacement between scans, while the Sicks will observe only a 0.2m displacement between scans.

4.2.7 Radar-Based Fast-Vehicle Detection

The radar subsystem complements the lidar subsystem by detecting moving objects at ranges beyond the reliable detection range of the lidars. In addition to range and bearing, the radars directly measure the closing rate of moving objects using Doppler, greatly simplifying data association. Each radar has a field of view of 18 degrees. In order to achieve a wide field of view, we tiled 15 radars (see Figure 4).

The radar subsystem maintains a set of active tracks. We propagate these tracks forward in time whenever the radar produces new data, so that we can compare the predicted position and velocity to the data returned by the radar.

The first step in tracking is to associate radar detections to any active tracks. The radar produces Doppler closing rates that are consistently within a few meters per second of the truth: if the predicted closing rate and the measured closing rate differ by more than 2m/s, we disallow a match. Otherwise, the closest track (in the XY plane) is chosen for each measurement. If the closest track is more than 6.0m from the radar detection, a new track is created instead.

Each track records all radar measurements that have been matched to it over the last second. We update each track's position and velocity model by computing a least-squares fit of a constant velocity model to the $(x, y, time)$ data from the radars. We weight recent observations more strongly than older observations since the target may be accelerating. For simplicity, we fit the constant velocity model using just the (x, y) points; while the Doppler data could probably be profitably used, this simpler approach produced excellent results. Figure 12 shows a typical output from the radar data association and tracking module. Although no lane data was used in the radar tracking module the vehicle track directions match well. The module is able to facilitate the overall objective of detecting when to avoid entering an intersection due to fast approaching vehicles.

Unfortunately, the radars cannot easily distinguish between small, innocuous objects (like a bolt lying on the ground, or a sewer grate) and large objects (like cars). In order to avoid false positives, we used the radars only to detect moving objects.

(a) (b)

Fig. 12. Radar tracking 3 vehicles. (a) Front right camera showing 3 traffic vehicles, one on coming. (b) *Points:* Raw radar detections with tails representing the doppler velocity. *Red rectangles:* Resultant vehicle tracks with speed in meters/second (rectangle size is simply for visualization).

4.3 Hazard Detector

We define hazards as object that we *shouldn't* drive over, even if the vehicle probably could. Hazards include pot-holes, curbs, and other small objects. The hazard detector is not intended to detect cars and other large (potentially moving objects): instead, the goal of the module is to estimate the condition of the road itself.

In addition to the Velodyne, Talos used five downwards-canted planar lidars positioned on the roof: these were primarily responsible for observing the road surface. The basic principle of the hazard detector is to look for z-height discontinuities in the laser scans. Over a small batch of consecutive laser returns, the z slope is computed by dividing the change in z by the distance between the individual returns. This slope is accumulated in a gridmap that records the largest slope observed in every cell. This gridmap is slowly built up over time as the sensors pass over new ground and extended for about 40m in every direction. Data that "fell off" the gridmap (by being over 40m away) was forgotten.

The Velodyne sensor, with its 64 lasers, could observe a large area around the vehicle. However, hazards can only be detected where lasers actually strike the ground: the Velodyne's lasers strike the ground in 64 concentric circles around the vehicle with significant gaps between the circles. However, these gaps are filled in as the vehicle moves. Before we obtained the Velodyne, our system relied on only the five planar Sick lidars with even larger gaps between the lasers.

The laser-to-laser calibration of the Velodyne was not sufficiently reliable or consistent to allow vertical discontinuities to be detected by comparing the z height measured by different physical lasers. Consequently, we treated each Velodyne laser independently as a line scanner.

Unlike the obstacle detector, which assumes that obstacles will be constantly re-observed over time, the hazard detector is significantly more stateful since the largest slope ever observed is remembered for each (x,y) grid cell. This "running maximum" strategy was necessary because any particular line scan across a hazard only samples the change in height along one direction. A vertical discontinuity along any direction, however, is potentially hazardous. A good example of this anisotropic sensitivity is a curb: when a line scanner samples parallel to the curb, no discontinuity is detected. Only when the curb is scanned perpendicularly does a hazard result. We mounted our Sick sensors so that they would likely sample the curb at a roughly perpendicular angle (assuming we are driving parallel to the curb), but ultimately, a diversity of sampling angles was critical to reliably sensing hazards.

4.3.1 Removal of Moving Objects

The gridmap described above, which records the worst z slope seen at each (x,y) location, would tend to detect moving cars as large hazards smeared across the moving car's trajectory. This is undesirable, since we wish to determine the condition of the road beneath the car.

Our solution was to run an additional "smooth" detector in parallel with the hazard detector. The maximum and minimum z heights occurring during 100ms integration periods are stored in the gridmap. Next, 3x3 neighborhoods of the gridmap

are examined: if all nine areas have received a sufficient number of measurements and the maximum difference in z is small, the grid-cell is labeled as "smooth". This classification overrides any hazard detection. If a car drives through our field of view, it may result in temporary hazards, but as soon as the ground beneath the car is visible, the ground will be marked as smooth instead.

The output of the hazard and smooth detector is shown in Figure 26(a). Red is used to encode hazards of various intensities while green represents ground labelled as smooth.

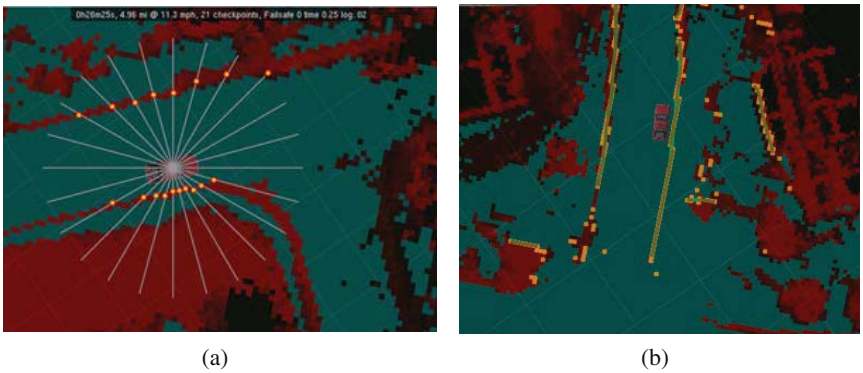
4.3.2 Hazards as High-Cost Regions

The hazard map was incorporated by the Drivability Map as high-cost regions. Motion plans that passed over hazardous terrain were penalized, but *not* ruled-out entirely. This is because the hazard detector was prone to false positives for two reasons. First, it was tuned to be highly sensitive so that even short curbs would be detected. Second, since the cost map was a function of the worst-ever seen z slope, a false-positive could cause a phantom hazard that would last forever. In practice, associating a cost with curbs and other hazards was sufficient to keep the vehicle from running over them; at the same time, the only consequence of a false positive was that we might veer around a phantom. A false positive could not cause the vehicle to get stuck.

4.3.3 Road-Edge Detector

Hazards often occur at the road edge, and our detector readily detects them. Berms, curbs, and tall grass all produce hazards that are readily differentiated from the road surface itself.

We detect the road-edge by casting rays from the vehicle's current position and recording the first high-hazard cell in the gridmap (see Figure 13(a)). This results in a number of road-edge point detections; these are segmented into chains based on



(a) (b)

Fig. 13. Hazard Map: Red is hazardous, cyan is safe. (a) Rays radiating from vehicle used to detect the road-edge. (b) Poly-lines fitted to road-edge.

their physical proximity to each other. A non-parametric curve is then fitted through each chain (shown in Figure 13(b)). Chains that are either very short or have excessive curvature are discarded; the rest are output to other parts of the system.

4.4 Lane Finding

Our approach to lane finding involves three stages. In the first, the system detects and localizes painted road markings in each video frame, using lidar data to reduce the false-positive detection rate. A second stage processes the road-paint detections along with lidar-detected curbs (see Section 4.3) to estimate the centerlines of nearby travel lanes. Finally, the detected centerlines output by the second stage are filtered, tracked, and fused with a weak prior to produce one or more non-parametric lane outputs.

4.4.1 Absolute Camera Calibration

Our road-paint detection algorithms assume that GPS and IMU navigation data are available of sufficient quality to correct for short-term variations in vehicle heading, pitch, and roll during image processing. In addition, the intrinsic (focal length, center, and distortion) and extrinsic (vehicle-relative pose) parameters of the cameras have been calibrated ahead of time. This "absolute calibration" allows preprocessing of the images in several ways (Figure 14):

- The horizon line is projected into each image frame. Only pixel rows below this line are considered for further processing.
- Our lidar-based obstacle detector supplies real-time information about the location of obstructions in the vicinity of the vehicle. These obstacles are projected into the image and their extent masked out during the paint-detection algorithms, an important step in reducing false positives.
- The inertial data allows us to project the expected location of the ground plane into the image, providing a useful prior for the paint-detection algorithms.

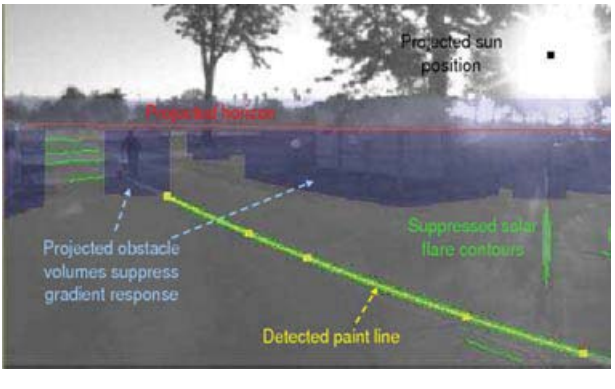


Fig. 14. Use of absolute camera calibration to project real-world quantities into the image.

- False paint detections caused by lens flare can be detected and rejected. Knowing the time of day and our vehicle pose relative to the Earth, we can compute the ephemeris of the sun. Line estimates that point toward the sun in image coordinates are removed.

4.4.2 Road-Paint Detection

We employ two algorithms for detecting patterns of road paint that constitute lane boundaries. Both algorithms accept raw frames as input and produce sets of connected line segments, expressed in the local coordinate frame, as output. The algorithms are stateless; each frame from each camera is considered independently, deferring spatial-temporal boundary fusion and tracking to higher-level downstream stages.

The first algorithm applies one-dimensional horizontal and vertical matched filters (for lines along and transverse to the line of sight, respectively) whose support corresponds to the expected width of a painted line marking projected onto each image row. As shown in Figure 15, the filters successfully discard most scene clutter while producing strong responses along line-like features. We identify local maxima of the filter responses, and for each maximum compute the principal line direction as the dominant eigenvector of the Hessian in a local window centered at that maximum. The algorithm finally connects nearby maxima into splines that represent continuous line markings; connections are established by growing spline candidates from a set of random seeds, guided by a distance transform function generated from the entire list of maxima.

The second algorithm for road-paint detection identifies potential paint boundary pairs that are proximal and roughly parallel in real-world space, and whose local gradients point toward each other (Figure 16). We compute the direction and magnitude of the image's spatial gradients, which undergo thresholding and non-maximal suppression to produce a sparse feature mask. Next, a connected components algorithm walks the mask to generate smooth contours of ordered points, broken at discontinuities in location and gradient direction. A second iterative walk then grows centerline curves between contours with opposite-pointing gradients. We enforce global smoothness and curvature constraints by fitting parabolas to the resulting curves and recursively breaking them at points of high deviation or spatial gaps. We finally remove all curves shorter than a given threshold length to produce the final road paint-line outputs.

4.4.3 Lane Centerline Estimation

The second stage of lane finding estimates the geometry of nearby lanes using a weighted set of recent road paint and curb detections, both of which are represented as piecewise linear curves. Lane centerlines are represented as locally parabolic segments, and are estimated in two steps. First, a centerline evidence image D is constructed, where the value of each pixel $D(\mathbf{p})$ of the image corresponds to the evidence that a point $\mathbf{p} = [p_x, p_y]$ in the local coordinate frame lies on the center of

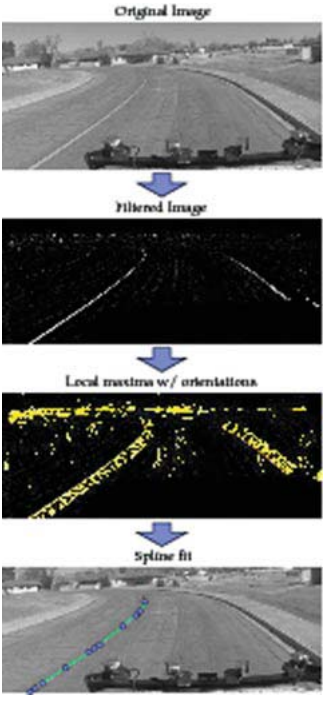


Fig. 15. The matched filter based detector from start to finish. The original image is convolved with a matched filter at each row (horizontal filter shown here). Local maxima in the filter response are enumerated and their dominant orientations computed. The figure depicts orientation by drawing the perpendiculars to each maximum. Finally, nearby maxima are connected into cubic hermite splines.

a lane. Second, parabolic segments are fit to the ridges in D and evaluated as lane centerline candidates.

To construct D, road paint and curb detections are used to increase or decrease the values of pixels in the image, and are weighted according to their age (older detections are given less weight). The value of D at a pixel corresponding to the point $\mathbf{p}$ is computed as the weighted sum of the influences of each road paint and curb detection d_i at the point $\mathbf{p}$:

$$D(\mathbf{p}) = \sum_i e^{-a(d_i)\lambda} g(d_i, \mathbf{p})$$

where $a(d_i)$ denotes how much time has passed since d_i was received, λ is a decay constant, and $g(d_i, \mathbf{p})$ is the influence of d_i at $\mathbf{p}$. We chose $\lambda = 0.7$.

Before describing how the influence is determined, we make three observations. First, a lane is more likely to be centered $\frac{1}{2}$ lane width from a strip of road paint or a curb. Second, 88% of federally managed lanes in the U.S. are between 3.05 m and 3.66 m wide

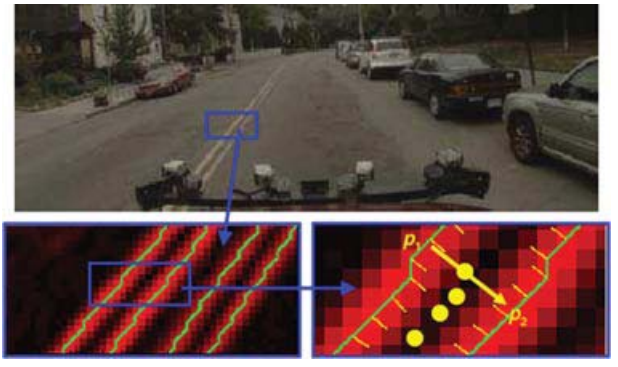


Fig. 16. Progression from original image through smoothed gradients, border contours, and symmetric contour pairs to form centerline candidate.

(USDOT Federal Highway Administration, Office of Information Management, 2005). Third, a curb gives us different information about the presence of a lane than does road paint. From these observations and the characteristics of our road paint and curb detectors, we define two functions $f_{rp}(x)$ and $f_{cb}(x)$, where x is the Euclidean distance from d_i to $\mathbf{p}$:

$$f_{rp}(x) = -e^{-\frac{x^2}{0.42}} + e^{-\frac{(x-1.83)^2}{0.14}} \tag{1}$$

$$f_{cb}(x) = -e^{-\frac{x^2}{0.42}}. \tag{2}$$

The functions f_{rp} and f_{cb} are intermediate functions used to compute the influence of road paint and curb detections, respectively, on D. f_{rp} is chosen to have a minimum at $x = 0$, and a maximum at one half lane width (1.83 m). f_{cb} is always negative, indicating that curb detections are used only to decrease the evidence for a lane centerline. This addressed our curb detector's occasional detection of curb-like features where no curbs were present. Let $\mathbf{c}$ indicate the closest point on d_i to $\mathbf{p}$. The actual influence of a detection is computed as:

$$g(d_i, \mathbf{p}) = \begin{cases} 0 & \text{if } \mathbf{c} \text{ is an endpoint of } d_i \\ f_{rp}(||\mathbf{p}-\mathbf{c}||) & \text{if } d_i \text{ is road paint} \\ f_{cb}(||\mathbf{p}-\mathbf{c}||) & \text{if } d_i \text{ is a curb} \end{cases}$$

This last condition is introduced because road paint and curbs are only observed in small sections. The effect is that a detection influences only those centerline evidence values immediately next to the detection, and not in front of or behind it.

In practice, D can be initialized once and incrementally updated by adding the influences of newly received detections and applying an exponential time decay at each update. Once D has been constructed, the set R of ridge points is identified by

Fig. 17. Our system constructs a centerline evidence image using road edge and road paint detections. Lane centerline candidates (blue) are identified by fitting parabolic segments to the ridges of the image. Front-center camera is shown in top left for context.

scanning D for points that are local maxima along either a row or a column, and also above a minimum threshold. Next, a random sample consensus (RANSAC) algorithm (Fischler and Bolles, 1981) is used to fit parabolic segments to the ridge points. At each RANSAC iteration, three ridge points are randomly selected for a three-point parabola fit. The directrix of the parabola is chosen to be the first principle component of the three points.

To determine the set of inliers for a parabola, we first compute its conic coefficient matrix $\mathbf{C}$ (Hartley and Zisserman, 2001), and define the set of candidate inliers L to contain the ridge points within some algebraic distance α of $\mathbf{C}$.

$$L = \{\mathbf{p} \in R : \mathbf{p}^T \mathbf{C}\mathbf{p} < \alpha\}$$

For our experiments, we chose $\alpha = 1$. The parabola is then re-fit once to L using a linear least-squares method, and a new set of candidate inliers is computed. Next, the candidate inliers are partitioned into connected components, where a ridge point is connected to all neighboring ridge points within a 1m radius. The set of ridge points in the largest component is chosen as the set of actual inliers for the parabola. The purpose of this partitioning step is to ensure that a parabola cannot be fitted across multiple ridges, and requires that an entire identified ridge be connected. Finally, a score for the entire parabola is computed.

$$score = \sum_{\mathbf{p} \in L} \frac{1}{1 + \mathbf{p}^T \mathbf{C}\mathbf{p}}$$

The contribution of an inlier to the total parabola score is inversely related to the inlier's algebraic distance, with each inlier contributing a minimum amount to the score. The overall result is that parabolas with many very good inliers have the greatest score. If the score of a parabola is below some threshold, then it is discarded.

After a number of RANSAC iterations (we found 200 to be sufficient), the parabola with greatest score is selected as a candidate lane centerline. Its inliers

are removed from the set of ridge points, and all remaining parabolas are re-fit and re-scored using this reduced set of ridge points. The next best-scoring parabola is chosen, and this process is repeated to produce at most 5 candidate lane centerlines (Figure 17).

4.4.4 Lane Tracking

The primary purpose of the lane tracker is to maintain a stateful, smoothly time-varying estimate of the nearby lanes of travel. To do so, it uses both the candidate lane centerlines produced by the centerline estimator and an *a-priori* estimate derived from the RNDF. For the purposes of our system, the RNDF was treated as a strong prior on the number and type of lanes, and a weak prior on their position and geometry.

As the vehicle travels, it constructs and maintains representations of all portions of all lanes within a fixed radius of 75m. The centerline of each lane is modeled as a piecewise linear curve, with control points spaced approximately every 2m. Each control point is given a scalar confidence value indicating the certainty of the lane tracker's estimate at that point. The lane tracker decays the confidence of a control point as the vehicle travels, and increases it either by detecting proximity to an RNDF waypoint or by updating control points with centerline estimates produced from the second stage.

As centerline candidates are generated, the lane tracker attempts to match each candidate with a tracked lane. If a matching is successful, then the candidate is used to update the lane estimate. To determine if a candidate c is a good match for a tracked lane l, the longest segment s_c of the candidate is identified such that every point on s_c is within some maximum distance τ to l. We then define the match score $m(c,l)$ as:

$$m(c,l) = \int_{s_c} 1 + \frac{\tau - d(s_c(x),l)}{\tau} dx$$

where $d(\mathbf{p},l)$ is the distance from a point $\mathbf{p}$ to the lane l. Intuitively, if s_c is sufficiently long and close to this estimate, then it is considered a good match. We choose the matching function to rely only on the closest segment of the candidate, and not on the entire candidate, based on the premise that as the vehicle travels, the portions of a lane that it observes vary smoothly over time, and previously unobserved portions should not adversely affect the matching as long as sufficient overlap is observed elsewhere.

Once a centerline candidate has been matched to a tracked lane, it is used to update the lane estimates by mapping control points on the tracked lane to the centerline candidate, with an exponential moving average applied for temporal smoothing. At each update, the confidence values of control points updated from a matching are increased, and others are decreased. If the confidence value of a control point decreases below some threshold, then its position is discarded and recomputed as a linear interpolation of its closest surrounding confident control points.

5 Planning and Control Algorithms

This section explains the planning and control algorithms developed for the Talos vehicle. The Navigator dictates the mission-level behavior of the vehicle. The Motion Planner, Drivability Map and Controller operate in a tight coupling to achieve the required motion control objective set by the Navigator though the often complex and unpredictable driving environment.

5.1 Navigator

The Navigator is responsible for planning the high-level behavior of the vehicle including:

- Shortest route to the next MDF checkpoint
- Intersection precedence, crossing, and merging
- Passing
- Blockage replanning
- Generation of the *goal* for the Motion Planner
- Generation of the failsafe timers
- Turn signaling

The key innovation of the Navigator is that high-level planning tasks (as in the list above) are cleanly separated from low-level motion planning by a compact message exchange. The Navigator directs the action of the Motion Planner (described below in Section 5.3) by manipulating the position of the *goal*, a point in the local frame where the Navigator intends the vehicle to travel next over a 40–50 meter horizon. It then becomes the Motion Planner's responsibility to avoid obstacles, vehicles, and obey lane constraints while attempting to reach this goal.

The primary inputs to the Navigator are the lane information, MDF, and the vehicle pose. Twice per second the Navigator recomputes the closest lane to the vehicle and uses that as the starting point for the search to the next MDF checkpoint. This search of the road network uses the A* algorithm (Hart and Raphael, 1968) to find the lowest cost path (smallest time) to the next checkpoint. The speed limit of each road segment is used for this cost estimate with additional time penalties for each lane change and intersection traversal. Since this search is run continuously at 2Hz, dynamic replanning comes "for free" as conditions change since the costs of the search are updated.

The primary output of the Navigator is the goal point which is sent to the Motion Planner. The goal is generally located at RNDF waypoints since these locations are guaranteed to be on the road. As the vehicle gets close to a goal, the goal is moved ahead to the next waypoint before the vehicle is so close that it would need to slow down to avoid overshooting. In this way, the goal acts as a "carrot" to motivate the Motion Planner. If the Navigator wishes the car to stop at an intersection, it keeps the goal fixed on the stop line. The Motion Planner will then bring the vehicle to a controlled stop. Once the intersection is clear, the goal is switched to the waypoint at the exit of the intersection. Parking is executed in a similar fashion.

Fig. 18. The Navigator's view of intersection precedence. PX means there is a car with precedence at that entrance, and PC means there is no car, or the car at the stop line does not have precedence. IX means there is a moving object in the intersection. Talos is clear to proceed when all PX states have transitioned to PC and IX has transitioned to IC.

5.1.1 Intersection Precedence

The logic for intersection precedence, crossing, and merging with moving traffic lies entirely within the Navigator. As previously described, moving the goal is the mechanism by which the Navigator influences the Motion Planner in such situations. This separation has the extra benefit of significantly reducing the complexity of the Motion Planner.

When our vehicle arrives at an intersection, the other intersection entrances are inspected for large obstacles. If a large obstacle is present, then it is considered to be another vehicle and given precedence. Then, as the vehicle waits, if any of the following three conditions become true, the other vehicle no longer has precedence: 1) that vehicle begins moving into the intersection, 2) that obstacle disappears for more than four seconds, or 3) the traffic jam timer expires. Talos also waits whenever there is a moving obstacle present in the intersection whose trajectory will not take it outside the intersection within one second. Figure 18 shows a snapshot of an intersection with these tests in progress.

Crossing and merging is implemented using time-to-collision (TTC) logic. Upon arrival at an intersection, Talos comes to a stop if forward progress would cross or merge with any lane of traffic that does not have a stop sign. For each of these lanes, Talos finds the point where its path intersects the other lane's path and measures the TTC for any incoming traffic from that lane. If the TTC is less than 9 seconds, Talos yields to the moving traffic. Talos came to a full stop whenever the vehicle is on an RNDF "exit" that crosses another RNDF exit and both do not have stop signs. This addresses the fact that the RNDF format does not differentiate between exits in which Talos can proceed without stopping and exits in which a full stop is required.

5.1.2 Passing

The Navigator can control passing behavior using an additional state that it sends to the Motion Planner. Besides the goal, the Navigator continuously informs the Motion Planner whether only the current lane is acceptable for travel, or both the current and opposing lanes are acceptable. When Talos comes to a stop behind a stopped vehicle, the Navigator first ascertains whether passing is allowed (i.e. on a two-lane road and not in a safety area). If allowed, the Navigator checks that the opposing lane is clear and if so, signals to the Motion Planner that travel is allowed in the opposing lane. The goal position is not changed. If the motion planner is able to find a path around the stopped vehicle, it will then begin moving again.

5.1.3 Blockages and Failsafe Modes

In order to handle unexpected or unpredictable events that could occur in an urban environment, we use failsafe and blockage timers. The *failsafe timer* ticks upward from zero whenever the vehicle is not making forward progress. Upon making progress, the timer resets to zero. Upon reaching 80 seconds, we increment the *failsafe mode* and reset the timer back to zero. Thus, normal operation is failsafe mode 0. Once Talos is in failsafe mode 1, the vehicle has to traverse a pre-determined distance in the RNDF before the mode is decremented back to zero. This combination of timer and mode ensures that the failsafe behaviors are phased out slowly once Talos starts making progress rather than immediately reverting as soon as the vehicle starts moving. Other modules in the system can change their behavior based on the value of the failsafe mode and timer to encourage the vehicle to get "un-stuck".

The following summarizes the multiple layers of failsafe logic implemented in various modules:

- Failsafe mode 0:
  10 sec: Relax the center line constraint of the road to allow passing
  10 sec: Shrink the margin retained around obstacles from 30 *cm* to 15 *cm*
  15 sec: Enable reverse gear motion plans
  20 sec: Shrink the margin retained around obstacles from 15 *cm* to 0 *cm*
  30 sec: Make curbs drivable with a high penalty instead of impassable
  35 sec: Unrestrict the area around the goal
  80 sec: Go to *Failsafe mode 1*
- Failsafe mode 1:
  0 sec: Do not observe standoff distances from stationary obstacles
  0 sec: Allow crossing of zone boundaries anywhere
  80 sec: Go to *Failsafe mode 2*
- Failsafe mode 2:
  0 sec: Do not observe standoff distances from moving obstacles
  0 sec: Drop lane constraints completely and navigate as if the area were an obstacle field

0 sec: Shrink the vehicle footprint used for the feasibility check; when the vehicle moves forward, neglect the part of it behind the rear axle; when in reverse, neglect the part of it in front of the front axle
70 sec: Skip the next MDF checkpoint
80 sec: Go to *Failsafe mode 3*

- Failsafe mode 3:
  0 sec: Restart all the processes except the logger, ADU, and process manager. Since the Navigator is restarted Talos will be in *Failsafe mode 0* after the restart.

In addition, detection parameters for the underlying obstacle detectors are relaxed in higher failsafe modes, although never to the point that Talos would drive into a clearly visible obstacle.

The *blockage timer* behaves similarly to the failsafe timer but only ticks upward if Talos is on a two-way road where a U-turn is possible. If the timer reaches 50 seconds of no progress, Talos begins a U-turn by moving the goal to a point behind the vehicle in the opposite lane.

When maneuvers such as U-turns and passing are in highly confined spaces, they can take appreciable time without making much forward progress. In order to ensure that the maneuver being executed is not interrupted, the failsafe and blockage timers increment more slowly when the Motion Planner has found a solution to execute.

5.2 Drivability Map

To enable the path planning algorithm to interface with the perceived environment, the perception data is rendered into a Drivability Map, shown in Figure 19. The Drivability Map consists of: (a) *infeasible regions* which are no-go areas due to proximity to obstacles or undesirable locations; (b) *high-cost regions* which should be avoided if possible by the motion plan, and (c) *restricted regions* that may only be entered if the vehicle can stop in an unrestricted area further ahead. Restricted regions are used to permit minor violations of the lane boundaries if it makes progress down the road. Restricted regions are also used behind vehicles to enforce the requisite number of car lengths' stand-off distance behind a traffic vehicle. If there is enough room to pass a vehicle without crossing the lane boundary (for instance if the vehicle is parked on the side of a wide road), Talos will traverse the restricted region and pass the vehicle and continue in the unrestricted region in front. If the traffic vehicle blocks the lane, Talos will not enter the restricted region because there is no unrestricted place to go. The vehicle will stop behind the restricted region in a vehicle-queuing behavior until the traffic vehicle moves or a passing maneuver begins.

As discussed previously, the Navigator contains a cascade of events triggered by a prolonged lack of progress. For example, after 10 seconds of no progress queuing behind a stationary vehicle the Navigator will enter the passing mode. This mode amounts to a relaxation of the lane center-line constraint. The Drivability Map will

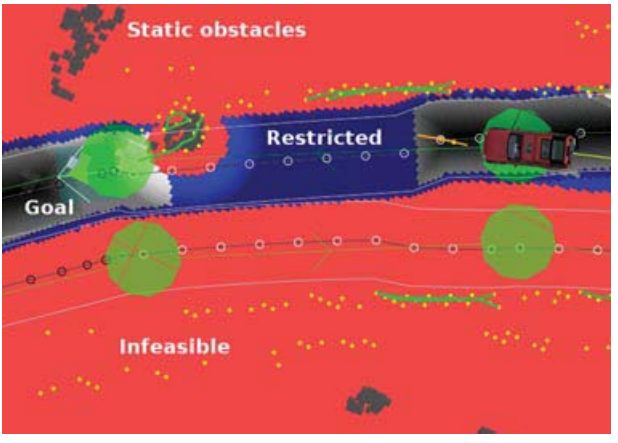
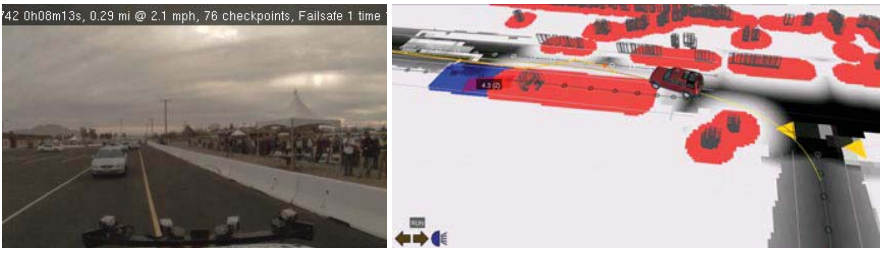


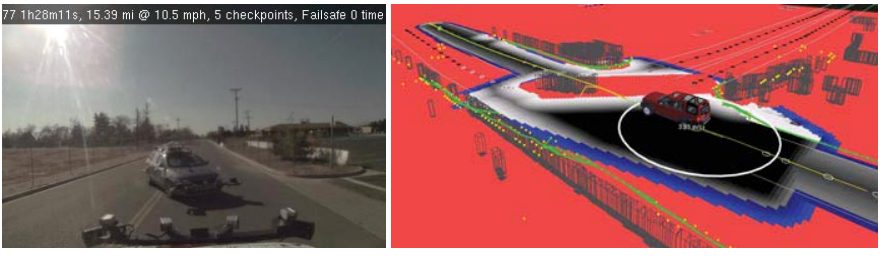
Fig. 19. Drivability Map visualization. [White on Green] Short-term goal location. [Red] Infeasable regions are off-limits to the vehicle. [Blue] Restricted regions may only be entered if the vehicle can stop in an unrestricted region further ahead. [White or Gray] High-cost regions indicate regions accessible to the vehicle.

then carve out the current and oncoming lanes as drivable. Given no obstacles, Talos will then plan a passing trajectory around the stopped vehicle. Static obstacles are rendered in the drivability map as infeasible regions expanded by an additional 30cm to permit a hard margin of safety. If the vehicle makes no progress for a prolonged period of time this margin reduces down to 0 to enable the vehicle to squeeze through a tight fit. The vehicle still should not hit an observed obstacle. As mentioned previously, no explicit vehicle detection is done; instead, moving obstacles are rendered in the Drivability Map with an infeasible region projected in front of the vehicle in proportion to the instantaneous vehicle velocity. As shown in Figure 20(a), if the moving obstacle is in a lane, the infeasible region is projected down the lane direction. If the moving obstacle is in a zone, there is no obvious intended direction so the region is projected in the velocity direction only. In an intersection the obstacle velocity direction is compared with the intersection exits. If a good exit candidate is found, a second region is projected from the obstacle to the exit waypoint (Shown in Figure 20(b)).

Originally only the length of the path that did not collide with an obstacle or lane was used to find optimal trajectories. This could select paths very close to obstacles. A significant refinement of this approach was the inclusion of the notion of risk. In the refined approach, when evaluating the feasibility of a trajectory, the Drivability Map also returns a penalty value, which represents how close the trajectory is to constraints such as obstacles and lane boundaries. The Motion Planner uses the sum of this penalty and the time required to reach the goal point as the cost of each trajectory. Using this combined metric, the best trajectories tend to stay away from obstacles and lane boundaries, while allowing the car to get close to constraints on a narrow road.



(a)



(b)

Fig. 20. (a) An infeasible region is projected down the lane excluding maneuvers into oncoming vehicles. (b) Within an intersection an infeasible region is created between a moving obstacle and the exit matching the velocity direction.

Object tracks where the intended path was not known, such as in intersections or zones, were propagated by three seconds using a constant velocity model.

5.2.1 Lane Boundary Adjustment

When the vision system is used as the primary source of lane estimates, the difference between the RNDF-inferred lanes and the vision-based lanes can be significant. When the vision system suddenly loses a tracked lane or acquires a new lane, the lane estimate can jump by more than a meter, rendering the current pose of the car in an "infeasible" region (outside of the estimated lane). To provide the Motion Planner with a smooth transition of lane boundary constraints, the lane boundaries are adjusted using the current vehicle configuration. Figure 21 shows a case where the vehicle is not inside the latest estimate of the lane boundaries. By marking the region from the current configuration to some point in front on the lane as drivable, the adjustment resolves the initial infeasibility issue. This is also useful when the car happens to drive across a lane boundary, because the Motion Planner will no longer apply hard braking simply because the car has violated a lane boundary constraint.

A similar adjustment is also made when the vision system does not detect any signs of a lane, but curbs are detected by the lidars. In such a case, the RNDF-inferred lanes are adjusted to match the curbs, to avoid having conflicting constraints for the curbs and RNDF-inferred lanes.

Fig. 21. "Stretchy" lane adjustment. When the vehicle is off the lane, the lane is adjusted so that the vehicle does not brake as a result of crossing the lane boundary.

Each iteration of the Motion Planner performs many checks of potential trajectories against the Drivability Map, so for computational efficiency the Drivability Map runs in a separate thread inside the Motion Planner module.

5.3 Motion Planner

The Motion Planner receives an RNDF point from the Navigator as a goal. The output is a path and a speed command that the low-level controller is going to use. The plan to the controller is sent at 10 Hz. The approach is based on the Rapidly-exploring Random Tree (RRT) (LaValle and Kuffner, 2001), where the tree of kino-dynamically feasible trajectories is grown by sampling numerous points randomly. The algorithm is shown in Algorithm 2. The basic idea is to generate a sample and run the forward simulation of the vehicle-controller system. The simulated trajectory is checked with the Drivability Map, and the sample is discarded or added to the tree based on the feasibility. In order to efficiently generate the path in the dynamic and uncertain environment, several extensions have been made (Frazzoli, 2001) to the standard RRT, as discussed in the subsections below.

5.3.1 Planning over Closed-Loop Dynamics

The first extension is to sample the input to the controller and run closed-loop simulation. RRT approaches typically sample the input to the vehicle. However, if the vehicle is unstable, it is difficult for random sampling to construct stable trajectories. Furthermore, the input to the vehicle must change at a high rate to achieve smooth overall behavior, requiring either samples be taken at a very high rate or an arbitrary smoothing process be used. By first closing the loop on the vehicle with a

Algorithm 2. RRT-based planning algorithm

```
1: repeat
2:     Receive the current vehicle states and environment.
3:     Propagate the states by the computation time limit.
4:     repeat
5:         Take a sample for the input to the controller
6:         Select a node in the tree using heuristics
7:         Propagate from the selected node to the sample
8:         if The propagated path is feasible with the drivability map then
9:             Add branch nodes on the path.
10:            Add the sample and the branch nodes to the tree.
11:            for Each newly added node v do
12:                Propagate to the target
13:                if The propagated path is feasible with the Drivability Map then
14:                    Add the path to the tree
15:                    Set the cost of the propagated path as the upper bound of cost-to-go at v
16:                end if
17:            end for
18:        end if
19:    until Time limit is reached
20:    Choose the best trajectory in the tree, and check the feasibility with the latest Driv-
       ability Map
21:    if The best trajectory is infeasible then
22:        Remove the infeasible portion from the tree and Go to line 20
23:    end if
24:    Send the best trajectory to the controller
25: until Vehicle reaches the target.
```

stabilizing controller and then sampling the input to the vehicle-controller system, our approach easily handles vehicles with unstable dynamics.

The behavior of the car is then predicted using forward simulation. The simulation involves a model of the vehicle and the exact same implementation of the execution controller that is discussed in Subsection 5.4. Because the controller tracks the reference, the prediction error of this closed-loop approach is much smaller than the open-loop prediction that uses only the vehicle dynamics in a forward simulation. As shown in Figure 22, the tree consists of the input to the controller (shown in blue) and the predicted trajectory (shown in green and red).

This closed-loop RRT has several further advantages. First, the forward simulation can easily incorporate any nonlinear controller or nonlinear dynamics of the vehicle. Second, the output of the closed-loop simulation is dynamically feasible by construction. Third, since the controller handles the low-level tracking, the RRT can focus on macro behaviors by giving the controller a straight-line path to the target or a path that follows the lane center. This significantly simplifies the tree expansion and is suitable for real-time planning.

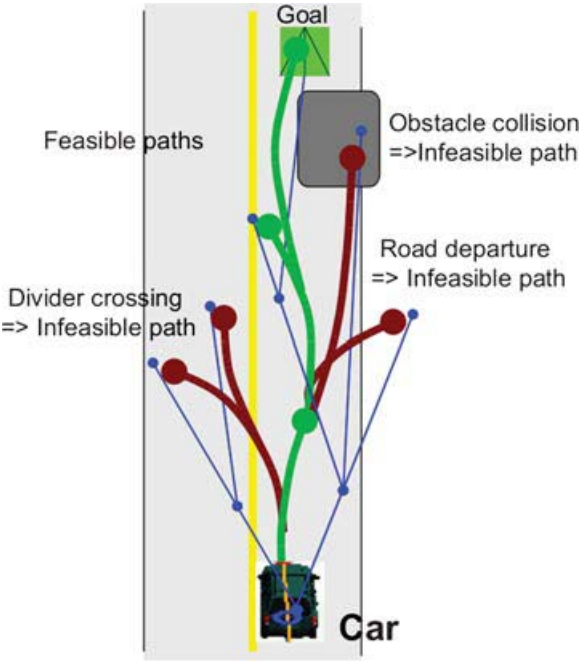


Fig. 22. Illustration of RRT Motion planning. Each leaf of the tree represents a stopping location. The motion control points (in blue) are translated into a predicted path. The predicted paths are checked for drivability (shown in green and red).

5.3.2 Maintaining Safety as an Invariant Set

Ensuring the safety of the vehicle in a dynamic and uncertain environment is the key feature of our planning system. Under normal driving conditions, once a car comes to a stop, it can stay there for indefinite period of time and remain safe (Schouwenaars et al., 2004). Using this stopped state as a safe invariant state, our RRT requires that all the branches in the tree end with a stopped state. The large circles in Figure 22 show the stopping nodes in the tree, and each forward simulation terminates when the car comes to a stop. The existence of the stopping nodes guarantees that there is always a feasible way to come to a safe stop when the car is moving. Unless there is a safe stopping node at the end of the path, Talos does not start executing it.

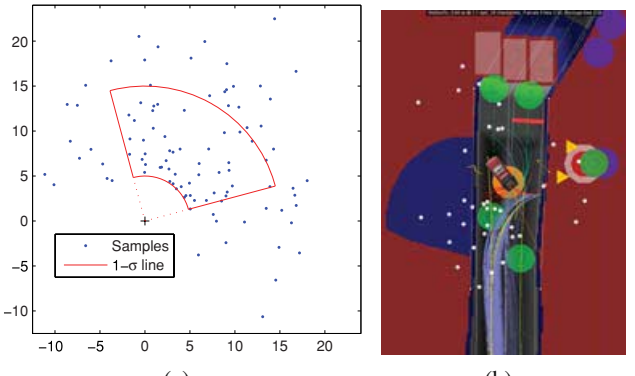
5.3.3 Biased Sampling

Another extension to the RRT algorithm is that it uses the physical and logical structure of the environment to bias the sampling. The samples are taken in 2D and they are used to form the input to the steering controller. To take a sample $(x_{\text{sample}}, y_{\text{sample}})$, the following equation is used

$$\begin{bmatrix} x_{\text{sample}} \\ y_{\text{sample}} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + r \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$$
$$r = \sigma_r |n_r| + r_0$$
$$\theta = \sigma_\theta n_\theta + \theta_0$$

where n_r and n_θ are random variables that have Gaussian distributions, σ_r and σ_θ give the 1-σ values of the radial and circumferential direction, r_0 and θ_0 are the offsets, and (x_0, y_0) is the center of the Gaussian cloud. Figure 23(a) shows 100 samples and the 1-σ lines, with the following parameter values: $\sigma_r = 10$, $\sigma_\theta = \pi/4$, $r_0 = 5$, $\theta_0 = \pi/3$, and $(x_0, y_0) = (0, 0)$. Different bias values are used based on the vehicle location, such as a lane, an intersection, or a parking lot. The situational information from the Navigator such as speed limits, passing allowed, and U-turn allowed, was also used to generate different sampling biases.

Figure 23(b) shows the samples generated while designing a U-turn maneuver. To perform general N-point turns in cluttered environments, the sampling includes both the forward and reverse traveling directions. A cone of forward samples is generated to the left front of the vehicle to initiate the turn (it appears at the top left of the road shown). A set of reverse samples is also generated, which appears to the right of the road shown. These samples will be used after executing the first forward leg of the turn. Then, another set of forward samples is generated to the left of the current vehicle location (it appears at the bottom left of the road shown), for use when completing the turn. For example, the parameter values used for each of these three sets determining a U-turn maneuver are, respectively, $\sigma_{r1} = 8$, $\sigma_{\theta 1} = \pi/10$, $r_{01} = 3$, $\theta_{01} = 4\pi/9$; $\sigma_{r2} = 10$, $\sigma_{\theta 2} = \pi/10$, $r_{02} = 5$, $\theta_{02} = -\pi/4$; $\sigma_{r3} = 12$, $\sigma_{\theta 3} = \pi/10$, $r_{03} = 7$, $\theta_{03} = \pi$. The first Gaussian cloud is centered on the location of the vehicle before initiating the U-turn maneuver, whether the two other clouds are centered on the location of the previous sample.



(a) (b)

Fig. 23. (a) Biased Gaussian samplings. The x, y axes are in [m]. (b) Biased sampling for three-point turns – Talos shown in position after the first forward leg of the turn.

The use of situational/environmental structure for biasing significantly increases the probability of generating feasible trajectories, making the RRT suitable for the real-time applications. Team MIT used a single planner for the entire race, which shows the flexibility and the extensibility of this planning algorithm.

5.3.4 Lazy Re-evaluation

In a dynamic and uncertain environment, the situational awareness is constantly changing, but checking the feasibility of the entire tree against the latest Drivability Map is time consuming. The system checks the feasibility of the path when it is generated (Algorithm 2, line 8), but not re-evaluate its feasibility until it is selected as the best path to be executed (Algorithm 2, line 21). This "lazy check" approach significantly reduced the time spent checking the feasibility using the Drivability Map, but still ensured that the path that was sent to the Controller was always feasible with respect to the latest perceived environment.

5.4 Controller

The Controller takes the motion plan and generates gas, brake, steering and gear shift commands (collectively referred to as the control signals) that track the desired motion plan. The motion plan contains the same information as the controller input used in the planner prediction, and consists of a list of (x, y) points that define the piece-wise linear reference path for the steering controller and the associated reference speed. The Controller has two components: a pure-pursuit steering controller and the proportional-integral (PI) speed controller. A pure-pursuit algorithm is used for steering control because it has demonstrated excellent tracking performance for both ground and aerial vehicles over many years (Kelly and Stentz, 1997; Park et al., 2007). A simple PI controller is implemented to track the commanded speed. These two core modules are embedded in the execution controller, but also within the motion planner for trajectory prediction, as discussed in Subsection 5.3. The generated control signals are sent to ADU for actuation, and the Controller loop runs at 25 Hz.

5.4.1 Steering Controller

The low-level steering control uses a modified version of the pure pursuit control law (Kelly and Stentz, 1997; Park et al., 2007) to steer the vehicle along the desired path. The steering control law is given by

$$\delta = -\tan^{-1}\left(\frac{L\sin\eta}{\frac{L_1}{2} + l_a\cos\eta}\right)$$

where L is the constant vehicle wheelbase, l_a is the constant distance between the pure pursuit anchor point and the rear axle, η is the angle between the vehicle heading and reference path direction, and L_1 is the look-ahead distance that determines how far ahead on the reference path the controller should be aiming.

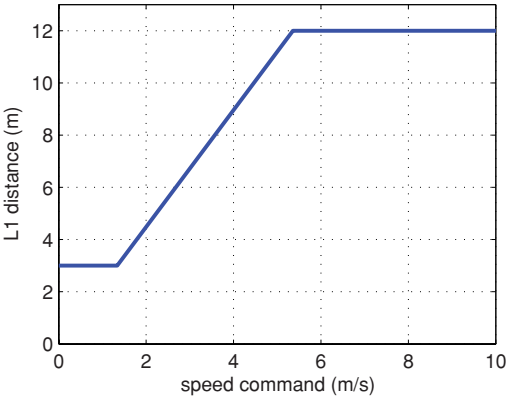


Fig. 24. L_1 distance as a function of the commanded speed.

A smaller L_1 produces a high-gain controller with better tracking performance. However, to ensure stability against the system delay, L_1 must be enlarged with speed (Park et al., 2007). Figure 24 plots the relation between the L_1 and the commanded speed. The L_1 has a minimum value to ensure that the controller is stable at low speed. The L_1 is also capped from above, to ensure that the look-ahead point stays on a path within a reliable sensing range.

To improve trajectory tracking performance, the controller scales L_1 as a function of the commanded speed. Up to the time of site visit in June 2007, L_1 was determined as a function of the measured vehicle speed. The result was that any error in the speed prediction would translate into a different L_1 being used by the Motion Planner prediction and the Controller execution, which effectively changes the gain of the steering controller. In the final approach, the RRT planner determines the commanded speed profile, with the result that the speed and steering controllers are decoupled.

5.4.2 Speed Controller

The speed controller is a low-bandwidth controller with the following gains

$$u = K_p(v - v_{\text{ref}}) + K_i \int (v - v_{\text{ref}})dt$$
$$K_p = 0.2$$
$$K_i = 0.04.$$

The output of the speed controller u is a normalized value between -1 and $+1$. Using a piecewise linear mapping shown in Figure 25, u is converted to the voltage command to ADU. Note that the initial testing revealed that the EMC vehicle interface has a deadband between 1850 mV and 3200 mV. To achieve a smooth coasting behavior, when the normalized controller output is small, (i.e. $|u| \leq 0.05$), no gas or brake is applied. To skip the deadband and quickly respond to the controller command, the small positive output ($u = 0.05$) corresponds to the upper limit of the

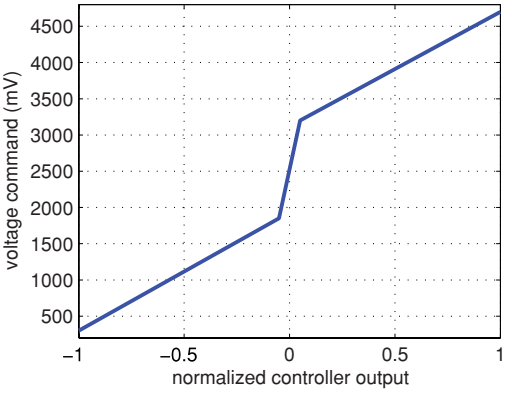


Fig. 25. Conversion from the speed controller output to the ADU command voltage.

deadband 3200 mV, and the small negative output ($u = -0.05$) corresponds to the lower limit of the deadband 1850 mV.

To help reduce the prediction error, the commanded speed is tied to the predicted vehicle location, rather than time. The time-based reference leads to a coupling between the steering and speed controllers, even when L_1 is scheduled as a function of the commanded speed. For example, if the actual vehicle speeds up slower than the prediction with a ramp-up speed command, the time-based speed command would make L_1 larger than the predicted L_1 when reaching the same position. This difference in L_1 can lead to significant steering error. The space-based reference makes the steering performance relatively insensitive to these types of speed prediction errors.

6 Challenge Results

To complete the DARPA Urban Challenge, Talos successfully negotiated first the National Qualifying Event (NQE) and then race itself. This section reviews the vehicle's performance in these events.

6.1 National Qualifying Event (NQE) Performance

The NQE trials consisted of three test areas. Area A tested merging into traffic and turning across traffic. Area B tested navigation in suburban crescents, parking and passing of stopped vehicles. Area C tested intersection precedence and route blockage replanning. The NQE was also the first chance to test Talos in a DARPA-designed course and RNDF. On day one we were testing not only our ability to complete the mission, but also the compatibility of coordinate systems and RNDF conventions. Team MIT completed one mission a day for the first three days of the qualifier, with a five-mission endurance test on the fourth day, as shown in Table 1.

Successful negotiation of the NQE trials, and later, the race, required macro-level behavior tuning to manage trade-offs in uncertain scenarios:

Table 1. Results for all of Talos' NQE tests

Day	Date	NQE Schedule	Outcome
1	Sat. 27^{th} Oct	Area B 1^{st} trial	Completed.
2	Sun. 28^{th} Oct	Area C 1^{st} trial	Completed, but went around the roadblock.
3	Mon. 29^{th} Oct	Area A 1^{st} trial	Completed – Safe, but slow (7 laps in 24 minutes)
4	Tue. 30^{th} Oct	Area B 2^{nd} trial	Still progressing, but ran out of time.
		Area C 2^{nd} trial	Went off-road after 2^{nd} K-turn at blockage.
		Area A 2^{nd} trial	Completed – Safe and faster (10 laps in 12 minutes)
		Area B 3^{rd} trial	Completed.
		Area C 3^{rd} trial	Completed after recovery from K-turn at first blockage.
5	Wed. 31^{st} Oct	—	

- No progress due to a road blockage versus a perception failure (such as a mis-detected curb cut).
- No progress due to a vehicle to queue behind & pass versus a perception failure (like a lane positioning error).
- Safe versus overly cautious behavior.

After leaving the start chute, Talos was reluctant to leave the Start Zone. The boundary from the raised Start Zone into Challenge Lane was in fact a six-inch drop smoothed by a green ramp. This drop-off was detected by our vehicle as a ditch. Figure 26(a) shows how the drop-off appeared to our vehicle. Reluctant to drive down such a drop-off, the vehicle looked for an alternate route. Unable to make progress, the failsafe logic eventually relaxed the constraint that had been avoiding the ditch. The vehicle then drove down Challenge Lane.

Figure 26(b) shows how our system relies on local perception to localize the RNDF map data. The lane ahead of the vehicle is dilated, representing the potential ambiguity in where the lane may actually be. The dilation contracts to the lane position at certain control points either because of a close GPS waypoint or lane tracking.

During Talos' first parking attempt, we struck the first difference in the way Team MIT and DARPA interpreted the RNDF. Figure 26(c) shows that the goal point our vehicle is trying to drive to is under the parked vehicle in front. For parking spots and other checkpoints, we attempted to get the vehicle center to cross the checkpoint. To achieve this, we placed the goal location ahead of the checkpoint to make the vehicle pass over the checkpoint. The positioning of the vehicles and the parking spots indicates that DARPA simply required the vehicle to drive up to the checkpoint in this test. The sampling strategy of the RRT planner assumed that the parking spot was empty. The blocked parking spot caused many of the samples to be discarded because the last portion of the trajectory was infeasible. This is why Talos spent more than a minute in the parking zone. For the final race, a new sampling strategy was developed that caused Talos to come as close to the checkpoint in the parking spot as possible, which can be performed much more quickly. This figure also shows some transient phantom obstacle detections caused by dust in the gravel parking zone to the left of Talos.

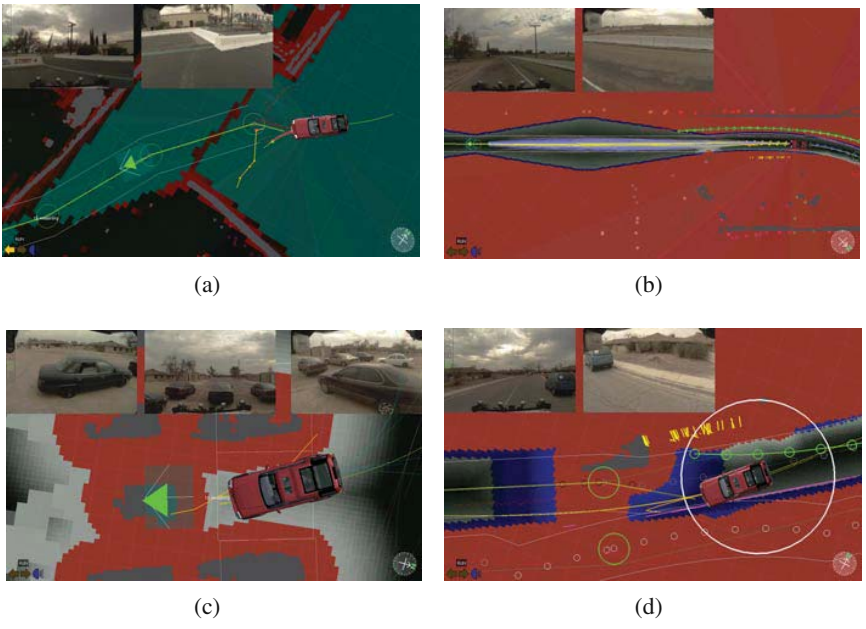


Fig. 26. Area B 1^{st} trial highlights. (a) Road-hazard map showing red line along end of zone. The drop-off onto Challenge Lane was detected as a ditch. (b) Lane position uncertainty between control points reflected by lane dilation. The road past where Talos can perceive it is dilated reflecting the potential ambiguity in lane position. (c) Parking goal position under car in front. (d) a virtual blockage used to enforce passing behavior causing undesired results.

To ensure that Talos would queue behind a slow-moving vehicle yet still pass a stationary vehicle or obstacle, the system was designed to artificially choke off the road beside an obstacle in the lane. Since Talos could then not make progress, it would wait 10 seconds to determine if the obstacle was a vehicle moving slowly or a stationary object. If the object remained still, Talos would begin a passing maneuver. In the Gauntlet, this choke-off behavior misfired. Figure 26(d) shows our vehicle waiting to go into passing mode beside a parked car. The road curvature causes the obstacle to appear more directly in our lane than was actually the case. Stuck for a time between the impassable regions generated from the vehicle on the right and a Drivability Map rendering artifact on the left, Talos entered into the failsafe mode with relaxed lane boundary constraints. Talos then sailed through the rest of the Gauntlet and completed the mission. Note that the parked cars and obstacles still appear as red infeasible regions off-limits to the vehicle.

Area C tested intersection precedence and blockage replanning. The vehicle did very well at intersection precedence handling in many different scenarios. Figure 27(a) shows Talos correctly giving precedence to three traffic vehicles before going ahead of the second oncoming traffic vehicle. Figure 27(b) shows Talos queueing behind a traffic vehicle before giving precedence at the intersection.

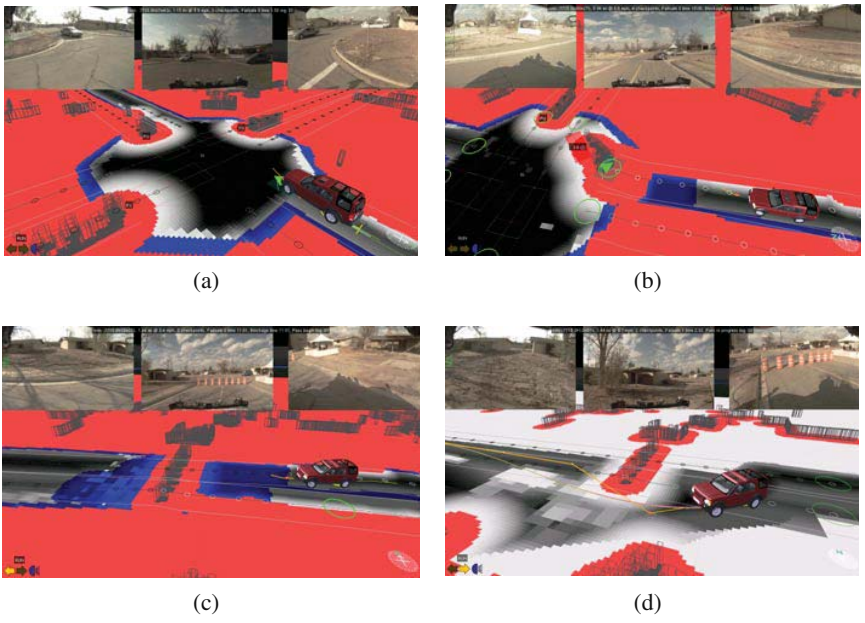
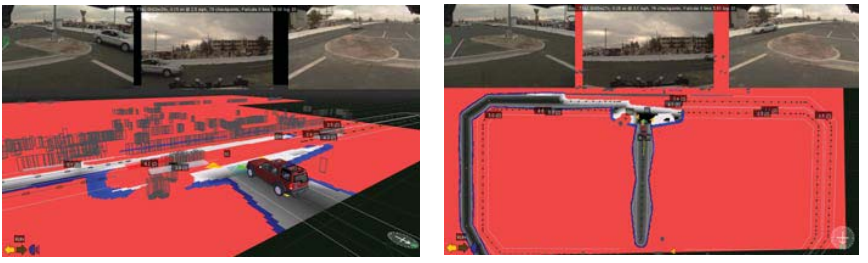


Fig. 27. Area C 1st trial highlights. (a) Intersection precedence with four traffic vehicles. (b) Queuing before an intersection. (c) Attempting to go around a blockage. (d) "Failsafe mode 1" permits the vehicle to go around the blockage.

Blockage replanning was more challenging. Talos correctly detected and stopped at the line of traffic barrels (see Figure 27(c)), and then its programming caused it to try a passing attempt to drive around the blockage. After a predetermined period where no progress was made, the system relaxed some of the perception constraints (to account for the possibility that the road position had been poorly estimated, for example) or declare a blockage and turn around. At the time of this test, the logic was set up to relax the lane constraints prior to declaring a blockage, assuming that a blockage would be truly impassable. Section 5.1.3 contains the logic. Figure 27(d) shows the perceived world once in "Failsafe mode". Once the lane constraints were dropped, the route around the blockage was high cost, but passable, so Talos drove around the blockage and finished the mission.

The Area A trial was a merging test with human-driven traffic vehicles. Leading up to the trial, much emphasis was placed on safety, so on the evening before the trial Team MIT reexamined and tested the logic used to determine when it was safe to merge. Increased caution and an unexpected consequence of a bug fix prompted the increase of Talos' safety margin for merging from an 8-second window to a 13-second window. As a result, during the Area A trial, Talos performed safely, but very cautiously, as it was waiting for a 13-second window in the traffic, and such a window rarely appeared. In the 24-minute trial, Talos completed only 7 laps.



(a) (b)

Fig. 28. Area A 1st trial highlights. (a) Vehicle approaching on the right is tracked despite being occluded by a closer vehicle. (b) High traffic density makes for a long wait.

Figure 28(a) shows the vehicle track on the right approaching at 3.5m/s despite being occluded by a vehicle in the closer lane tracked by the radar and lidar as traveling at 3.9m/s and 4.3m/s, respectively.

Although the failsafe mode permitted Talos to complete the first Area B trial, the team decided to fix the bugs that led to Talos ending up in this mode and only use it as a last resort. On the second trial at Area B, many of the bugs seen during the first trial were fixed, including: the dip leading out of the Start Zone, the rendering artifact bug, and the parking spot location ahead of the checkpoint. However, a few new issues arose.

On its route through the Gauntlet, Talos became stuck due to a combination of a poor lane estimate, the virtual object used to force a passing behavior, and a bug that would not permit Talos to go into passing mode if it was not fully within the estimated lane, as shown in Figure 29(e). Eventually Talos made no progress for long enough that a blockage was assumed. Talos then planned to turn around and approach the checkpoint from the opposite direction. Figures 29(a) and (b) show the originally planned route and the alternate route through the Gauntlet from the opposite direction, respectively. Talos completed the Gauntlet in the opposite direction and then needed to turn around again to hit the original checkpoint. Figure 29(c) shows the intended route since the blockage now seems to have been removed. En-route, Talos came across a legitimate road blockage shown in Figure 29(f). Talos completed a K-turn and executed the revised plan shown in Figure 29(d). Talos continued to make progress, but given the extra distance traveled, it ran out of time before completing the course.

During the second trial of Area C, the macro-behavior tuning had improved such that Talos correctly inserted a blockage and made a K-turn instead of simply driving around the blockage. However, during the second K-turn on the far side of the blockage, a poor lane estimate and a restricted region generated by obstacles perceived to be in the lane conspired to stall progress (shown in Figure 30(a)). Eventually, Talos entered failsafe mode and proceeded with relaxed lane constraints and a reduced restricted region. Unfortunately, as Talos was driving out of a successful K-turn the no-progress timer triggered and Talos reconsidered its blockage choice. Talos elected to block the current path and try the original route. In the recovery mode

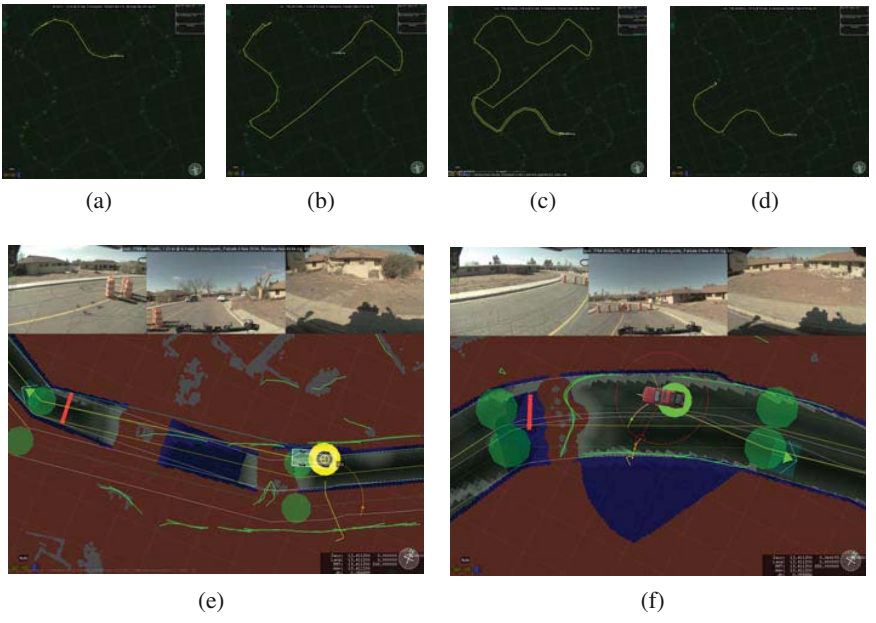


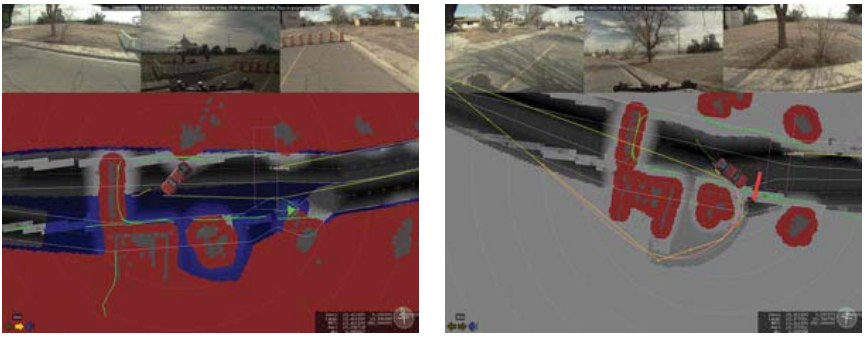
Fig. 29. Area B 2^{nd} trial highlights. (a), (b), (c), & (d) show a sequence of navigator plans. After a blockage is declared in the Gauntlet, Talos attempts to reach the checkpoint from the opposite direction. (e) Talos gets stuck and cannot go into passing mode due to a poor lane estimate resulting in the appearance that it was not entirely in its lane. (f) This blockage was real. Talos detects the blockage and completes a K-turn.

Talos was enabled to drive across curbs, behind the DARPA observation tent and around the blockage to complete the mission (shown in Figure 30(b)). The DARPA officials intervened.

For the second trial of Area A, the safety margin for merging (which, at 13 seconds, had caused a significant amount of waiting in the first trial) was reduced to 9 seconds. The planning sequence was also modified so that the RRT planner could prepare paths for Talos to follow while the Navigator waited for the crossing traffic to clear the intersection. This improved the response time of the vehicle, and the overall results were much better, with 10 laps in just 12 minutes. Figure 31 shows a photo of Talos and a screenshot of the viewer output for this mission.

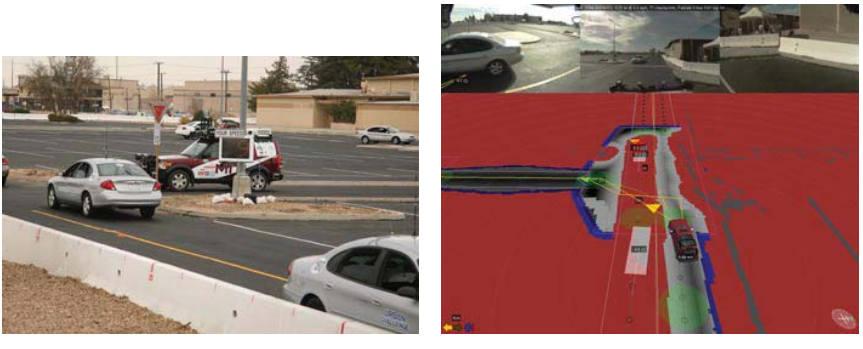
Figure 32 illustrates Talos' performance in its Area B 3^{rd} trial. In the Gauntlet, the artificial choke on the road was removed, and passing was successful. Talos got stuck on curbs a few times, probably due to inaccurate lane estimates, but otherwise executed the mission well.

A consequence of our decision to treat environmental perception as a higher authority than map data was that, at times, the lane estimate would snap to a new confident estimate. Some basic transitioning was implemented in the drivablility map to attempt to smooth the transition. In the best case the Motion Planner would discover a new trajectory to the goal within the next planning iteration. If no forward



(a) (b)

Fig. 30. Area C 2nd trial highlights. (a) Progress is stalled during the second K-turn by a poor lane estimate and a restricted region. (b) Failsafe mode is entered and would have permitted a successful K-turn, except that the goal location now reverts to the far side of the blockage.



(a) (b)

Fig. 31. Area A 2nd trial highlights. (a) Talos is looking for smaller gaps than in the 1st trial. (b) The RRT planner is working while waiting for oncoming traffic to clear.

plan could be found, the vehicle would begin an emergency brake. Occasionally the vehicle would be placed too close to detected curbs. Although the vehicle footprint is cleared of infeasible curbs, the areas around the vehicle were not edited in this way. If curbs impeded progress, the vehicle would become "ship wrecked". After the no-progress timer got sufficiently high, the curbs were rendered as high cost instead of infeasible and the vehicle would proceed. Figures 32(a) and (b) show how the lane estimate can shift based on new data. The problem is a consequence of limited development time. The intention was to use the detected curbs in the lane estimation process for the race. As described in Section 4.4, the capability in the software was present, but unfortunately the integration was a little too immature to use in the race so the simpler "curbs as obstacles" approach was used.

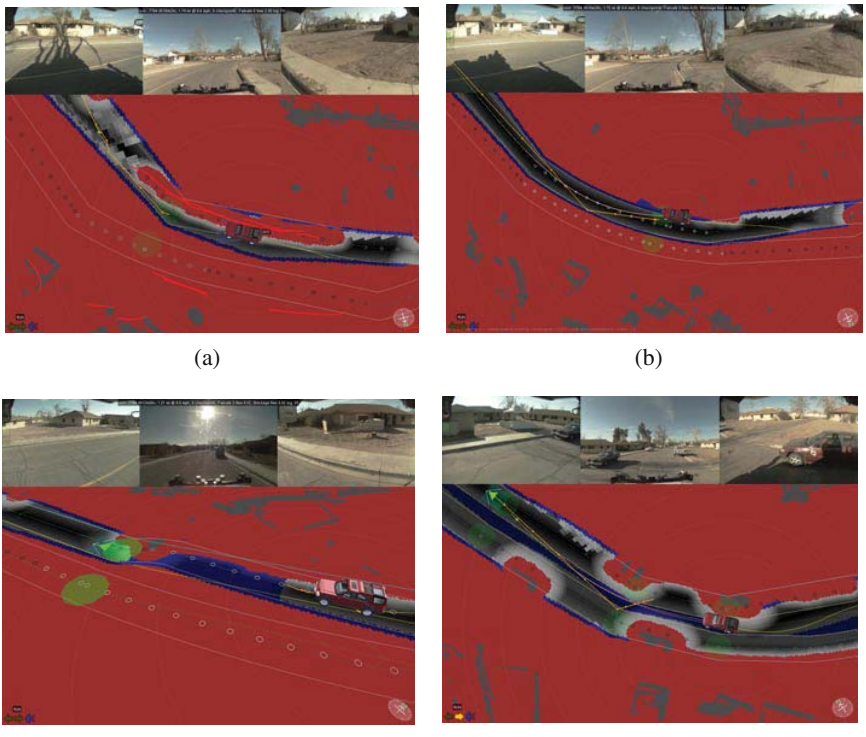
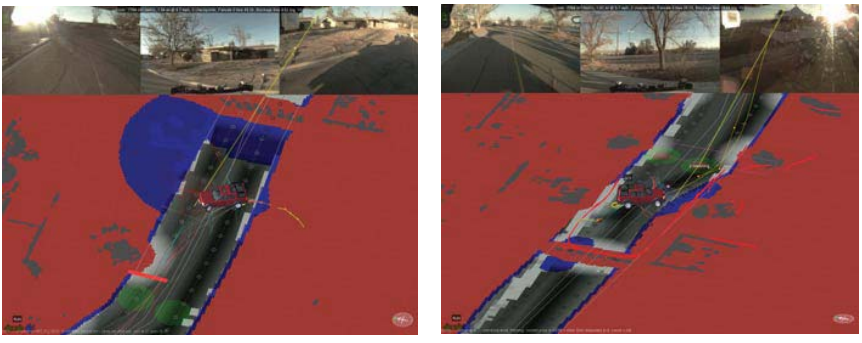


Fig. 32. Area B 3^{rd} trial highlights. (a) Without visual lane tracking, a curb-free space algorithm localizes the lane. (b) Visual lane tracking often recovers, providing an improved road estimate. (c) Without a virtual obstacle, passing will still occur as long as the object occupies enough of the lane. Here the parked car still induces a passing behavior. (d) Once in passing mode, parked cars are easily maneuvered around.

Figure 33 illustrates Talos' performance in its Area C 3^{rd} trial. After correctly detecting the blockage, during the first K-turn, the vehicle drove off the road and the pit crew was called to reposition the vehicle; after this intervention, Talos completed the mission successfully.

6.2 UCE Performance

Overall, the team was very pleased with the performance of the vehicle during the race. Figure 34 shows some general highlights of Talos' performance during the UCE. Figure 34(b) shows Talos driving down Phantom East. The vehicle speed was capped at 25mph as this was the highest speed for which we had validated our vehicle model. The radar in the picture reads 24mph. Figure 34(d) shows Talos queuing patiently behind a metal pipe gate that had blown across an intersection safety region. The gate was later forcefully removed by the DARPA officials. After initially passing the Cornell chase vehicle, Figure 34(c) shows Talos slowing to



(a) (b)

Fig. 33. Area C 3^{rd} trial highlights. After correctly detecting the blockage, Talos begins a K-turn. The navigator's plan changes. Talos enters failsafe mode, drives off-road, and is recovered by the pit crew. (b) After the intervention, Talos resumes, and its second K-turn goes well.

merge safely behind the Cornell chase vehicle as the two lane road merges back to one at the end of George Boulevard. Figure 34(e) shows an incident found while reviewing the logs. Talos correctly yields to a traffic vehicle traveling at over 35mph. The early detection by Talos' radar suite on the crescent road potentially saved the vehicle from a race-ending collision. Finally, Figure 34(f) shows Talos crossing the finish line after completing the final mission.

The race consisted of three missions. Figure 35 shows periods of no progress during the missions. An analysis of the peaks in these plots permits us to examine the principal failure modes during the race. During the first mission, Failsafe Mode 1 was entered twice, once at 750 sec due to being stuck on the wrong side of an artificial zone perimeter fence. Figure 36 shows how the lane next to parking zone is narrow in the RNDF (12ft) – the actual lane is over 30ft and extends into the zone. The lane perception snaps to real lane-edge, trapping the vehicle against the zone boundary virtual fence. The second failsafe mode change came at 7200 sec due to a bad lane estimate on gravel road, which is discussed in Section 6.2.1. Several other times during this first mission Talos was "ship wrecked" and made no progress for 30 sec until the curb constraints were relaxed. In Mission 2 the zone perimeter fence bug occurred at 4000 sec. The third mission required three traversals of the gravel road, which caused a large number of 30 sec intervals of no progress until curb constraints were relaxed. Once at 7200 sec a bad lane estimate on the gravel road caused Talos to enter Failsafe Mode 1.

Outback Road was the Achilles heel of Talos' UCE performance. We now examine the cause.

6.2.1 Outback Road

As noted above, Talos' third mission during the UCE required three traversals of the steep gravel road known as Outback Road. Figure 37 illustrates why three traversals

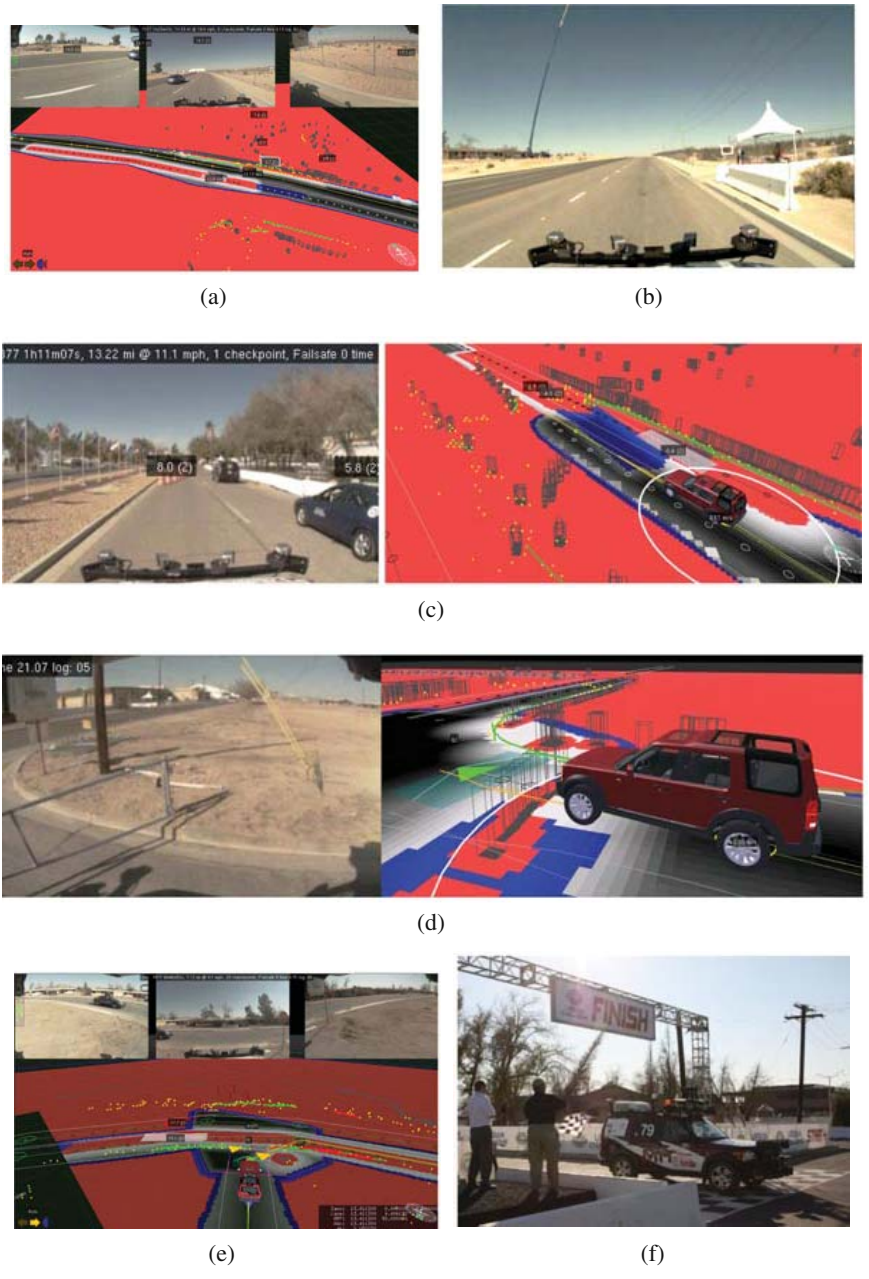
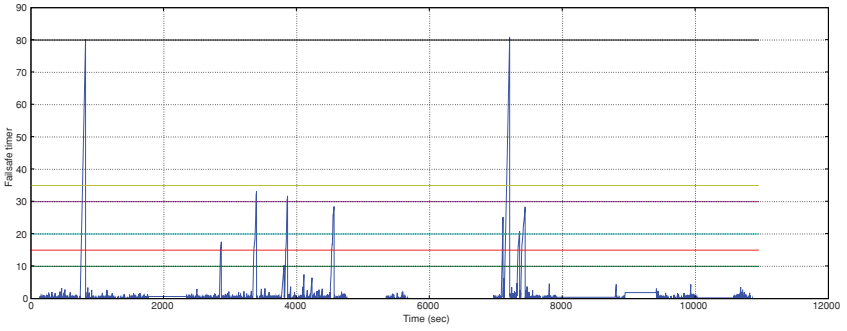
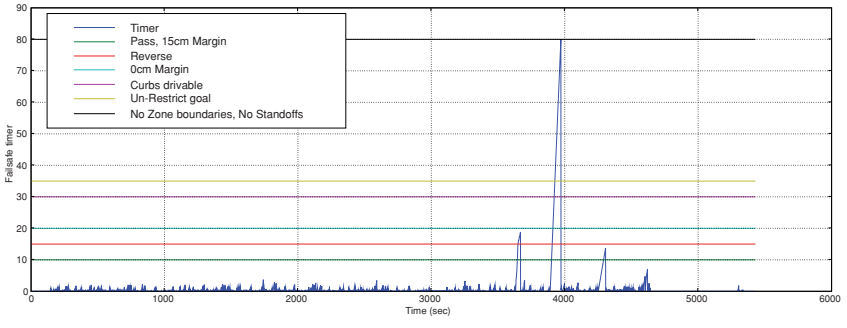


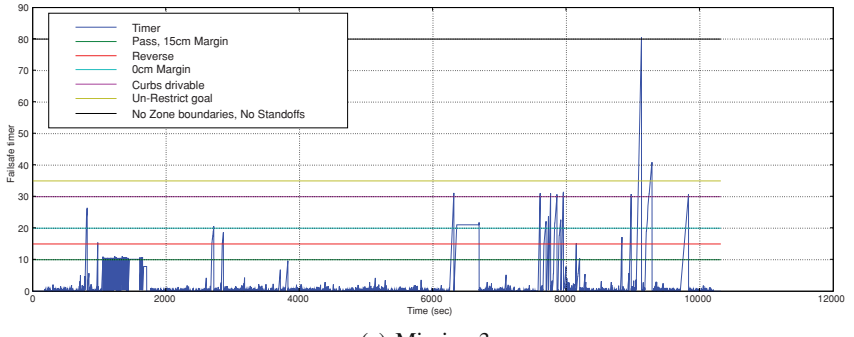
Fig. 34. UCE highlights. (a) Talos overtaken by traffic vehicle. (b) Talos reaches its target speed of 25mph (24 shown on the sign) traveling up Phantom East. (c) Talos slows to merge safely behind the Cornell chase vehicle. (d) Talos waits patiently behind a gate that was blown across an intersection safety zone. (e) A fast traffic vehicle is detected early by radars, and correct intersection precedence keeps Talos in the race. (f) Mission accomplished.



(a) Mission 1



(b) Mission 2



(c) Mission 3

Fig. 35. No progress timer during the race. The timer is stalled during DARPA Pauses. The X axis is the wall clock time. (a) During the first mission, failsafe mode (80 sec of no progress) is entered twice. *750 sec*: Zone perimeter fence bug. *7200 sec*: Bad lane estimate on gravel road. Other times Talos was "ship wrecked" and made no progress for 30 seconds until curb constraints were relaxed. (b) Mission 2: *4000 sec*: Zone perimeter fence bug. (c) Mission 3: *7200 sec*: Bad lane estimate on gravel road. Many times during the three traversals of the gravel road section no progress was made for 30 seconds until the curb constraints were relaxed.

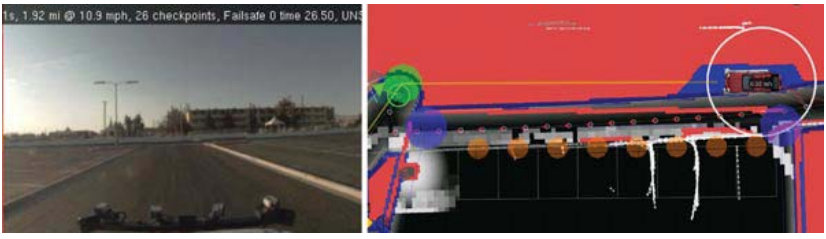


Fig. 36. Lane perception snaps the narrow (12 ft) RNDF lane to the (30 ft) actual lane boundary to the left (*gray circles*: lane centerline, *white lines*: lane boundary detections). The road would be drivable except that the zone virtual boundary (*red line*) extends into the physical lane blocking the road.

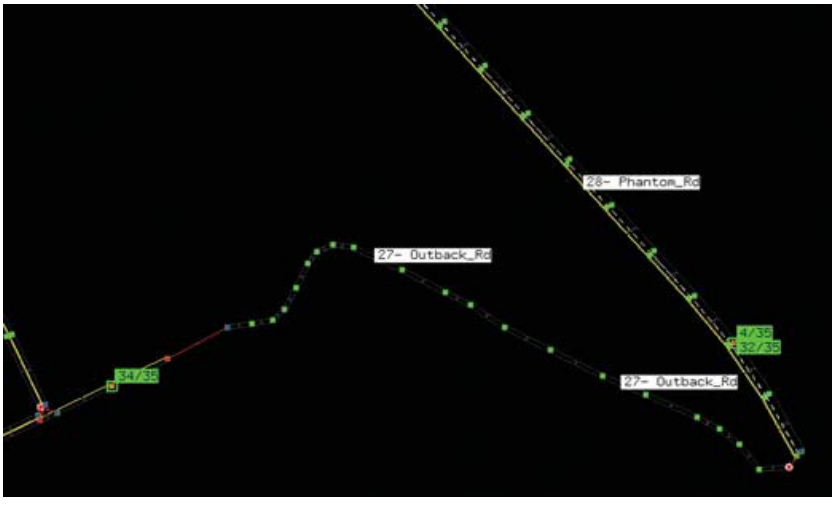


Fig. 37. To complete the third mission, three traversals of the Outback Road were required.

were required to hit the checkpoints. Of the 35 checkpoints in the mission, checkpoints 4, 32, and 34 all required Talos to complete the one-way Outback, Phantom East circuit to hit the checkpoint and complete the mission. As far as we know other teams were not required to complete this circuit more than once.

Figure 38 provides snapshots of Talos' performance while driving down the dirt road; clearly, the system encountered difficulties on this part of the course. The drop-off in the road profile was detected as a phantom curb or ditch. When a steep section of road was directly ahead of Talos, the road-edge detector would occasionally detect the hill as a road edge. (As described below in Section 4.3, the road-edge detector was intended to detect berms as well as curbs.) The road-edge system incorporated a work-around designed to combat this problem: road edges that were strongly perpendicular to the direction of the road (as indicated by the RNDF) were

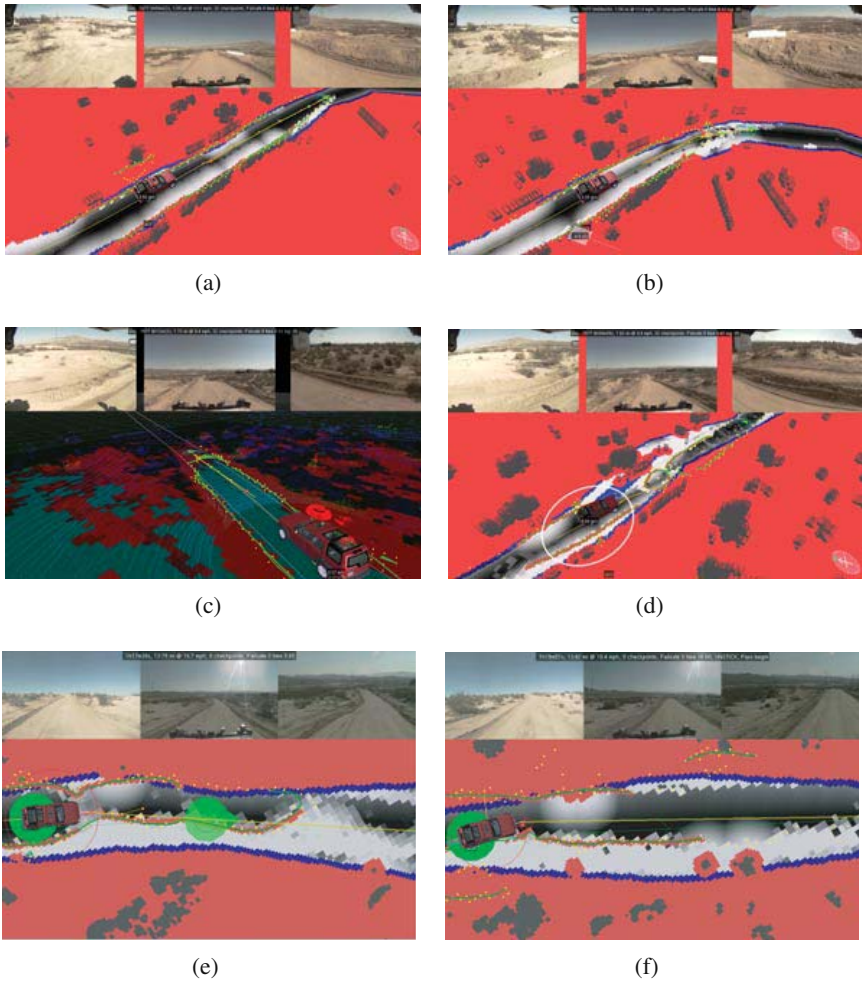
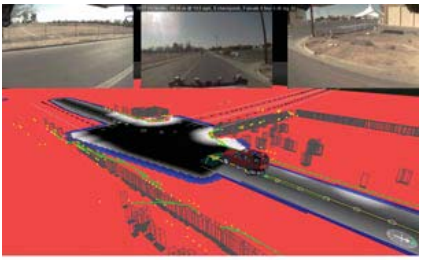
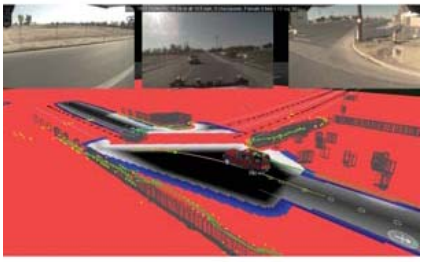


Fig. 38. Steep gravel road. (a) & (b) Talos had no problem with gravel roads of gentle slope. (c) Roll-off of the gravel road appears hazardous in the curb hazard map. (d) A phantom curb is detected at the road crest. (e) & (f) Phantom curb detections contort the road corridor, choking the drivable region.

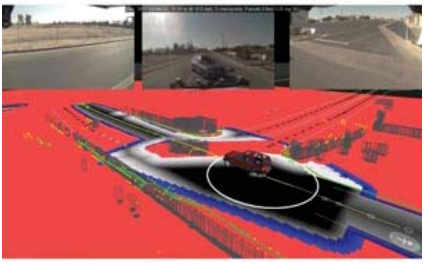
culled. We expected this feature to solve this problem, but it did not. A flat road that only curved in the direction of travel would be detected as a road edge perpendicular to the road. However, the dirt road was crowned (had a side-to-side curvature) that caused the maximum curvature to appear not to be directly across the travel lane, and instead caused it to appear as two diagonal lines converging further down the road. The slope of these lines was sufficiently parallel to the direction of travel that they were not culled. Consequently, Talos would get stuck on the dirt road until a timeout elapsed (at which point the road edges were no longer treated as obstacles).



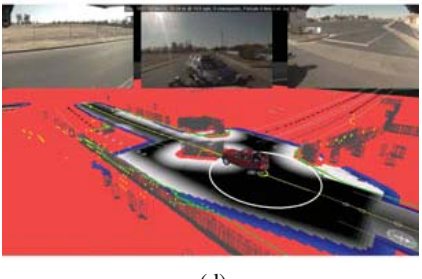
(a)



(b)

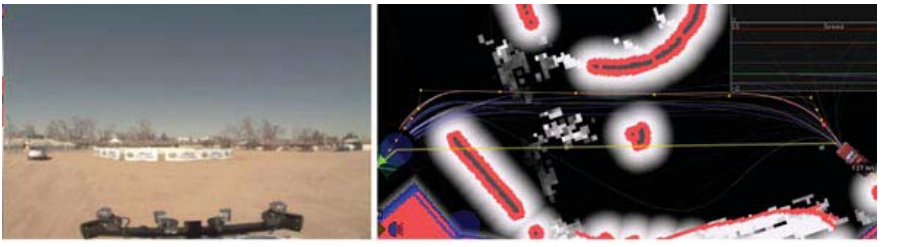


(c)

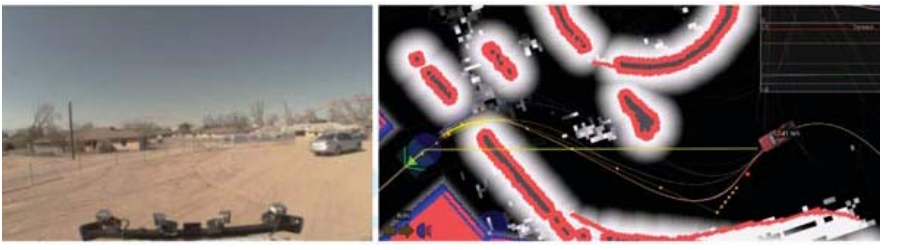


(d)

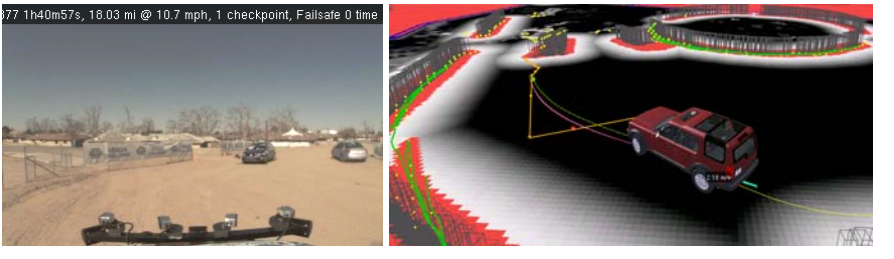
Fig. 39. 1st Team CarOLO's Caroline – Talos near-miss. (a) Talos stops upon entering intersection. (b) Talos detects the moving object across its path and begins to plan a path around it. (c) Talos begins an emergency stop. (d) Talos comes to a stop. Caroline no longer appears to be moving, and instead is viewed as a static object.



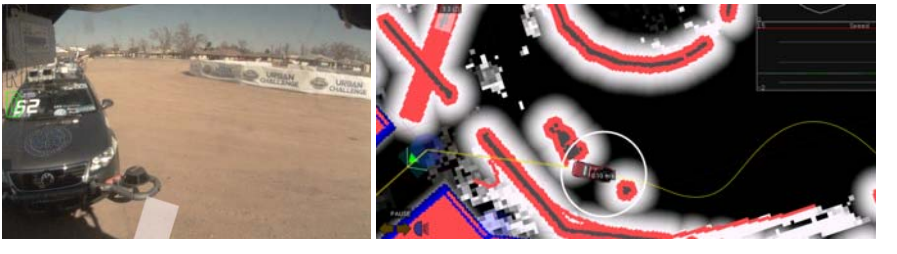
(a)



(b)



(c)



(d)

Fig. 40. Second Caroline-Talos incident. (a) Talos drives around the Caroline chase vehicle. (b) Talos drives around Caroline, which is moving sufficiently slowly to appear as a static object. (c) Talos continues to replan around Caroline, which was perceived as an static object in a different location. (d) Talos continues to drive around Caroline, and then initiates an emergency stop, but cannot stop in the space left and collides with Caroline.

Again, as described earlier, the intended approach of using the curb data in the lane estimate, if mature, would have gone a long way towards addressing this problem.

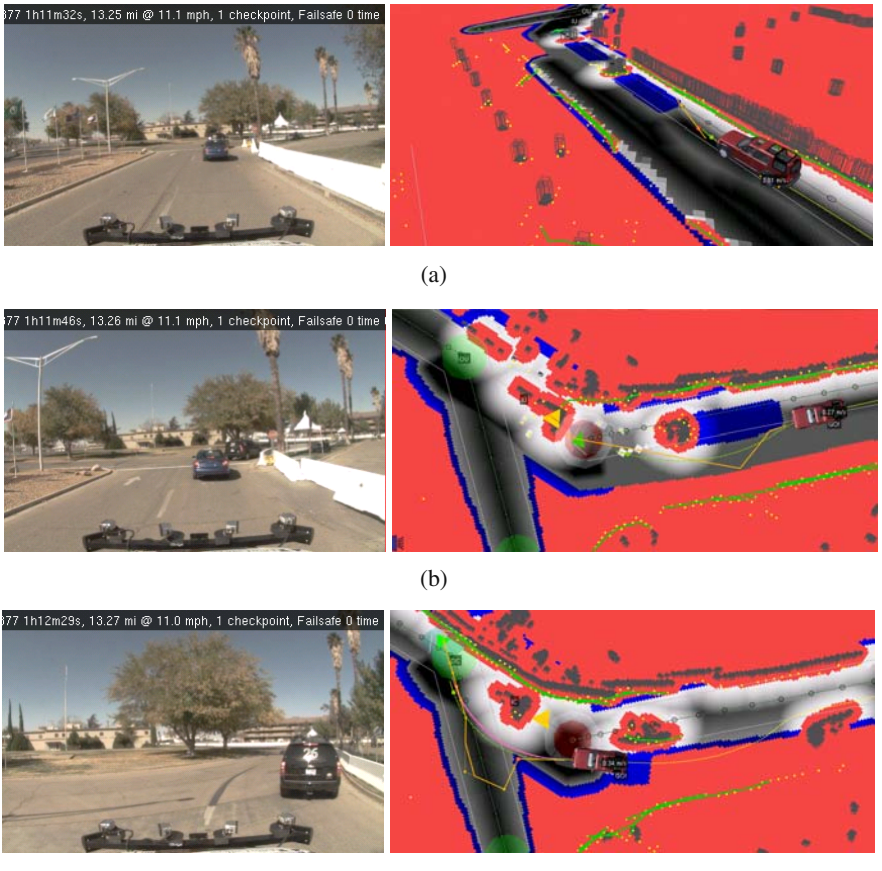


Fig. 41. Lead-up to Skynet – Talos incident. (a) Talos initially queues behind the Skynet chase vehicle. (b) Lane position is low, so Talos finds a route around the chase vehicle. (c) Talos yields at the intersection. There are no moving vehicles so it proceeds through.

6.2.2 Collisions

Our vehicle had two incidents with Team CarOLO's vehicle "Caroline" during the UCE. In the first encounter with Caroline, Talos paused as it entered the intersection, following the process described in Section 5.1.1, and after resuming its forward motion, Caroline attempted to make a left turn directly across Talos' path. The system initiated a "planner e-stop" just before DARPA issued a Pause command to both vehicles. These events are illustrated in Figure 39.

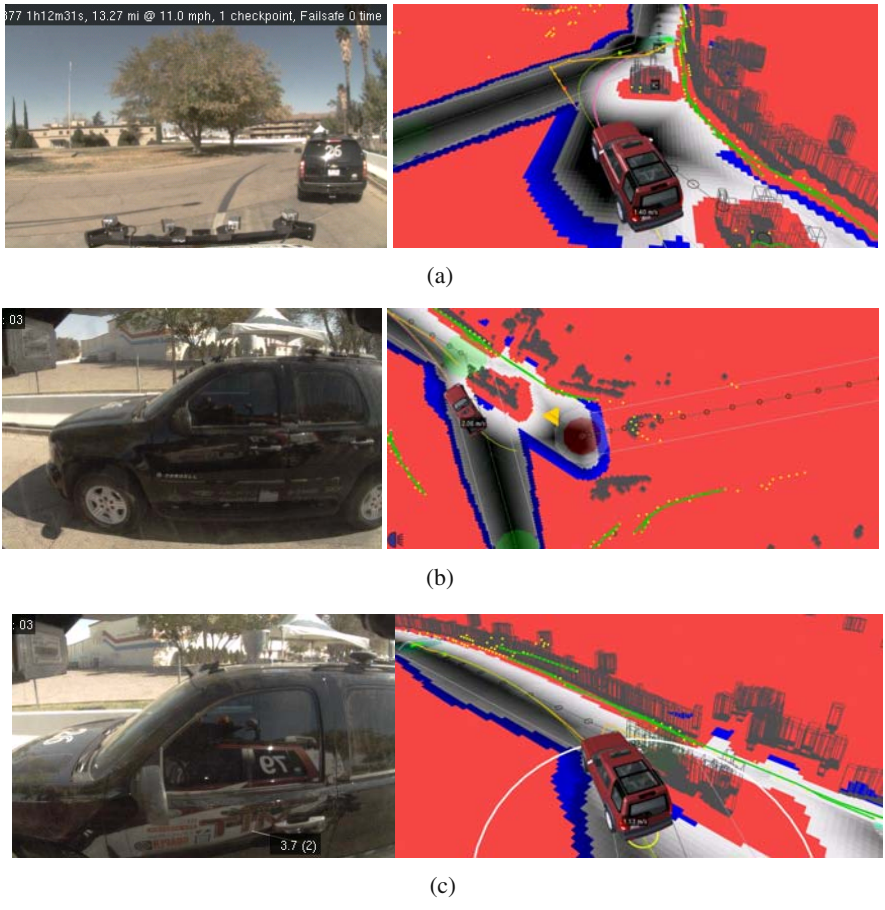


Fig. 42. Skynet - Talos incident. (a) Talos plans a route around Skynet, which appears as a static object. (b) While Talos is passing, Skynet begins to accelerate. (c) While applying emergency braking, Talos turns into the accelerating Skynet.

In a second incident with Team CarOLO's vehicle, Talos was attempting to drive toward the zone exit, between what appeared to be a fence on the left and some static objects to the right. Caroline drove toward Talos, which applied hard braking, but did not come to a stop in time to avoid the collision. We do not know why Caroline did not choose a path through the free space to Talos' right, or why it continued to advance when Talos was directly ahead of it. We had made a software architectural decision not to attempt to explicitly detect vehicles for the Challenge. Instead, Talos simply treated slow or stationary obstacles as static and faster moving obstacles as vehicles. Unfortunately Caroline's speed, acceleration, stopping and starting fell into a difficult region for our software. Talos treated Caroline as a static obstacle and was constantly replanning a path around it (to Talos' left and Caroline's right). Just

before the collision, the system executed "planner emergency stop" when Caroline got sufficiently close. Unfortunately, due to Caroline's speed and trajectory, this could not prevent physical contact. These events are illustrated in Figure 40.

Talos' collision with Cornell's vehicle, Skynet, was another notable incident during the UCE, and is illustrated in Figures 41 and 42. As described earlier in this report, Talos used a perception-dominated system. It was designed to use the waypoints in the RNDF with limited confidence. Upon approaching the intersection, Talos interpreted Skynet's DARPA chase vehicle as being close enough to the road shoulder to be a static feature (such as a tree or barrier on the side of the road). Therefore, the road entry point was oriented to the left of the chase car. Talos drove up, gave way at the intersection, and then continued to the left. Because Skynet and the chase car were stopped, Talos again interpreted them to be stationary obstacles (such as K-rails). Talos drove through the intersection and was attempting to get back into the exit lane when Skynet started to move. Again its speed was below Talos' tolerance for treating it as a moving vehicle, and again Talos would have avoided Skynet if it had remained stationary. As in the collision with Caroline, Talos was applying emergency braking when it collided with Skynet. The root cause was a failure to anticipate unexpected behavior from a stopped or slow-moving robot in a zone or intersection.

7 Discussion

Overall, we were pleased with the performance of our vehicle through the NQE and UCE competitions. By creating a general-purpose autonomous driving system, rather than a system tuned to the specific test cases posed by DARPA, our team made substantial progress towards solving some of the underlying problems in autonomous urban driving. For example, in the NQE and UCE, there were a lot of traffic and intersection scenarios we had never previously tested, but the software was able to handle these situations with little or no tweaking.

Our investment in creating a powerful new software architecture for this project paid off in innumerable ways. The software developers devoted a significant amount of time to implementing a generic, robust software infrastructure for logging, playback, single-vehicle simulation, and visualization. Such an investment of energy would be hard to justify to achieve a single product such as the development of a lane tracker or an obstacle detector. However, the development and roll-out of these innovations to support the whole team produced a more stable code base, and enabled shared support of modules between developers as well as quick and effective debugging.

7.1 Perception-Driven Approach

For the final race, we added approximately 100 waypoints such that our interpolation of the RNDF waypoints more closely matched the aerial imagery provided by DARPA. Our system was designed to handle the original race description of

perceiving and navigating a road network with a sparse description, and Talos demonstrated its ability to do this by completing the NQE without a densified RNDF. When it became apparent that this capability was not going to be tested in the UCE, we added waypoints to improve our competitive chances. Nonetheless, during the UCE, Talos still gave precedence to perception-based lane estimates over GPS and RNDF-derived lanes, in accordance with our overall design strategy.

7.2 Slow-Moving Vehicles

Another key lesson learned was the difficulty of dealing with slow-moving objects. We attempted to avoid the error-prone process of explicitly classifying obstacles as vehicles. Instead, our software handled the general classes of static obstacles and moving obstacles. While this strategy worked well during the NQE, in the race, the collisions or near-misses involving Talos often came about due to the difficulty in handling changing traffic vehicle behavior, or slow-moving traffic vehicles. Better handling of slow-moving vehicles, for example through fusion of vision and lidar cues to explicitly recognize vehicles versus other types of obstacles, are avenues for future research.

7.3 Improved Simulation

Further investment in simulation tools for complex multi-robot interactions is warranted. For this project, we developed a useful simulation for a *single* robotic vehicle in a complex environment (including traffic vehicles following pre-defined trajectories). We discussed the possibility of developing a more complex simulation that would enable us to test robot-against-robot (i.e. running our system "against itself"), but decided against this endeavor due to time constraints. In hindsight, this capability would have been quite useful.

While the vehicle generally operated in the vicinity of human-driven traffic without incident, problems were encountered when interacting with other autonomous vehicles at slow speed. The nature of these interactions likely arose due to some implicit assumptions of our algorithms that were put in place to address the DARPA rules. These situations might have been detected from simulation of multiple autonomous vehicles running missions against each other on the same course.

7.4 Verification of Failsafe Approaches

A key capability for long-term autonomous operation was the creation of comprehensive failsafe modes. The judicious use of failsafe timers enabled the system to drop constraints and to free itself in difficult situations, such as when perceptual estimates of the lane boundaries did not match reality. In any complex system of this type, the assumptions of the designers will always be violated by unpredictable situations. The development and verification of more principled and robust approaches to recovering from mistakes is an important issue for robotics research.

8 Release of Logs, Visualization and Software

In the interests of building collaboration and a stronger research base in the field, Team MIT has made its work available to the research community. The complete Talos UCE race logs, the viewer software and video highlights from the race (made from the logs) are publicly available at:

`http://grandchallenge.mit.edu/public/`

In addition, several core components developed for the Urban Challenge have been released as open source software projects. The Lightweight Communications and Marshalling (LCM) software library and the libcam image processing toolchain have been released as open source projects:

`http://lcm.googlecode.com/`
`http://libcam.googlecode.com/`

These software components were described in Section 3.3.

9 Conclusion

This paper describes the developed software architecture for a perception-driven autonomous urban vehicle designed to compete in the 2007 DARPA Urban Challenge. The system used a comprehensive perception system feeding into a powerful kino-dynamic motion planning algorithm to complete all autonomous maneuvers. This unified approach has been "race proven", completing the Urban Challenge mission and driving autonomously for approximately 55 miles in under 6 hours. A key novel aspect of our system, in comparison to many other teams, is that autonomous decisions were made based on locally sensed perceptual data in preference to pre-specified map data wherever possible. Our system was designed to handle the original race description of perceiving and navigating a road network with a sparse description. Another innovative aspect of our approach is the use of a powerful and general-purpose RRT-based planning and control algorithm, achieving the requirements of driving in lanes, three-point turns, parking, and maneuvering through obstacle fields with a single, unified approach. Our system was realized through the creation of a powerful new suite of software tools for autonomous vehicle research, tools which have been made available to the research community. Team MIT's innovations provide a strong platform for future research in autonomous driving in GPS-denied and highly dynamic environments with poor *a priori* information.

Acknowledgments

Sponsored by Defense Advanced Research Projects Agency, Program: Urban Challenge, ARPA Order No. W369/00, Program Code: DIRO. Issued by DARPA/CMO under Contract No. HR0011-06-C-0149.

Our team also gratefully acknowledges the sponsorship of: MIT School of Engineering, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL),

MIT Department of Aeronautics and Astronautics, MIT Department of Electrical Engineering and Computer Science, MIT Department of Mechanical Engineering, The C. S. Draper Laboratory, Franklin W. Olin College of Engineering, The Ford-MIT Alliance, Land Rover, Quanta Computer Inc., BAE Systems, MIT Lincoln Laboratory, MIT Information Services and Technology, South Shore Tri-Town Development Corporation and Australia National University. Additional support has been provided in the form of in-kind donations and substantial discounts on equipment purchases from a variety of companies, including Nokia, Mobileye, Delphi, Applanix, Drew Technologies, and Advanced Circuits.

References

Atreya et al., 2006. Atreya, A.R., Cattle, B.C., Collins, B.M., Essenburg, B., Franken, G.H., Saxe, A.M., Schiffres, S.N., Kornhauser, A.L.: Prospect Eleven: Princeton university's entry in the 2005 DARPA Grand Challenge. J. Robot. Syst. 23(9), 745–753 (2006)

Bertozzi, 1998. Bertozzi, M., Broggi, A.: Gold: a parallel real-time stereo vision system for generic obstacle and lane detection. IEEE Transactions on Image Processing 7(1), 62–81 (1998)

Blom and Bar-Shalom, 1988. Blom, H., Bar-Shalom, Y.: The interacting multiple model algorithm for systems with Markovian switching coefficients. IEEE Transactions on Automatic Control 33(8), 780–783 (1988)

Braid et al, 2006. Braid, D., Broggi, A., Schmiedel, G.: The TerraMax autonomous vehicle. J. Robot. Syst. 23(9), 693–708 (2006)

Chen 2006. Chen, Q., Ozguner, U.: Intelligent off-road navigation algorithms and strategies of team desert buckeyes in the DARPA Grand Challenge 2005. J. Robot. Syst. 23(9), 729–743 (2006)

Cremean etal., 2006. Cremean, L.B., Foote, T.B., Gillula, J.H., Hines, G.H., Kogan, D., Kriechbaum, K.L., Lamb, J.C., Leibs, J., Lindzey, L., Rasmussen, C.E., Stewart, A.D., Burdick, J.W., Murray, R.M.: Alice: An information-rich autonomous vehicle for high-speed desert navigation. J. Robot. Syst. 23(9), 777–810 (2006)

DARPA, 2007. DARPA, Darpa urban challenge rules (2007), `http://www.darpa.mil/GRANDCHALLENGE/rules.asp`

Fischler and Bolles, 1981. Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24(6), 381–395 (1981)

Frazzoli, 2001. Frazzoli: Robust Hybrid Control for Autonomous Vehicle Motion Planning. PhD thesis, MIT (2001)

Grabowski et al., 2006. Grabowski, R., Weatherly, R., Bolling, R., Seidel, D., Shadid, M., Jones, A.: MITRE meteor: An off-road autonomous vehicle for DARPA's grand challenge. J. Robot. Syst. 23(9), 811–835 (2006)

Hart and Raphael, 1968. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. 4(2), 100–107 (1968)

Hartley and Zisserman, 2001. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge (2001)

Iagnemma:JFR:2006. Iagnemma, K., Buehler, M.: Special issue on the DARPA Grand Challenge: Editorial. Journal of Field Robotics 23(9), 655–656 (2006)

Kelly and Stentz, 1997. Kelly, A., Stentz, A.: An approach to rough terrain autonomous mobility. In: International Conference on Mobile Planetary Robots (1997)

LaValle and Kuffner, 2001. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. International Journal of Robotics Research 20(5), 378–400 (2001)

Leedy etal., 2006. Leedy, B.M., Putney, J.S., Bauman, C., Cacciola, S., Webster, J.M., Reinholtz, C.F.: Virginia Tech's twin contenders: A comparative study of reactive and deliberative navigation. J. Robot. Syst. 23(9), 709–727 (2006)

Mason etal., 2006. Mason, R., Radford, J., Kumar, D., Walters, R., Fulkerson, B., Jones, E., Caldwell, D., Meltzer, J., Alon, Y., Shashua, A., Hattori, H., Takeda, N., Frazzoli, E., Soatto, S.: The Golem Group / UCLA autonomous ground vehicle in the DARPA Grand Challenge. Journal of Field Robotics 23(8), 527–553 (2006)

Mertz et al., 2005. Mertz, C., Duggins, D., Gowdy, J., Kozar, J., MacLachlan, R., Steinfeld, A., Suppe, A., Thorpe, C., Wang, C.: Collision Warning and Sensor Data Processing in Urban Areas. In: Proc. 5th international conference on ITS telecommunications, pp. 73–78 (2005)

Newman, 2003. Newman, P.M.: MOOS - A Mission Oriented Operating Suite. Technical Report OE2003- 07, MIT Department of Ocean Engineering (2003)

Park et al., 2007. Park, S., Deyst, J., How, J.P.: Performance and Lyapunov stability of a nonlinear path following guidance method. Journal of Guidance, Control, and Dynamics (6), 1718–1728 (2007)

Rivest and Leiserson, 1990. Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill, Inc., New York (1990)

Schouwenaars et al., 2004. Schouwenaars, T., How, J., Feron, E.: Receding Horizon Path Planning with Implicit Safety Guarantees. In: Proceedings of the IEEE American Control Conference. IEEE, Los Alamitos (2004)

Stanford Racing Team, 2007. Stanford Racing Team: Stanford's robotic vehicle Junior: Interim report (2007), `http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Stanford.pdf`

Stein et al., 2000. Stein, G., Mano, O., Shashua, A.: A robust method for computing vehicle ego-motion. In: Proc. IEEE Intelligent Vehicles Symposium, pp. 362–368 (2000)

Stein et al., 2003. Stein, G., Mano, O., Shashua, A., Ltd, M., Jerusalem, I.: Vision-based ACC with a single camera: bounds on range and range rate accuracy. In: Proc. IEEE Intelligent Vehicles Symposium, pp. 120–125 (2003)

Tartan Racing Team, 2007. Tartan Racing, Tartan racing: A multi-modal approach to the DARPA urban challenge (2007), `http://www.darpa.mil/GRANDCHALLENGE/TechPapers/TartanRacing.pdf`

Thorpe et al., 2005. Thorpe, C., Carlson, J., Duggins, D., Gowdy, J., MacLachlan, R., Mertz, C., Suppe, A., Wang, B., Pittsburgh, P.: Safe Robot Driving in Cluttered Environments. In: Robotics Research: The Eleventh International Symposium (2005)

Thrun et al., 2006. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley: The robot that won the DARPA Grand Challenge. J. Robot. Syst. 23(9) (2006)

Trepagnier et al., 2006. Trepagnier, P., Nagel, J., Kinney, P., Koutsourgeras, C., Dooner, M.: KAT-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain. Journal of Field Robotics: Special Issue on the DARPA Grand Challenge 23, 467–508 (2006)

Urmson et al., 2006. Urmson, C., Anhalt, J., Bartz, D., Clark, M., Galatali, T., Gutierrez, A., Harbaugh, S., Johnston, J., Kato, H., Koon, P., Messner, W., Miller, N., Mosher, A., Peterson, K., Ragusa, C., Ray, D., Smith, B., Snider, J., Spiker, S., Struble, J., Ziglar, J., Whittaker, W.: A robust approach to high-speed navigation for unrehearsed desert terrain. Journal of Field Robotics: Special Issue on the DARPA Grand Challenge 23, 467–508 (2006)

USDOT Federal Highway Administration, Office of Information Management, 2005. USDOT Federal Highway Administration, Office of Information Management, Highway Statistics 2005. U.S. Government Printing Office, Washington, D. C (2005)

Wang, 2004. Wang, C.-C.: Simultaneous Localization, Mapping and Moving Object Tracking. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (2004)

Little Ben: The Ben Franklin Racing Team's Entry in the 2007 DARPA Urban Challenge

Jon Bohren[1], Tully Foote[1], Jim Keller[1], Alex Kushleyev[1], Daniel Lee[1,⋆], Alex Stewart[1], Paul Vernaza[1], Jason Derenick[2], John Spletzer[2], and Brian Satterfield[3]

[1] University of Pennsylvania
Philadelphia, PA 19104
ddlee@seas.upenn.edu
[2] Computer Science and Engineering
Lehigh University
Bethlehem, PA 18015
[3] Lockheed Martin Advanced Technology Laboratories
3 Executive Campus, suite 600
Cherry Hill, NJ 08002

Abstract. This paper describes "Little Ben," an autonomous ground vehicle constructed by the Ben Franklin Racing Team for the 2007 DARPA Urban Challenge in under a year and for less than $250,000. The sensing, planning, navigation, and actuation systems for Little Ben were carefully designed to meet the performance demands required of an autonomous vehicle traveling in an uncertain urban environment. We incorporated an array of GPS/INS, LIDAR's, and stereo cameras to provide timely information about the surrounding environment at the appropriate ranges. This sensor information was integrated into a dynamic map that could robustly handle GPS dropouts and errors. Our planning algorithms consisted of a high-level mission planner that used information from the provided RNDF and MDF to select routes, while the lower level planner used the latest dynamic map information to optimize a feasible trajectory to the next waypoint. The vehicle was actuated by a cost-based controller that efficiently handled steering, throttle, and braking maneuvers in both forward and reverse directions. Our software modules were integrated within a hierarchical architecture that allowed rapid development and testing of the system performance. The resulting vehicle was one of six to successfully finish the Urban Challenge.

1 Introduction

The goal of the 2007 DARPA Urban Challenge was to build an autonomous ground vehicle that could execute a simulated military supply mission safely and effectively in a mock urban area. Compared with previous DARPA Grand Challenges, this particular challenge necessitated that robot vehicles perform autonomous maneuvers safely in traffic (DARPA, 2007). To address this challenge, the Ben Franklin Racing Team was formed by students and faculty at

⋆ Corresponding author.

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 231–255.
springerlink.com

Fig. 1. Little Ben is a Toyota Prius hybrid vehicle modified for drive-by-wire operation with an onboard array of sensors and computers.

the University of Pennsylvania, Lehigh University, and engineers at Lockheed Martin Advanced Technology Laboratory. In under a year and with a limited budget, the Ben Franklin Racing Team was able to construct "Little Ben," a drive-by-wire Toyota Prius with an array of onboard sensors and computers shown in Figure 1. The following sections detail our team's approach in designing and constructing hardware and software algorithms for our entry in the Urban Challenge.

1.1 Design Considerations

The Urban Challenge presented unique challenges to autonomous sensing, navigation, and control. Some of the scenarios that our vehicle needed to be able to handle included the following:

- Maintain appropriate safety margins at all times.
- Accurately follow a lane within prescribed lane boundaries.
- Detect and avoid moving traffic.
- Stop and drive into a new lane in the presence of other vehicles.
- Park in constrained spaces in dynamic environments.

These situations required that obstacles and lane markings were detected at a distance, and that the vehicle reacted quickly and appropriately while following the local traffic laws and conventions. An overarching requirement was that a successful system adhere to a stringent set of real-time processing constraints in its detection and reaction to its environment. This was mainly reflected in the system reaction time, as governed by the processing sample rate. Low sample rates increase the distance at which obstacles and other traffic vehicles must be detected for safe operation. Conversely, high sample

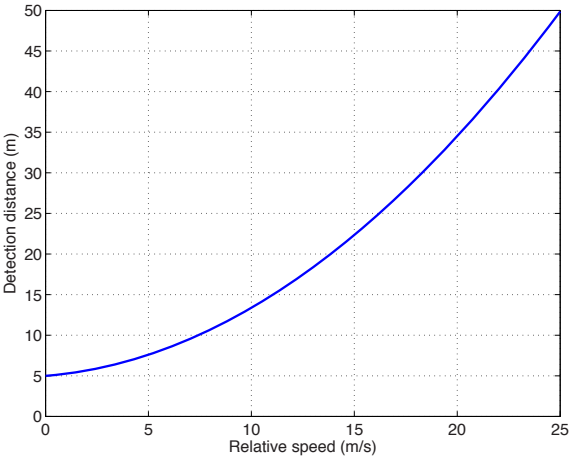


Fig. 2. Required detection distance at various speeds taking into account worst-case latencies in system processing time.

rates are attainable only by using overly simplified sensing and control algorithms.

The design of our vehicle's hardware and software systems was predicated on achieving a reaction time that ensured safe operation of driving maneuvers at the mandated upper speed limit of 30 mph (13.4 m/s). As an example of our design methodology, Figure 2 shows our calculation of the required detection distance of another vehicle in order for our vehicle to properly react and stop at various relative speeds up to 60 mph (26.8 m/s). In our calculations, we required that at least one vehicle length of separation was maintained at the end of the maneuver. We have also identified the maximum braking acceleration that can be introduced in this speed range without triggering the anti-lock brake system (ABS). Therefore, the ABS reaction dynamics were reserved as an additional safety margin when dry pavement conditions were not present.

We evaluated a range of possible system sample rates, and selected 10 Hz as the desired system processing rate. Our calculations in Figure 2 took into account a two sample period delay (200 ms) as the worst case scenario for detection and reaction; the first sample period could elapse just before an obstacle crossed the sensor detection threshold, and the second sample period was assumed to be used for the necessary computational processing.

The hardware and software systems were selected to meet the desired detection distance and processing time objectives. Sensors and their respective mounting positions were chosen to maximize their long range detection characteristics. Drive-by-wire actuation and computer hardware systems were selected to minimize processing latencies. Similarly, our software modules were also optimized to maximize detection distance and minimize processing delays. This combination of hardware sensing systems with efficient, reactive

software modules allowed Little Ben to achieve the requisite safety margins for driving in urban traffic situations.

2 Vehicle Platform

Little Ben was built from a 2006 Toyota Prius hybrid vehicle with modified controls to allow drive-by-wire as well as manual operation. Since the Urban Challenge took place in a *mostly* urban setting, there was no need for a large off-road vehicle. The Prius' compact size made many driving maneuvers easier to accomplish compared to other larger vehicles, and also proved to be very stable, reliable, and easy to work with.

Unlike most standard automobiles, Little Ben did not have an alternator. Instead it used power provided by the built-in 200 V hybrid battery via a DC-DC converter to power all standard 12 V vehicle components as well as the additional hardware that we installed. As shown in Figure 3, the total peak power consumption of the additional hardware systems was less than 700 W peak - well below the maximum 1 kW power output of the stock DC-DC converter. Thus, Little Ben did not require any specialized alternators or additional generators or cooling hardware. This overall power and fuel efficiency enabled Little Ben to finish the 57 miles of the Urban Challenge using only about one gallon of gasoline.

2.1 Drive-by-Wire Actuation

As depicted in Figure 4, the drive-by-wire vehicle actuation was performed by Electronic Mobility Controls (EMC) of Baton Rouge, Louisiana. This conversion included DC servomotors to actuate the steering wheel and gas/brake

Device	Peak Power Consumption (Watts)
EMC (Drive-by-wire)	150
3 SICK LMS291	60
2 SICK LDLRS	60
1 Velodyne	60
7 Mac Minis	250
3 Hokuyo URG-04LX	15
1 OxTS Pose System	10
2 Serial Device Servers	5
Ethernet Switches, Router	5
Total Peak Power	~650
Actual Steady-state	~450

Fig. 3. Power consumption of sensors, computers, and vehicle actuation on Little Ben.

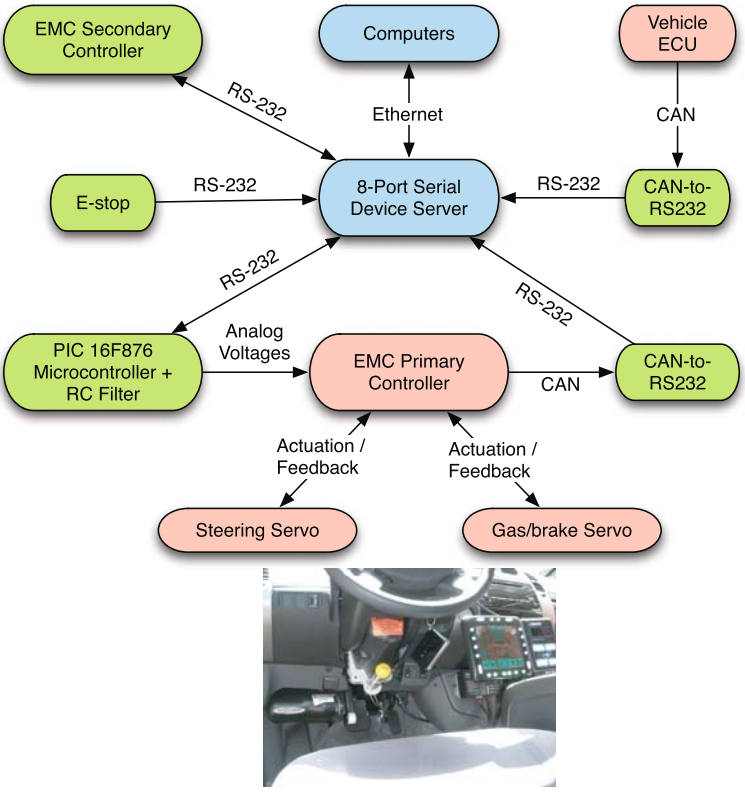


Fig. 4. Components that interface to the drive-by-wire system.

pedals, along with triple redundant motor controllers to ensure safe operation. Two analog DC voltage inputs were provided by EMC to control the steering wheel and gas/brake pedal position. In order to transmit the digital control signal from the computers to the drive-by-wire system, we implemented a very simple digital-to-analog converter using a cheap digital PIC microcontroller with a RC-filtered PWM output. Given that the drive-by-wire analog signals were sampled at 100 Hz, the RC time constant of the filter was chosen to be approximately 10 ms. This ensured that the full actuation bandwidth was preserved while smoothing any electrical noise interference in the vehicle. The PWM frequency of the microcontroller was set to 20 kHz with 8-bit resolution, which was sufficient for smooth and accurate control of the actuators.

Other vehicle controls such as transmission shifting, turn signals, and parking brake were interfaced via a single RS-232 connection to EMC's secondary controller unit. Additionally, we installed a CAN bus interface to the Toyota on board diagnostic (OBD) connector in order to verify vehicle state information directly from the car's electronic control unit (ECU). The CAN

interface provided accurate brake pedal position at 100 Hz, steering encoder feedback at 70 Hz, and other vehicle state information such as transmission shift setting at slightly lower rates.

2.2 Emergency Stop

Since safety is a top priority with autonomous vehicles, we took major steps toward minimizing the risk of injury or damage due to undesired behavior of the vehicle. The emergency stop system was designed to make human intervention safe, quick, and reliable. In order to achieve fail-safe operation, redundancy was incorporated on multiple levels using watchdog timers and heartbeat monitors as shown in Figure 5.

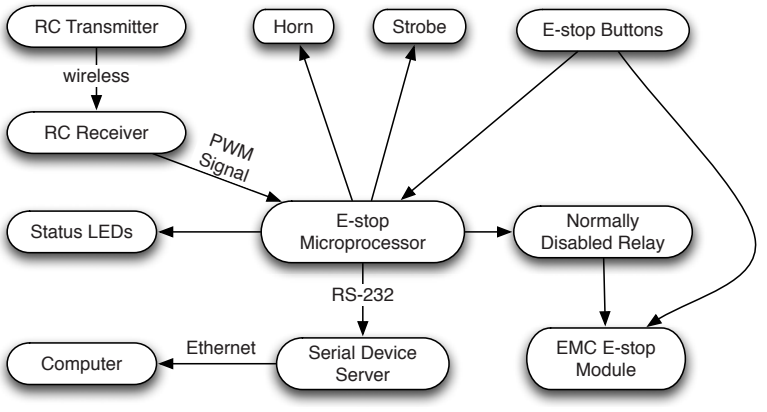


Fig. 5. Block diagram depicting the emergency stop and safety systems incorporated into the vehicle.

When the "pause" mode was activated, either by a human operator or when the radio-controlled transmitter was out of range, the throttle commands from the computers were automatically overridden and the brake was applied at near maximimum braking acceleration to ensure smooth stopping within the allowed distance. In this mode, the computers were unable to drive the vehicle unless the "run" command was explicitly given. After the "run" command was given, the audible and visual strobe warning devices were activated, and control was returned to the computer systems after a five second delay.

Our "disable" mode was an extension of the "pause" mode. In addition to braking the vehicle and disabling computer control of the throttle, the E-stop processor verified the vehicle speed was zero on the Toyota CAN bus, and set the transmission to *park*. All the Toyota Prius systems were then powered down. In this state, the vehicle would need to be manually restarted in order to reactivate autonomous control. The vehicle could easily be disabled via

the dedicated remote control or the manual E-stop buttons located on either side of the car.

To achieve high reliability, the most crucial components of the safety system were implemented using simple PIC microcontrollers and fail-safe mechanical relays. These were powered using the backup battery system of the EMC drive-by-wire system, so even without vehicle power or computers, the car was guaranteed to respond properly to "pause" and "disable" commands.

2.3 Roof Rack

Due to constraints on our local storage facilities, our primary sensor rack was designed such that it could be quickly mounted and unmounted as a single structure without having to re-calibrate the sensors. The stock beams from a Yakima roof rack were replaced with aluminum pipes onto which an 80/20 aluminum structure was rigidly fixed. Once locked into place, the sensor rack was connected to the vehicle power and computing systems through a single umbilical connector.

In order to maximize the detection range of our sensors, the rack shown in Figure 6 was custom designed to allow optimal viewing angles for as many of these sensors as possible. In particular, we designed the rack to accomodate a Velodyne HD LIDAR to give the omnidirectional sensor a fully unoccluded 360 degree azimuthal view. By mounting the Velodyne 8" above the rest of the sensor rack, we took advantage of the full complement of elevation angles in the sensor to provide a sensing range from 4 to 60 meters around the vehicle. Its lateral left-of-center position was also optimized for navigating around obstacles on American roads.

The rack also integrated a set of forward and rear facing SICK LMS-291 S14 LIDAR sensors. These 90 degree field of view sensors were tilted downward in order to intersect the ground at approximately 6–7 meters ahead and behind the vehicle. In these positions, the SICK LIDAR's were well within their range limitations, and could provide for both ground plane, obstacle and lane marking detection. These sensors were also arranged so that they did not

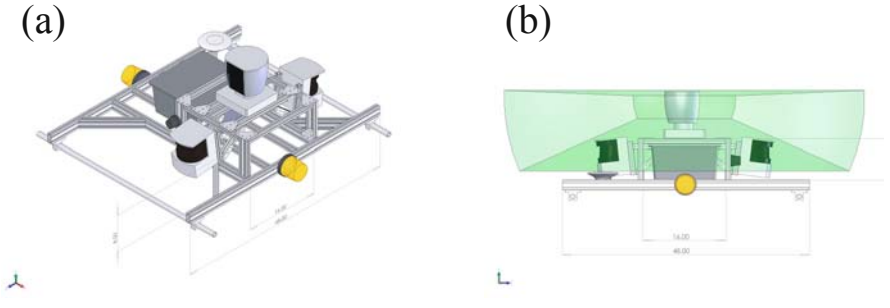


Fig. 6. Roof sensor rack designed to provide optimal viewing angles.

occlude the Velodyne's field of view, and provided a complementary stream of range data.

The rack also contained the warning siren, strobe lights, and mounting points for the GPS antennas. Additionally, it provided space for a weatherproof electronics enclosure. This contained and protected the power distribution block and connectors for sensors mounted on other parts of the vehicle.

2.4 Hood Sensors

Little Ben also integrated several sensors that were mounted on its hood. At the front center of the hood was a vertical scanning SICK LMS-291 LIDAR, as well as a high-resolution Point Grey Bumblebee stereo camera as shown in Figure 7. The 1024×768 resolution color camera had a horizontal 50 degree field of view, with a framerate of 15 Hz. To minimize the potential for image blooming caused by sunlight, the camera was pitched down 15° to minimize the field of view over the horizon. A visor was also integrated as a further level of protection.

Fig. 7. Sensors mounted on the hood of Little Ben.

In addition, two SICK LD-LRS LIDAR scanners were mounted parallel to the road surface at the front left and front right corners of the hood. These scanners provided overlapping 270 degree fields of coverage, and were used to detect obstacles and track moving vehicles in front and at the sides of Little Ben. The LD-LRS sensors employed a scanning frequency of 10 Hz, reporting laser returns at 0.5 degree increments. Due to their vulnerable position and possible misalignment in the event of a crash, a simple cardboard fiducial marker was attached to the hood and used to automatically verify correct operation of these sensors.

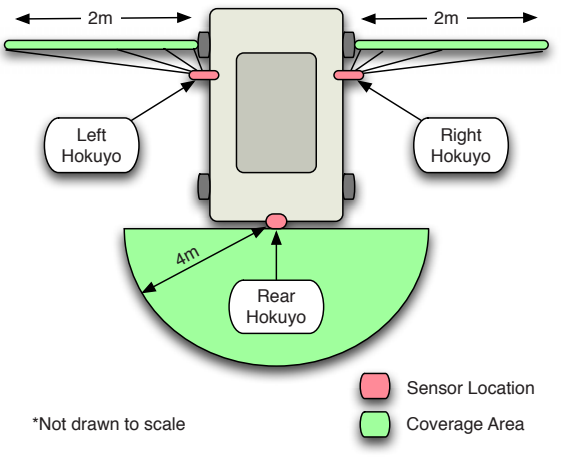


Fig. 8. Additional Hokuyo scanners used to eliminate blind spots at short range.

2.5 Other Sensors

Three compact Hokuyo URG-04LX LIDAR scanners were also used to cover blind spots in sensor coverage at short range around the vehicle as shown in Figure 8. Although these sensors were only rated for indoor use, through experimentation we found that they could be used in outdoor conditions as long as they were properly shielded from water and from light in the back. Two Hokuyo scanners were mounted underneath the side mirrors for detecting obstacles such as curbs at the sides of the front wheels, as well as nearby lane markings on the ground. The third sensor was mounted slightly above the rear bumper, and allowed for accurate maneuvering between tightly spaced obstacles while in reverse.

3 Software Architecture

As depicted in Figure 9, the software architecture was divided hierarchically into a series of modules, connected via interprocess communication messages. At the lowest level was the driving module which was responsible for interfacing to the vehicle controller hardware and verifying correct operation of the steering, throttle, braking, and transmission. Also present at this low level was the pose software module which integrated readings from the GPS and inertial navigation system to provide the latest pose information at 100 Hz. These two hardware interface modules could be readily replaced by a simulation module which allowed us to rapidly test the software without requiring the processes to be physically connected to the vehicle systems.

At the highest level, the Mission planning module read the appropriate RNDF and MDF files to determine the optimal sequencing of waypoints needed to complete the mission objectives. Next were the sensor modules

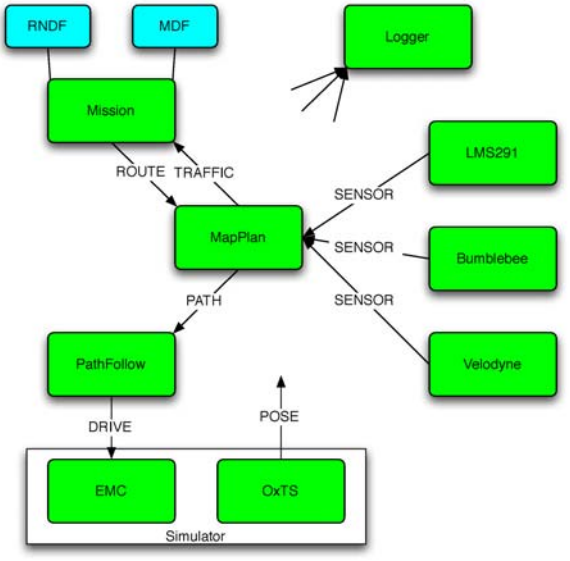


Fig. 9. Software architecture showing system modules and corresponding interprocess communication messages.

which gathered data from all the LIDARs and the stereo camera to provide probabilistic real-time estimates of the terrain, road markings, and static and dynamic obstacles. These modules consolidated the large amount of sensor data into a compact representation in the vehicle's local reference frame before sending this information onto the MapPlan process.

The MapPlan process was then responsible for integrating all the sensor information into a probabilistic dynamic map, and then computing the appropriate vehicle path to reach the next desired waypoint as determined by the high-level Mission planner. It also checked to ensure that this path avoided all known obstacles, while obeying vehicle dynamic constraints as well as local traffic rules. The PathFollow module took the desired vehicle path from the MapPlan process and generated the optimal steering, throttle, and braking commands needed by the low-level driving module.

All the processes communicated with each other via well-defined message formats sent through the Spread messaging toolkit (Amir et al., 2004). This open-source messaging system provided message reliability in the presence of machine failures and process crashes, while maintaining low latencies across the network. It also enabled convenient logging of these messages with appropriate timestamps. These logs allowed us to rapidly identify and debug bad processes, as well as replay logged messages for diagnostic purposes.

These modules were written mainly in Matlab with some ancillary C++ routines. The use of high-level Matlab enabled the software system to be written in less than 5000 lines of code. We also implemented a development

environment that incorporated Subversion for source code tracking, Bugzilla for assigning tasks, and a Wiki for writing documentation. All the documentation was readily accessible to the whole team with convenient search functionality to allow easy collaboration. During field testing, local copies of the Bugzilla and source code repository were stored within the vehicle to allow us to make offline changes that were merged with our central servers after testing. With these tools, rapid prototyping and development was accomplished by the team both in the laboratory and in the field.

3.1 Computing System

All the software processes were run on a small computer cluster, consisting of seven Mac Minis with Core 2 Duo processors running Ubuntu Linux. The computers were interconnected through a gigabit ethernet network in the vehicle trunk as shown in Figure 10. Serial connections to the vehicle's low-level hardware and sensors were also provided over the ethernet network via Comtrol serial device servers. In the event of a computer failure, our system automatically switched the affected processes over to a redundant computing node without having to manually reconfigure any connections. Special monitoring software (monit) was used to constantly check the status of all the computers and processes to detect software crashes and other possible failures.

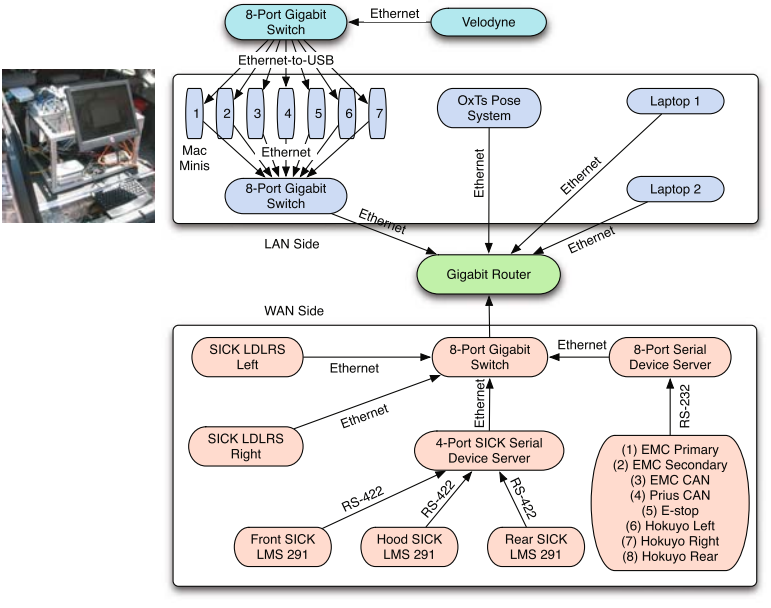


Fig. 10. Computing and networking systems on board Little Ben.

To prevent the various data streams from interfering with one another, Little Ben contained three separate subnetworks isolated using hardware routers and switches. The first subnet was used for normal interprocess communications between the Mac Mini's. The second subnet was used to isolate the various LIDAR sensors–it was found that some of the SICK sensors contained buggy network protocol implementations and would lock up in the presence of extraneous ethernet traffic. Finally, the third subnetwork was used to isolate the large amounts of data broadcast by the Velodyne LIDAR sensor (approximately 3 MB per second); only those computers processing this data stream would subscribe to this subnet.

4 Perception

Little Ben's perception system was responsible for providing information about the locations of static obstacles, traversable ground, moving vehicles, and lane markings on the road. This processing was performed in a highly redundant manner by the various sensors on the vehicle: Velodyne LIDAR, SICK LIDAR's, Hokuyo LIDAR's, and Bumblebee stereo camera.

4.1 Velodyne Processing

The Velodyne HDL-64E LIDAR was Little Ben's primary medium to long-range sensor. It was used for geometric obstacle/ground classification, road marking extraction, as well as dynamic obstacle velocity tracking. The Velodyne houses sixty-four 905 nm lasers and can spin between 5 and 15 Hz, yielding a field of view of 360° azimuth, and -24° to +2° zenith. We configured the sensor to spin at 10 Hz in order to acquire a high point density and capture frames at the same rate as our control system. A sample scan is shown in Figure 11.

Although we were supplied with the factory horizontal and vertical correction factors, we found that the individual lasers required an additional distance offset that we calibrated using comparisons to the readings from the SICK LMS-291 sensors. Even with this extra calibration, we found that our particular Velodyne sensor would sometimes report laser ranges with uncertainties on the order of 30 cm, much larger than the stated 5 cm accuracy. Because of this large uncertainty, it was necessary to process the Velodyne data as 64 independent scans, rather than aggregating returns between different lasers.

Some of the individual lasers would also sporadically return large noisy outliers. Because of this, the range and reflectivity values from each laser were carefully monitored online and rejected if any inconsistent outliers were detected. Classifications of ground versus obstacle were also never based upon a single return, but were based upon the statistics of a consecutive set of 4–5 points.

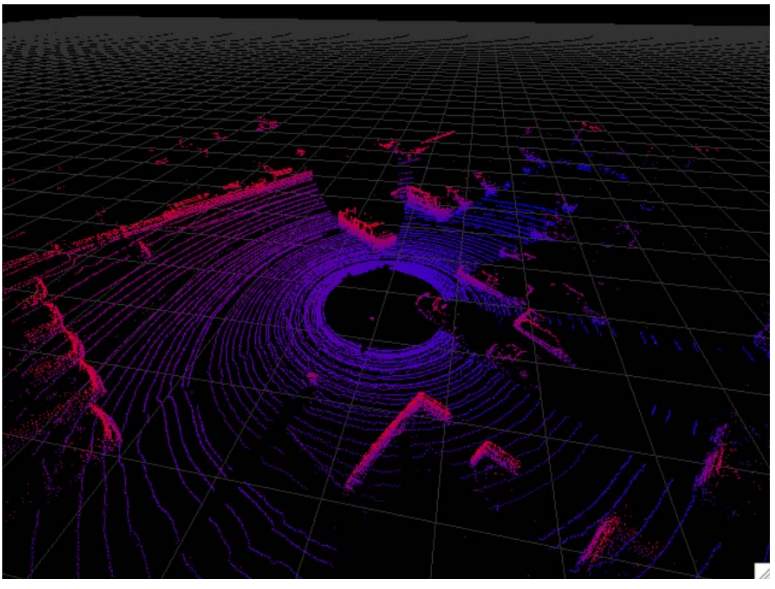


Fig. 11. 3D point cloud from a single Velodyne scan classified as ground and obstacle.

In this manner, we were able to classify reflective obstacles such as other vehicles out to 60 meters, and detect ground points out to 30 meters under good conditions, depending upon the reflectivity of the ground. The reflectivity data were also used to detect lane markings during the Urban Challenge within a range of 15 meters.

4.2 LIDAR Ground/Obstacle Detection

The range scans from the downward angled LIDAR's, SICK LMS-291's and side mounted Hokuyo's, were processed in the following manner. In the first phase of the algorithm, a ground plane was fit in a robust fashion to the observed points. The range values from a scan were first passed through a median filter to remove spurious returns due to airborne dust particles, rain, etc. Then given the observed Cartesian points from the filtered LIDAR scan: (x_i, z_i), we minimized the following objective:

$$\min_{m,b} \sum_i f(z_i - mx_i - b) \tag{1}$$

where f was an error measure that was quadratic near zero, but decreased much more slowly for larger values. This minimization was performed in an incremental fashion using interative least squares (Guitton, 2000).

To improve the robustness of the ground plane fit, we also employed regularization of the ground plane parameters based upon the relative geometry of

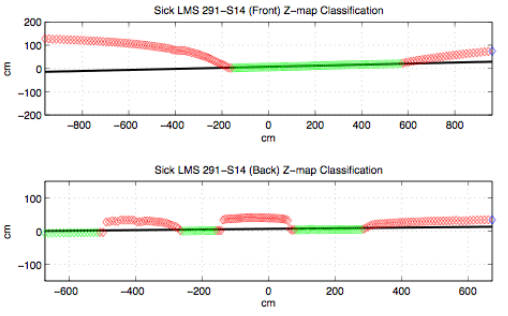


Fig. 12. Robust ground plane extraction and ground/obstacle classification from the front and rear facing SICK LMS-291 sensors.

the vehicle and sensor calibration. This enabled our algorithm to accurately track the ground as shown in Figure 12, even in the presence of significant pitch changes as well as highly linear obstacle features.

Once an accurate ground plane had been determined, it was relatively easy to classify the various observed scan points as obstacle or ground based upon their deviation to the ground plane. Another example of this classification is shown in detecting a nearby curb from a side-mounted Hokuyo scanner as shown in Figure 13.

4.3 LIDAR Lane Marking Detection

In addition to streaming range information, the Velodyne, SICK LMS-291-S14, and Hokuyo LIDAR's also returned corresponding reflectivity values. This information was used for detecting and identifying lane markers in order to compensate for any positional shifts in our pose system.

To detect lane markings from the LIDAR returns, the reflectivity readings of identified ground points were first analyzed. Sections of ground whose

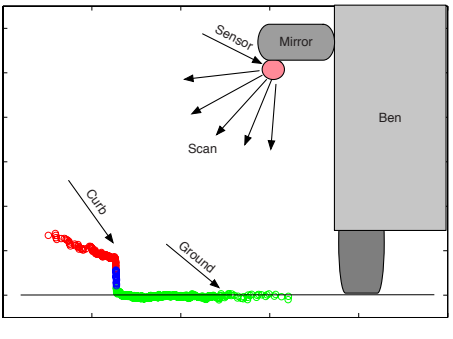


Fig. 13. Curb/ground detection from side facing Hokuyo LIDAR.

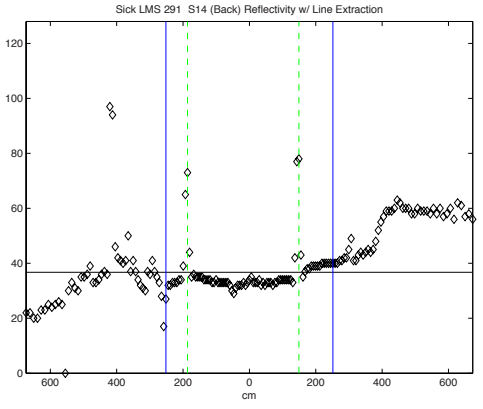


Fig. 14. Lane marking detection and identification from front SICK LMS-291.

reflectivity values were significantly higher than the surrounding road surface were then identified as potential markings. These sections were then checked to see if they corresponded to line widths between 10–15 cm wide. Figure 14 shows an example of lane marking detection and identification from the front facing SICK LMS-291 sensor. The two peaks correspond to the left and right lane markings present in the lane.

4.4 Dynamic Obstacle Tracking

To successfully navigate through dynamic obstacles, obstacle velocities as well as positions needed to be accurately estimated. Budgetary and time constraints precluded us from incorporating any type of RADAR sensors, so Little Ben tracked dynamic obstacles using consecutive LIDAR returns from the Velodyne sensor as well as consecutive returns from the hood-mounted SICK LD-LRS scanners.

First, classified obstacle range returns were grouped into local line features. The line features were then tracked across consecutive scans using a multiple hypothesis Kalman filter. This filter rejected spurious detections based upon prior constraints on the size and velocities of known obstacles.

Figure 15 shows the output of the algorithm tracking two vehicles based upon consecutive range readings from one of the hood-mounted SICK LD-LRS scanner. In this manner, moving obstacles within a range of approximately 60 m could be tracked with an accuracy of about 1 m/s.

4.5 Vision

The Bumblebee stereo vision system was also used to recover road markings at ranges from 4–15 m ahead of the vehicle. Constraining our interest to this region yielded more robust feature segmentation, more reliable stereo

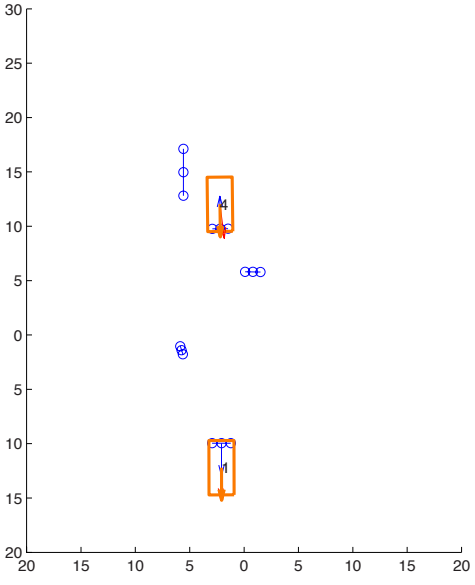


Fig. 15. Vehicle heading and velocity tracking from SICK LD-LRS sensor; Little Ben is located at the origin of the figure

disparity estimates, and allowed a linear model to be used in reconstructing the lane markings.

Images were processed at 512×384 resolution at a frame rate of 15 fps, using approximately 50% of a single core of one of the Mac Mini processors. Figure 16 shows an example of the output of our vision system in detecting and locating lane markings relative to the vehicle.

Images from the stereo camera were first enhanced, and then subtracted from corresponding pixel-shifted locations in the left and right images. The appropriate pixel shifts were determined by calibrating the camera relative to the ground plane. This could also be done adaptively using the observed stereo disparity values. Valid lane markings were then detected using a variety

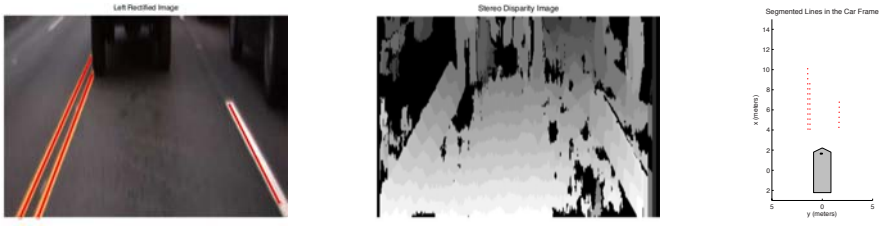


Fig. 16. (Left) Camera image with segmented lanes; (Center) Corresponding disparity image; (Right) Reconstructed lane relative to vehicle.

of filters that test image region candidates based upon width, length, and area constraints. Candidate lines were then projected onto the road surface, and placed into the map relative to the current vehicle location.

5 Mapping

Our previous experience with autonomous outdoor navigation has underscored the need for robust mapping that is consistent with perceptual data as well as prior information about the environment (Vernaza and Lee, 2006). As the vehicle traversed its environment, perceptual data were distilled into local occupancy grid maps (Elfes, 1989). These maps were referenced to the local coordinate system of the vehicle and reflected the state of the world as observed at a specific instant in time.

As information about static obstacles, dynamic obstacles, and lane markings were sent by the perceptual modules, the MapPlan module updated the various ground/obstacle and lane marking likelihoods in a 300×300 m map, roughly centered at the current vehicle location. The current vehicle pose was obtained from an Oxford Technical Solutions RT-3050 unit. The RT-3050 is a self-contained unit which uses a Kalman filter based algorithm to combine inertial sensors, Global Positioning System (GPS) updates with differential

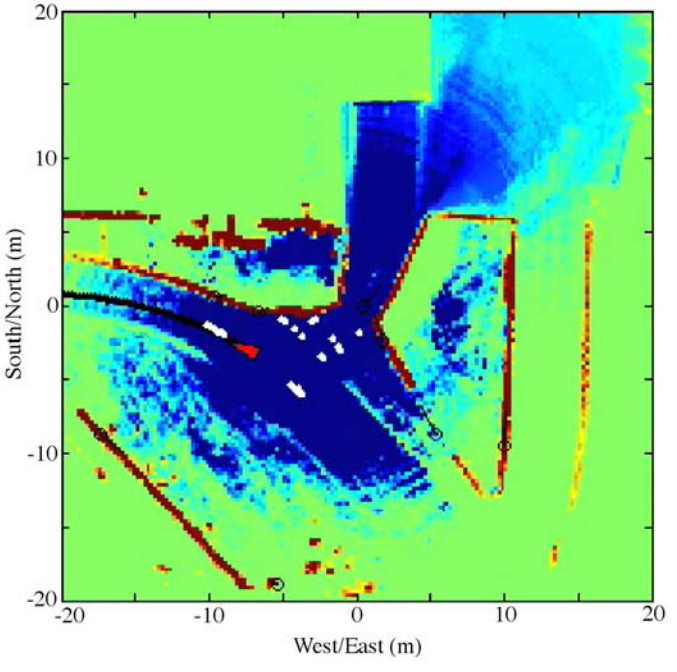


Fig. 17. Probabilitic map with obstacle/road (red/blue) likelihoods along with lane markings (white) generated near the beginning of Urban Challenge course.

corrections from the OmniStar VBS service and vehicle odometry information from the native embedded vehicle systems (Kalman and Bucy, 1961). The RT-3050 was able to provide pose estimates at a high update rate of 100 Hz with a stated accuracy of 0.5 meter. The unit was specifically designed for ground vehicle testing applications, and was capable of providing pose estimates during periods of sustained GPS outages or poor GPS performance due to environmental effects such as multipath reflections.

Given the vehicle pose estimates, the various perceptual measurements were fused into the current map. Figure 17 shows a snapshot of the map shortly after the beginning of the Urban Challenge, when Little Ben entered the two-lane loop. The walls, road surface, as well as road markings can be clearly seen in this map.

6 Planning and Control

6.1 Mission and Path Planning

In our hierarchical software architecture, planning was performed in two stages. At the highest level, the mission planner estimated travel times between waypoints and then computed the optimal sequence of waypoints to traverse in order to minimize the overall mission time. When a particular lane or intersection was blocked, the mission planner recomputed an alternative sequence of waypoints using Dijkstra's algorithm to adaptively respond to traffic conditions (Dijkstra, 1959). Figure 18 illustrates the display from the mission planner as it monitors the progress of the vehicle through a route network.

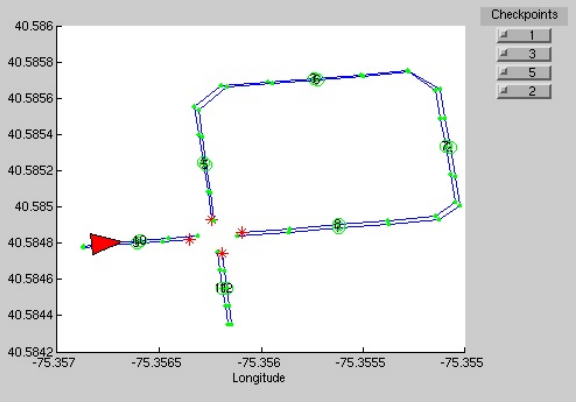


Fig. 18. Mission planner uses information from the RNDF and MDF to plan optimal routes through the traffic network.

The next stage of planning incorporated information from the dynamic map by computing a detailed path to the next waypoint. Depending upon the current sequence and next waypoint type in the RNDF, a specialized local planner that dealt with lane following, U-turns, intersections, and zones separately was selected. The lane following planner optimized a continuous set of lateral offsets from the path given by the RNDF. The U-turn planner monitored the road edges and obstacles while transitioning between forward and reverse driving modes. On the other hand, the intersection planner first monitored the waiting time and other vehicle positions while computing the optimal path through the intersection box. Finally, the zone parking planner used a fast non-holonomic path planner to find an optimal path to the next waypoint in the zone. Each of these planners computed the desired geometric path using the current map costs and a maximum safe driving speed by computed time to possible collisions from the tracked dynamic obstacles.

6.2 Path Following

The path follower module was responsible for calculating the vehicle steering and throttle-brake actuation commands required to follow the desired trajectory as accurately as possible. The trajectory specified the desired route as a set of points for which the spatial position and the first and second derivatives were defined.

Previous approaches to steering control for autonomous car-like vehicles have used PID control based methods with error terms that combine both the lateral and heading offsets from the desired trajectory (Coulter, 1992; Thrun et al., 2006; Rasmussen et al., 2006). A weakness of these controllers in this application is that they do not explicitly consider the kinematic constraints of the vehicle when calculating the steering command. These controllers also typically require significant reparameterization in order to operate the vehicle in reverse.

In order to avoid these shortcomings, we developed an alternative approach for steering control which integrated the dynamics of a vehicle model

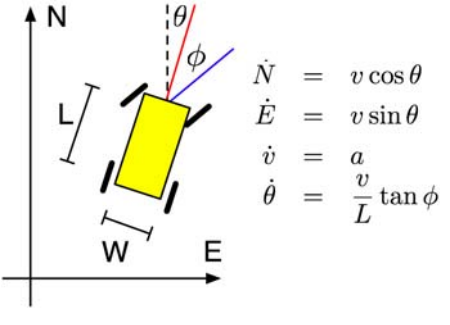


Fig. 19. "Bicycle" model of the car dynamics used for control.

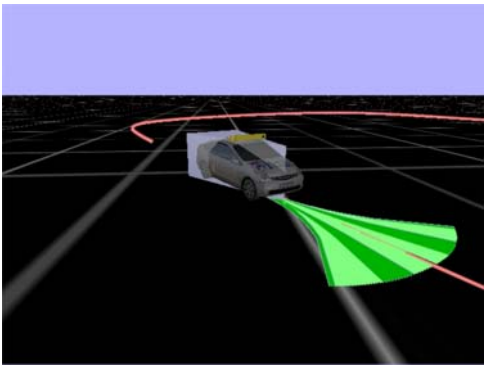


Fig. 20. Estimated future vehicle poses for a set of possible steering commands in a simulated environment.

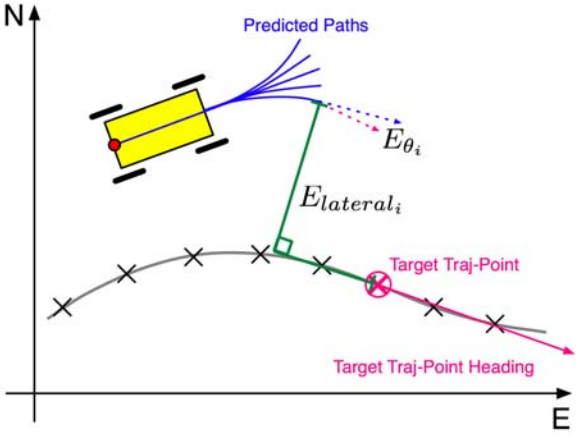


Fig. 21. Graphical representation of the terms included in the controller cost function.

(Figure 19) to predict the resulting change in pose after a short period of time under a set of possible steering commands as shown in Figure 20 (Gillespie, 1992). A cost function is then evaluated for each of the predicted poses and the steering command which minimized this cost function was chosen.

As illustrated in Figure 21, the particular form of the cost function used in our controller was as follows:

$$C(\phi_i) = E_{lateral}^2 + \left[R_\theta \sin(\frac{E_{\theta_i}}{2}) \right]^2 \tag{2}$$

where $E_{lateral}$ and E_{θ_i} are the lateral and heading offsets of the vehicle relative to the target point on the trajectory. Note that there is a length

parameter R_θ in the cost function which was used to scale the heading error relative to the position error. This parameter was adaptively tuned to maximize performance in the different Urban Challenge scenarios.

The advantage of this value-based controller was that it was quite robust to vehicle dynamics, and could be used just as effectively when the vehicle was operated in either reverse or forward. This allowed us to accurately control the vehicle in situations requiring tight navigation such as in lane changing or parking.

The speed of the vehicle was controlled by a proportional-integral (PI) controller after linearization of the throttle and brake dynamics. The controller's error term was the difference between the desired speed set by the path planner, and the current speed as measured by the pose system.

6.3 Overhead Imagery Registration

We used the DARPA-provided overhead imagery to aid in preprocessing the RNDF to yield a more precise description of the roadways. In particular, our primary goal in processing the RNDF was to add a heading to each lane point corresponding to the tangent vector to the road at that point. These tangents were then used to better plan smooth trajectories connecting pairs of waypoints. We did not artifically add extra waypoints to "densify" the given RNDF.

Before this could be accomplished, we needed a good estimate of the mapping from UTM coordinates to image coordinates in the overhead imagery. We found this transformation by fitting an affine model mapping the UTM coordinates of the known corner points to their known pixel locations. To refine this fit, we found additional fiducial points to include in the regression, in the form of the four surveyed points in the team pits whose GPS coordinates were given by DARPA. We deduced the image coordinates of these points by measuring their real-world distances from visible fiducials in the image.

7 NQE Performance

Prior to the NQE, a "Red Team" was formed by engineers from Lockheed Martin Advanced Technology Laboratory who were not involved in the design and development of Little Ben. This team came up with a series of eight weekly tests at three different location sites in the two months before the NQE. These tests stressed various aspects of autonomous driving as specified in the DARPA guidelines. This independent validation gave the team confidence in Little Ben's abilities in unknown environments during the NQE.

On the basis of its performance during the qualifying rounds, Little Ben was chosen as a finalist for the Urban Challenge Final Event (UFE). However, several significant refinements were made to both the vehicle hardware and software prior to the UFE in order to account for deficiencies identified during the NQE test phase.

For example, prior to the NQE the dynamic vehicle tracking described in Section 4.4 was performed entirely by the Velodyne system. However, the merge operation required in NQE Course A exposed significant blind spots of the Velodyne due to occlusions from road signage, as well as the large LED speed monitor used by DARPA test vehicles for speed management. This motivated the additional integration of the SICK LD-LRS hood-mounted units for vehicle tracking. As the Velodyne was already processed as 64 independent LIDAR instances, integrating data from the two additional LD-LRS units was relatively straightforward.

A second potential blind spot was identified during the parking operation required in NQE Course B. As part of this test, the vehicle was required to pull straight into an open parking spot with parked cars on both the left and right sides. Since the front of the parking spot was also blocked by a third car, exiting would require Little Ben to reverse out. However, reversing was aggravated by the placement of a large obstacle parallel to the row of cars. While during the NQE there was sufficient clearance for Little Ben to exit without incident, a more severe test during the UFE might result in losing sight of a low height obstacle. It was this requirement that motivated the integration of the rear-bumper mounted Hokuyo LIDAR described in Section 2.5. In fact, this sensor was integrated the evening before the UFE! While the parking test during the UFE was in fact far simpler than the NQE requirement, the ability to seamlessly integrate another sensor only hours before the final event – and with very limited testing – is itself a testament to the flexibility and modularity of our software architecture.

One final observation from the NQE was regarding our approach to processing the LIDAR data. Since Little Ben relied almost entirely on LIDAR for exteroceptive sensing (no RADARs), we placed significant emphasis upon robust estimation and outlier rejection. We observed other vehicles misclassify dust clouds thrown up by their tires as phantom obstacles, and stop until these clouds dissipated. However, the temporal filtering and spatial smoothness constraints used with the SICK and Velodyne systems, respectively, made Little Ben robust to such false positives.

8 UFE Performance

On the basis of its performance during the NQE, Little Ben was seeded fourth entering the UFE competition. Overall, Ben's performance during the final event was quite good. The most significant shortcoming during the 57 mile race occurred during the first mission of the UFE. This involved the off-road portion of the course known as "the Outback" that connected Montana Street with Phantom Road East. During development, we had operated exclusively on paved roads and approximately planar off-road surfaces. In contrast, the Outback was a steep grade with dramatic pitch changes over short distances.

Fig. 22. Little Ben temporarily stalled on the transition from the Outback to Phantom Road East. The steep pitch of the transition resulted in the road surface being falsely classified as an obstacle.

As a result, Little Ben stalled at the bottom of the Outback as it transitioned to Phantom Road East. Due to the extreme pitch of the dirt road at the requisite stop line and low suspension and bumper clearance of Little Ben, the paved road surface was nearly touching the front bumper at this point in the course. This is shown in Figure 22. From this pose, the perceptual system interpreted the road surface as an obstacle immediately in front of the vehicle and refused to proceed through the intersection. By repositioning Little Ben a few meters ahead, he was able to continue through the remainder of the course.

Mission 1 also saw Little Ben execute what was arguably the most intelligent maneuver of the UFE. This occurred at a 4-way intersection, and required Little Ben to interact with 3 other robot vehicles and an even larger number of human operated traffic vehicles, as shown in Figure 23. Little Ben (in dashed box) arrived at the intersection with the UCF entry temporarily stalled to the right, and the MIT vehicle stopped to the left (upper left). Little Ben obeyed intersection precedence, and waited for MIT and UCF to proceed. The MIT vehicle made a right turn, but then became stopped temporarily against a curb. Upon determining the UCF vehicle was stalled, Little Ben began his planned right turn (upper right). Immediately, this brought Little Ben face-to-face with the Cornell vehicle, which had stopped temporarily in the wrong lane while attempting to pass other traffic (lower right). After several seconds, Little Ben executed a pass to maneuver around the Cornell vehicle, and then returned to his own lane (lower left). While the correctness of this behavior may seem obvious, what is significant is that of the 4 robots that appeared at this intersection, only Little Ben appeared to proceed as a human operator would have done.

Little Ben finished the 57 mile course in approximately 305 minutes, not including any penalty time that may have been assessed by DARPA. Remarkably, he was the only Track B entry that was able to complete the challenge.

Fig. 23. (Clockwise from upper left) Little Ben performs a right turn while passing a robot vehicle stopped in the wrong lane.

9 Summary

This paper has presented some of the technical details of Little Ben, the autonomous ground vehicle built by the Ben Franklin Racing Team for the 2007 DARPA Urban Challenge. After quantifying the sensing and reaction time performance requirements needed for the upcoming challenge, the hardware and software systems for Ben were designed to meet these stringent criteria. An array of GPS/INS, LIDAR's and vision sensors were chosen to provide both omnidirectional and long range sensing information. The software modules were written to robustly integrate information from the sensors, build an accurate map of the surrounding environment, and plan an optimal trajectory through the traffic network. This allowed the vehicle to successfully complete the 2007 DARPA Urban Challenge, even though we were severely constrained by time and budget constraints.

Acknowledgments

The team would like to express our gratitude to the following individuals who contributed to the success of this project: Allen Biddinger, Brett Breslow, Gilad Buchman, Heeten Choxi, Kostas Daniilidis, the Footes, Rich Fritz, Daniel Garofalo, Chao Gao, Erika Gross, Drew Houston, Ani Hsieh, Steve Jamison, Michael Kozak, Vijay Kumar, Samantha Kupersmith, Bob Lightner, Gerry

Mayer, Tom Miller, George Pappas, Ray Quinn, Ellen Solvibile, CJ Taylor, Chris Wojciechowsk and many others we have forgotten. We would also like to thank Mitch Herbets and Thales Communications for sponsoring our entry. Lastly, thanks to Global360 for allowing the use of video images in this report.

References

Amir, Y., Danilov, C., Miskin-Amir, M., Schultz, J., Stanton, J.: The Spread toolkit: Architecture and performance. Technical Report CNDS-2004-1, Johns Hopkins University, Baltimore, MD (2004)

Coulter, R.: Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Carnegie Mellon University, Pittsburgh, PA (1992)

DARPA (2007), http://www.darpa.mil/grandchallenge/rules.asp (retrieved July 24, 2008)

Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)

Elfes, A.: Using occupancy grids for mobile robot perception and navigation. IEEE Computer Magazine 22 (1989)

Gillespie, T.D.: Fundamentals of vehicle dynamics. In: Society of Automotive Engineers, Warrendale, PA (1992)

Guitton, A.: The iteratively reweighted least squares method. Stanford University Lecture Notes (2000)

Kalman, R.E., Bucy, R.S.: New results in linear filtering and prediction theory. Transactions of the ASME 83, 95–107 (1961)

Rasmussen, C., Stewart, A., Burdick, J., Murray, R.M.: Alice: An information-rich autonomous vehicle for high-speed desert navigation. Journal of Field Robotics 23(9), 777–810 (2006)

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., James Diebel, P.F., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley: The robot that won the darpa grand challenge. Journal of Field Robotics 23(9), 661–692 (2006)

Vernaza, P., Lee, D.D.: Robust GPS/INS-aided localization and mapping via GPS bias estimation. In: Proceedings of the 10th International Symposium on Experimental Robotics, Rio de Janeiro, Brazil (2006)

Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment

Isaac Miller[⋆,⋆⋆], Mark Campbell[⋆⋆,⋆⋆⋆] Dan Huttenlocher[†,‡], Aaron Nathan[§], Frank-Robert Kline[‡], Pete Moran[⋆⋆], Noah Zych[⋆⋆], Brian Schimpf[¶], Sergei Lupashin[§], Ephrahim Garcia[⋆⋆], Jason Catlin[§], Mike Kurdziel[⋆⋆], and Hikaru Fujishima[⋆⋆]

Cornell University, Ithaca NY 14853
itm2@cornell.edu, mc288@cornell.edu, dph@cs.cornell.edu, amn32@cornell.edu, fk36@cornell.edu, pfm24@cornell.edu, ncz2@cornell.edu, bws22@cornell.edu, svl5@cornell.edu, eg84@cornell.edu, jac267@cornell.edu, msk244@cornell.edu, hf86@cornell.edu

Abstract. Team Cornell's 'Skynet' is an autonomous Chevrolet Tahoe built to compete in the 2007 DARPA Urban Challenge. Skynet consists of many unique subsystems, including actuation and power distribution designed in-house, a tightly-coupled attitude and position estimator, a novel obstacle detection and tracking system, a system for augmenting position estimates with vision-based detection algorithms, a path planner based on physical vehicle constraints and a nonlinear optimization routine, and a state-based reasoning agent for obeying traffic laws. This paper describes these subsystems in detail, before discussing the system's overall performance in the National Qualifying Event and the Urban Challenge. Logged data recorded at the National Qualifying Event and the Urban Challenge are presented and used to analyze the system's performance.

1 Introduction

Team Cornell's 'Skynet,' shown in Figure 1, is an autonomous Chevrolet Tahoe built to compete in the 2007 DARPA Urban Challenge. Skynet was built and developed at Cornell by a team composed primarily of members returning with experience from the 2005 DARPA Grand Challenge. The team remained small, with 12 core members supported by 9 part-time contributors. Experience levels included professors, doctoral and master's candidates, undergraduates, and

⋆ Graduate Research Fellow.
⋆⋆ Sibley School of Mechanical and Aerospace Engineering.
⋆⋆⋆ Associate Professor.
† Dan Huttenlocher is the John P. and Rilla Neafsey Professor of Computing, Information Science and Business and Stephen H. Weiss Fellow.
‡ Computer Science Department.
§ School of Electrical and Computer Engineering.
¶ School of Operations Research and Information Engineering.

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 257–304.
springerlink.com

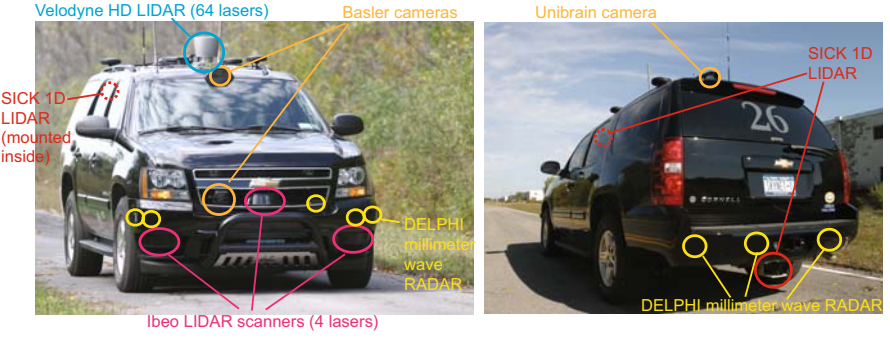


Fig. 1. Team Cornell's 'Skynet.'

Cornell alumni. The team was selected from a grant proposal as one of 11 research-oriented teams to receive funding from DARPA to compete in the Urban Challenge. Additional support was gained through corporate sponsors, including Singapore Technologies Kinetics, Moog, Septentrio, Trimble, Ibeo, SICK, MobilEye, The Mathworks, Delphi, and Alpha Wire.

Team Cornell's development cycle proceeded as a carefully-monitored research and engineering endeavor. The team ethos is one of maintaining knowledge across generations of researchers, and the system has largely been developed and tested amidst lessons learned from the team's participation in the 2005 DARPA Grand Challenge (Miller et al., 2006). The entire team designed and reviewed the system in weekly research meetings. As the system matured to testing, the entire team relocated to the Seneca Army Depot in Romulus, NY, where Team Cornell built and maintained a private road network for autonomous vehicle testing. Autonomous testing was conducted formally at the Seneca Army Depot, with a safety rider selected to monitor Skynet's decisions from the driver's seat for each test. Tests were coordinated between the safety driver, traffic drivers, and developers through the use of two-way radios. A remote emergency disabling switch was also actively maintained from a support vehicle during each test. In general, the system was brought online cautiously: development began in simulation, progressed to evaluation with sensor data logs, then to unit testing with actuators disabled, and finally to full closed-loop autonomous driving.

The remainder of the paper describes the system architecture, components, and performance in the Urban Challenge, all in the context of lessons learned from the Grand Challenge. Section 2 begins with a description of the Skynet's system architecture and timing and interface protocols. Section 3 continues with a detailed algorithmic description of each major subsystem in Skynet. Section 4 presents results for individual systems tested at the Urban Challenge National Qualifying Event. Section 5 presents general results and performance in the Urban Challenge Final Event, with several case studies used to highlight unique scenarios. Section 6 concludes with a review

of lessons learned in the 2005 Grand Challenge, as well as new lessons and research questions posed by the results of the Urban Challenge.

2 System Architecture and Data Flow

The general system architecture for Team Cornell's Skynet is shown in Figure 2 in the form of key system blocks and data flow. These blocks form the multi-layer perception and planning / control solution chosen by Team Cornell to successfully drive in an urban environment. Details of each of these blocks are given in section 3.

The world is perceived using two groups of sensors. Skynet itself is sensed using a combination of Global Positioning System (GPS) receivers, an Inertial Measurement Unit (IMU), and vehicle odometry. These raw measurements are then fused in real time in the 'pose estimator,' producing tightly-coupled position, velocity, and attitude estimates in an Earth-fixed coordinate frame. Skynet's environment, defined in the Urban Challenge as parked and moving

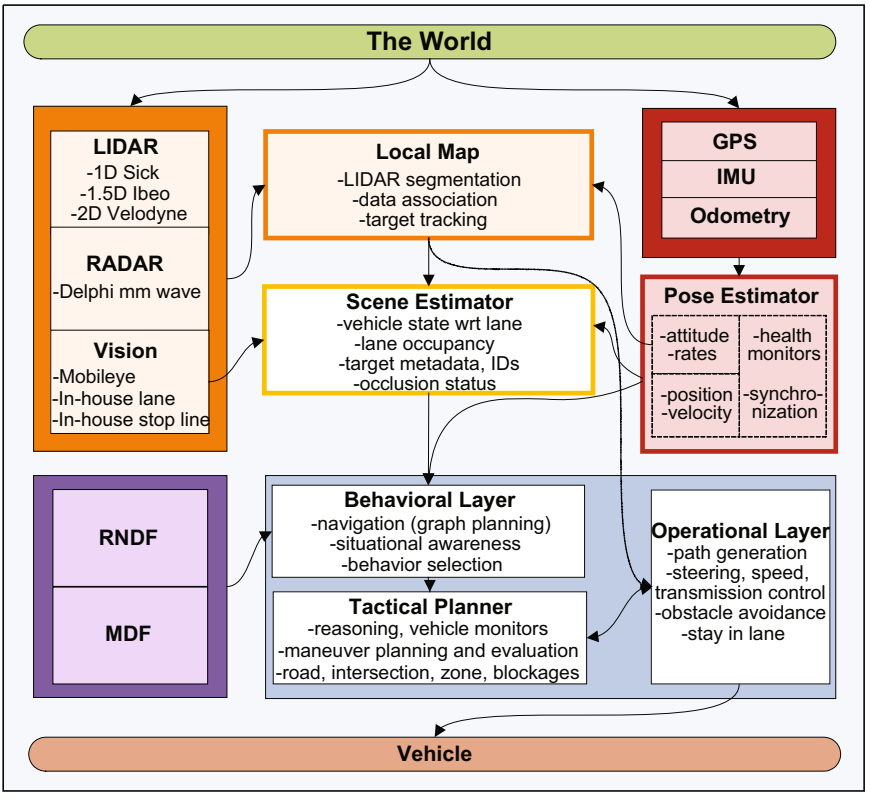


Fig. 2. System architecture of Team Cornell's Skynet.

cars, small and large static obstacles, and attributes of the road itself, is sensed using a combination of laser rangefinders, radar, and vision.

Two levels of probabilistic perception are used in Team Cornell's solution. The 'local map' fuses laser, radar, and vision data, along with vehicle rotation rate and ground velocity measurements, to initialize, locate, and track static and dynamic obstacles over time. The local map tracks obstacles relative to Skynet, making no distinction between small / large or stationary / moving obstacles. The 'scene estimator' then uses the raw object tracking estimates, pose estimates, and road cues from processed vision measurements in order to develop key statistics about Skynet and obstacles for the intelligent planner. Two sets of statistics are generated: those concerning Team Cornell's vehicle, such as its location with respect to the road and which lane it occupies, and those concerning other obstacles, including position / velocity, an identification number, lane occupancy, car likeness, and occlusion status based on other obstacles.

Planning and control for Skynet begins with road map and mission files: the Route Network Definition File (RNDF) and the Mission Data File (MDF), respectively. These files are supplied by DARPA and required at the start of each mission. The RNDF lists all legally traversable lanes as ordered sequences of GPS waypoints; it also marks certain waypoints as 'checkpoints,' stop lines, and exit or entry points connecting multiple lanes. The MDF specifies an ordered series of checkpoints Skynet must achieve to complete its mission, as well as maximum speed limits Skynet may travel in each part of the RNDF.

Planning over the RNDF and MDF occurs at three layers of intelligent planning. The topmost 'behavioral layer' uses the RNDF and MDF, along with obstacle information from the scene estimator and inertial estimates from the pose estimator, to interpret the environment and plan routes to achieve mission progress. The behavioral layer also decides which of four behavior states should be executed: road, intersection, zone, and blockage. Each of these four behaviors is then executed in the 'tactical layer,' where more finely detailed reasoning and planning occurs. The 'operational layer,' the lowest level of planning, produces a target path by smoothing an initial coarse grid-based path into one physically achievable by Skynet without violating constraints imposed by speed limits, road boundaries, and vehicle capabilities. The operational layer also has the responsibility of avoiding nearby obstacles, and it is therefore referenced only to Skynet itself. It uses no Earth-fixed information such as GPS positioning, similar to the local map. The interface between the tactical and operational layers can be iterative and is complex; this interface is described in more detail in section 3.5.

The operational layer combines the target path defined by desired curvature and speed with models of Skynet to produce steering control (desired wheel angle), speed control (desired throttle and brake position), and transmission commands for Skynet. The commands are implemented by individual automation actuators (steering wheel, throttle, brake, transmission, and turn

signals) on a stock SUV chassis. Standard aerospace motors are used for the wheel, brake, and transmission actuation, while the throttle is drive-by-wire through the stock General Motors CAN network.

2.1 Real-Time Data Distribution Network

Learning from system integration difficulties experienced in the 2005 Grand Challenge, Team Cornell addressed communication, synchronization, and data flow in its Urban Challenge entry before the constituent systems were designed. With the complex system architecture shown in Figure 2, three problems impact the design the most. First, there are simply massive amounts of data to be distributed between software and hardware modules around the car; approximately 76 Mb of data is transported across the car's networks each second. Second, the different software modules on the car each have varying requirements on the level of synchronization and timing of the data necessary to satisfy their design requirements. Third, the many different types of sensors, actuators and software modules each require a custom interface design.

In order to overcome these challenges, Team Cornell developed and used a Real-time Data Distribution Network (RDDN). The RDDN uses dedicated real-time microprocessors to interpret, timestamp, and broadcast sensor data using the User Datagram Protocol (UDP) over a standardized Ethernet network. This RDDN allows sensor data to receive accurate time stamps through synchronization with a master microcontroller, a benefit critical to higher level sensor fusion. The common Ethernet interface also allows any computer to listen to any sensor without specialized hardware, and it allows real-time data to be simulated in playback over the network.

Specifically, the RDDN is composed of a 100-BaseT network of Motorola 9S12NE64 microcontrollers, each with embedded Ethernet MAC and PHY. Custom built Ethernet-ready microcontrollers are interfaced to all sensors and actuators, including cameras, laser rangefinders, radars, the IMU, GPS receivers, the GM CAN network, and actuation controllers. The microcontrollers are synchronized with a single, master microcontroller with 0.1 ms accuracy using the IMU time stamp as a reference. Software modules are distributed on separate servers, each linked with switches. The modular microcontroller / server / switch system used a UDP multicast distribution of all sensor data, which enables universal availability of all data sources with accurate time stamping.

3 Component Descriptions

This section provides algorithmic descriptions of the independent subsystems implemented in Team Cornell's Skynet. Discussion begins with a description of the vehicle actuation and power distribution system in section 3.1. Section 3.2 continues with a description of Team Cornell's tightly-coupled attitude

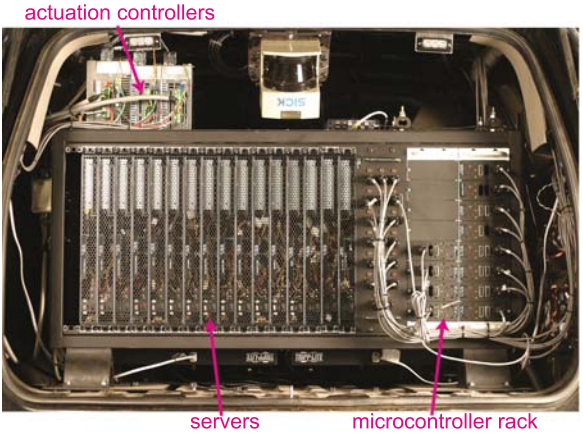


Fig. 3. Real time data distribution system, with servers and switches (left) and microcontrollers (right).

and position estimator. Section 3.3 describes Skynet's obstacle detection and tracking system. Section 3.4 describes Skynet's algorithms for applying contextual information from the urban environment. Section 3.5 completes the discussion of Skynet's subsystems with a description of its intelligent planner.

3.1 Vehicle Hardware

Team Cornell's Skynet is built on a 2007 Chevrolet Tahoe converted for autonomous operation. Selection of the Tahoe and its subsequent modifications were driven by two primary design requirements: responsiveness and reliability. Both design requirements are motivated directly by the environment of the Urban Challenge. An autonomous agent capable of fast response can adapt to potentially erratic behavior displayed by other robotic agents that might not be characteristic of a human-populated urban environment. The short development cycle of Urban Challenge vehicles also makes reliability a key requirement, since a vehicle that can be repaired quickly without specialized parts can get back to autonomous testing without much downtime. Team Cornell addresses these primary design requirements in four aspects of hardware selection: the vehicle chassis, the power subsystem, the drive-by-wire actuation, and the hardware packaging, each described below.

3.1.1 Chassis

Team Cornell's selection of the 2007 Chevrolet Tahoe as its vehicle chassis is based on design decisions intended to bolster responsiveness and reliability. One factor relevant to the decision is the fact that the Tahoe, as a full-size SUV, has sufficient room for a large number of computers, actuators, and human safety riders. Additionally, as an SUV it is larger and heavier than a

typical sedan, making it more likely to survive low-speed collisions without serious damage. Also, the stock Tahoe has a large engine bay with provisions for auxiliary power generation, as Tahoes are prepared for conversion to emergency vehicles. Finally, the 2007 Tahoe comes equipped with a large set of easily-accessible sensors, including stock throttle, odometry, and health monitoring sensors, which are all used without modifying the Tahoe's electronics and throttle.

The commercially-available Tahoe also addresses many reliability issues that affected Team Cornell's 2005 DARPA Grand Challenge entry (Miller et al., 2006). First, the manufacturer's warranty and stock parts ensure reliable operation, short lead time in acquiring replacement parts, and fast repair times. Second, the stock chassis has many rigid mounting points far inside the frame, allowing a computer rack to be mounted out of harm's way. Third, the 1776 lb. payload capacity ensures that the vehicle's performance is unaffected by large numbers of computers, batteries, and sensors. Finally, the common use of the Tahoe as an emergency vehicle makes a wide variety of commercially-available after-market bumper, roll cage, and alternator kits available for additional reliability.

3.1.2 Power Subsystem

The design of Team Cornell's power subsystem was largely driven by lessons learned from the 2005 Grand Challenge. From that experience and a study of current computational hardware, initial power requirements were set at 2400 W, with an additional 1500 W budgeted for actuators at peak load. In addition, Team Cornell's power subsystem was designed to tolerate short periods with no power generation due to inevitable minor power and equipment failures. Finally, the system was designed to switch readily between onboard power generation and wall power in the laboratory to permit extended algorithm testing without starting Skynet or using a generator.

Team Cornell used a separate power generator in the 2005 Grand Challenge, resulting in significant heat, noise, and reliability issues (Miller et al., 2006). In the Urban Challenge, the team opted instead to generate additional power with a secondary alternator, manufactured by Leece-Neville, mounted in the engine bay. The secondary alternator provides 200 amps peak output current at 24 volts to drive a pair of Outback Power Systems inverters, mounted behind the front seats. Each inverter is capable of sourcing up to 3500 W at peak. Both the alternator and the inverters exceed design requirements with a margin of safety. As a secondary system they also function independently from the stock electrical system on the Tahoe, so critical vehicle electronics do not compete with the computers and actuators for power. In addition, the design is redundant and can operate if one inverter fails. Finally, the inverters generate clean enough power that sensitive electronics operate seamlessly, even during the transition from Skynet to wall power.

While the inverters provide the primary source of power to Skynet, they also charge four Optima deep-cycle automotive batteries mounted behind the

inverters. These batteries are charged while the throttle is actively applied, and they are able to provide peak power even when the vehicle idles. The batteries also provide temporary power if the engine stops or Skynet needs to be restarted. The batteries act much like an uninterruptible power supply, but with an extended duration of approximately 6 hours of reserve power.

3.1.3 Automation

The most significant decision affecting Team Cornell's development cycle was the choice to design and build the actuation scheme in-house for converting Skynet to drive-by-wire operation. This decision was only made after an extensive review of performance specifications, costs, and features of three commercially available alternatives: EMC, AB Dynamics, and Stahle. Three factors were considered critical in the decision. Scheduling was the first design constraint, and only the EMC conversion and in-house actuation were feasible within the Urban Challenge development cycle. The second factor was the ability to repair or modify the actuation quickly, which is necessary to resume testing rapidly in the event of a failure. This factor was perhaps most critical in the decision, as a failure in any commercial solution would require significant time spent transporting the vehicle to a factory for repairs. The final factor was cost, measured both in time and in money. Team Cornell's relationship with Moog Aerospace allowed the team to obtain actuators at no cost, and the knowledge and feasibility of repairing or upgrading an in-house system far outweighed the time spent designing it.

Once Team Cornell decided to develop the vehicle actuation in-house, design specifications were created based on the most demanding maneuvers Skynet might need to perform during the Urban Challenge. In particular, a maximum steering angle performance requirement was defined as the ability to achieve a 700° change in steering wheel angle in 1 second. This requirement is taken from a standard NHTSA fishhook maneuver, which is used to test commercial SUV rollover at 35 mph. Similarly, a braking force requirement was defined as the ability to achieve 100 lbs. force of pedal pressure in 0.1 sec., a maneuver defined as a Class A stop in the Consumer Braking Information Initiative. Vehicle modifications were also designed without compromising Skynet's human interface and safety systems. As a result of these design specification the actuation scheme places no additional restrictions on Skynet's maneuverability: limiting factors are equal to or better than those available to a human.

3.1.4 Packaging

Packaging hardware into Skynet was performed considering requirements of easy access, reliability, and impact survival. The front driver and passenger seats in Skynet remain unmodified to hold two passengers, one as a safety driver and one as a developer. The actuation scheme is also packaged into the front seats: the steering actuator is mounted in parallel with the steering

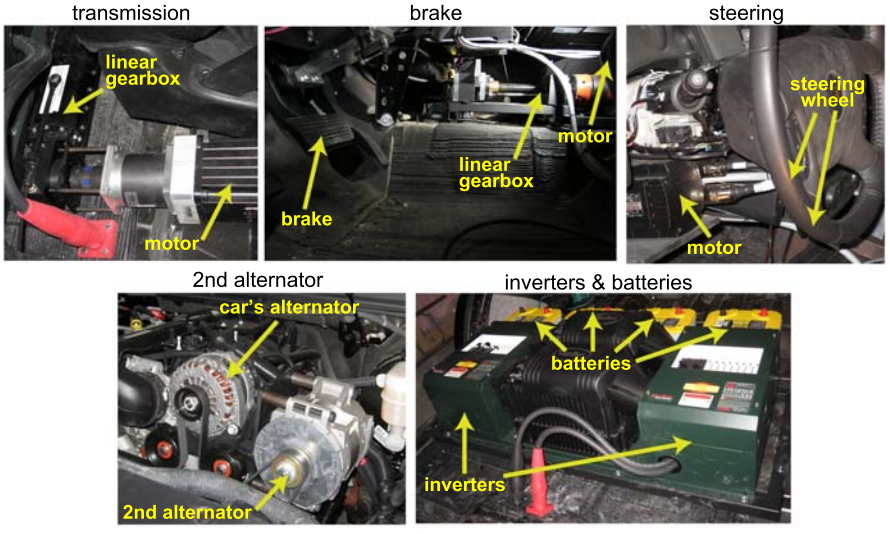


Fig. 4. (top): Team Cornell's custom actuation. (bottom): Team Cornell's power generation and backup solution.

column without affecting driver legroom, the brake actuator is mounted in the passenger leg area with a pull cable attached to the brake pedal, the throttle actuator is entirely electronic and built into a computer console between the front seats, and the transmission actuator is mounted behind the driver's seat with a push / pull cable attached to the transmission. The middle seats of Skynet are removed entirely and are replaced with the inverters and batteries for power generation and storage. A protective steel case covers both the batteries and inverters and allows easy access to the trunk. The third row of seats is also entirely removed, and is replaced by Team Cornell's computer rack. The computer rack assembly mounts rigidly to Skynet's frame, but is isolated from shock and vibration by four flexible mounting brackets provided by Barry Controls, Inc.

The rack itself is a steel frame custom built in-house to hold 17 single rack unit (1U) form factor computers, identical for rapid replacement. The rack is deliberately designed to house as many computers as possible, as its design was based on pessimistic overestimates of Skynet's computing requirements. In fact, Skynet fully utilizes only 7 of its 17 computers: one for position estimation, one for obstacle tracking, one for scene estimation, one for lane-finding, two for constrained path planning, and one for high level rule-based path planning. The remainder of the computers run comparatively lightweight sensor processing algorithms. No attempt has been made to package these algorithms more efficiently, as Skynet has been designed as an expandable research platform. Each of Skynet's computers houses a dual-core Pentium Mobile laptop processor, with clock speeds ranging from

1.66 GHz to 2.13 GHz and 2 Gb RAM. Laptop processors are used for heat and power savings, permitting the computers to be run safely in midday heat using Skynet's stock air conditioning system. All computers run Windows Server 2003, selected due to the team's extensive experience with programming in the Windows environment. Potential Windows timing and thread scheduling problems have been avoided through the use of accurate microcontroller time stamping. Skynet's time stamped Ethernet RDDN, discussed in section 2.1, allows each process to maintain correct temporal ordering of data while tolerating common deviations in thread scheduling and execution times.

Cable control around Skynet was also considered in packaging the hardware during Skynet's autonomous conversion. All roof cables were routed together through a single waterproof (IP67) roof breach. Power cables are run on the floor of the car upward to power computers and microcontrollers in the rack, and Ethernet data cables flow down from the interior ceiling.

To improve survivability in the case of a collision, most sensors were mounted inside Skynet. Forward and rear-facing Ibeo and SICK laser rangefinders were all embedded into the front and back bumpers. All forward, side, and rear-facing radars were also placed behind the bumper plastic, which did not affect their performance. Forward and rear looking cameras were mounted on the roof and behind Skynet's front grille. Side mounted SICK laser rangefinders were also mounted inside Skynet, and the back door windows were replaced with infrared transparent plastic to allow detection and scanning from a safe vantage point.

3.2 Position, Velocity, and Attitude Estimation

Team Cornell's pose estimator fuses external satellite navigation signals with onboard sensing to supply the real-time position, velocity, and attitude (pose) solution used for absolute positioning. Team Cornell's pose estimator was designed and built with several objectives in mind. First, as the sole source of external absolute position information, the pose estimator must by itself meet the absolute positioning requirements of the Urban Challenge: driving within potentially unmarked lanes, stopping within one meter of designated stop lines, and parking in potentially unmarked parking spaces. Each of these requirements mandates at least sub-meter positioning accuracy in the solution produced by the pose estimator. Second, the pose estimator must supply estimates of Skynet's differential motion to permit transformation of a vehicle-fixed coordinate frame over time. This requirement stems largely from deliberate design decisions made to reference all obstacles and path constraints in a vehicle-fixed coordinate frame, which is not affected by discontinuous changes in the absolute position solution as the quality of the GPS signal evolves over time. Finally, and most importantly, the pose estimator must produce a smooth and robust solution in the changing urban environment. It must be designed to cope with rapidly changing GPS

satellite visibility, multipath and other signal distortion, and potential short-term GPS blackouts due to the presence of trees and buildings. In essence, biased, overconfident, and brittle pose estimates are all particularly fatal as the pose solution is ultimately used to determine which rules of the road apply at each vehicle planning cycle.

Although off-the-shelf pose estimators are sufficient to satisfy the design requirements, a cost analysis drove Team Cornell to develop a custom pose estimator in-house. In particular, Team Cornell already possessed all the necessary hardware and expertise from the 2005 DARPA Grand Challenge to implement the pose estimator in-house, so the tradeoff fell between time spent developing the custom solution and the up-front price paid to purchase a top quality off-the-shelf equivalent. In the end, better understanding of GPS signal behavior for debugging, the freedom to design a custom interface, and the ability to make filtering considerations specific to the Urban Challenge tipped the scales against off-the-shelf equivalents.

The pose estimator that evolved from these design considerations fuses information from four sensors: a Litton LN-200 inertial measurement unit (IMU), Skynet's ABS wheel encoders, a Septentrio PolaRx2e@ GPS receiver, and a Trimble Ag252 GPS receiver. The LN-200 IMU is a combined three-axis fiber optic rate gyroscope and a three-axis silicon accelerometer, rigidly mounted on the floor of Skynet, along its centerline, just above the rear axle. The LN-200 integrates its rate gyros and accelerometers to report vector measurements of changes in orientation and velocity of its internal IMU-fixed coordinate frame as a digital signal at 400 Hz. These measurements may then be integrated to determine the position and orientation of the IMU relative to an initial configuration. Skynet's stock ABS sensors, optical encoders mounted at each wheel, are also used to aid in this dead-reckoning integration scheme. Information from these encoders is retrieved over Skynet's stock GM CAN network at 30 Hz and is used to provide a measurement of vehicle speed to help slow IMU integration errors' rate of growth. The two GPS receivers, the Septentrio and the Trimble, are both used to keep the integration errors in the dead-reckoning scheme bounded when GPS signals are available. The Septentrio, a three-antenna, single-clock, 48 channel GPS receiver, provides raw pseudorange, Doppler shift, and carrier phase measurements of all visible GPS satellites on all its antennas at 5 Hz synchronized measurement intervals. Its antennas are mounted in an 'L' pattern on Skynet's roof, as far apart as possible to promote greater observability in the differential signal between the antennas. The Septentrio also decodes the WAAS signal at the same rate to provide higher fidelity GPS error models. Finally, the Trimble, an agricultural grade single-antenna GPS receiver, is used solely to decode high precision (HP) OmniSTAR differential corrections. These corrections are supplied at 10 Hz with an advertised accuracy of 10 cm (Trimble, 2004). This HP signal, when statistically validated, is the primary source of sub-meter positioning information for satisfying the first pose estimator design requirement.

Team Cornell's pose estimator blends its four sensors in a tightly-coupled estimator that utilizes the strengths of each sensor while compensating for their weaknesses with sensor diversity. The LN-200 provides the fast and accurate update rates necessary to produce a smooth and statistically robust solution, but it must be integrated and therefore suffers from integration errors. The wheel encoders slow down the rate of growth of IMU integration errors, but are subject to errors due to wheel slip. The Septentrio, although providing data at a much slower rate, corrects integration errors with absolute positioning information obtained from GPS and WAAS. These three sensors can't consistently achieve sub-meter accuracy even when fused, but they can when supplemented with occasional updates from the HP signal received by the Trimble. The HP signal tends to be brittle, occasionally biased, and difficult to track for long periods of time, but statistical hypothesis tests using information fused over time from the other sensors is sufficient to determine when the HP signal is usable and when it is not. The resulting pose estimator, described below, uses these sensor strengths and diversity to satisfy the system's design requirements.

Team Cornell's pose estimator builds upon techniques and lessons learned from the pose estimator built for Cornell's 2005 Grand Challenge entry (Miller et al., 2006). Like the 2005 pose estimator, it collects and fuses information from its constituent sensors via an extended square root information filter (SRIF). The SRIF is a numerically robust implementation equivalent to the traditional Kalman Filter (KF), but it achieves twice the numerical stability and precision by maintaining and propagating a square root of the state's information matrix instead of a state covariance matrix (Bierman, 1977). The SRIF also has the advantage of being able to correctly initialize state estimates with infinite covariance, which are simply represented in the SRIF as estimates with zero information. Beyond these two convenient features, the extended SRIF is functionally identical to a traditional extended Kalman Filter (EKF). The SRIF is divided into familiar prediction and update steps, which, for this application, correspond to a dead-reckoning numerical integration step and a GPS, HP, or encoder measurement correction step.

Similarities with Cornell's 2005 pose estimator end with the SRIF. Numerical integration for dead-reckoning and prediction is performed as a series of discrete Euler steps, each corresponding to a single change in orientation and velocity reported by the LN-200. Corrections for the Earth's rotation rate and coriolis and centripetal accelerations measured by the LN-200 are also made using Euler approximations in a method similar to that of Savage (Savage, 1998a), (Savage, 1998b). Velocity changes due to gravity are also subtracted from the integration using an Euler approximation, with the gravity vector calculated using a numerically stable Legendre polynomial construction algorithm and the full 360x360 EGM-96 gravity potential model (Lundberg and Schutz, 1988), (Lemoine et al., 1998). Time correlated GPS satellite range bias estimates, Septentrio receiver clock offset and drift rates, constant double-differenced carrier phase ambiguities, and constant rate gyro

and accelerometer biases are also filtered as part of the pose estimator, using the autocorrelated and random walk dynamic process models described by Bar-Shalom *et al.* (Bar-Shalom et al., 2001).

Asynchronous updates are performed within the pose SRIF as measurements become available from the wheel encoders, Septentrio, and Trimble. For wheel encoder measurements, a filter update is calculated from the measured speed of Skynet. For Trimble measurements, a filter update is calculated from the location of the Trimble, measured using the HP signal. For Septentrio measurements, the measured pseudoranges, Doppler shifts, and double differenced carrier phases are all used to update the pose estimate in the SRIF on a satellite by satellite, antenna by antenna basis. This updating technique 'tightly couples' the Septentrio to the IMU in the pose estimator by estimating vehicle pose, IMU biases, receiver clock errors, and satellite errors all within a single centralized SRIF. It has the advantage that information is processed from each GPS satellite individually, allowing the filter to gain information even when fewer than four satellites are visible (Artes and Nastro, 2005). More importantly, it allows satellite signals to be modeled individually, so that each satellite can be statistically tested for significant errors and weighted in the SRIF according to its signal quality. Team Cornell's pose estimator takes advantage of this opportunity by expanding the broadcasted GPS satellite signal model to include weather-based tropospheric corrections, time correlated multipath models, and receiver thermal noise (Saastamoinen, 1972), (Davis et al., 1985), (Bar-Shalom et al., 2001), (Sleewaegen et al., 2004), (Psiaki and Mohiudden, 2007). This expanded signal error model is then combined with previously fused pose estimates to perform a χ^2 hypothesis test on the entire set of GPS measurements to evaluate measurement validity (Bar-Shalom et al., 2001). If the set of measurements fails the hypothesis test, individual satellites are tested in an attempt to find a set of satellites with lower signal distortion. If no such set is found, the entire measurement is abandoned. Similar filter integrity monitoring hypothesis tests are performed on each wheel encoder measurement and each HP measurement to prevent measurements with significant errors from corrupting the filtered pose solution and disturbing vehicle behavior.

Team Cornell's pose estimator is implemented in C++ and runs dead-reckoning integrations at 400 Hz on a 64-bit dual-core Pentium-Mobile processor running Windows Server 2003. Full predictions of the square root information matrix are performed at 200 Hz due to the computational expense of a large QR-factorization mandated by the SRIF. The full pose solution is reported at 100 Hz to all other systems in Skynet.

3.3 Obstacle Detection and Tracking

Team Cornell's obstacle detection and tracking system, called the local map, fuses the output of all obstacle detection sensor measurements over time into one vehicle-centric map of the local environment surrounding Skynet.

The local map is built to satisfy four design requirements. First, diversity in both output information and placement of obstacle detection sensors around the car at a minimum requires a centralized algorithm to collect and fuse information to distinguish between traversable and occupied regions of the car's surroundings. Second, the local map needs to resolve conflicts between sensors. In this respect mapping the sensors to a single coordinate frame is not enough; the sensors must be actively fused to a single interpretation of the environment for the planner to act upon. Third, the system must predict and track the motion of dynamic obstacles, to allow the planner to respond differently to moving traffic vehicles and static obstructions. Finally, the local map must be stable and robust over time, despite the rapidly changing urban environment. In particular, the local map must remain stable as other intelligent maneuvering agents pass in and out of view of various sensors in Skynet. Such stability is required to perceive complex intersection, merging, and traffic scenarios correctly as well, making it a critical requirement for success in the Urban Challenge.

The local map fuses information from three sensing modalities: laser rangefinders, radars, and optical cameras. Team Cornell's Skynet is equipped with 7 laser rangefinders: 3 Ibeo ALASCA XT rangefinders, 2 SICK LMS 291 rangefinders, 1 SICK LMS 220 rangefinder, and one Velodyne HDL-64E rangefinder. The 3 Ibeos, mounted in Skynet's front bumper, each return range and bearing pairs taken from four separate laser beams at 12.5 Hz over a 150° field of view with angular resolution of approximately 1°. The 2 SICK 291s, mounted with vertical scan planes inside Skynet's rear doors, each return range and bearing pairs over a 90° field of view with angular resolution of approximately 0.5° at 75 Hz. The SICK 220, mounted in Skynet's rear bumper with a single horizontal scan plane, returns range and bearing pairs over a 180° field of view with angular resolution of approximately 1° at 37.5 Hz. The Velodyne, mounted on the centerline of Skynet's roof above the front seats, returns range and bearing pairs from 64 separate laser beams over a 360° field of view at 15 Hz. Skynet is also equipped with 8 Delphi FLR radars, each of which returns data for up to 20 tracked objects at 10 Hz. The Delphis are mounted 5 in the front bumper for forward and side-facing detection, and 3 in the rear for backward detection. Finally, Skynet is equipped with one backward-facing Unibrain Fire-i 520b optical camera, mounted just above the trunk. This camera runs MobilEye SeeQ software that reports tracked obstacles at approximately 15 Hz. Table 1 summarizes Skynet's obstacle detection sensors, and Figure 5 gives a top-down view of Skynet's laser rangefinder and radar coverage.

Like the pose estimator, the local map relies on sensor diversity to circumvent the shortcomings of each individual sensor. At the highest level, the laser rangefinders are most effective at determining obstacle position, but they do not measure obstacle speed. When combined with Delphi radars, which measure accurate speed (range rate) but poor range and bearing, the overall set of sensors yields accurate position and speed measurements of all obstacles near

Table 1. Skynet's obstacle detection sensors

Sensor	Location	Type	Rate	FoV	Resolution
Ibeo ALASCA XT	front bumper left	laser	12.5 Hz	150°	1°
	front bumper center	laser	12.5 Hz	150°	1°
	front bumper right	laser	12.5 Hz	150°	1°
SICK LMS 291	left back door	laser	75 Hz	90°	0.5°
	right back door	laser	75 Hz	90°	0.5°
SICK LMS 220	back bumper center	laser	37.5 Hz	180°	1°
Velodyne HDL-64E	roof center	laser	15 Hz	360°	0.7°
Delphi FLR	front bumper left (2x)	radar	10 Hz	15°	20 tracks
	front bumper center	radar	10 Hz	15°	20 tracks
	front bumper right (2x)	radar	10 Hz	15°	20 tracks
	back bumper left	radar	10 Hz	15°	20 tracks
	back bumper center	radar	10 Hz	15°	20 tracks
	back bumper right	radar	10 Hz	15°	20 tracks
Unibrain Fire-i 520b	back roof center	optical	15 Hz	20° − 30°	N / A

Skynet. This set of measurements are used within the local map to generate a central fused estimate of Skynet's surroundings, distinguishing traversable areas from those laden with obstacles, and identifying other moving agents.

To keep sensor interfaces with the local map simple, all sensor measurements are fused at the object level. That is, each sensor measurement is treated as a measurement of a single object, whether another intelligent agent or a typical static obstacle. Both the Delphi radars and the MobilEye SeeQ software fit easily into this framework, as their proprietary algorithms automatically transmit lists of tracked obstacles. The laser rangefinders, which all return lists of range and bearing pairs, are post-processed to fit into this object level framework. First, laser returns corresponding to the ground and any other objects too low to the ground to be vehicles are removed from the set of rangefinder points under consideration. This is accomplished through the use of a ground model constructed by rasterizing returns from each Velodyne frame into a time-filtered gridded ground map with techniques similar to those used in the 2005 Grand Challenge (Miller et al., 2006), (Miller and Campbell, 2006). The primary difference between this method of ground modeling and the 2005 approach is that it represents the ground in a vehicle-fixed coordinate frame, using differential vehicle motion estimates from the pose estimator to keep the terrain grid fixed to Skynet as it moves.

Once ground and low obstacle laser returns are removed from consideration, a clustering step is performed to group laser returns into distinct objects for measurement. The clustering algorithm uses Euclidean distance thresholds to assign cluster membership, and it is run separately with thresholds of 0.5m and 1m to generate clusters. Only clusters that are identical across both thresholds are considered distinct enough to be treated as measurable

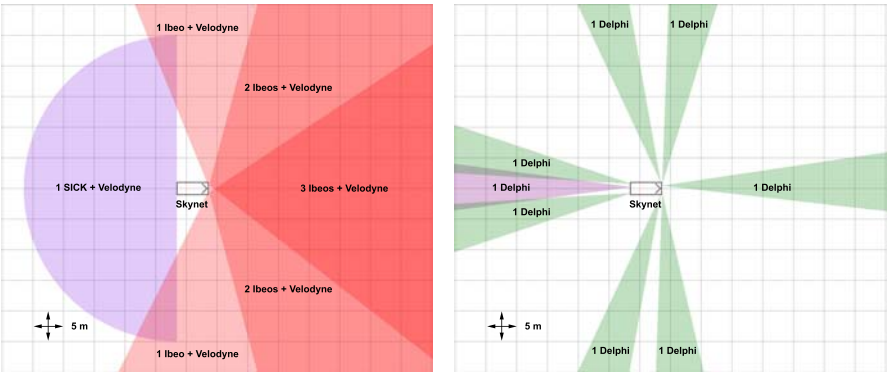


Fig. 5. (left): Laser rangefinder azimuthal coverage diagram for Team Cornell's Skynet. (right): Radar azimuthal coverage diagram. Skynet faces right in both coverage diagrams. A rear-facing optical camera is not shown, nor are two laser rangefinders with vertical scan planes that detect obstacles immediately to the left and right of Skynet.

obstacles. The number of clusters is also further reduced by analyzing occlusion boundaries in the ranges of nearby points; only clusters that are completely visible are considered distinct enough to generate measurements. From there, each distinct obstacle cluster is used to generate measurements which are then passed to the local map to be fused at the object level.

The local map fuses object measurements from its constituent sensors by treating the obstacle detection and tracking problem as the joint estimation problem of simultaneously tracking multiple obstacles and determining which sensor measurements correspond to those obstacles (Miller and Campbell, 2007). In the language of classical Bayesian estimation the problem is formulated as estimating the joint posterior:

$$p\left(N\left(1:k\right), X\left(1:k\right) | Z\left(1:k\right)\right) \tag{1}$$

where $N\left(1:k\right)$ is a set of discrete random variables indicating which sensor measurements correspond to which tracked obstacles at time indices 1 through k, $X\left(1:k\right)$ is a set of continuous random variables representing the states of the obstacles being tracked at time indices 1 through k, and $Z\left(1:k\right)$ are the full set of measurements from the first measurement frame to the current time index k. Note the number of obstacles is also represented implicitly as a random variable in the cardinality of $N\left(k\right)$ and $X\left(k\right)$ at any particular time index, and must also be determined in the local map's framework. In the Urban Challenge environment the joint estimation problem posed in equation 1 is generally too large to be solved by a direct brute force estimator, such as a particle filter, evaluating hypotheses over the full multivariate state space. Instead, Team Cornell factorizes the posterior to yield two manageable components:

$$p\left(N\left(1:k\right)|Z\left(1:k\right)\right)\cdot p\left(X\left(1:k\right)|N\left(1:k\right),Z\left(1:k\right)\right) \quad (2)$$

where, intuitively, $p\left(N\left(1:k\right)|Z\left(1:k\right)\right)$ describes the task of determining the number of obstacles to track and assigning measurements to those obstacles, and $p\left(X\left(1:k\right)|N\left(1:k\right),Z\left(1:k\right)\right)$ describes the task of tracking a known set of obstacles with known measurement correspondences. In the local map, these two densities are estimated separately using a particle filter to draw hypotheses about measurement correspondences and banks of extended Kalman Filters (EKFs) to track obstacles using each particle's hypothesis about measurement assignments (Miller and Campbell, 2007). This approach solves the joint estimation problem without wasting computational resources assigning particles over continuous obstacle states, which can be estimated effectively and inexpensively with standard EKFs.

Each EKF within a local map particle tracks one potentially moving obstacle under the measurement correspondence hypothesis of that particle. Each obstacle is modeled as a set of cluster points storing the geometric information known about that obstacle, and all those cluster points are referenced to an obstacle-fixed coordinate frame. The EKF then estimates the following states for each obstacle at time index k: the (x, y) location of the origin of that obstacle's coordinate frame, the angle ϕ orienting the obstacle's coordinate axes with respect to Skynet, the angle θ denoting the obstacle's heading relative to Skynet, and the obstacle's ground speed s. Obstacle maneuvers, i.e. changes in speed and heading, are modeled as random walks, with noise parameters set to encompass feasible accelerations and turning rates at Urban Challenge speeds. In the prediction step of an obstacle's EKF, these 5 state variables are numerically integrated with a 4^{th} order Runge-Kutta algorithm. This EKF prediction uses Skynet's differential motion estimates from the pose estimator to keep all tracked obstacles in a Skynet-centric coordinate frame. Although such a moving coordinate frame couples obstacle tracking errors to errors in Skynet's differential motion estimates, the quality of Skynet's IMU prevents this coupling from having any substantial impact on obstacle tracking. More importantly, the Skynet-centric coordinate frame ensures that absolute positioning errors, which depend heavily on the quality of the GPS environment, do not affect obstacle tracking at all.

The update step of each obstacle's EKF changes depending on the type of sensor providing the information. Both MobilEye and radar updates are performed as traditional EKF update steps with measurements consisting of a range, bearing, and range rate toward or away from the appropriate sensor. Updates from the side SICK 291s are scalar measurements consisting of distance from the side of Skynet to the obstacle. For the Ibeos and the rear SICK 220, 3 measurements are extracted from each clustered set of rangefinder points: bearings of the most clockwise and most counterclockwise points in the cluster, and range to the closest point in the cluster. This set of measurements is chosen specifically for its favorable and stable behavior under linearization, though the linearization is never computed explicitly (Miller and Campbell, 2007). Instead, the update step

utilizes the unscented transform from the Sigma Point Filter (SPF) to compute an approximate linearization of the measurement about the current obstacle state estimate and the current cluster of points representing that obstacle (Julier and Uhlmann, 1997). In all update types, measurement noise is assumed to be additive and Gaussian. This allows measurement covariance matrices to be set according to the characteristics of each sensor or obstacle detection algorithm, while still ensuring proper updates to obstacle state covariance matrices through the use of EKF and SPF linearizations.

Each particle of the local map represents one complete hypothesis about the set of obstacles in Skynet's environment and which sensor measurements correspond to those obstacles. At each sensor frame, the local map's particles choose non-deterministically whether each sensor measurement corresponds to a new obstacle, should be assigned to the EKF of a specific existing obstacle, or should be ignored as clutter. These choices are made according to the likelihood that the measurement corresponds to a model of where new obstacles are likely to be seen, to existing obstacles, or to the class of false measurements commonly made by each sensor. The local map is resampled at the end of any sensor frame in which the effective number of particles is less than half the true number of particles (Arulampalam et al., 2002). The most likely particle and all its tracked obstacles are then broadcasted at 10 Hz on Skynet's data network.

Like the pose estimator, the local map ensures that sensor measurements are processed in correct temporal order by keeping a queue of recent sensor measurements, sorted by age. The oldest measurements in the queue are used to update the local map once every sensor has either contributed a newer piece of data or timed out. This queuing structure, along with the rest of the local map, occupies a single core of a dual core Pentium-Mobile processor running Windows Server 2003. It reports the current most likely list of obstacles at 10 Hz to Skynet's data network. It also maintains a set of laser rangefinder points confirmed to be obstacles but not tracked, either because they were deemed unstable during clustering or because they were too small to be moving vehicles. This list is also reported on Skynet's data network to ensure the planner avoids all moving and static obstacles.

3.4 Environment Structure Estimation

Team Cornell's local map, described in section 3.3, broadcasts a list of tracked and untracked obstacles at 10 Hz to Skynet's internal data network for collision avoidance. These obstacles are maintained in a vehicle-centric coordinate frame, and are never referenced to absolute coordinates in the local map tracking scheme. This approach keeps absolute positioning errors from affecting obstacle avoidance, which only depends on the relative positioning of obstacles with respect to Skynet. This approach also avoids the potentially incorrect assumption that other moving agents obey the rules of the road, as

those agents face exactly the same pose, perception, and planning difficulties Skynet does.

Still, the Urban Challenge is more than basic obstacle avoidance. In order to obey the rules of the road, the vehicle must at some point modify its behaviors according to its and others' absolute locations, for queuing at intersections, maintaining safe following distances, and appropriately ignoring cars in oncoming lanes. When the environment is structured, the road constraints provide strong cues to improve Skynet's localization and perception of its surroundings. Team Cornell's scene estimator, described below, identifies and takes advantage of these environment cues.

3.4.1 Posterior Pose

The scene estimator consists of two algorithms, called 'posterior pose' and 'track generator,' that act as a data pipeline from the localization and perception systems to the planner. The first, posterior pose, is built to take advantage of the position cues hidden in the DARPA road network and the constraints of the Urban Challenge. In particular, it takes advantage of the DARPA-stated assurance that all waypoints are surveyed accurately, all lanes are marked as indicated in the route network definition file (RNDF), and all stop lines are painted correctly. Under these assumptions, posterior pose uses vision-based lane line and stop line detection algorithms to improve the pose estimator's estimate of Skynet's absolute position. The algorithm, based on GPS map aiding techniques, sets out to determine a simplified *a posteriori* joint density of Skynet's pose given road cues:

$$p\left(E\left(1:k\right), N\left(1:k\right), \Theta\left(1:k\right) | M, Z\left(1:k\right)\right) \tag{3}$$

where $\left(E\left(1:k\right), N\left(1:k\right)\right)$ is Skynet's East-North planar position in the RNDF at time indices 1 through k, $\Theta\left(1:k\right)$ is Skynet's heading measured counterclockwise from East at time indices 1 through k, M is the information obtained from the (static) DARPA RNDF, and $Z\left(1:k\right)$ are the available sensor measurements, including output from the pose estimator, two lane-finding algorithms, and a stop line detection algorithm (Miller and Campbell, 2008). In this simplified formulation of the pose estimation problem, Skynet has been constrained from free movement in a three-dimensional environment to planar motion. Note, however, that it still does not assume Skynet always stays on the road. This design choice simplifies the estimation problem considerably by reducing the size of the pose state vector to 3, compared to $40 - 50$ states in the pose estimator.

The substantial reduction in the size of the pose state vector allows the posterior pose estimation problem to be solved feasibly by a particle filter (Miller and Campbell, 2008). This type of filter has been chosen because it is more appropriate for the constrained pose estimation problem than a traditional EKF, as the posterior density in equation 3 is often strongly multimodal. In particular, vision-based road cue detection algorithms often commit

errors with strong modalities: detecting the wrong lane, detecting two lanes as one, or detecting a shadow as a stop line. The particle filter handles these errors appropriately by maintaining large numbers of hypotheses about vehicle pose, one per particle. Ambiguity about which lane Skynet occupies can then be represented accurately in the distribution of the particles across the road.

Each particle in the posterior pose filter contains one hypothesis of Skynet's three position states: East, North, and heading $[e(k), n(k), \theta(k)]$ within the RNDF. The particle filter is initialized by drawing an initial set of particles according to the approximate posterior Gaussian density implied by the mean and covariance matrix of a single pose estimator packet. From there, particles are predicted forward using differential vehicle motion estimates from the pose estimator. Updates are performed by adjusting the weight on each particle according to how likely its hypothesis about Skynet's pose is correct, as in a traditional bootstrap particle filter (Arulampalam et al., 2002).

The posterior pose particle filter performs one of three types of updates to its particles, depending on which of three sensing subsystems produces a measurement. The first update is an update from the pose estimator, where the particle filter simply adjusts the weights of its particles based on how closely they agree with the most recent information from the pose estimator. The remaining two updates, lane updates and stop line updates, compare expected road data extracted from the RNDF with local road data measured by three vision algorithms: two lane detection algorithms, and a stop line detection algorithm.

Team Cornell utilizes two vision-based lane-finding algorithms to generate measurements of Skynet's distance from the centerline of a lane and heading with respect to the lane. Both of these lane-finding algorithms operate on black and white images taken from a Basler A622F mounted on the centerline of Skynet just above the front windshield. One, the MobilEye SeeQ lane-finding software, is available commercially as a lane departure warning system. This system reports position and heading offsets of the lane Skynet currently occupies at approximately 15 Hz. The SeeQ system reliably detects painted lane lines, though it requires several seconds of uninterrupted tracking to become confident in its detections. The other lane-finding algorithm, designed in-house to act as the MobilEye's complement, uses slower but more accurate texture segmentation to find lanes even when roads are not painted (Felzenszwalb and Huttenlocher, 2004). This secondary algorithm produces the same type of lane offset and heading measurements as the MobilEye, though it runs at 2 Hz and starts from scratch at each image frame to avoid creating explicit temporal correlation in its lane estimates. When a measurement from either of these lane-finding algorithms is used to update the posterior pose particle filter, the filter uses the RNDF to compute what each particle expects its position and heading offset to be from the closest lane in the RNDF. Each of these lane hypotheses is then compared with the lane measurement generated by the vision algorithm, and the likelihood of the

measurement is used to update particle weights as in the traditional filter update.

Team Cornell also uses a vision-based stop line detection algorithm to generate measurements of Skynet's distance to a stop line. This detection algorithm operates on color images taken from a Basler A311F optical camera mounted in the front grille of Skynet and pointed toward the ground. The algorithm utilizes traditional Canny edge detection to search each image for properly-oriented pairs of edges, one a transition from dark road to white stop line paint, and its pair a transition from white paint back to road. Distances from the camera are computed for any matches found in this step. These distances are sent to update the posterior pose filter at a rate of 17.5 Hz. When one of these measurements arrives at the posterior pose filter, the filter uses the RNDF to compute the distance between each particle and its closest stop line. These expected distances are then compared with the output of the stop line algorithm, and the corresponding measurement likelihoods are used to update particle weights.

The posterior pose particle filter uses 2000 particles and runs on a single core of a dual core Pentium-Mobile system running Windows Server 2003. It processes all measurements from the pose estimator, the lane detection algorithms, and the stop line detection algorithm asynchronously, using a queuing structure similar to the local map to ensure that measurements are applied in order of their time stamps. The algorithm runs at 100 Hz, the rate at which the pose estimator broadcasts differential vehicle motion estimates. It reports a minimum mean square error (MMSE) posterior pose estimate at 10 Hz to Skynet's data network, along with other metadata, such as a mean square error (MSE) matrix and lane occupancy probabilities. This fused GPS / INS / vision posterior pose estimate is used as the sole position feedback signal in Skynet's path planner.

3.4.2 Track Generator

The second scene estimator algorithm, the track generator, combines Skynet's best position estimates from posterior pose with all the tracked obstacles from the local map to generate high level obstacle metadata required by the planner. Its primary purpose is to provide a track identification number for each tracked obstacle, one that remains constant over time and allows the planner a means to recognize different intelligent agents over time. This piece of data is one thing the local map particle filtered problem framework cannot provide outright, as it only reports the most likely map of tracked obstacles at any point in time: no effort is made within the local map to draw correspondences between similar tracked obstacles residing in separate local map particles.

The track generator poses this track identification problem as an estimation problem, driven by the output of the local map. Since the local map provides a list of all obstacles around Skynet at any point in time, the track generator simply has to match each of those obstacles with its current list

of obstacle tracks. Any obstacles that do not match the current list are used to create new tracks. The track generator performs this task using a global maximum likelihood estimator (MLE), implemented in a dynamic programming framework (Cormen et al., 2003). First, the likelihood $\lambda_{ij}(k)$ that the i^{th} local map obstacle at time k corresponds to the j^{th} previously identified track is computed for each obstacle / track pair. Three pieces of data are used to compute the correspondence likelihood: bearings of the most clockwise and most counterclockwise points in the obstacle, and range to the closest point in the obstacle. The unscented transform is used here as in section 3.3 to calculate a statistical mean $\mu_i^o(k)$ and covariance $P_i^o(k)$ for the two bearings and the range of the i^{th} obstacle, as well as a mean $\mu_j^t(k)$ and covariance $P_j^t(k)$ for the j^{th} track (Julier and Uhlmann, 1997). From there, the obstacle and track means and covariances are used to compute the likelihood of obstacle / track correspondence as if each were an independent Gaussian random variable:

$$\lambda_{ij} \sim \mathcal{N}\left(\mu_i^o(k) - \mu_j^t(k), P_i^o(k) + P_j^t(k)\right) \tag{4}$$

The track generator's dynamic programming scheme then searches the sets of correspondences matching each previously existing track to exactly one local map obstacle. The correspondence chosen is the one that maximizes the global likelihood $\Lambda(k)$:

$$\max_{j_1,\ldots,j_n} \Lambda(k) = \max_{j_1,\ldots,j_n} \prod_{i=1}^{n} \lambda_{ij_i}(k) \tag{5}$$

over the correspondences $\{j_1, \ldots, j_n\}$ under the constraint that no two correspondences are the same, i.e. $j_1 \neq j_2 \neq \ldots \neq j_n$. Any old tracks not matched by this scheme are immediately deleted, and any new local map obstacles not matched are assigned fresh track identifications. Each identification number assigned in this manner is unique over the track's lifetime, allowing the planner to reason about an agent by its corresponding identification number.

In addition to the identification number, the track generator also estimates four pieces of track metadata that it supplies to the path planner along with the track's state obtained from the local map. The first piece of metadata is lane occupancy: the probability that the track occupies any lanes near it in the RNDF. This probability is calculated via Monte Carlo quadrature. First, all sources of error modeled in estimating the track's location in the RNDF are transformed into a single East-North covariance matrix using a linearized covariance matrix transform similar to that used by the EKF (Bar-Shalom et al., 2001). The East-North covariance matrix is Monte Carlo sampled as a Gaussian random variable, and all sampled particles vote on which lane they occupy to generate lane occupancy probabilities for the track. The second piece of metadata generated for each track is an estimate of whether the track is stopped or not. This estimate is filtered using a Hidden Markov Model (HMM) over states 'is stopped' and 'is not stopped' using local map speeds as a measurement update (Russell and Norvig, 2003). The

third piece of metadata is an estimate of whether the obstacle's size indicates that it is carlike. This estimate is also implemented as an HMM, but uses the physical size of the track's cluster to provide evidence of whether it is too big or too small to be a car. The final piece of metadata is a track's visibility: whether it is occluded or not. Track visibility is computed by using the track's cluster points to mask out regions of sensor space under the assumption that the sensors can't see through any tracks. Any track that is detected as being blocked by another is marked as occluded, and any occluded tracks are allowed to persist in the track generator for up to 1 minute without any supporting evidence from the local map. This occlusion reasoning allows the track generator to remember other agents waiting at an intersection that are otherwise blocked from view by a car passing through. It also helps in dense traffic, where other agents may pass in and out of view rapidly.

The track generator reports its list of tracked obstacles and their metadata at 10 Hz on Skynet's data network. It runs on a single core of a dual core Pentium-Mobile processor, sharing the same process as posterior pose.

3.5 Intelligent Planning

Team Cornell's intelligent planning system uses the scene estimator's probabilistic perception of the environment to plan mission paths within the context of the rule-based road network. The system has been designed with three main objectives. First, the planner has been designed to be expandable such that additional behaviors could easily be added as the system was developed. Team Cornell used this design requirement to bring portions of the intelligent planner online independently, thus dividing mission planning into more manageable bits of development and testing. Second, the planner has been designed to operate asynchronously, so that low-level actuator control loops, middle level path following, and high level behavioral planning could all be run at separate rates as computational resources would allow. This design requirement also permitted multiple layers of the planner to be developed in parallel to cope with the aggressive schedule of the Urban Challenge. The final design requirement is stability and robustness: the system should remain stable over time despite unpredictable actions of other intelligent agents.

The intelligent planner developed against these design requirements is split into three primary layers. The top-level behavioral layer combines offline mission information with sensed vehicle and environment information to decide which high level behavioral state should be executed given Skynet's current context. The middle level tactical layer then plans a contextually-appropriate set of actions to perform given Skynet's current pose and the states of other nearby agents. The low-level operational layer then translates these abstract actions into actuator commands, taking into account road constraints and nearby obstacles. The operational layer also acts as a feedback sensor, verifying whether Skynet achieves the desired behaviors and reporting that completion status to the higher levels of the planner. The planner also consists

of several other smaller components, such as the messaging service and communications layer, which facilitate data transfer between the three primary layers of the planner. The following sections describe each of the three primary layers of the planner.

3.5.1 Behavioral Layer

The behavioral layer is the most abstract layer of Team Cornell's planner. Its goal is to decide the fastest route to the next mission checkpoint, and then to determine which of four high level behavior states to use to make progress along that route in the current vehicle state. The first part of that task, route planning, is solved using a modified version of the A* graph search algorithm (Russell and Norvig, 2003), (Ferguson et al., 2004). First, the DARPA road network is converted from the RNDF format to a graphical hierarchy of segments defining large contiguous portions of the road, ways defining directions of travel, lanes dividing a direction of travel according to the number of side-by-side vehicles that may be accommodated, and partitions that define a portion of a lane between two GPS waypoints (Willemsen et al., 2003). Zones and intersections are also represented in this graphical framework, but as general polygons rather than hierarchical lists of waypoints. The graph search algorithm then determines the shortest-time path to the next mission checkpoint using dynamically calculated traversal times as costs for road partitions, lane changes, turns, and other required maneuvers. Initial traversal time costs are generated from static information processed from the RNDF and the mission itself, including speed limits, lengths of partitions, lane right-of-way, and the configuration of stop lines at intersections. Dynamic traversal costs, such as road blocks, are also incorporated as large and slowly decaying time penalties on all road partitions adjacent to the location of each blockage as it is discovered.

Once a shortest-time path is planned to the next checkpoint, the behavioral layer repeatedly selects at each planning cycle one of an expandable list of high level behavior states as most appropriate for making progress along the desired path. The list of behavioral states used for the Urban Challenge, road, intersection, zone, and blockage, are deliberately defined as broadly as possible to promote stable vehicle behavior through infrequent state changes. Each of these high level behaviors executes a corresponding tactical component that drives Skynet until the next state change.

3.5.2 Tactical Layer

Each of the four components of the tactical layer is executed when Skynet transitions to its corresponding high level behavior, as described in section 3.5.1. All components are similar in that they divide the area surrounding Skynet into mutually exclusive and jointly exhaustive regions. All components also access a common list of intelligent agents currently monitored by the planner, each assigned a region based on the location of its point closest

to Skynet. These agents are tracked as a list of independent entities according to their track identification numbers, using state information from the track generator to monitor the agent's speed and shape, what partition the agent occupies, and whether the agent is disabled. A Hidden Markov Model (HMM) is also run over each agent's partition occupancy probabilities to determine which partitions it most likely occupies in the face of track generator uncertainty (Russell and Norvig, 2003). Differences between the tactical components lie in the types of region monitors they use and in the actions they take in response to events that occur. Each of these are set on a case-by-case basis for each tactical component.

The first tactical component is the road tactical, which controls Skynet when it drives down an unblocked road. This component is responsible for maneuvering into the proper lane to follow the shortest-time path, monitoring other agents in that lane for possible passing maneuvers, and ignoring agents in other lanes when it is safe to do so. Aside from basic lane keeping, these actions are largely performed in response to other agents. The road tactical monitors the agents by dividing the area surrounding Skynet into octants. At each planning cycle, octant monitors check the list of agents to determine how best to proceed along the road. In particular, separate checks are performed in front of Skynet for speed adjustment, adjacent to Skynet for possible lane changes, and behind Skynet for possible closing vehicles and possible reverse maneuvers (Sukthankar, 1997). The octant monitors are also used as inputs to a decision tree that evaluates the feasibility of a passing maneuver, the only optional maneuver in Skynet's repertoir. The decision tree was developed largely in simulation; it uses heuristic rules based on the distance to Skynet's next mission checkpoint and Skynet's closing speed to slower agents to determine whether Skynet can complete a passing maneuver in the space available. From the decision tree and the octant monitors, the road tactical selects a desired speed and lane for Skynet to follow. This target speed and the lane's local geometry near Skynet are then passed along to the operational layer as a path to track.

The second tactical component is the intersection tactical, which controls Skynet after it achieves an exit or a stop line waypoint. This component is responsible for achieving proper intersection queuing behavior and safe merging. When executed at an all-stop intersection, the intersection tactical creates monitors for each intersection entry that record agent arrival times by identification number for proper queuing order. The same monitors are set up in merging situations, except that lanes with right-of-way are constantly checked for oncoming vehicles and automatically given priority before Skynet is allowed to merge. When the intersection monitors determine that Skynet is first in the intersection queue, a target speed, goal point, and a polygon defining the intersection are passed along to the operational layer as a path to track.

The third tactical component is the zone tactical, which controls Skynet after it achieves an entry waypoint into a zone. This component is responsible

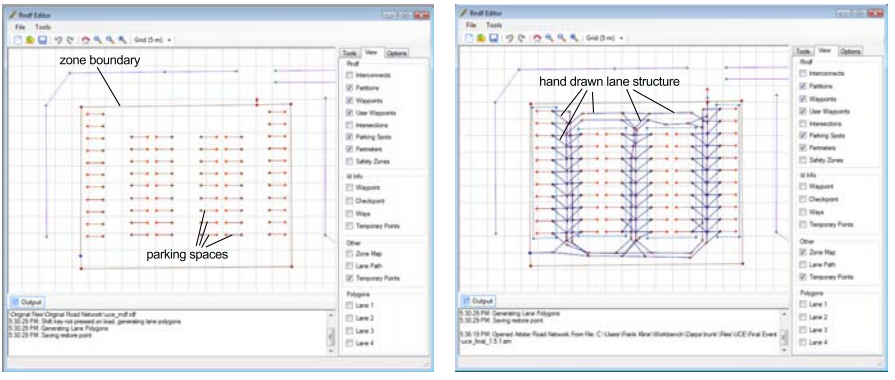


Fig. 6. (left): A portion of a parking lot zone from the DARPA Urban Challenge RNDF. (right): The same zone, but with Team Cornell's human-drawn lane structure applied during map preprocessing. Skynet uses the lane structure to treat zones like roads, much like a human does when assigning implied directions of travel in a parking lot.

for basic navigation in unconstrained zones, including basic obstacle avoidance and initial alignment for parking maneuvers. The zone tactical operates by planning over a human-annotated graph drawn on the zone during RNDF preprocessing. The graph imposes wide artificial lanes and directions of travel onto portions of the zone, allowing Skynet to treat zones as if they were roads. In this way the zone tactical operates much like the road tactical, by traveling along lanes selected by the A* algorithm in the behavioral layer. Figure 6 shows an example lane structure applied to a zone during map preprocessing. Such a lane structure was originally intended to be generated automatically, but time constraints mandated a hand-annotated solution instead.

In parking situations, the zone tactical is simply responsible for navigating into a polygon near the desired parking spot. The obstacle-free polygon, constructed as a workspace for Skynet to use to achieve a desired orientation, is built from sensor data as Skynet approaches the desired parking spot. After Skynet enters the workspace polygon, a cost-based heuristic search method is used to find a series of adjacent arcs to align Skynet to the parking spot. The same algorithm is then used in reverse to pull Skynet out of the parking spot, at which point the zone tactical resumes navigating the imposed lane structure. With the exception of parking, the zone tactical generates the same type of local lane geometry information as the road tactical to send to the operational layer as a path to track.

The final tactical component is the blockage tactical, which controls Skynet when forward progress on the current route is impossible due to obstacle positioning. This component is responsible for detecting road blocks, deciding whether they are temporary traffic jams, and acting accordingly. Team Cornell's blockage detection and recovery relies heavily on the operational layer's

constrained nonlinear optimization strategy, described in section 3.5.3. In particular, the operational layer informs the blockage tactical as to whether there is a forward path through the blockage, what the distance is to the blockage, and whether a reversing or repositioning maneuver is recommended. The operational layer and the blockage tactical then proceed with an escalation scheme to recover from the blockage. First, the blockage is confirmed over multiple planning cycles to ensure that it is not a short-lived tracking mistake. Second, a reversing or rerouting maneuver is executed to find an alternate route on the RNDF, if one is available. If an alternate route is available, the blocked route is given a large time penalty to encourage exploration of alternate routes. The magnitude of the penalty is finite and decays slowly in time, however, to allow Skynet to reexplore the original route if other routes are later discovered to be blocked as well. If no alternate route is available, Skynet's blockage recovery system escalates once more and resets the local map and scene estimator in an attempt to remove any catastrophic mistakes in obstacle detection. If this step fails, planning constraints are relaxed: first the admissible lane boundaries are widened, then obstacles are progressively ignored in order of increasing size. This process occurs over the course of several minutes without any vehicle progress. This tactical reasoning is unique among the four tactical components, as it relies solely on obstacle avoidance in the operational layer to recover to normal driving.

3.5.3 Operational Layer

The operational layer converts local driving boundaries and a reference speed supplied by the active tactical component into steering, throttle, and brake commands that are used to drive Skynet from one point to the next. The operational layer ultimately has the primary responsibility of avoiding all obstacles when choosing its path, as well as detecting blockages and infeasible paths from the set of obstacles in the commanded driving area. To accomplish this task, the operational layer first processes obstacle estimates from the track generator to prepare them for the trajectory optimizing planner. An obstacle processing loop first transforms obstacle point clouds into polygonal obstacles using a convex hull algorithm (Cormen et al., 2003). Both tracked obstacles and untracked obstacle points undergo this transformation, and metadata is attached to each resulting convex hull to retain all obstacle information sent from the track generator. Required and desired spacing constraints to safely avoid the obstacle are also generated based on the type of object being considered. All obstacle polygons are considered in a vehicle-centric coordinate frame, so they are not affected by GPS error.

Once obstacles are converted to polygons, they are combined with the desired region and speed of travel received to generate a path to follow. First, an initial unsmoothed path is planned through a discretized representation of the obstacles, formed by using nearby obstacle polygons to generate a vehicle-fixed occupancy grid (Martin and Moravec, 1996). The A* search algorithm is then run on the unoccupied portion of this occupancy grid to generate a

shortest path through the nearby obstacle field (Russell and Norvig, 2003). The resulting path is never explicitly driven; instead, it is used to determine which obstacles should be avoided on the right of Skynet, and which on the left. The initial path is then used to seed a nonlinear trajectory optimization algorithm for smoothing.

The nonlinear trajectory optimization algorithm attempts to smooth the initial base path into a physically drivable path subject to actuator constraints and obstacle avoidance. The algorithm first discretizes the base path into a set of n equally-spaced base points p_i, $i \in \{1, n\}$. This discretized base path is formulated such that the path originates from Skynet's current location, and the first base point lies in front of Skynet. A set of n unit-length 'search vectors' u_i, $i \in \{1, n\}$ perpendicular to the base path are also created, one for each base point. The trajectory optimizer then attempts to find a set of achievable smoothed path points $z_i = p_i + w_i \cdot u_i$, $i \in \{1, n\}$ by adjusting search weights w_i, $i \in \{1, n\}$. Target velocities v_i, $i \in \{1, n\}$ are also considered for each point, as well as a set of variables q_i^l and q_i^r, $i \in \{1, n\}$ indicating the distance by which each smoothed path point z_i violates desired spacings on the left and right of Skynet created by the list of polygonal obstacles.

Search weights, velocities, and final obstacle spacings are chosen to minimize the cost function J:

$$\begin{aligned} J\left(w_i, v_i, q_i^l, q_i^r\right) = \alpha_c \sum_{i=2}^{n-1} c_i^2 + \alpha_d \sum_{i=2}^{n-2} \left(c_{i+1} - c_i\right)^2 \\ + \alpha_w \sum_{i=1}^{n} \left(w_i - w_i^t\right)^2 + \alpha_q \sum_{i=1}^{n} \left(q_i^l + q_i^r\right) \\ + \alpha_a \sum_{i=1}^{n-1} a_i^2 - \alpha_v \sum_{i=1}^{n} v_i \end{aligned} \tag{6}$$

where α_c, α_d, α_w, α_q, α_a, and α_v are tuning weights, c_i is the approximated curvature at the i^{th} path point, w_i^t is the target search weight at the i^{th} path point, and a_i is the approximated forward vehicle acceleration at the i^{th} path point. Note the true curvature k_i at each discretized path point is:

$$k_i = \frac{2\left(z_{i-1} - z_i\right) \times \left(z_{i+1} - z_i\right)}{\left\|z_{i-1} - z_i\right\| \cdot \left\|z_{i+1} - z_i\right\| \cdot \left\|z_{i+1} - z_{i-1}\right\|} \tag{7}$$

To simplify differentiation, the following approximate curvature is used instead:

$$c_i = \left(z_{i-1} - z_i\right) \times \left(z_{i+1} - z_i\right) \tag{8}$$

This approximation is made noting that the initial path points p_i are equally spaced, and the search weights w_i are constrained to be small, so the denominator of equation 7 is approximately equal for all path points. The approximate forward vehicle acceleration a_i is also calculated under a similar approximation:

$$a_i = \frac{v_{i+1}^2 - v_i^2}{2 \cdot \|p_{i+1} - p_i\|} \tag{9}$$

The optimized cost function in equation 6 has 6 terms, each corresponding to a particular undesirable driving behavior. The first term penalizes large curvatures, which correspond to undesirably sharp turns. The second term penalizes rapid changes in curvature, which correspond to unstable swerving. The third term penalizes large deviations from the target path offset w_i^t, which force Skynet to move closer to the boundary of its allowed driving region. The fourth term penalizes violations of desired obstacle spacing. The fifth term penalizes sharp accelerations and braking. The sixth term penalizes slow velocities, encouraging faster plan completion time. Heuristic adjustments made to the relative weighting of these penalty terms determines Skynet's driving behavior: whether it prefers aggressive maneuvers or smoother driving.

The cost function $J\left(w_i, v_i, q_i^l, q_i^r\right)$ presented in equation 6 is optimized subject to a set of 6 rigid path constraints:

1. The path must begin at Skynet's current location and heading.
2. Each search weight w_i cannot push the smoothed path outside the boundary polygon supplied by the tactical layer.
3. Each obstacle spacing variable q_i^l and q_i^r cannot exceed any obstacle's minimum spacing requirement.
4. Each true curvature k_i cannot exceed Skynet's maximum turning curvature.
5. Total forward and lateral vehicle acceleration at each path point cannot exceed maximum limits defined by the acceleration ellipse:

$$\left(\frac{a_i}{a_{F,max}}\right)^2 + \left(\frac{k_i \cdot v_i^2}{a_{L,max}}\right)^2 \leq 1 \tag{10}$$

   where $a_{F,max}$ is the maximum allowed forward acceleration and $a_{L,max}$ is the maximum allowed lateral acceleration.
6. Each search weight w_i and set of slack variables q_i^l and q_i^r must never bring Skynet closer to any obstacle than its minimum allowed spacing.
7. The difference between consecutive path weights w_i and w_{i+1} must not exceed a minimum and maximum.

Additional constraints on initial and final path heading are also occasionally included to restrict the smoothed path to a particular end orientation, such as remaining parallel to a lane or a parking spot.

The constrained optimization problem is solved using LOQO, an off-the-shelf nonlinear non-convex optimization library. Two optimization passes are made through each base path to reach a final smoothed path. The first step of the smoothed path is then handed to two independent low-level tracking controllers, one for desired speed and one for desired curvature. The speed

controller is a proportional-integral (PI) controller with feedback linearization to account for engine and transmission inertia, rolling inertia, wind resistance, and power loss in the torque converter (Centa, 1997), (Gillespie, 1992), (Wong, 2001). The curvature controller uses an Ackermann steering model with cornering stiffness to convert desired curvature into desired steering wheel angle, which is then passed as a reference signal to a proportional-integral-derivative (PID) steering wheel angle controller (Gillespie, 1992), (Wong, 2001). This controller only feeds back on path curvature: lateral offset is not used as a feedback signal because all paths are planned from Skynet's current location and tracked in a Skynet-centric coordinate frame. The optimization is restarted from scratch at each planning cycle, and is run at 10 Hz on a dual core Pentium-Mobile processor running Windows Server 2003.

4 Performance at the National Qualifying Event

The National Qualifying Event (NQE) was held in Victorville, California, USA from October 25, 2007 to October 31, 2007 at the Southern California Logistics Airport. At the NQE, the 35 Urban Challenge vehicles invited to participate were tested on 3 courses, called 'Area A,' 'Area B,' and 'Area C.' Each area, described below in turn, tested one or more specific aspects of autonomous urban driving with focused and carefully-monitored scenarios. Only one robot was tested at a time, and any necessary traffic was provided by human-driven vehicles in preset patterns. Safe and sensible driving were paramount in the NQE, with stability and repeatability emphasized in the structure of the event.

4.1 Area A

In Area A, autonomous agents were required to merge into bidirectional traffic flowing at 10 mph in small concentric loops, extending approximately 100 m in the East / West direction and 50 m in the North / South. Vehicles were required to merge into and out of an unoccupied one-way North / South cross street that bisected the two loops of traffic. Successful merges entered and exited the cross street by completing left-hand turns across oncoming traffic. Traffic was dense, and vehicles were typically given windows of approximately 8 to 10 seconds to complete a merge. The lane geometry in Area A was restrictive, with nominal lane widths set at 12 ft for the inside lane and 10 ft for the outside lane. Concrete barriers were also placed flush with the boundary of the outside lane, so vehicles could not turn wide while merging. Vehicles were to complete as many of these merges safely as possible in an allotted time of approximately 30 minutes.

Team Cornell's vehicle ran in Area A twice, completing 5 successful laps (merges into and out of traffic) in the first attempt at Area A and 10 in the second attempt. In the first attempt, Skynet made 2 significant mistakes.

First, incorrect occlusion reasoning after the second lap led Skynet to conclude that there was an occluded obstacle permanently blocking its turn. The root cause of this was a measurement assignment mistake: a passing vehicle was mistakenly identified as part of a nearby concrete barrier. Team Cornell's vehicle waited for this phantom obstacle for several minutes before being paused and manually reset. This error prompted the team to restrict the lifetime of occluded obstacles to 1 minute if it is not supported by sensor data, as indicated in section 3.4.2.

A different problem occurred on Skynet's fifth lap, where the tactical layer pulled too far to the right of Skynet's lane in response to a nearby oncoming vehicle. During this maneuver, Skynet drove close enough to nearby concrete barriers to violate the operational layer's obstacle spacing constraints. The operational layer reported the path infeasible, and Skynet came to an abrupt stop before being paused by DARPA. When allowed to continue, the tactical layer issued a reverse command to Skynet, and the operational layer performed a reverse maneuver to align Skynet in the lane before resuming the mission. No human intervention was required to resolve the error, though the incident drew focus to the fact that the tactical layer could command Skynet to drive to the right of the lane's center without evaluating whether such a command would violate obstacle spacing constraints. This oversight was corrected after the first attempt at Area A, and remained in effect for the rest of the Urban Challenge. Minimum and desired obstacle spacings were also permanently adjusted to discourage large evasive maneuvers.

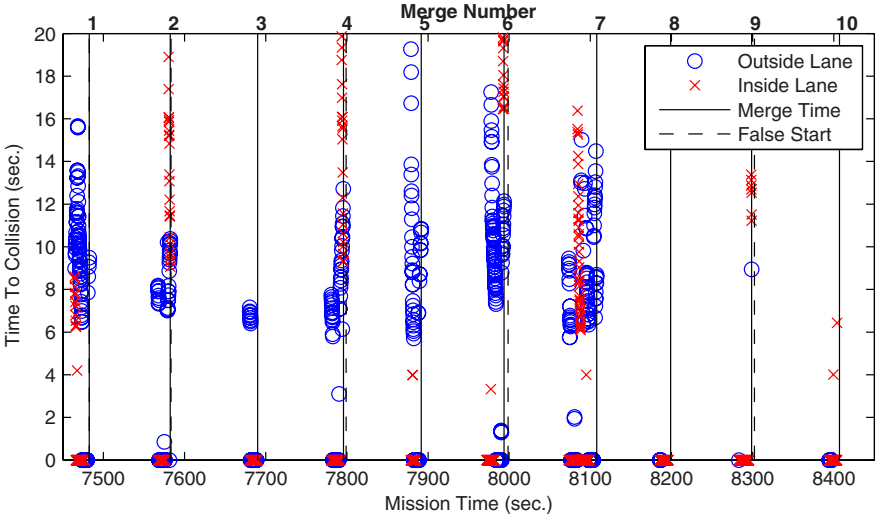


Fig. 7. Perceived times to collision with oncoming vehicles in the outside and inside lanes of traffic in Team Cornell's second run of NQE Area A. Solid vertical lines indicate decisions to merge. Dashed vertical lines indicate false starts: times when the planner thinks it is safe to merge and then changes its mind.

Adjustments resulting from the first attempt at Area A made the second attempt more successful. Figure 7 plots the perceived times to collision from oncoming traffic in both lanes in Team Cornell's second run of Area A. Skynet's 10 selected merge times are plotted as solid vertical lines, and times at which Skynet changed its mind after deciding to merge are plotted as dashed vertical lines. Note that although the local map and track generator often track several cars deep in a line of vehicles, only collision times to the closest vehicles in each lane are plotted. These closest vehicles are those used by the planner to make merge decisions; other tracked vehicles are ignored. Only times at which Skynet is waiting to merge at the stop line are plotted. Times at which the intersection and the 30 m 'safety zone' surrounding it are physically occupied by an obstacle are plotted as zero time to collision. Skynet will not attempt a merge when these areas are occupied, unless tracked obstacles in these areas are reliably determined to be stationary. Data is plotted at approximately 10 Hz, once per planning cycle.

Phantom obstacles did not impede vehicle progress in Team Cornell's second attempt at Area A as they did in the first attempt: there are no large periods of time in which the intersection is occupied. Notice in four of these laps Skynet made false starts, deciding it was safe to merge and then changing its mind. In two of these cases, merges 2 and 6, the tactical layer decided the oncoming vehicles were threatening enough to postpone the merge. These two cases were subsequently resolved correctly, resulting in successful merges. In the other two, merges 4 and 9, Skynet pulled into the intersection before deciding the oncoming vehicles posed a threat. As it pulled into the intersection, it received more information: improved obstacle speed estimates, and therefore more accurate times to collision. In these cases the operational layer could not complete the merge fast enough, and the tactical layer stopped Skynet when it detected approaching obstacles. Neither failed merge was dangerous, though both were undesirable: oncoming traffic stopped, and Skynet continued when it sensed no obstacles approaching. An improved turn reasoning test was added after this test to check whether Skynet had reached a point of no return at the intersection, where it would be more dangerous to stop than to keep going, based on time to collision.

Figure 8 shows a magnified view of collision times in the first merge of Team Cornell's second run in Area A. Each obstacle tracked through the intersection has a smoothly decreasing or increasing time to collision, indicating accurate obstacle speed estimates provided by the local map and track generator. Occasional increasing collision times reflect uncertainty as to which lane the tracked vehicles occupy. In cases of high uncertainty, the tactical layer conservatively places uncertain vehicles in oncoming lanes and calculates collision times as if they were approaching. These uncertain vehicles are combined with the pool of vehicles with certain lane estimates, and the closest vehicle in each lane is used to evaluate the feasibility of a merge. In the case of Figure 8, the tactical layer attempted a merge when the intersection was free for 10 seconds, which occurred near time stamp 7482.

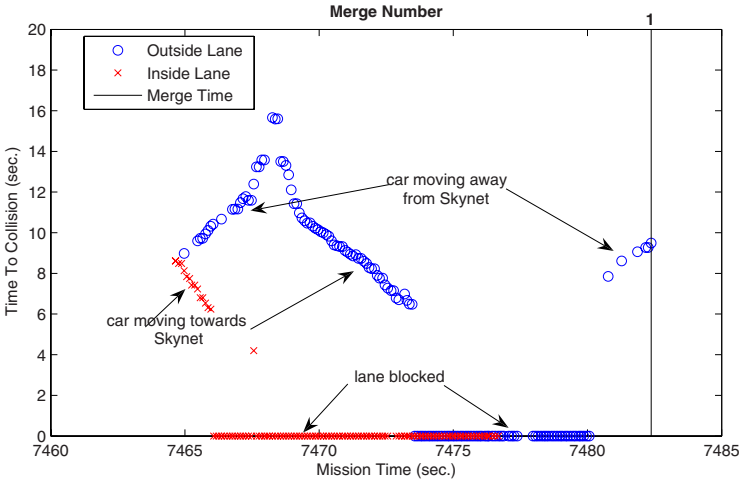


Fig. 8. Perceived times to collision in the first merge of Team Cornell's second run of NQE Area A. Here Skynet merges when the nearest obstacle is 10 sec. away.

4.2 Area B

In Area B, autonomous agents were required to navigate a lengthy route through the urban environment, leaving from a start chute near the grand stands and returning to the mission finish after navigating the course. The environment was devoid of moving traffic, but contained a zone with spaces to test parking. A portion of the street was also filled with orange barrels in the center of the street and cars parked at the side of each lane to test obstacle avoidance capabilities. Vehicles also encountered a number of empty intersections, and were required to remain in the appropriate lane at all times.

Team Cornell's vehicle ran in Area B twice, failing at the first attempt and succeeding at the second. Skynet failed the first attempt at Area B while trying to park. It successfully aligned itself to the parking space, but refused to pull into the space: other unoccupied cars parked nearby violated admissible obstacle spacing constraints in the operational layer. After sitting for several minutes, Skynet was manually stopped, positioned inside the parking space, and allowed to continue. Skynet proceeded as normal, entering the portion of the course testing obstacle avoidance. There Skynet passed several parked cars and orange barrels, until coming upon a section of road with a parked car in either lane. Believing both lanes to be blocked, Skynet began a U-turn maneuver, but ran out of time before completing the course. All behavior until the parking difficulty was normal, and Skynet navigated without incident.

Behavior in the first attempt at Area B prompted the team to make adjustments to spacing constraints in the optimization problem presented in section 3.5.3 and to the tactical layer. In particular, minimum spacing

constraints in the tactical layer were reduced from 0.9 m to 0.3 m for stopped obstacles, and from 1.4 m to 0.9 m for moving obstacles. These adjustments allowed Skynet to park in parking spaces closely surrounded by obstacles. They also made Skynet less sensitive to concrete barriers and other stationary obstacles at the sides of the lanes. Figure 9 (top) plots the sensed spacing between Skynet and other obstacles in a portion of the Area B course lined with parked cars: 10 in Skynet's lane, 4 in the opposing lane, and 1 group of 4 orange barrels clustered in the center of the road. The lane width in this portion of the course was marked in the RNDF as 12 ft (3.6576 m). Changes to spacing constraints after the first attempt at Area B allowed Skynet to navigate this portion of the course without stopping, as shown in Figure 9 (bottom). Notice that to achieve these spacings, however, Skynet at times crossed the center line of the road by up to 1 m. Such behavior is heavily penalized at the operational layer, and only occurred when the obstacle constraints mandated it.

Navigation in the remainder of Area B proceeded without incident. Skynet did, however, receive several spurious disabling commands over the DARPA-provided emergency stop wireless interface. The source of these commands is still unknown, though DARPA technical staff concluded that they came from an external source. Skynet was allowed to be restarted after the first of these spurious signals, though it was eventually removed from the course as more emergency stop signals were received. When removed, Skynet was approximately 60 m from the final checkpoint of the mission.

4.3 Area C

In Area C, autonomous agents were required to navigate a short loop with two four-way intersections. The intersections were populated with an increasing number of traffic vehicles to test various configurations of intersection queuing. If a vehicle successfully negotiated all intersection scenarios correctly, road blocks were placed across the road to force the vehicle to plan a new route to complete its mission. These road blocks were left in place for the remainder of the mission to ensure that vehicles remembered previously encountered road blocks when planning new routes.

Team Cornell's vehicle ran in Area C only once, completing the entire course without incident. Figure 10 shows output of the posterior pose and track generator algorithms at one of the most difficult intersection scenarios. In this scenario, human-driven vehicles populated each entrance to the intersection. Each of these vehicles had precedence over Skynet, and, when turning, passed in front of Skynet to block the remaining vehicles from view. This scenario exercised Skynet's occlusion reasoning, which was invoked each time one vehicle passed in front of the others. One example is given in Figure 10, which shows Skynet waiting at the northern entrance to the intersection. A second vehicle, which has precedence over Skynet, entered the intersection from the West. As it passed in front of the vehicle sitting at the southern

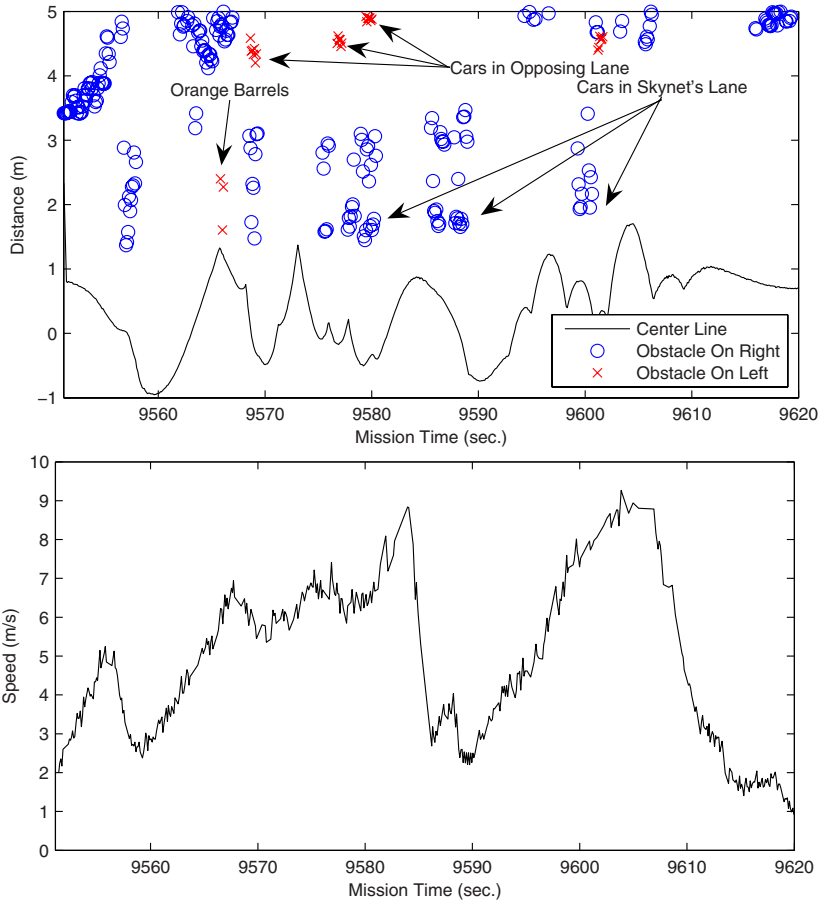


Fig. 9. (top): Spacing between Team Cornell's vehicle and stationary obstacles during a portion of the second run of NQE Area B. (bottom): Vehicle speed through the same portion of Area B.

entrance, occlusion reasoning marked that vehicle as occluded. The planner held the occluded vehicle's place in the intersection queue, and the vehicle was tracked correctly through the entire intersection scenario. The vehicle waiting at the eastern entrance was similarly marked as occluded when it was blocked from view, and successfully tracked through the scenario as well.

In completing Area C, Team Cornell's vehicle correctly solved 7 intersections: one empty, one with 1 other vehicle, three with 2 other vehicles, one with 3 other vehicles, and one with 4 other vehicles. Of these scenarios, 3 resulted in occlusions; each occlusion was correctly handled by obstacle tracking and occlusion reasoning in the local map and track generator, respectively.

After the intersection scenarios, Skynet was forced to plan around road blocks. Figure 11 shows the different steps in Skynet's reasoning while

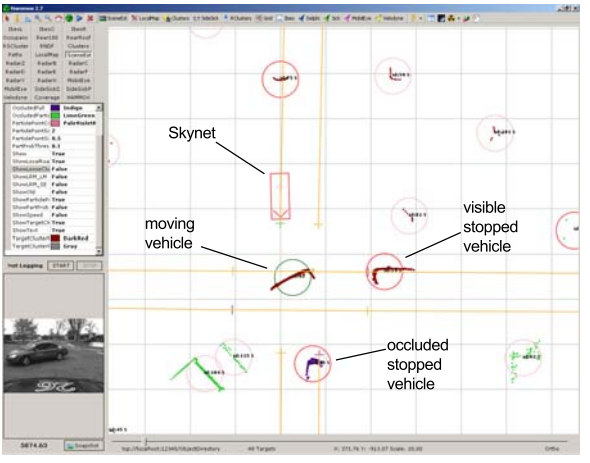


Fig. 10. Occlusion reasoning at a sample intersection configuration in NQE Area C allows Skynet to remember vehicles at the intersection even when blocked from view. Here Skynet waits at the North entrance to the intersection, the topmost in the screen.

handling these road blocks. First, Figure 11 (top) shows Skynet traveling along its initial path, the shortest path to the next mission checkpoint. This path took Skynet down the southern route of Area C, where it encountered the first road block (8 orange barrels blocking both lanes), shown in Figure 11 (middle left). Figure 11 (middle right) shows Skynet's path around the blockage, which took it through the central route of Area C. Halfway down that new path, Skynet discovered a second road block (a horizontal metal bar with 6 stop signs blocking both lanes), shown in Figure 11 (lower left). Figure 11 (lower right) shows the path planned around the second road block, which took Skynet around the northern path of Area C. In planning this third path, Skynet remembered the locations of both the road blocks it had previously encountered. These blockages heavily penalized the central and southern routes of Area C, as described in section 3.5.1. This made the northernmost path least expensive, even though it was longest in length.

5 Performance at the Urban Challenge Event

The Urban Challenge Event (UCE) was held in Victorville, California, USA on November 3, 2007. Of the 35 teams originally invited to participate in the NQE, only 11 were invited to continue on to the UCE. The UCE course, shown in Figure 12, subsumed NQE Areas A and B and roads connecting them. Most roads were paved with a single lane in each direction, as they belonged to a former residential development. Several roads admitted two lanes of traffic in each direction. One road, in the southeastern corner of the network, was a dirt road that descended a steep gradient.

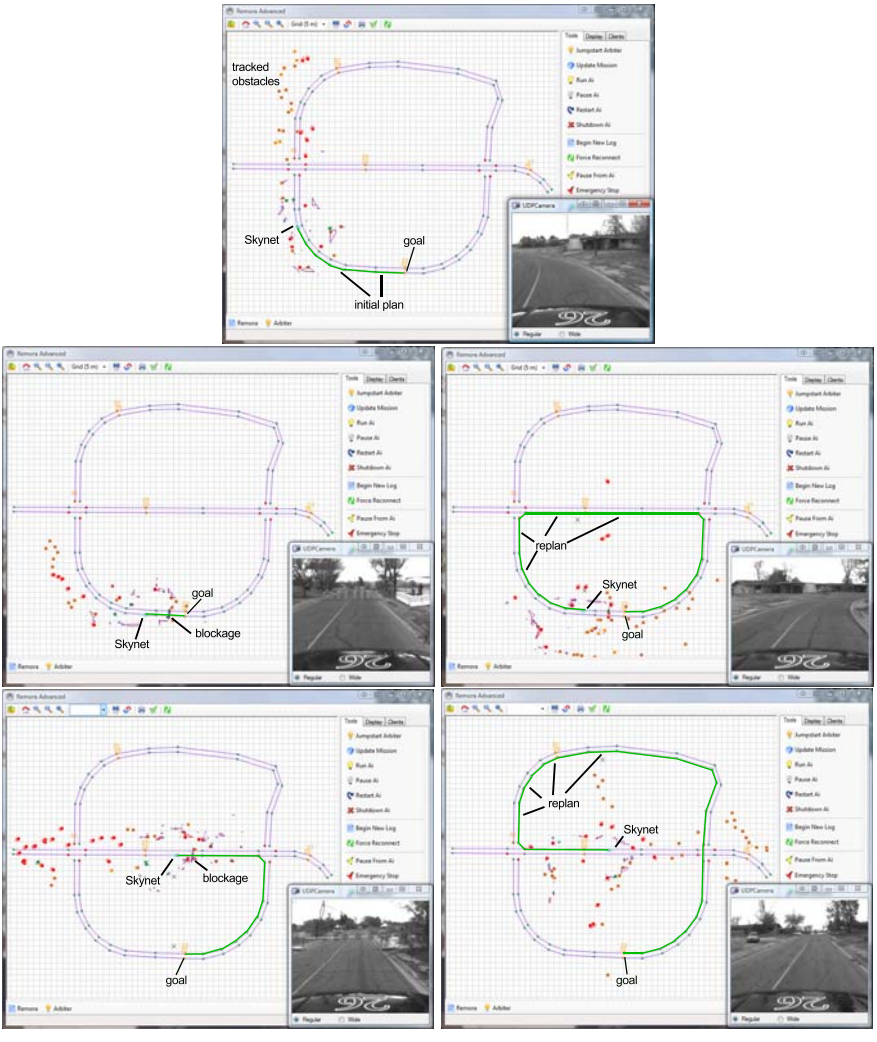


Fig. 11. Blockage recovery behaviors in NQE Area C. (top): The initial planned path. (middle left): A road block forces Skynet to plan a new path. (middle right): Skynet follows its new plan. (lower left): Skynet encounters a second blockage. (lower right): Skynet plans a third path, remembering the positions of both blockages.

Each vehicle competing in the UCE was required to complete 3 missions, defined by separate mission definition files. Each mission began among an array of start chutes in the western end of the road network. The first 2 missions given to Team Cornell were each subdivided into 6 'sub-missions,' which required Skynet to achieve a series of checkpoints before returning to perform a loop in the traffic circle at the western end of the road network. The

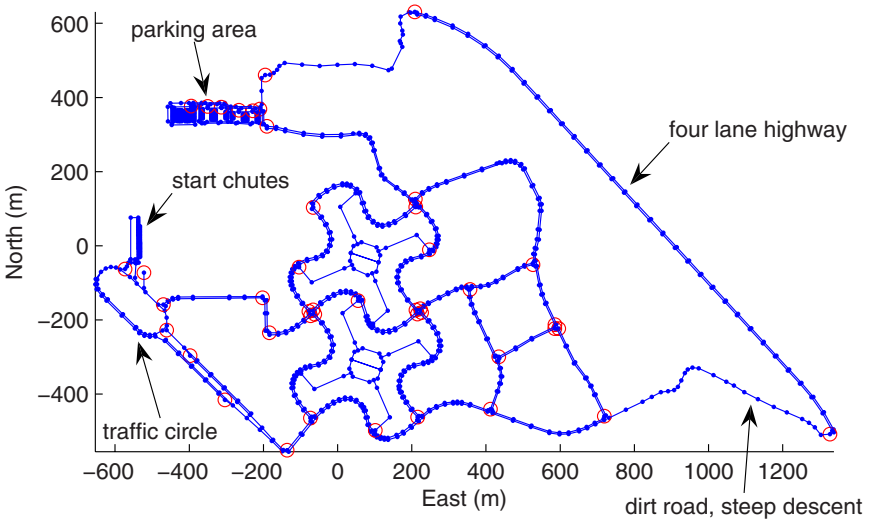


Fig. 12. The Urban Challenge road network. Waypoints are represented by small dark solid dots, with traversable lanes and zone boundaries represented as lines connecting them. Stop lines are represented by large circles.

final mission for Team Cornell was subdivided into 7 such sub-missions. Each mission ended with Skynet returning to a finishing area near the start chutes, where the team could recover Skynet and reposition it for the next mission. The missions given to Team Cornell totaled approximately 55 miles of driving on the shortest path. All 11 qualifying robots were allowed to interact in the UCE course simultaneously, with additional traffic supplied by human-driven Ford Tauruses. As in the NQE, safety and sensible behavior were paramount, though no official judging criteria or scores were made public for the UCE.

Team Cornell's performance in the UCE is presented in 3 sections. Section 5.1 gives Skynet's general behavior and performance in the UCE. Section 5.2 presents a small case study of another unique event, where Skynet encountered a human-driven Ford Taurus driving the wrong way on a one way road. A second event, where Skynet correctly tracked two slow-moving vehicles and passed them, is presented in section 5.3. Section 5.4 presents a small case study of a third unique event occurring near the corner of Washington St. and Utah St., where Team Cornell's Skynet waited for approximately 12 minutes on the wrong side of the road. A fourth event, a minor collision with the MIT robot, is left as the topic of a companion paper.

5.1 Overall UCE Performance

Team Cornell's Skynet was one of only 6 vehicles to complete the Urban Challenge; the other 5 vehicles were disqualified at various points in their

respective first missions. Skynet completed the Urban Challenge in $5h55m31s$, with a total of approximately 55 miles of autonomous driving.

Although Skynet's overall performance in the UCE was successful, several poor behaviors were displayed repeatedly. First, a low-level electrical grounding problem in the throttle actuation system developed over the course of the UCE, forcing Skynet to travel much slower in the final mission of the UCE than in the first two. Figure 13 (left) shows evidence of the problem in the final mission of the UCE. In the final mission, 33.9% of all commands received at Skynet's engine control unit (ECU) exactly matched those issued by the operational layer. In contrast, 64.4% of the commands issued were larger than those received at the ECU, and only 1.7% were smaller. Figure 13 (right) shows the same plot for the first mission of the UCE: only 24.5% of the commands were larger than those received, 36.42% were smaller, and 39% were equal. The obvious disparity in commanded and received throttle positions in the last mission prevented Skynet from achieving its commanded speed most of the time, resulting in significantly slower progress in the final mission of the UCE. The problem is most apparent in mission completion times: Team Cornell spent $1h51m39s$ on the first mission, $1h18m40s$ on the second, and $2h45m12s$ on the third. In simulations without any traffic, Skynet finished the missions in $1h15m$, $1h17m$, and $1h40m$, respectively. Note the timing in the second mission: Skynet completed the mission only 2 minutes slower than it could in simulation. The disparity in the completion times of the first mission, due to the Washington and Utah event, is explained in section 5.4.

Ultimately, Skynet's low-level grounding problem owes to an oversight in circuit design at the actuation level. Had Team Cornell opted for a commercial actuation package, the problem might have been avoided. However, it is unfair to conclude that a commercial package is superior based on this one issue, as commercial options would have undoubtedly presented their own unforseeable growing pains and implementation drawbacks. With an in-house implementation, the team was at least able to modify the actuation scheme as the rest of the system matured. And, all considerations of the Urban Challenge aside, the in-house scheme gave team members a beneficial learning opportunity.

In addition to low-level speed problems, Skynet also occasionally commanded fast stops as a result of perception and tracking errors. Such fast stops, defined as emergency zero-speed commands issued from the tactical layer, occurred 53 times during the UCE. Of these, 24 occurred in the first mission, 17 in the second, and 12 in the third. Because not all of these fast stops are perception errors, it is instructive to look at the location of Skynet at the times of these emergency commands. These locations are represented as black squares in Figure 14 (left). Most of the emergency commands issued by the tactical layer occurred in the southwestern portion of the UCE road network, in areas where concrete barriers lined the wall of the course. These concrete barriers presented significant challenges to the clustering, tracking,

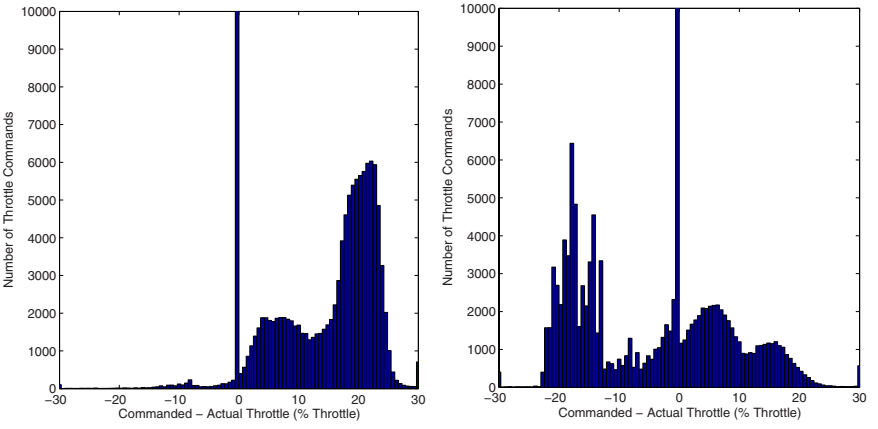


Fig. 13. (left) Histogram of the difference between commanded and received throttle positions in the final mission of the Urban Challenge. A low-level electrical grounding problem resulted in Skynet achieving significantly slower speeds in the final mission of the Urban Challenge. (right) A similar histogram, from the first mission of the Urban Challenge, shows no evidence of the problem.

and path planning algorithms. The full extent of the barriers were often not visible, so the clustering algorithm occasionally clustered the concrete barriers with other obstacles in the lane. Furthermore, measurement assignment mistakes occasionally caused measurements of concrete barriers to be pulled into the lane and applied to other vehicles, causing phantom obstacles to appear in the local map. Finally, the concrete barriers were occasionally close enough to the lane boundary that the operational layer would report a path infeasible. This latter problem was most catastrophic, as it caused Skynet to think the road was blocked.

Figure 14 (right) plots the location of Skynet at each of the 10 times the blockage recovery tactical component was executed: once in the first mission,

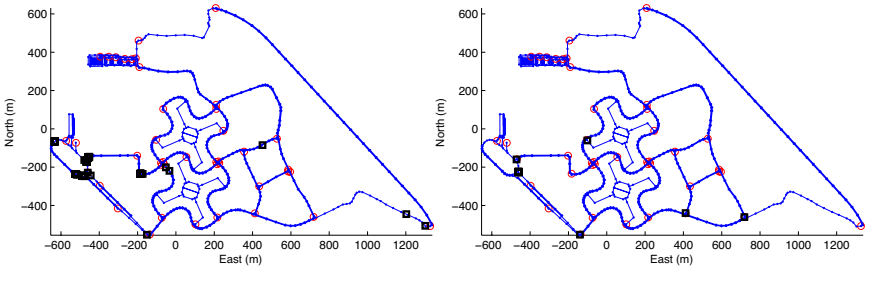


Fig. 14. (left): Location of Skynet (black square) during the 53 emergency brake slams it performed during the Urban Challenge. (right): Location of Skynet (black square) during the 10 times it went into blockage recovery during the Urban Challenge.

6 times in the second, and 3 times in the third. These were the worst behaviors displayed by Skynet, resulting in reverse maneuvers or basic unconstrained obstacle avoidance. Those in the western part of the course were caused by the position of the concrete barriers relative to the DARPA-supplied RNDF waypoints. The remainder were generally caused by perception mistakes: either incorrect clustering or incorrect tracking.

Despite the throttle problem and the emergency commands, Skynet's overall performance was still safe and reliable enough to finish the UCE. The local map and track generator tracked a total of 175252 distinct obstacles during the course of the UCE, with only 53 causing mistakes in Skynet's behavior. The average tracking lifetime of the obstacles was 6.8 seconds. Of these, 26.5% were estimated as being on the RNDF at some point in their lifetime; these obstacles had an average tracking lifetime of 10.7 seconds. Track identification numbers were also very stable during the race, with 13609 tracks lasting for more than 15 seconds. On average, the local map and track generator maintained 48.5 tracked obstacles at each iteration, with a maximum of 209 obstacles tracked in one iteration of the algorithms. The intelligent planner was also similarly stable, despite the 10 mistaken blockage recovery actions. The system operated in the road tactical component 76.1% of the time, in the intersection tactical component 20.8% of the time, in the zone tactical component 2.7% of the time, and in the blockage tactical component 0.5% of the time.

5.2 The Wrong Way Vehicle

One unique event that happened to Skynet over the course of the UCE occurred early in the first mission, where Skynet encountered and properly dealt with a moving vehicle traveling the wrong way on a one-way road. Skynet encountered this vehicle, a human-driven Ford Taurus, on the dirt road at approximately $(1202E, -446N)$ in the coordinate frame of Figure 12. Figure 15 (top) shows the oncoming Taurus from the point of view of the optical cameras and the operational layer. Note the Taurus pulled to the left of the lane as far as possible to minimize the chance of collision.

As the wrong way vehicle moved closer to Skynet, the road tactical component largely ignored it due to assumptions made in the road tactical component. The road tactical component's forward vehicle monitor assumes all vehicles move in their lanes in the proper direction; therefore, only the closest vehicle in front of Skynet is monitored. As a result, the wrong way vehicle was entirely ignored until it was the closest obstacle to Skynet. At that point the road tactical component began to monitor the vehicle, and it would have commanded a fast stop if the vehicle continued to move. Instead, the wrong way vehicle stopped moving. The clustering algorithm subsequently grouped it with nearby bushes and the dirt berm. As a result, the operational layer avoided the wrong way car as a static obstacle, as shown in Figure 15 (bottom). After successfully avoiding the car, Skynet continued with its mission.

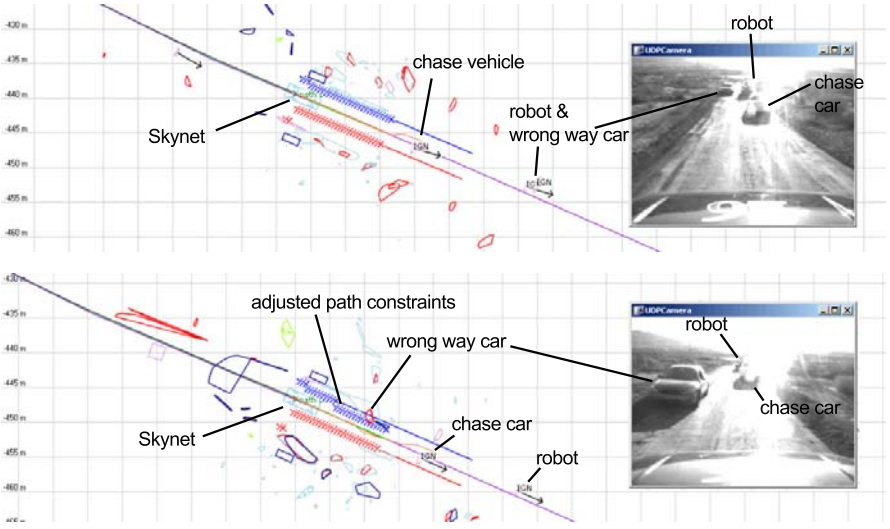


Fig. 15. (top): Team Cornell encounters a human-driven vehicle traveling the wrong way on a one way dirt road. (bottom): The wrong way vehicle is avoided by the operational layer, which adjusts path constraints as if the vehicle were a static obstacle.

This unique incident is one that highlights the capabilities of the local map and constraint-based planner to handle unforseen circumstances: had these systems made more brittle assumptions about what they were tracking or avoiding, this situation could have easily resulted in a collision.

5.3 The Pass

A second unique event that Skynet encountered in the UCE was a successful high-speed pass of a slower moving robot and a human-driven Ford Taurus following it. This event occurred in the first mission, at approximately $(947E, -40N)$ in the coordinate frame of Figure 12, shortly after Skynet encountered the wrong way vehicle. As Skynet traveled along in its lane, it encountered the slow-moving vehicle and trailing human-driven Ford Taurus. Figure 16 shows the encounter.

Vehicles in the desired lane of travel must satisfy two conditions for the road tactical component to execute a passing maneuver. First, the vehicle must be classified as slow-moving by the forward vehicle monitor of the road tactical component. This is defined by satisfying three criteria: the vehicle must be tracked consistently for at least 3 seconds, the vehicle must not be in an intersection safety zone, the vehicle must be traveling slower than Skynet, and the vehicle must be traveling at less than 70% of the maximum speed allowed on that particular road. Second, the road tactical component must determine that there is enough time to pass the vehicle, a calculation

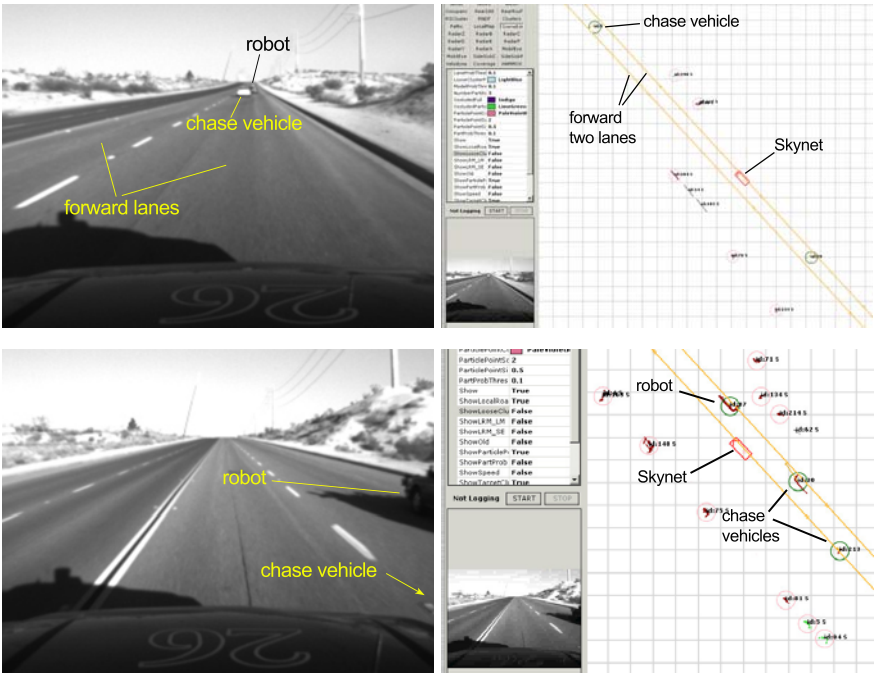


Fig. 16. (top): Team Cornell approaches two slow-moving vehicles. (top right): The tracked vehicles are determined to be slow-moving, and Skynet decides a passing maneuver is feasible. (bottom left): Skynet passes both vehicles at 30 mph. (bottom right): Skynet tracks both vehicles as it passes them.

based on the distance remaining before the next mission checkpoint and the difference in speed between the tracked vehicle and Skynet. In this situation, both these conditions were met. Skynet correctly tracked the trailing vehicle at 118 m, determining its speed to be 7.1 m/s compared to Skynet's 13.7 m/s. The vehicle was considered slow-moving at a time when the next checkpoint was 1070 m away, which was sufficient time to complete a passing maneuver. Figure 16 (top left) shows the vehicles' configuration when Skynet decided to pass. Figure 16 (top left) shows the track generator tracking the trailing vehicle just prior to beginning the pass.

After checking for other fast-moving vehicles approaching from the rear and other slow-moving vehicles in the adjacent lane, Skynet deemed the passing maneuver safe. As Skynet moved to the adjacent lane, the forward vehicle dropped from Skynet's front radar detection cone. This caused a measurement assignment mistake, and a second duplicate track was created near the original vehicle. This duplicate appeared near the lane boundary, so Skynet slowed down to match its speed. The local map and track generator resolved the mistake after several seconds, allowing Skynet to speed up to pass.

Fig. 17. (top left): Skynet approaches the traffic jam at the corner of Washington and Utah. (top right): Skynet begins to pass a stopped vehicle deemed disabled. (middle left): Skynet continues its pass, but gets trapped in the wrong lane as its gap closes. (middle right): Other vehicles pass Skynet as it waits for the intersection to clear. (bottom left): The traffic jam clears. (bottom right): Skynet pulls to the stop line to continue the mission.

As Skynet passed the first slow-moving vehicle, shown in Figure 16 (bottom left), it also tracked the slow moving robot in front. Figure 16 (bottom right) shows the output of the track generator at one iteration, with tracks assigned to both the slow-moving robot being passed by Skynet and to the slow-moving Taurus that was just passed. Unable to change back into the right lane while the slow-moving robot occupied it, Skynet continued to drive in the passing lane. Once clear of both slow-moving vehicles, Skynet sensed no obstructions

in the right lane, either in front or to its side. Skynet therefore pulled back into the right lane, to complete the passing maneuver and continue with its mission.

The passing maneuver executed by Skynet illustrates the complex dependencies among all its constituent subsystems. The actuation provided real-time feedback necessary to complete the maneuver smoothly. The pose estimator and posterior pose produced a smooth and reliable solution that allowed precision path tracking. The local map and track generator tracked all the slow-moving vehicles with consistent speed estimates, even as the vehicles passed from one sensor to another. The road tactical utilized all information from the sensing to reason about whether a pass could be completed successfully, and the operational layer executed the desired maneuver. The passing event was one significant instance among many where all components of Skynet functioned together correctly, and is an accurate representation of Skynet's performance in the UCE.

5.4 The Traffic Jam at Washington and Utah

A final unique event for Team Cornell occurred during a traffic jam near the corner of Washington St. and Utah St., at approximately $(-41E, -208N)$ in the coordinate frame of Figure 12. As Skynet approached the intersection, it encountered a traffic jam: a robot was disabled at the stop line. In line behind the vehicle were two human-driven Ford Tauruses, one following the robot, the other acting as traffic. Skynet pulled to a stop behind the last of these three vehicles, as shown in Figure 17 (top left).

The last of the vehicles, a human-driven Taurus, left a large space between it and the other two vehicles. Unfortunately, this large space was farther than the 30 m DARPA-specified safety zone that surrounds each intersection. Because the vehicle was outside the safety zone, Skynet was allowed to consider it disabled and pass. After waiting 10 seconds, Skynet did exactly that: concluded the Taurus was disabled, and began to pass it.

Shortly after Skynet began to pass, it observed another robot turning into the oncoming lane, as shown in Figure 17 (top right). The planner immediately executed an emergency brake command, one of the 53 mentioned in section 5.1. The planner prevented Skynet from moving while the oncoming vehicle passed, and a collision was avoided. A human-driven Taurus following the oncoming robot also turned into Skynet's lane, but immediately drove onto the sidewalk in order to avoid potential collisions. With the lane clear, Skynet continued its passing maneuver of the original Taurus, which still had not moved. As Skynet pulled fully into the opposing lane, however, the Taurus closed the gap, preventing Skynet from returning to the proper lane to complete the pass.

With nowhere left to go, Skynet concluded that its desired lane was full and began to wait, as shown in Figure 17 (middle left). While waiting, Skynet invoked the blockage recovery tactical component, one of the 10 instances

mentioned in section 5.1. Although the recovery process escalated several levels, including resetting all layers of the planner, Skynet retained the correct view of the situation. During this period a number of other vehicles also passed by without incident, as shown in Figure 17 (middle right). Although Skynet had ample opportunity to navigate the intersection as an unstructured zone, conservative design choices deliberately prevented Skynet from adopting a pure obstacle avoidance strategy so close to an intersection for safety reasons.

Approximately 12 minutes later, the disabled robot resumed its mission, as shown in Figure 17 (bottom left). As soon as the intersection cleared, Skynet pulled into the proper lane, stopped at the stop line, and continued with its mission, as shown in Figure 17 (bottom right). The event at Washington and Utah is unique in the Urban Challenge because no rules were broken: robots were allowed to pass other disabled robots after waiting for 10 seconds, provided they were outside intersection safety zones. Despite following the rules, Skynet still performed undesirably. Such a situation emphasizes the importance of higher level reasoning about other vehicles' behavior that was not incorporated into the Urban Challenge: had Skynet understood the root cause of the traffic jam, it would never have tried to pass in the first place.

6 Conclusions

This paper has presented a high level system-by-system view of Team Cornell's Skynet, one of 6 vehicles to successfully complete the 2007 DARPA Urban Challenge. Skynet consists of several sophisticated subsystems, including in-house actuation, power, and networking design, a tightly-coupled attitude and pose estimator, a multitarget obstacle detection and tracking system fusing multiple sensing modalities, a system to augment the pose solution with computer vision algorithms, an optimization-based local path planner, and a state-based urban reasoning agent. The vehicle built from these components can solve complex merging and intersection scenarios, navigate unstructured obstacle fields, park in designated spaces, and obey basic traffic laws.

The success of Team Cornell's Skynet was largely based on lessons learned in the 2005 Grand Challenge (Miller et al., 2006). In particular, the team opted for standard automotive components, while still designing the actuation and power distribution systems in-house. In addition, the team learned to be wary of GPS, designing a more robust tightly-coupled pose estimator than the loosely-coupled version used in the Grand Challenge. Mistrust of GPS signals also caused the team to build both an obstacle tracking system and an obstacle avoidance system that operated in a vehicle-fixed coordinate frame, completely independent of GPS. The instability of greedy search spline-based planners motivated Team Cornell to adopt a local path planner that selected paths based on physical constraints in the real world, including actuator constraints, obstacle avoidance constraints, and preferences for

smoother, straighter paths. These lessons led to Team Cornell's success in the Urban Challenge.

Though successful, Team Cornell exposed several critical areas for continued investigation. In particular, modeling and estimation of other vehicles' behavior would have vastly improved Skynet's performance in the Washington and Utah traffic jam, and would have allowed more sophisticated reasoning to take place elsewhere. Further modeling of the environment itself, including a GPS-independent local road model, would also aid in Skynet's situational awareness in cases where the road is poorly-surveyed. Finally, the Urban Challenge has exposed the need for a better understanding of the interplay between probabilistic sensor information and robust planning. The state-based planner used on Skynet was stable, but only after many months of careful tuning. Each of these research areas remains largely unexplored, though progress will be necessary to permit improvements in autonomous driving.

Acknowledgments

The authors would like to acknowledge additional members of the Cornell DARPA Urban Challenge Team: Filip Chelarescu, Daniel Pollack, Dr. Mark Psiaki, Max Rietmann, Dr. Bart Selman, Adam Shapiro, Philipp Unterbrunner, and Jason Wong.

This work is supported by the DARPA Urban Challenge program (contract no. HR0011-06-C-0147), with Dr. Norman Whitaker as Program Manager.

References

Artes, F., Nastro, L.: Applanix POS LV 200: Generating continuous position accuracy in the absence of GPS reception. Technical report, Applanix (2005)

Arulampalam, M., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. IEEE Transactions on Signal Processing 50(2), 174–188 (2002)

Bar-Shalom, Y., Rong Li, X., Kirubarajan, T.: Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software. John Wiley & Sons, Inc., New York (2001)

Bierman, G.: Factorization Methods for Discrete Sequential Estimation. Academic Press, New York (1977)

Centa, G.: Motor Vehicle Dynamics: Modeling and Simulation. World Scientific, Singapore (1997)

Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press, Cambridge (2003)

Davis, J., Herring, T., Shapiro, I., Rogers, A., Elgered, G.: Geodesy by radio interferometry: Effects of atmospheric modeling errors on estimates of baseline length. Radio Science 20(6), 1593–1607 (1985)

Felzenszwalb, P., Huttenlocher, D.: Efficient graph-based image segmentation. International Journal of Computer Vision 59(2), 167–181 (2004)

Ferguson, D., Stentz, A., Thrun, S.: Pao* for planning with hidden state. In: Proceedings of the 2004 International Conference on Robotics and Automation, vol. 3, pp. 2840–2847 (2004)

Gillespie, T.: Fundamentals of Vehicle Dynamics. Society of Automotive Engineers, Inc., Warrendale, Pennsylvania (1992)
Julier, S., Uhlmann, J.: A new extension of the Kalman filter to nonlinear systems. In: Proceedings of the SPIE: Signal Processing, Sensor Fusion, and Target Recognition VI, vol. 3068, pp. 182–193 (1997)
Lemoine, F., Kenyon, S., Factor, J., Trimmer, R., Pavlis, N., Chinn, D., Cox, C., Klosko, S., Luthcke, S., Torrence, M., Wang, Y., Williamson, R., Pavlis, E., Rapp, R., Olson, T.: The development of the joint NASA GSFC and NIMA geopotential model EGM96. Technical Report NASA/TP-1998-206861, NASA Goddard Space Flight Center (1998)
Lundberg, J., Schutz, B.: Recursion formulas of legendre functions for use with nonsingular geopotential models. Journal of Guidance, Control, and Dynamics 11(21), 31–38 (1988)
Martin, M., Moravec, H.: Robot evidence grids. Technical Report CMU-RI-TR-96-06, The Robotics Institute, Carnegie Mellon University, Pittsburgh (1996)
Miller, I., Campbell, M.: A mixture-model based algorithm for real-time terrain estimation. Journal of Field Robotics 23(9), 755–775 (2006)
Miller, I., Campbell, M.: Rao-blackwellized particle filtering for mapping dynamic environments. In: Proceedings of the 2007 International Conference on Robotics and Automation, pp. 3862–3869 (2007)
Miller, I., Campbell, M.: Particle filtering for map-aided localization in sparse gps environments. In: Proceedings of the 2008 International Conference on Robotics and Automation (2008)
Miller, I., Lupashin, S., Zych, N., Moran, P., Schimpf, B., Nathan, A., Garcia, E.: Cornell University's 2005 DARPA Grand Challenge Entry. Journal of Field Robotics 23(8), 625–652 (2006)
Psiaki, M., Mohiudden, S.: Modeling, analysis and simulation of GPS carrier phase for spacecraft relative navigation. Journal of Guidance, Control, and Dynamics 30(6), 1628–1639 (2007)
Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Pearson Education, Inc, Upper Saddle River (2003)
Saastamoinen, J.: Atmospheric correction for the troposphere and stratosphere in radio ranging of satellites. In: Henriksen, S., Mancini, A., Chovitz, B. (eds.) Geophysical Monograph Series, vol. 15, pp. 247–251 (1972)
Savage, P.: Strapdown inertial navigation integration algorithm design part 1: Attitude algorithms. Journal of Guidance, Control, and Dynamics 21(1), 19–28 (1998a)
Savage, P.: Strapdown inertial navigation integration algorithm design part 2: Velocity and position algorithms. Journal of Guidance, Control, and Dynamics 21(2), 208–221 (1998b)
Sleewaegen, J.-M., De Wilde, W., Hollreiser, M. (2004). Galileo Alt-BOC receiver. In: Proceedings of GNSS 2004 (2004)
Sukthankar, R.: Situational Awareness for Tactical Driving. PhD thesis, The Robotics Institute, Carnegie Mellon University (1997)
Trimble, AgGPS 252 Receiver User Guide Version 1.00, Revision A (2004)
Willemsen, P., Kearney, J.K., Wang, H.: Ribbon networks for modeling navigable paths of autonomous agents in virtual urban environments. In: Proceedings of IEEE Virtual Reality 2003, pp. 22–26 (2003)
Wong, J.: Theory of Ground Vehicles, 3rd edn. John Wiley & Sons, Inc., New York (2001)

A Practical Approach to Robotic Design for the DARPA Urban Challenge

Benjamin J. Patz[1], Yiannis Papelis[2], Remo Pillat[3], Gary Stein[3], and Don Harper[3]

[1] Coleman Technologies, Inc.
Orlando, FL 32801
bpatz@ctiusa.com
[2] Virginia Modeling Analysis & Simulation Center
Old Dominion University
ypapelis@odu.edu
[3] College of Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
rpillat@cs.ucf.edu,gstein@mail.ucf.edu,harper@cs.ucf.edu

Abstract. This article presents a practical approach to engineering a robot to effectively navigate in an urban environment. Inherent in this approach is the use of relatively simple sensors, actuators, and processors to generate robot vision, intelligence and planning. Sensor data is fused from multiple low cost 2-D laser scanners with an innovative rotational mount to provide 3-D coverage with image processing using both range and intensity data. Information is combined with Doppler radar returns to yield a world view processed by a Context-Based Reasoning control system to yield tactical mission commands forwarded to traditional PID control loops. As an example of simplicity and robustness, steering control successfully utilized a relatively simple follow-the-carrot guidance approach that has been successfully demonstrated at speeds of 60 mph. The approach yielded a robot that reached the finals of the Urban Challenge and completed approximately two hours of the event before being forced to withdraw as a result of a GPS data failure.

1 Introduction

The Urban Challenge is the third in a series of competitions launched by the Defense Advanced Research Projects Agency (DARPA) with the goal of developing technology to keep warfighters off the battlefield and out of harm's way. The specific objective of the Urban Challenge was to develop a robot capable of autonomously navigating a typical urban environment at speeds up to 30 mph. Urban scenarios involved other manned vehicles as well as robots traversing the same course at the same time and resulted in robot-on-robot autonomous decision making challenges. The final event of the competition took place in Victorville, CA on November 3, 2007, but this event was a culmination of 18 months of work and numerous other formal qualification procedures. TeamUCF and the Knight Rider robot successfully passed these qualification procedures to make it to the finals of the Urban Challenge.

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 305–358.
springerlink.com

Fig. 1. NQE Site in Victorville Consisted of Several Test Areas

1.1 Urban Challenge Overview

The Urban Challenge program was announced in May of 2006 and proposals from interested teams were solicited shortly thereafter. Successful proposals were divided into two categories: 11 Track A teams received $1M in supporting funding from DARPA while 78 Track B teams were on their own. TeamUCF was a Track B team. Track A teams had to meet several programmatic milestones, but any team advancing in the competition had to submit a comprehensive technical report and pass an on-site visit by DARPA. Site visits were conducted in June and July of 2007 and 35 semi-finalists were selected in August. Semi-finalists were eligible to participate in the National Qualifying Event (NQE) in Victorville, CA (Figure 1) during the last two weeks of October 2007. The NQE consisted of a series of rigorous vehicle tests from which 11 finalist teams were selected. These finalists participated in the final event on November 3, 2007, with the winner of the competition announced the following day.

The overall urban driving objective as defined by DARPA was to demonstrate an autonomous robot's ability to complete a series of driving missions in traffic, over the course of 6 hours, while obeying California driving rules, utilizing a moderate level of a-priori information associated with the road network, but being expected to deduce any missing information. Some key observations can be made with respect to this definition; some of which were clearly specified in DARPA provided rules, while others simply became apparent over the stages leading up to the final event:

- Robots were limited to street legal motor vehicles, modified for autonomous operation.
- Autonomous operation meant no real-time interaction with the robot except for a remote control safety (E-Stop) system which could pause or completely disable the robot.
- Urban environment consisted of typical US streets with one-way and two-way single lane roads, multi-lane roads, traffic circles, intersections with zero or more stop signs, and parking lots (zones). No stop lights were encountered. Surprisingly a modest amount of off road driving was required in the final event.
- Traffic vehicles meant that following, passing, avoidance, and stop sign precedence behavior was required.
- A-priori information consisted of a Route Network Definition File (RNDF) which defined road segments/lanes, provided by a sparse, but accurate, collection of latitude/longitude "waypoints" and a waypoint to waypoint connectivity graph (although connectivity was not guaranteed since roads could be blocked by design or by accident). The RNDF also provided stop sign locations. Nominally, the RNDF was provided a day or more before any test.
- The driving mission was to traverse a certain set of waypoints (i.e, checkpoints) in a given order as defined in a Mission Definition File (MDF). Nominally, the MDF was provided minutes before a test.

1.2 TeamUCF

TeamUCF's Knight Rider robot was initially conceived over three years ago at the beginning of the 2005 DARPA Grand Challenge event. With the inception of the DARPA Urban Challenge, TeamUCF built on the existing capabilities of the Knight Rider robot (Harper et. al., 2005), augmented as necessary to meet the specific mission objectives of DARPA. The three member team that participated in the 2005 event was expanded only slightly for the 2007 Urban Challenge (5 core team members) with the inclusion of an industry partner, Coleman Technologies, Inc. (CTI). CTI is a system engineering firm specializing in real-time guidance, navigation, and control as well as products associated with GPS measurements in urban environments and CTI provided funding, technical support, and overall team leadership for TeamUCF. TeamUCF was kept small, partly by design, but mainly as a natural result of the team's goal to reuse much of the 2005 robot hardware and a clearly stated objective to implement only those systems necessary to meet stated DARPA Urban Challenge objectives.

1.3 Overall Project Approach

TeamUCF's overall approach to this challenge was to maximize its limited resources. The basic robot control hardware was re-used from the 2005 robot. The sensor suite was an enhanced version of the 2005 robot's sensor suite which

had proved very robust, and which also turned out to be a sensor suite used by many of the participating teams. Three major weaknesses with the 2005 robot were specifically addressed:

- A relatively poor GPS system was replaced with a highly capable RT3000 GPS/INS from Oxford Technical Solutions. This decision was a key to TeamUCF's initial success but confidence in the GPS/INS ultimately led to an unrecoverable failure in the final event.
- A relatively poor simulation model was replaced with a real-time simulator running actual robot software and could operate with full hardware-in-the-loop capability to allow any system element to be real or simulated.
- A more focused pre-event testing strategy was employed which allowed the team to have a clear understanding of the capabilities and limitations of the robot prior to participation in the NQE. This level of knowledge allowed the team to make software changes between tests at the NQE and was perhaps the single most important factor in the team's success.

Resource constraints limited the team's ability to invest in sophisticated 3-D laser scanners, so the team opted for an investment in an innovative rotating laser scanner system designed by one of the authors (Pillat). The limited reliance on sophisticated third-party systems, with capable but inherently un-modifiable software, proved to be a fundamental advantage for TeamUCF.

Simulation modeling allowed all major software elements to be tested prior to robot integration, but there was no substitute for actual robot testing and the team spent as many hours as possible testing the robot hardware. Unfortunately testing a full autonomous automobile-sized robot at speed poses numerous safety issues and TeamUCF was forced to settle on relatively small test areas whose access could be controlled.

The system design approach could be considered a cross between "requirements based" and "capabilities based" in that overall Urban Challenge objectives flowed down to scenarios (Figure 2) and subsequently to overall system level capabilities, but detailed subsystem performance requirements were not derived from these. Rather, since subsystems were effectively selected at the start of the project, the challenge for TeamUCF became one of determining "how" to meet a specific objective with a given system, rather than what system would be best for meeting a particular objective.

In approaching the software functional design, the team followed the following basic principles:

- Safety was of primary importance. The goal was to provide a system that would protect the Knight Rider from collisions and kinematic limits; protect other robots from collision or perceived collision; and protect obstacles.
- Mission completion was of secondary importance. The goal was to complete as much of the provided mission as possible, potentially skipping checkpoints if the robot determined that they were not achievable.

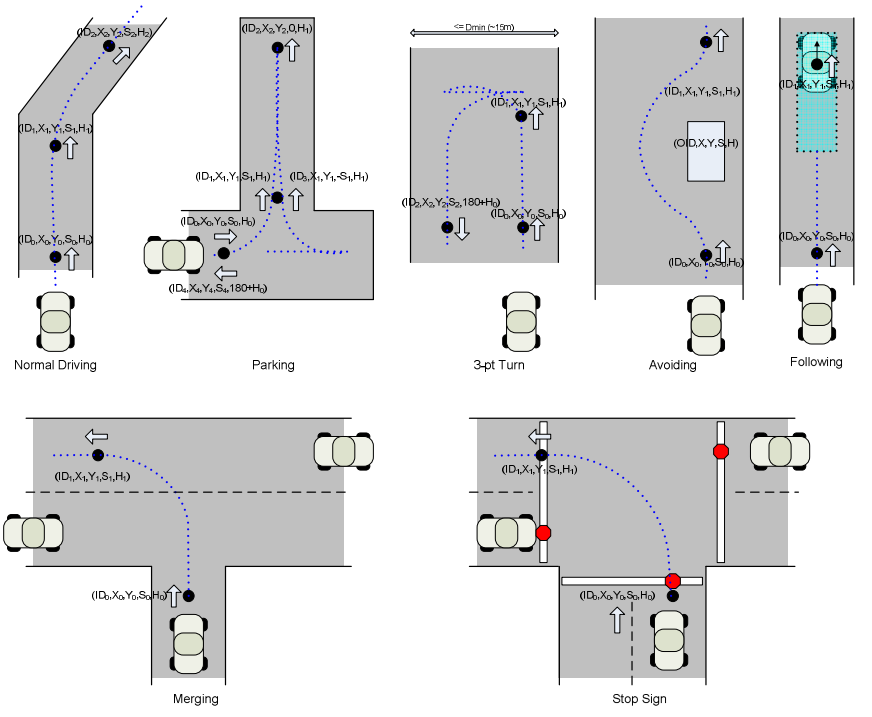


Fig. 2. Basic driving scenarios (not all cases shown)

- Legality was of tertiary importance meaning that the system's software was allowed to violate rules if there was no other way to meet an objective.
- Speed was of least importance. Despite its relative lack of importance, it turned out that the Knight Rider was one of the quickest robots at the NQE.

2 Robot Vehicle

The Knight Rider robot is a 1996 Subaru Outback Legacy (Figure 3) with minimal modifications. Key performance parameters for this robot are provided here:

- 4.8m overall length w/ mounting brackets
- 2.0m overall width w/ mounting brackets
- 2.6m wheelbase
- 5.5m turning radius
- Speed: -2.2 to 13.5 m/s (-5 mph to 30 mph, DARPA restricted)
- Axial Acceleration: ~3.5 m/s^2 (practical limits for comfortable driving)
- Lateral Acceleration: ~2 m/s^2 (practical limits for comfortable driving)

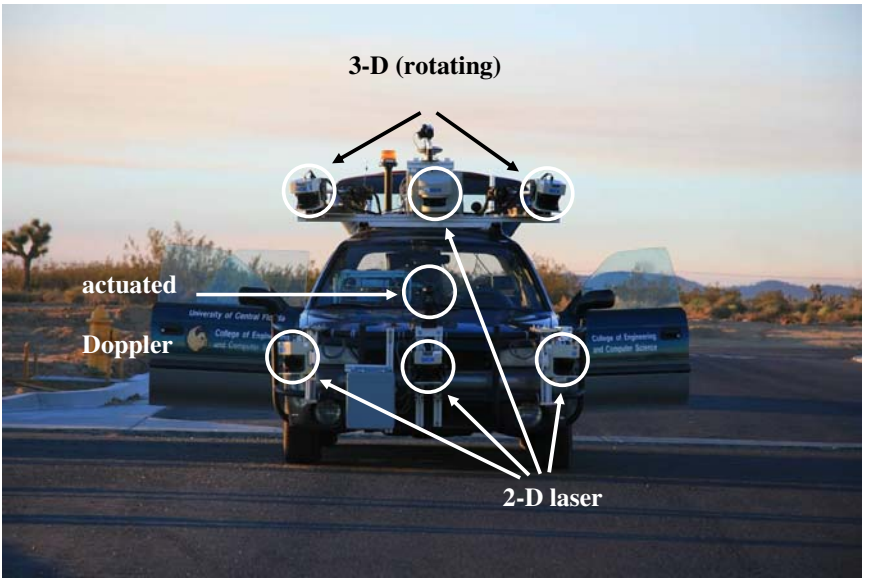


Fig. 3. Knight Rider robot before an early morning test

Adopting this vintage vehicle prevented the use of drive-by-wire or other sophisticated integration into an automobile control system. This limitation turned out to be of no impact to robot performance. Actuators were designed to control existing robot hardware in a manner analogous to a human operator. For example, the steering servo, mounted along the robot centerline, controlled the steering wheel with a belt system similar to the way a driver would control that system. This system easily allowed both robotic operation as well as driver operation. Because of safety concerns, most testing prior to the NQE, was conducted with a driver in the vehicle. Servo torques and belt slippage were adjusted to allow driver override even in the event of full system failure. Throttle and brake actuators were mounted under the passenger seat and similarly provide failsafe operation. In particular, the brake actuator causes the vehicle brake to be depressed and was in the "always on" position via a spring mechanism. The brake was "released" via a pneumatic actuator that, should it fail, would cause the brake to return to the depressed position. Turn signal integration was accomplished via the vehicle's existing wiring infrastructure.

Own-state estimation (position, speed, heading as well as full robot attitude) was provided by a differentially corrected RT3000 GPS/INS from Oxford Technical Solutions. Previous experience indicated that this was an area where commercial systems outperformed developed software. The integrated Inertial Navigation System provides high quality measurements, including obscured-sky speed measurement, in environments where the GPS alone struggles; lateral acceleration in a horizontal direction without the need to zero the accelerometer; and

roll/pitch/yaw measurements which are accurate during continuous turns. GPS/INS data was available to other systems at a 100Hz data rate, although low level control systems only operated at 20Hz and the highest sensor data rate was 35Hz.

SICK LMS291 laser scanners mounted on a forward and rear mounting bracket, and rotating laser scanners mounted to the top rack, provide range, angle, and intensity information to obstacles as small as traffic cones. The sensors provide information only for the leading edge of obstacles, but after multiple looks from varying angles obstacle geometry is refined. Scanner pointing direction and type were selected to optimize forward sector coverage. This approach also provided 100% overlap in coverage directly in front of the robot which proved particularly valuable in the case where a single scanner would lose data principally because of looking directly into the sun. TeamUCF saw no benefit to mounting scanners in the "upside down" position that some teams employed in an attempt to reduce the effect of solar glare.

An actuated Doppler radar (Stalker Speed Sensor) mounted at the front of the robot augmented laser scanner data specifically in long range moving obstacle detection scenarios. This particular sensor employed by TeamUCF provided no effective range or angle information, but rather was limited to return intensity and (signed) speed information. This relatively primitive information, however, proved to be a significant advantage in developing the overall system design since it greatly simplified the decision making logic. Effectively any large object moving sufficiently quickly toward the robot was an obstacle to be avoided.

A Sony HDR-HC3 digital camcorder was mounted on top of the robot and provided a reasonable sensor for lane detection in certain scenarios. Unfortunately early testing at the NQE showed that solar glare caused by early morning and late evening operation, coupled with DARPA's decision to use large concrete k-rail barriers as lane markers in many cases made this video system redundant. For the NQE, the vision system's principal duty was providing a video record of robot performance.

Processing was provided by 3 core-duo computers (mixed Linux and Windows XP) located in a shock mounted frame in the passenger's seat. Intelligence functions were performed on one computer, vision functions on a second, and the third computer provided real-time system control including autopilot and navigation functions. Computers communicated over a local Ethernet network, and various processes establish connection with one another, in a broadcast/subscribe manner, independent of their actual physical processor location. Communication utilized the Internet Communications Engine (ICE) framework which is a simplified derivative of the CORBA architecture.

One principle benefit of the relatively simple system architecture and small number of computers was the relatively low power consumption of the robot. Power consumption was ~600W which included all sensors and processors. No special alternator was used and by choice of mounting location, cooling could be provided directly by the robot's A/C system. Computer and sensor power was provided by four deep cycle marine batteries which were trickle charged by the alternator. DC-DC converters provided appropriate power levels to various sensors.

This system provided stable and clean power and repeatedly demonstrated operation of over 8 hours. Although never required, it was fairly clear that by simply upgrading the alternator, even longer durations could be obtained.

Decomposition of the core software elements is illustrated in Figure 4. For clarity, the detailed interfaces associated with health and status monitoring elements and E-Stop are not shown. Clearly visible are overall mission inputs provided by the RNDF (providing an initial seed of the system's environmental model), and the MDF (defining the overall mission objectives in terms of checkpoints and speed constraints). Viewed as a control system, the elements can be considered as follows: 1) intelligence develops a mission as a set of tactical goals to be achieved, 2) path planning efficiently plans a legal and drivable path to meet those tactical goals, 3) the autopilot maintains the robot on path and within performance limits, and 4) PID controllers command various actuators to meet autopilot commands. Although difficult to see in the diagram, feedback effectively consists of four nested loops. The innermost loop consists of PID controller feedback (actuator position, etc.). The next loop consists of navigation information (position, speed, heading, etc.) used by the autopilot to develop control commands to maintain the robot on course. Beyond this is a path planning loop which effectively manages the tactical path based on tactical goals, bounds, and obstacles. At the outermost level is the overall intelligence loop that monitors whether or not the robot has met its tactical and strategic objectives. This outer loop is closed through vision as well as navigation.

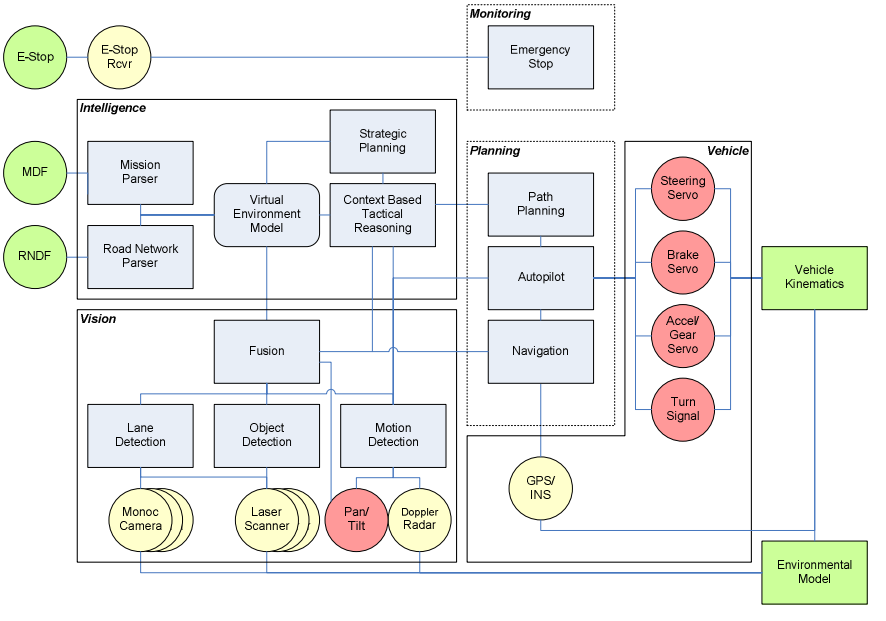


Fig. 4. Overall System Block Diagram

System operation is straightforward. After the robot boots-up and runs an internal self-test, it sits in a wait state ready to accept an RNDF and corresponding MDF. Once files are loaded and successfully processed, the robot remains waiting until released to execute the mission. The detailed mission plan is generated dynamically as the operational environment is discovered. Data logging is performed allowing mission playback for analysis. Upon mission completion, the robot stops.

2.1 Actuators

The robot's actuator system was comprised of four modules, each controlling an existing automotive system. The design of the actuator systems was driven by two overarching principles: to allow for human intervention in any situation and failsafe operation when no safety driver was present. The ability for a human operator to take full control of the robot at any point is indispensable in extensive testing and during most testing prior to the NQE, a safety driver was present in the vehicle. To ease the process of relocating and positioning the robot the actuators were mounted to not interfere with the robot's existing hardware when powered off, allowing a human driver to drive the robot like a normal car. In case the robot is operating fully autonomous with no safety driver, the actuators are constructed to bring the car to a complete stop in the event of a power failure.

The steering controller consisted of a three-phase brushless motor driving a large pulley attached to the existing steering wheel. A six-splined v-belt transferred the torque from the servo motor through a 12:1 mechanical advantage. This small ratio, coupled with the possibility of slip provided by the v-belt allowed a human safety driver to easily overcome the motor during an emergency situation. The belt design also allowed some compliance to help absorb wheel shock due to potholes and other sudden lateral forces imposed on the front tires. The brushless motor was driven by a 12A control line from an Elmo 12/60 Harmonica digital servo controller.

Although our design of the steering system allows the belt around the steering wheel to slip, we never encountered any appreciable slip in testing or operation. This was first and foremost a safety feature, by allowing a human driver to either overpower a servo motor or slip the belt. Small slippages are compensated by the PID steering controller. For these reasons we did not mount an encoder on the steering column to keep track of the actual steering angle.

The throttle controller consisted of a Bowden cable attached at one end to the original cruise control throttle body linkage. The other end was driven by a magnetic linear motor by Linmot. This type of linear motor was chosen because of its natural ability to release when the DC power was removed. This important safety feature allows the existing throttle return spring to force the throttle closed in the event of an emergency stop or other type of power loss.

The brake controller was a two part redundant system that allowed control using a linear motor for normal actuation and a separate pneumatic/spring arrangement for emergency stop situations. The linear motor was a larger version of the throttle motor also by Linmot. The force was transmitted to the brake pedal from behind the firewall using a Bowden cable routed to the actuator located

under the passenger seat. The second half of the braking system was only used in emergency situations when there was either a power loss or a disable E-Stop had occurred. It consisted of a large spring that, in its natural position, constantly applies force to the brake pedal. During normal operation, a pneumatic cylinder provides a countering force that overcomes this spring and allows the brake to be completely controlled by the linear motor. In the event of an emergency, an electric valve opens to release the pneumatic cylinder, forcing the pedal to be depressed by the spring. An air-release valve controls the rate at which the pneumatic cylinder releases which in turn controls the stopping distance.

The emergency braking system was specifically designed for the case of a power loss. The electric valve is a three-way solenoid valve that controls the CO_2 flow to and from the pneumatic cylinder that provides a countering force for the mechanical spring. If power is applied to the valve, CO_2 from a reservoir enters the pneumatic cylinder and overcomes this spring so the brake can be completely controlled by a linear motor. In case of a power failure, the solenoid valve releases the CO_2 from the pneumatic cylinder through its third port, allowing the brake pedal to be depressed by the spring.

The gear shift mechanism utilized yet another linear actuator to provide control over the shifter position. All of the standard gears (P,N,R,1,2,D) could be reached, although normal operation only involved P, R, and D. The existing shift safety interlock was circumvented in this application.

A separate single board computer running the real-time QNX operating system managed each actuator through either 0-10V control voltages or in the case of the steering controller, through a serial port.

2.2 Sensors

Sensor system design was driven by available hardware and proven capabilities, especially from experience gained in the 2005 Grand Challenge. The key requirement of navigation in the Urban Challenge was a safe course traversal in diverse traffic situations. Most scenarios required detection of static or near-static obstacles while the robot was either static or moving slowly (i.e., stop sign scenarios, parking, etc.), but the sensors needed to able to detect and distinguish obstacles at different height levels as well as negative obstacles (potholes). The types of obstacles ranged in size from traffic cones and low curbs to cars, trucks and major road blockages. As demonstrated by DARPA at the NQE, obstacles were not required to have ground contact with driving lanes. These reflections led TeamUCF to employ laser scanners as the main means of acquiring sensory information. These sensors work very well at moderate range (<50m) and for the classes of obstacles encountered in an urban environment.

Perhaps the most challenging scenario in the Urban Challenge was the requirement to merge into high speed (13.5 m/s, ~30 mph) traffic. Considering car axial acceleration capabilities, safe following considerations, and decision timelines this required detection ranges of almost 100m (135m if a true 10 sec gap is to be detected). Range constraints of the laser scanners available to TeamUCF

forced the use of a longer range alternative and TeamUCF employed a Doppler radar to provide the extended range since long range scenarios only involved high speed obstacles.

2.2.1 2-D Laser Scanners

A laser scanner employs emitted laser light and the time-of-flight principle to deduce distances very accurately. 2-D laser scanners (LADARs) that use a rotating mirror to provide angular distance measurements in a plane are relatively inexpensive and widely available, especially through the German manufacturer SICK. The biggest disadvantage of those 2-D laser scanners is that they only provide distance information in one scanning plane and hence only output sparse information about the environment. The usable range of distance measurements is 0.5m – 50m with measurement accuracy in the cm range.

The disadvantage of just one scanning plane can be partially relieved by mounting several 2-D laser scanners in different orientations. This approach has been successfully employed by Stanford's winning robot in the Grand Challenge 2005 (Thrun et. al., 2006). TeamUCF decided to place four SICK LMS291-S05 scanners tactically around the car to allow for a near 360° field-of-view (Figure 5). This enabled the detection of static and dynamic obstacles in many possible locations relative to the car. Individually these scanners provide a 180° scanning range. They provide complete scans at 70 Hz with 1° angular resolution and scans at 35 Hz with 0.5° angular resolution (both frame rates were used at various times, although NQE testing utilized 35Hz frame rates).

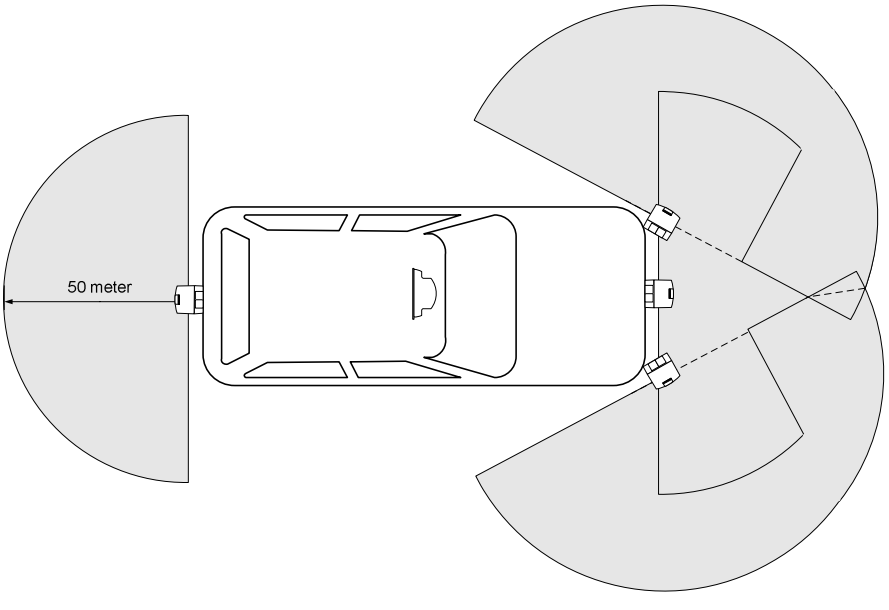


Fig. 5. The mounting points of the 2-D laser scanners, plan view.

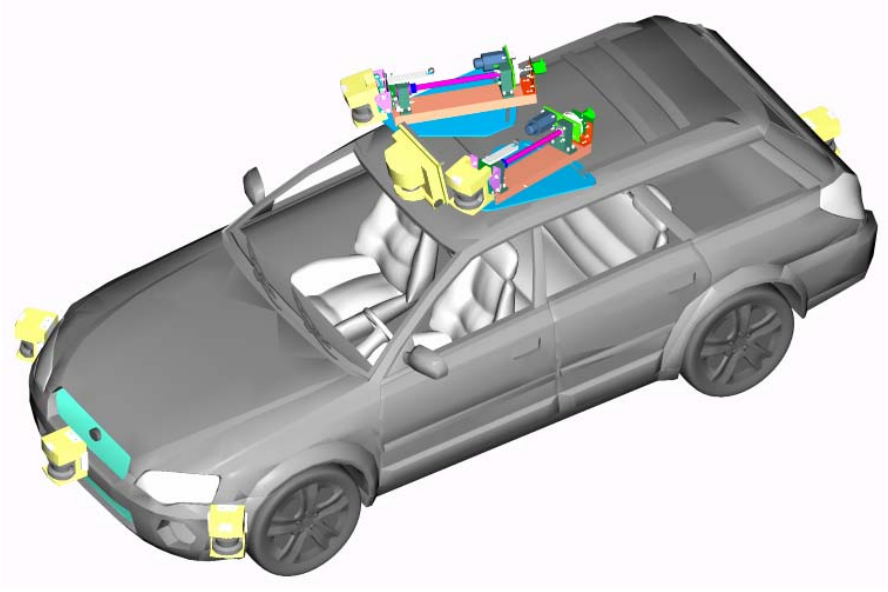


Fig. 6. The placement of the laser scanners around the car and on top of the roof.

For most scenarios the laser scanners in the front of the car provide sufficient sensory information to navigate an urban course. The side scanners in the front are mounted on different height levels than the central front scanner. Additionally, they are attached in a slightly rolled position, so that the scanning planes of the three frontal laser scanners overlap in front of the car. These crossing planes focus the attention of the sensors in the area right in front of the car.

Additionally, one scanner is mounted centrally on the roof. With a slight tilt downward, this scanner detects curbs and lane markings approximately 10 meters in front of the car (Figure 6).

2.2.2 3-D Laser Scanners

Despite the design considerations presented, the ability to detect a robust set of obstacles necessitates the use of scanners that perform significantly outside of a single plane (i.e., 3-D laser scanners). Commercial 3-D laser scanner systems are very expensive. TeamUCF chose to emulate 3-D scanning by combining a 2-D laser scanner with a servo motor, such that the scanning plane can be rotated along a chosen rotation axis (e.g. in (Surmann 2003) and (Wulf 2003)). In collaboration with the Mechanical Engineering department of the University of Central Florida, we developed an actuated mount that rotates a 2-D scanner to generate 3-D samples, using a single rotational axis and relatively low rotation rate (Figure 7).

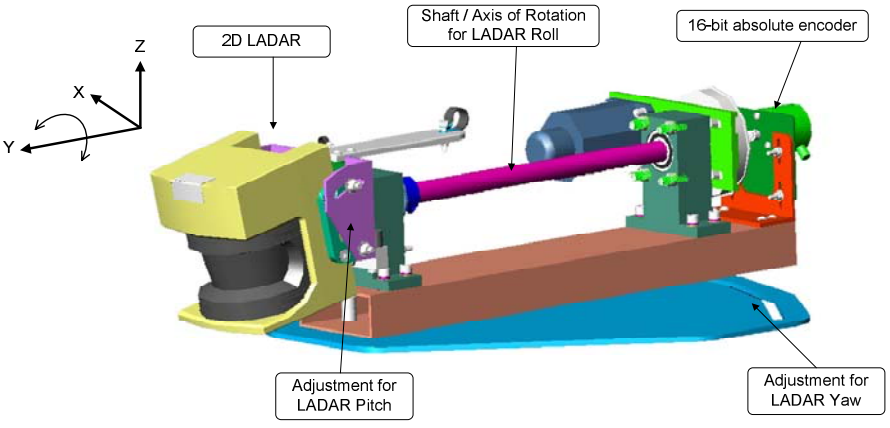


Fig. 7. CAD drawing of the actuated laser scanner mount.

In the design of the actuated mount TeamUCF was guided by three main design paradigms:

- Adjustability: A slotted design allows rapid adjustment of most angles.
- Robustness: Anodized aluminum mount with stainless steel fasteners for continuous outdoor application. Sealed radial ball bearings resist encroachment of debris and retain lubricant. Teflon plain bearings serve as thrust bearings on either of the front bearing carriers. Rubber bump stops are incorporated to minimize impulse to components should any failure lead to over travel of the sensor. The cable harness is routed to minimize strain from the repetitive motion and ruggedized with braided sleeving and plastic conduit. All electrical components meet IP65 specification.
- Maintainability: All fasteners, bearings and electrical parts are off-the-shelf products that are widely available.

TeamUCF considered different motion patters. The advantage of a continuous 360 degree motion is that the scanners are moved with a constant velocity and hence the interpolation of roll positions is simplified. Unfortunately, there are challenges involved when the electrical and data connections have to made through that rotating assembly (e.g. with slip rings). In our testing we achieved a sufficient coverage (e.g. see figure 8) with a cyclic movement of +/- 20 degrees. There is no special connection for the power or data necessary and the roll movement of the shaft is closely tracked by an optical encoder. The main reason we chose a cyclic over a continuous movement was the ease of mechanical implementation coupled with a sufficient scanning coverage. Based on simulations of several configurations and movement patterns, we used two rotating laser scanners that have a yaw angle of ±22° and a pitch angle of -11° relative to the sensor roll axis. This configuration yields a high scanning point density in front of the robot, where most on-road obstacles are expected.

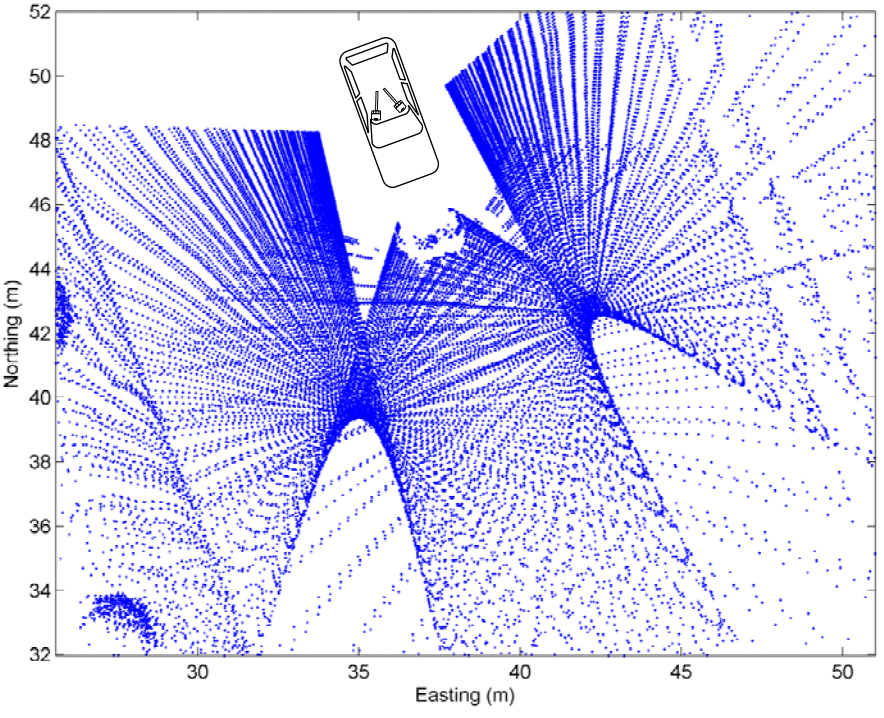


Fig. 8. Point density achieved after a 2 second scan.

Each scanner is continuously rotating around the y-axis (roll) with a motion radius of ±20°. The most current roll position is determined by an absolute 16-bit optical encoder that is directly attached to the rotating shaft. Since each mount has a separate encoder, no roll movement synchronization between the two mounts is necessary.

The point density that is achieved on the ground plane after a 2 second scan is shown in Figure 8. Clearly, the highest point density is achieved in front of the robot. Moderate point densities towards the far frontal left and the far frontal right of the robot favor the detection of robots in intersection and merging scenarios. Notice that the blind spot of one sensor is covered by scan lines of the other laser scanner. Although the point density is low in these areas, both scanners complement each other in achieving a complete coverage.

2.2.3 Cameras

During early development, TeamUCF used video cameras for lane detection and long-range obstacle recognition. These systems proved problematic in testing being particularly susceptible to variable lighting conditions, ubiquitous shadows and non-uniform street texture. The basic failure mode in the presence of these

conditions was a temporary loss of valid lane data. Promising results in road marking detection and long-range recognition of oncoming vehicles could not be extended to a robust framework that worked in diverse situation. Further, the performance of the top 2-D laser scanner in detecting lane markings had proved to be at least as robust as vision approaches, and could be substantially better in some scenarios. The problems with vision were exacerbated when TeamUCF arrived at the NQE and observed the lighting conditions during expected test windows, the actual quality of road markings, and the extent to which non-traditional road markers (specifically concrete k-rails) were used to designate lane boundaries. TeamUCF made a real-time decision at the NQE and abandoned the idea of using cameras and relied on the laser scanners as main source of sensor information.

2.2.4 Doppler Radar

The laser scanners employed by TeamUCF were unable to detect obstacles beyond 50 m, but high speed merge scenarios dictated longer range. To overcome this range limitation, an actuated Doppler radar sensor was mounted in the front of the robot. The "Stalker Radar Speed Sensor" returns the speed of the strongest moving object in its measuring cone (3 dB beamwidth of 12°) and has an advertised range of 3km (ideal for speed traps). In practice, it proved to be a disadvantage to have that extensive range, because the Urban Challenge scenarios effectively limit required range to 100m. Since the sensor only outputs the speed of the strongest moving target and its direction of movement (incoming or outgoing) but not its distance, a distance-based filtering is not possible. This problem is resolved by simply pitching the radar, so the maximum detection range is determined by the 3dB beamwidth (Figure 9). Given the known values of height h, 3dB beamwidth β and the desired range d, we can calculate the pitch angle α by

$$\alpha = -\tan^{-1}(h/d) - \beta/2$$

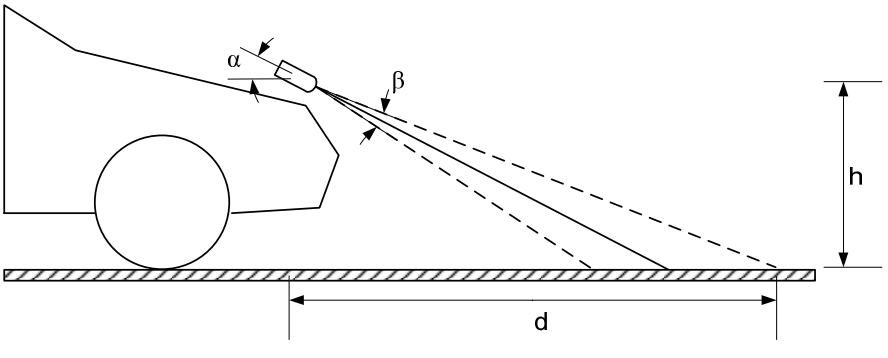


Fig. 9. The Doppler radar is pitched to achieve a desired detection range.

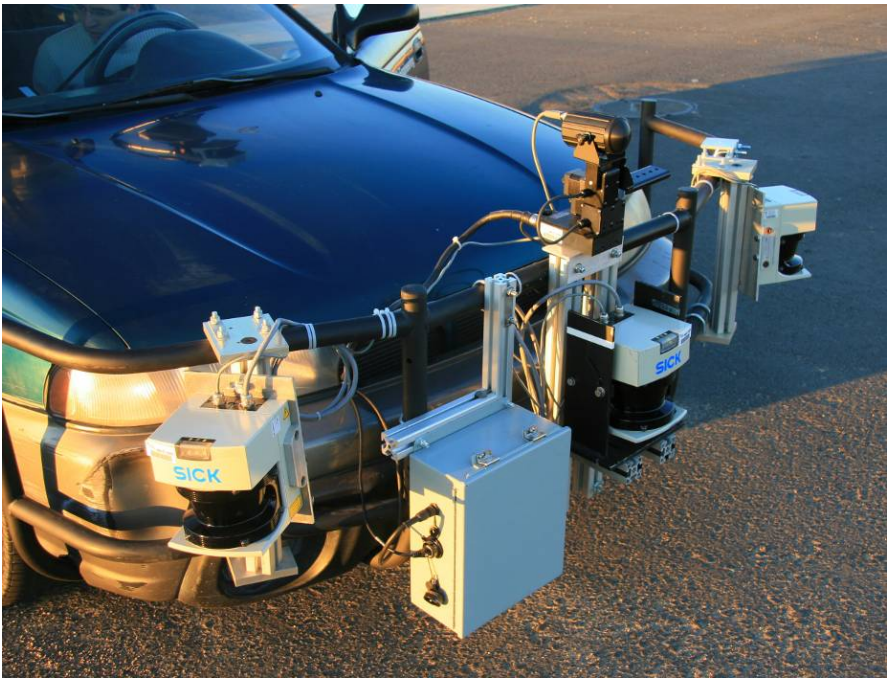


Fig. 10. The front rack of Knight Rider with 3 laser scanners + actuated Doppler radar.

To account for a diverse range of intersection geometries, the radar was mounted on an outdoor pan-tilt unit PTU-D47 manufactured by Directed Perception (Figure 10). With a maximum yaw speed of 300°/s and a maximum pitch speed of 60°/s, the radar could view all pockets of the intersection sequentially within a couple of seconds.

2.3 GPS/INS

Knight Rider navigation fuses a number of sensors to provide an accurate determination of the current robot state which includes position, heading, speed, and attitude. Attitude information is used specifically by sensor subsystems to transform sensor relative geometry measurements into a world frame for inclusion in the environmental model. The vehicle's existing ABS sensors could be used to provide the current speed of all four wheels and this information augments states maintained in the Oxford GPS/INS. Differential corrections are provided to the GPS/INS. UCF had investigated dual antenna performance to augment attitude information, but performance was insufficiently different from the single antenna system now employed to warrant the complexity and idiosyncrasies of such a system. Position accuracy of the operational system was <<10cm while angular accuracy was approximately 0.3°. GPS/INS data was made available to all processes at a 100Hz data rate.

3 Software Architecture

Because of the relatively small development team, there was little to be gained by extensive software partitioning. Software was effectively divided into six areas: laser data processing, vision data processing, sensor fusion, intelligence, planning, and control. Each area was owned by one team member (one member owned two areas) who had overall software responsibility, but all team members contributed to all areas of the architecture. A common interface specification allowed seamless data exchange.

3.1 Laser Data Processing and Sensor Fusion

The data from the laser scanners is transmitted over a serial data line with a nonstandard baud rate of 500 kBaud. This serial data is read by a Moxa UC-7110 embedded computer that was modified to support the unusual baud rate. Each embedded computer is capable of receiving data from two laser scanners simultaneously, assigning a timestamp to each scan and publishing it over a UDP unicast to a central receiver module. Through the middleware infrastructure, the sensor data is made available to the robot's computer network. Several subscriber software modules receive the published data and extract road features and obstacles. Effectively sensor processing algorithms have access to time-stamped range and intensity data as a function of scan angle, which can be transformed into world coordinates through an appropriate transformation involving sensor mounting angles and real-time measurements from the GPS/INS.

3.1.1 Lane and Curb Detection

DARPA provided a moderately dense collection of waypoints (<100m spacing), but not quite sufficiently dense to rely on waypoint definitions alone to accurately describe road geometry. Segments with sparse waypoints were part of the tested road network and sensory road following techniques were essential for a safe traversal. Furthermore, even with INS aided GPS, an intermittent GPS outage and the resulting deteriorating position estimate could have led to inaccuracies in navigation (although TeamUCF never observed this type of GPS failure).

The reflectance of painted road markings is in most cases enhanced by additives like reflective glass beads. This property facilitates the detection of those markings by a laser scanner. In addition to the range measurement, the SICK laser scanners employed by TeamUCF outputs the intensity of the reflected beam. Range and intensity variations (Figure 11) can be used to define road boundaries.

The road detection strategy employed by TeamUCF was twofold. First detect the curb discontinuity in the laser range scan (using the top scanner), and then detect the lane marking discontinuity in the intensity scan. Both detections were accomplished in the native polar space of the laser scanner output. By calculating a simplistic range-normalized operator:

$$r_{dot} = \frac{r_{i+1} - r_{i-1}}{r_{i+1} + r_{i-1}}$$

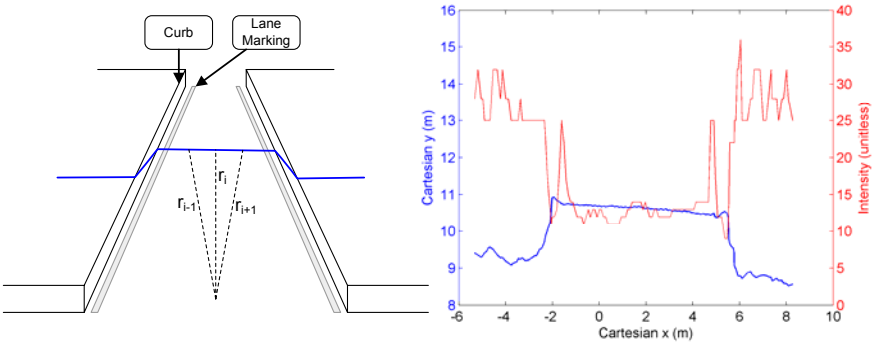


Fig. 11. Left - Notional laser scan line. Right - Range (lower) and Intensity (upper) observed on actual road data.

and thresholding the results, the discontinuities can be easily identified. Data association was relatively simplistic, but adequate for the challenges presented. On each measurement, all rightmost curb boundaries within ½ lane width were associated with the right curb, while all leftmost curb boundaries within ½ lane width were associated with the left curb.

The detected road/lane boundaries are then tracked by a second-order Kalman filter, which ensures that broken curbs or broken lane markings do not seriously impact the estimated boundary points. Since RNDF input format guarantees that waypoints, when present, are accurate and that lane widths, when present, reasonably represent the lane, the only point of interest for the environmental model is the world coordinate of the center point for the current lane of travel. This point is used by the environmental model if the spacing between known points is larger than a threshold and essentially became an additional, lower confidence, waypoint (TeamUCF sought a waypoint spacing of 10m).

3.1.2 Obstacle Detection with 2-D Laser Scanners

In order to detect and extract obstacles, laser scanner points were transformed into world coordinates and fed into a probabilistic occupancy grid originally developed by (Moravec, 1988) and excellently treated in (Thrun et al., 2005). The occupancy grid is probabilistic in the sense that it represents the map as a field of random variables, arranged in an evenly spaced grid. Each grid cell is either occupied or not, hence the random variable is binary. An occupancy grid mapping algorithm implements an approximate posterior estimation of those random variables. TeamUCF utilized a grid cell size of 0.5m x 0.5m. If a scan point is within the grid cell, the cells counter is incremented and compared to an occupancy threshold. The line between the laser's origin and the scan point is traced and the counter of traversed grid cells is decremented.

TeamUCF modified the standard concept of an occupancy grid to specifically suit mapping of an urban driving environment. In particular

- TeamUCF utilized a 2-D occupancy grid. It is not important to know at which particular height an object resides, but that it exists in a height bracket above (or below) the road level that poses a danger for the robot. Points outside this band (either too high or too low) are discounted from consideration in the grid.
- TeamUCF used a dynamic moving map in order to minimize memory and computation expense. Intelligence and planning systems are concerned about detailed obstacles only within the vicinity of the robot (obstacles outside this range are likely to change). TeamUCF used a robot-centered 120 m x 120 m occupancy grid which was moved whenever the robot moved 10 m. This ensured that all obstacles in at least a radius of 50 m around the car are mapped (effectively the maximum range of the laser scanners).
- TeamUCF required data for each grid cell to be refreshed repeatedly or lost over time. This reduced the impact of potentially outdated data from cells not effectively re-sampled by the laser scanner within 2 seconds(due to robot motion or more likely orientation). This "fading" of occupied cells was implemented with timestamps.
- TeamUCF used a dynamically generated lane mask to further eliminate obstacles outside the road network. The lane mask was maintained by the environmental model and reflected the best estimate of the road network. Cells sufficiently far from the road network were simply ignored. The mask was communicated as a collection of potentially overlapping polygons. The vehicle relies on accurate GPS data and masks were selected based on nominal GPS drift of < 1m. Masks were only "re-aligned" to the extent that roadway control points (initially RNDF points) were updated as the vehicle traversed the path. In fact this roadway update was a far more significant source of path changes than GPS drift, often adjusting the roadway by many meters.

The point transformation and occupancy grid mapping was executed separately for each of the four statically mounted laser scanners at a rate of 20 Hz. At the end of each iteration the resulting occupied cells of all 4 grids were published to the sensor fusion module.

3.1.3 Obstacle Detection with Rotating 3-D Laser Scanners

An important observation from the process of curb detection is the apparent smoothness of the road surface. In fact, all obstacles that have to be avoided by the robot distinguish themselves as a discontinuity in respect to their spatial surroundings. The simplified operator used in the curb detection process is insufficient in cases where the laser beam hits the scanned surface at an extremely acute angle, because larger changes in the measured ranges can be expected even if the observed surface is smooth.

One key advantage of using a rotating 2-D laser scanner to emulate a 3-D scanner is the preservation of spatially continuous scan lines. That is, each pair of adjacent scan points in a given scan line is in most cases spatially close in the

observed environment, so that an evaluation of surface smoothness along the scan line is possible. Ideally, an operator on the scanning data returns gradient changes independently of incidence angle of the laser beam and scanning location relative to the environmental feature.

As elaborated in (Adams, 2001), the change in gradient from two scan points A, B to the new scan point C can be described by:

$$\Delta \left| \frac{dy_s}{dx_s} \right|_{x_s = B} = \frac{(d_i d_{i+1} + d_{i+1} d_{i+2} - 2 d_i d_{i+2} \cos\alpha) \sin\alpha}{{d_{i+1}}^2 - d_i d_{i+1} \cos\alpha - d_{i+1} d_{i+2} \cos\alpha + d_i d_{i+2} \cos(2\alpha)}$$

where α denotes the angular resolution of the laser scanner and $d_i, d_{i+1},$ and d_{i+2} the observed range measurements to points $A, B,$ and C respectively. The key here is the definition of a local coordinate system (x_s, y_s) in the sensor space once the first two points are scanned, where the x_s axis is joining the two points A and B as illustrated in Figure 12. The computed gradient is then the gradient in any chosen coordinate system, no matter from which side the environmental feature is scanned.

Thresholding the resulting gradient changes in a given scan line yields the points that describe the sought for spatial discontinuities in the environment. Appropriately choosing the threshold allows detection of as diverse objects as road curbs and cars. The resulting points are transformed into world coordinates similar to the transformation for the static laser scanners, with the added degree of freedom for the roll motion. An interpolation of the roll angles for each point of a scan line accounts for the continuous movement of the scanners. An example

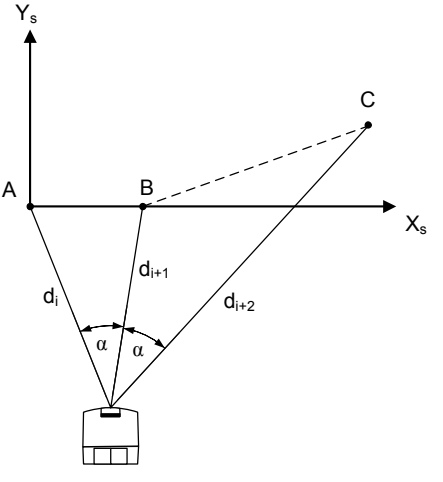


Fig. 12. Three sequential scan points A, B, and C

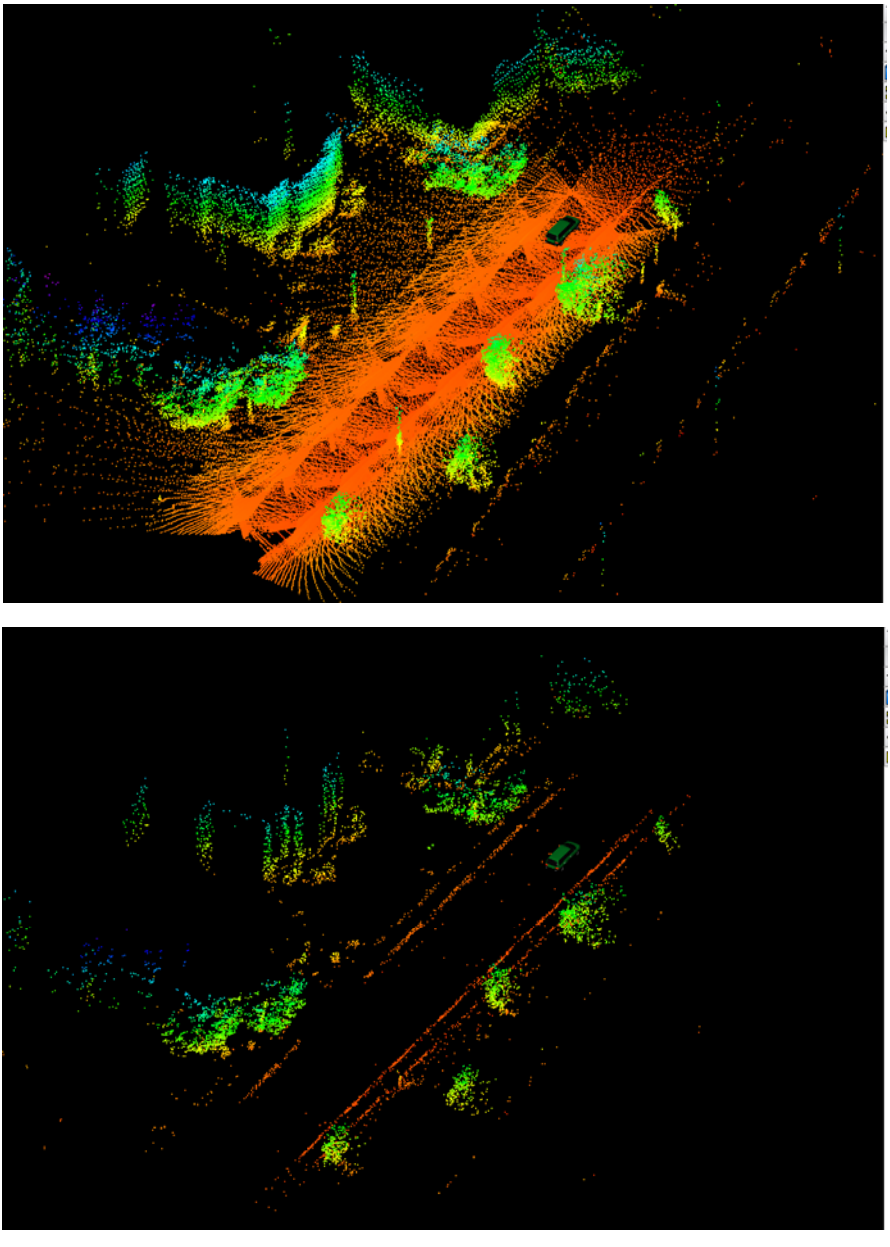


Fig. 13. Top- Typical 3-D point cloud. Bottom - Processed points prior to occupancy extraction.

point cloud before and after the described processing is shown in Figure 13. All the remaining 3-D scan points were inserted into a separate occupancy grid as was described for the 2-D scanners.

The scan line gradient processing, transformation of the remaining points to world coordinates, and occupancy grid mapping is performed data-driven at the full laser scan rate of approximately 35 Hz.

3.2 Sensor Fusion

The sensor fusion module receives five occupancy grids from the laser processing modules, four from the static 2-D scanners and one from the 3-D laser scanners. Each of these grids represents a probabilistic "best guess" about obstacle locations from the particular sensor's point of view, suggesting to merge the grids disjunctively. The resulting disjunctive occupancy grid contains all known obstacle cells in a perimeter around the robot (Figure 14).

From experiments it became clear that the occupancy grids from the 2-D scanners were more likely to contain false positives due to unusual road geometry or rapid elevation changes in the environment. In contrast, obstacle cells extracted by the 3-D laser scanners proved to be more reliable indicators of real obstacles. To avoid deadlocks due to false positive obstacles, obstacle cells from the 2-D laser scanners that are non-existent in the grid of the 3-D scanners were deleted after a predefined deadlock time. This mechanism is essential and formed the basis of our approach to driving on hilly terrain.

Separate obstacles were extracted by a connected component analysis and subsequently their polygonal outline and centroid were determined. The grid-based representation of the world allows for a natural quantization of the coordinates of polygon vertices. Based on position and velocity from the previous sensor fusion iteration, expected obstacle locations were extrapolated and current obstacles were assigned consistent IDs based on minimum Euclidean distance to the expected locations. If no correspondences for previous objects could be found in a certain radius, their ID was deleted. Conversely, if new objects appear that could not be matched, a novel unique ID was assigned to them.

Accurate tracking of obstacle velocity cannot be achieved on the grid level due to the coarseness of the occupancy grid. Fortunately, there is a good chance that the object is directly visible in at least one of the laser scanners, whose distance measurements are accurate to within 4 cm. By transforming the obstacle centroid from world coordinates to laser coordinates, it is determined if it is within line-of-sight of any of the scanners. If a corresponding scan point can be found at the expected obstacle range, its transformed world coordinates can be used to track velocity through sensor fusion iterations. The resulting velocity assigned to a specific object ID is stabilized by an exponential moving average filter. If an obstacle's velocity is below 1 m/s it is assigned a static flag and an associated time it has been observed not to be moving.

Another task for the sensor fusion module was to estimate the geometry of the travel lane by storing and extrapolating the lane center points extracted by the curb/lane detection. The center points received within the last 20 meters of travel were approximated by a second-order least squares fit parameterized to work directly on UTM world coordinates. Specified points in front of the car at 5, 10, 15, and 20 meters distance along the second-order curve were extracted.

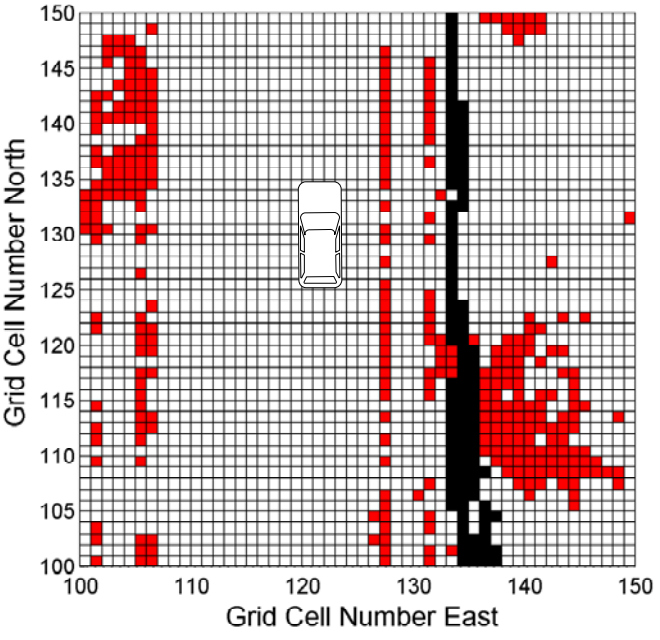


Fig. 14. Snapshot of an occupancy grid fragment, 2-D (darker) and 3-D (lighter) cells.

All the sensor fusion processing is performed at a rate of 5 Hz. A full list of known obstacles with ID, velocity, static flag, and static time as well as a list of lane center points in front of the car is published to the AI.

The calibration of the positions and static angles of the laser scanners with respect to the car coordinate frame was performed in testing prior to arrival at the NQE. The calibration approach was practical. We scanned previously known features (like a corner of building) that showed up in multiple laser scanners and found yaw/pitch/roll and translation with respect to the car with a 3D Iterative Closest Point (ICP) algorithm. The changing roll angle of the rotating scanners was very closely tracked by the optical encoders mounted on the moving shaft. Time stamping the car state information, received laser data and encoder data, allowed interpolation of car state and laser state for each scanning point. This approach worked very reliably, although it relies heavily on the accuracy of the GPS/INS system.

3.3 AI - Intelligence

The AI module was responsible for the high level planning and tactical level decision making for Knight Rider. In designing the AI module, heavy emphasis was placed on existing research in driver modeling approaches. Development of driver models is integral to the quest of better understanding how humans drive which in turn supports effective in-vehicle interfaces, better collision warning and

avoidance technologies, and improved driver-related human factors. Driver models used for simulating traffic in immersive driving simulators are particularly appealing because of the requirement of realistic looking behaviors that extend all the way to faithfully reproducing motion trajectories. The AI module used in Knight Rider was based on a driver model derived from prior work in driving simulation (Cremer, 1995 and Papelis, 2001), but with several extensions and enhancements to address incomplete awareness of the driving environment and rules and requirements of the competition.

Competition specific issues aside, the driver model was structured according to Michon's three level hierarchy (Michon, 1985) that breaks the driving task into strategic, tactical, and operational levels. The strategic level is concerned with high level goals such as navigation. The strategic level mapped these goals into a series of sub-goals, which remain unchanged unless affected from external factors. The tactical level was responsible for generating sequential tasks to implement a given sub-goal. The operational level was responsible for low level guidance of the robot. The three level hierarchy provided an effective cognitive model and is consistent with the view that driving is a compromise between achieving goals and addressing ongoing constraints (Boer, 1998). Through a temporal process of selection of alternatives, the driver model pursues goals in a top-down fashion, starting at the strategic and ending at the operational. Constraints flow the opposite way, starting either at a tactical or the operational level and reaching the strategic level, which in turn adapts accordingly.

3.3.1 AI Architecture

Figure 15 depicts the decomposition of the driver model in the three behavioral levels and associated flow of tasks, from top to bottom, and constraints, from bottom to top. The road network information was read from the RNDF and converted into an indexed data structure that better supports robot navigation. This step also created several needed associations that are not explicitly provided in the RNDF, for example lane adjacency and direction information. The MDF was read and used to plan a mission which in turn was used by the mission planning logic to create a list of tasks. These tasks were implemented within the core AI module, which used sensor data and a-priori knowledge to execute the specific tasks. Low level motion requirements in the form of a series of geometrical points to drive along was passed to the path planner which interacted with the low level control mechanism to ensure proper robot motion.

3.3.2 Strategic Level

For the strategic level, establishment of the goals and the associated optimization functions was done by interpreting the competition rules. Materials provided before the competition provided specific operational boundaries, but no quantitative scoring information was given. As a result, the strategic level is focused almost exclusively on route/mission planning, and dynamic re-planning upon discovery of road blockages. The output of mission planning was a list of tasks, each of which corresponds to a tactical operation, such as driving, parking etc.

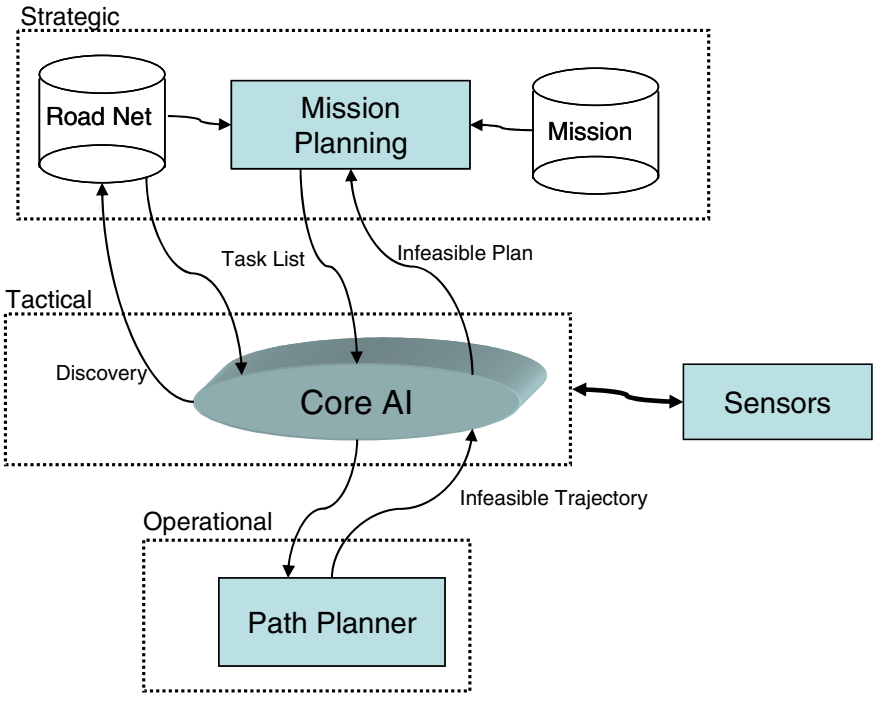


Fig. 15. Behavioral Model Block Diagram

Early performance testing using the hardware employed in the robot indicated that a straightforward implementation of Dijkstra's algorithm performed almost instantaneously on graphs with hundreds of nodes. At that point, work was underway on generating a graph from an RNDF, but even under the worst case assumptions, Dijkstra's $O(N^2)$ algorithm performed adequately, so the decision was made to utilize this approach for determining the route from one check point to the next. A simple algorithm was designed that starts by finding the best route from the current position of the robot to the 1st checkpoint, then augmenting that route by the best route from the 1st checkpoint to the 2nd, and continuing until all checkpoints have been exhausted. In order to successfully utilize this approach, two specific issues had to be addressed: first, development of an algorithm that would convert an RNDF into a graph amenable to min-path search, and second, developing a mapping between a route and a series of tasks that could be delivered to the tactical level for execution.

3.3.3 Graph Generation and Task Mapping

The traditional min-path algorithm finds an optimal route between two nodes on a graph. Optimality is defined in terms of the route cost, which is the cumulative sum of the cost of traversing each node and edge of a given route. In generating a graph from an RNDF, it is important to capture all navigation possibilities

inherent in the topology as well as capture a rational cost function that can be used to compute node traversal cost. The graph generation algorithm developed to address these issues involves two phases. The first phase was responsible for creating the graph nodes and the second phase was responsible for generating appropriate edges. The algorithm built a graph at the beginning of the mission and re-built the graph as needed during the mission.

To generate nodes, the algorithm considered all waypoints and included as unique nodes any waypoints that were:

- An exit originating on a lane
- An exit originating on a zone
- An exit target, either on a lane or a zone
- The 1st or last waypoint of a lane
- A parking spot that was a checkpoint of the current mission

Edges were generated from each node under the following conditions:

- If the node represented an exit from a segment or from a zone, edges were created to all destination nodes
- If the node was not an exit on a segment, a single edge was created to the nearest node located downstream on the same lane
- If the node represented an entry zone waypoint, edges were created to all parking spot nodes in the same zone and to all exits in the zone.
- If the node represented a parking spot, edges were created to all nodes representing zone exits
- To represent lane changes, edges were added between nodes on the same road that were on different lanes and downstream from each other. Such edges were added only when the RNDF topology allowed a lane change, i.e., when a dashed white lane separates the lanes
- For roads with two lanes of opposite direction, and for which a corridor allowing a U-Turn did not exit, an edge was added between the last node of a lane and the first node on the adjacent lane. This edge allowed the routing algorithm to schedule UTurns at the dead-end of two lane roads.

Figure 16 illustrates an example of the graph generation process.

Once edge generation was completed, a classifier was used to assign each edge to a tactical-level behavior that could handle the narrow problem of navigating the robot so it traversed from one node of the graph to the next. Associated with each tactical-level behavior was a cost function that produced an estimate of the cost associated with navigating this edge. Once costs were associated with the edges, the min-path algorithm was applied to generate a linked list of edges. The list defined the anticipated series of tactical-level behaviors that were invoked during the mission. A final step allowed reaching checkpoints by performing mid-road U-turns. During this step, the min-path search was performed four times, once with no U-turns, once with a U-turn from the current location and

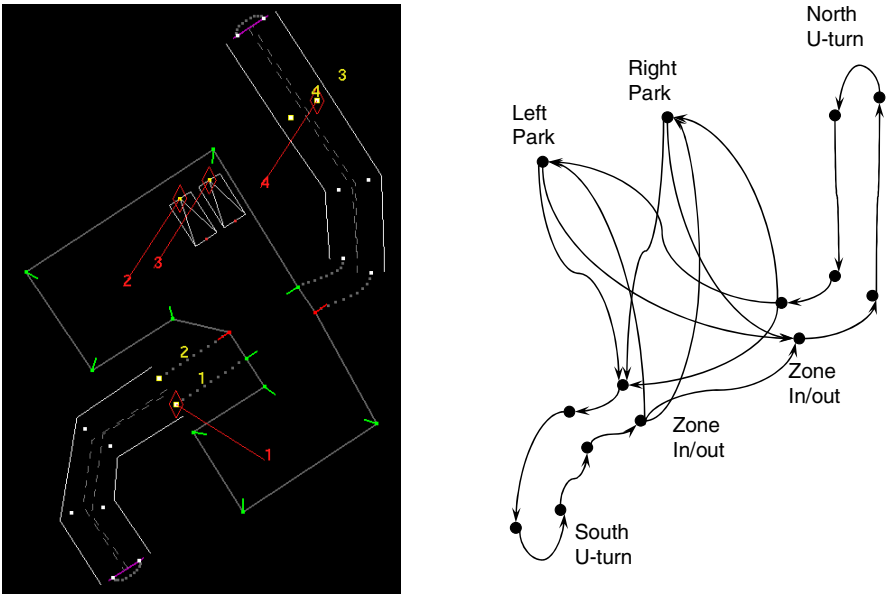


Fig. 16. Example of Route Generation.

straight arrival to the checkpoint, once with a straight departure but arrival to the checkpoint after a U-turn, and once starting and ending with a U-turn. The minimum cost path was selected.

It is important to note that consideration of mid-road U-Turns was incorporated in the algorithm even though such U-Turns were considered illegal. The rational for this decision was simple. Without knowing the relative cost of time performance versus illegal moves, it was unclear if the penalty of the illegal U-Turn would by offset by the time gain. Incorporating the mid-road U-Turns in the algorithm provided more options than not having this capability at all.

Modifications to the weight function can be used to bias the behavior of the robot. For example, the cost of U-turns affects the choice between driving down a dead-end road and performing a U-turn versus driving around a larger loop that involves no U-turn. During testing and during the competition, experience and improved rule understanding yielded several calibrations of the weight functions which proved critical in the success of TeamUCF during NQE.

One example of such a calibration was elimination of mid-road U-Turns. During the NQE, it became clear that the time it took to complete any one of the courses had little weight when compared to safely finishing the course. The decision was made to eliminate mid-road U-Turns, which was achieved by modifying the weight function so it assigned a very large cost to such a maneuver.

3.3.4 Tactical Level

The tactical level was focused on implementing the list of tasks produced by the strategic level. The tactical level was also responsible for road discovery. Road discovery is the process by which existing lanes are augmented with sensor data that provides a fuller centerline description than what was originally available. To support road discovery, a confidence value was associated with each waypoint. Initially, all known waypoints receive a confidence of 0.9, to indicate full knowledge of the (x,y) coordinate but incomplete knowledge of the heading. As the robot travels over a waypoint, the heading was updated and the confidence reached the maximum value of 1.0. At the same time, when waypoints density was below a threshold, guidance was provided by tracking the lane centerline ahead. This information was used to add waypoints into a lane, but with a lower confidence than the points specified in the RNDF, which were considered ground truth. The confidence of new points was passed from the sensor module. This process allowed the incremental increase in the confidence of newly inserted waypoints when repeated traversals over the same road segment occurred.

The tactical level implementation framework was a hybrid model that borrowed elements of context based reasoning and state machines. Context based reasoning allows a functional decomposition of the problem space into subspaces that are easier to handle. Each context is responsible for observing the current situation and "offering" to solve the problem at hand. Even though the context based formulation does not directly address concurrency, it does allow non-orthogonal activities to exist in multiple contexts, and in practice this is simply implemented by cleverly designing re-usable behavior objects. A fixed priority assignment was used to pick the context that takes control, if more than one context was willing to do so.

Even though the context-based approach has several advantages, it also presents some disadvantages. In particular, it does not lend itself to implementing procedural, step-by-step actions that are typically encountered in driving. A state machine approach is much better suited to this type of behavior. To facilitate modularity, a Hierarchical State Machine (HSM) model was used within each context to implement the appropriate behavior.

Figure 17 depicts the hybrid model of a context. The enable function is used to indicate if the context is willing to handle the situation. If the result is affirmative, the entry function executes to provide consistent initialization activities. The HSM then takes over while the context is active, and upon exit, a termination function provides a consistent point that performs context specific cleanup activities.

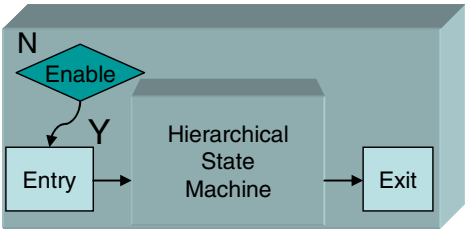


Fig. 17. Internal structure of a Context

The full execution semantics are illustrated in Figure 18**.** At start, the enable function of each context was executed and the first one that returned true activated the context and the associated entry function. The HSM code then executed periodically. If a higher priority context took over, the exit function was called and the selection process repeated.

A common problem associated with context-based behavioral modeling is maintaining continuity of behaviors during context changes. The localization achieved by using contexts is inherently incompatible with the need to maintain smooth transitions during context changes. For example, consider a context responsible for driving along a lane on a road with the speed limit set to 30 mph. Let us further assume that the road leads into a stop sign which is handled by a different context. The context responsible for driving is not aware of which context follows; that would violate the locality inherent in the framework. As a result, the driving context maintains the maximum speed and depending on where the transition occurs, the context dealing with the stop sign can receive control so near the threshold that stopping is not physically possible. To address this problem, each context was structured so that it composed the control inputs passed to the lower

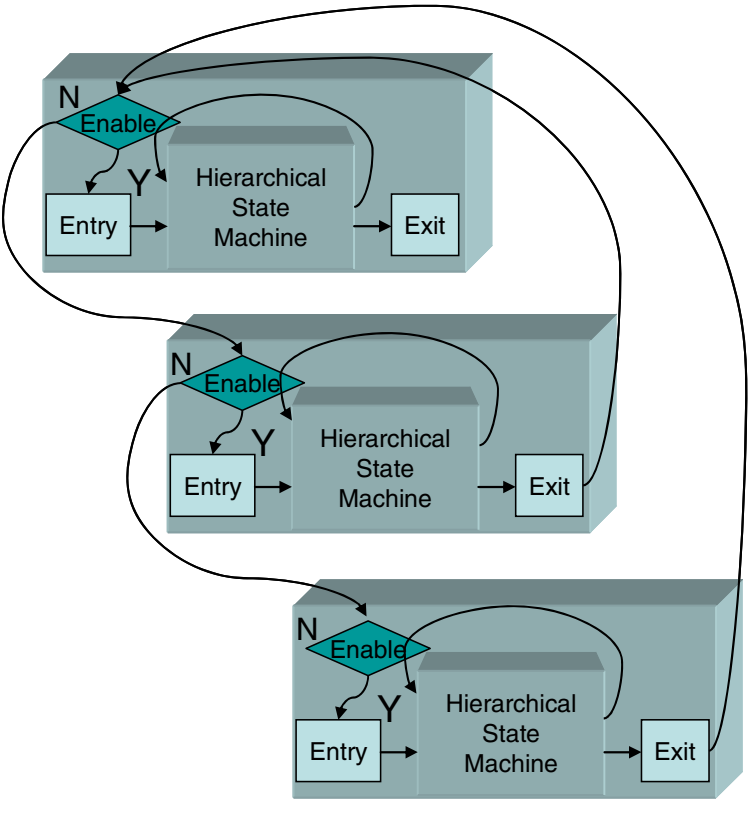


Fig. 18. Context Execution.

levels through an overloaded method that accumulated trajectory waypoints using the best available information at any time. It was thus possible for a context to recursively call the trajectory augmentation routine of subsequent contexts without explicit knowledge of which context followed. By maintaining a minimum length of trajectory specification, the path planner could anticipate speed as well as direction changes and plan accordingly. Using an overloaded method maintained the context independence while satisfying the need to plan ahead.

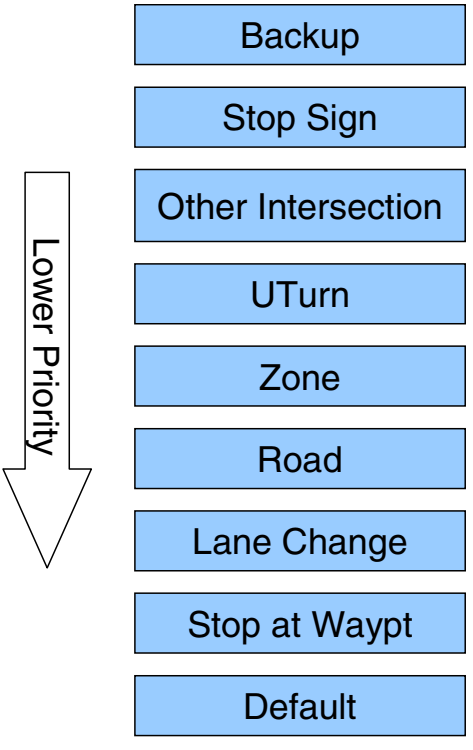


Fig. 19. Context Design.

This loop executed in periodic fashion in soft real-time mode. The tactical level thread was the primary thread within the AI process, with the strategic and operational levels implemented as separate threads that executed when triggered by the tactical level. In the actual robot, the execution rate was set to 10Hz, leaving 100ms per iteration. Use of asynchronous threads facilitated development and de-coupled the tactical control of the robot from the variable execution performance associated with the other threads.

Figure 19 illustrates the specific context design used in the Knight Rider. The prioritization order was designed to arbitrate between overlapping domains.

For example, handling an intersection with stop signs was higher precedence than a regular intersection. The lowest priority context (Default) served several purposes. During development, it acted as a self check mechanism that pointed out gaps in the system. During autonomous navigation, it served as the central place in which a last effort could be pursued to handle an unexpected situation.

Backup

The intention of this context was to drive the robot in reverse when doing so would allow meeting a checkpoint located behind the robot. This context did not directly map to an activity produced by the route planner, but as the highest priority context had the opportunity of checking for this situation. The context consisted of a single state that attempted to backup only when the next checkpoint was located within 3 robot lengths and there were no obstacles in the way.

Stop Sign

This context was activated when the robot must cross an intersection from a lane that was controlled by a stop sign. The structure of this context is straightforward, as illustrated in Figure 20. Note that sub-states used to implement timeouts are not shown.

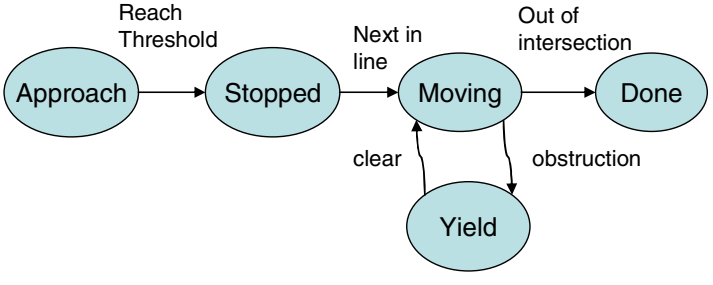


Fig. 20. HSM for Stop Sign.

Upon activation, the robot approached the threshold and came to a stop. Information about the intersection geometry was utilized to create a set of *pockets*, each representing other lanes into the same intersection. Pockets were classified as *peer* or *high* priority. Peer pockets were ones controlled by a stop sign, whereas high pockets had no signage. The operation of the Stopped state differs between the cases when all other pockets are peer versus having at least one high priority pocket, but in both cases, the robot remained in the threshold as long as an object was inside the intersection. Figure 21 illustrates an example intersection. The robot is approaching from the south. In this case, pocket P1 is peer and pockets P2 and P3 are high priority.

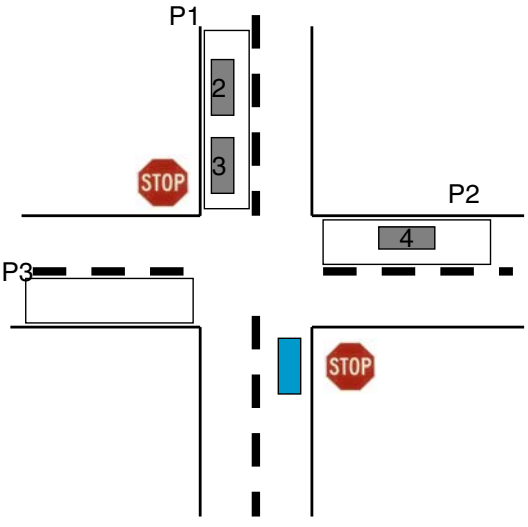


Fig. 21. Illustration of Pockets.

When all pockets are peer, the presence and velocity of objects in the pockets was used to determine right of way. Empty pockets or pockets with moving objects were ignored; pockets with a stopped object were assigned right of way. Once an object in a right-of-way pocket moved, the pocket was eliminated. This ensured that the robot only waited for the lead object when multiple objects were queued on a peer pocket. This logic brings up an important observation. The Knight Rider robot assumed other vehicles (robots or cars) would behave according to the rules of the road and only as a final resort (object all the way in the intersection) did the Knight Rider stop. During testing it became apparent just how many subtle cues human operators obtain from other driver's behavior and from the drivers themselves; cues that were not available to the Knight Rider.

When high priority pockets were present, the Stopped state did not transition as long as other objects in these pockets were in conflict. The velocity of the oncoming traffic was used along with their distance from the respective thresholds to determine if a conflict existed. When conflicts and right of way rules had been resolved, the robot transitioned into the Moving state which lasted while inside the intersection. Traffic entering the intersection forced a transition to the yield state, during which the robot stopped. Upon clearing or after a timeout period, the robot proceeded.

Timeouts were used in all waiting states to prevent live lock, which could be caused by other robots that intentionally or unintentionally violated the rules, or could be caused by phantom objects caused by sensor artifacts. Such timeouts were set at such a high value that they would never interfere with typical interactions. Further, TeamUCF made a calculated decision to prevent initiation of

a passing maneuver when near an intersection, but once a pass maneuver was initiated it would be completed, even if that meant passing while approaching a STOP sign.

Intersection

This context was responsible for controlling the robot through intersections for which there was no stop sign. The most obvious situation is a left turn that crosses opposite lane traffic. Because California driving rules dictate a full stop before crossing a yellow line, the design was similar to the Stop context, but with two key differences. Because there was no intersection area, once a go decision was made the robot proceeded without monitoring for side obstacles (it was assumed those moving obstacles would stop). In addition, there was no consideration for incident lanes controlled by Stop signs as they had lower priority.

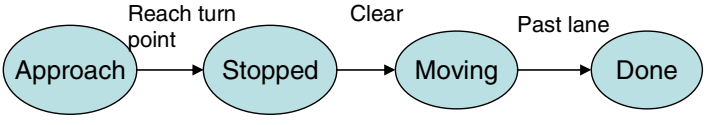


Fig. 22. HSM for Intersection Context.

UTurn

This context was responsible for implementing U-turns. The context activated in two situations. The first case was planned when the next activity in the route list explicitly called for a U-Turn. The second case was when a blocked road was encountered. The only difference between these two cases was that the latter case also triggered a re-routing operation at the strategic level, which generated a route starting at the lane that was the destination of the U-turn. In both cases, the context terminated upon completion of the maneuver.

To ensure that the re-route operation would not create a route that traverses the same blocked road, the edge representing the blocked road was tagged with a marker indicating the location of the block. According to competition rules, blockages were not persistent and the block marker was removed after the robot crossed a corridor, in effect forgetting the blockage.

Despite the relatively complex sequence of operations necessary to implement a U-turn, the behavioral complexity of this context is trivial, as shown in Figure 23.

The first state commanded the maneuver and monitored progress. Once the maneuver was completed, the state transitioned into the end state. In case of a collision threat, the Stop state waited for the obstacles to clear. Under certain conditions, for example when an obstacle was detected during the last backup maneuver, it was possible to transition directly to the end state (i.e., the U-turn had completed sufficiently to resume operation).

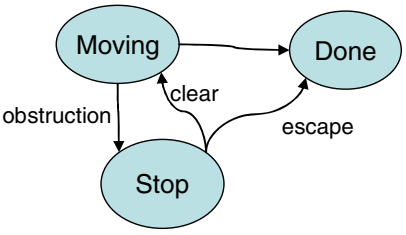


Fig. 23. HSM for U-Turn.

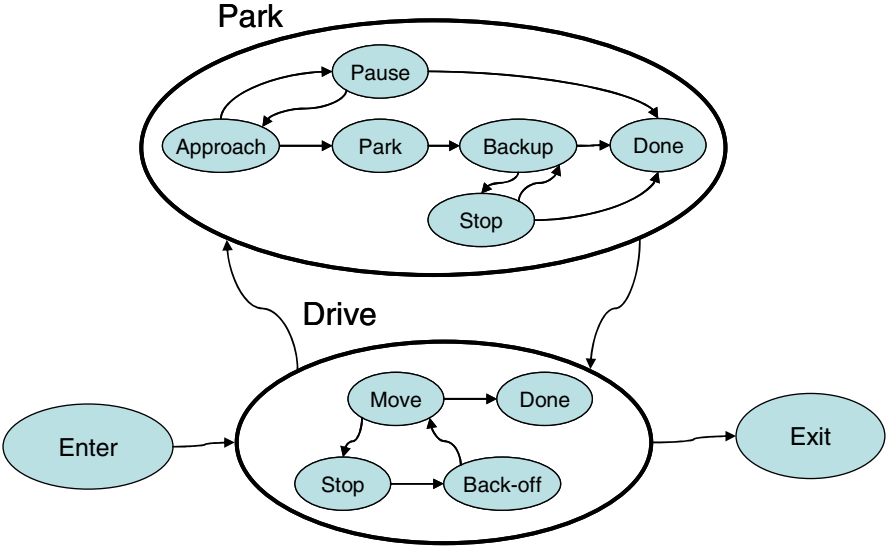


Fig. 24. HSM for the Zone Context.

Zone

The Zone context was responsible for guiding the robot during entry into, exit out of and driving while within zones. The controlling HSM is shown in Figure 24, with hierarchical states shown inside each other. The Entry state took over immediately upon reaching the waypoint that led into the zone. The Drive state was designed to move the robot from the current location to any point in the zone, while avoiding other stationary and moving obstacles. When a parking task was necessary, the Drive state moved the robot to a pre-parking spot, located adjacent or on the extended centerline of parking slot, then transitioned to the Park state. When a parking task was not necessary, the Drive state moved the robot to the zone exit and transitioned to the Exit state.

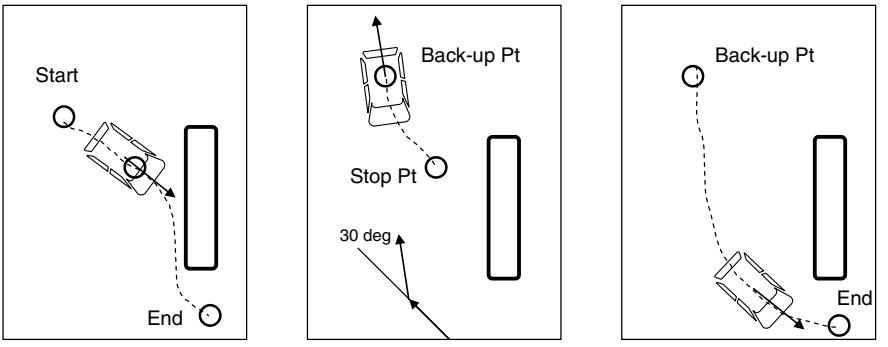


Fig. 25. Example of Back-off Operation.

While inside a zone the maximum speed was set to 5 mph, independent of the guidance provided in the MDF. All objects were set to avoid and all parking spots, except the target of the parking maneuver, were treated as obstacles thus ensuring the robot would not drive over them in accordance with California driving rules and DARPA instructions.

Even though the path planner could resolve the vast majority of situations it encountered, there are pathological cases during which the robot could "paint itself in a corner" (although this situation was never encountered at the NQE). The purpose of the Back-off state was to perform a backup maneuver that allowed the robot to get out of that situation. In order to compute the appropriate backup maneuver several geometrical approaches were tested in simulation. The most straightforward yet effective approach was to develop a set of deterministic backup maneuvers and pick the one to use at random. If the new position did not allow progress, the system cycled through the Stop and Back-off states and a different maneuver was attempted. An illustration of the approach is shown in Figure 25. In this example, the maneuver is to back up 1½ robot-lengths and turn 30° to the left of the centerline.

When a parking maneuver was necessary, the Drive state moved the robot to the pre-park position and the system transitions into the Park state. A straightforward sequence of state changes within the Park guided the robot in and out of the parking spot.

This design was tested extensively on various parking lots in the UCF campus and TeamUCF was pleasantly surprised at the robot's ability to navigate and park in very constrained parking lots that were filled with islands, obstacles and parked vehicles, significantly more complex than even the challenging scenarios presented in the NQE.

Road

This context was responsible for guiding the robot from one waypoint on a lane to a subsequent waypoint on the same lane. Because the mechanics of generating and tracking the trajectory were handled elsewhere, this context was behaviorally

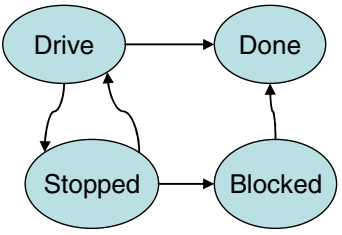


Fig. 26. HSM for the Road Context.

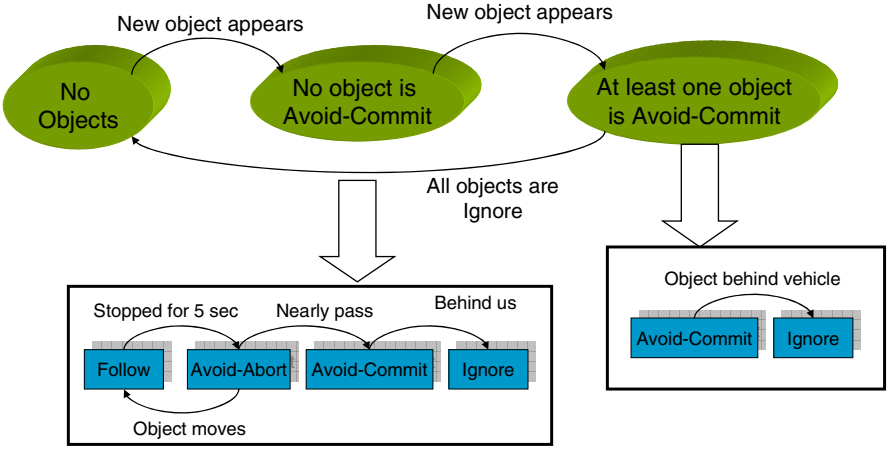


Fig. 27. Handling Object Disposition.

simple. The upper level of the associated HSM is shown in Figure 26. Lacking any significant interaction with other objects, the Drive state moves the robot along. If needed, road discovery was handled within the Drive state.

Given an object, the robot must decide if the object is something to follow or something to avoid. The approach utilized in Knight Rider was heavily biased by the characteristics of the data provided by the sensors. The approach utilized, illustrated in Figure 27, performed adequately given the competition rules. The top level states represent object classification states and are not directly related to the behavioral states of the HSM. The states in the bottom reflect object disposition.

The initial condition is driving with no objects in sight, and is shown by the left most state. Once a new object appears it is classified as *Follow*. An object which interrupts the baseline trajectory of the robot and is classified as Follow will cause the robot to stop at a safe distance behind the lead object. If the robot stops for a certain period, the disposition of the lead object changes to *Avoid-Abort* causing the path planner to plan around the object. As the robot goes around the object, one of two things can happen. The object can move, in which case its disposition reverts to Follow, or the robot will travel past the object (Knight Rider used

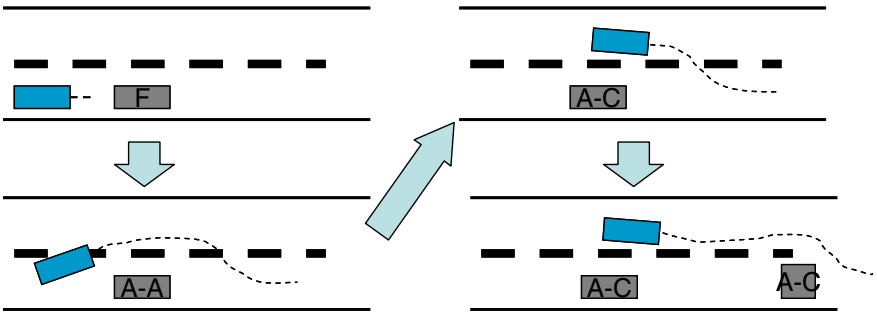


Fig. 28. Example Object Disposition Sequence.

committed once the front of the robot reached the rear of the object) in which case the object's disposition is set to Avoid-Commit. Once an object becomes Avoid-Commit it cannot revert back to Follow. Once behind the robot, its disposition is set to *Ignore*, which eliminates it from consideration. After at least one object has been classified Avoid-Commit, any new object is automatically classified as Avoid-Commit (i.e., once the Knight Rider started passing it continued to pass until it returned to the lane of travel). As objects are passed, they were labeled Ignore, and once all objects were labeled Ignore, the system reverted back to the start state with no objects. Figure 28 illustrates an example as a series of four snapshots showing operation of this approach.

The first snapshot, located on the upper left depicts the situation in which the object is set to Follow causing the robot to stop. After a brief pause, the object is set to Avoid-Abort causing the passing maneuver. In the third snapshot, on the upper right, the object switches to Avoid-Commit. The final snapshot illustrates how a new object appearing at that point is automatically set to Avoid-Commit providing a continuous passing maneuver.

Lane Change

The lane change context was responsible for guiding the robot while performing a lane change. Lane changes are planned during route generation, and were defined with an approximate start and end location. Because of the a-priori planning, lane changes were behaviorally rather simple, blending seamlessly between two Drive contexts.

3.4 Path Planning

The Path Planner (PP) acted as the bridge between the tactical missions defined by AI and commands that could be executed by the autopilot. It performed this operation by effectively acting as a function call for the AI that would generate a dense list of waypoints from a sparse set of waypoints provided by AI. It insured that the path generated by that dense list of points (0.5m spacing) was

kinematically feasible, met the explicit driving rules imposed by DARPA, and did not violate any constraints imposed by AI (such as speed limits or roadway boundaries). In addition to providing a dense path, the path planner provided path length, estimated time to complete path, and avoidance information associated with every obstacle encountered on the path. By effectively acting as a function call, AI could explore different scenarios with the path planner and select one to be forwarded to AI. Execution times were short enough that several scenarios could be explored in 100 ms.

The nominal problem for the path planner was to create an inbounds, kinematically feasible path from point P_0 to P_n, passing through intermediate points P_i, (Figure 29). Feasibility includes speed and acceleration limits as well as boundary constraints. Initial work followed a unique analytical approach (Qu, 2004 and Yang, 2005) but was modified significantly as it was realized that assuming flexible objectives yielded substantially better performance in many scenarios. Specifically, the path planner could explicitly violate objectives in the following manner:

- Any speed could be changed as long as the overall speed limits on segments were not violated and kinematic limits of the robot were not violated.
- Heading at a waypoint, if provided, was a suggestion and could be violated if required to keep a path in bounds. Direction of travel (forward or reverse) at a waypoint could not.
- Obstacles were classified as to be followed or to be avoided. If the desired path crossed a to be followed obstacle, the path was shortened based on the obstacle speed and type of area the obstacle was in (zone, road, near a stop sign, etc.). Basically if a stopped obstacle was likely in a certain scenario the robot could get closer to the obstacle than if it was unexpected.
- Intermediate waypoints could be moved perpendicular to the direction of travel if required to do so to avoid an obstacle.
- The final waypoint could be moved along the direction of travel, if required to do so because of an obstacle.
- Obstacles to be avoided, are to be avoided by at least 1 m if possible, but if not possible a path as close as 0.25 m is acceptable.

The approach taken was one of iteratively generating piecewise, continuous first derivative, cubic splines with updated control points as necessary to avoid obstacles and keep paths within bounds. Constraints were gradually relaxed if no solution was found. The use of splines generated smooth curves (although not necessarily optimum time of traversal paths) which could easily be evaluated for axial and lateral acceleration constraints.

Path planner operation when driving on a road or navigating a zone was fundamentally the same. Dense path information provided the path to follow until such time as goals, constraints, or obstacles changed. It also provided a convenient way for multiple systems to know where the robot was with respect to meeting its

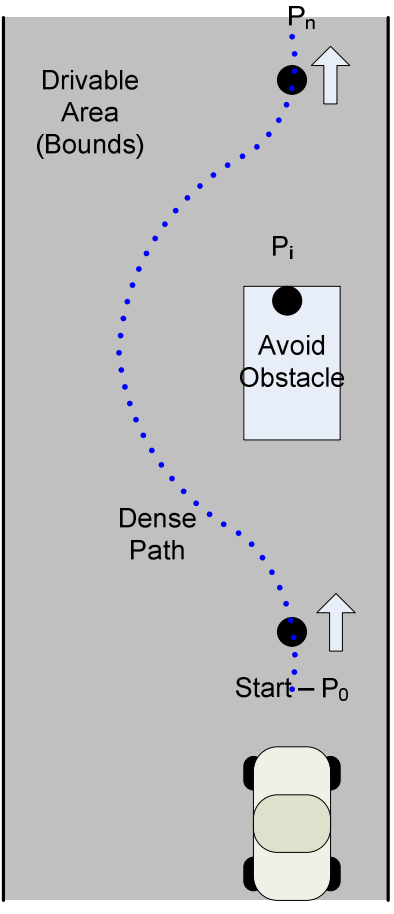


Fig. 29. Dense path generation from sparse goals

tactical objectives (namely, by associating the robot with the closest dense path point). While the path planner generated paths that could be driven by the vehicle, it was the autopilot's responsibility to follow the path generated by the path planner.

3.5 Control Systems

The purpose of the low level control system / autopilot (AP) was to physically actuate the plan put forth by the path planner. The overall operational inputs to the system were a list of dense waypoints with 0.5m spacing and the current navigation status. Each of these waypoints had an associated position, heading, and velocity, each of which should be physically realizable based on the robot dynamics. The overall requirements on the AP control systems were not near the vehicle or actuator limits, nor were their specific scoring parameters based on how precisely speed or steering was followed. Because of this, no optimal control

system design was performed and only rudimentary modeling of actual subsystems (i.e., second order response characteristics, rate and magnitude limits, etc.) was performed. Control system parameters were selected to mimic human drivers operating in similar circumstances. While no formal comparison to multiple human drivers was performed, there is clearly significant diversity in driver performance. The team selected control system parameters based on one particular driver that we collectively judged to operate the vehicle in the most reasonable manner.

All control systems ran on a single processor QNX using the real-time scheduling and interprocess communication systems of the operating system in order to minimize data latency and maximize predictability associated with operation.

3.5.1 Steering Control

Steering control was provided by a follow-the-carrot controller (Barton, 2001) coupled with a state machine to handle three specific driving modes (normal, stopped, and three-point turn). In the later case, tighter control of vehicle steering is essential to meet turn requirements. By passing a multi-point dense path between the PP and AP, the two systems did not need to maintain tight timing coupling and in fact the PP could operate significantly slower (e.g. 1Hz demonstrated in testing) than the AP (20 Hz) during periods where the environment or obstacle mix was not rapidly changing.

At a 20Hz update rate, the steering controller would 1) compute the closest dense path point to the current vehicle location, 2) look ahead on the path a fixed look ahead "time" of 1.5 seconds, and 3) determine the effective carrot point. Using the carrot point, a heading was calculated between the current position of the robot and the position of the carrot point. This heading is then compared to the physical heading of the robot. The difference in these heading becomes the error for a PI controller which feeds the steering wheel actuator. Steering command limits, steering rate limits, and integrator limits were included. In addition, integration was only performed during periods of the path where the path curvature was below a threshold. Special end of path logic (effectively linearly extending the path based on the last heading) avoided any steering discontinuities should path lengths become small or the vehicle commanded to a stop. The objective here clearly is to keep the robot generally on the path without undue precision (10 cm error is tolerable).

The steering angle calculated through the follow-the-carrot method was passed to the Elmo motor controller. Through the actuation system, the desired steering angle is converted to an absolute encoder position and the Elmo's internal PID controller physically maintained steering for any quick impulses or external stimulus feed back through the steering wheel from the environment. Stop to stop performance of the steering wheel was approximately 1.5 seconds.

Other path following schemes like Pure Pursuit (Coulter, 1992) and the hybrid controller employed by Stanford in 2005 (Thrun et al., 2006) were explored, but the simplicity of follow-the-carrot coupled with its robustness and accuracy led to its selection.

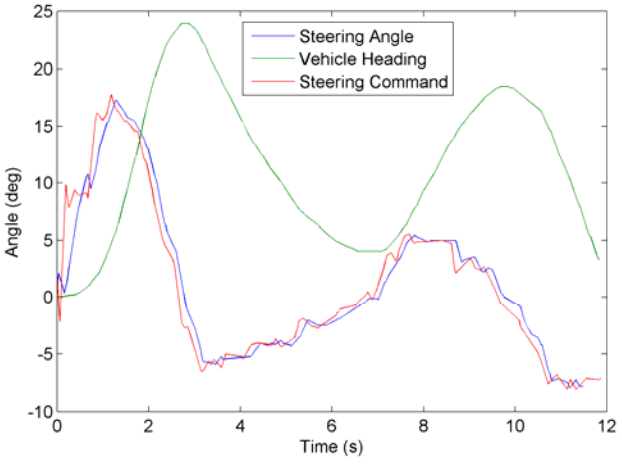


Fig. 30. Typical steering controller performance

3.5.2 Speed Control

While the steering controller looked forward into the planned path to obtain a steering command, the speed controller used a linearly interpolated value of the current desired speed as an effective cruise control set point. Again special end of path logic forced commanded speed to be zero at the end of a path and further forced a speed ramp down if the robot ever got so close to the end of a path it could not stop without violating acceleration constraints. Of course, PP logic should prevent this, but this strategy of hierarchical checking for basic system performance constraints proved critical during vehicle testing.

The subtle challenge associated with cruise control set points is determining when to brake and when to coast. TeamUCF utilized dual PI controllers, one for throttle and one for the brake, with a hysteresis crossover. The PI controller would have two internal states: throttle and brake. If the controller was in the throttle state, a positive value from the controller would be interpreted as a voltage to feed the throttle actuator. A negative value from the controller would represent cruising, which means that no throttle and no brake would be applied. A large enough negative value caused the system to transition to the brake state. In the brake state, a separate PI controller controlled the brake actuator. In this way the vehicle effectively operated in four states: 1) accelerating, 2) coasting throttle, 3) braking, and 4) coasting brake.

The performance and repeatability of the speed controller for subsequent runs over the same course can be seen in Figure 31. All elements of the system from the AI to the PP to the speed controllers were incredibly repeatable. This ability led to predictable behavior and simplified tasks for other systems. In the figure, the commanded volts curve illustrates throttle being applied (+ volts), brake applied (- volts), coasting (0 volts). The performance here is typical showing the vehicle operating within ~60% of its capability (10 volt peak).

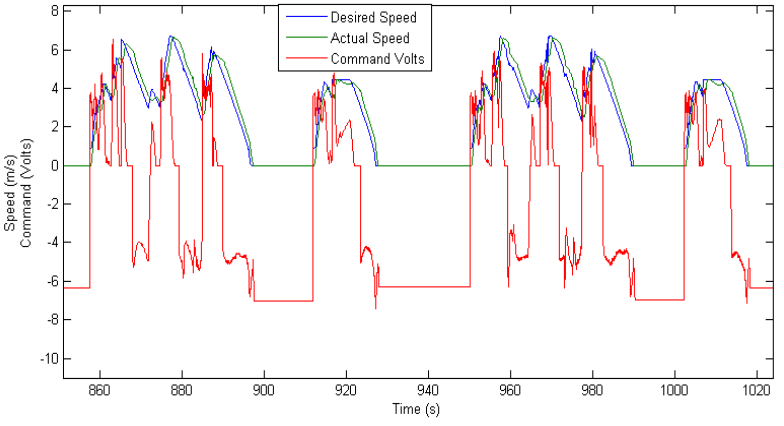


Fig. 31. Speed repeatability over 2 laps of NQE test area A

3.5.3 Drive State Control

Since the underlying robot vehicle was an automatic, no shifting was required, however, the system still needed to be brought into the proper gear (park, forward, and reverse). The main function of the shift controller was to ensure there was enough time between shifting and further timing operation for the robot to be safe. For example, if the robot was currently moving 1 m/s and in the forward gear and was just commanded a -1 m/s speed, the state would switch from forward to pre-reverse. Pre-reverse would smoothly stop the robot and wait until the speed was 0 m/s for a second. The brake would then be fully applied, and the state would switch to shift-reverse. In shift-reverse, the controller would send the voltage to physically actuate the robot to reverse and wait for another second to ensure the shift finished. At the end of that time, the system would then switch to the reverse state and reset the dual PI integrator error.

3.5.4 E-Stop Pause

The pause system also consisted of a series of state changes. The digital output line of the DARPA E-Stop device was read on an I/O pin and debounced to ensure that a false reading was unlikely. Upon receiving a pause command, the vehicle would be brought to stop and the appropriate combination of sirens and lights activated. Further, once in pause, a run command would cause the robot to wait 5 seconds before actually beginning operation.

3.6 ICE

The main communication system used throughout the robot was provided by the Internet Communication Engine (ICE) by ZeroC. This highly efficient middleware package allowed the robot's software to be distributed over a heterogeneous network of machines. Data types were handled though ICE's mechanisms which allowed data to be shared to multiple destinations regardless of Operating System

(OS) or programming language. TeamUCF utilized the publish and subscribe model in which multiple programs would be able to either request the most recent information by name or execute when new information was available. This framework worked well for the distributed architecture used and ICE overhead was never a factor even when publishing large sensor data structures.

Perhaps most importantly, the module independence provided by ICE provided the ability to test the robot off-line through simulation. Simulated modules that would take the same robot actuation signals over ICE and implement them on the actual robot were gathered and run through a dynamic physics simulation. Intelligence, planning, control system algorithms, and sensor processing algorithms could receive simulated or real data while still providing outputs in real time. In this way there was only one version of production and test code.

4 Project Process

TeamUCF participated in the original proposal submission, but failed to gain Track A status. Undeterred, the team executed on a capabilities driven implementation approach. At each stage of this approach, the key capabilities to be demonstrated next were determined. These capabilities were occasionally defined by DARPA (as in the case of a site visit) but were more likely to be defined by TeamUCF leadership. All systems were developed in parallel and to the level necessary to demonstrate the capability. In this way, incremental testing of the robot was performed for many months prior to the NQE. The downside of the approach is of course that full operational capability was invariably ready only just prior to the NQE.

4.1 Simulation and Modeling

Simulation was critical to the overall success of the project. The ability to test the robot in a multitude of scenarios, virtually, allowed different team members to test changes quickly on their own computer or over a network of computers. Further with a small team effectively responsible for both software development and robot testing and a limited window for testing, there were simply not enough hours in the day to conduct all the desired tests without simulation.

Coupling the simulation environment with a source management tool, in this case SVN, allowed problems that were detected on the real robot to be checked in as data files, sent to distributed team members, replicated in simulation, and resolved. That error could then be corrected in simulation, the source code checked in, and operation corrected on the robot, typically the same day. TeamUCF maintained no software lab or significant facility for any development activity.

The key to this ability was a product of the ICE middleware distribution and the modular nature of the software design. The ICE architecture allowed the real robot code to be used with virtual sensors. This software-in-the-loop scheme of sensor replacement was implemented by creating a threaded package each tasked with publishing realistic data. Each of these threads then published to the middleware level for use by the other modules.

The synthetic sensor fusion module had the ability to inject moving obstacles with complex obstacle behaviors into the virtual world in the same fashion as the post-processed data that the actual sensor fusion would develop from the laser scans and Doppler data. All synthetic data generation had the ability to be perfect ground truth, contain random fluctuations consistent with the noise levels measured in sensor systems, or perhaps most importantly to play back an actual vehicle log of the same data.

The synthetic E-Stop module reflected the DARPA E-Stop unit that could send the different pause and disable commands. Based on experiences in 2005, TeamUCF felt it was essential to test the operational effects of these commands to the rest of the system. The pauses were known to be a factor in the Urban Challenge due to the number of robots on the course and TeamUCF spent significant time pausing and restarting the robot in as many different scenarios as possible.

Synthetic kinematics was generated from a robot dynamics model using Ackermann steering with 2^{nd} order response functions with rate and position limiters for all actuators. Synthetic navigation was generated from this ground truth by adding appropriate filtered white noise which closely matched the performance of the actual sensor systems. Most importantly, these navigation and control processes contained appropriate process delays, modeled via FIFOs, to account for transport and process delay observed in the system. An early software error caused an asynchronous clock skew that was debugged using this technique and resolved with a combination of error correction and more fault tolerant algorithms.

All of the data produced by these systems was published to ICE. A graphical visualizer, Vevis, was developed in OpenGL to display the robot and environment in real-time. The software developer could then view the entire process unfold and observe the actions of the robot from movement to turn signals to direction of the radar. This overview could then display the route planned by the AI, the dense points created by the PP, and the movements produced by the AP. Vevis allowed the team to zoom on specific regions, load RNDF and MDF file for simulations of entire runs, or load the maps files on available textures to verify operation of the calculated road network. Because Vevis requested all usable information from ICE, this system was also used in real-time on the actual robot.

4.2 Testing Methodology

While simulation was key to TeamUCF's success, the team spent an equal amount of time on the actual robot. TeamUCF's test site was unfortunately only available after hours and therefore the majority of tests were conducted in the evening, which of course has added benefits when testing in the summer in Florida.

The test philosophy dictated that the team would run numerous shorter duration tests that could be quickly validated via simulation. A typical testing evening consisted of six hours of tests, consistent with amount of time expected at the final event, but the vehicle was never tested in a single six hour mission. While the majority of test runs were performed at night (for safety and facility access reasons) extensive test runs also occurred during the day. This included the DARPA site visit. From daytime test runs and through our experience from

the DARPA Grand Challenge 2005, we concluded that laser performance (from dazzle) was impacted but didn't significantly degrade the obstacle detection capabilities of the algorithms employed.

5 NQE and Race Results

The NQE and final event were held at George AFB in Victorville, CA. DARPA spent considerable time and energy preparing the facility to act as a safe, but challenging test environment for the robots. 35 semi-finalists were invited to participate in the NQE, including teams from all over the United States and several teams with a large international contingency. TeamUCF had arguably spent the least amount of money and had the smallest team to make it to the finals and was possibly the smallest team in the semi-finals as well.

NQE

The details of the testing to be performed at the NQE were unknown to the participants until they arrived at the event. Shortly after arriving, teams learned that the qualifying event would consist of a series of missions in 3 test areas, creatively named A, B and C, which stressed different aspects of the robot. Each team would have 2 chances to perform each test. Test area A was visible to all team members, but the details of test areas B and C were not. Teams were not permitted to drive on any of the test areas or for that matter much of the AFB. TeamUCF was assigned the test areas in order B, C, A.

Figure 32 illustrates the layout of test area B and also illustrates the simulation and visualization software utilized by TeamUCF. This is a screen display from either inside the vehicle or the simulation, they are identical. The baseball diamond near the center of the figure gives some sense of scale. A series of k-rail launch chutes at the upper left of the figure defined the launch location. The vehicle immediately enters a zone driving area with no specific waypoint information other than an exit goal at the bottom left of the zone. Upon leaving the zone the vehicle must traverse a narrow pathway bounded by k-rails and proceed around a round-about and eventually out into a double cloverleaf road network. At the center of each cloverleaf is another zone area, with the bottom zone modeling a parking lot and requiring a parking maneuver. In the center of each zone, and not shown, were two large circular barricades that were to be detected and avoided. There were numerous other static obstacles on the course to be avoided. There were no moving obstacles. The mission wound through much of the course with the robot required to return back to the starting location completing a course of about 6.5km in 30 min.

The challenges presented by test area B were effectively:

- Navigating a zone with no waypoint information.
- Navigating over a relatively long distance and relatively long time.
- Navigating in the presence of sparse waypoints (note upper right portion of the figure).

- Navigating through a complex field of static obstacles.
- Navigating narrow roads with barriers on either side of the road.
- Navigating stop signs.
- Parking with nearby parking spots occupied by vehicles.

TeamUCF's first attempt at test area B resulted in the robot making it through the majority of the course (~ 5km) in approximately 20 min after successfully navigating the parking maneuver and a "gauntlet" of parked cars. At about this time the sensor fusion algorithm failed due to a software bug in a polygon clipping module. The vehicle effectively lost all forward looking sensors and crashed into one of the barricades resulting in minor damage to a cable. TeamUCF was allowed to restart the vehicle from here but we did not complete the course. Our second attempt at test area B was completed successfully in just under 19 minutes, one of the fastest qualifying times.

Test area B was an amazing awakening event for TeamUCF as we watched the robot leave our sight to enter into the heart of area B. We realized only then that this was the first time, in all of our testing, we had ever let the robot out of our sight.

Fig. 32. Test Area B, driving, parking and obstacle avoidance

Fig. 33. Test area C - Stop sign and rerouting

Test area C (Figure 33) was designed to test stop sign and rerouting behavior. The mission definition file for the robot required a path be taken around the outer loop of the "belt buckle". Each time the robot reached a crossing 4-way stop, a different configuration of cars was presented. The robots objective was to correctly determine precedence and continue the mission in the correct order. After completing a series of loops, the course was adjusted, and a blockage inserted on the bottom loop of the buckle. The robot needed to reroute and determine a path to a checkpoint on the other side of the blockage. This proved problematic for many robots, including initially for TeamUCF. A strict interpretation of rules would imply that such a point is unreachable in normal driving since in would require a u-turn on the far side of the barricade, but a u-turn is only legal on a blocked road, so the robot would have to assume the blockage remained in place. DARPA rules (and in fact actions on this course) indicated that blockages could not be assumed to be static. Upon understanding these new constraints, TeamUCF was able to redefine the routing behavior to make this checkpoint.

One of the more interesting things in this figure is the mis-registration between the ground truth waypoint data and the imagery data provided by DARPA to all

participants. While useful for determining conceptually what a course looked like, this imagery data was of poor enough quality to not be usable for any type of pre-mission planning or environmental modeling. As a note, data from several popular web-based mapping tools is similarly inaccurate.

Test Area A (Figure 34) proved to be the most interesting area, perhaps because it was fully visible to spectators and perhaps because it was specifically designed to promote robot – manned vehicle interaction. The fundamental objective was to have the robot complete as many loops of the right hand side of the course as possible in 30 minutes. Traffic crossed in front of the robot at the top of the course and the vehicle was required to merge into traffic from the stop sign at the bottom center of the course. Traffic speed was relatively modest at ~ 10mph, but traffic configurations were continually altered by the drivers and unless the robot was relatively aggressive none of the maneuvers could be performed without some close calls with the manned vehicles (Figure 35). TeamUCF completed 17 transits of the course in under 30 minutes, and terminated the run early in order to avoid pushing “our luck”. This was among the most laps performed.

Fig. 34. Test area A - Merging and crossing traffic

Fig. 35. Illustration of crossing traffic

5.1 Finals

By successfully completing all three test areas, TeamUCF earned a place in the finals of the Urban Challenge along with 10 other competitors. DARPA narrowed the field of finalists from the initially stated goal of 20 to only 11 competitors.

The Final Event took place on November 3rd, 2007 and was comprised of a series of 3 missions covering a distance of 60 miles through a complex urban environment and driving time-limited to a total of 6 hours. Test tracks A and B of the NQE were incorporated as subsets into the final road network, but the network extended to other new areas with significantly more elevation change than the relatively flat NQE test areas. The 3 missions were designed to demonstrate all of the scenarios previously tested, although in a somewhat less stressing manner, plus some novelties. In addition to some 50 human-driven traffic vehicles, all finalist robots were on the track at the same time, creating for TeamUCF the never tested scenario of encountering live robot traffic. To add a level of complexity, DARPA announced a day before the final event that one section of the urban course will be a steep unpaved road negotiating an elevation difference of 50 meters. TeamUCF was surprised to find off-road performance tested in a contest labeled as "Urban".

During the first 30 minutes of the race Knight Rider behaved as expected, mastering encounters with other robots without any problems (Figure 36) and driving road segments reliably and repeatably. At 9:11 a.m. the robot got stuck at a stop sign, not entering the intersection even when all other pockets were empty. Most likely a misreading by the sensors produced a phantom obstacle in the

intersection essentially deadlocking Knight Rider. The situation resolved itself after a few minutes when another vehicle entered the pocket and cleared the obstacle. At 9:42 a.m. “Little Ben” from the University of Pennsylvania came within a few inches of Knight Rider when switching lanes after passing TeamUCF’s chase vehicle.

After 19.8km (~ 12.3 miles) and a running time of 2 hours, 7 minutes and 20 seconds, the Knight Rider GPS/INS returned a NaN (Not a Number) for the latitude and longitude of the robot. The manufacturer-provided communication library used to read the UDP packages from the GPS reported a non-suspecting “Data Valid” for these values. In the IEEE floating-point standard, any arithmetic operation involving NaN always results in NaN, and any numerical comparison with it fails. This invalid data cascaded through the system, but had the most detrimental impact on the steering controller. In the steering controller, the robot’s position is used to calculate the angle to the carrot point as a set point for the PI controller. The integral part of this controller preserved the value for subsequent iterations essentially locking the value into the system. Confronted with this the digital servo drive locked the steering wheel in the center position, leading the robot to deviate from the road, jumping a curb, and driving towards an abandoned house, eventually stopping in front of a wall where it stayed paused for the next 6 hours (Figure 37) before the team was allowed to recover it. TeamUCF was officially retired from the DARPA Urban Challenge at 10:38 a.m.

Fig. 36. Knight Rider encountering MIT in the traffic circle.

Fig. 37. Knight Rider paused at his final resting place right in front of a house.

6 Discussion

The experience gained during the Urban Challenge competition has been invaluable to advancing TeamUCF's capabilities with autonomous robotic vehicles operating in an urban environment. The performance of a number of teams clearly indicates that commercial autonomous vehicle operation is closer to reality than many expect.

While TeamUCF encountered a series of mechanical issues during the NQE, none of these issues severally hampered performance. The overall competition was not particularly stressful on the robots from a mechanical point of view. Furthermore, all sensors operated within expected performance bounds throughout the competition. This was partially because adverse environmental conditions that could have impacted sensor performance were reduced by the choice of venue and the time of year of testing. The use of relatively simple algorithms, robust simulation tools, and partitioning of the control system in the manner used contributed to the ability to make the few minor modifications that were necessary during the NQE. In hindsight, the choice of the platform for the robot, the choice of sensor systems, and the overall control approach was a good one and one the team would use again.

In many ways the final event was easier than the tests required to be passed to qualify for the finals. The final event focused on a robot's ability to repeatedly perform a series of moderately challenging missions over the course of hours. Teams that had significant experience with long duration tests faired better than those that did not. The criticality of long duration testing can not be

underestimated, but the implication on pre-event test area configuration should be understood. A test area with the ability to drive the vehicle over 10km and with the ability to drive the vehicle for hours without fear of unintended interaction with other vehicles was a deciding factor in determining the outcome of this event. A small test area offered the ability to investigate "scenarios" or demonstrate the robot's ability to meet virtually every individual objective. Testing at a nearby parking lot during evenings while at the NQE allowed specific algorithms associated with Test Area A to be validated and refined. Testing at an off-road site verified the off-road capability of the robot prior to participating in the final event. Several teams took advantage of this scenario testing. Nevertheless, scenario testing was woefully inadequate in verifying the robot's long term performance. In hindsight, TeamUCF might have faired better in the final event by verifying the robot could drive around a circular course for 8 hours straight rather than verifying off road performance.

The criticality of GPS or GPS/INS systems can not be underestimated. On the day prior to the final event, finalists were asked to launch their robots from the starting chutes and make a relatively simple loop course ostensibly to verify starting procedures and timing. A robot that had performed virtually flawlessly in previous tests nearly collided with two other robots that were stopped behind the start line. The team with the malfunctioning robot claimed the issue was traceable to a malfunctioning GPS. On the day of the final event, the pole setting team failed to launch on time again due to a stated GPS malfunction attributed to interference from a large TV screen near the vehicle. TeamUCF's failure to complete the final event is directly attributable to a GPS failure. A significant GPS outage for any team would likely have crippled that team.

Several other items are noteworthy:

- SICK laser scanners are highly reliable and robust measurement devices, but with relatively limited operational range. Certain lighting conditions or obstacles types bring those ranges well below 50m, making these sensors questionable for vehicle speeds much above those demonstrated in this challenge.
- The Oxford RT3000 GPS/INS is highly capable and accurate, however, external data validity checking, separate from the software tools provided by the vendor, must be performed to insure that the rare data error does not have catastrophic consequences.
- TeamUCF struggled with camera-based vision systems primarily because of lighting conditions and obstacles diversity, but other systems made up for these deficiencies.
- ICE worked well for inter-process communications and effectively supported hardware-in-the-loop capability. Coupled with SVN, these open source tools provide an incredible software development environment for robotic systems.
- A large scale test site with essentially unlimited access is essential for adequate testing. Long duration performance can only be adequately tested at such a facility. TeamUCF conducted the majority of its testing on a large open parking deck, at night.

- TeamUCF and many other teams pursued this effort as a competition, with the goal of meeting the specific objectives of the competition. As such, many systems and approaches were tailored to specifically follow the rules as defined by DARPA.
- Money matters, but only if you use it to allow robust vehicle testing.
- NaN fails every test.

Acknowledgements

TeamUCF would like to thank the generous sponsorship of the University of Central Florida College of Engineering and Computer Science, Coleman Technologies, Inc., and for the flexibility of Old Dominion University in providing the time for Dr. Papelis to complete a project he started while at UCF. Furthermore, TeamUCF thanks Richard "Ed" Johnson from the University of Central Florida for his contributions in designing the 3-D laser scanner assemblies.

References

Adams, M.: On-line gradient based surface discontinuity detection for outdoor scanning range sensors. In: Proceedings of 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 1726–1731 (2001)

Barton, M.: Controller development and implementation for path planning and following in an autonomous urban vehicle. Undergraduate thesis, University of Sydney (2001)

Boer, E., Hoedemaeker, M.: Modeling driver behavior with different degrees of automation: A hierarchical decision framework for interactive mental models. In: Proceedings of the 17th European Annual Conference on Human Decision making and Manual Control, Valenciennes, France (1998)

Coulter, R.C.: Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (1992)

Cremer, J., Kearney, J., Papelis, Y.: HCSM: A framework for behavior and scenario control in virtual environments. ACM Transactions on Modeling and Computer Simulation 5(3), 242–267 (1995)

Harper, D., Hua, D.K., Foroosh, D.H., Leonessa, D.A., Qu, D.Z., Pillat, R., Norvell, D., Santiago, S., Collins, T., Stein, G., Stickler, S., Decker, G., Andres, R., Shen, Y., Chen, H., Xie, F.: Technical Paper - DARPA Grand Challenge, Technical report, University of Central Florida (2005)

Michon, J.: A critical view of driver behaviour models: What do we know, what should we do. In: Evans, L., Schwing, R. (eds.) Human behaviour and traffic safety. Plenum Press, New York (1985)

Moravec, H.: Sensor fusion in certainty grids for mobile robots. AI Magazine 9(2), 61–74 (1988)

Papelis, Y., Ahmad, O.: A comprehensive Microscopic Autonomous Driver Model for Use in High-Fidelity Driving Simulation Environments. In: Proceedings of the Transportation Research Board Meeting, Washington, DC (2001)

Qu, Z., Wang, J., Plaisted, C.E.: A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles. IEEE Transactions on Robotics 20, 978–993 (2004)

Surmann, H., Nuechter, A., Hertzberg, J.: An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. Robotics and Autonomous Systems 45, 181–198 (2003)

Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. In: Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge (2005)

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stand, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics: Special Issue on the DARPA Grand Challenge, Part 2 23(9), 661–692 (2006)

Wulf, O., Wagner, B.: Fast 3d-scanning methods for laser measurement systems. In: International Conference on Control Systems and Computer Science (2003)

Yang, J., Daoui, A., Qu, Z., Wang, J., Hull, R.: An optimal and real-time solution to parameterized mobile robot trajectories in the presence of moving obstacles. In: IEEE International Conference on Robotics and Automation, Barcelona, Spain, April 18-22, 2005, pp. 4423–4428 (2005)

Team AnnieWAY's Autonomous System for the DARPA Urban Challenge 2007

Sören Kammel[1], Julius Ziegler[1], Benjamin Pitzer[1], Moritz Werling[2], Tobias Gindele[3], Daniel Jagzent[3], Joachim Schöder[3], Michael Thuy[4], Matthias Goebl[5], Felix von Hundelshausen[6], Oliver Pink[1], Christian Frese[3], and Christoph Stiller[1]

[1] Institute for Measurement and Control
University of Karlsruhe
76131 Karlsruhe, Germany
[2] Institute for Applied Computer Science/Automation
University of Karlsruhe
76128 Karlsruhe, Germany
[3] Industrial Applications of Informatics and Microsystems
University of Karlsruhe
76131 Karlsruhe, Germany
[4] Institute for Distributed Measurement Systems
Technical University of Munich
80290 Munich, Germany
[5] Institute for Institute for Real-Time Computer Systems
Technical University of Munich
80290 Munich, Germany
[6] Department of Aerospace Engineering
University of the Federal Armed Forces
85577 Neubiberg, Germany

Abstract. This paper reports on AnnieWAY, an autonomous vehicle that is capable of driving through urban scenarios and that has successfully entered the finals of the *2007 DARPA Urban Challenge* competition. After describing the main challenges imposed and the major hardware components, we outline the underlying software structure and focus on selected algorithms. Environmental perception mainly relies on a recent laser scanner which delivers both range and reflectivity measurements. While range measurements are used to provide 3D scene geometry, measuring reflectivity allows for robust lane marker detection. Mission and maneuver planning is conducted using a concurrent hierarchical state machine that generates behavior in accordance with California traffic laws. We conclude with a report of the results achieved during the competition.

1 Introduction

The capability to concurrently perceive a vehicle's environment, to stabilize its motion and to plan and conduct suitable driving maneuvers is a remarkable competence of human drivers. For the sake of vehicular comfort, efficiency, and safety, research groups all over the world have worked on building

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 359–391.
springerlink.com

autonomous technical systems that can in part replicate such capability [Bertozzi et al., 2000], [Franke et al., 2001], [Nagel et al., 1995], [Thorpe, 1990], [Dickmanns et al., 1994].

The *DARPA Urban Challenge 2007* has been a competition introduced for expediting research on this kind of systems. Its finals took place on Nov. 3rd, 2007 in Victorville, CA, USA. As in its predecessors, the Grand Challenges of 2004 and 2005 [DARPA, 2005], [Thrun et al., 2006], the vehicles had to conduct missions fully autonomously without intervention of human team members (see Fig. 1). In contrast to the earlier competitions, the Urban Challenge required operation in a mock urban scenario, including traffic made up from both competing autonomous vehicles and human driven cars. The major challenge imposed was collision-free driving in traffic in compliance with traffic rules (e.g. right of way at intersections) while completing the given mission. This required for passing parked cars, performing U-turns, parking, and merging into regular flow of traffic. Finally, recovery strategies had to be demonstrated in deadlock situations or in traffic congestions that cannot solely be handled by strictly following traffic rules.

Fig. 1. AnnieWAY stopping at an intersection on track A during the NQE.

The scope of Team AnnieWAY was to extract early research results from the *Cognitive Automobiles* project that would allow real-time operation of the vehicle under the restricted traffic environment in the Urban Challenge. Its team members are professionals in the fields of image processing, 3D perception, knowledge representation, reasoning, real time system design, driver assistance systems and autonomous driving. Some of the team members were in the 'Desert Buckeyes' team of Ohio State University and Universität Karlsruhe (TH) and developed the 3D vision system for the Intelligent Off-road Navigator (ION) that traveled successfully 29 miles through the desert during the Grand Challenge 2005 [Özgüner et al., 2007], [Hummel et al., 2006].

2 Hardware Architecture

The basis of the AnnieWAY automobile is a VW Passat Variant (see Fig. 2). The Passat has been selected for its ability to be easily updated for drive-by-wire use by the manufacturer.

2.1 Computing System

AnnieWAY relies on an off-the-shelf quad-core computer offering enough processing capacity to run all required software components for perception, situation assessment, and trajectory generation. The chosen hardware architecture is optimally supported by the real-time-capable software architecture which is described in Sec. 3.

The main computer is augmented by an electronic control unit (ECU) for low-level control algorithms. It directly drives the vehicle's actuators. Both computer systems communicate over an Ethernet link. The drive-by-wire system as well as the car odometry are interfaced via the Controller Area Network (CAN) bus. The DGPS/INS system allows for precise localization and connects to the main computer and to the low-level ECU.

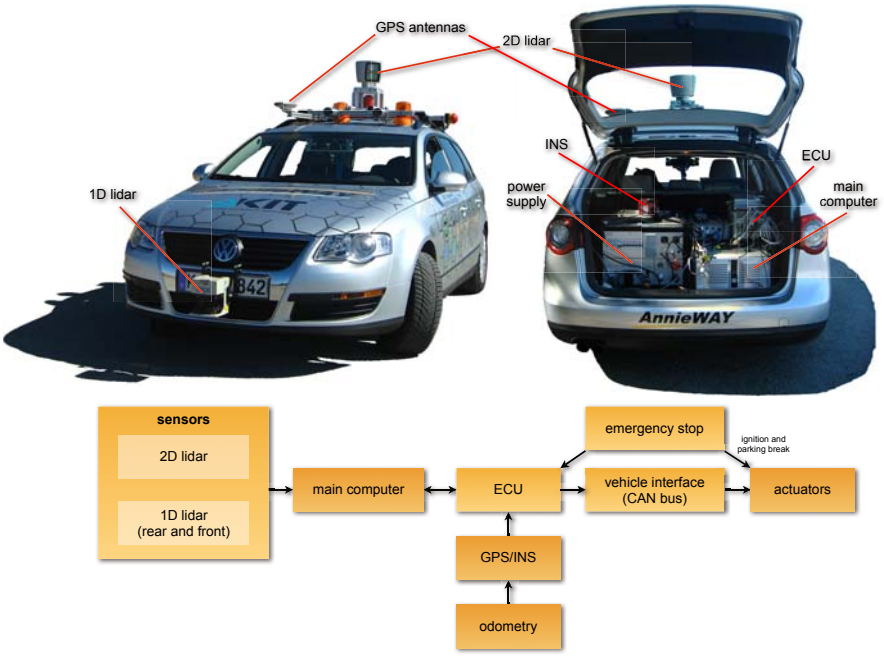


Fig. 2. Architecture and hardware components of the vehicle.

2.2 Laser-Based Range and Intensity Sensors

Since LIDAR units produce their own light, low light conditions have no effect on this kind of sensor. In our car we use a rotating laser scanner comprising 64 avalanche photo diodes that are oriented with constant azimuth and increasing elevation covering a 26.5° vertical field of view. The lasers and diodes are mounted on a spinning platform that rotates at a rate of 600 rpm. Thus, the LIDAR provides a 360° field of view around the vehicle producing more than 1 million points per second at an angular resolution of 0.09° horizontally and a distance resolution of 5 cm with distances up to 100 m. The result is a dense, highly accurate scan representation of almost the entire scene surrounding the vehicle. For each point, the sensor measures range and reflectivity. The reflectivity map is well suited for monoscopic image analysis tasks like lane marker detection. The inherent association of each reflectivity pixel with a range measurement alleviates information fusion of these data significantly. For parking maneuvers, the main LIDAR is supported by two 2D laser scanners that cover the area directly in front and behind the vehicle.

2.3 DGPS/INS

A precise localization is provided by a dead reckoning system which consists of an advanced six-axis inertial navigation system with an integrated RTK/GPS receiver for position and a second GPS Receiver for accurate heading measurements. Odometry is taken directly from AnnieWAY's wheel encoders. The dead reckoning system delivers better than 0.02 m positioning accuracy under dynamic conditions using differential corrections and 0.1° heading accuracy using a 2 m separation between the GPS antennas.

2.4 Emergency Stop System

As the vehicle had to operate unmanned, a wireless stop system has been integrated for safety reasons as required by DARPA. This E-Stop system allows to remotely command run-, pause-, or emergency-stop mode. The system is connected directly to the ignition and the parking brake to assert appropriate emergency stop regardless of the state of the computer system. Run and pause mode are signaled to the low-level control computer.

3 Software Architecture

The core components of the vehicle are the perception of the environment, an interpretation of the situation in order to select the appropriate behavior, a path planning component and an interface to the vehicle control. Fig. 3 depicts a block diagram of the information flow in the autonomous system. Spatial information from the sensors is combined to a static 2D map of the environment. Moving objects are treated differently. Such dynamic objects also

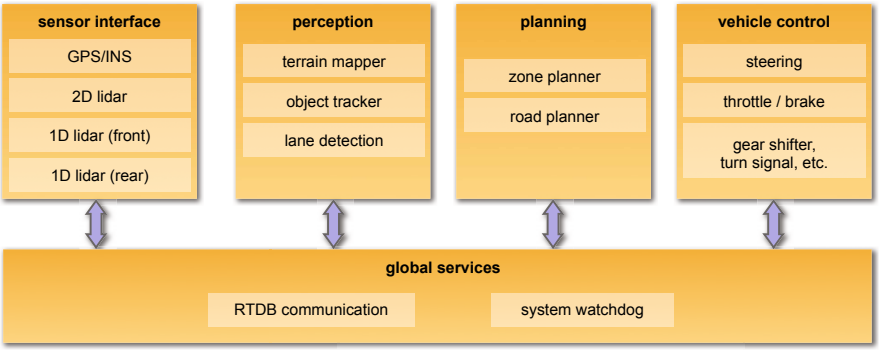


Fig. 3. Overview of the software architecture and the information flow.

include traffic participants that are able to move but have zero velocity at the moment. To detect moving objects, the spatial measurements of the LIDAR sensor are clustered and tracked with a multi-hypothesis approach. To detect possibly moving objects, a simple form of reasoning is used: If an object has the size of a car and is located on a detected lane, it is considered to be probably moving. Lane markings are detected in the reflectance data of the main LIDAR. Together with the road network definition file (RNDF), the absolute position obtained from the dead reckoning system and the mission data file (MDF), this information serves as input for the situation assessment and the subsequent behavior generation. Most of the time, the behavior will result in a drivable trajectory. If a road is blocked or the car has to be parked, modules for special maneuvers, like the parking zone navigation module, are activated.

All data exchange between processes is done via a central communication framework, the real-time database for cognitive automobiles *KogMo-RTDB* [Goebl and Färber, 2007b]. All data within the RTDB is represented as time-stamped objects. The centralized data storage gives the opportunity to easily log and replay all or selected objects. For performance reasons the database is completely memory based. It is capable of distributing even large data objects, like LIDAR raw sensor data, to several processes and at the same time relay vehicle control commands at a rate of 1 kHz between a vehicle control process and the ECU [Goebl and Färber, 2007a].

4 Perception

4.1 Environmental Mapping

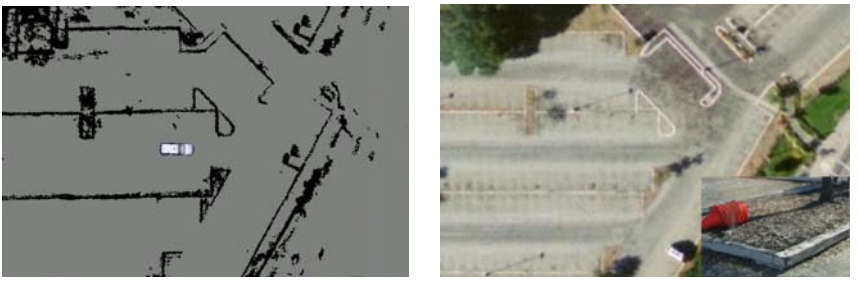
Accurate and robust detection of obstacles at a sufficient range is an essential prerequisite to avoid obstacles on the road and in unstructured environments like parking lots. The basic idea is to maintain an evenly spaced 2D grid structure g where each cell g_i represents a random variable. Each

random variable is binary and corresponds to the occupancy it covers. Therefore, in the literature this approach is also called *occupancy grid mapping* ([Thrun, 2002], [Thrun, 2003]) which has the goal to calculate the posterior over maps $p(g|\mathbf{z}, \mathbf{x})$ where $\mathbf{z}$ is the set of all measurements and $\mathbf{x}$ is the path of the vehicle defined through a sequence of poses. An example of a resulting evidence map is depicted in Fig. 4.

Fig. 4. Example for the evidence mapping of 3D LIDAR data onto a 2D grid. Darker spots correspond to high evidence for an obstacle while white cells correspond to drivable area. Unknown cells are marked as grey.

AnnieWAY uses a grid that is always centered at the vehicle position but aligned with a global coordinate system. The grid is shifted at each time step to account for the new vehicle position. This restricts the size of the map to an area around the vehicle while the cells are bound to an absolute position. The size of each grid cell is 15cm×15cm. Fig. 5 shows an example of our mapping algorithm. The grid is generated mainly from multi-layer, high resolution LIDAR data. Algorithms for the integration of low resolution LIDAR data can be found in [Thrun, 2002], [Thrun, 2003], [Biber and Strasser, 2006], [Bosse et al., 2003].

Integrating the data of the laser scanners into an environmental map consists of three steps. In the first step the range measurements $z_{l \in L}$ of one revolution L are projected into a global coordinate system under consideration of the vehicle's motion x_l. In the second step, different measures are



(a) Map generated from a parking lot.

(b) Aerial imagery of the parking lot with a detail photo of the curb in the lower right corner.

Fig. 5. Example for a generated evidence map and an aerial image of the corresponding region.

extracted from the data for each cell g_i. Two straightforward measures are the number of measurements n_i and the number of different laser beams b_i. The most important measure we use is the elevation difference

$$e_i(g_i, z_l) = \max_{l \in L} h(g_i, z_l) - \min_{l \in L} h(g_i, z_l) \ , \tag{1}$$

where h is the vertical component of each measurement.

In the third step, we compute the evidence for each measure by using an inverse sensor model. E.g. the inverse sensor model for the elevation difference returns l_{occ} if e_i exceeds a certain threshold (e.g. 15cm) and l_{free} otherwise. The inverse models for n_i and b_i are slightly more complex since they are learned by a supervised learning algorithm. The result of the learning procedure is a forward model that accepts g_i and n_i or b_i respectively as parameters and returns the appropriate evidence.

Finally, we can compute the combined occupancy evidence $o_{i,t}$ as a weighted sum of the three partial evidences:

$$o_{i,t} = o_{i,t-1} + \alpha_1 \cdot n_i + \alpha_2 \cdot b_i + \alpha_3 \cdot e_i \ , \tag{2}$$

and the estimated occupancy for a single cell

$$p(g_i|\mathbf{z}, \mathbf{x}) = 1 - \frac{1}{1 + \exp o_i} \ . \tag{3}$$

As already mentioned, AnnieWAY is equipped with different sensors and ideally one wants to integrate information from all sensors into a single map. A naive solution is to update the map for each sensor separately which neglects the different characteristics of each sensor, e.g. field of view, maximal range and noise characteristic. To ensure safe driving we use the most pessimistic approach to fuse sensor data: We compute the maximum of all estimated occupancies, where K is the number of sensors:

$$p(g_i) = \max_{k \in K} p(g_i^k) \tag{4}$$

If any sensor detects a cell as occupied it will be occupied in the combined map.

The standard occupancy grid mapping algorithm suffers from a major drawback: It is only suitable for static environments. Driving environments are typically highly dynamic and the result is very poor without modifications. Moving objects create virtual obstacles with high evidence while moving. To overcome this problem we introduce a temporal evidence decay. The evidence is reduced at each time step by a factor ϵ_t for cells which are not updated. The intuition is that the uncertainty increases for cells not augmented by any sensor. Equation 2 turns now into

$$o_{i,t} = \text{argmax}(0\ , o_{i,t-1} + \alpha_1 \cdot n_i + \alpha_2 \cdot b_i + \alpha_3 \cdot e_i - \epsilon_t)\ , \tag{5}$$

where the argmax operator enforces positive evidences.

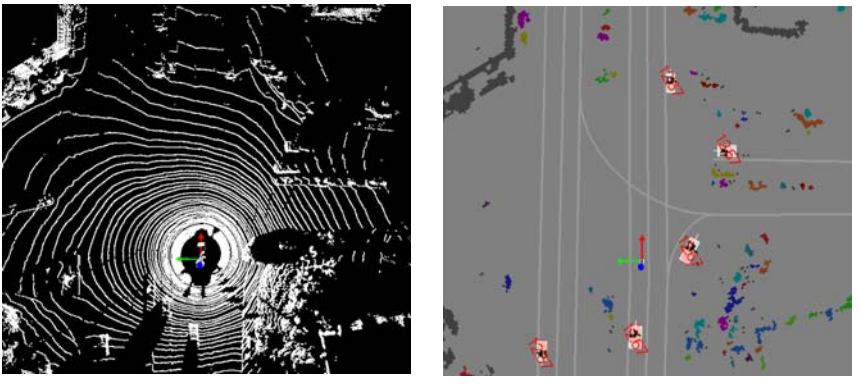
5 Tracking of Dynamic Objects

Driving in urban environments requires to capture and estimate the dynamics of other traffic participants in real-time. AnnieWAY uses a processing pipeline that takes raw sensor data (from different lasers) and generates a list of dynamic obstacles, along with their estimated locations, sizes, and relative velocities. This pipeline consists of a number of parts, including

1. **Data preprocessing:** Removing irrelevant readings: noise, ground readings, readings from obstacles outside the road, etc.
2. **Obstacle detection:** Creating a list of obstacles raw readings...includes segmentation for laser
3. **Obstacle tracking:** Corresponding obstacles time step with those of another time step in order to determine their headings, relative velocities, etc.
4. **Obstacle post-processing and publishing**

The data preprocessing step used for tracking was discussed earlier as part of Sec. 4.1. The result of this part is a grid map with occupancy probabilities attached to each cell. All the sensors' information has been condensed within this grid.

The first stage of dynamic object tracking is the object detection which is—in the sense of a statistical approach—equivalent to the identification of object hypotheses. AnnieWAY uses an occupancy grid map which has been segmented using a connected components approach. Therefore, we treat each grid cell as a node in a graph G. Two points are connected if and only if the distance between them is within a threshold d (e.g. 0.5 m). We then find all the connected components in the graph and assign the same label to those cells. To reduce noise, we discard any connected component with less



(a) High resolution LIDAR data.

(b) Tracked vehicles on segmented grid map.

Fig. 6. Tracking of dynamic objects with occupancy grid map and linear Kalman filter.

than a minimum number of cells. Due to the uniform angle resolution of the scanners, the number of cells an object consists of depends on its distance. The closer an object is located to the scanner, the more laser rays will hit the object.

The connected components are analyzed in a second step for their probability of being a traffic participants. Several heuristics are used based on their shape and location relative to the road network. Only 'good' candidates are augmented in the following tracking step. Fig. 6(b) displays the resulting objects after post-processing.

With this procedure, not all captured and tracked objects are relevant to be published to other modules. This is due to noisy observations, occlusion, dynamic objects leaving of our sensors' fields of view, etc. All these effects lead to unlikely object hypotheses, but nevertheless they are internally tracked. In order to decide when to publish relevant obstacles, we define a notion of confidence that works similarly to log-likelihood updates in an occupancy grid map as mentioned earlier. If an obstacle is observed, we increment its confidence, in case it goes unobserved in our field of view, we decrement it. Thus defined, the confidence allows us to set minimum thresholds for the tracking and publishing obstacles: if the object's confidence exceeds the threshold, the obstacle is published to all other attached modules. If its confidence undercuts a certain threshold, the object is removed from the obstacle list. Hypotheses within both thresholds are internally tracked, but not published.

Tracking of dynamic objects mainly serves two purposes. First, it aids the correspondence of obstacles detected in one sensor frame at time $t = k$ with those in subsequent sensor frames at time $t = k + 1$. This can be easily achieved with distance-based methods or more sophisticated 3D fitting and registration algorithms like iterative closest point (ICP). However, these

methods do not take into account the noise and uncertainty of our sensors. The second and equally important purpose of tracking is to return estimates of other vehicle's relative velocities and headings.

AnnieWAY uses a linear Kalman filter [Kalman, 1960] to model a simplified dynamic obstacle with its appropriate state vector $\left[x, y, \dot{x}, \dot{y}\right]^T$. Obviously, this model ignores completely the underlaying physical and non-linear behavior of a car, but the frequency of sensor updates (10 Hz) means that cars move very little between them which allows us to assume linear dynamics. Transition updates are linear with an overlaid Gaussian noise characterized by its covariance matrix $\mathbf{Q}$:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{q,\dot{x}}^2 & 0 \\ 0 & 0 & 0 & \sigma_{q,\dot{y}}^2 \end{bmatrix}. \tag{6}$$

Since we are extracting the obstacle's position $\left[x, y\right]^T$ from the measurement, the observation matrix $\mathbf{O}$ looks as described below. Further, we assume mutual independent Gaussian noise sources characterized by the covariance matrix $\mathbf{R}$:

$$\mathbf{O} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} \sigma_{r,x}^2 & 0 \\ 0 & \sigma_{r,y}^2 \end{bmatrix}. \tag{7}$$

After performing an observation, we do not know which detected obstacles within the measurements are already tracked or if they are new objects. Thus, we are required to solve a problem of correspondence between observations and the internally tracked dynamic obstacles. This is a nontrivial problem, requiring that we define both a measure of distance and a procedure for finding the optimal correspondence. AnnieWAY uses a maximum-likelihood matching algorithm to find the optimal assignment of observations to existing Kalman filters. This matching is a one-to-one function from filters to observations.

6 Lane Marker Detection

Digital maps of a road network are often not up-to-date or resemble the real road network only approximately. Therefore, a local offset between the digital and the real road network may exist. The detection of lane markings helps to minimize this offset. An accurate and continuous detection of lane markings even enables the creation of new road network maps.

In the context of this paper, lane markings can be either painted markings or curbs. Painted lane markings are detected within the intensity readings of the LIDAR whereas curbs cause small height changes in the range data of the LIDAR. A combined intensity/range plot is depicted on the left side of

Fig. 8. Both kind of lane markings form one dimensional structures that can be approximated by line segments locally. In contrast to camera based intensity images, the laser reflectivity and range data is insensitive to background light and shadows. However, the sensor samples the road very sparsely, especially at distance. In order to increase the density of lane marker information, subsequent scans are registered spatially and accumulated employing absolute positioning information from the dead reckoning system. The first step in order to obtain a dense bird eye's view representation of lane marker features is a classification of data points in each scan into obstacle and ground by the algorithms described in Sec. 4.1. Lane markings are expected to occur on the road surface (painted markings) or at its borders (curbs) only. Therefore, points of each individual laser labeled as ground are searched for large continuous chunks (chunks that do not exhibit height changes exceeding the height of curbs) representing the road. Only within those large chunks high intensity gradients are detected. In addition, only measurements exhibiting absolute intensities larger than the median intensity of each laser scan are taken into account. Both types of features – painted markings and curbs – are mapped into a feature grid $g(\mathbf{x})$ similar to the evidence grid described in Sec. 4.1, see Fig. 8(right). Features are detected first in the single scans and mapped afterwards (instead of creating a dense map first and extracting the features afterwards) to minimize the effect of errors in the vehicle localization. A summary of the detection algorithm is shown in Fig. 7.

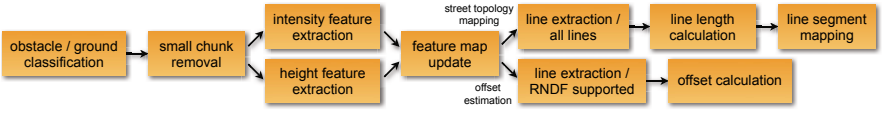


Fig. 7. Overview of the offset estimation and street topology mapping.

Lane segments are detected by applying the Radon transform to the accumulated feature map data. Since the Radon transform is an algorithm operating globally on the map it proved to be robust against occlusions, noise and outliers. Compared to the Hough transform, the Radon transforms exhibits the advantage of a calculation time independent of the numbers of lane markings and the capability to handle gray-scale images efficiently and without thresholding. For a real-time calculation in the car, an implementation exploiting the central-slice theorem was used [Bracewell, 1990]. The position and direction of lane boundaries can be calculated by locating their corresponding maxima in the Radon plane. Since we observed a systematic error of RNDF data in some areas, it appeared sensible to determine a correcting offset from the detected lane markings. To accomplish this, the lane markings specified in the RNDF are first projected into the Radon plane. Assuming that the offset of the road map data does not exceed one lane width, the deviation is obtained in a second step from the distances to the maxima in

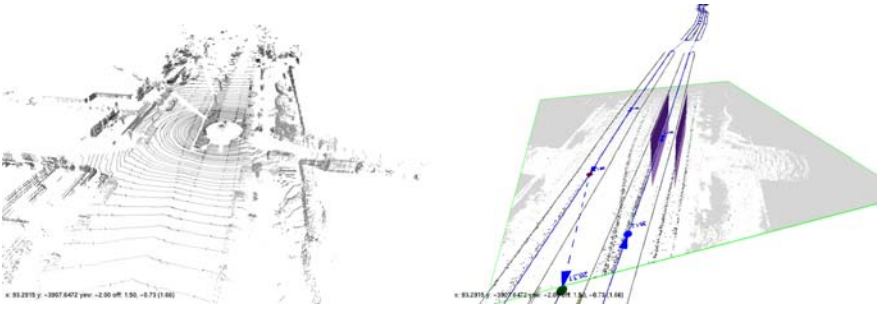


Fig. 8. Combined range and intensity readings of the LIDAR (left) and lane marker map with the estimated current lane segment and an overlay of a part of the original road network map (right).

the Radon plane closest to the predicted positions. Assuming further that predicted and estimated lane boundaries are close to parallel, the vertical distance is sufficient to determine the offset.

7 Reactive Layer

Our system integrates a reactive layer that allows AnnieWAY to modify a planned trajectory based on GPS waypoints. While the obstacle tracker easily handles objects like cars, small or extended objects like rocks or pavement edges are more difficult to track explicitly. Hence, we integrated a reactive mechanism that gets as input a vehicle-centered occupancy grid (built from the LIDAR data) and the trajectory planned so far. The algorithm then first evaluates, whether the given trajectory is clear and - only if not - starts a more complex evaluation of the grid that results in a modification of the initially given trajectory. This mechanism is biologically motivated and resembles an insect's use of its antennae to avoid obstacles. What are antennae in nature, are precomputed trajectory primitives (we call them *tentacles*) in our system. Here, all tentacles are simple circular arcs, but depending on the speed of the vehicle, the parameters of these arcs vary such that at high speeds no dangerous actions can be taken (see Fig. 9). To select the appropriate primitive the occupancy grid is investigated in an area around and underneath that primitive. The final selection is done on the basis of four aspects:

1. Could the vehicle drive the primitive without causing damage? In particular, within a distance the vehicles needs to stop, is the ground along the tentacle clear of anything having a height above 0.1m?
2. How smooth is the terrain under the primitive?
3. How far is the next obstacle along that primitive?
4. How well does the primitive follow the original trajectory?

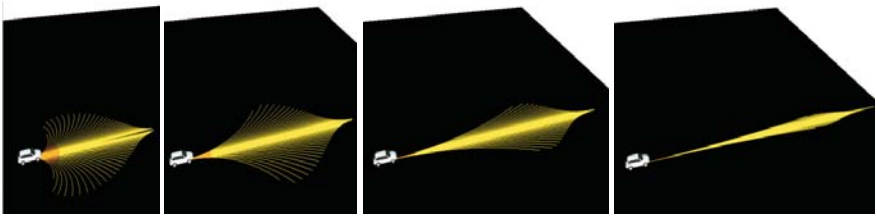


Fig. 9. The reactive system uses a precomputed set of motion primitives that vary with the speed of the vehicle. As detailed in [v. Hundelshausen et al., 2008], those primitives are used to evaluate a vehicle-centered occupancy grid to avoid obstacles.

By considering these aspects as detailed more precisely in [v. Hundelshausen et al., 2008] the vehicle follows the given trajectory if possible, but avoids obstacles, if not. To coordinate this reactive layer with the obstacle tracker, tentacles were only evaluated up to the first explicitly tracked obstacle. In this way, only unexpected obstacles were avoided.

As detailed in [v. Hundelshausen et al., 2008] the overall reactive mechanism was tested excessively by intentionally defining bad GPS-trajectories, e.g having a large offset to the real road (passing through the front gardens of neighboring houses), passing through a traffic circle (instead of leading around it), abbreviating a crossing through a complete house, and other tests including moving vehicles. At the final of the urban challenge this mechanism was important at narrow passages.

8 Planning

The major challenge imposed by the competition was collision-free driving in traffic in compliance with traffic rules, e.g. right-of-way at intersections. It included special maneuvers, like overtaking, U-turns, parking, and merging into regular flow of traffic while completing the given missions. To accomplish this, the robot must be capable of analyzing the situation, assessing developments, choosing the appropriate behavior and executing it in a controlled way. AnnieWAY uses a planning module organized in three layers to address these problems:

1. **Mission Planning** computes a strategic plan to accomplish mission
2. **Maneuver Planning** applies California traffic rules and plans actual driving maneuvers (e.g. turns, intersection, passing) and generates a corresponding path.
3. **Collision Avoidance** tests whether the planned path is collision free taking into account the obstacle map acquired from the perception module. If a collision is probable it chooses an alternative path.

In a first preprocessing step, all elements of the RNDF (lanes, checkpoints, exits, etc.) are converted to a graph-based, geometrical representation. RNDF

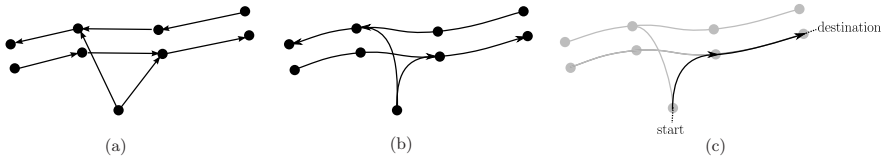


Fig. 10. Preprocessing of RNDF data. (a) Geometric graph derived directly from RNDF. (b) Smoothing with splines. (c) Complete path for mission. It has continous curvature.

waypoints form the vertices of the graph; lanes and exits are represented by graph edges (Fig. 10(a)). Edges yield a geometric representation by smoothing them by spline interpolation (Fig. 10(b). Information such as distances, lane boundaries, and speed limits annotate the graph edges. These annotations can be updated dynamically, to incorporate results from the perception module (e.g. road blockages).

Dynamic objects recognized by the perception module are matched to the most probable edge of the geometrical graph representation, based on their position and orientation. This allows for attributing a role to every object, e.g. identification of a leading vehicle, or semantically localizing an object within an intersection scenario.

Mission Planning is the most abstract form of planning used by AnnieWAY. It finds the optimal route from one checkpoint to another using an optimal graph search algorithm operating on the geometric graph representation of the road network. The criterion that is minimized by the search process is travel time. The search process is repeated for every pair of subsequent checkpoints in the MDF. In this way the mission planner finds the optimal route traversing all mission checkpoints. It is a piecewise-defined spline curve, as shown in Fig. 10(c). Generally the mission planner runs only once while loading the mission file and whenever AnnieWAY has to diverge from the planned route caused by situation dependent reasons (e.g. blocked roads). The route is passed to the downstream maneuver planning.

The high-level plan and the AnnieWAY's current position is used by *Maneuver Planning* to compute actual driving maneuvers. The maneuver planner is implemented as a *Concurrent Hierarchical State Machine (CHSM)* with every state representing a driving behavior. The key aspect of a *hierarchical* state machine is to design and group the states in a way that a sub-state is a specialization of its parent state, and only extensions to the more general behavior of the parent state have to be modeled explicitly. Thereby, the functional redundancy of the states and the amount of transitions is reduced, so it is easier to capture the complex reactional behavior of a system. Fig. 11 shows the UML state chart of the machine's main level, with important sub-states annotated as well.

Every behavior the car is capable of, is modeled as a state organized within a state hierarchy. The state *Drive* comprises all regular driving maneuvers

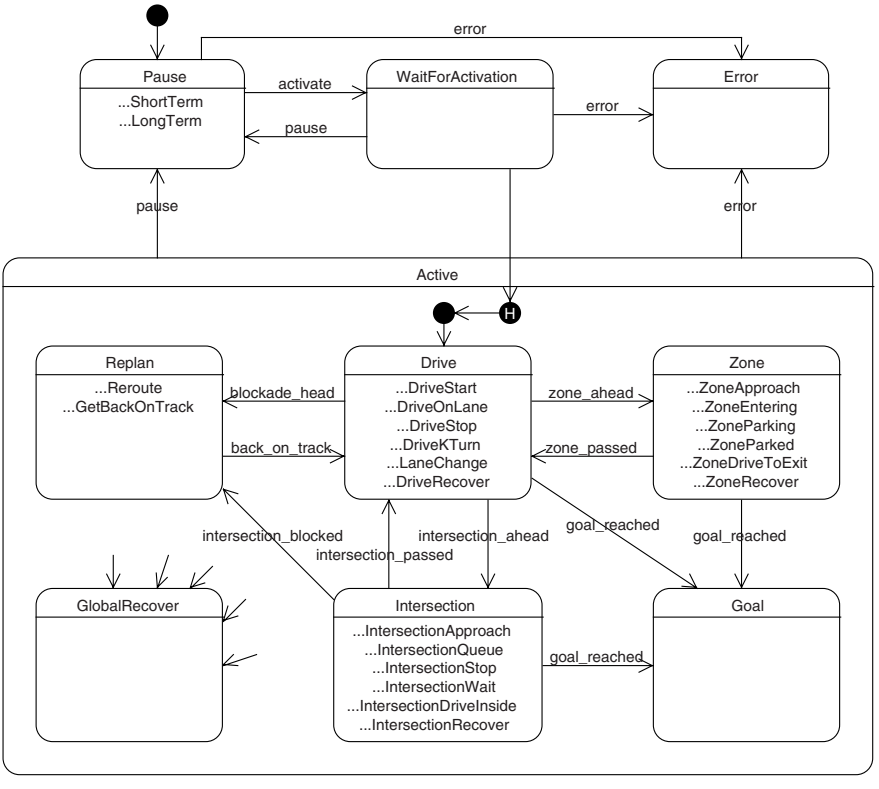


Fig. 11. Overview of the concurrent hierarchical state machine used to model traffic situations and behavior.

on normal roads. It has several sub-states that cover different situations like following the course of a lane (*DriveOnLane*), making a k-turn (*DriveKTurn*) or changing the lane (*LaneChange*). All behavior at intersections is handled by the *Intersection* state. It comprises some specialized sub-states for different types of intersections. Some more insight on the real functionality and architecture of the state machine is given in Sec. 9, where handling of moving traffic in an intersection scenario is explained in detail. The navigation in unstructured environments and parking maneuvers is controlled by the state *Zone* and its sub-states. These states control invocation of the navigation module described in Sec. 10. In some situation it becomes necessary for the robot to replan its route, e.g when the road ahead is blocked. This is triggered by the state *Replan*, that re-activates the mission planning module. Most states implement a recovery state that is activated whenever the car makes no progress at all for a certain amount of time. If all situation dependent recovery handling fails, a global recovery state is invoked to navigate back on track using the navigation module.

When all situation assessment has taken place and all state transitions are made, the reached state generates a path stub, that is input to the closed loop control module (Sec. 11). It reaches approximately 30 m ahead and consists of densely sampled waypoints combined with heading and curvature information. In the most common case, when the car is driving on roads stored within the graph representation, the trajectory is generated in a straight forward way by sampling the graph edges ahead. These points are smoothed by a spline approximation to generate a continuous curvature path. In areas that lack road geometry description and whenever sensible localization within the road network graph is not possible, the free navigation module in Sec. 10 is used to plan a collision free path to a given target configuration.

Paths generated by the state machine may be overwritten by the low level avoidance system described in Sec. 7.

9 Moving Traffic

This section describes an algorithm which reduces dynamic maneuvers, such as merging into moving traffic and crossing intersections with oncoming traffic, to static maneuvers, such as simple turns. Unfortunately, the actual behavior of the other traffic participants cannot be exactly predicted. Therefore certain assumptions, simplifications, and conservative estimates have to be made in an appropriate way, such that the unmanned vehicle operates safely as well as effectively.

9.1 Problem Abstraction and Simplifications

In the following, it is assumed that (1) the other traffic participants with the right-of-way neither slow down nor speed up, (2) stay in the middle of the road, (3) AnnieWAY's longitudinal controller accelerates at a known constant rate until the desired maneuver velocity is met, and (4) all traffic participants' velocities and positions are known.

Assumption (1) and (2) have to be made, since the actual behavior of the other vehicles (B_i) cannot be precisely predicted. Therefore it is assumed, that the considered vehicles travel at a constant velocity in the center of the priority road. Introducing $t_{\overline{BP}}$ as the time needed for traveling a distance $d_{\overline{BP}}$ in the road center and v_B as the other vehicle's constant velocity, leads to

$$t_{\overline{BP}} = \frac{d_{\overline{BP}}}{v_B}. \tag{8}$$

Assumption (3) is based on the longitudinal control strategy, which is described in Sec.11. The resulting drive-off characteristic $v(t)$ from a start velocity v_0 to a new desired velocity v_d can be seen on the left in Fig.12 as a dashed line along with the approximation $\hat{v}(t)$ as a solid line.

Here t_{sw} denotes the time, when the approximated velocity $\hat{v}(t)$ reaches v_d. It can be calculated by

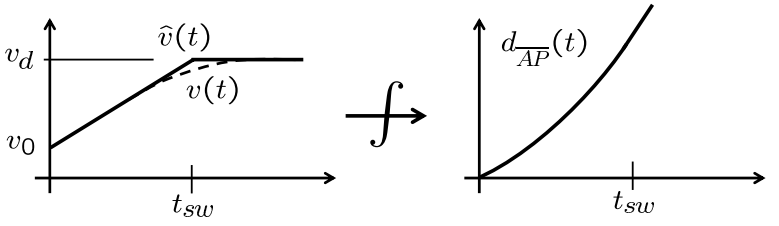


Fig. 12. Actual and approximated drive-off characteristic.

$$t_{sw} = \frac{v_d - v_0}{a_{\text{sat}}}. \tag{9}$$

An integration of $\hat{v}(t)$ over time (see Fig.12) yields the traveled distance of AnnieWAY (A)

$$d_{\overline{AP}}(t) = \begin{cases} v_0 t + \frac{a_{sat}}{2} t^2 & t \leq t_{sw} \\ v_0 t_{sw} + \frac{a_{sat}}{2} t_{sw}^2 + v_d(t - t_{sw}) & t > t_{sw}, \end{cases} \tag{10}$$

Solving (10) for t with

$$t_a = \frac{d_{\overline{AP}} - v_0 t_{sw} - \frac{a_{sat}}{2} t_{sw}^2}{v_0 + a_{sat} t_{sw}} + t_{sw} \tag{11}$$

yields

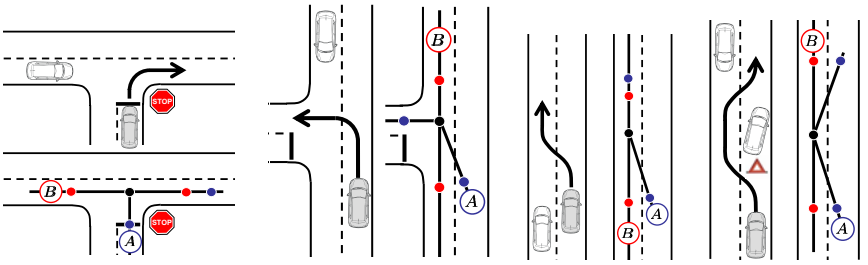
$$t_{\overline{AP}} = \begin{cases} t_a & t_a > t_{sw} \\ \frac{1}{a_{sat}}(-v_0 + \sqrt{v_0^2 + 2 a_{sat} d_{\overline{AP}}}) & t_a \leq t_{sw}, \end{cases} \tag{12}$$

whereas the ambiguity of the solution was resolved.

Fig.13 illustrates the transfer of different traffic scenarios to the equivalent graphs, whose generic graph can be found left in Fig.14 along with the four relevant quantities to be measured, the current distances $d_A(t)$ and $d_B(t)$ to MP, and the current velocities $v_A(t)$ and $v_B(t)$ (assumption (4)). As can be seen, traffic participants are all assumed to be point masses. Based on the previous equations and graphs, the movement of the vehicles can be predicted and used for collision detection in the next section.

9.2 Spatial and Temporal Verification

On the one hand, at low speed it has to be guaranteed that the autonomous vehicle avoids collisions by not getting too close to other traffic participants. Therefore spatial safety distances were introduced (see Fig.14, right-hand side). On the other hand, spatial safety distances are not a proper measure at higher speeds. In this case a temporal safety distance assures certain time gaps between AnnieWAY and the other traffic participants. Since time gaps become too small referred to the ground at low speed in turn, both spatial and temporal conditions have to be fulfilled at the same time.



(a) Right turn with stopping (b) Left turn without stopping (c) Lane change maneuver (d) Double lane change maneuver with oncoming traffic.

Fig. 13. Different moving traffic scenarios

For the sake of simplicity only a single vehicle is considered initially. In order to be the first to enter the critical area, the following two conditions have to be met:

1. At time $t_{\overline{BP}_{B1}}$, when B reaches P_{B1}, A has to be beyond P_{A2}.

$$free_{\text{spat},AB} = (d_{\overline{AP}}(t_{\overline{BP}_{B1}}) > d_A + D_{A2}) \tag{13}$$

2. After A has passed MP, a given time span ΔT_{AB} has to elapse, before B reaches MP.

$$free_{\text{temp},AB} = (t_{\overline{BMP}} > t_{\overline{AMP}} + \Delta T_{AB}) \tag{14}$$

In order to be the second to enter the critical area, the following two conditions have to be met:

1. At time $t_{\overline{BP}_{B2}}$, when B reaches P_{B2}, A may not have passed P_{A1} yet.

$$free_{\text{spat},BA} = (d_{\overline{AP}}(t_{\overline{BP}_{B2}}) < d_A - D_{A1}) \tag{15}$$

2. After B has passed MP, a given time span ΔT_{BA} has to elapse, before A reaches MP.

$$free_{\text{temp},BA} = (t_{\overline{AMP}} > t_{\overline{BMP}} + \Delta T_{BA}) \tag{16}$$

This means if

$$free = \ (free_{\text{spat},AB} \wedge free_{\text{temp},AB}) \ \vee \ (free_{\text{spat},BA} \wedge free_{\text{temp},BA})$$

is true, it is assured that neither A is between P_{A1} and P_{A2} as long as B is between P_{B1} and P_{B2} nor the time gaps in MP are shorter than permitted.

The extension from a single vehicle B to n vehicles B_i is straightforward As long as one vehicle fails the verification, A is not allowed to enter the critical zone:

$$free_{tot} = free_1 \wedge free_2 \wedge \cdots \wedge free_n \tag{17}$$

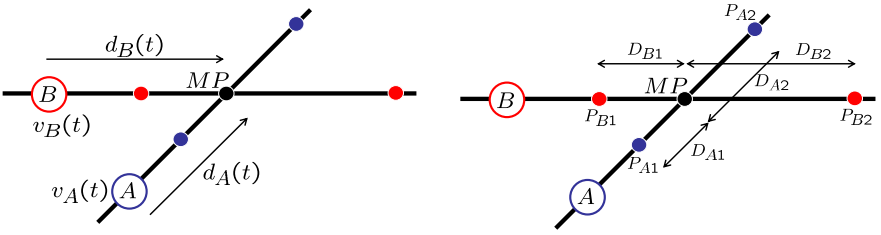


Fig. 14. Measured quantities and geometric parameters of the graph.

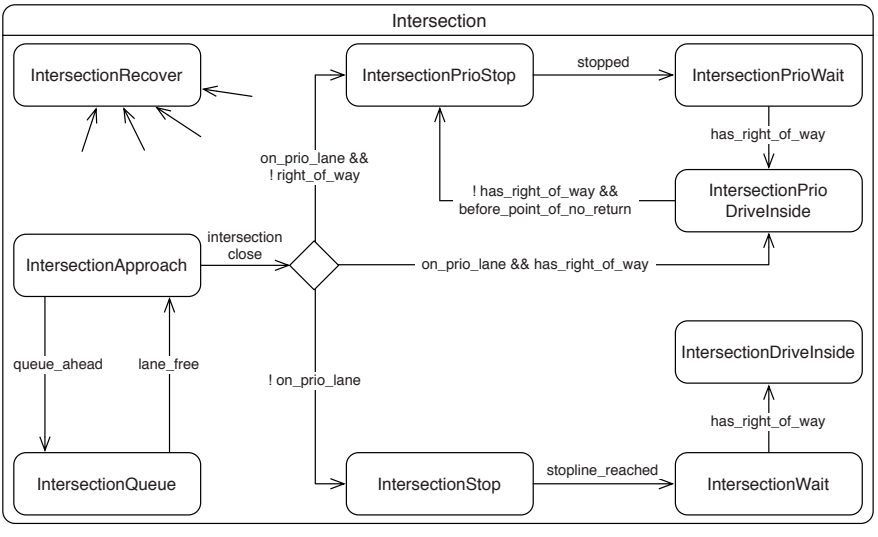


Fig. 15. UML diagram of substate Intersection.

9.3 Integration into the State Machine

The planner of Sec. 8 always deploys the moving traffic check (MTC) when AnnieWAY might come into conflict with other traffic participants demanding the same traffic space (*conflict spaces*). Contingent upon the result obtained from the MTC and the particular situation (*conflict situations*), state transitions are triggered and the resulting state generates the desired path and approves the free section for the longitudinal control.

In order to prevent frequent switching back and forth between states due to measurement noise and control inaccuracy, hysteresis in the MTC is introduced by slightly reducing the requirements once the autonomous vehicle set itself in motion.

Since the actual behavior of the other traffic participants can be roughly predicted at best, additional safety layers are introduced that prevent imminent collisions (see Sec. 8), in ticklish situations with emergency braking.

The *conflict situations* that arise from the competition, are limited to

- intersections,
- passing other cars,
- and changing lanes.

Due to the general formulation of the MTC, the different traffic situations can be accounted for with a corresponding parameter set.

For expository purposes the integration of the MTC in the intersection scenario will be described. Fig.15 shows the corresponding block diagram in UML notation. When the vehicle approaches the intersection, the hierarchical state machine changes into the sub-state *Intersection* with the entry state *IntersectionApproach*. This state is active until the vehicle enters the intersection unless another traffic participant is perceived on the same lane between AnnieWAY and the intersection. In this case *IntersectionQueue* is activated until the other vehicle has passed the intersection and the lane is free.

In *IntersectionApproach*, as soon as AnnieWAY gets close to the intersection, the state transition splits up into

(a) *IntersectionStop* if AnnieWAY is on a stop road,
(b) *IntersectionPrioDriveInside* if AnnieWAY is on a priority road and no other vehicle has the right-of-way,
(c) or *IntersectionPrioStop* if AnnieWAY is situated on a priority road, but needs to yield the right-of-way to priority vehicles, e. g. approaching traffic, before it may turn left.

In case (a) AnnieWAY stops at the stop line and changes into the state *IntersectionWait*. In this state all vehicles are registered that are already waiting on another stop line which have the right-of-way according to the driving rules (4-Way-Stop). As soon as these vehicles have passed the intersection and the MTC turns out positive for all visible priority vehicles, the state machine changes to *IntersectionDriveInside* and AnnieWAY merges into the moving traffic according to the safety parameters.

In case (b) AnnieWAY drives into the intersection without stopping. If a priority vehicle is perceived shortly after driving inside the intersection (point of no return has not been passed yet) and the MTC turns out negative, the state machine switches to *IntersectionPrioStop* which is equivalent to (c).

In case (c) in *IntersectionPrioStop* AnnieWAY stops before crossing the opposing lane, waits until the MTC confirms that no danger comes from priority vehicles anymore, and turns left.

10 Navigation in Unstructured Environment and Parking

As has been described in Sec. 8, paths can be generated in a straight forward way by sampling from the geometric road network graph where

sufficient road geometry information is available. However, Urban Challenge regulations require for navigating in unstructured environments (*zones*) that are only described by a boundary polygon. In the Urban Challenge, zones are used to outline parking lots and off-road areas. In this kind of area, a graph for path planning is not available. AnnieWAY's navigation system comprises a path planning algorithm that transcends the requirement for precise road geometry definition. It has also proven useful to plan narrow turns and as a general recovery mechanism when the vehicle gets off track, the road is blocked or a sensible localization within the given road network is impossible.

10.1 Configuration Space Obstacles

We restrict search to the collision free subset of configuration space (the vehicle's free space) by calculating configuration space obstacles from an obstacle map obtained from a 360°-laser range scanner (cf. Sec. 4.1). The discrete nature of this obstacle map motivated dealing with configuration space obstacles in a discrete way as well [Kavraki, 1995], as opposed to more traditional approaches that require obstacle input in the form of polygonal data [Schwartz and Sharir, 1983, Šwestka and Overmars, 1997]. Figs. 16(a) and 16(b) illustrate how the robots free space can be generated for a discrete set of orientations. By precomputing the free space in discretized form, a collision check for a certain configuration can be performed quickly in $\mathcal{O}(1)$ by a simple table lookup.

10.2 Search Graph and A*

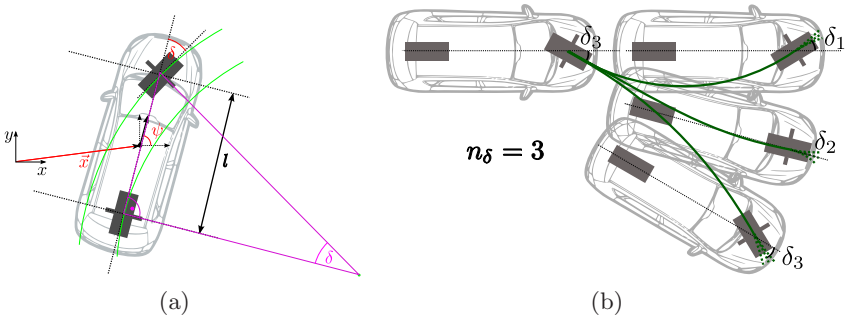
We define an implicit search graph in which all paths are feasible. It is directly derived from a kinematic model of the car and not only guarantees

Fig. 16. Configuration space obstacles. (a): A 1 m safety distance is added to the shape of the vehicle. Subsequent rotation and rasterization yields a convolution kernel for configuration space obstacle generation. (b): Result of convolving obstacle map with kernel from (a). If the robot has the same orientation as the kernel and is placed in the red area, it must intersect with an obstacle. (c): Voronoi lines are generated as a set of 8-connected pixels.

feasibility of the generated path, but also allows for straight forward design of a combined feed forward/feed backward controller (see Sec. 11).

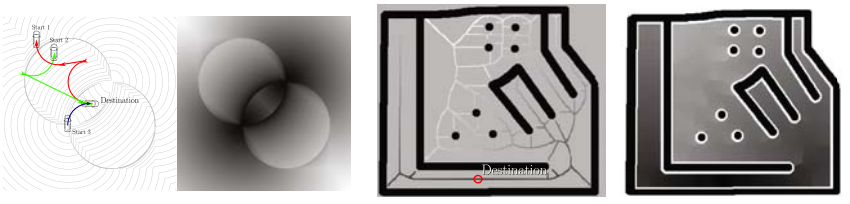
A node of the search graph can be completely described by a tuple ($\mathbf{x}$,ψ,δ), with $\mathbf{x}$, ψ and δ denoting position, orientation and steering angle (i.e. the deflection of the front wheels) of an instance of a kinematic one track model (see Fig. 17(a)). Steering angle δ is from a set of n_δ discrete steering angles that are distributed equidistantly over the range of feasible steering: $D = \{\delta_1 \dots \delta_{n_\delta}\}$. To generate successors of a node, the kinematic model equations are solved for initial values taken from the node, a fixed arc length s and a constant steering rate $\dot{\delta} = \frac{\delta_p - \delta_i}{s}$, spanning clothoid like arcs between the nodes. It is equivalent of driving the car model over a distance s at constant speed while uniformly turning the front wheels from δ_p to δ_i. For the set of nodes $\{(\mathbf{0}, 0, \delta_i), \delta_i \in D\}$, this results in n_δ^2 successors, and another n_δ^2 if backward motion is allowed. Successors of other nodes can be generated quickly from this precomputed set by subsequent rotation and translation (cf. Fig. 17(b)).

The search graph is expanded in this way by an A* search algorithm. A* search is a well known concept in the domain of robotic path planning [Hwang and Ahuja, 1992], that allows for accelerating exploration of the search space by defining a cost function that gives a lower bound of expected cost-to-go for each node of the search graph. If the cost function underestimates the actual distance to the goal, A* is guaranteed to find the least-cost path. If the error of the cost function is big, A* quickly degenerates to an exponential time algorithm. This is common when a metric cost function is used that does not account for obstacle positions, so that search can get stuck in a dead end configuration. We avoid this problem by designing an obstacle sensitive cost function that accounts for the topology of the free space.



(a) (b)

Fig. 17. (a) Kinematic one track model underlying both search graph and closed loop control. (b) Search graph. Successors are generated for n_δ discrete steering angles.



(a) (b)

Fig. 18. Cost functions. (a): RTR-metric for three different starting positions. Left hand side shows the minimum RTR-paths, right image the value of the RTR metric, densely evaluated on $\mathbb{R}^2$ (bright: high value, dark: low value). (b): Voronoj based cost function. Left: Voronoj graph labeled with distance by Dijkstras algorithm. Right: Voronoj based cost function evaluated densely on $\mathbb{R}^2$ by matching to the Voronoj-graph.

10.3 Cost Function

To guide the search process, we combined two different cost functions. The first one accounts for kinematic constraints of the vehicle, while the second one is derived from the Voronoi graph of the vehicle's free space and thus incorporates knowledge of shape and position of the obstacles.

10.3.1 Local Cost Function

As a local cost function, the so called *RTR metric* is used. RTR (rotation-translation-rotation) paths connect two configurations by two circular arcs of minimum turning radius and a straight segment tangenting both. It can be shown easily (cf. [Švestka and Overmars, 1997]), that for every pair of configurations a finite number of such paths can be constructed. The RTR metric is the arc length of the shortest such path. RTR paths do neither have continous curvature nor are they optimal (the optimal - in terms of arc length - solution to the local navigation problem are the so called Reeds and Shepp paths, cf. [Reeds and Shepp, 1991]), but are preferred by us due to their computational simplicity. Fig. 18(a) illustrates the RTR metric.

10.3.2 Voronoi Based Cost Function

We construct a powerful, obstacle sensitive cost function based on the Voronoi graph of the free space of the vehicle. Actually, a superset of the free space is used that is invariant to the vehicles orientation. It is generated by generating configuration space obstacles for a disk shaped structure that is the intersection of all structuring elements from Fig. 16(a).

Our algorithm to calculate Voronoi lines from a binarized obstacle map is similar to [Li and Vossepoel, 1998], however, instead of using the vector distance map, we use the approximate chamfer metric to be able to label Voronoi lines using only two passes over the obstacle map. The method is

derived from an algorithm [Borgefors, 1986, Li and Vossepoel, 1998] for calculating the euclidean distance transform. It gives the Voronoi lines as a set of 8-connected pixels.

After matching the target position to the closest point on the Voronoi graph, Dijkstras algorithm is used to calculate the shortest path distance to the target position for every point on the graph. Cost for a position not on the graph is derived by matching to the closest point on the graph and incorporating the matching distance in a way that yields a gradient of the cost function that is slightly sloped towards the Voronoi lines. Fig. 18(b) shows an example.

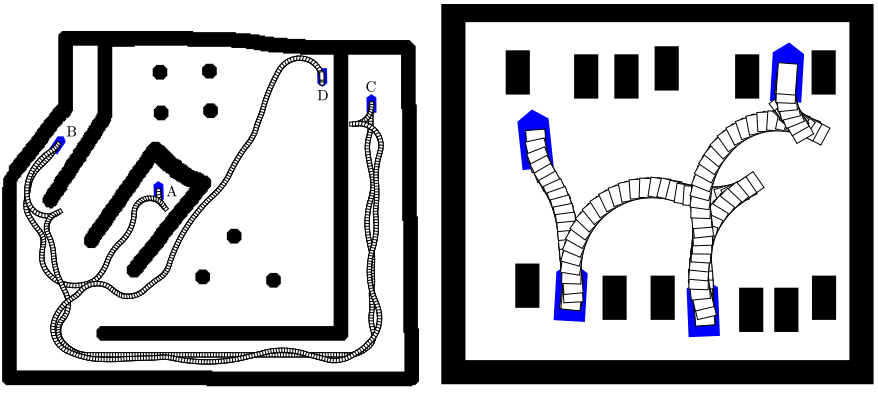
Using this heuristic function is appealing for several reasons. Since the Voronoi lines comprise the complete topology of the free space, search cannot get stuck in a dead end configuration, as is common with heuristics that do not incorporate knowledge of free space topology and therefore grossly underestimate the cost in such a case. Additionally, the Voronoi lines have - as the centers of maximum inscribing circles - the property of being at the farthest distances possible from any obstacle. This is conveyed to the planned paths, giving reserves to account for control- and measurement errors.

10.3.3 Combination of Cost Functions

We combine the two cost functions into one by the maximum operator. This procedure can be justified from the admissibility principle for heuristics in the context of A* search. A heuristic is called admissible, if it consistently underestimates the cost to the target node. Consequently, combining two heuristics via the maximum operator still gives an admissible heuristic. Result of comparing both costs coincides with the practical experience that in the vicinity of the target position, cost is dominated by the necessity to maneuver in order to reach the destination in right orientation, while cost at long distances often is caused by the necessity to avoid obstacles. Fig. 10.3.3 shows some results of A* search using the search graph from Sec. 10.2 and the combined cost function.

11 Vehicle Control

The last step of the processing chain is the vehicle control which can be separated into lateral and longitudinal control. Since the distances to dynamic objects are fairly big in the *Urban Challenge* 2007 competition, for high-level decision making the problem of trajectory planning (coordinates of the desired vehicle position as a function of time) can be reduced to a combination of path planning (path geometries with no time dependencies) and determining the free section of the path rather than an exact desired position. The longitudinal strategy is thereby assigned to a lower level, which evaluates the free section of the path and induces the vehicle to go faster or slower. The information transfer of the interface is undertaken by so-called curve points, a discrete representation of the path geometry.



(a) (b)

Fig. 19. Some results of path planning on simulated map data. (a): Navigating long distances in a maze like environment. Planning was from A to B, B to C and C to D subsequently. (b): Some difficult parking maneuvers performed subsequently. Robot started on the right.

As the emphasis of the competition is on low to medium velocities, the non-holonomic single track model holds and an orbital tracking controller (e. g. [Müller, 2007]) is chosen for the lateral dynamics in Sec. 11.1. This offers the advantage of a velocity independent transient lateral behavior for the closed loop system. Suppose the vehicle had an offset from the planned path of a couple centimeters caused by sensor drift of the navigation system, the lateral controller would reduce the error over a certain traveled distance rather than over time and avoids unpredictable overshoots of the front end which might lead to collisions.

From the longitudinal controller's point of view, the vehicle drives on rails, as the lateral controller minimizes the lateral offset. Thus, the longitudinal control strategy faces solely the task of following moving objects, stopping at certain points, maintaining the maximum speed, and changing direction along the given path. For this purpose different controllers are designed in Sec. 11.2 that are included in an override control strategy ensuring bumpless transfers between them. The output of every longitudinal controller is the vehicle's acceleration a. This acceleration will be converted to the manipulated variables accelerator pedal value ϕ_{gas} and brake pressure p_{brake} in a cascaded acceleration controller exceeding the scope of this contribution.

11.1 Orbital Tracking Controller

The dynamics of a non-holonomic vehicle (Fig. 20) in local coordinates s_c, d, and $\Delta\psi$ are given by

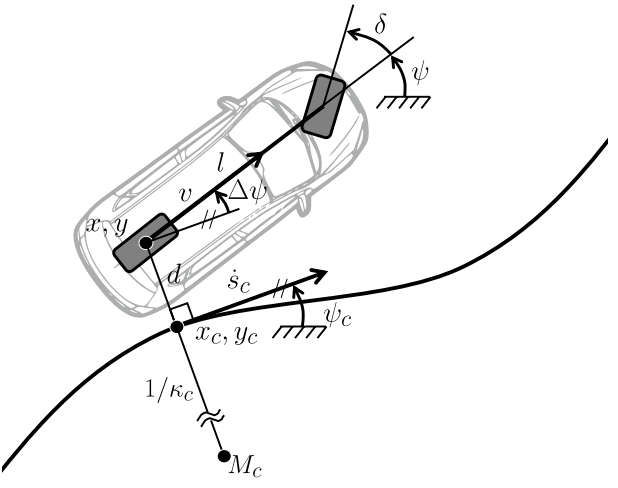


Fig. 20. Non-holonomic one track model.

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} s_c \\ d \\ \Delta\psi \end{bmatrix} = \begin{bmatrix} \frac{\cos\Delta\psi}{1-d\kappa_c(s_c)} \\ \sin\Delta\psi \\ \frac{\tan\delta}{l} - \kappa_c(s_c)\frac{\cos\Delta\psi}{1-d\kappa_c(s_c)} \end{bmatrix} v, \tag{18}$$

whereas the steering wheel angel δ and the longitudinal velocity v is the system's input, d is the lateral offset to the path, $\Delta\psi$ is the angle between the vehicle and the tangent to the path, and l the distance between the rear and the front axle. The singularity at $1 - d\kappa_c(s_c) = 0$ is no restriction in practice since $d \ll \frac{1}{\kappa_c(s_c)}$.

Since orbital tracking control does not have any time dependencies, (18) can be rewritten with the arc length s_c as the new time parametrization.

With $\frac{\mathrm{d}}{\mathrm{d}t}() = \frac{\mathrm{d}}{\mathrm{d}s_c}() \cdot \frac{\mathrm{d}s_c}{\mathrm{d}t}$ it becomes

$$\frac{\mathrm{d}}{\mathrm{d}s_c}\begin{bmatrix} s_c \\ d \\ \Delta\psi \end{bmatrix} = \begin{bmatrix} 1 \\ \sin\Delta\psi \cdot \frac{1-d\kappa_c(s_c)}{\cos\Delta\psi} \\ \frac{\tan\delta}{l} \cdot \frac{1-d\kappa_c(s_c)}{\cos\Delta\psi} - \kappa_c(s_c) \end{bmatrix}. \tag{19}$$

For small deviations d and $\Delta\psi$ from the desired curve and $\frac{\mathrm{d}}{\mathrm{d}s_c}() = ()'$, a partial linearization leads to

$$\begin{bmatrix} d \\ \Delta\psi \end{bmatrix}' = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} d \\ \Delta\psi \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\kappa_c + \begin{bmatrix} 0 \\ \frac{1}{l} \end{bmatrix}\tan\delta. \tag{20}$$

The linearizing control law

$$\delta = \arctan(-lk_0 d - lk_1\Delta\psi + l\kappa_c) \tag{21}$$

$$= \arctan(-k_1^\star d - k_2^\star\Delta\psi + l\kappa_c) \tag{22}$$

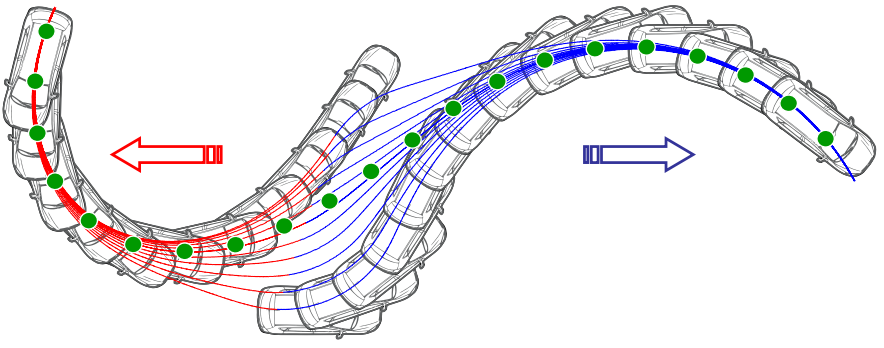


Fig. 21. Trajectories for different initial positions.

with $k_0, k_1 > 0$ yields the stable linear error dynamics

$$\frac{\mathrm{d}}{\mathrm{d}s_c}\begin{bmatrix} d \\ \Delta\psi \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k_0 & -k_1 \end{bmatrix}\begin{bmatrix} d \\ \Delta\psi \end{bmatrix} \tag{23}$$

with respect to s_c with the characteristic polynomial $\lambda^2+k_1\lambda+k_0 = 0$. As long as $\dot{s}_c > 0$, the system is also stable with respect to time. For backward driving the signs of k_0 and k_1 have to be adjusted to the applied sign convention and yields exactly the same error dynamics as for forward driving.

Fig. 21 shows the transient behavior to different initial errors $\Delta\psi$ and d for forward (blue) and backward driving (red) simulated with MATLAB/SIMULINK. As parameters for the simulation the Passat's axis distance $l = 2.72$, a maximum steering angle of $\delta_{\max} = 30°$, the controller parameters $k_0 = 0.25\,l$ and $k_1 = 1.25\,l$ and equidistant curve point with $\Delta = 2$m were chosen. Obviously neither the input saturation $\delta_{\max}$ nor the discrete representation of the curve cause any significant problems.

11.2 Longitudinal Controller System

11.2.1 Following Controller

Since the acceleration of a leading vehicle is hard to determine, it is assumed that the vehicle keeps its velocity v_B constant. Choosing the distance d_f and its time derivative $\dot{d}_f$ as the state variables and AnnieWAY's acceleration $a_f = \dot{v}$ as the input, the system's dynamics are given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} d_f \\ \dot{d}_f \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} d_f \\ \dot{d}_f \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} a_f \tag{24}$$

As DARPA requires the vehicle to maintain a minimum forward vehicle separation of one vehicle length minimum and one length for every additional 10 mph, the desired distance $d_{f,d}$ can be calculated by

$$d_{f,d} = d_{f,0} + \tau v \tag{25}$$

with the according parameters $d_{f,0}$ and τ. Considering the acceleration $\dot{v}_B$ of the leading vehicle an unmeasurable disturbance, the linear set-point control law

$$a_f = c_0(d_f - d_{f,d}) + c_1\dot{d}_f \tag{26}$$

$$= c_0(d_f - d_{f,d}) + c_1(v_B - v) \tag{27}$$

and $v = v_B - \dot{d}_f$ yields the total system

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} d_f \\ \dot{d}_f \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -c_0 & -c_0\tau - c_1 \end{bmatrix}\begin{bmatrix} d_f \\ \dot{d}_f \end{bmatrix} + \begin{bmatrix} 0 \\ c_0(d_{f,0} + \tau v_B) \end{bmatrix}. \tag{28}$$

The characteristic polynomial $\lambda^2 + (c_0\tau + c_1)\lambda + c_0 = 0$ can directly be read off from (28). A double Eigenvalue $\lambda_{1/2} = -1$ leads to a pleasant and yet safe following behavior.

11.2.2 Stopping Controller

The following controller of the previous section leads to a behavior, which can best be described as *flowing with the traffic*. By contrast, the stopping controller should come to a controlled stop at a certain point as fast as possible without exceeding any comfort criteria. The control law

$$a_s = -\frac{v^2}{2(d_f - d_\Delta)} \tag{29}$$

leads to a constant deceleration until the vehicle is d_Δ away from the stop point. To prevent the controller from decelerating too soon and switching on and off, a hysteresis with the thresholds $a_{s,\max}$ and $a_{s,\min}$, as shown in Fig. 22, is introduced. The singularity at $d_f = d_\Delta$ is avoided by a PD position controller that takes over via a *min*-operator and ensures a smooth and save stop at the end.

11.2.3 Velocity Controller

As $\dot{v} = a$, the simple proportional velocity control law

$$a_v = -c_v(v - v_d) \tag{30}$$

stabilizes AnnieWAY's velocity v to the desired velocity v_d with a PT_1 behavior.

11.2.4 Override Control Strategy

All three previously introduced controllers are combined by an override control strategy depicted in Fig. 22. The bumpless transfer between velocity control and following/stopping control is assured by the max operator. Additional saturation, realized by $a_{\max}$ and $a_{\min}$, prevent the vehicle from inappropriately high acceleration or deceleration without reducing safety.

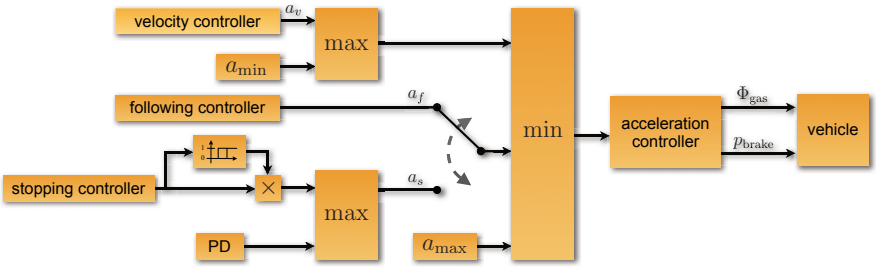


Fig. 22. Longitudinal override control strategy.

12 Results

Originally 89 teams have entered the competition, 11 of which were sponsored by the organizer. After several stages, 36 of those teams were selected for the semi-final. There, AnnieWAY has accomplished safe conduction of a variety of maneuvers including

- regular driving on lanes
- turning at intersections with oncoming traffic
- lane change maneuvers
- vehicle following and passing
- following order of precedence at 4-way stops
- merging into moving traffic

Although the final event was originally planned to challenge 20 teams, only 11 finalists were selected by the organizers due to safety issues. AnnieWAY has entered the final and was able to conduct a variety of driving maneuvers. It drove collision-free, but stopped due to a software exception in one of the modules.

Fig. 23 depicts three examples of the vehicles actual course taken from a log-file and superimposed on an aerial image. The rightmost figure shows the stopping position in the finals.

The following section points out some results of the navigation module. Fig. 24(a) illustrates one test driven in a parking area close to our test ground. Unlike the required navigation task in the Urban Challenge, the chosen setup features many surrounding obstacles such as other cars and curbs.

Search time remains below 2 seconds in all practical situations. Though the environment is assumed to be static, this is fast enough to cope with slow changes in the environment by continuous replanning. Additionally, to avoid collision with fast moving objects, a lower level process continuously determines the free section of the planned path and, if necessary, invokes a new search. The lateral controller follows the generated paths precisely enough to implement all of the intended maneuvers.

Besides path planning in parking areas, the zone-navigation module was used as recovery option in case of continuous blocking of lanes or intersections.

Fig. 23. Three steps of AnnieWAYs course driven autonomously in the finals.

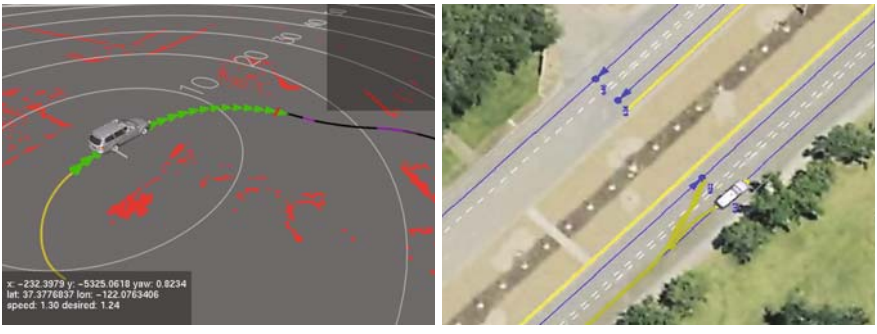


Fig. 24. (a) Path planning in heavily occupied zone with mapper input (red) and sampled waypoints as output to the controller (green). (b) Recovery maneuver during final event, driven path is marked olive.

A wrong detected obstacle on the left lane forced activation of the navigation module during the final event. The vehicle was successfully brought back on track after a backup maneuver as can be seen in Fig. 24(b).

13 Conclusions

The autonomous vehicle AnnieWAY is capable of driving through urban scenarios and has successfully entered the finals of the *2007 DARPA Urban Challenge* competition. In contrast to earlier competitions, the Urban Challenge required to conduct missions in 'urban' traffic, i.e. in the presence of other autonomous and human-operated vehicles. The major challenge imposed was collision-free and rule-compliant driving in traffic. AnnieWAY is based on a simple and robust hardware architecture. In particular, we rely on a single computer system for all tasks but low level control. Environment perception is mainly conducted by a roof-mounted laser scanner that measures range and reflectivity for each pixel. While the former is used to provide 3D scene geometry, the latter allows robust lane marker detection. Mission and maneuver selection is conducted via a concurrent hierarchical state machine

that specifically asserts behavior in accordance with California traffic laws. More than 100 hours of urban driving without human intervention in complex urban settings with multiple cars, correct precedence order decision at intersections and - last not least - the entry in the finals underline the performance of the overall system.

Acknowledgments

The authors gratefully acknowledge the great collaboration of their partners from Universität Karlsruhe, Technische Universität München and Universität der Bundeswehr München. Special thanks are directed to Annie Lien for her instant willingness and dedication as our official team leader. This work had not been possible without the ongoing research of the Transregional Collaborative Research Centre 28 'Cognitive Automobiles'. Both projects cross-fertilized each other and revealed significant synergy. The authors gratefully acknowledge support of the TCRC by the Deutsche Forschungsgemeinschaft (German Research Foundation).

References

Bertozzi et al., 2000. Bertozzi, M., Broggi, A., Fascioli, A.: Vision-based intelligent vehicles: State of the art and perspectives. J. of Robotics and Autonomous Systems 32, 1–16 (2000)

Biber and Strasser, 2006. Biber, P., Strasser, W.: nscan-matching: Simultaneous matching of multiple scans and application to slam. In: IEEE International Conference on Robotics and Automation (2006)

Borgefors, 1986. Borgefors, G.: Distance transformations in digital images. Computer Vision, Graphics, and Image Processing 34(3), 344–371 (1986)

Bosse et al., 2003. Bosse, M., Newman, P.M., Leonard, J.J., Soika, M., Feiten, W., Teller, S.J.: An atlas framework for scalable mapping. In: Proceedings of the 2003 IEEE International Conference on Robotics and Automation, Taipei, Taiwan, pp. 1899–1906 (2003)

Bracewell, 1990. Bracewell, R.N.: Numerical transforms. Science 248(4956), 697–704 (1990)

DARPA, 2005. DARPA, Grand Challenge 2005 (2005), http://www.darpa.mil/grandchallenge/overview.html

Dickmanns et al., 1994. Dickmanns, E.D., Behringer, R., Dickmanns, D., Hildebrandt, T., Maurer, M., Thomanek, F., Schielen, J.: The seeing passenger car "VaMoRs-P". In: Proc. Symp. On Intelligent Vehicles, Paris, pp. 68–73 (1994)

Franke et al., 2001. Franke, U., Gavrila, D., Gern, A., Goerzig, S., Jansen, R., Paetzold, F., Wöhler, C.: From door to door – Principles and applications of computer vision for driver assistant systems. In: Vlacic, L., Harashima, F., Parent, M. (eds.) Intelligent Vehicle Technologies: Theory and Applications, ch. 6, pp. 131–188. Butterworth Heinemann, Oxford (2001)

Goebl and Färber, 2007a. Goebl, M., Färber, G.: Eine realzeitfähige Softwarearchitektur für kognitive Automobile. In: Berns, K., Luksch, T. (eds.) Autonome Mobile Systeme 2007, Informatik Aktuell, pp. 198–204. Springer, Heidelberg (2007a)

Goebl and Färber, 2007b. Goebl, M., Färber, G.: A real-time-capable hard- and software architecture for joint image and knowledge processing in cognitive automobiles. In: Proc. IEEE Intelligent Vehicles Symposium, Istanbul, Turkey, pp. 734–739 (2007b)

Hummel et al., 2006. Hummel, B., Kammel, S., Dang, T., Duchow, C., Stiller, C.: Vision-based path-planning in unstructured environments. In: IEEE Intelligent Vehicles Symposium, Tokyo, Japan (2006)

Hwang and Ahuja, 1992. Hwang, Y.K., Ahuja, N.: Gross motion planning - a survey. ACM Comput. Surv. 24(3), 219–291 (1992)

Kalman, 1960. Kalman, R.: A new approach to linear filtering and prediction problems. Journal of Basic Engineering 82(1), 35–45 (1960)

Kavraki, 1995. Kavraki, L.: Computation of configuration-space obstacles using the fast fourier transform. IEEE Transactions on Robotics and Automation 11(3), 408–413 (1995)

Li and Vossepoel, 1998. Li, H., Vossepoel, A.M.: Generation of the euclidean skeleton from the vector distance map by a bisector decision rule. In: CVPR 1998: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Washington, DC, USA, p. 66. IEEE Computer Society, Los Alamitos (1998)

Müller, 2007. Müller, B., Deutscher, J.: Orbital tracking control for car parking via control of the clock. In: Methoden und Anwendungen der Regelungstechnik, Erlangen-Münchener Workshops 2005 und 2006, Shaker Verlag, Aachen (2007)

Nagel et al., 1995. Nagel, H.-H., Enkelmann, W., Struck, G.: FhG-Co-Driver: From map-guided automatic driving by machine vision to a cooperative driver support. Mathematical and Computer Modelling 22, 185–212 (1995)

Özgüner et al., 2007. Özgüner, Ü., Stiller, C., Redmill, K.: Systems for safety and autonomous behavior in cars: The DARPA Grand Challenge experience. IEEE Proceedings 95(2), 1–16 (2007)

Reeds and Shepp, 1991. Reeds, J., Shepp, R.: Optimal paths for a car that goes both forward and backward. Pacific Journal of Mathematics 145, 144–154 (1991)

Schwartz and Sharir, 1983. Schwartz, J.T., Sharir, M.: On the piano movers' problem, ii: General techniques for computing topological properties of real algebraic manifolds. Advances in Applied Mathematics 4, 298–351 (1983)

Thorpe, 1990. Thorpe, C.: Vision and navigation - The Carnegie Mellon Navlab. Kluwer Academic Publishers, Dordrecht (1990)

Thrun, 2002. Thrun, S.: Robotic mapping: A survey. In: Lakemeyer, G., Nebel, B. (eds.) Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann, San Francisco (2002)

Thrun, 2003. Thrun, S.: Learning occupancy grid maps with forward sensor models. Auton. Robots 15(2), 111–127 (2003)

Thrun et al., 2006. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics 23(9), 661–692 (2006)

v. Hundelshausen et al., 2008. v Hundelshausen, F., Himmelsbach, M., Mueller, A., Wuensche, H.-J.: Tentacles- a biologically inspired approach for robot navigation. Journal of Field Robotics (under submission)

Šwestka and Overmars, 1997. Šwestka, P., Overmars, M.: Motion planning for car-like robots using a probabilistic learning approach. The International Journal of Robotics Research 16(2), 119–143 (1997)

Driving with Tentacles - Integral Structures for Sensing and Motion

Felix v. Hundelshausen, Michael Himmelsbach, Falk Hecker, Andre Mueller, and Hans-Joachim Wuensche

Autonomous Systems Technology
Department of Aerospace Engineering
University of the Federal Armed Forces Munich
85579 Neubiberg, Germany
felix@unibw.de, michael.himmelsbach@unibw.de, falk.hecker@unibw.de, andre.mueller@unibw.de, joe.wuensche@unibw.de

Abstract. In this paper we describe a LIDAR-based navigation approach applied at both the C-Elrob (European Land Robot Trial) 2007 and the DARPA Urban Challenge 2007. At the C-Elrob 2007 the approach was used without any prior knowledge about the terrain and without GPS. At the Urban Challenge the approach was combined with a GPS-based path follower. At the core of the method is a set of "tentacles" that represent precalculated trajectories defined in the ego-centered coordinate space of the vehicle. Similar to an insect's antennae or feelers, they fan out with different curvatures discretizing the basic driving options of the vehicle. We detail how the approach can be used for exploration of unknown environments and how it can be extended to combined GPS path following and obstacle avoidance allowing save road following in case of GPS offsets.

1 Introduction

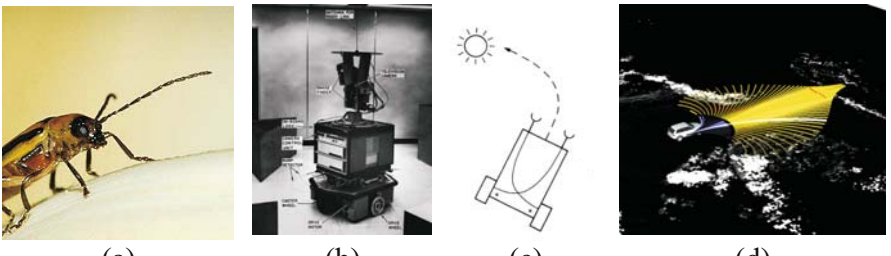
In this paper, we put forth a very simple method for autonomous robot navigation in unknown environments. The method is not restricted to a particular robot or sensor, however we demonstrated it on the two vehicles in which the approach was integrated. Our method was applied at the Civilean European Land Robot Trial 2007 (C-Elrob 2007) on our VW-Touareg MuCAR-3 (Munich Cognitive Autonomous Robot, 3rd generation) and the DARPA Urban Challenge 2007 (UC07) on the VW-Passat of Team AnnieWay. Both vehicles had an almost identical hardware setup, in particular both being equipped with a Velodyne 64 beam 360 degrees LIDAR (see figure 1).

While quite complex approaches to mobile robot navigation exist (e.g solving the SLAM problem (Dissanayake et al., 2001; Julier and Uhlmann, 2001), methods based on trajectory planning (Sariff, 2006)), our research was driven by the search for simplicity: **What is the simplest approach that lets a robot safely drive in an unknown environment?** Intentionally, we write

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 393–440.
springerlink.com

(a) (b)

Fig. 1. The two vehicles our approach was tested with. (a) The VW-Passat of Darpa Urban Challenge finalist Team AnnieWay (b) The VW-Touareg MuCAR-3 (Munich Cognitive Autonomous Robot, 3rd generation), a C-Elrob 2007 Champion. Both vehicles are equipped with a 360 degree Velodyne-LIDAR as primary sensor scanning its environment at 10 Hz using 64 laser beams.



(a) (b) (c) (d)

Fig. 2. (a) Among other functions insects use antennae as tactile sense (b) Shakey (Nilsson, 1984) - one of the first robots - uses mechanical "cat-whiskers" to detect collisions (c) Braitenberg vehicles use some sort of antennae with simple, hard-wired reactive mechanisms to produce complex behaviors (d) In our approach we use speed depending sets of antennaes that we call "tentacles".

"drive" instead of "explore" because we do not demand to construct a map of the environment (in contrast to SLAM approaches (Dissanayake et al., 2001; Julier and Uhlmann, 2001)), but just demand safe driving within that environment. Our basic intention underlying this paper is to let our robot move within an unknown environment similarly to how a beetle would crawl around and uses its antennae to avoid obstacles. Indeed, our basic approach consists of using a set of virtual antennae that we call "tentacles" probing an ego-centered occupancy grid for drivability. Of course, the idea of using antennae is not new: In fact, one of the first robots,"Shakey" (Nilsson, 1984), used "cat-whiskers" (micro-switches actuated by a 6 inch long coil spring extended by piano wires to provide longer reach) to sense the presence of a solid object within the braking

distance of the vehicle when traveling at top speed (see fig 2b). In his cybernetic thought games (Braitenberg, 1984), Valentino Braitenberg showed that complicated behavior can emerge by very simple mechanisms, and almost all of his vehicles (known under the term *Braitenberg vehicles*) use sensors resembling an insects's antennae (figure 2c).

Some systems in mobile robotics integrate modules that are similar to our approach in that some sort of precalculated trajectories are verified to be drivable: In (Kelly and Stentz, 1998) the authors follow the idea of model referenced control (Landau, 1979) implemented through command space sampling and evaluating a set of candidate trajectories. In (Lamon et al., 2006) a set of feasible arcs is used as a first step in a path planning algorithm. Stanley, the robot that won the DARPA Grand Challenge in 2005, used a set of candidate paths ("nudges" and "swerves") for path planning (Thrun et al., 2006).

In the DAMN framework (Distributed Architecture for Mobile Navigation) (Rosenblatt, 1995) four different arbitration schemes for integrating the results of different distributed behaviors are proposed. One of those arbitration schemes is actuation arbitration, and the work of (Rosenblatt, 1995) sketches an example of that scheme, where different turn behaviors cast votes on candidate vehicle curvature commands. The difference of our approach is that the candidate commands are not only used as candidate instances in a command space but our "tentacles" are also used as perceptual primitives and a very detailed process of how those primitives are used to evaluate an occupancy grid is proposed. This evaluation process includes some new structures and aspects, including the differentiation between a support and a classification area, the use of a longitudinal histogram for classifying tentacles as being drivable or not, determining the distance to the first obstacle along a tentacle, a speed depending evaluation length (the crash distance) that allows to reduce the number of required primitives drastically.

Another work in the context of planetary rover exploration that looks very similar to our approach at first glance is GESTALT (Grid-based Estimation of Surface Traversability Applied to Local Terrain) (Goldberg et al., 2002). Similar to our approach, GESTALT selects an arc with maximum goodness according to various criteria, however the main difference to our approach is that GESTALT uses a global grid and accumulates data over time in this grid. In doing so, the approach becomes an instance of the SLAM problem:

"At the end of each step, sensors are expected to provide a reasonably accurate estimate of the rover's new position. GESTALT does not require that the rover motion exactly match that with the commanded, but it does assume that whereever the rover ended up, its relative position and orientation can be reasonably be inferred and provided as input. That is one limitation of the system, that it relies on other modules to deal with myriad position estimation problems (slipping in sand, getting stuck on a rock, freeing a jammed wheel, etc)" ((Goldberg et al., 2002), page 4, last paragraph).

In contrast, our approach does not accumulate data and uses a local ego-centered grid, thereby avoiding the SLAM problem. Another difference of our approach is that it provides speed-depending mechanisms for evaluating tentacles. This speed-dependency is a crucial point in our approach because it allows us to restrict the motion-primitives to a small set while still being able to drive through narrow passages having shapes other than circular arcs.

The work most similar to our approach is (Coombs et al., 2000): Here, a fixed tree structure with 5 levels (the root being located at the robot's position) is used. The first level consists of 20 m long precalculated clothoids while the other levels consist of straight line connections. Within this tree, all possible paths from the root to the leaves form the set of possible trajectories. The whole tree consist of about 4000 edges, resulting in a combinatorial power of over 15 million trajectories. In contrast to this work our approach works well with only 81 precalculated "tentacles" at a given speed, and no combinations of adjacent path fragments are required. This low number of possible path fragments is possible due to the special speed depending mechanism in our tentacle-evaluation process (to be detailed later), and is even sufficient in scenarios with narrow curved roads. All path fragments take the shape of circular arcs. In contrast to the aforementioned approaches, the basic samples for maneuvers are more short-termed and more reactive in the sense of simple *Braitenberg-vehicles* in our work, however - as will be shown - the way of evaluating the different driving-options is much more detailed and multi-faceted than existing approaches. The approach as detailed in this paper was tested excessively on common residential roads and offroad terrain: Our algorithm was successfully demonstrated at both the C-Elrob 2007 (driving record time in a combined urban and non-urban course including serpentines without using GPS, driving 90 percent of the course autonomously) and the DARPA Urban Challenge 2007 (getting into the final)[1]. Our approach is fully described, simple to implement and ready to use.

The remainder of the paper is organized as follows: In section 2 we detail the generation process of an ego-centered occupancy grid - the input domain for our method. In section 3 we detail the structure and generation of our "tentacles". Section 4 describes how the "best tentacle" is selected in each iteration and section 5 how this tentacle is executed. Section 6 analyzes our approach with respect to vehicle dynamics, computing upper bounds for path deviations. Section 7 describes experiments and the performance at competitions where our approach was used and details its overall system integration. Also, some lessons learned are described in this section. Finally section 8 concludes our paper. Throughout our paper, we specify the values of all parameters, such that our approach may be reproduced and serve as a reference for future improvements.

[1] The algorithm was integrated in the DARPA Urban Challenge 2007 team "AnnieWay" (Kammel, 2007).

2 Occupancy Grid

We use a two dimensional occupancy grid with 512×512 cells, each covering a small ground patch of 25cm $\times$ 25cm. Each cell stores a single floating point value expressing the degree of how occupied that cell is by an obstacle. In our implementation this value is a metric length with the physical unit "meters". Before we detail its meaning and calculation, note that in our approach we create a new occupancy grid on each new LIDAR rotation, i.e every 100ms as our LIDAR is set to turn at 10Hz. Thus, we do not accumulate data for a longer time. The reasons for this decision are: First one rotation of our 64-beam Velodyne-sensor supplies about 100.000 3D points, which proved to be sufficient. Second, the quality of an accumulated occupancy grid can easily deteriorate, if the physical movement of the sensor is not estimated with very high precision. Small angular deviations in the estimate of the sensor's pose can result in large errors. Registering scans against each other, e.g. using the ICP (Besl and McKay, 1992) algorithm or some of its derivatives could solve this problem but would require substantial additional computational load. Similarly, accumulating data in a grid requires additional time-consuming maintenance operations, like copying or removing data from the grid. A scrolling or wrappable map as proposed in (Kelly and Stentz, 1998) is not expedient in our case, because our method requires an ego-centered grid for efficiency. Hence, we decided not to accumulate data although this question might remain controversial:

Consider, for example, the case of the vehicle approaching a pavement edge. Depending on how the sensor is mounted, the Velodyne LIDAR typically has a "blind area" of some meters around the vehicle not hit by any beam. Thus, part of the pavement edge might get invisible. When no explicit representation (besides the grid) of such a pavement edge is used, grid accumulation would be helpful. However, this accumulation should be done with care as will become clearer when detailing how the grid values are computed. We first assume that every single LIDAR measurement of the last frame has instantly been transformed to a global cartesian 3D coordinate system, taking into account the vehicle's own motion (exploiting IMU and odometric

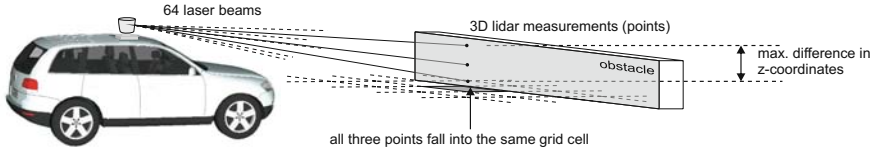


Fig. 3. To show how the grid is computed only 3 of the 64 laser beams of our sensor are shown as they hit an obstacle. The corresponding points in 3D space fall into the same grid cell. A grid value is computed to be the maximum difference of z-coordinates of points in 3D space falling into the same grid cell.

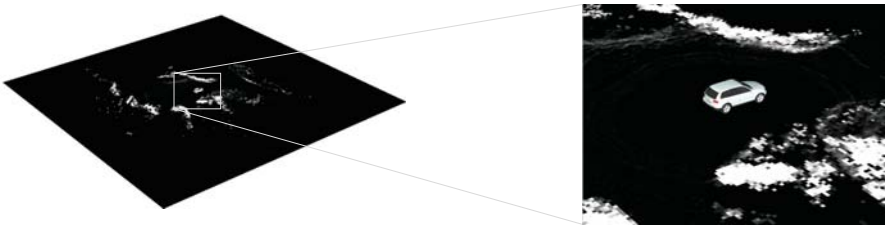


Fig. 4. At the right hand side a cut out of the occupancy grid on the left is shown. The position of the vehicle in the grid is always in the center. The grid is "hard-mounted" to the vehicle. Cells with large z-differences are shown white, smaller z-differences are shown as gray values.

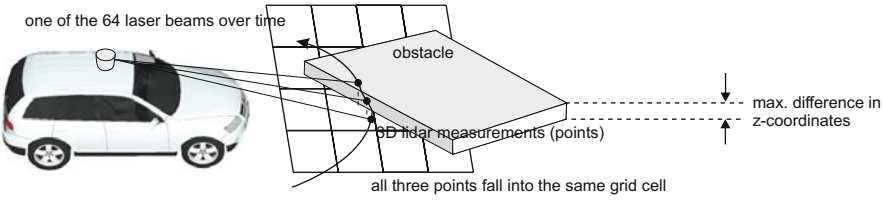


Fig. 5. One of the 64 LIDAR beams slices an obstacle over time. Due to the high horizontal resolution of the LIDAR, even a single beam might produce different z-values of points falling into the same grid cell.

information). This is done by simultaneously moving the coordinate system of the vehicle while transforming the local LIDAR measurements to global 3D space. After a frame is completed all points are transformed back into the last local coordinate system of the vehicle, simulating a scan as if all measurements were taken at a single point of time instead of the 100ms time period of one LIDAR revolution. While integrating IMU and odometric data over a long period of time leads to large drifts, the drift can be neglected for the short time period of 100ms. Each grid value is then computed to be the maximum difference in z-coordinates of all points falling into the respective grid cell. We refer to this value as *grid* or *occupancy value* throughout the text. To see the rational behind this computation consider an obstacle in front of the vehicle as shown in figure 3. Such a z-difference doesn't necessarily need to emerge from different beams. Due to the high horizontal resolution of the LIDAR, even a single beam might slice an obstacle and produce different z-values of points that fall into the same grid cell (see figure 5).

Note that by using z-coordinate differences, we only get a $2\frac{1}{2}$D model of the world. While this model can capture both obstacles that are in contact with the ground as well as ones that aren't, it can not differentiate between both - it is missing absolute z-coordinates. As a result of this limitation, the obstacle avoidance behavior will be conservative, e.g. the vehicle will try to

avoid tree branches even if it could safely pass beneath them. Likewise, it will avoid obstacles such as barriers, as desired.

If our approach should be modified in a way that data accumulation is done, we recommend to use points only from the same LIDAR turn when calculating the differences in z-coordinates and rather accumulate the results than the raw 3D point data . Figure 4 shows an example of our grid (with no data accumulation).

3 Tentacle Structure and Generation

In our system we use 16 sets of tentacles. As shown in figure 6 each of these "speed sets" contains 81 tentacles corresponding to a specific velocity of the vehicle. The speed sets cover a range of velocities from 0 to 10 m/s with lower speeds being represented more frequently. All tentacles are represented in the local coordinate system of the vehicle. They start at the vehicle's center of gravity and take the shape of circular arcs. Each arc represents the trajectory corresponding to a specific steering angle. At low speeds, higher curvatures are present than at high speeds. The tentacle's lengths increase with higher speed sets, whereas within a set less curved tentacles are longer.

For a justification of the choice of circular arcs consider that each tentacle will be executed only for 0.1 seconds, and hence the overall resulting trajectory can be thought of a concatenation of small circular fragments. For the above choice of a maximum speed of $10m/s$ the maximum length of a fragment is $1m$. Hence, we can reasonably well approximate all possible clothoids the vehicle can drive.

3.1 Geometry

Our tentacles have the shape of circular arcs. They are used for both perception and motion execution. For perception, they span two areas, a classification and a support area that are used to probe the grid underneath the tentacle. For execution, an initial fragment of a selected tentacle is considered

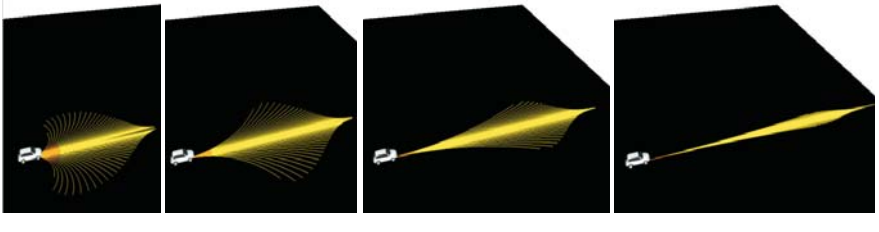


Fig. 6. The range of speeds from 0 to 10 m/s is represented by 16 speed sets, each containing 81 tentacles. Only four of these sets are shown here. The tentacles are circular arcs and start at the center of gravity of the vehicle.

as a path to be driven for one LIDAR frame (0.1 seconds). Different execution options exist, with the simplest method just setting the appropriate steering angle corresponding to the selected tentacle - combined with a temporal filtering method to avoid all-too sudden changes in curvatures. As we will see in section 6, the execution modes will cause the vehicle to not precisely drive along the sequences of tentacles. Our rational is not to care about the precise trajectory but just to ensure that the resulting path is within a corridor spanned by the tentacles. The trick is to make the tentacles broader than the vehicle, such that the maximum possible path deviation is included in the area that is evaluated to be free of obstacles.

3.1.1 The Tentacles in Our Experiments

Originally, our algorithm was conceived, implemented and used at the C-Elrob and DARPA Urban Challenge 2007 without analyzing the vehicle dynamics. By large, the design was put forth in an ad hoc manner and verified and refined by excessive experiments. Some design issues have a physical reasoning, but with no real theoretic derivation that is founded on vehicle dynamics. Many of the parameters and formulas that define them were determined empirically. The justification for this is that all our choices are far behind the physical limits of the vehicle. And within this space of physical feasibility we just exploit the given freedom in design. In section 6, we will provide a theoretic study of the vehicle dynamics and we are able to theoretically justify our choices in the retrospective. The analysis also suggests some improvements, however we first want to precisely describe our original ad-hoc design, such that this paper is still a valid reference on the Darpa Urban Challenge 2007 implementation and the method can exactly be reproduced.

We briefly detail the geometry of the tentacles used: With $n = 16$ being the number of speed sets, the radius r_k of the kth tentacle in a set is given by

$$r_k = \begin{cases} \rho^k R_j & | \; k = 0, ..., 39 \\ \infty & | \; k = 40 \\ -\rho^{k-41} R_j & | \; k = 41, ..., 80 \end{cases}, \tag{1}$$

where the exponential factor $\rho = 1.15$ and the initial radius R_j of speed set $j = 0, ...15$ is

$$R_j = \frac{l}{\Delta\phi(1 - q^{0.9})} \tag{2}$$

and

$$l = 8\text{m} + 33.5\text{m}\; q^{1.2} \tag{3}$$

is the length of the outmost tentacles, with $q = j/(n-1)$ and $\Delta\phi = 1.2\frac{\pi}{2}$ being the angle subtended by the outmost tentacle of the lowest speed set (the most curved one). The length of the kth tentacle is given by

$$l_k = \begin{cases} l + 20\text{m}\sqrt{\frac{k}{40}} \mid k = 0, ..., 40 \\ l + 20\text{m}\sqrt{\frac{k-40}{40}} \mid k = 41, ..., 80 \end{cases}. \tag{4}$$

For the UC07, the velocity for speed set j was computed by

$$v_j = v_s + q^{1.2}(v_e - v_s), \tag{5}$$

where the speed of the lowest speed set is $v_s = 0.25m/s$ and the maximum speed is $v_e = 10m/s$. This choice has no physical justification other than that the resulting curves are comfortably drivable with the specified speeds. The design was set up empirically and tested by experiments. The motivation for the exponential factor $q^{1.2}$ is to sample low speeds more frequently. Similarly, the reason for the exponential form in (1) is to have more tentacles with small curvatures and a coarser distribution at larger curvatures. The idea is that the tentacle approach should have more options for "fine-motor" primitives. However, there is no physical reason for this choice. A uniform distribution of radii would probably work, too. An interesting idea for the future is not to design the tentacles but to learn the distribution from a human driver by observing and segmenting the trajectories he drives. Equation (2) defines the base radius R_j of each speed set that is the radius of the outmost and most curved tentacle of speed set j. The tentacles that are directed straight need to have a certain length to ensure a sufficient lookahead distance, depending on the speed. When all tentacles of a given speed set would have this minimum length, the outmost tentacles would actually bend behind the vehicle. To avoid this, the outmost tentacles need to be shorter than the ones pointing straight. The reason why the straight tentacles should have a certain length is that we want to allow the selection mechanism to probe a larger portion of space, detecting obstacles far before they cause danger and require immediate braking. Hence, if we ignore the term $(1 - q^{0.9})$ for a moment the initial radius R_j is calculated such that the angular scope of the tentacle arc is $\Delta\Phi$, slightly more than 90 degrees for the slowest speed set. This means, that at very low speeds the vehicle can look around e.g. a road branch, slightly more than 90 degrees. The term $(1 - q^{0.9})$ lets the radii of the outmost tentacles increase with successive speed sets. One could argue that it would have been better to define the outmost radii according to an underlying physical property, e.g limiting the centripetal force at a given speed. However, while such a definition sounds more sophisticated at first glance, it is not pure physics that defines the proper structure. Of course, for safety reasons, we want to stay far beyond the physical limits of the vehicle. But within those bounds, physics might not be the desired property. For instance, restricting or allowing curvatures might also depend on the environment. For instance, when driving on highways, one might want to restrict curvatures far beyond the physical limits, knowing that highways are not built to have those high curvatures. Hence, the design of the tentacles has an ecological perspective, too. According to our design, the lengths of the tentacles then increase from outer

to inner tentacles and from lower to higher speed sets. The given design lets the center tentacle of the highest speed-set together with its classification and support areas exactly reach the far end of the occupancy grid (see figure 6).

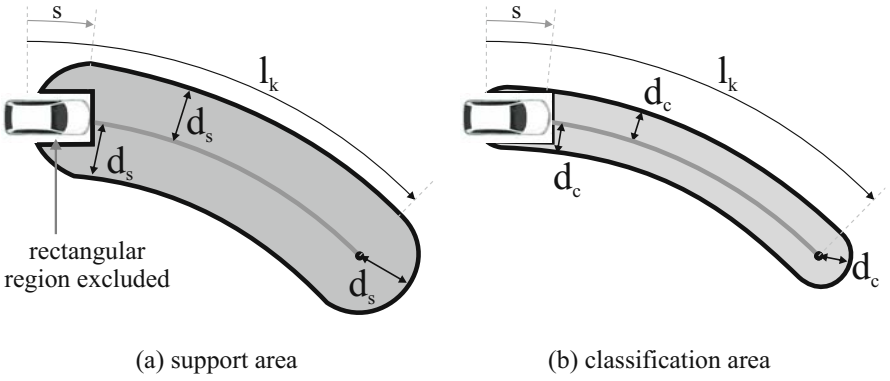
3.2 Support and Classification Area

When a tentacle is evaluated, occupancy cells within a radius d_s to any point on the tentacle are accessed. Those cells form the *support area* of the tentacle. A subset of those cells form the *classification area* comprising cells within a radius $d_c < d_s$ to any point on the tentacle. The geometric definition of these areas is illustrated in figure 7. While the classification area is later used to determine tentacles that are drivable in principle, the support area is used to determine the "best" of all drivable tentacles. For low speeds the classification area is only slightly wider than the vehicle, allowing to drive along narrow roads or through narrow gates. For higher speeds, the width of the area increases. The reason for this is that the additional width has to ensure that the trajectory that results from executing the tentacle is within the corridor defined by the classification area. This is due to the fact that when executing a tentacle, the vehicle will not precisely follow its shape. This is no problem as long as we can guarantee that the resulting path is within the classification area, and hence free of obstacles. A theoretical investigation on this issue will be conducted in section Section 6. In contrast, the support area is much wider than the vehicle. This allows the algorithm to be parameterized in a way, that e.g the vehicle exploits free space and drives in the center of a wide path, instead of driving always close to the border of the path. This behavior might not be desired in some scenarios and can be disabled as explained in later sections. The distances d_s and d_c increase with the speed of the vehicle (speed sets), such that the vehicle would take a larger lateral security distance when driving around obstacles at high speeds or declaring a narrow gate as impassable at high speeds while classifying it as drivable at low speeds.

An important point is that the geometric relation between a specific tentacle and the occupancy grid cells remains always the same, since the grid is specified in the coordinate system of the vehicle. Thus, the cell offset $o = \Delta x \cdot y + x$ ($\Delta x = 512$, the width of the occupancy grid, x, y being cell indices) of the cells (x, y) indexed by a tentacle's support and classification area remain always the same. This static geometric relation allows to precompute all cell offsets for each tentacle and store the n_s cells of the support area as the set $\mathcal{A} = \{\mathbf{c}_0, ..., \mathbf{c}_{n_s-1}\}$, with

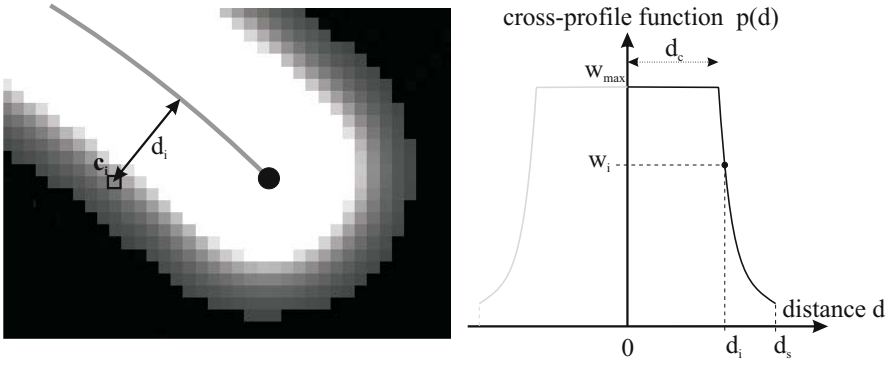
$$\mathbf{c}_i := (o_i, w_i, k_i, f_i). \tag{6}$$

Here, o_i is the offset of the cell, uniquely specifying its position in the grid as described above and allowing fast memory access later. The value w_i is a weight and k_i is a histogram index (both to be detailed later) associated with the respective cell of the support area. The flags f_i are set such that the subset $A_c \subset A$ describing the classification area is given by $A_c = \{\mathbf{c}_i \in A | f_i = 1\}$.



(a) support area (b) classification area

Fig. 7. (a) The support area covers all cells within a distance d_s of the tentacle (start point shifted about s). The cells of the support area are subject to different weights as explained in the main text. (b) The classification area is a subset of the support area covering all cells within a distance $d_c < d_s$ of the tentacle (start point again shifted about s). No weights are specified for the classification area. A rectangular region around the vehicle (no LIDAR measurements are made in this area) is excluded from both the support and the classification area.



(a) (b)

Fig. 8. The weight w_i of a cell $\mathbf{c}_i$ is calculated by passing its distance d_i to the tentacle (see (a)) to the cross-profile function $p(d)$ (see (b) and equation 7).

As illustrated in figure 8 the weight w_i of a cell $\mathbf{c}_i$ is computed by passing the cell's distance d_i to the tentacle (orthogonal distance except at the start and end point of the tentacle) as argument to the cross-profile function

$$p(d) = \begin{cases} w_{max} & | \; d <= d_c \\ \frac{w_{max}}{\kappa + \frac{d - d_c}{\sigma}} & | \; d > d_c \end{cases}, \tag{7}$$

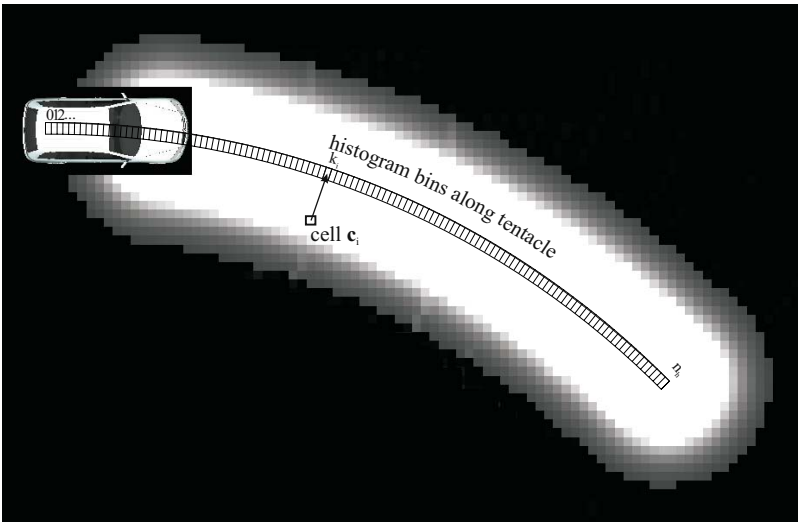


Fig. 9. Each tentacle has a longitudinal histogram with its bins aligned along the tentacle, discretizing the tentacle's length. Grid cells are projected orthogonally onto the tentacle to compute the histogram index k_i of a cell $\mathbf{c}_i$. (see definition 6 on page 402)

where $\kappa = 1, \sigma = 0.16$ and $w_{max} = 10$ in our implementation. For the C-Elrob 2007 and the DARPA Urban Challenge 2007 the value d_c was empirically specified for each speed set j with corresponding velocity v by:

$$d_c = \begin{cases} 1.7m + 0.2m\frac{v}{3m/s} & | \; v < 3m/s \\ 1.9m + 0.6m\frac{(v-3m/s)}{10m/s} & | \; 3m/s < v <= 10m/s \end{cases} \tag{8}$$

A weakness of this design is that it not theoretically but only empirically justified. However, we will provide a theoretical analysis in section 6, showing that our choice was above a required minimum width that can theoretically be justified considering the dynamics of the vehicle.

3.3 Longitudinal Histogram

We aim to binary classify all tentacles whether they are drivable or not. In case they are occupied, we also wish to compute the distance to the first obstacle along the tentacle. For these calculations, we define a histogram for each tentacle with its bins aligned along the curve of the tentacle, discretizing its length into a sequence of n_h bins $0, ..., n_h - 1$ ($n_h = 200$ in our implementation). To determine the cells that contribute to a bins value, we orthogonally project every cell onto the tentacle retrieving the cell's histogram index k_i. To speed up later histogram calculations, all k_i are precomputed (see figure 9). Here, k_i is the second to last value in definition 6 on page 402.

4 Selection Mechanism

With the description of the tentacle related data-structures and their computation being completed, in this section we show how a tentacle is selected with each new LIDAR frame (10 Hz). This tentacle is then used to derive the final trajectory to drive. To decide for the "best" tentacle, first all drivable tentacles are determined. For these valid tentacles, three values are calculated (clearance, flatness and trajectory value). Later, they will be linearly combined to derive a single decision value which is minimized. Our description continues detailing the classification step and the three aforementioned decision affectors. We assume that a new cycle has started and the occupancy grid has been updated with the latest scan.

4.1 Tentacle Classification

Classifying a tentacle whether it is drivable or not happens at the same time as determining the distance to the first obstacle (if any) along that tentacle. The longitudinal histogram provided with each tentacle is used for this purpose: Initially, the bins are all cleared. Then, all cells $\mathbf{c}_i = (o_i, w_i, k_i, f_i) \in A_c$ of the classification area are used to access the occupancy grid with memory offset o_i, obtaining the occupancy value $v_i = g(o_i)$ at the cell's location. Here, $g(o)$ is the function that returns the z-difference of the grid cell at location o. Due to the wheel radius of our vehicle we are interested in obstacles greater than 0.1m. Thus, if v_i exceeds a threshold of $t_c = 0.1m$, the histogram bin k_i is incremented (see figure 9). Only the classification area and not the support area is used for this step, allowing the vehicle to drive through narrow areas being only slightly wider than the vehicle. Also, no weighting takes place in this step, since an obstacle in the classification area will cause damage to the vehicle independent of its lateral position within the classification area. As illustrated in figure 10, an obstacle is detected if the sum of all bins within a sliding window (of n_w bins) exceeds a threshold of n_o (in our implementation $n_w = 5$, $n_o = 2$ and the total number of bins is $n_h = 200$). If no obstacle is detected, the tentacle is classified as drivable. However, a peculiarity of our approach is that if an obstacle is detected, the tentacle is classified as undrivable, solely if the distance to this obstacle is below a so-called *crash distance* l_c. Roughly speaking, if the obstacle is distant, the tentacle is still drivable for some time. Here, the *crash distance* is the distance the vehicle needs to stop using a constant convenient deceleration a plus a security distance l_s. It depends on the speed v of the vehicle and is calculated by

$$l_c = l_s + \frac{v^2}{2a}. \tag{9}$$

In our implementation, $l_s = 6\text{m}$ and $a = 1.5\frac{\text{m}}{\text{s}^2}$. Summarizing, a tentacle is classified as non-drivable, only if an obstacle is within a distance l_c along

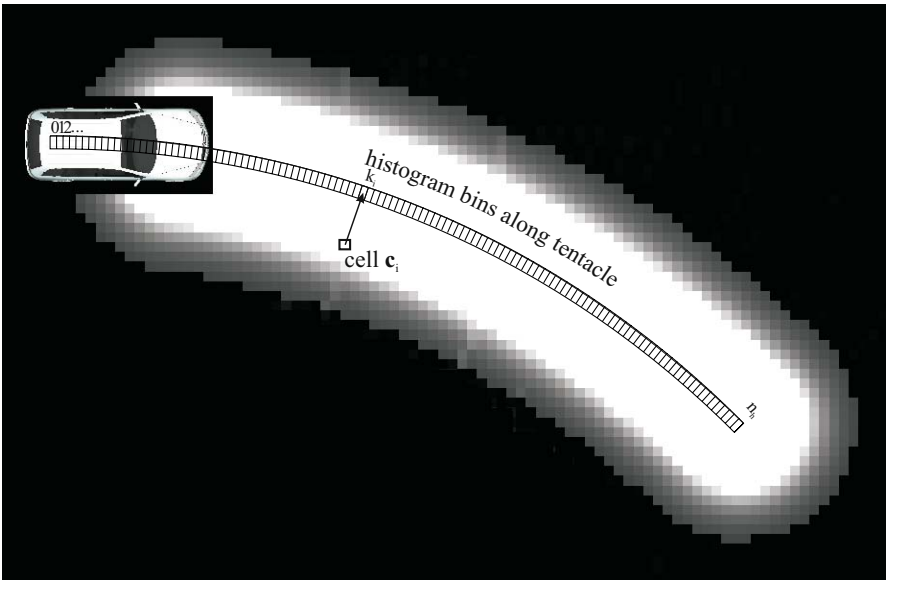


Fig. 10. A sliding window is used to determine the position of the first obstacle. The window is initially placed at bin 0 and successively slid to higher bin indices. If the sum of bin values within this window exceeds a threshold n_o ($n_o = 1$ in our experiments), an obstacle is detected and the position of the sliding window yields the distance l_o to this first obstacle.

that tentacle. While this mechanism might seem like an optional gadget, it is actually very important. To see why, consider the scenario as illustrated in figure 11a. Here, a car is blocking the lane leaving only a narrow passage to drive around the car without hitting either the car or the pavement edges of the road. As can be seen, the geometry of the scenario renders all tentacles occupied by either the car or the road side, and no tentacle is completely free. Hence, an approach that would classify all occupied tentacles as non-drivable would not be able to drive around the car. This is the reason why similar approaches like (Thrun et al., 2006) or (Coombs et al., 2000) require geometries different from arcs. However, when introducing the concept of classifying tentacles as "undrivable" only if an obstacle is within the crash-distance, the case can be handled as illustrated in figure 11b. One might argue, that introducing the crash distance is just the same as defining shorter tentacles. However this is not the case, since tentacles with a more distant obstacle are still preferred. This mechanism will be detailed in the next sections, where we describe the calculation of three decision-affecting values. Those values will only be calculated for tentacles that were classified as drivable, and only these constitute the set the "best" tentacle is selected from.

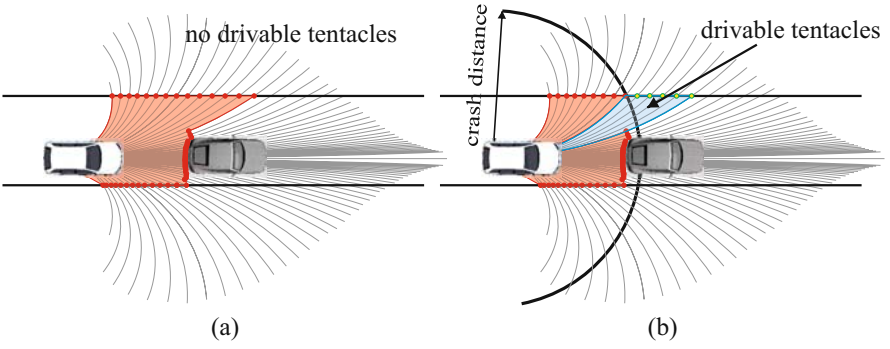


Fig. 11. Both figures (a) and (b) show the case where a car is blocking the right lane of a road and only a narrow passage is left to pass the car. The red points mark the locations along the tentacles where the vehicle would hit either the car or the road border. As can be seen, no tentacle is free of obstacles. Hence, by neglecting the distance to an obstacle, all tentacles would be classified undrivable (a). In contrast, (b) shows that the concept of classifying tentacles as undrivable only in case of being occupied within a speed depending *crash distance* (see main text). In this case, some drivable tentacles remain, allowing to pass the car.

4.2 Braking and Crash Distance

The influence of the crash distance l_c (see (9)) on classifying a tentacle as being drivable and its interplay with a simple braking mechanism is the main reason why our approach accomplishes to use only 81 arcs but still can drive along narrow roads and avoid obstacles in difficult situations. The crash distance can be seen as cutting the evaluation length (only for the purpose of classification) of the tentacle before its end. This mechanism has to be seen in combination with the simple braking mechanism that acts as follows:

If no tentacle is drivable the tentacle with the largest distance to the first obstacle is selected and the vehicle chooses this tentacle for braking. The vehicle then brakes with a constant deceleration along this tentacle. This execution of this *brake tentacle* is only performed for a period of 0.1 seconds, then all tentacles are evaluated anew at the next time step.

Now consider the case that the vehicle is entering a narrow way with a speed set and crash distance that has no tentacle that is drivable, just because the shape of the narrow way doesn't correspond to any of the tentacles. Hence, the vehicle will brake and the velocity and crash distance reduces. However, reducing the crash distance lets some obstacles that were before the old crash distance now be behind the new crash distance, "freeing" some tentacles. In the extreme, the vehicle brakes until it almost stops. However, at low speeds the crash distance is very small, meaning that even quite close obstacles along a tentacle are not considered as making the tentacle undrivable for one time step, simply because there is enough space left in order to postpone braking to a later time. Hence, if a way is narrow, the vehicle typically just slows

down, until the low crash distance allows to select tentacles that don't match the exact shape. This mismatch in shape does not constitute a problem, since only a small fragment of the tentacles is executed, before a new curvature is selected at the next time step.

4.3 Clearance Value

The *clearance value* is the first of three decision-affecting values computed for each drivable tentacle. To make those values comparable they are normalized to the range $[0, ...1]$, where a value of 0 designates a preference of such a tentacle. The calculation of the *clearance value* directly uses the distance to the first obstacle l_o calculated in the classification step. It expresses how far the vehicle could drive along a tentacle before hitting an obstacle. It is calculated by the sigmoid-like function

$$v_{\text{clearance}}(l_o) = \begin{cases} 0 & \text{if the tentacle is entirely free} \\ 2 - \frac{2}{1+e^{-c_{\text{clearance}} \cdot l_o}} & \text{otherwise} \end{cases}, \quad (10)$$

where the constant $c_{\text{clearance}}$ is calculated by (11) to yield $v_{\text{clearance}}(l_{0.5}) = 0.5$ at a distance $l_{0.5} = 20\text{m}$ in our implementation.

$$c_{\text{clearance}} = \frac{\ln 1/3}{-l_{0.5}} \quad (11)$$

As shown by the plot of $v_{\text{clearance}}(l_o)$ in figure 12 the clearance value converges against zero for $l_o \longrightarrow \infty$. The higher the distance to the first obstacle, the less is the change in the clearance value and the change of impact in the tentacle selection process. The clearance value is part of a linear combination (with positive coefficients) of three values that is minimized later, and thus, tentacles with a large distance to the next obstacle are preferred.

4.4 Flatness Value

The *flatness value* has the goal to prefer tentacles leading over smooth terrain. While all tentacles passing the classification step are drivable without causing damage to the vehicle, it might still be desired to prefer a smooth path. The second purpose of the *flatness value* is to exploit free space, if possible: For instance, if there is a broad dirt-road without any obstacle, the flatness value lets the vehicle drive in the center of that path, instead of driving always close to the border of that path. It is computed accessing the whole support area of a tentacle by

$$v_{\text{flatness}} = \frac{2}{1 + e^{-c_{\text{flatness}} \cdot v_{\text{avg}}}} - 1, \quad (12)$$

where

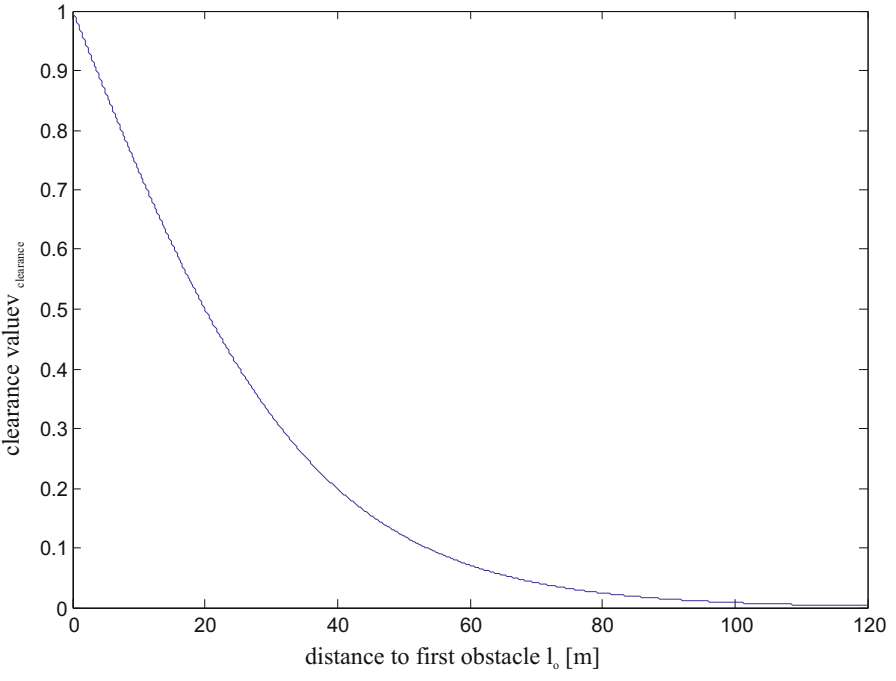


Fig. 12. The plot of $v_{\text{clearance}}(l_o)$ (see equation (10) with $c_{\text{clearance}}$ as in (11)) shows that the higher the distance to the first obstacle, the less is the change in the clearance value and the change of impact in the overall tentacle selection process.

$$v_{\text{avg}} = \frac{\sum_{\mathbf{c}=(o,w,k,f)\in\mathcal{A}} wg(o)}{\sum_{\mathbf{c}=(o,w,k,f)\in\mathcal{A}} w} \tag{13}$$

and c_{flatness} is computed in analogy to (11) such that v_{flatness} reaches a value of 0.5 at $v_{\text{avg}} = 0.3$ in our implementation. The set $\mathcal{A}$ describes the precomputed support area as described on page 402, and $g(o)$ is the value of the occupancy grid at location o.

4.5 Trajectory Value

While the latter two values aim at obstacle avoidance in an unknown environment, the *trajectory value* pushes the vehicle towards following a given trajectory, e.g. defined by GPS waypoints. For each tentacle, a quality of how much the tentacle follows or leads to a given trajectory is calculated. Different distance measures, like the Hausdorff or Frechet distance (Preparata and Shamos, 1985) can be considered. However, the simplest method is to consider a single point on the tentacle and a corresponding point on the trajectory. The point on the tentacle is taken at the crash distance l_c, such that at high speeds the point is more distant than at low

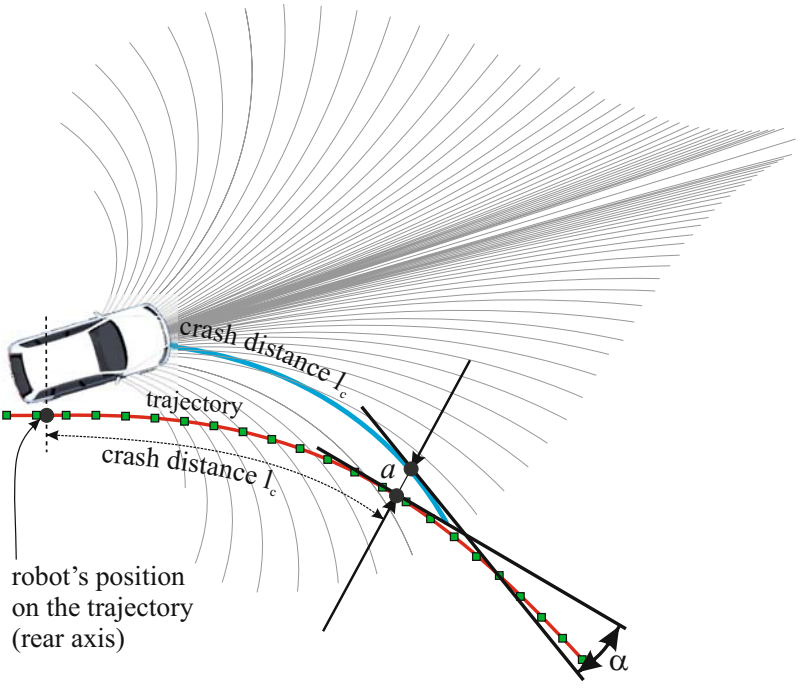


Fig. 13. For each tentacle a *trajectory value* is computed by considering the distance and tangent orientations of two corresponding points - one on the tentacle and the other on the (GPS-)trajectory to be followed.

speeds. If the steering angles of the tentacles are executed directly while following a given set of GPS waypoints, and ignoring the flatness and clearance contributions, the approach is very similar to "Pure Pursuit" (Coulter, 1992) which calculates an optimal circular arc to reach a GPS waypoint from the current robot's position and sets the steering angle to the curvature of the calculated arc. The corresponding point is calculated by first matching the robot on the trajectory as shown in figure 13 and then sampling a point from the trajectory - located at a distance l_c from the matched position. The effect of sampling the points at the crash distance is that the vehicle tries to follow the trajectory more closely at low speeds while at high speeds the vehicle tries to recover a lost track further afar.

For each tentacle the distance measure v_{dist} and finally the *trajectory value* $v_{\text{trajectory}}$ is calculated by taking both the distance a between the point on the tentacle and its corresponding point on the trajectory as well as its relative tangent orientations α into account:

$$v_{\text{dist}} = a + c_\alpha \alpha \tag{14}$$

$$v_{\text{trajectory}} = \frac{v_{\text{dist}} - v_{\min}}{v_{\max} - v_{\min}}. \tag{15}$$

Here, $c_\alpha = 3.0\frac{\text{m}}{\text{rad}}$ is a constant, a and α are illustrated in figure 13 and $v_{\max}, v_{\min}$ are the minimum/maximum values of v_{dist} over all tentacles of the current speed set. Equation (15) produces normalized values within the range of $[0, ..., 1]$. Compared to the normalization procedures of the other two values v_{flatness} and $v_{\text{clearance}}$ the normalization process is not independent of the v_{dist}values of the other tentacles here. The reason is, that if the vehicle would be distant from the trajectory (say 20 m), equation (15) would produce quite high values. When these values would be normalized using a sigmoid-like function (as is the case for v_{flatness} and $v_{\text{clearance}}$) all tentacles would receive similar values for $v_{\text{trajectory}}$, such that when later combined with the flatness and clearance values, the geometric variation of the tentacles with respect to the trajectory would have little influence in the final decision. However, in our case (15) produces normalized values that reflect the tentacles' geometric variation independent of the vehicle's gross distance from the trajectory.

4.6 Combining Clearance, Flatness and Trajectory Values

For each tentacle, classified as *drivable*, the three values $v_{\text{clearance}}$, v_{flatness} and $v_{\text{trajectory}}$ are within the range $0, ..., 1$. They are now linearly combined to a single value

$$v_{\text{combined}} = a_0 v_{\text{clearance}} + a_1 v_{\text{flatness}} + a_2 v_{\text{trajectory}}. \tag{16}$$

Here, a_0, a_1 and a_2 are parameters that can be used to change the behavior of our approach at a gross level. For instance, at the C-Elrob 2007 we used $a_0 = 0$, $a_1 = 1$ and $a_2 = 0$ resulting in that the vehicle chose to freely drive over flat area without following a given trajectory. In contrast, we used $a_0 = 1$, $a_1 = 0$ and $a_2 = 0.5$ for the DARPA Urban Challenge 2007, letting the vehicle follow a given trajectory while avoiding obstacles at the same time. Note, that the primary obstacle avoidance mechanism is not accomplished by the three values $v_{\text{clearance}}$, v_{flatness} and $v_{\text{trajectory}}$ but by the classification step: Only drivable tentacles constitute the set for which v_{combined} is calculated. If no drivable tentacles exist, the tentacle with the largest clearance value is selected, and the vehicle is commanded to brake along this tentacle.

The linear combination of (16) means that different contribution can balance each other if the respective a_i values are non-zero. For instance, consider the case of driving along a dirt road with GPS having a drift of $2m$ such that following the GPS path would cause the vehicle to drive at the very boundary of the dirt road. Assume that the wheels at one side of the vehicle constantly drive slightly aside the road. If there was a pavement edge at this side, the respective tentacles would have been classified as non-drivable and the vehicle would not even have driven into this situation. But let's assume that the obstacles are just stones with their size being at the limit of allowing to drive over them. Hence, let's assume that the tentacles classify the respective areas as being drivable. Now, the contributions in (16) can be seen as forces competing with each other. The trajectory would let the vehicle neglect the small

stones and just drive along the drifted GPS trajectory. The clearance value is difficult to predict and depends on the situation. For instance, if within the area next to the road no high obstacles exist, the clearance value would not prevent the vehicle from continuing to drive aside the road. In contrast, the flatness value gives the vehicle a tendency to prefer smooth areas. In this way, it acts as a force typically pulling the vehicle onto the road, because in most cases the road is the flattest area. By tuning a_0, a_1 and a_2 different behaviors can be produced. The fine tuning however occurs by the parameters of the individual functions $v_{\text{clearance}}$, v_{flatness} and $v_{\text{trajectory}}$. For the UC07, we first tuned our system by experiments testing that our vehicle would drive along narrow passages in the presence of GPS drifts (just simulating them by adding an artificial drift). The important parameter for this is the crash distance l_c (see (9)). Also, we performed various other experiments as described in the experimental results section in order to tune the system. For the UC07, we decided to ignore flatness and drive along GPS if drivable, even when driving over uneven terrain such as smaller rocks, etc. However, this was not a decision without dissentient votes.

To avoid inconsistent selections of tentacles at successive time steps we use the following hysteresis mechanism: We first determine the set S of drivable tentacles with a combined value v_{combined} that is at most $\epsilon = 0.00001$ worse than the value v_m of the best tentacle. From this set of "approximately equally good" tentacles we then finally select the one which is geometrically most similar to the tentacle selected in the last time step. Here, our similarity measure is simply the absolute difference of the tentacles' curvatures. Hence, in ambiguous situations, the one tentacle is selected that is most consistent with the last decision.

5 Tentacle Execution

There exist various options for steering the vehicle according to the selected tentacle.

5.1 Direct Execution

By the fact that each tentacle corresponds to a circular arc and that the the speed of the vehicle is given, a respective steering angle δ_F can be calculated. It is then possible, to directly command this steering angle to a low-level controller at the frequency of the tentacle selection process (10Hz). However, since the tentacles represent discrete curvatures, the resulting driving behavior might be jerky. Hence, it might be desired to generate smoothed steering angles $\delta_s(k)$, e.g by the simple recursive filter

$$\delta_s(k) = \kappa\delta_F(k) + (1-\kappa)\delta_s(k-1), \tag{17}$$

where $\delta_F(k)$ is the steering angle of the current selected tentacle, $\delta_s(k-1)$ is the last smoothed curvature and $0 <= \kappa <= 1$ is a constant. However,

such a filter introduces a temporal delay until a constantly sensed curvature takes effects. At the C-Elrob 2007 we used this simple filtering method with a value of $\kappa = 0.9$. At the Urban Challenge 2007 a different execution mode was used.

We also experimented with a more sophisticated approach using a clothoid model for the road, a bicycle model for the vehicle dynamics and additional tentacles with different lateral displacements and yaw angles. Then selecting a tentacle can be seen as measuring a curvature, a displacement and a yaw angle and a Kalman filtering approach like (Dickmanns, 2007; Dickmanns, 1994) can be used to produce a model-based filtered estimate of the curvature, displacement and yaw angle. However, this is at the cost of using much more tentacles and is either error-prone in case of discrepancies between the road model and the real shape of the environment or requires an explicit recognition of different road models (e.g at crossings).

5.2 Fragment Execution

Another option for executing tentacles is to consider the currently selected tentacle as a trajectory to execute. In this case, the vehicle is controlled by a path controller that derives the steering commands from the geometric relationship between the vehicle's pose in some cartesian space and the trajectory specified in the same coordinate system. For instance, when using GPS, an UTM-coordinate space could be used, and in this case the selected tentacle has to be transformed in that space. Note that the tentacles are originally specified in the vehicle-centered coordinate system. When no GPS is used, a global but drifting position can still be computed by integrating odometric and inertial measurements. The drift can be neglected for tentacle execution, since a tentacle is used only for a small period of time in this *drift space* (0.1s). The advantage of this method is that trajectory execution can be controlled at a higher frequency allowing to compensate local disturbances using IMU information. However, as shown in figure 14, the sequence of tentacles does not necessarily yield a continuous trajectory over time. Deviations from the true trajectory will occur due to effects of time delay, the mass of the vehicle and imprecisions of the low-level controller for the steering angle. Since the new tentacle starts from the new position again, it will start with an offset to the old tentacle not continuing the old trajectory.

5.3 Trajectory Blending

The fact that the sequence of selected tentacles does not form a continuous trajectory can conflict with the underlying path execution controller. Assume for instance, that the path-execution controller uses the lateral displacement of the vehicle with respect to the trajectory as one of its feedback values and that due to some imprecisions a lateral displacement exists just before a new tentacle is selected. Then suddenly, the displacement jumps to a value of zero, because the new tentacle starts with no displacement to the vehicle.

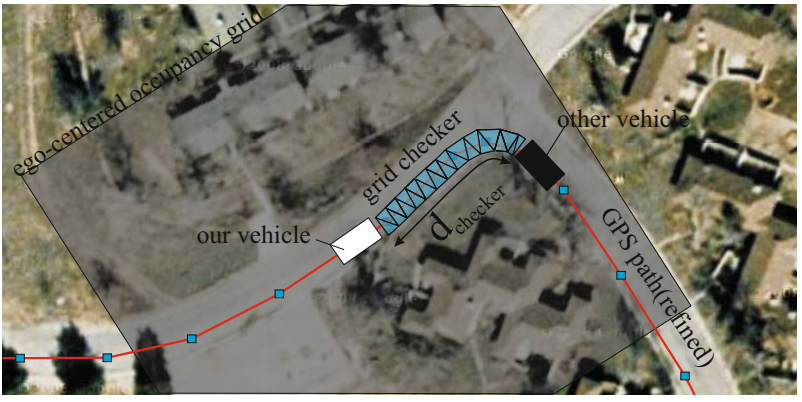


Fig. 14. The sequence of selected tentacles typically does not yield a continuous trajectory, since only a small fraction of a tentacle is actually driven before a new tentacle is selected (The fractions are shorter than shown in this figure. They have been exaggerated here for the sake of clarity.) When a new tentacle is selected it starts at the vehicle's center of gravity again not continuing the old tentacle in case of control deviations.

If the path execution controller includes an integral term this might cause undesired effects, since the controller aims at correcting a deviation that is suddenly reset to zero.

To avoid this discontinuity, we do not directly execute a selected tentacle but calculate a trajectory that continues the old trajectory and blends over to the end of the newly selected tentacle as shown in figure 15.

Let the old trajectory be parameterized as

$$\begin{aligned} \mathbf{p}_{\text{old}} : [0, ..., 1] &\longrightarrow I\!R^2 \\ s &\mapsto \mathbf{p}_{\text{old}}(s), \end{aligned} \tag{18}$$

such that $\mathbf{p}_{\text{old}}(0)$ is the vehicle's reference point on the trajectory as used by the path execution controller and $\mathbf{p}_{\text{old}}(1)$ is the end point of the old trajectory (see figure 15). Let the trajectory of the current selected tentacle be parameterized as

$$\begin{aligned} \mathbf{p}_{\text{tentacle}} : [0, ..., 1] &\longrightarrow I\!R^2 \\ s &\mapsto \mathbf{p}_{\text{tentacle}}(s), \end{aligned} \tag{19}$$

where $\mathbf{p}_{\text{tentacle}}(0)$ is the start and $\mathbf{p}_{\text{tentacle}}(1)$ is the end point of the tentacle. Then the blended trajectory is calculated by

$$\mathbf{p}_{\text{blend}}(s) = \beta(s)\mathbf{p}_{\text{old}}(s) + (1 - \beta(s))\mathbf{p}_{\text{tentacle}}(s), \tag{20}$$

where the blend function $\beta(s)$ is a monotonically increasing bijective mapping

$$\beta : [0, ..., 1] \longrightarrow [0, ..., 1]. \tag{21}$$

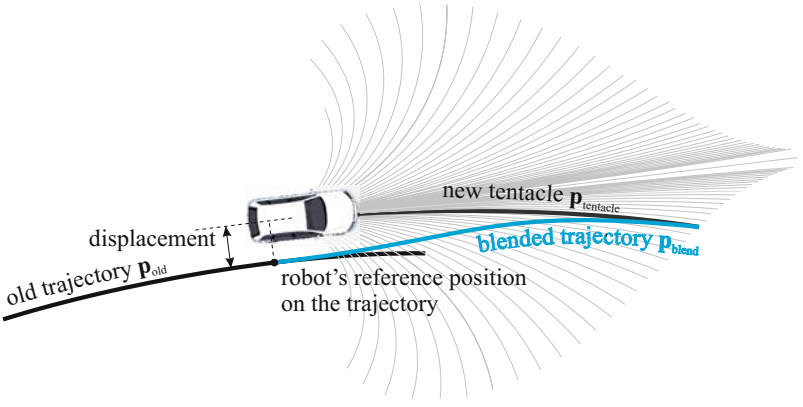


Fig. 15. To avoid discontinuities in path execution when selecting a new tentacle the vehicle does not directly execute the new tentacle but executes a blended trajectory continuing the old trajectory at the vehicle's reference pose and blending over to the newly selected tentacle. The reference pose is the position used by the path execution module to calculate its feedback values for control (e.g. displacement and yaw angle).

In our implementation we used the trivial blend function $\beta(s) = s$. Finally, we pass the blended trajectory to the path execution controller. In the next iteration, this trajectory serves as input $\mathbf{p}_{\text{old}}$ for the next blending step.

6 Considering Vehicle Dynamics

At each time step our method uses the current velocity of the vehicle to choose the closest existing speed set, selects a tentacle from this set and executes the tentacle. In the previous section, we described different options for this execution. At the C-Elrob 2007 we used the direct method and at the Urban Challenge we used fragment execution, that is we commanded the tentacles as trajectories to the low-level path following module. This decision to use the paths was only due to being compatible with the already existing low-level interface to the vehicle. For our dynamic analysis we will stick to the "direct execution" mode. This mode is simpler to analyze, because no control loop for path following has to be included into the analysis. Later, the question will be of how or whether our results are valid for the other execution modes. In this section we will see that our method commits various errors, when considering a tentacle as a piece of trajectory that has to be precisely followed. However, this is not a required goal in our approach and hence the term "error" is inappropriate. In our approach, the final goal is to avoid obstacles and we allow a deviation of the resulting trajectory from the tentacle's trajectory. This is possible because we can show that the deviations are small enough to be within a corridor (corresponding to the classification area of the tentacle),

that is ensured to be free of obstacle. Before we detail this reasoning, it is first necessary to understand the different effects that let the vehicle's final trajectory deviate from the center line of the tentacle. To understand and predict those effects, we have to consider vehicle dynamics.

6.1 Motion Equations

In this section we derive the motion equations to model and predict the dynamics of our vehicle. While it might seem overambitious to derive those equations, in doing so, the variables and symbols we are using are clearly defined and well illustrated from the very beginning. In this way our argumentation is self-contained and comprehensible without having to refer to other literature. Also, our approach differs from the classical formulation (Mitschke and Wallentowitz, 2004) in that we do not make many of the linearization simplifications, but numerically integrate over the non-linear differential equations. Our consideration of dynamics is limited to four-wheeled vehicles steered by two frontal wheels. Not our principal approach but rather our theoretical analysis in this paper is restricted to those type of vehicles. We consider a simplified model of such a vehicle assuming that the center of gravity lies in the ground plane.

In doing so the centrifugal force that acts on the center of gravity doesn't change the load on the wheels. This allows us to reduce the precise geometric layout of the four wheels to the well-known "bicycle model" (Mitschke and Wallentowitz, 2004). Using this model, various assumptions are made, e.g. ignoring roll motions, assuming a uniform load onto inner and outer wheels, etc... For a full enumeration of assumptions see (Mitschke and Wallentowitz, 2004). There exists a well-known closed form solution of the differential equations of the linearized version of the bicycle model (also known as "linearized bicycle model"). However, those equations are only valid for small steering angles, constant velocities and exhibit numerical instabilities for low velocities. Since some of our tentacles require high steering angles and because we want to be able to consider velocity changes, we do not make most of the linearization assumptions of the closed-form solution but derive our own set of non-linear coupled differential equations in the following: For our analysis, we will later solve those equations by numerical integration.

As can be seen in figure 16 a) the velocity $\mathbf{v} = \mathbf{v}_{CP}$ of the vehicle's center of gravity CG is tangent to the trajectory, the same being valid for the tangential acceleration $\dot{\mathbf{v}}$. In contrast, the centripetal acceleration $\frac{v^2}{\rho}$ is directed to the center of curvature M. The distance ρ is the radius of curvature. The angle between $\mathbf{v}$ and the heading direction (direction of the center line) is the sideslip angle β. The yaw angle Ψ is the angle of the heading direction measured against the global x-axis x_0. The course angle of the vehicle is $\nu = \beta + \Psi$.

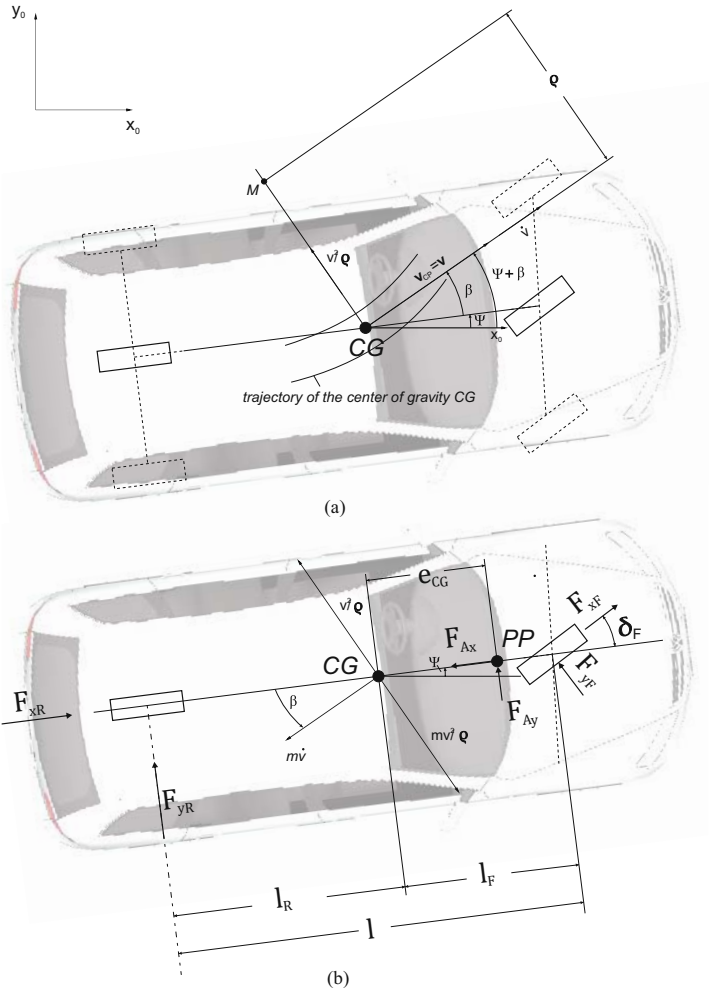


Fig. 16. (a) kinematic entities and (b) forces using a bicycle model. For a description of the symbols see the main text.

Figure 16 b) shows the forces acting on the vehicle. The longitudinal forces F_{xF} and F_{xR} are heading along the direction of the front ("F"=front) and rear wheels ("R"=rear). The lateral force of air ("A"=air) F_{Ay} in case of side wind acts on the pressure point PP, its distance to the center of gravity being denoted with e_{CG}. The air resistance is expressed in terms of the force F_{Ax}.

Using the mass of the vehicle m, the moment of inertia around the z-axis J_z and the steering angle δ_F, we get the following equilibrium of forces in longitudinal direction of the vehicle

$$m\frac{v^2}{\rho}\sin\beta - m\dot{v}\cos\beta + F_{xR} - F_{Ax} + F_{xF}\cos\delta_F - F_{yF}\sin\delta_F = 0, \quad (22)$$

and in lateral direction

$$m\frac{v^2}{\rho}\cos\beta + m\dot{v}\sin\beta - F_{yR} - F_{Ay} - F_{xF}\sin\delta_F - F_{yF}\cos\delta_F = 0. \quad (23)$$

For the balance of moments we get

$$J_z\ddot{\Psi} - (F_{yF}\cos\delta_F + F_{xF}\sin\delta_F)l_F + F_{yR}l_R - F_{Ay}e_{CP} = 0. \quad (24)$$

The lateral forces acting at the front F_{yF} and real wheel F_{yR} emerge due to their slip angles α_R and α_F - the angle between the velocity vector at the wheel and the orientation of the wheel as depicted in figure 17. The corresponding linearized relationships between slip angle and lateral forces are

$$F_{yF} = c_{\alpha F}\alpha_F \quad (25)$$
$$F_{yR} = c_{\alpha R}\alpha_R \quad (26)$$

where the lateral constant force coefficients $c_{\alpha F}$ and $c_{\alpha R}$ with physical units $[N/rad]$ depend on the tire. We aim at expressing the slip angles α_R and α_F as functions of the velocity vector $\mathbf{v}$ of the vehicle's center of gravity, the yaw rate $\dot{\Psi}$ and the sideslip angle β. To derive these dependencies consider figure 17 and the fact that the components of the velocity vectors $\mathbf{v}$,$\mathbf{v}_R$ and $\mathbf{v}_F$ in the longitudinal direction of the vehicle have to be equal. This is simply because the vehicle can't stretch, resulting in:

$$v\cos\beta = v_R\cos\alpha_R \quad (27)$$
$$v\cos\beta = v_F\cos(\delta_F - \alpha_F), \quad (28)$$

where v, v_R and v_F are the lengths of the vectors $\mathbf{v}, \mathbf{v}_R$ and $\mathbf{v}_F$. The velocity components in lateral direction to the vehicle's center line differ as a consequence of the yaw rate $\dot{\Psi}$ evolving different velocity contributions over the lengths l_R and l_F:

$$v_R\sin\alpha_R = l_R\dot{\Psi} - v\sin\beta \quad (29)$$
$$v_F\sin(\delta_F - \alpha_F) = l_F\dot{\Psi} + v\sin\beta \quad (30)$$

With $\tan x = \sin x/\cos x$ we combine the two pairs of equations, yielding

$$\tan\alpha_R = \frac{l_R\dot{\Psi} - v\sin\beta}{v\cos\beta} \quad (31)$$

$$\tan(\delta_F - \alpha_F) = \frac{l_F\dot{\Psi} + v\sin\beta}{v\cos\beta} \quad (32)$$

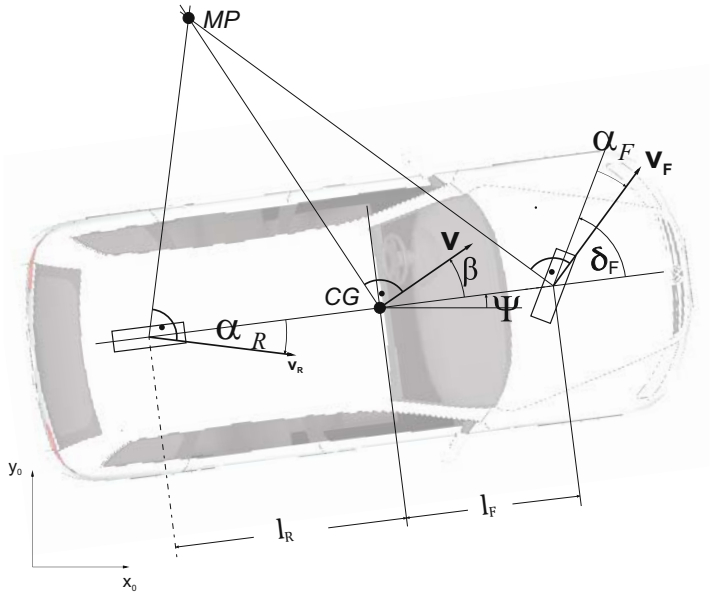


Fig. 17. The slip angles α_R and α_F are the respective angular discrepancies between the velocity vector of a point at the center of the wheel and the orientation of the respective wheel. The lateral forces F_{yR} and F_{yF} that act on each wheel depend on these angles. The pole MP is in general different from the center of curvature M in figure 16a).

Resolving for α_F and α_R yields

$$\alpha_R = \arctan \frac{l_R \dot{\Psi} - v \sin \beta}{v \cos \beta} \tag{33}$$

$$\alpha_F = \delta_F - \arctan \frac{l_F \dot{\Psi} + v \sin \beta}{v \cos \beta}. \tag{34}$$

Inserting the derived equations for α_R and α_F into 26 and 25, the lateral forces can be expressed as functions of $\beta, \dot{\Psi}$ and v:

$$F_{yR}(\beta, \dot{\Psi}, v) = c_{\alpha R} \arctan \frac{l_R \dot{\Psi} - v \sin \beta}{v \cos \beta} \tag{35}$$

$$F_{yF}(\beta, \dot{\Psi}, v, \delta_F) = c_{\alpha F}(\delta_F - \arctan \frac{l_F \dot{\Psi} + v \sin \beta}{v \cos \beta}) \tag{36}$$

Next, we want to express the centripetal acceleration $\frac{v^2}{\rho}$ in 22 and 23 in terms of the velocity v, the sideslip rate $\dot{\beta}$ and the yaw rate $\dot{\Psi}$. The curvature of the trajectory at the center of gravity of the vehicle is $\frac{1}{\rho}$. Another way to express the curvature of the trajectory is to consider the change of course

angle $d(\beta + \Psi)$ along an infinitesimal step $du = vdt$ within the infinitesimal time step dt along the trajectory, equating to

$$\frac{1}{\rho} = \frac{d(\beta + \Psi)}{du} = \frac{d(\beta + \Psi)}{vdt} = \frac{\dot{\beta} + \dot{\Psi}}{v}. \tag{37}$$

Hence, the centripetal acceleration can be expressed by

$$\frac{v^2}{\rho} = v^2 \frac{\dot{\beta} + \dot{\Psi}}{v} = v(\dot{\beta} + \dot{\Psi}). \tag{38}$$

Inserting (35),(36) and (38) into (22), (23) and (24) yields the following balances:

$$mv(\dot{\beta} + \dot{\Psi}) \sin\beta - m\dot{v}\cos\beta + F_{xR} - F_{Ax} + F_{xF}\cos\delta_F - F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\sin\delta_F = 0 \tag{39}$$

$$mv(\dot{\beta} + \dot{\Psi})\cos\beta + m\dot{v}\sin\beta - F_{yR}(\beta, \dot{\Psi}, v) - F_{Ay} - F_{xF}\sin\delta_F - F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\cos\delta_F = 0 \tag{40}$$

$$J_z\ddot{\Psi} - (F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\cos\delta_F + F_{xF}\sin\delta_F)l_F + F_{yR}(\beta, \dot{\Psi}, v)l_R - F_{Ay}e_{CP} = 0. \tag{41}$$

Multipyling (40) with $sin\beta$ and adding (41) multiplied with $\cos\beta$ yields

$$\begin{aligned} &mv(\dot{\beta} + \dot{\Psi})(\sin^2\beta + cos^2\beta) + \sin\beta[F_{xR} - F_{Ax} + F_{xF}\cos\delta_F - \\ &F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\sin\delta_F] + \cos\beta[-F_{yR}(\beta, \dot{\Psi}, v) - F_{Ay} - F_{xF}\sin\delta_F - \\ &F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\cos\delta_F] = 0 \end{aligned} \tag{42}$$

$$\begin{aligned} &\dot{\beta} = -\dot{\Psi} - \frac{\sin\beta[F_{xR} - F_{Ax} + F_{xF}\cos\delta_F - F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\sin\delta_F]}{mv} - \\ &\frac{\cos\beta[-F_{yR}(\beta, \dot{\Psi}, v) - F_{Ay} - F_{xF}\sin\delta_F - F_{yF}(\beta, \dot{\Psi}, v)\cos\delta_F]}{mv} = \\ &: f_2(\beta, \dot{\Psi}, v, \delta_F, F_{xF}, F_{xR}, F_{Ax}, F_{Ay}) \end{aligned} \tag{43}$$

Resolving (41) for $\ddot{\Psi}$ yields

$$\begin{aligned} &\ddot{\Psi} = \frac{1}{J_z}[(F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\cos\delta_F + F_{xF}\sin\delta_F)l_F - \\ &F_{yR}(\beta, \dot{\Psi}, v)l_R + F_{Ay}e_{CP}] =: f_1(\beta, \dot{\Psi}, v, \delta_F, F_{xF}, F_{Ay}). \end{aligned} \tag{44}$$

Multipyling (40) with $cos\beta$ and subtracting (41) multiplied by $\sin\beta$ yields

$$\begin{aligned} &-m\dot{v}(\cos^2\beta + \sin^2\beta) + \cos\beta[+F_{xR} - F_{Ax} + F_{xF}\cos\delta_F - F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\sin\delta_F] - \\ &\sin\beta[-F_{yR}(\beta, \dot{\Psi}, v) - F_{Ay} - F_{xF}\sin\delta_F - F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\cos\delta_F] = 0 \end{aligned} \tag{45}$$

Solving for $\dot{v}$ yields

$$\dot{v} = \frac{\cos\beta[+F_{xR} - F_{Ax} + F_{xF}\cos\delta_F - F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\sin\delta_F]}{m} - \frac{\sin\beta[-F_{yR}(\beta, \dot{\Psi}, v) - F_{Ay} - F_{xF}\sin\delta_F - F_{yF}(\beta, \dot{\Psi}, v, \delta_F)\cos\delta_F]}{m} = : f_3(\beta, \dot{\beta}, \dot{\Psi}, v, \delta_F, F_{xF}, F_{xR}, F_{Ax}, F_{Ay}) \quad (46)$$

Summarizing, we have the set of three coupled non-linear differential equations

$$\ddot{\Psi} = f_1(\beta, \dot{\Psi}, v, \delta_F, F_{xF}, F_{Ay}) \quad (47)$$

$$\dot{\beta} = f_2(\beta, \dot{\Psi}, v, \delta_F, F_{xF}, F_{xR}, F_{Ax}, F_{Ay}) \quad (48)$$

$$\dot{v} = f_3(\beta, \dot{\beta}, \dot{\Psi}, v, \delta_F, F_{xF}, F_{xR}, F_{Ax}, F_{Ay}) \quad (49)$$

6.2 Numerical Integration

Given an initial velocity $v(t_0)$, initial yaw rate $\dot{\Psi}(t_0)$ and sideslip angle $\beta(t_0)$ at time step $t_0 = 0$, our goal is to numerically calculate the functions $v(t), \dot{\Psi}(t)$ and $\beta(t)$ for a duration of $T = 0.1s$ (one LIDAR frame), since those functions can then be used to calculate the trajectory of the vehicle. The control commands are the steering angle $\delta_F(t)$ and the force $F_{xR}(t)$ for velocity control. We assume a rear wheel drive setting $F_{xF} = 0$. We ignore lateral wind $F_{Ay} = 0$. For numerical integration we use explicit forward Euler with temporal step size $dt = 1.0^{-3}s$. The overall integration scheme is:

$$\begin{aligned}
\dot{\Psi}(t_k + dt) &= \dot{\Psi}(t_k) + f_1(\beta(t_k), \dot{\Psi}(t_k), v(t_k), \delta_F(t_k), F_{xF} = 0, F_{Ay} = 0)dt \\
\beta(t_k + dt) &= \beta(t_k) + f_2(\beta(t_k), \dot{\Psi}(t_k), v(t_k), \delta_F(t_k), F_{xR}(t_k), F_{xF} = 0, F_{Ax})dt \\
v(t_k + dt) &= v(t_k) + f_3(\beta(t_k), \dot{\beta}(t_k), \dot{\Psi}(t_k), v(t_k), \delta_F(t_k), F_{xR}(t_k), \\
&\quad F_{xF} = 0, F_{Ax}, F_{Ay} = 0)dt
\end{aligned}$$

The trajectory $x(t), y(t)$ of the vehicle's center of gravity is then computed by numerical integration of

$$x(t) = x(t_0) + \int v(t)\cos(\beta(t) + \Psi(t))dt \quad (50)$$

$$y(t) = y(t_0) + \int v(t)\sin(\beta(t) + \Psi(t))dt \quad (51)$$

For all our numerical integrations we use the following parameters:

- mass $m = 2900kg$
- inertial moment $J_z = 5561kgm^2$
- lateral force coefficient for front wheels $c_{\alpha F} = 80000N/\text{rad}$
- lateral force coefficient for rear wheels $c_{\alpha F} = 110000N/\text{rad}$
- distance from center of gravity to front axis $l_f = 1.425$

- distance from center of gravity to rear axis $l_f = 1.425$ (center of gravity right between the axes)
- integration step width $dt = 0.001s$

Since we have a low-level controller for the velocity and only consider slow changes in the velocity we ignore the longitudinal forces, that is we set $F_{xR} = 0, F_{xF} = 0, F_{Ax} = 0$. We also ignore lateral wind, which is simply unknown, hence $F_{Ay} = 0$.

6.3 Can the Curvatures of Our Tentacles Be Executed?

When our vehicle drives with a constant velocity v and steering angle for some time period, the state $(\dot{\Psi}, \beta, v)^T$ of the vehicle becomes steady and the vehicle drives on a circle. This steady state and its geometric relation to the final circular trajectory is depicted in figure 18a). Note, that the steering angle δ_F is not part of the system state. Rather it can be calculated from a given system state. For the closed-form solution of the linearized bicycle model there exists an analytical solution for calculating the state $(\dot{\Psi}, \beta, v)^T$ and the corresponding steering angle δ_F. Indeed, the yaw rate $\dot{\Psi}$ - with $c = 1/r$ being the curvature of the circle with radius r - is quite simple to calculate:

$$\dot{\Psi} = cv \tag{52}$$

However calculating the sideslip angle β and the required steering angle δ_F is more complicated: In the case of our non-linear model we use a recursive search algorithm to calculate the values. The search algorithm exploits the fact that once the steering angle δ_F is known, one can run a simulation (by integrating the differential equations), initializing the state to $(0, 0, v)^T$ and commanding the constant steering angle. Simulating over a period of T one can observe whether the state becomes steady (by continuously comparing the current state and a state ΔT_{back} back in time) and read out the values for $\dot{\Psi}$ and β in this case (v is known from the very beginning). The search algorithm then divides the range of potential solutions for δ_F $(-0.5\pi, ...0.5\pi)$ into n intervals (n=10 in our implementation), determines the steady states (in case of convergence) for each of the $n + 1$ interval boundaries, finds the interval that encloses the solution (by comparing the respective curvatures that can be computed via (52)) and recursively narrows down the interval. This method is possible because of the monotonic dependency of the steering angle δ_F and the resulting steady curvature $\frac{\dot{\Psi}}{v}$.

Since each tentacle is a circular arc, we can calculate the vehicle's corresponding steady state with the above method such that the vehicle drives a circle with the same curvature. We will call this steady state corresponding to the tentacle **state of the tentacle**. The velocity of this state is the velocity of the tentacle's speed set. At first glance, it is surprising that even if the vehicle's state is initialized such that the final trajectory has the curvature of the tentacle, the resulting trajectory is rotated against the tentacle (see figure 18b).

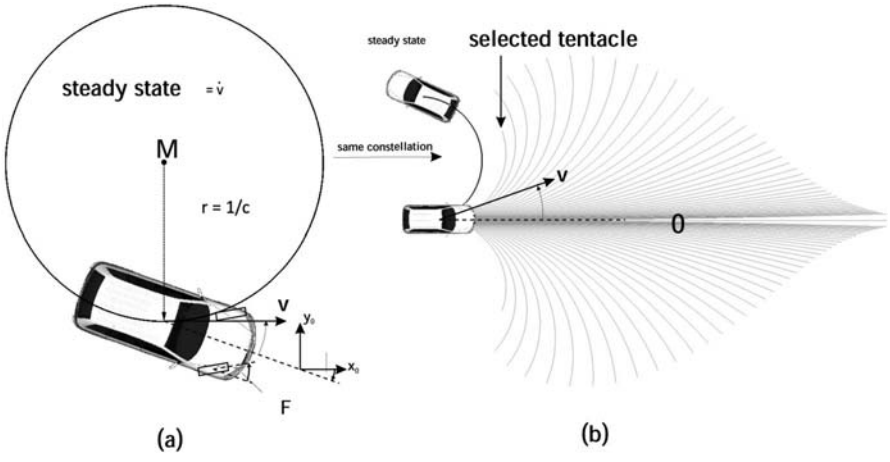


Fig. 18. (a) Given a circle with radius r and a velocity v, a steady state $(\dot{\Psi}, \beta, v)^T$ and a steering angle δ_F can be calculated, such that the vehicle drives on the circle. (b) When calculating the steady state that corresponds to a tentacle and assuming that the vehicle has the correct state from the very beginning, the vehicle's resulting trajectory doesn't correspond to the tentacle. The reason lies in the sideslip angle β.

Even under perfect conditions, the vehicle cannot drive along the tentacle. The reason is, that driving constantly along a circle requires a non-zero sideslip angle β, meaning that in contrast to the velocity vector the vehicle is not heading tangential to the circle. Thus, when starting as illustrated in figure 18b), the final trajectory is rotated against the selected tentacle. The radius of the resulting circular trajectory is correct, however. Hence, we can store with each tentacle a steering angle, that at least produces the correct curvature once the state is steady. The error is not in a wrong system state, but the initial geometric constellation. Right at the beginning the state of the vehicle has a non-zero sideslip angle β. Hence, if both trajectories should be aligned, the vehicle would have to start rotated about $-\beta$. This is illustrated in figure 19.

6.4 Determining Path Deviations

In general, the state of the vehicle before executing a new tentacle will not be the steady state of the new tentacle. Also, the steering angle δ_F will not correspond to the tentacle's curvature at the beginning. To predict the error that happens in such cases, we aim at predicting the trajectory the vehicle executes and comparing this trajectory against the tentacle's curve. To calculate the resulting trajectory, we assume that the steering angle is controlled to the goal steering angle δ_F by a low-level controller that linearly adjusts δ_F to the desired value. In our system this controller has a maximum rate of $0.3\text{rad}/s$. Note that on MuCAR-3 the steering angle controller was itself based on an angular rate controller. For the vehicle of Team AnnieWay

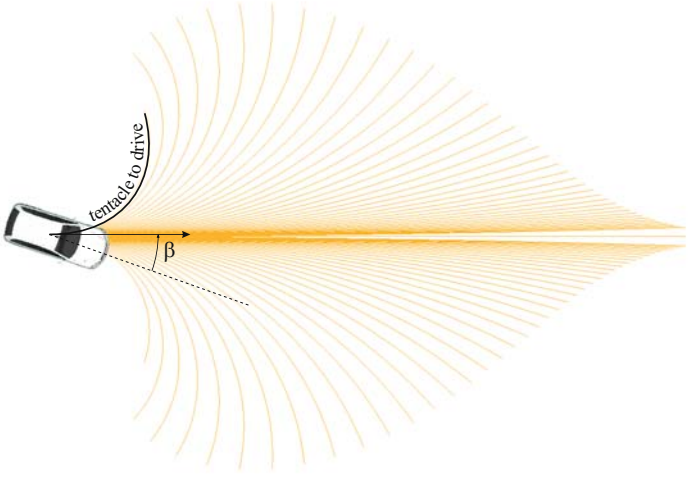


Fig. 19. Given a tentacle the vehicle should precisely drive, the tangent at the first point of the tentacle has to be aligned with the expected sideslip angle β . In other words, the whole tentacle has to be rotated about β in order to be precisely executable. This is a first potential improvement suggested by this analysis. Note, that this doesn't mean that the tentacles as designed in section 2 are invalid, since the resulting path will be shown to still be in the corridor of the classification area. Also this figure shows the an extreme case with a large sideslip angle β. The tentacle considered is the most curved one of all tentacles in all speed sets.

the angular rate controller was itself the lowest unit. A complication arises due to the fact that in the direct execution mode the goal steering angle is not that of the tentacle, but the value calculated by the recursive formula (17) on page 412.

6.4.1 Worst Case Transitions

To calculate upper bounds for path deviations, we will now proceed by considering transitions from a source tentacle t_s to a destination tentacle t_d. First, we only consider tentacles from the same speed set and we assume that the vehicle has initially acquired the steady state of the source tentacle and that it starts at the correct angle $\Psi(t_0) = -\beta$[2] such that the resulting trajectory perfectly aligns with the source tentacle. Then we switch to the second tentacle, calculating the filtered goal steering angle by (17). When the transition to the new steering angle is executed, we simulate the low-level controller changing the steering angle at maximum rate ($0.3\text{rad}/s$).

The assumption that the vehicle initially has reached the steady state of the first tentacle is not true in general. However, if we select the extreme case for an initial tentacle - that is the outmost left or outmost right one - and

[2] Not $\dot{\Psi}(t_0)$!, see (51) and figure 19.

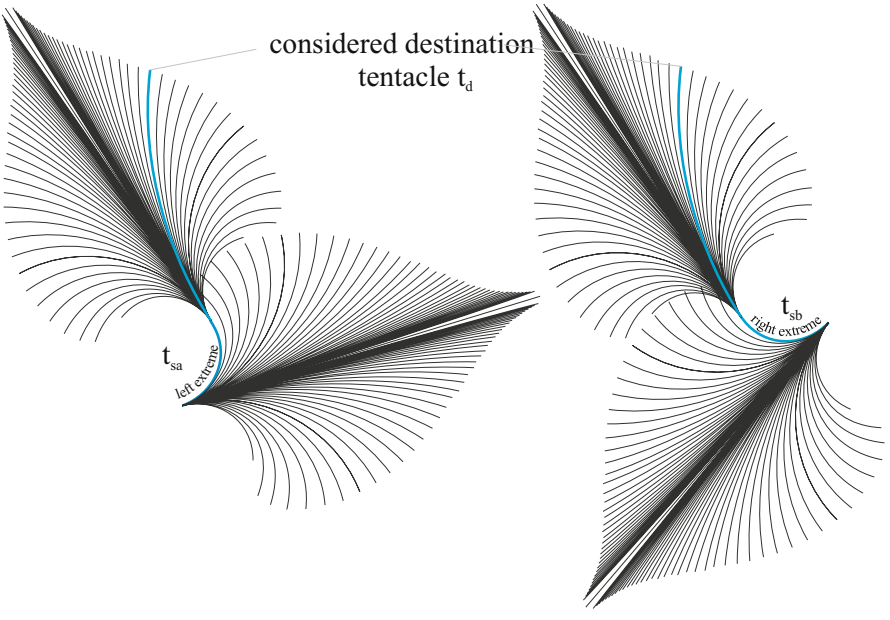


Fig. 20. This figure shows two cases, in each of which two tentacles are executed sequentially. In both cases the top tentacle is the same destination tentacle t_d considered and only the first tentacle (source tentacle) varies (t_{sa} or t_{sb}). For a given destination tentacle t_d, we consider the left and right extreme transitions to this tentacle. We assume that the vehicle has reached a steady state on the source tentacle and then executes the destination tentacle. The source tentacles t_{sa} and t_{sb} are shown in its full length such that they can be better identified within their speed set. However, a tentacle is executed only for the short time period of $0.1s$, hence the first tentacle will not be completely driven as shown in this figure. Only a small fragment of the first tentacles is driven and the second tentacle (the tentacle considered) directly follows this small fragment. Before a transition to the destination tentacle is initiated, we assume that the vehicle is in the steady state that corresponds to the respective source tentacles such that the resulting trajectories and the source tentacles align perfectly. As shown in figure 19 this requires the vehicle to start rotated about $-\beta$ -the expected sideslip angle- in relation to the start orientation of the respective source tentacles.

consider the transition from this tentacle, then this represents an upper bound on the deviation of the resulting path. Figure 20 shows the considered worst case scenarios and the tentacle's trajectories against which we want to compare the simulated trajectories. The assumption made here is that the larger the deviation in the steady states of the source and destination tentacle is, the larger is the resulting deviation of the trajectory. This assumption seems to be valid for the normal case, that is within the scope of validity of the bicycle model. To simulate the results we will let the vehicle drive along the

source tentacle for a time period of T_s and then switch to the destination tentacle in the above described manner, executing it for a time period of T_d seconds. While the length of the first time period T_s doesn't matter, since the path deviation is zero, the second tentacle is executed only for one LIDAR frame ($0.1s$), because then a new tentacle is selected again. Nevertheless, to see the qualitative behavior of the deviations well, we will first consider the full length execution of the second tentacle. Note that executing a tentacle longer than $0.1s$ means that the tentacle is repeatedly selected and hence, the smoothing filter (17) has to be applied consecutively every 0.1 seconds. For (17) we will use a value of $\kappa = 0.9$. Figure 21 shows the resulting trajectories when simulating the cases shown in figure 20. At low speeds the results can

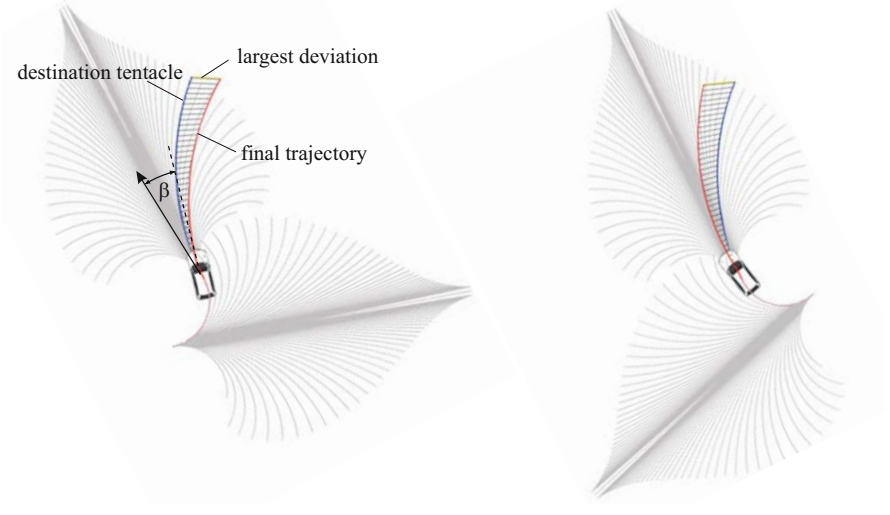


Fig. 21. The resulting trajectories for the low velocity of $0.25m/s$ simulated with the non-linear bicycle model when executing the worst combinations shown in figure 20 using the direct execution mode with $\kappa = 0.9$ (see (17)) and a steering rate limitation of $0.1\text{rad}/s$. For low velocities the result is counter-intuitive: While one might expect that the final trajectory would be left of the destination tentacle (as it is the case in figure 22) on the left hand side, the resulting path is deviated to the right. The reason is the sideslip angle: While driving on the source tentacle the sideslip angle β is quite high, because of the high curvature. When the vehicle switches to the destination tentacle it has a heading direction pointing rightwards of the destination tentacle. Because of the low speed, the new steering angle is adapted without the vehicle having moved far and the curve of the destination tentacle starts out at an angle that lets the final trajectory be located at the right side of the destination tentacle. At the right hand side the opposite case happens. This behavior appears at only low speeds. The lines connecting the destination tentacle and the final trajectory show the deviations by interconnecting corresponding points at same lengths along the respective curves. The largest deviation is used to represent the overall deviation of the two trajectories.

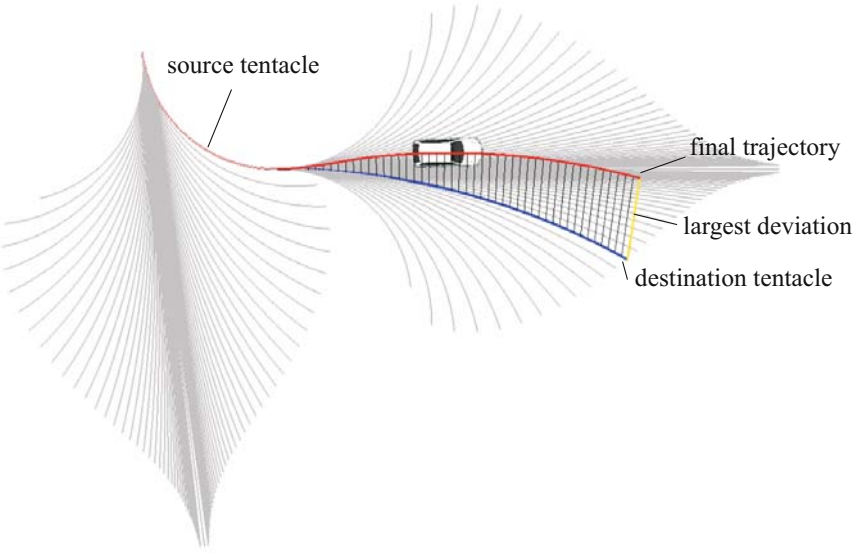


Fig. 22. In this example the velocity is $1.66m/s$ and the deviation of the resulting trajectory is as one would intuitively expect. A counter-intuitive example is shown in figure 21.

be quite counter-intuitive: The large sideslip angle driving the source tentacle can let the final deviated trajectory appear at an intuitively unexpected side of the destination tentacle (see figure 21). The more intuitive case that occurs at higher velocities is shown in figure 22.

6.4.2 Calculating Deviations

We define a distance measure for the deviations by considering the vehicle's true center of gravity at time t while traveling along the true trajectory and a second point on the destination tentacle at the same travel length (but along the destination tentacle). We consider all correspondences while integrating over the differential motion equations. From all corresponding pairs of points and their distances, the largest distance is taken as deviation between the two curves. In figure 21 and 22 some of those corresponding points are sampled and shown by interconnecting them with straight line segments. The maximum deviations often occur at the end of the curves, however there exist cases where the curves intersect, and because of this non-monotonic behavior, we need to consider all correspondences. The measure is conservative in the sense that it does not only punish deviations in space, but also deviations in time. This is because the correspondences are established by means of the velocity and time depending traveled length from the beginning of both respective curves.

In figures 21 and 22 we considered the deviation of executing the full length of the destination tentacle. Now we are interested in the maximum deviation that can occur for each tentacle within one LIDAR frame, that is within 0.1 seconds. For this sake, we consider each tentacle in each speed set as a destination tentacle in the sense of the above described evaluation and calculate the maximum deviation that can happen within this 0.1 seconds. We only need to consider one extreme tentacle and calculate the deviations to all the other tentacles, because of the symmetry of the tentacles. The other worst case then is automatically included in terms of the deviation of the corresponding mirrored destination tentacle. Figure 23 shows the resulting deviations $dev(j,i)$ for all speed sets j and all tentacles i. Interestingly, the deviation increases first with higher velocities and than decreases again from speed set 6 on ($3.5m/s$). As expected, the deviation is larger for more curved tentacles. However, the important fact is that for each tentacle, we have calculated an upper bound for a deviation. For every tentacle, this upper deviation is the maximum of the both extreme transitions to this tentacle. In terms of figure 23 the upper bound d_{ji} for tentacle i in speed set j is

$$d_{ji} := max(dev(j,i), dev(j, n_j - i)), \tag{53}$$

where n_j ($n_j = 81$ for all speed sets j in our case) is the number of tentacles in speed set j. As can be seen, within 0.1s, no deviation exceeds 6 cm. If the classification area of the respective tentacle is about d_{ji} larger than the vehicle, then we can ensure that within the 0.1s the vehicle will not collide with a (static) obstacle. This is because a tentacle is classified as non-drivable if an obstacle is within that area. It remains to show, that this can neither happen at the next time step.[3]

Note, that this tolerance of $6cm$ was well included in our definition of the classification areas we used for the UC07 (see (8)). Here, even the smallest width of any classification area exceeds $2 \cdot 1.7m = 3.4m$ but the VW-Passat is only about $2.01m$ wide.

6.5 Considering an Extreme Case

We showed that, if a tentacle is selected - and therefore was classified as drivable before - the method guarantees that the vehicle will not hit an obstacle within the next 0.1 seconds. However, it is not automatically given that no collision can happen in the successive frames. To avoid collisions the braking behavior plays an essential role. For the UC07, we use the simple mechanism, that if no drivable tentacle was available, the vehicle selected the tentacle with the largest distance to the first obstacle and decelerated with $a = -0.5g$ for the next 0.1 second frame along that **brake tentacle**.

[3] Note, that only static obstacles are considered here. How the approach is combined with a separate obstacle tracker for moving objects (as was the case for the UC07) is detailed in the results section.

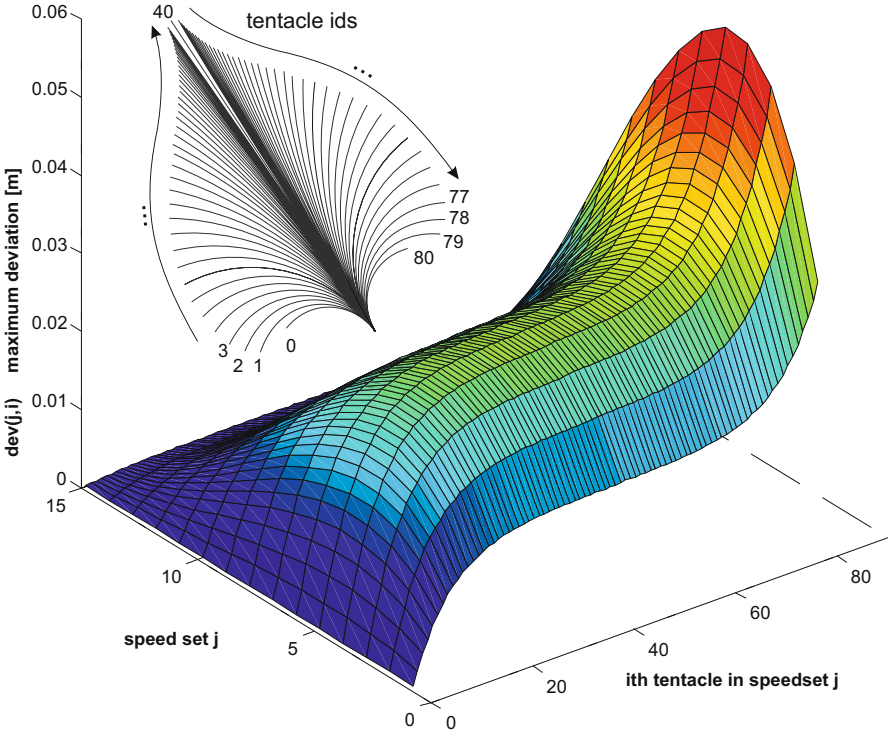


Fig. 23. This figure shows trajectory deviations computed while letting a vehicle perform tentacle transitions, executing the destination tentacle for 0.1 seconds. The considered transitions are from the most curved tentacle 0 of a given speed set (left side of the plot) to all the other tentacles i of the same speed set j. As can be seen, the ego-transition from tentacle 0 to tentacle 0 has an expected deviation of zero. The maximum deviation is below 6 cm. Transitions from one speed set to another are not considered, assuming that the vehicle performs only slow velocity changes.

The speed reduction leads to decreased crash distance and - depending on the case - can free new tentacles or, if all obstacles are too close, lead to a continuation of braking. Freeing tentacles can happen by the reduction of the crash distance itself, letting some obstacles move behind the new distance or by switching to a new speed set and freeing new geometric primitives. The intricate cases do not emerge from obstacles in front of the vehicle, because the crash distance is designed such that there is plenty of room for stopping, but from those obstacles the vehicle passes closely by. This shall be shown at an example. Consider the admittedly very unrealistic case shown in figure 24. In the considered case the classification area of the last selected tentacle is enclosed by a wall, such that the classification area is still free of obstacles. Also the wall cuts the tentacle just behind the crash distance, such that the vehicle detects the obstacle, but considers it being still too distant to be a

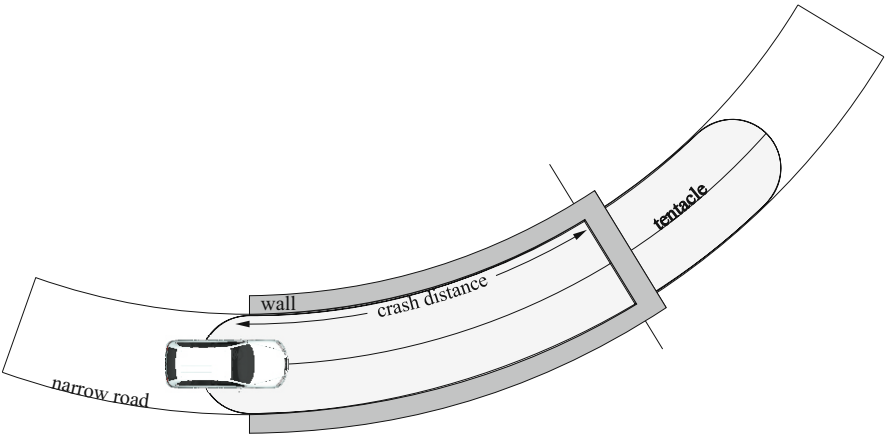


Fig. 24. The constructed case, where the vehicle follows a narrow road with exactly the same curvature than the tentacle the vehicle uses. The road has exactly the width of the classification area, such that the wall is not recognized as obstacle at the sides and neither at the front, since the wall is at a distance slightly above the crash distance. Hence, any little movement of the car, will bring an obstacle into the classification area. This case is used in the main text to discuss whether our method could potentially fail. It is evident, that only the selected tentacle will be classified as drivable, all the other tentacles intersect the wall before the crash distance.

threat for the current speed. Hence, the vehicle drives along the tentacle for 0.1 seconds. If the execution of the tentacle is perfect, no problem arises, because the crash distance will be undercut at the next time frame and the vehicle can safely brake using the remaining part of the crash distance. That is the space for braking only reduces by $v \cdot 0.1s$ if v is the speed of the vehicle. However, if the vehicle deviates from the tentacle's trajectory in terms of an angular deviation, parts of the wall can suddenly protrude into the classification area at a distance far before the crash distance reduced by $v \cdot 0.1s$. This situation is shown in figure 25. Since the wall protrudes into the classification area, the tentacle is classified as non-drivable. It is evident that all the other tentacles are classified as non-drivable, too. Hence the vehicle will decelerate for the next 0.1 seconds choosing the tentacle with the largest distance to the first obstacle as the path for braking. Braking with a deceleration of $a = -0.5g \approx -0.5 \cdot 9.81 m/s^2$ then takes place along the brake tentacle. The process then repeats until the vehicle has stopped. A formal proof that a collision can be avoided in all cases is very hard to achieve because of the complex interplay between geometric aspects, the process of drivability classification, the tentacle selection for braking, the modification of the crash distance and the final deviations of the tentacle's trajectories due to vehicle dynamics.

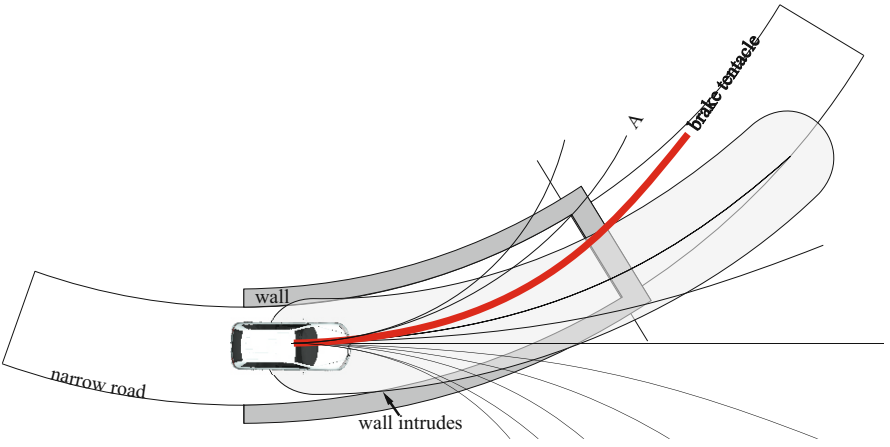


Fig. 25. The vehicle has moved for 0.1 seconds and we assume that a deviation from the tentacle's path occurred, such that the vehicle has an angular deviation from the desired trajectory. As a consequence the wall at the bottom intrudes into the classification area at a distance close to the vehicle. Hence, the tentacle will be classified as non-drivable. Evidently, all other tentacles will also be classified as non-drivable and the vehicle will start to brake for the next 0.1 seconds. For braking, it will choose the tentacle with the largest distance to the first obstacle. Note, that this is not the tentacle marked with A, because obstacles are mapped orthogonally onto the classification histogram of the tentacle (see page 404).

7 Experiments and Competitions

Our approach was tested and demonstrated in several scenarios experimenting with different parameter settings and execution modes.

7.1 System Integration at the C-Elrob 2007

At the C-Elrob 2007 we ran the vehicle in exploration mode. Here, both factors a_0 and a_2 of (16) were zero. That is the vehicle decided to drive always towards the flattest areas without regarding any given trajectory. Tentacle execution was done by temporal filtering of the selected tentacles and fragment execution as described above. Even this simple approach worked surprisingly well, and at the C-Elrob 2007 our CoTeSys[4] demonstrator MuCAR-3 drove a combined urban and non-urban course in record time[5]. Manual intervention was only necessary at some crossings to force the vehicle to take the correct course (since we did not exploit any GPS knowledge). In separate demonstrations we let the vehicle drive along narrow serpentines. It was able to drive the serpentines from the base camp at top of Monte Ceneri[6] down to main

[4] German Cluster for Exellence: Cognition For Technical Systems.
[5] `http://www.unibw.de/lrt13/tas/pressestimmen`
[6] Monte Ceneri, Swizerland.

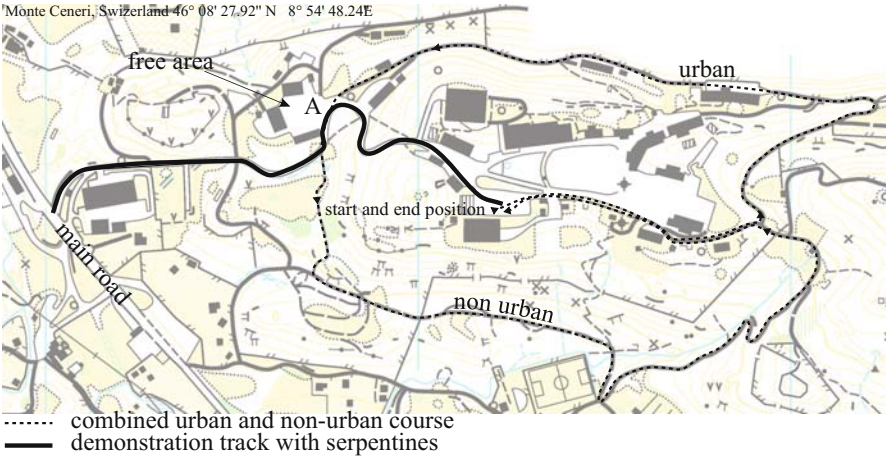


Fig. 26. The solid line shows the serpentines from the base camp at Monte Ceneri, Swizerland down to the main road our vehicle was able to drive fully autonomously without using any GPS information at the C-Elrob 2007. Occasionally, the vehicle entered the free area **A**. The dashed-line shows the combined urban and non-urban course our vehicle drove in record time (manual intervention at some crossings).

road without using any GPS information (see figure 26). Occasionally, the vehicle decided to enter the free area at point **A** in figure 26 driving around some parking trucks there, leaving the area again after some time or getting stuck in a dead end.

7.2 System Integration within the DARPA Urban Challenge 2007 Finalist AnnieWay

Due to the success of our approach at the C-Elrob 2007 we integrated our method into the system of DARPA Urban Challenge Team *AnnieWay* (Kammel, 2007). A description of the overall system architecture is given by (Kammel et al., 2008). In this setup tentacles were only used, if the areas in the occupancy grid that corresponded to the current relevant section of the GPS trajectory were unexpectedly not free of obstacles. To check this we computed a speed depending lookahead distance d_{check} and defined a triangular mesh structure along the GPS-section as illustrated in figure 28. We then projected each triangle into the ego-centered occupancy grid and scanned the cells of the triangles using a standard triangle scan-conversion method from computer graphics. The area was regarded as free if less than $n = 2$ grid cell exhibited a z-difference more than 0.1m. If the area was regarded as occupied, the tentacle approach was switched on. Note, that a separate obstacle tracker existed in the overall system. This obstacle tracker was run before the grid-checking mechanism, and the distance d_{check} was cropped by the distance to the first obstacle. Hence, the tentacles only reacted on obstacles that were unseen by the obstacle tracker.

Fig. 27. This figure shows the test area and road network we used in preparation for the Urban Challenge 07. In excessive tests, our approach was proven to be able to safely drive in this conventional residential area avoiding pavement edges, parking cars and other obstacles. No traffic was present and a security driver was onboard for safety reasons. In particular, our approach mastered the following two difficult cases: In (b) the GPS-trajectory was intentionally defined to lead straight through a traffic circle. Our approach was repeatedly shown to drive around the traffic circle, not following collision course of the GPS-trajectory directly. In (c) the GPS-trajectory was defined to shortcut a road crossing colliding with neighboring houses. However, using our approach the vehicle followed the road to the next crossing and turned right correctly, not strictly following the GPS-trajectory.

The obstacle tracker did detect and track obstacles only of the approximate size of a vehicle, and evaluated the occupancy grid for obstacles only at the expected area of the road as given by the GPS waypoints. Hence, in cases of GPS-offsets the obstacle tracker could miss obstacles. In these cases the tentacles provided a reactive protective shelter. However, even when the tentacles were switched on, they tried to follow the GPS-trajectory if possible (because of the trajectory value). For the Urban Challenge setup we run the tentacle approach with the parameters of equation (16) being $a_0 = 1$,$a_1 = 0$ and $a_2 = 0.5$ such that the vehicle had a preference to follow the given GPS-trajectory but avoided obstacles at the same time. The choice $a_0 = 0$ means that flatness was actually ignored for the Urban Challenge. The rationale for the Urban Challenge was that as long as a tentacle is drivable, we would prefer the one that follows the GPS-path instead of preferring flat tentacles in the extreme. In general, the driving speed was determined by the behaviors of the overall system. However, the tentacle method was allowed to undercut this speed. During the UC07 the speed was cut down to $2.0m/s$ when tentacles became active (if no lower speed was initially selected). The reason for this low speed was that if the

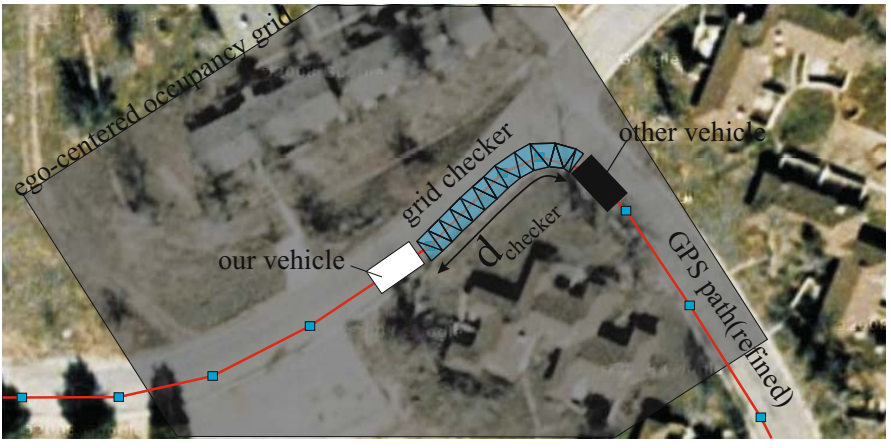


Fig. 28. The grid checker was a triangular mesh that was defined along the current section of the RNDF in front of the vehicle. The triangles were projected into the ego-centered occupancy grid to see whether the respective area was free of obstacles. Tentacles were switched on, only if the area was not clear. To avoid the tentacles reacting on other cars that were previously seen by the separate obstacle tracker([Kammel et al., 2008]), the speed depending length $d_{checker}$ was cropped by the distance to the first obstacle.

tentacles became active, one could conclude that something unpredicted had happened, e.g due to a GPS-drift or an obstacle not detected by the obstacle detection/tracking module.

7.3 Experiments in Preparation for the Urban Challenge

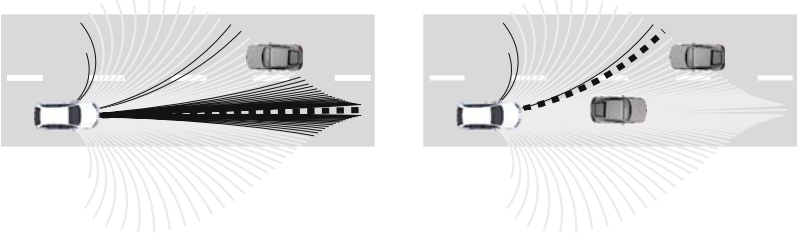
In preparation for the Urban Challenge our approach was tested excessively at both a parking lot with manually placed obstacles and vehicles and a residential area (under safe conditions, security driver). The most difficult tests were performed in the residential area using a road network definition file as shown in figure 27a): Here, the GPS trajectory was intentionally defined directly through a traffic circle, such that if the GPS trajectory was executed blindly, the vehicle would collide with the circular area in the center (see figure 27b). In dozens of repetitions our approach was confirmed to safely drive around the traffic circle not strictly following the GPS-trajectory. In a similar experiment, we intentionally misplaced the GPS trajectory at a straight road by an offset of about 10 meters, such that the trajectory passed through the front gardens of the neighboring houses. Despite of this severe offset, our approach lets the vehicle drive along the road, avoiding pavement edges, parking cars, trees and other obstacles. Occasionally the vehicle tried entering wide doorways at the side corresponding to the GPS-offset but stopped safely at dead ends. In the scenario shown in figure 27c) we intentionally misdefined the GPS-trajectory short cutting the road crossing by about 50

meters through nearby buildings. Using our approach the vehicle did not follow the collision course, but continued driving along the straight road, taking the next road correctly to the right.

Care must be taken when dealing with moving objects. Oncoming traffic as shown in figure 29a) is handled well by our algorithm and we did not experience a single dangerous behavior in all our experiments. However the case in 29b) is dangerous (for a human driver, too), since the vehicle would try overtaking the car on the right lane, although another car is approaching on the left lane. Hence, our approach should not be used in a stand-alone manner, in case of traffic scenarios. Instead a separate tracker for moving objects should be run in parallel (as we did for the Urban Challenge 2007). One option for combining our method with a separate obstacle tracker is to predict future obstacle positions and explicitly draw them into the grid, blocking the corresponding tentacles. Another option (we applied at the Urban Challenge 2007) is to evaluate tentacles just up to the first tracked object, such that a car driving ahead is not sensed by the tentacles. Speed control then has to be handled in a way that the robot follows the car. In this way, our approach can be restricted to avoid only those obstacles that were not detected by the object tracker.

7.4 Performance at the Urban Challenge

At the Urban Challenge 2007 our method was active in all preliminary tests, test A, test B and test C and also at the final race. Unfortunately, tentacle logging was off, and no quantitative data is available of how often or when the



(a) (b)

Fig. 29. The situation shown in (a) is handled well by our method. Some tentacles are blocked by the oncoming traffic (light gray) and the pavement edges of the road leaving three groups of tentacles (black). It would be dangerous to select a tentacle leading to the left lane. However, one of the straight tentacles is selected (dashed black) because of the hysteresis step (see page 412) - preferring tentacles that are more similar to the last selected tentacle in ambiguous situations. The situation shown in (b) is dangerous, since the vehicle would try to overtake the car in front, notwithstanding the oncoming traffic. Hence, our approach should be combined with a separate object tracker in traffic environments (e.g. predicting future obstacle positions and explicitly drawing them into the grid).

Fig. 30. At the Urban Challenge 2007 our approach was important at narrow passages where small GPS offsets could cause a collision.

tentacles where active. The only certainty about them having been active was the slow speed of the AnnieWay vehicle observable when driving along narrow passages. This can also be seen in videos we recorded. A typical scene where tentacles were active is depicted in figure 30 showing narrow passages where even small GPS offsets could be dangerous. When tentacles where active the velocity was cropped to at most $2m/s$. In one test, our car slightly hit a wall. The reason was an error in the tentacle code but evidently a software bug. With this bug (that happend by switching from debug to release mode) the vehicle had computed NaN-floats as curvatures and transmitted them to the low-level controller. In the final race, the AnnieWay car stopped at the entry of a sparse zone. This was also due to a software bug, known to the team and not related to the tentacle method. Overall, we believe that our stop in the final was not because of a conceptual problem, but because of insufficient time for testing and bug fixing.

7.5 Lessons Learned

The major lesson learned at the DARPA Urban Challenge is the inevitability to handle problems arising due to GPS inaccuracies. Many teams participating at the challenge reported that one of the key enabling techniques was to constantly keep track of the offset between GPS measurements and the DARPA provided GPS waypoints. In this spirit, the approach presented in this paper allows to follow a given GPS trajectory while avoiding obstacles at the same time. Using feature based references for estimating the GPS offset is particularly hard to accomplish as free space areas are often lacking such features lane markings. For example, the path planner that was active at zones had to cope with the problem that when the GPS destination point was sometimes believed to be occluded by an obstacle due to GPS drifts.

We belief that this lesson points towards a new direction of how plans for navigation should be represented. To open this perspective, we briefly consider the merits of reactive schemes, such as the one this paper describes, and path planning schemes. The most obvious difference is the temporal behavior of the both: while reactive schemes are fast and react immediately, trajectory planning methods are comparatively slow. Often, replanning is done at a rate too low to guarantee that no dynamic obstacle has moved into the planned track. On the other hand, path planning allows to anticipate and avoid traps a reactive scheme might just run into. However, the fundamental difference we would like to emphasize is the way both methods relate to the world as perceived. In contrast to path planning methods, where planning takes place in abstract models of the world (e.g. SLAM, and cartesian map approaches in general), reactive schemes directly couple perception with action. Consider, for instance, a reactive navigation approach driving along a road, guided by the principle to drive on flat ground. Then, the road essentially corresponds to what a planned path is in a path planning method.

We do not argue for reactive approaches in general, but for their inherent property of direct reference to the world. Thus, the critical question is whether this property can somehow be transferred to planning methods. We believe that a change of the planning domain from cartesian coordinate systems fixed to the ground to what one might call perceptual references is a promising direction. With perceptual references, plans would be described similar to how humans communicate plans: "Overtake the obstacle to the left, pass the next one at the right side". We believe that successful future navigation systems will follow this direction, instead of doing what most researchers do now: implementing "GPS-trains" following "GPS trails".

8 Conclusions and Future Work

In this paper we proposed a simple, very efficient and fast to implement reactive navigation method intended for the use with multi-beam LIDARs. The approach can be used for two purposes: It allows an autonomous robot to safely navigate in previously unknown static terrain. As a second option it can be used as protective shelter for systems that are based on following GPS-waypoints, such that the method follows the GPS trajectory in case it is safe, but avoids obstacles aside the track in case of GPS drift. The method is not intended to deal with moving obstacles, but the paper shows how the method can be combined with a separate dynamic obstacle tracker. Both options have been demonstrated at robotic competitions, the exploration of unknown terrain at the C-Elrob 2007, where our vehicle drove a difficult combined urban and non-urban course 90 percent autonomously without using any GPS (manual intervention at crossings to make the vehicle drive the correct course), and the protective shelter option was implemented within the system of DARPA Urban Challenge finalist Team AnnieWay. At the core of the system is a set of motion primitives (called tentacles) that are used for both perception and

motion execution. In contrast to other methods, our algorithm uses an ego-centered occupancy grid. In this way the geometric relation between the tentacles and the grid is static, allowing to pre-compute grid addresses. This static relationship is the main reason for the efficiency of the method. However, the efficiency comes at the cost of violating vehicle dynamics in the sense that the geometric shape of the tentacles cannot be precisely executed. However, the interesting aspect is - and we provide a theoretic analysis of this aspect - that instead of this violation the algorithm can ensure that the deviation of the resulting path to the tentacle is bound to be in an area that is ensured to be free of obstacles. The second reason for the efficiency of the method is that it only evaluates a small set ($n = 81$) of tentacles at a time, all having the shape of circular arcs. Instead of this seeming geometric restriction the method achieves a huge space of possible trajectories by using a speed-varying evaluation mechanism of the tentacles and chaining only small fragments of the circular arcs. The reactive method does not accumulate data and is completely data driven thereby avoiding the SLAM problem. No global position of the vehicle or obstacles has to be estimated.

Several interesting extensions of our method are possible. The first improvement would be to consider a set of tentacles for a large number of sampled points in the state space of the vehicle dynamics. In this way, the dynamically correct shapes could be picked up in every iteration. This improvement would only be at the cost of memory, not of computational load. Further, it would be interesting to learn the tentacles of each state by observing and sampling the states and trajectories recorded while a human drives. Another interesting idea arises by observing that the classification procedure of the tentacles not only determines whether a tentacle is drivable or not, but also determines the distance at which the first obstacle occurs. When driving along a conventional road with pavement edges, one can observe that the set of tentacles finds these edges well. The idea is to see the tentacles as a curb-detector and try to group those obstacle positions, e.g. by verifying whether they constitute a smooth sequence of positions, in this case detecting the border of the road. Then, navigation could be done relative to the detected and tracked road boundary. In case no detection is possible, the method could fall back on the pure reactive navigation mode. In this way, reactive and cognitive navigation in relation to perceived road boundaries could nicely be combined.

Acknowledgements

This research was supported in part by the German Excellence Cluster CoTeSys “Cognition For Technical Systems”

References

Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14(2), 239–256 (1992)

Braitenberg, V.: Vehicles: Experiments in Synthetic Psychology. MIT Press, Cambridge (1984)

Coombs, Murphy, Lacaze, Legowik: Driving Autonomously Offroad up to 35 km/h. In: Procs. IEEE Intelligent Vehicles Symposium 2000, Detroit, USA, pp. 186–191 (2000)

Coulter, R.C.: Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (1992)

Dickmanns, E.D.: The 4d-approach to dynamic machine vision. In: Proceedings of the 33rd IEEE Conference on Decision and Control, vol. 4, pp. 3770–3775 (1994)

Dickmanns, E.D.: Dynamic Vision for Perception and Control of Motion. Springer, Heidelberg (2007)

Dissanayake, M., Newman, P., Clark, S., Durrant-Whyte, H., Csorba, M.: A solution to the simultaneous localization and map building (slam) problem. Robotics and Automation, IEEE Transactions on 17(3), 229–241 (2001)

Goldberg, S., Maimone, M., Matthies, L.: Stereo vision and rover navigation software for planetary exploration. In: Proceedings of the IEEE Aerospace Conference, vol. 5, p. 2025 (2002)

Julier, S., Uhlmann, J.: A counter example to the theory of simultaneous localization and map building (2001)

Kammel, S.: DARPA Urban Challenge, Team AnnieWay, team homepage (2007), `http://annieway.mrt.uni-karlsruhe.de`

Kammel, S., Ziegler, J., Pitzer, B., Werling, M., Gindele, T., Jagzent, D., Schröder, J., Thuy, M., Goebl, M., von Hundelshausen, F., Pink, O., Frese, C., Stiller, C.: Team annieway's autonomous system for the darpa urban challenge 2007. International Journal of Field Robotics Research (2008)

Kelly, A., Stentz, A.T.: Rough terrain autonomous mobility - part 2: An active vision, predictive control approach. Autonomous Robots 5, 163–198 (1998)

Lamon, P., Kolski, S., Triebel, R., Siegwart, R., and Burgard, W.: The smartter for elrob 2006 a vehicle for fully autonomous navigation and mapping in outdoor environments. Technical report, Autonomous Systems Lab, Ecole polytechnique Fdrale de Lausanne, Switzerland, Autonomous Intelligent Systems, Albert-Ludwigs-University of Freiburg, Germany (2006)

Landau, Y.D.: Adaptive Control: The Model Reference Approach. Marcel Dekker, Inc., New York (1979)

Mitschke, M., Wallentowitz, H.: Dynamik der Kraftfahrzeuge. Springer, Heidelberg (2004)

Nilsson, N.J.: Shakey the robot. Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025 (1984)

Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Monographs in Computer Science. Springer, Heidelberg (1985)

Rosenblatt, J.: Damn: A distributed architecture for mobile navigation (1995)

Sariff, N., Buniyamin, N.: An overview of autonomous mobile robot path planning algorithms. In: 4th Student Conference on Research and Development, 2006. SCOReD 2006, pp. 183–188 (2006)

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., Niekerk, J.v., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley: The robot that won the darpa grand challenge: Research articles. J. Robot. Syst. 23(9), 661–692 (2006)

Caroline: An Autonomously Driving Vehicle for Urban Environments

Fred W. Rauskolb[1], Kai Berger[2], Christian Lipski[2], Marcus Magnor[2], Karsten Cornelsen[3], Jan Effertz[3], Thomas Form[3], Fabian Graefe[3], Sebastian Ohl[3], Walter Schumacher[3], Jörn-Marten Wille[3], Peter Hecker[4], Tobias Nothdurft[4], Michael Doering[5], Kai Homeier[5], Johannes Morgenroth[5], Lars Wolf[5], Christian Basarke[6], Christian Berger[6], Tim Gülke[6], Felix Klose[6], and Bernhard Rumpe[6]

[1] Herzfeld & Rubin, P.C.
40 Wall Street
New York, NY 10005
[2] Institute of Computer Graphics
Mühlenpfordtstraße 23
38106 Braunschweig, Germany
[3] Institute of Control Engineering
Hans-Sommer-Straße 66
38106 Braunschweig, Germany
[4] Institute of Flight Guidance
Hermann-Blenk-Straße 27
38108 Braunschweig, Germany
[5] Institute of Operating Systems and Computer Networks
Mühlenpfordtstraße 23
38106 Braunschweig, Germany
[6] Institute of Software Systems Engineering
Mühlenpfordtstraße 23
38106 Braunschweig, Germany
carolo-uc@tu-bs.de

Abstract. The 2007 DARPA Urban Challenge afforded the golden opportunity for the Technische Universität Braunschweig to demonstrate its abilities to develop an autonomously driving vehicle to compete with the world's best competitors. After several stages of qualification, our team CarOLO qualified early for the DARPA Urban Challenge Final Event and was among only eleven teams from initially 89 competitors to compete in the final. We had the ability to work together in a large group of experts, each contributing his expertise in his discipline, and significant organisational, financial and technical support by local sponsors who helped us to become the best non-US team.

In this report, we describe the 2007 DARPA Urban Challenge, our contribution "Caroline", the technology and algorithms along with her performance in the DARPA Urban Challenge Final Event on November 3, 2007.

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 441–508.
springerlink.com

1 Motivation and Introduction

Focused research is often centered around interesting challenges and awards. The airplane industry started off with awards for the first flight over the British Channel as well as the Atlantic Ocean. The Human Genome Project, the RoboCups and the series of DARPA Grand Challenges for autonomous vehicles serve this very same purpose to foster research and development in a particular direction. The 2007 DARPA Urban Challenge is taking place to boost development of unmanned vehicles for urban areas. Although there is an obvious direct benefit for DARPA and the U.S. government, there will also be a large number of spin-offs in technologies, tools and engineering techniques, both for autonomous vehicles, but also for intelligent driver assistance. An intelligent driver assistance function needs to be able to understand the surroundings of the car, evaluate potential risks and help the driver to behave correctly, safely and, in case it is desired, also efficiently. These topics do not only affect ordinary cars, but also buses, trucks, convoys, taxis, special-purpose vehicles in factories, airports and more. It will take a number of years before we will have a mass market for cars that actively and safely protect the passenger and the surroundings, like pedestrians, from accidents in any situation.

Intelligent functions in vehicles are obviously complex systems. Large issues in this project where primarily the methods, techniques and tools for the development of such a highly critical, reliable and complex system. Adapting and combining methods from different engineering disciplines were an important prerequisite for our success. For a stringent deadline-oriented development of such a system it is necessary to rely on a clear structure of the project, a dedicated development process and an efficient engineering that fits the project's needs. Thus, we did not only concentrate on the single software modules of our autonomously driving vehicle named Caroline, but also on the process itself. We furthermore needed an appropriate tool suite that allowed us to run the development and in particular the testing process as efficient as possible. This includes a simulator allowing us to simulate traffic situations and therefore achieve a sufficient coverage of test situations that would have been hardly to conduct in reality. Only a good collaboration between the participating disciplines allowed us to develop Caroline in time to achieve such a good result in the 2007 DARPA Urban Challenge.

In the long term, our goal was not only to participate in a competition but also to establish a sound basis for further research on how to enhance vehicle safety by implementing new technologies to provide vehicle users with reliable and robust driver assistance systems, e.g. by giving special attention on technology for sensor data fusion and robust and reliable system architectures including new methods for simulation and testing. Therefore, the 2007 DARPA Urban Challenge provided a golden opportunity to combine several expertise from several fields of science and engineering. For this purpose, the interdisciplinary team CarOLO had been founded, which drew its members

from five different institutes. In addition, the team received support from a consortium of national and international companies.

In this paper, we firstly introduce the 2007 DARPA Urban Challenge and derive the basic requirements for the car from its rules in section 2. Section 3 describes the overall architecture of the system, which is detailed in section 4 describing sensor fusion, vision, artificial intelligence, vehicle control and along with safety concepts. Section 5 describes the overall development process, discusses quality assurance and the simulator used to achieve sufficient testing coverage in detail. Section 6 finally describes the evaluation of Caroline, namely the performance during the National Qualification Event and the DARPA Urban Challenge Final Event in Victorville, California, the results we found and the conclusions to draw from our performance.

2 2007 DARPA Urban Challenge

The 2007 DARPA Urban Challenge is the continuation of the well-known Grand Challenge events of 2004 and 2005, which were entitled "Barstow to Primm" and "Desert Classic". To continue the tradition of having names reflect the actual task, DARPA named the 2007 event "Urban Challenge", announcing with it the nature of the mission to be accomplished.

The 2004 course, as shown in Fig. 1, led from the Barstow, California (A) to Primm, Nevada (B) and had a total length of about 142 miles. Prior to the main event, DARPA held a qualification, inspection and demonstration for each robot. Nevertheless, none of the original fifteen vehicles managed to come even close to the goal of successfully completing the course. With 7.4 miles as the farthest distance travelled, the challenge ended very disappointingly and no one won the $1 million cash prize.

Thereafter, the DARPA program managers heightened the barriers for entering the 2005 challenge significantly. They also modified the entire quality inspection process to one involving a step-by-step application process, including a video of the car in action and the holding of so-called Site Visits, which involved the visit of DARPA officials to team-chosen test sites. The rules for these Site Visits were very strict, e.g. determining exactly how the courses had to be equipped and what obstacles had to be available. From initially 195 teams, 118 were selected for site visits and 43 had finally made it into the National Qualification Event at the California Speedway in Ontario, California. The NQE consisted of several tasks to be completed and obstacles to overcome autonomously by the participating vehicles, including tank traps, a tunnel, speed bumps, stationary cars to pass and many more.

On October 5, 2005, DARPA announced the 23 teams that would participate in the final event. The course started in Primm, Nevada, where the 2004 challenge should have ended. With a total distance of 131.6 miles and several natural obstacles, the course was by no means easier than the one from the year before. At the end, five teams completed it and the rest did significantly

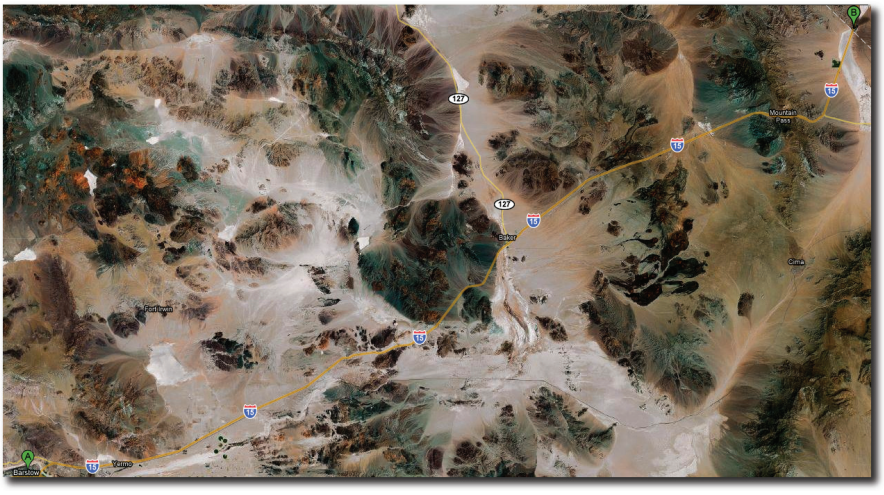


Fig. 1. 2004 DARPA Grand Challenge Area between Barstow, CA (A) and Primm, NV (B).

better as the teams the year before. The Stanford Racing Team was awarded the $2 million first prize.

In 2007, DARPA wanted to increase the difficulty of the requirements, in order to meet the goal set by Congress and the Department of Defense that by 2015 a third of the Army's ground combat vehicles would operate unmanned. Having already proved the feasibility of crossing a desert and overcome natural obstacles without human intervention, now a tougher task had to be mastered. As the United States Armed Forces are currently facing serious challenges in urban environments, the choice of such seemed logical. DARPA used the good experience and knowledge gained from the first and second Grand Challenge event to define the tasks for the autonomous vehicles. The 2007 DARPA Urban Challenge took place in Vicorville, CA as depicted in Fig. 2.

The Technische Universität Braunschweig started in June 2006 as a newcomer in the 2007 DARPA Urban Challenge. Significantly supported by industrial partners, five institutes from the faculties of computer science and mechanical and electrical engineering equipped a 2006 Volkswagen Passat station wagon named "Caroline" to participate in the DARPA Urban Challenge as a "Track B" competitor.

Track B competitors did not receive any financial support from the DARPA compared to "Track A" competitors. Track A teams had to submit technical proposals to get technology development funding awards up to $1,000,000 in fall 2006. Track B teams had to provide a 5 minutes video demonstrating the vehicles capabilities in April 2007. Using these videos, DARPA selected 53 teams of the initial 89 teams that advanced to the next stage in the

Fig. 2. 2007 DARPA Grand Challenge Area in Victorville, CA.

qualification process, the "Site Visit" as already conducted in the 2005 Grand Challenge.

Team CarOLO got an invitation for a Site Visit that had to take place in the United States. Therefore, team CarOLO accepted gratefully an offer from the Southwest Research Insitute in San Antonio, Texas providing a location for the Site Visit. On June 20, Caroline proved that she was ready for the National Qualification Event in fall 2007. Against great odds, she showed her abilities to the DARPA officials when a huge thunderstorm hit San Antonio during the Site Visit. The tasks to complete included the correct handling of intersection precedence, passing of vehicles, lane keeping and general safe behaviour. After the demonstration, the team returned to Germany together with Caroline.

On August 9, the team received the results of the Site Visit event together with an inivitation to the next stage of the qualification process: The National Qualification Event in Victorville, California. Being a semi-finalist team, the team returned at the end of September to the Southwest Research Institute in San Antonio to finalize the development and tests. Three weeks later, Caroline and the team arrived in Victorville, California and participated in the National Qualification Event. To qualify for the Final Event, three courses had to be mastered by the vehicles, each one covering a certain part of the requirements. At the first course, called "Track A", the robots needed to merge into moving traffic, "Track B" required the handling of very long and complex routes with stationary obstacles and "Track C" tested intersections and how the vehicles handle the blockage of roads. Demonstrating repeatedly

the performance of Caroline in all tracks of the National Qualification Event, Caroline qualified early for the final stage, the DARPA Urban Challenge Final Event held on November 3. In chapter 6, the overall performance of Caroline in the National Qualification Event and the DARPA Urban Challenge Final Event is illustrated.

3 System Architecture

Caroline is a standard 2006 Volkswagen Passat station wagon equipped with a variety of sensors, actuators and computers to function as an autonomous mobile robot. In front, two multi-level laser scanners, one multi-beam lidar sensor and one radar sensor cover a field of view up to 200 meters for approaching traffic or stationary obstacles. In addition, four cameras detect and track lane markings in order to allow precise lane keeping. The stereo vision system behind the windshield and another color camera combined with two laser scanners mounted on the roof were installed to provide information about the drivability of the terrain in front of the vehicle. Very similar to the front of the vehicle, one multi-level laser scanner, one medium range radar, one lidar and two radar-based blind-spot-detectors enable Caroline to detect obstacles at the rear. All these sensors are depicted in Fig. 3.

An array of automotive PCs mounted on a rack shown in Fig. 4 functions as the hardware platform for a distributed software architecture with all internal communication based upon Ethernet. The access to Caroline's by-wire steering, brake and throttle system as well as to other low level actuators

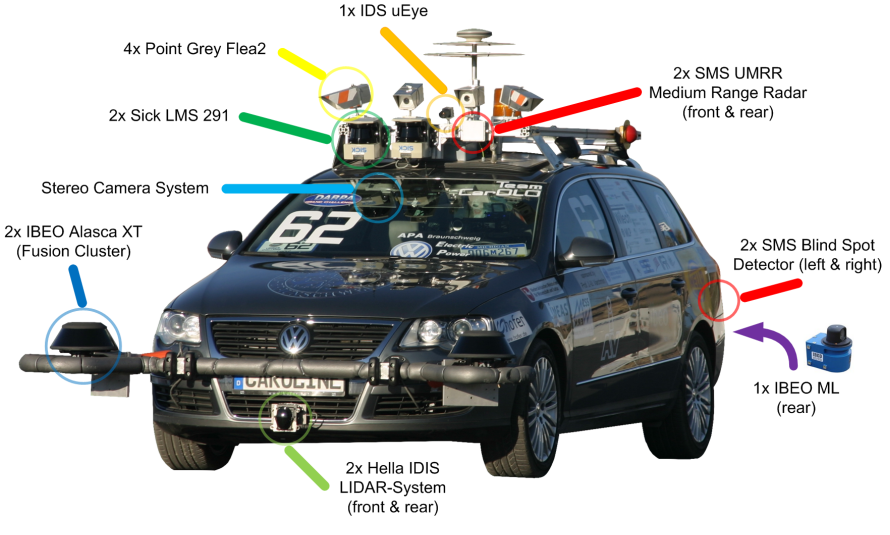


Fig. 3. The perception system.

is provided through a CANLOG III command interface, which also connects to the vehicle's E-stop system to provide emergency stop functionality even if the complete software system described below should fail. Regardless to those lower level components described above, all computing and control hardware is based on industrial PC technology, thereby reducing hardware variety, simplifying failure management along with component replacement.

The development of Caroline is divided among a number of institutes and disciplines, including faculties for computer science and mechanical and electrical engineering. Mirroring this internal structure, Caroline's architecture is grouped into eight principal modules, interconnected with predefined interfaces as shown in Fig. 5: Sensor Data Acquisition, Sensor Data Fusion, Image Processing, Digital Map, Artificial Intelligence, Vehicle Path Planning and Low Level Control, Supervisory Watchdog and Online-Diagnosis, Telemetry and Data Storage for Offline Analysis. Due to the intentionally linear signal flow between each function module without major signal loops, we are able to develop different modules independently and with minimum interference.

Starting at the bottom of this linear flow, the data acquisition unit provides necessary hard- and software modules to collect and process incoming data from all active sensors for object recognition. Since all of the sensors used are standard components originating from contemporary automotive driver assistance systems, they are equipped with a Controller Area Network (CAN) communication interface. Taking into account the limitation of this bus standard regarding data throughput and determinism, a private sensor CAN was chosen for each sensor to keep latencies small and to avoid bus conflicts.

The acquisition of GPS and INS data (referred as Ego State in the following) was moved directly into the real-time vehicle control unit in order to avoid large latencies within the closed loop dynamic control. The time of

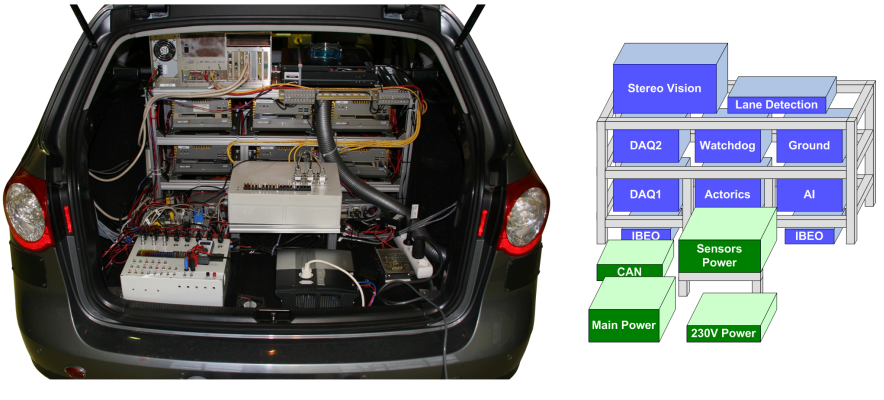


Fig. 4. Computer rack and power supply.

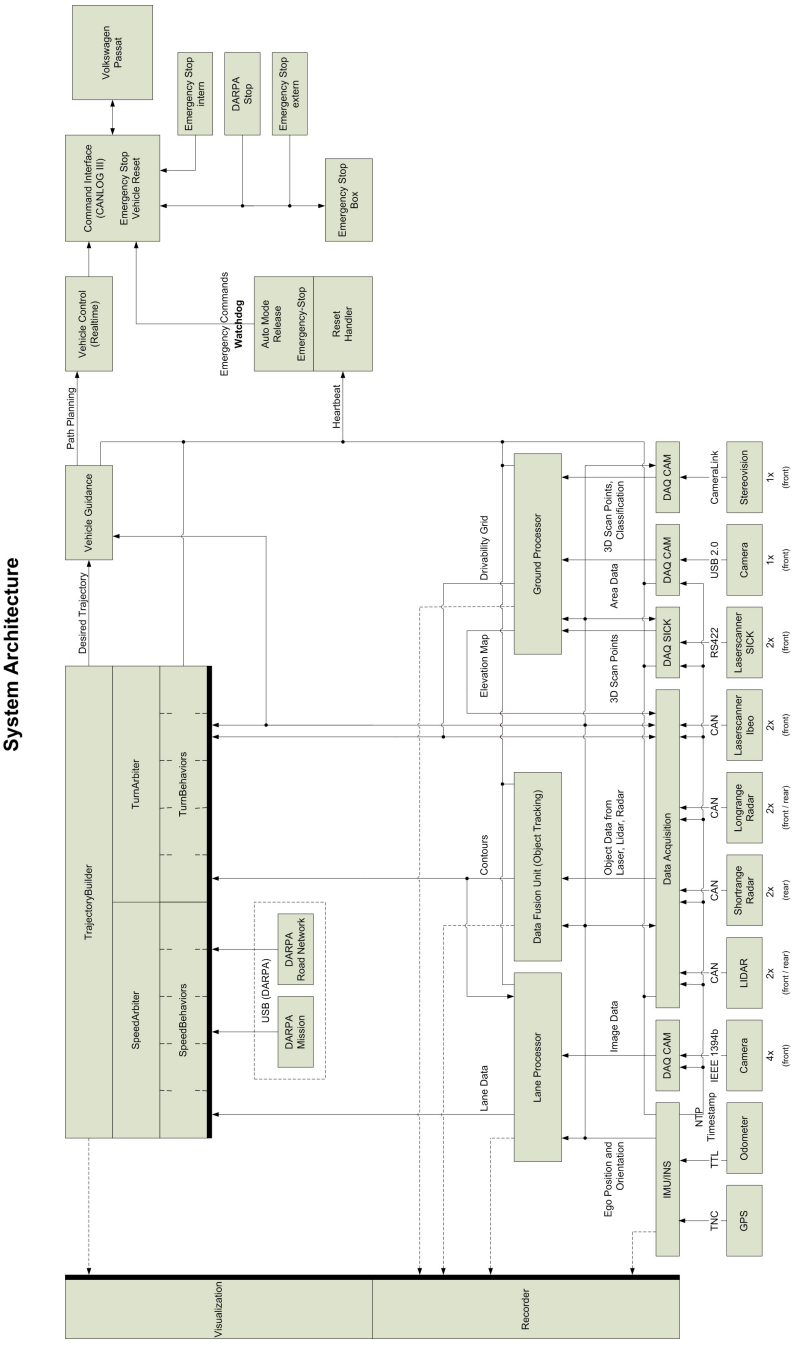


Fig. 5. System architecture.

day is obtained from the GPS and distributed via the network time protocol (NTP) to all subsystems.

Incoming video data is sampled from the assigned IEEE 1394 interface, preprocessed and interpreted directly on the image acquisition PCs to avoid overload of the vehicle's internal network by image data. Lane detection data is directly passed to the artificial intelligence. The stereo vision system delivers 3D scan points along with area data describing the drivability of the road. This data is fused with further scan points obtained from the laser scanners and area data from the additional color camera observing the ground in front of the vehicle. This fusion results in a drivability grid which is sent to the artificial intelligence module.

Furthermore, following Caroline's signal flow, sensor data of all object-recognition sensors is processed within a central sensor data fusion unit as described in section 4.1, which transmits the object-based vehicle's surroundings containing all static and dynamic targets in Carolines field of view to the digital map. The digital map combines online environmental information with available offline information generated from mission definition files (MDF) and route network definition files (RNDF) provided for the mission. This combined data is the basis for the artificial intelligence to generate driving decisions based on a Distributed Architecture for Mobile Navigation scheme (DAMN) as proposed by [Rosenblatt, 1997] and described in section 4.3.

The driving commands obtained, e.g. "follow a given road" are issued to the soft real time control module, which carries out trajectory generation and optimization based on driving dynamics of the vehicle. The driving trajectories generated are then passed along into hard real time control that addresses the vehicle actuators.

All modules previously described are supervised by a central watchdog process with the possibility to kill and restart one or several processes, computers or sensors independently. Thus, a maximum of self-healing capability is installed in Caroline's systems.

The visualization module is used during development in order to display all exchanged object data. This data consists of e.g. obstacles, lanes, terrain drivability, the planned path and mission data files. A recorder and a player module which logs data for the purpose of offline-analysis, are also integrated in this module.

4 System Modules

Caroline's software system consists of five modules. Tasks to be mastered in order to compete in the 2007 DARPA Urban Challenge are environment recognition, road finding, situation assessment and vehicle control supervised by a safety module. These core modules are described below.

4.1 Sensor Fusion

Perception is one of Caroline's key systems. The system detects obstacles as well as the drivability of the environment. The sensor fusion system is separated in two parts. The first one is responsible for obstables, such as other cars, walls or pedestrians. The other one takes care of the drivability of the environment. Thus, it is possible to keep the car on the road even in rough evironments. Based on this information, the artificial inteligence is able to find a safe path through traffic. The perception system will be described in greater detail in the following sections. The following section introduces the sensor concept, followed by the object-based data fusion and end with the grid based fusion of the drivability.

4.1.1 Sensor Concept

A variety of sensor types originating from the field of driver assistance systems were chosen to provide detection of static and dynamic obstacles in the vehicle's surroundings as depicted in Fig. 3:

- Dark green: A stationary beam LIDAR sensor placed in the front and rear of the vehicle, have a range of approximately 200 meters with an opening angle of 12 degrees. The unit has an internal preprocessing stage and thus delivers its readings in an object oriented fashion, providing target distance, target width and relative target velocity with respect to the car's fixed sensor coordinate frame.
- Red: 24 GHz radar sensors were added to the front, rear, rear left and right side of the vehicle. While the center front and rear sensors provide a detection range of approximately 150 meters with an opening angle of 40 degrees, the rear right and left sensors function as blind-spot detectors with a range of 15 meters and an opening angle of 140 degrees due to their specific antenna structure. The front sensor acts as a stand-alone unit delivering object-oriented target data, such as position and velocity through its assigned external control unit (ECU). The three radar sensors in the rear section operate as a combined sensor cluster using an additional ECU, providing object-oriented target data in the same fashion as the front system. From the perspective of the post processing fusion system, the three rear sensors can therefore be regarded as one unit.
- Blue: Two Ibeo ALASCA XT laser scanners were installed in the vehicle's front section, each providing an opening angle of 240 degrees with a detection range of approximately 60 meters. The raw measurement data of both front laser scanners is preprocessed and fused on their corresponding ECU, delivering complex object-oriented target descriptions consisting of target contour information, target velocity and additional classification information. Additionally, the raw scan data of both laser scanners can be read by the fusion system's grid-based subsection.

- Purple: One Ibeo ML laser scanner was added to the rear side, providing similar detection capabilities as the two front sensors, with a reduced opening angle of 180 degrees due to its mounting position. All Ibeo sensors are based on a four-plane scanning principle with a vertical opening angle of 3.2 degrees between the top and bottom scan plane. This opening angle enables smaller pitch movements of the vehicle to be covered.
- Green: Two SICK LMS-291 laser scanners were mounted on the vehicle's front roof section. These scanners are based on a single-plane technology. They were set to measure the terrain profile at 10 and 20 meters, respectively. The view angle was limited to 120 degrees by software.
- Light blue: A stereo vision system mounted behind the vehicle's front window covers an area of approximately 60 degrees within a range of 50 meters, providing 3-dimensional terrain profile data for all stereo vision points retrieved. A simple classification into the driveway, curb and obstacles classes is also available.
- Orange: A USB-based color mono camera installed on the front roof section, covering an opening angle of approximately 60 degrees.

The sensors view areas are shown in Fig. 6. These view areas overlap for a more reliable view of the environment.

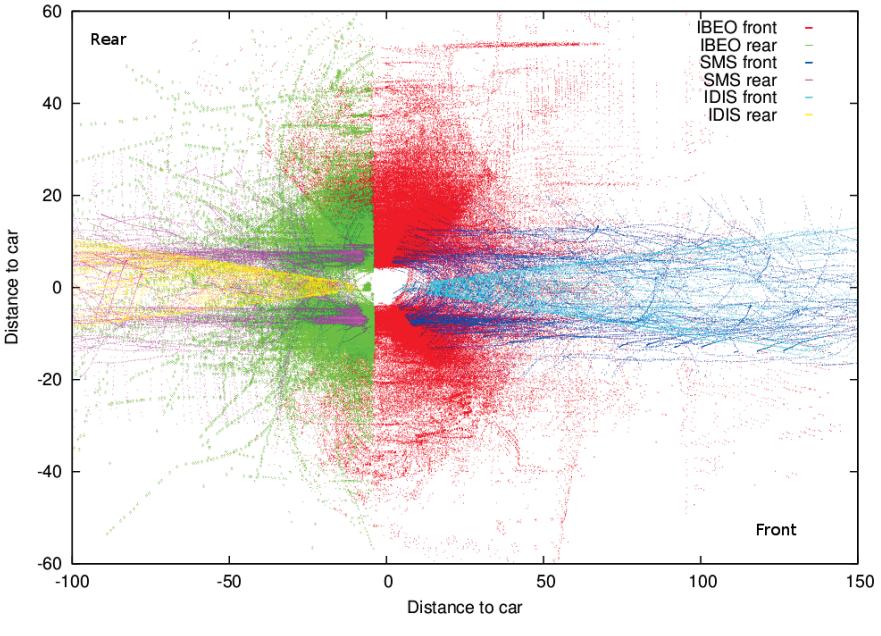


Fig. 6. Sensor view areas.

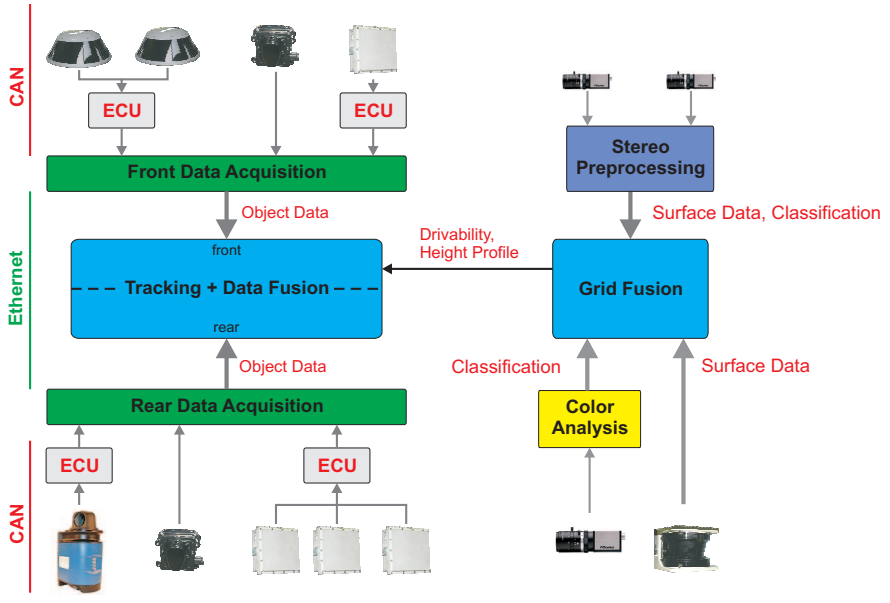


Fig. 7. Fusion architecture.

The sensor architecture described reflects the hybrid post-processing scheme applied. While the first four sensors deliver their data in an object-oriented fashion and are therefore treated within the system's object tracking and data fusion stage, the three last sensors described are evaluated based on their raw measurement data in the grid-based subsection. A distributed data fusion system consisting of three interconnected units was set up. In order to equally balance the available computing power, the object tracking system was split into two independent modules, covering the front and rear sections independently. Therefore, two automotive computers carry out data acquisition and data fusion of the front and rear object detecting sensors, while the third PC is used to fuse the raw sensor readings of the SICK scanners, stereo vision system and mono color camera as shown in Fig. 7.

4.1.2 Object Tracking Fusion

The object fusion system is based on a pipes and filters pattern as depicted in Fig. 8. All incoming sensor data is queued and then processed sequentially using a first in - first out strategy. Within the first step, data association is carried out in order to assign incoming sensor objects to their corresponding tracks in the fusion system that are taken from a real-time track database. In case of a positive match between an existing track and incoming sensor object, this pairing is then pushed into the processing queue of the system's Extended Kalman Filter in order to correct the track with new measurement data. If no match can be found, the sensor object is regarded as a potential

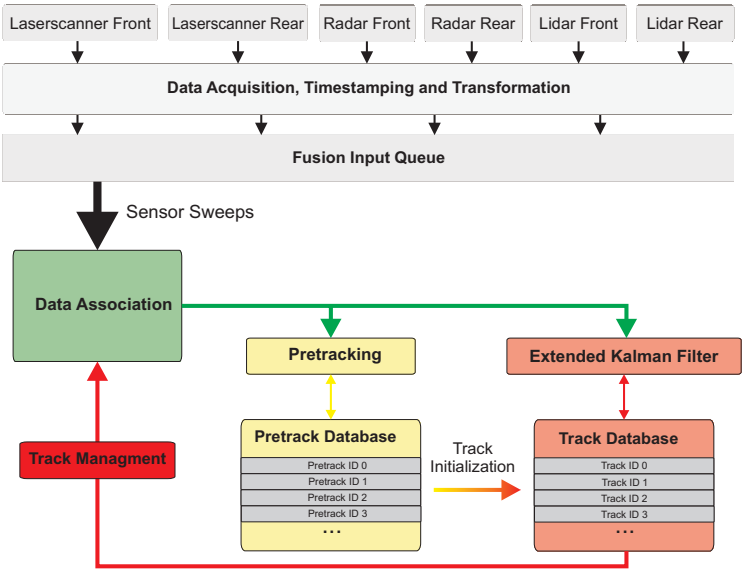


Fig. 8. Object fusion system architecture.

new target and pushed into the pretracking system. Within pretracking, sensor data is justified against time and all other sensors taking into account sensor redundancy where applicable. Pretracking and data association will be described later in greater detail.

If a sensor object has reached a certain level of justification, a new track will be instantiated and pushed into the real-time track database. Parallel to data association, pretracking and final object tracking, a track management unit periodically scans the track database for "dead" tracks - i.e. fusion objects that have not been updated for a certain amount of time. In addition to this garbage collection, all valid tracks are compared to each other for track merging and track splitting, which is necessary to handle situations including a passenger entering or leaving his vehicle or any other situation where two objects in the real world converge or split. Instead of transferring a whole track database image to downstream modules, create, update and delete messages of the track database are issued via the network. Every client is then capable of maintaining it's own track database. Therefore, network load can be significantly reduced without any loss of information.

Data Association and Pretracking. Data association and pretracking have a key functionality within Caroline's fusion system. Imperfect data association leads inevitably to incorrect tracks, whereas incorrect track initialization during pretracking leads to imperfect data association, since correct tracks and false alarms will then compete for incoming measurement data.

With this central position, the association and pretracking stage dominates the state estimator in the main tracking stage, since no state estimator can transform falsely associated sensor readings into useful update information for a track. In classical tracking approaches where objects are mostly described through a state vector consisting of a generalized object position, velocity and, if applicable, further derivatives of these quantities, data association can be performed in a point-to-point matching process.

Within Caroline's fusion system, these approaches had to be extended in order to handle spread objects with complex shapes. Three different types of sensor objects have to be processed: complex contours delivered by laser scanners, line-shaped objects delivered by the LIDAR system and classical point-shaped objects received from radar sensors. It is not possible to define a common general object position seen by all sensors, since each sensor will most likely see the target differently. For example the point of reflection delivered from a radar is unknown compared to precise contour measurements gained from a laser scanner. Additionally, as the vehicle moves through the real world, the point of reflection of each sensor type moves on the outline of a real-world object. Therefore a multi-point track model was chosen, describing a detected object by an arbitrary number of contour points and postulating a common movement vector following a rigid body assumption. This way each contour measurement can be matched to the tracked contour point with the best fit. A two-staged data association process was set up, with the first stage serving as a justification as to whether or not track and measurement describe the same real-world object and in the second stage then calculating the optimal contour association between measured and tracked object points. Within stage one, a weighting function counting for the minimum Euclidian distance and similarity of velocities is calculated,

$$w_{i,j} = a \cdot min[|x_k^i - x_l^j|, \forall k, l] + b \cdot |v_i - v_j| \tag{1}$$

with $w_{i,j}$ being a scalar weight for association between track i and measurement j with tracked and measured velocity vectors v_i, v_j, x_k^i, x_l^j being the k^{th} and l^{th} contour point position of track i and measurement j and a, b serve as tuning parameters. A threshold for this weight is further defined and an association below that threshold level will be pushed into stage two.

In stage two, an optimal match between all measured and tracked contour points is calculated based on an association matrix ,

$$\Omega = \begin{bmatrix} |x_1^i - x_1^j| & \dots & |x_1^i - x_l^j| \\ \dots & \dots & \dots \\ |x_k^i - x_1^j| & \dots & |x_k^i - x_l^j| \end{bmatrix} \tag{2}$$

Optimization can be carried out with standard algorithms such as the Hungarian/Munkres method, Nearest Neighbor or similar approaches. We used the fast Minimum-algorithm. This two-staged association process avoids unnecessary computational load on the system, since unlikely associations will

be filtered out in stage one while the computational challenging minimization is only carried out for positive matches.

During pretracking, incoming sensor data is first associated with preliminary track objects (pretracks) using methods similar to those described above. A pretrack carries along a vector of sensor assignments, storing for each sensor type the last assigned sensor object id. A simple Kalman filter based on a constant velocity motion model is calculated for each pretrack to update its position given by incoming sensor data. In addition to the vector of sensor assignments, an update counter is carried along storing the number of positive association events. Taking into account sensor redundancies read from a configuration file, a threshold for track activation is evaluated based on this update counter, depending on the level of redundancy in the affected observation area of that object. A simple description language was implemented to efficiently model these redundancies and to influence the update count threshold for track activation, e.g.:

```
polygon={0,2;10,2;10,-2;0,-2}
modifyCount=2000
condition=( RADARFront && !( LASERFront || LIDARFront ) ),
```

which means for the fusion system "Activate track in a 2 x 10 meter, box-shaped view area after 2000 positive matches when it is only seen by the front radar system and not by the laser scanners or LIDAR sensors", which, in this case, serves as protection against random, unstable false alarms from the radar sensor directly in front of the vehicle.

Tracking and Data Fusion. For the main tracking algorithm, a model-switching Extended Kalman Filter, based on two track motion models was implemented. A six-dimensional motion model describes fast-moving objects using a state vector,

$$x^{6D} = \begin{pmatrix} x_{1\ldots n} \\ y_{1\ldots n} \\ v \\ a \\ \alpha \\ \omega \end{pmatrix} \tag{3}$$

with $x_{1\ldots n}$ and $y_{1\ldots n}$ being the x and y coordinate of the n contour points, the common velocity, acceleration, course angle and course angle velocity with respect to the global earth-fixed reference frame. For slow or static objects, a simpler four-dimensional state vector was chosen,

$$x^{4D} = \begin{pmatrix} x_{1\ldots n} \\ y_{1\ldots n} \\ v \\ a \end{pmatrix} \tag{4}$$

thus taking into account that the majority of detected objects are of a rather static nature and distribution of available sensor information in unnecessary

many state variables is suboptimal in that case. As seen in equations (4) and (5), the classical state vector has been enriched by the number of contour points, thus making it necessary to extend the Kalman Filter algorithm (see [Kalman, 1960] for reference) to handle multiple positions within the same state vector. Similarly, we define the sensor measurement vector for a sensor object consisting of m contour points,

$$y = \begin{pmatrix} x_{1\ldots m} \\ y_{1\ldots m} \\ v_x \\ v_y \end{pmatrix} \tag{5}$$

with x_1, y_1, v_x, v_y being measured contour point x- and y-coordinates as well as x- and y-velocity components with respect to the global earth fixed reference frame. Postulating a common position noise covariance for all contour points within track and measurement, the update algorithm can be extended as follows:

$$\begin{aligned} x_k(v+1|v) &= f(x_k(v)) \\ P(v+1|v) &= F^T \cdot P \cdot F + Q \\ s_{k,l} &= y_l(v+1) - h(x_k(v+1|v)) \\ S(v+1) &= H \cdot P(v+1|v) \cdot H^T + R \\ K(v+1) &= P(v+1|v) \cdot H^T \cdot S(v+1)^{-1} \\ r_{k,l}(v+1) &= K(v+1) \cdot s_{k,l}(v+1) \end{aligned} \tag{6}$$

with x_k being the track state vector regarding contour point k, $f(x)$ the nonlinear system transfer function, P the common state covariance matrix, F the system transfer Jacobian, Q the system noise covariance, $s_{k,l}$ the innovation vector of tracked contour point k compared with measured point l of the associated sensor object, y_l the sensor measurement vector regarding measured point l, $h(x)$ the nonlinear system output function, S the common innovation covariance matrix, H the system output Jacobian, R the estimated measurement noise, K the Kalman gain in this update cycle and $r_{k,l}$ the correction vector for tracked contour point k getting updated with measured point l.

The tracked contour points can then be updated by adding the first two components of the associated vector $r_{k,l}$. In order to calculate updated common velocity, acceleration, course angle and course angle velocity in the six dimensional movement model, the mean value for vector $r_{k,l}$ is calculated over all given contour point associations,

$$r_{mean} = \frac{1}{N} \sum_{k,l=1}^{N} r_{k,l} \tag{7}$$

with N being the total number of acquired contour point matches within the second stage of data association. Corrected common values can then be

acquired by adding the last four components of vector r_{mean} to the corresponding elements in the track state vector.

Obviously, by postulating a common system and measurement noise covariance for all contour points, Kalman gain can be computed once per update cycle. While it would theoretically be possible to calculate a separate Kalman gain for each tracked contour point and therefore removing the limitations to system and measurement covariance, this would lead to a N-times bigger computational load, since matrix inversion of the system innovation covariance matrix is the most costly part of the algorithm. In this case, the algorithm would simply calculate a separate Kalman filter for each contour point, which is not practically realizable in a real-time application. In the approach described we have no significantly higher computational effort compared to a standard EKF, while at the same time realizing spread-contour functionality and removing the need for a stable point of reference for tracked objects.

In order to prevent the track from being flooded with contour points, a garbage collection mechanism was installed by carrying along update counters for each contour point, which stores the last update timestamp and the overall number of updates counted to that point in time. In this manner, inactive contour points can be detected easily and removed from the track's point list.

Object splitting and termination. Because of the track's polyline object model, it is necessary to implement a track splitting algorithm. If there is no such method, one track can collect points from many objects and grow to a rather huge but meaningless track. For example, a person dropping off a car and moveing away would still be part of the car track because of the data association algorithm depiced in Fig. 9. When the person just dropped off and is still near the car, it will become one track. After moving away from the car, the contour points will still be updated because there is an object at the position of the car and the person is also still there. Between these two objects there is nothing but the polyline from the track still describing an outline of an object.

To detect these false tracks, an algorithm was developed to split such tracks. The basic idea is to examine the objects based on the raw sensor object data and find indepented sets of objects. These independent sets will become the new tracks. Normally, there are no such sets but in the event of an unsplit track, there are two or more partitions. Polygonal objects around the track will be described as a planar undirected two colored graph. The algorithm contains the following steps:

1. Build planar undirected colorable rectangular graph. set the color of every node to black.
2. Set the polylines of every sensor object of the track to white.
3. Search for independend sets in the graph [Cormen et al., 2002].
4. If there is more than one set found, build new tracks from the points describing the outlines.

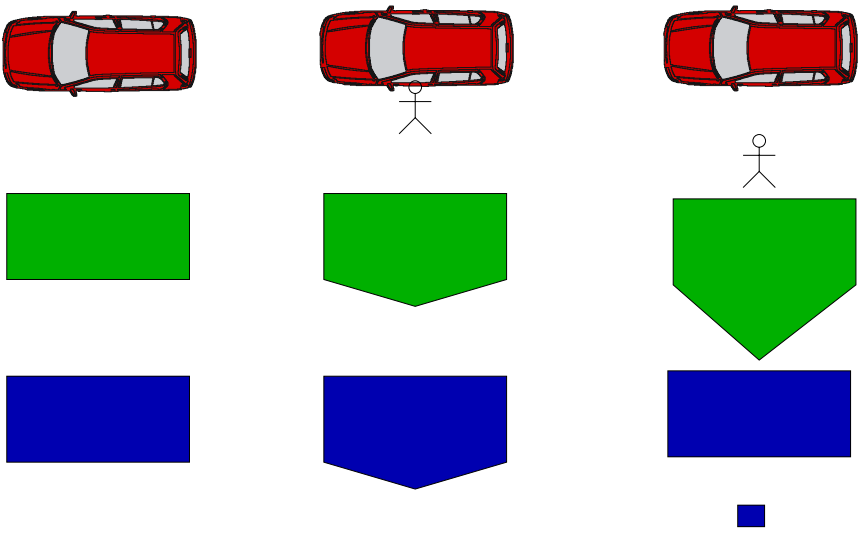


Fig. 9. Person who drops off a car. From left to right: Person still in the car, person just dropped off, person moves away. From top to bottom: Reality, track without splitting, track after splitting.

The algorithm runs periodically during track garbage collection. Although complexity depends on the maximal area (a) covered by a track ($O(a)$), this algorithm can be implemented efficiently with graphic libraries.

4.1.3 Grid-Based Fusion

In contrast to the object tracking subsystem, the grid fusion system does not describe agents in the vehicle's environment with discrete state vectors, but instead discretizes the whole environment into a rectangular matrix (grid) structure. Each grid cell carries a number of assigned features:

- a height value expressed in the global earth fixed reference frame,
- a gradient value describing the height difference to neighboring cells,
- a set of Dempster-Shafer probability masses counting for the hypotheses undrivable, drivable and unknown,
- a status flag stating whether or not measurement data has been stored within the corresponding cell and
- an update time stamp storing the last time a cell update was carried out.

Data Structure. The biggest challenge with grid based models in an automotive environment is the need for real time operation. High maneuvering speeds in automotive applications require update rates greater than 10 Hz, which is almost too low since this equals a travel distance of 1.4 meters at normal urban speeds. The approach of discretizing the environment into grid

cells leads to significantly high memory requirements and therefore calls for efficient data structures. As an example, the storage of a view area of only 100 x 100 meters with a resolution of 25 centimeters generates 160,000 grid cells. Assuming a 4-byte floating point value for each feature as described above, this grid extends up to 3 MByte. Together with an update rate of 10 Hz this leads to a constant data throughput of 256 MBit/s, which in any case is more than the standard automotive bus infrastructure would be able to handle. Efficient algorithms and data reduction prior to serialization is therefore the key to a successful application. For addressing these issues we implemented a rolling grid data structure wherein the vehicle's own position is a pointer to the corresponding grid cell. This position will be regarded as virtual origin for all incoming sensor readings, which can then be subsequently accessed by moving through the double linked data structure relative to that virtual origin. The main grid is again subdivided into sub grids whose size match the processor's caching mechanism for optimal usage of the available computing resources. While it would theoretically be possible to make the surface large enough to cover the expected maneuvering area, this would lead to extremely high memory usage and is therefore not feasible. Instead, when the vehicle moves through the world, the reference point shifts along the double linked spherical list. As soon as it crosses the border from one sub grid to the next, the corresponding sub grids at the new horizon of the data structure are cleared and are therefore available for new data storage.

Treatment of laser and stereo vision point data. As the first step within grid data fusion, the 3-dimensional point clouds received from the laser scanners and stereo vision system are transformed into the global earth fixed reference frame taking vehicle attitude into account (roll, pitch and yaw) as acquired from the GPS/INS unit and sensor-specific calibration information. The accuracy of these transformations is crucial to subsequent post-processing. Vehicle height as delivered by the GPS is especially important and is therefore subject to further filtering and justification. For each measured point, the corresponding grid cell is retrieved and a ray tracing algorithm (Bresenham) is carried out to update all cells from the sensor coordinate system's origin to the measured target point. Several versions of the Bresenham algorithm are described in the literature, in this case we will introduce the 2D version following Pitteway [Pitteway and M.L.V., 1967] for reasons of simplicity.

The lines are traced similar to the functionality of a plotter, which is basically the origin of that algorithm. On the way through the traced lines, each cell passed is updated according to following rules:

- If the cell lies on the path between sensor origin and measured target point and it's height value exceeds the current Bresenham line height value, reduce the stored value to that of the current Bresenham line.

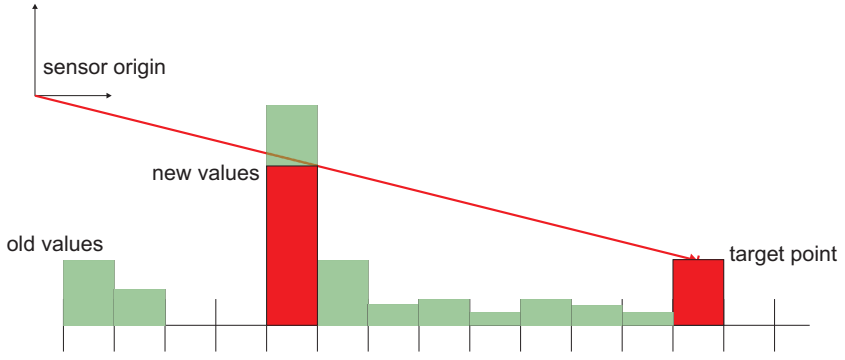


Fig. 10. Ray update mechanism.

- If the cell is the end point of the Bresenham line, store the associated height value.
- In both cases, store update time stamp and mark that cell as having been measured.

The grid is updated following the direct optical travel path of any laser ray (or virtual stereo vision ray) starting at the sensor origin and ending at the target point as depicted in Fig. 10. This model follows the assumption that any obstacle would block the passing optical ray and therefore any cells on the traveling path must be lower than the ray itself.

Data Fusion. Parallel to entering the 3-dimensional point data acquired from laser scanners and stereo vision, vision-based classification is processed using a Dempster Shafer approach [Shafer, 1976], [Shafer, 1990]. A sensor model was created for each data source, mapping the sensor specific classification into the Dempster Shafer probability mass set, which can then be fused into the existing cell probability masses using Dempster's rule of combination,

$$m_c^*(A) = m_c(A) \bigoplus m_m(A) = \frac{1}{1-K} \sum_{B \cap C = A \neq \emptyset} m_c(B) m_m(C), \qquad (8)$$

with m_c, m_m being the cell and measurement probability mass set and m_c^* the combined new set of masses for the regarded cell, while the placeholders A, B and C can describe any of the three hypotheses drivable, undrivable and unknown. The term K expresses the amount of conflict between existing cell data and incoming measurement, with

$$K = \sum_{B \cap C = \emptyset} m_c(B) m_m(C). \qquad (9)$$

The mass set m_m has to be modeled out of the acquired sensor data.

With respect to the stereo vision system, which is capable of classifying retrieved point clouds into the classes road, curb and obstacle, this mapping is trivial and can be done by assigning an appropriate constant mass set to each classification result. The exact values of these masses can then be subject to further tuning in order to trim the fusion system for maximum performance given real sensor data.

In the case of the mono vision system, Caroline assigns each pixel in the retrieved image a drivability value P_d between 0.0 representing undrivable and 1.0 representing fully drivable; a mapping function is then applied, which creates the three desired mass values D: drivable, U: undrivable and N: unknown, that can be either drivable or undrivable as follows:

$$\begin{aligned} m_m(D) &= D_{max} \cdot P_d, \\ m_m(N) &= (1 - D_{max}), \\ m_m(U) &= 1 - m_m(D) - m_m(N). \end{aligned} \tag{10}$$

The value D_{max} will serve as a tuning parameter, influencing the maximum trust placed into the mono vision system and based on the quality of its incoming data. Both, the classification mechanism of the stereo vision system would be beyond the scope of this paper and will therefore not be explained in detail. Basically, classification within the stereo system is based upon generating a mesh height model out of the point cloud obtained and applying adaptive thresholds to this mesh structure in order to characterize roadway, curb and obstacles. The mono vision system is based on a similar approach to [Thrun et al., 2006].

Prior to mapping the mono vision data into the grid data structure, the image must be transformed into the global world reference frame using the known camera calibration [Heikkil and Silvn, 1996] and height information which can easily be retrieved from the grid itself.

The creation of a sensor model for the 3-dimensional height data is more complex: First, a gradient field is calculated from the stored height profile. In Caroline's grid fusion system, the grid is mapped into image space by converting into a grayscale image data structure, with intensity counting for cell height values. Subsequently, the Sobel operator is applied in both image directions.The results of both convolutions are summed and - after proper normalization - transformed back into the grid structure, storing the gradient values $\frac{\partial h}{\partial x \partial y}$ for each grid cell. Any existing obstacle will usually lead to a bigger peak within the gradient field, which can easily be detected. During the process of forward and reverse transformation, the grid structure in- and out of a grayscale image would initially appear to be redundant, because the gradient operator could easily be applied to the height field itself. Yet, by transforming the information into a commonly used image format, the broad variety of image processing algorithms and operators found in standard image processing toolkits, such as the OpenCV library [OpenCV Website, 2007] can easily be applied, thereby significantly reducing development time.

The acquired gradient values will then subsequently be mapped into a Dempster-Shafer representation, which leads to the desired sensor model combining all acquired height values. Similar to the method with the mono vision system, a mapping function is defined as follows:

$$m_m(D) = \begin{cases} D_{max}, & \left|\frac{\partial h}{\partial x \partial y}\right| \leq G_{D_{max}} \\ 0, & G_{D_{max}} < \left|\frac{\partial h}{\partial x \partial y}\right| \leq G_{U_{min}} \\ 0, & \left|\frac{\partial h}{\partial x \partial y}\right| > G_{U_{min}} \end{cases}$$

$$m_m(U) = \begin{cases} 0, & \left|\frac{\partial h}{\partial x \partial y}\right| \leq G_{D_{max}} \\ \frac{U_{max}}{G_{U_{min}} - G_{D_{max}}} \cdot \left(\left|\frac{\partial h}{\partial x \partial y}\right| - G_{D_{max}}\right), & G_{D_{max}} < \left|\frac{\partial h}{\partial x \partial y}\right| \leq G_{U_{min}} \\ U_{max}, & \left|\frac{\partial h}{\partial x \partial y}\right| > G_{U_{min}} \end{cases}$$

$$m_m(N) = 1 - m_m(D) - m_m(U), \tag{11}$$

with D_{max} and U_{max} serving as parameters for maximum drivability/ undrivability assigned to the gradient field, $G_{D_{max}}$ being the maximum gradient value that is still considered to be fully drivable and $G_{U_{min}}$ the minimum gradient value that is considered to be fully undrivable. By carefully tuning those four parameters, it is possible to suppress unwanted smaller gradients resulting e.g. from unimportant depressions and knolls in the road while supporting higher gradients as originating from curbs or berms in order to correctly fuse this information into the grid cells by using Eq. 8.

4.2 Vision

Caroline's computer vision system consists of two separate systems. The first is a monocular color segmentation based system that classifies the ground in front of the car as drivable, undrivable or unknown. It assists in situations where the drivable terrain and the surrounding area (e.g. grass, concrete or shrubs) differ in color. The output of this algorithm contributes to the Grid Based Fusion as described in section 4.1.3. The second vision system is a multi-view lane detection that identifies the different kinds of lanes described by DARPA, such as broken and continuous as well as white and yellow lane markings. Using four high-resolution color cameras and state-of-the-art graphics hardware, it detects its own lane and the two adjacent lanes to the left and right with a field of view of 175 degrees at up to 35 meters. The output of the lane detection algorithm is directly processed by the artificial intelligence.

4.2.1 Lane Detection

Detecting lane markings on roads in an urban environment is a difficult but very important task. While concepts exist that depend on additional markings, such as magnetic bands in the street, a more useful method must make intelligent use of what is available on today's roads. Towards this goal, we

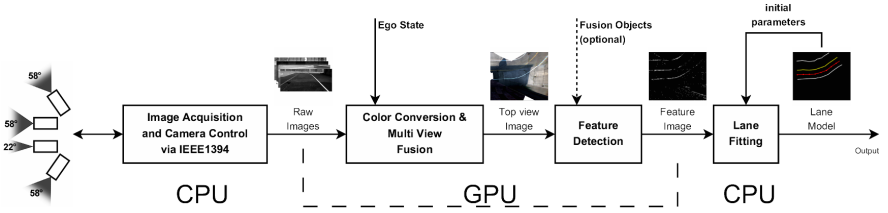
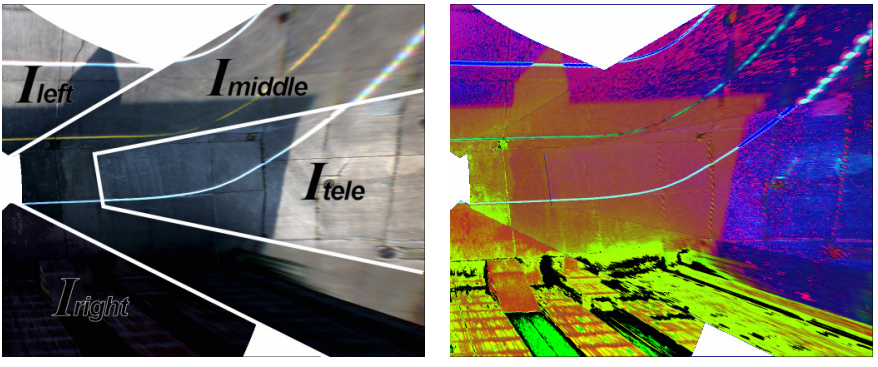


Fig. 11. The four stages of the lane detection algorithm.

developed a lane detection system that is capable of analyzing several high-resolution images simultaneously and in real-time. Our lane fitting algorithm uses a very versatile lane model and is robust with respect to outliers and artifacts. It also takes into account lane markings of adjacent lanes. It copes with different road setups, lane markings and lighting situations. The lane detection process is divided into four parts, as shown in Fig. 11. First, the raw images are downloaded from the cameras via the IEEE1394b interface. Second, they are uploaded to graphics hardware, the color information is retrieved from the raw Bayer pattern, and the images are transformed into a single top view perspective, Fig. 12. Third, lane marking features are detected in the image, Fig. 14. In the last step, a lane model is adjusted to match the features detected.

Data Acquisition. Three cameras with field of view of 58 degrees cover the area in front of the car. A 22 degrees telephoto lens camera provides a high-resolution view of the street ahead of the car. The four 1376x600 8-bit raw Bayer images are synchronously acquired via the IEEE1394b interface at 14 frames per second. The images are uploaded to the graphics card and converted to the RGB color space using bilinear interpolation. As the lane fitting algorithm works in a global coordinate system, the position and rotation of the vehicle, also referred to as Ego State, must be available. A transformation function $f_{ego} : p_{car} \mapsto p_{world}$ can be defined if the Ego State is known, where p_{car} is a point in the car's reference system, and p_{world} is a point in a global Cartesian reference system. An Inertial Measurement Unit corrected by a GPS signal was used to generate the Ego State.

Multi-View Fusion. Because local changes of the light intensity are an indicator for white lines, and local saturation changes indicate colored lane markings, the RGB images are converted to the HSV color space. This color space encodes saturation and color in separate channels. Knowing the intrinsic and extrinsic parameters of the camera, and including the orientation of the vehicle (pitch and roll), a lookup function that converts top view coordinates to image coordinates can be used to create a single HSV top view image. The lookup operation is applied to each source image. In regions where



(a) RGB top view (b) HSV top view

Fig. 12. The four different images (a, RGB color space used for visualization) are merged to a single HSV top view image (b).

the projected images overlap, precedence $I_{tele} > I_{middle} > I_{left} > I_{right}$ is maintained as shown in Fig. 12. The region of interest covers the area of up to 30 meters in front of the vehicle and 12 meters to the left and right at a scale of 35 pixels per meter.

Features. Lane markings can be described as a thin pattern of local differences of the road surface that cover long distances. Therefore, the basic concept underlying feature detection involves identification of these local differences in regions of 8x8-pixels that resemble road patches of approximately 25 by 25 centimeters. Analyzing the HSV top view image, the feature detection's output is a downsampled feature image that encodes the quality, direction and color, i.e., white or yellow, of the lane features in Fig. 14. As lane markings exist in various colors, qualities as well as widths, and appear differently under changing lighting conditions, only few stringent assumptions apply. When analyzing the top view image for features, we check three criteria that must be present:

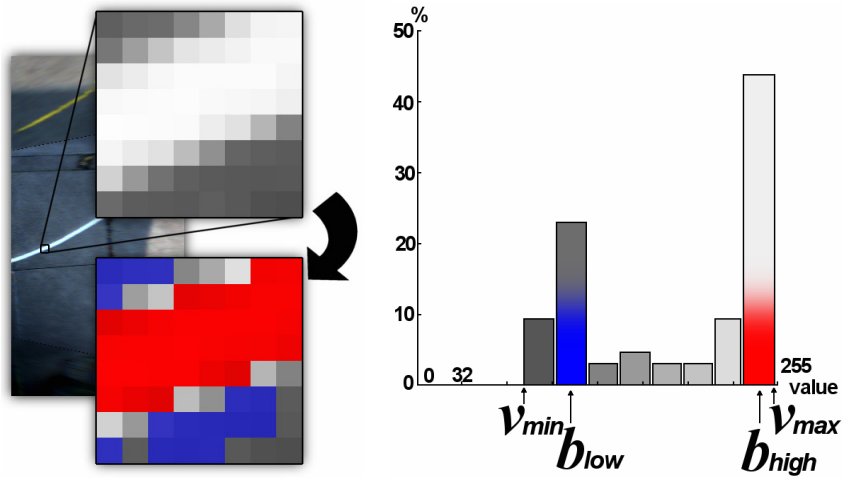
1. The local contrast v_{diff} must exceed a certain threshold. The local contrast is the difference between the local minimal and maximal value $v_{diff} = v_{max} - v_{min}$.
2. Analyzing a local adaptive histogram, the distance b_{diff} between the two largest bins b_{high} and b_{low} must exceed a certain threshold. This is because it can be assumed that b_{low} contains pixels depicting the street and b_{high} identifies the lane marking.
3. The pixels in b_{high} must have a recognizable shape and orientation. For several discrete orientations, the ratio of the variances of the pixels' x- and y-coordinates is checked.

A detailed description is given in Alg. 1. As this algorithm is prone to discretization errors, supersampling improves the quality of the feature detection.

Data: An 8x8 region of a HSV top view image, thresholds $t_{con}, t_{hist}, t_{dir}$ and t_{col}
Result: A feature quality q, direction $a \in \{0, 22.5, ..., 157.5\}$ and color $c \in \{white, yellow, undecided\}$

for *the saturation and lightness channel* **do**
  $v_{diff} = v_{max} - v_{min}$; v_{max} and v_{min} are the maximal and minimal values of the current channel
  if $v_{diff} < t_{con}$ **then**
    break;
  end
  compute adaptive histogram;
  determine two largest bins b_{high} and b_{low}, Fig. 13(b) ;
  $b_{diff} = b_{high} - b_{low}$;
  if $b_{diff} < t_{hist}$ **then**
    break;
  end
  set of pixels p_{high} = pixels in b_{high};
  determine center of mass R of p_{high};
  initialize r_{max} and a_{max} to 0;
  for $i = 0; i <= 157.5; i = i + 22.5$ **do**
    rotate p_{high} around R by i degrees. determine ratio of variances $r = \frac{Var(X)}{Var(Y)}$;
  end
  if $r_{max} < t_{dir}$ **then**
    break;
  end
  label this region as a feature;
  if *current channel is lightness* **then**
    $q_{white} = b_{diff}$; $a_{white} = a_{max}$
  else
    $q_{yellow} = b_{diff}$; $a_{yellow} = a_{max}$
  end
end
if $q_{white} > t_{col}$ *&* $q_{white} > q_{yellow}$ **then**
  $c = white$; $a = a_{white}$
end
if $q_{yellow} > t_{col}$ *&* $q_{yellow} > q_{white}$ **then**
  $c = yellow$; $a = a_{yellow}$
end
$q = max(q_{white}, q_{yellow})$;

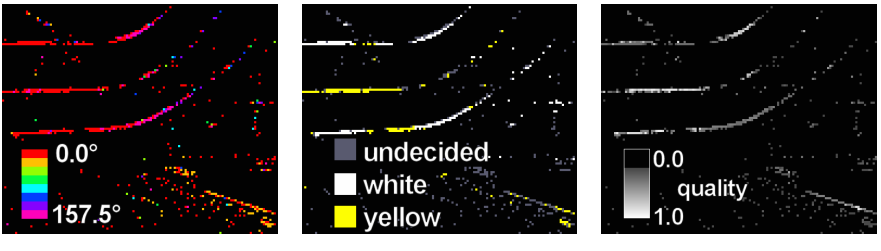
Algorithm 1. Feature detection algorithm.



(a) 8x8 regions are analyzed (b) 8 bin histogram of the 8x8 region

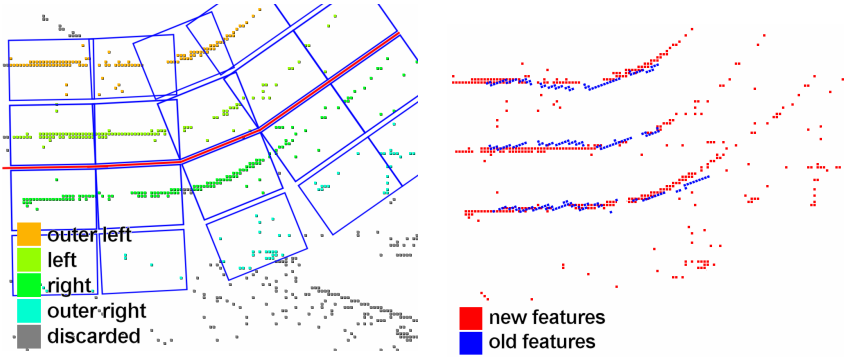
Fig. 13. 8x8-pixel regions of the top view image (a, up) are tested for possible features. The distance between the two largest bins b_{low} (b, blue) and b_{high} (b, red) of the histogram determines the quality of the feature. The pixels gathered in b_{high} must be arranged in a directed shape (a, red area).

Lane Model. The lane model consists of connected lane segments. Each segment s_i is described by a length l_i (given parameter), a width w_i and an angle $d_i = \alpha_i - \alpha_{i-1}$ describing the difference of orientation between this segment and the previous one as shown in Fig. 16. The first segment is initially placed on the current coordinates of the vehicle and facing in the driving direction, assuming that the vehicle is actually located on the street. Knowing the position c_0 of the initial segment as well as the lengths l_i and the



(a) direction information (red channel) (b) color information (green channel) (c) quality information (blue channel)

Fig. 14. The direction (a), color (b) and quality (c) of the features are encoded in an RGB image downloaded from the graphics card. For visualization purposes, the channels encoding the direction (a) and color (b) are colorized.



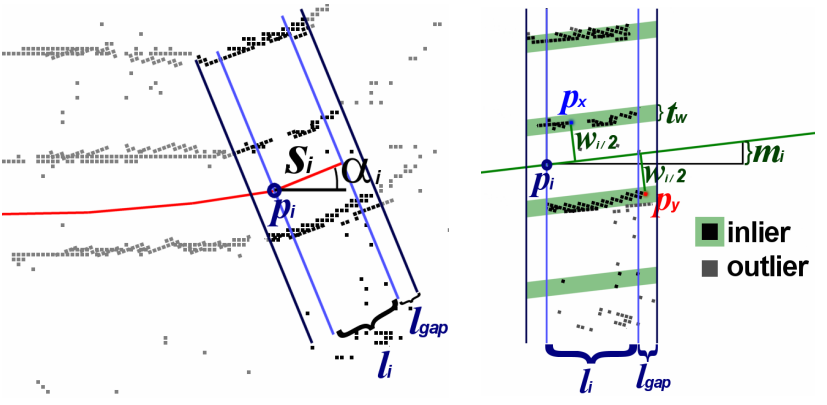
(a) labeling and discarding (b) mixing

Fig. 15. Regions of interest (a, blue boxes) determine to which lane marking features are assigned. Afterwards, old and new features are mixed (b).

angular changes d_i of all segments, the position p_i and global orientation α_i of each segment can be computed. Each segment contains information whether the vehicle's lane is confined by lane markings and whether additional lanes to the left and right exist. Straight streets, sharp curves and a mixture of both can all be described by the model.

Lane Fitting. The main goal of the lane fitting algorithm is to find a parameter set for a lane model that explains the features found in the current top view image and the previous frames. In order to create a global model of the lane, all feature points are mapped to world space coordinates and inserted into a list l_p. This is done using the function $f_{ego} : p_{car} \mapsto p_{world}$ defined by the current Ego State. Old data, i.e. feature points gathered during previous frames, may be kept if the features of a single image are too sparse. For each frame, the existing lane model or an initial guess is used to define four regions of interest as shown in Fig. 15(a). These are the regions expected to contain the own lane's markings and the lane markings of the adjoining lanes. If a feature is inside such a region, it is labeled as *outer left*, *left*, *right* or *outer right*. Otherwise, it is discarded. Afterwards, features from previous frames are mixed with the new data as depicted in 15(b).

As the first currently visible segment s_f of the lane model is determined, older segments are no longer considered. If the list of lane segments is empty, it is initialized with $s_0 \leftarrow s_f$. Starting from s_f, each segment s_i is estimated (or reestimated if it has previously been estimated). Therefore, an initial guess as to the orientation α_i of s_i is made as shown in Fig. 16. All local features relevant for estimating s_i are rotated by α_i around the starting point p_i of s_i. A RANSAC algorithm is used to estimate the parameter d_i and w_i: Iteratively, two feature points p_x and p_y are chosen. Assuming that they are located on the lane markings they were labeled for, the gradient $g_i = m_i/l_i$ as



(a) estimating segment s_i (b) RANSAC fitting

Fig. 16. p_i, α_i, l_i and l_{gap} identify the features relevant for s_i. After rotating around α_i, a RANSAC fitting eliminates outliers among the features.

well as the width w_i are derived from their coordinates. All features that are also sufficiently described by g_i and w_i are counted as inliers. This process is repeated n times and the parameter set with most inliers is used to define s_i. A quality function q takes into account the ratio of inliers and outliers, the amount of inliers, the quality of the features and states the quality of the segment. The quality is computed for every region of interest (*outer left*, *left*, *right* and *outer right*). If the maximum of these qualities exceeds a threshold t_q, the segment is considered to be valid and the next segment s_{i+1} is estimated. After all segments are estimated as shown in Fig. 17, a proposal about the lane markings' colors can be made by looking at the inliers' average color.

Results and Evaluation. The algorithm was thoroughly tested on several sites in northern Germany and Texas. A frame rate of 10 fps could be maintained using a 2 GHz Intel Core 2 Duo with a GeForce 7600 GTS graphics card. The testing sessions included different weather and lighting conditions. The amount of false positives was reduced significantly by utilizing the vehicle's other sensors. The objects detected by lidar and radar sensors were used to mask out regions in the feature image where other cars, walls, cones and poles caused irritating artifacts in the top view image.

4.2.2 Area Processor

The Area Processor consists of a single IDS color camera whose images are interpreted by a color segmentation algorithm suitable for urban environments. This algorithm separates an image into areas of drivable and non-drivable terrain. Assuming that a part of the image is known to be drivable terrain, other parts of the image are classified by comparing the Euclidean distance of each

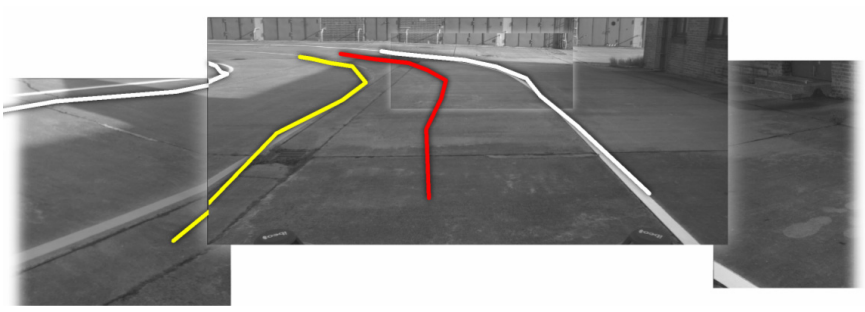
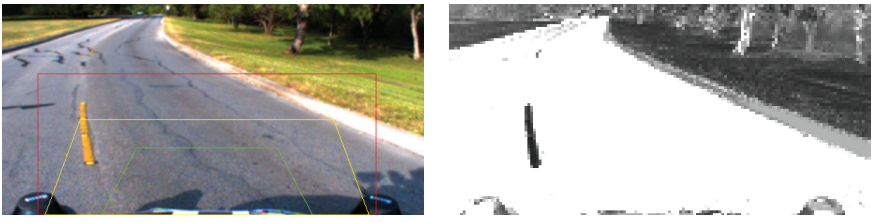


Fig. 17. The lane model reprojected onto the original images.

pixel's color to the mean colors of the drivable area in real-time. Moving the search area depending on each frame's result ensures temporal consistency and coherence. Furthermore, the algorithm classifies artifacts such as white and yellow lane markings and hard shadows as areas of unknown drivability. Although Caroline is able to perform basic driving tasks without this algorithm, it is needed in situations when terrain cannot be distinguished by other sensors, i.e., sections without proper lane markings, streets without high curbs and off-road tracks.

Related work. As a foundation for the area detection algorithm we used the real-time approach suggested by Thrun et al. [Thrun et al., 2006] in the 2005 DARPA Grand Challenge. The basic idea is to consider a given region in the actual image as drivable. The predominant mean color values in that area are retrieved and compared to the pixel values in the entire image. Similar pixels are marked as drivable. The algorithm was designed for off-road terrain, therefore it cannot be applied to urban scenarios without fundamental modifications. We will describe the algorithm in the next section. The Expectation Maximization (EM) algorithm used for color clustering in this approach is thoroughly described in [Duda and Hart, 1973] and [Bilmes, 1997]. Instead of the EM algorithm, the KMEANS algorithm that we used during the competition is also suitable for color clustering, as described in [Gary Bradski, 2005]. An algorithm similar to the one mentioned above points out the advantage of other color spaces than RGB [Ulrich and Nourbakhsh, 2000], e.g., the HSI space.

The Stanford University algorithm for detecting drivable terrain. The main idea of the algorithm is to use the output of the laser scanner, normally a scan-line, which is integrated over time to a height map in world coordinates. A polygon is defined that covers an area in front of the car identified as level and therefore as a drivable surface. This polygon is transformed into image coordinates from the camera and clipped to the image boundaries. The resulting polygon is considered as the area that is drivable. In this



(a) original picture (b) normal drivability grid

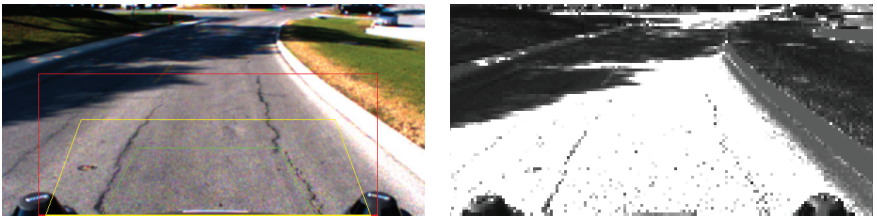
Fig. 18. The drivability grid (b) depicts the output of algorithm, the results differ from black (undrivable) to white (drivable) . A yellow line (a) is marked as undrivable (b, black) because the color differs by too much from the street color.

area the pixels' color values are collected and clustered by color, for example bright grey and yellow. These color clusters are compared to the color values of each pixel in the image using distance measurements in the color space. If a resulting distance is smaller than a given threshold, the area comprised by the pixel is marked as drivable. The main benefit of the algorithm is that the range in which drivability can be estimated is enhanced from only a few meters to more than 50 meters.

Problems arising in urban and suburban terrain. Designed for competing in a 60 mile desert course, the basic algorithm succeeds well in explicit off-road areas, which are limited by sand hills or shrubs. When tested in urban areas new problems occur, because there are streets with lane markings in different colors or tall houses casting long shadows. The yellow lane markings are often not inside the area of the polygon $P_{Scanner}$ (output of the laser scanner), so they are not detected as drivable. Especially non-dashed lines prohibit a lane shift as shown in Fig. 18 and stop lines seem to block the road.

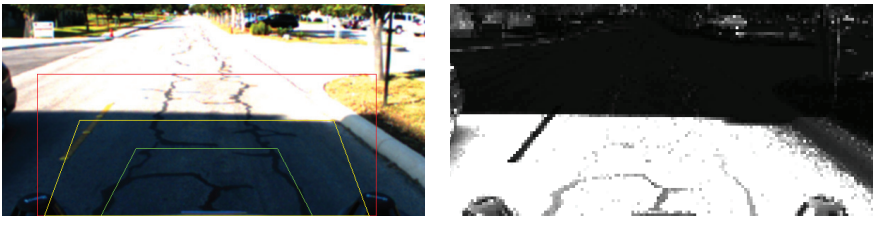
Another problem are shadows cast by tall buildings during the afternoon. Small shadows from trees in a fairly diffuse light change the color of the street only slightly and can be adapted easily. But huge and dark shadows appear as a big undrivable area as shown in Fig. 19. Even worse: Once inside a shadowed area, the camera auto exposure adapts to the new light situation, such that the area outside the shadow becomes overexposed and appears again as a big undrivable area as depicted in Fig. 20.

Another problem during the afternoon is the car's own shadow, in this paper referenced as "egoShadow", when the sun is behind the car. Sometimes it is marked as undrivable, sometimes it is completely adapted and marked as drivable, but the rest of the street is marked as undrivable as shown in Fig. 21. A fourth problem occurs when testing on streets without curbs but limited by mowed grassy areas. The laser scanner does not recognize the grass as undrivable, because its level is about the same as the street niveau. This



(a) original picture (b) normal drivability grid

Fig. 19. Large, dark shadows (a, left) differ too much from the Street Color (b, dark).

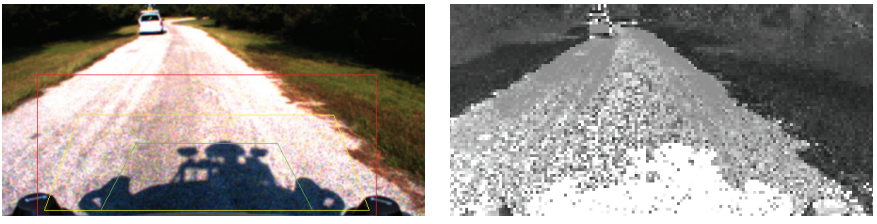


(a) original picture (b) normal drivability grid

Fig. 20. Exposure is automatically adapted inside shadows (a). Areas outside the shadow are overexposed and are marked as undrivable (b, dark).

causes the vehicle to move onto the grass, so that colors are adapted by the area processing algorithm, and consequentially keeps the car on the green terrain.

Alterations to the basic algorithm. Differing from the original algorithm, our implementation does not classify regions of the image as drivable and undrivable. The result of our distance function is mapped to an integer number ranging from 0 to 127, instead of creating a binary information via a threshold. In addition, a classification into the categories 'known drivability' and 'unknown drivability' is applied to each pixel. These alterations are required because the decision about the drivability of a certain region is not made by the algorithm itself, but by a separate sensor fusion application. For the sake of performance the KMEANS Nearest Neighbors algorithm was chosen instead of the EM-algorithm, because the resulting grids are almost of the same quality but the computation is considerably faster. Tests have shown that better results can be achieved by using a color space that separates the luminance and the chrominance in different channels, e.g. HSV, LAB, YUV. The problem with HLS and HSV is that chrominance information is coded in one hue channel and the color distance is radial. For example, the color at 358 degrees is very similar to that one at 2 degrees, but they are numerically very



(a) original picture (b) normal drivability grid

Fig. 21. The vehicle's own shadow can lead to problems (a), for example if only the shadowed region is used to detect drivable regions (b, white).

far away from each other. Thus a color space is chosen where chrominance information is coded in two channels, for example in YUV or LAB, where similarity between two colors can be expressed as Euclidean distance.

Preprocessing. To cope with the problems of large shadows and lane markings, a preprocessing system was developed. Before the camera picture is processed, it is handed over to the following preprocessors: White preprocessor (masking out lane markings and overexposed pixels), black preprocessor (masking out large, dark shadows), yellow preprocessor (masking out lane markings), egoShadow preprocessor (masking out the car's shadow in the picture). The output of each preprocessor is a bit mask (1: feature detected, 0: feature not detected), which is used afterwards in the pixel classifying process, to mark the particular pixel as "unknown", which means that the vision-based area processor cannot provide valid information about the area represented by that pixel. In the following, the concept of each preprocessor is described briefly:

White Preprocessor. In order to deal with overexposed image areas during shadow traversing, pixels whose brightness value is larger than a given threshold are detected. The preprocessor converts the given image into HSV color space and compares the intensity value for each pixel with a given threshold. If the value is above the threshold, the pixel of the output mask is set to 1.

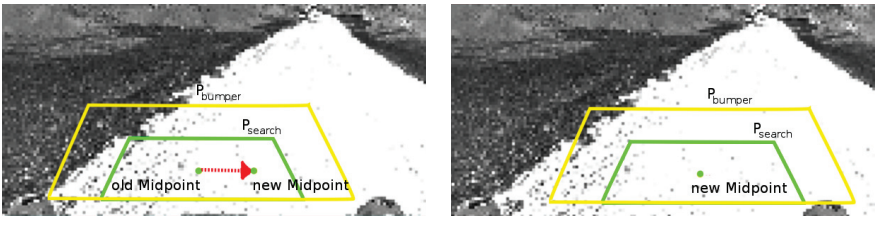
Black Preprocessor. As huge dark shadows differ too much from the street color and would therefore be labeled as impassable terrain, pixels whose brightness value is smaller than a given threshold are masked out. The preprocessor analogously converts the given image into HSV color space and compares the intensity value for each pixel with a given threshold. If the value is below the threshold, the pixel of the output mask is set to 1.

Yellow Preprocessor. Small areas of the image which are close to yellow in the RGB color space are detected so that yellow lane markings are not labeled

as undrivable but rather as areas of unknown drivability. For each pixel of the given image, the RGB ratios are checked to detect yellow lane markings. If the green value is larger than the blue value and larger or a slightly smaller than the red value, the pixel is not considered yellow. If the red value is larger than the sum of the blue and the green values, the pixel is also not considered yellow. Otherwise, the pixel is set to $\frac{min(R,G)}{B} - 1$. Afterwards, a duplicate of the computed bit mask is smoothed using the mean filter, dilated and subtracted from the bit mask to eliminate huge areas. For different areas of the image, different kernel sizes must be applied. In the end, only the relatively small yellow areas remain. A threshold determines the resulting bit mask of this preprocessor.

EgoShadow Preprocessor. When the sun is behind the car, the vehicle's own shadow appears in the picture and is either marked as undrivable, or it is the only area marked as drivable. Therefore, a connected area directly in front of the car is identified whose brightness value is low. At the beginning of the whole computing process a set of base points $p(x, y)$ is specified, which mark the border between the engine hood and the ground in the picture. The region of interesst in each given picture is set to y_{max}, the maximum row of the base points, so that the engine hood is cut off. From these base points the preprocessor starts a flood-fill in a copy of each given image, taking advantage of the fact that the car's shadow appears in similar colors. Then the given picture is converted to HSV color space and the flood-filled pixel are checked to determine if their intensity value is small enough. Finally, the sum of the flood-filled pixels is compared to a threshold, which marks the maximum pixel area that constitutes the car's own shadow.

The dynamic search polygon. Using the output of the laser scanner to determine the input polygon works quite well if the drivable terrain is limited by tall objects such as sand hills or shrubs. In urban terrain, however, the output of the laser scanner must be sensitized to level distances smaller than curbs (10 to 20 centimeters), which becomes problematic if the street moves along a hill where the distance is much higher. Thus, the laser scanner



(a) dynamic polygon at time t (b) dynamic polygon at time $t + 1$

Fig. 22. This Fig. shows how the dynamic search polygon (a, green trapezoid) is transposed to the right (b) because the calculated moment is positive in x-direction.

polygon does not remain a reliable source especially because both modules solve different problems: The laser scanner focuses on range-based obstacle-detection [Ulrich and Nourbakhsh, 2000], which is based on analysis of the geometry of the surroundings, whereas the vision-based area processor follows an appearance-based approach. For example, driving through the green grass next to the street is physically possible, and therefore not prohibited by a range-based detection approach, but it must be prevented by the appearance-based system. This led to the concept of implementing a self-dynamic search polygon which has a static shape, but is able to move along both the X- and the Y-axis in a given boundary polygon $P_{boundary}$. The initial direction is zero. Every movement is computed using the output of the last frame's pixel classification. For the computation a bumper polygon P_{bumper} is added, which surrounds the search polygon. The algorithm proceeds in the following steps:

Implementation and Performance. The algorithm has been implemented with the Intel OpenCV library [OpenCV Website, 2007]. The framework software is installed on an Intel Core 2 Duo Car PC with a Linux operating system and communicates with an IDS uEye camera via USB. The resolution of a frame is 640*480, but the algorithm applied downsampled images of size 160*120 to attain a manually adjusted average performance of 10 frames per second. The algorithm is confined to a region of interest of 160*75 cutting of the sky and the engine hood.

In Fig. 23 the difference between normal area processing and processing with the black preprocessor is shown. Without the preprocessor, the large shadow of a building to the left of the street is too dark to be similar to the street color and is classified as undrivable. The black preprocessor detects the shadowy pixels, which are classified as unknown (red).

The problem of overexposed areas in the picture is shown in Fig. 24, where the street's color outside the shadow is almost white and therefore classified as undrivable in the normal process. The white preprocessor succeeds in marking the critical area as unknown, so that the vehicle has no problem in leaving the shadowy area.

(a) original picture (b) normal drivability grid (c) with black preprocessor

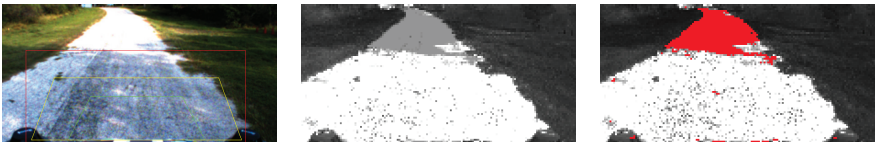
Fig. 23. The results with black preprocessor. The picture in the center (b) shows the classification results without the black processor. In picture on the far right (c) the critical region is classified as unknown (red).

Data: last frame's grid of classified pixels, actual bumper polygon P_{bumper}
Result: updated position of the Polygons P_{bumper}

begin
  Initialize three variables $pixelSum$, $weightedPixelSumX$, $weightedPixelSumY$ to zero
  foreach *pixel of the grid which is inside the bumper* **do**
    count the amount $pixelSum$ of visited pixels
    if *drivability of the actual pixel is above a given threshold* **then**
      Add the pixel's x-Position relative to the midpoint of P_{bumper} to $weightedPixelSumX$
      Add the pixel's y-Position relative to the midpoint of P_{bumper} to $weightedPixelSumY$
    end
  end
  Perform the division $x_{moment} = \frac{weightedPixelSumX}{pixelSum}$ and $y_{moment} = \frac{weightedPixelSumY}{pixelSum}$ and round the results to natural numbers
  /* The value x_{moment} gives the amount and direction of the movement of P_{bumper} in x-direction, the value y_{moment} gives the amount and direction of the movement of P_{bumper} in y-direction. */
  Add the values x_{moment} and y_{moment} to the values of the actual midpoint of P_{bumper} to retrieve the new midpoint of P_{bumper}
  Check the values of the new midpoint of P_{bumper} against the edges of $P_{boundary}$ and adjust the values if necessary
  Add the values x_{moment} and y_{moment} to the values of the actual midpoint of the search polygon to retrieve the new midpoint of the search polygon as shown in Fig. 22
  To prevent that the search polygon gets stuck in a certain corner, it is checked, if $x_{moment} = 0$ or if $y_{moment} = 0$
  /* For example if $x_{moment} = 0$, it is evaluated, if the midpoint of P_{bumper} is located right or left to the midpoint of $P_{boundary}$; x_{moment} is set to 1, if P_{bumper} is located left, otherwise it is set to -1. An analogous check can be performed for the y_{moment}. */
end

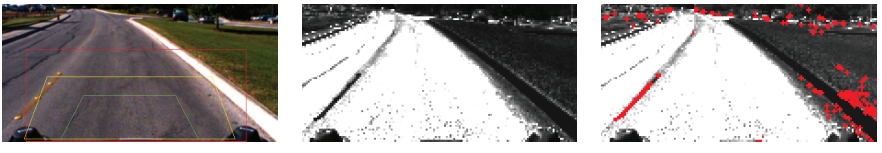
Algorithm 2. Dynamic search polygon algorithm.

Yellow lane markings differ from pavement in color space so that a human driver can easily detect them even under adverse lighting conditions. This advantage turns out to be a disadvantage for a standard classification system, which also classifies the lane markings as undrivable, as shown in Fig. 25: Lane markings are interpreted as tiny walls on the street. To counteract this problem, we use a preprocessing step, which segments colors similar to yellow. To deal with different light conditions, the color spectrum must be wider so that a brownish or grayish yellow is also detected. This leads to some false



(a) original picture (b) normal drivability grid (c) with white preprocessor

Fig. 24. The results with white preprocessor.The picture in the center (b) shows the classification results without white processor. In picture on the far right (c) the critical region is classified as unknown (red).



(a) original picture (b) normal drivability grid (c) with yellow preprocessor

Fig. 25. The results with yellow preprocessor. The picture in the center (b) shows the classification results without yellow preprocessor. In picture on the far right (c) the lane marks are classified as unknown (red).

(a) original picture (b) normal drivability grid (c) with egoShadow preprocessor

Fig. 26. The results with egoShadow preprocessor. The picture in the center (b) shows the classification results without egoShadow processor. In picture on the far right (c) the car's own shadow is classified as unknown (red).

positives as shown in Fig. 25, but the disturbing lane markings are clearly classified as unknown. The vehicle is now able to change lanes without further problems.

A problem with the vehicle's own shadow only occurs when the sun is located behind the vehicle, but in these situations the classification can deliver insufficient results. Figure 26 shows the shadowy area in front of the car as unknown.

The benefit of a search polygon that is transposed by the output of the last frame is tested by swerving about so that the car moves very close to the edges of the street. Figure 27 shows the results when moving the car close to the left edge. As the static polygon touches a small green area, a

(a) original picture (static Polygon) (b) drivability grid (static Polygon) (c) original picture (dynamic Polygon) (d) normal drivability grid (dynamic Polygon)

Fig. 27. The same frame first computed with a static search polygon (a, b), then with the dynamic polygon (c, d). The dynamic movement calculation caused the polygon to move to the right (c).

somewhat green mean value is gathered and so the resulting grid shows a certain amount of drivability in the grassland, whereas the dynamic polygon moves to the right of the picture to avoid touching the green pixels so that the resulting grid does not show drivability on the grassland.

4.3 Artificial Intelligence

4.3.1 The DAMN-Architecture

To control Caroline's movement, the artificial intelligence computes a speed and a turning wheel angle for every discrete step. Turning the steering wheel results in different circle-radii on which the car will move. Instead of the radii, the approach is based on the inverse, a curvature.

A curvature of 0 represents driving straight ahead, while negative curvatures result in left and positive curvatures in right turns as shown in Fig. 28.

This curvature, as the most important factor to influence, is selected in an arbiter as described in the DAMN-architecture [Rosenblatt, 1997]. This architecture models each input as behavior, which gives a vote for each possible curvature. More behaviors can be added easily to the system, which makes it very modular and extendable. The following behaviors are considered:

- Follow waypoints: Simply move the vehicle from point to point as found in the RNDF.
- Stay in lane: Vote for a curvature that keeps Caroline within the detected lane markings.
- Avoid obstacles: Vote for curvatures that keep the vehicle as far away from obstacles as possible and forbid curvatures leading directly into them.
- Stay on roadway: Avoid curb-like obstacles detected by grid-based fusion with laser scanners and color camera.
- Stay in zone: Keep the vehicle in the zone, defined by perimeter points in the RNDF.

All collected votes are weighted to produce an overall vote. The weights again are not fixed, they depend on factors including distance to an intersection, presence of lanes and more. A trajectory point is calculated by following the best voted curvature for one meter. A trajectory point holds information

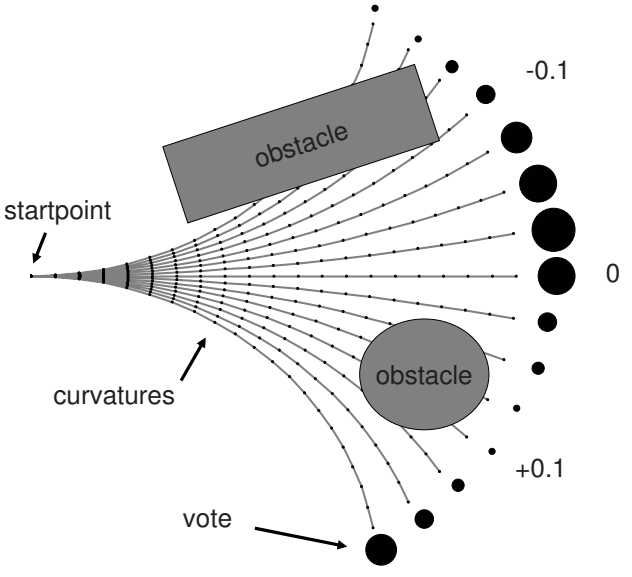


Fig. 28. Curvature field: Larger black circles represent preferred votes.

such as position, orientation and speed. Starting at this trajectory point, all behaviors vote again for curvatures to find the next point until a list of points is computed. This list has to be long enough to come to a complete stop at current speed. The speed is controlled by another arbiter influenced by different behaviors, which each provide a maximum speed. The arbiter simply selects the lowest of these speeds. These behaviors are: RNDFMax, sensor health, zone, reverse, safety zone, obstacle distance and following other obstacles. Based on the trajectory points calculated iteratively we design a drivable corridor for further processing by the next module in the chain, the path planner.

4.3.2 Interrupts

Because the AI has to deal with more complex situations, e.g. stopping at a stopline and yielding the right-of-way, than the DAMN-architecture is designed for, we extended DAMN by an interrupt system. At each trajectory point found each interrupt is called upon to decide if it wants to be activated at its location. If so, the speed stored in the trajectory points is reduced to bring the car to a smooth stop. If the point is reached, the interrupt is activated and the arbiters are stopped until the interrupt returns control to the arbiters. Some of our interrupts are:

- Intersection: Activated at a stopline until it is our turn.
- Queue: Wait in a line at an intersection.

- Overtake: Stop the car when the lane is blocked and wait for other lane to clear to start passing maneuver.
- U-turn: Activated at a dead-end street - this interrupt actually performs the U-turn and turns the car around.
- Road blocked: Activated if the entire road is blocked - this interrupt then activates the U-turn interrupt when appropriate.
- Parking: Activated at a good alignment in front of the parkbox - this interrupt returns control after the parking maneuver is finished.
- Pause: Active as long as the car is in pause mode.
- Mission complete: Final checkpoint is reached.

An example can be seen in Fig. 29, where the queueing interrupt has to be activated at some point in the future and the speed must therefore be reduced.

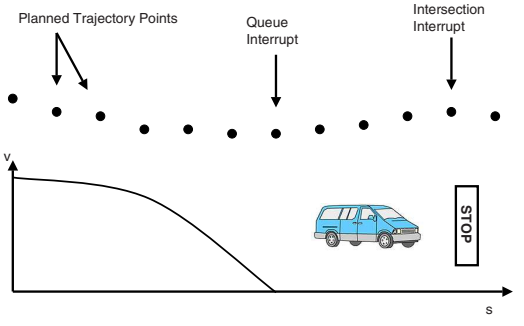


Fig. 29. Interrupt example.

4.3.3 Example

An example of how different behaviors interact is shown in Fig. 30. In the recorded situation, Caroline just started overtaking another car, blocking its lane. The plots represent the calculation of one trajectory: 20 trajectory points are calculated from the front to the back. For each point votes for 40 curvatures are made, these are displayed from left to right.

The lane behavior (a) demands a sharp left for the first four curvatures, then a right turn which finally transitions to straight driving. This would bring Caroline quickly to the free lane to pass the obstacle vehicle. The obstacle behavior (b) has two obstacles effecting the votes: On the left, a wall forbids going farther to the left, on the right one can see the car that is be passed. Finally the waypoint behavior (c) wants to go to the right all the time, because that is the lane where Caroline should be and where the waypoints are, but is outvoted by the other behaviors in (d).

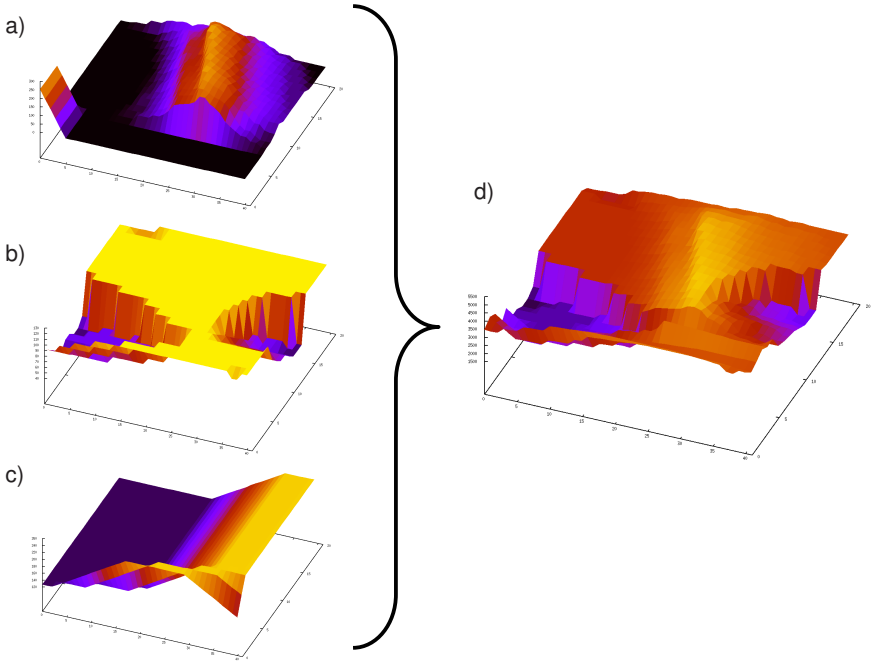


Fig. 30. Votes of a) stay in lane, b) avoid obstacles, c) follow waypoints, d) weighted sum.

4.4 Vehicle Control

Lateral and longitudinal control are the basics of autonomous vehicle guidance. In the following, both concepts as installed in Caroline for the DARPA Urban Challenge are discussed in detail.

4.4.1 Longitudinal Control

While the maximum and minimum speed of the vehicle is chosen by the artificial intelligence, the controller must calculate the braking and accelerator set points in order to maintain a given speed.

For this purpose, the longitudinal controller is separated into an outer and an inner loop controller. Based on the given speed set point, the outer loop controller determines the required acceleration. Finally, the inner loop controller calculates throttle and brake input to track the required acceleration. The acceleration of the vehicle, which is needed for feedback of the lower controller, is provided in high resolution by the GPS/INS system.

Gear shifting is handled via an automatic gear box. However, to switch between forward, backward and parking state, an automatic lever arm is attached at the gearshift. The lever arm position can be commanded with a CAN (Controller Area Network) interface.

Longitudinal Dynamics. The driving power must be greater than the sum of all driving resistances, that is the sum of rolling, air and acceleration resistance. Engine torque M_M is a function of throttle α_A, engine speed n_M and engine acceleration $\dot{n}_M$.

$$M_M(\alpha_A, n_M, \dot{n}_M) = \frac{r}{\eta_k \, i_k} \left(f_R \, m \, g + c_w \, A \, \frac{\rho}{2} \left(\frac{n_M \, 2 \, \pi \, R_0}{i_k} \right)^2 + \lambda \, m \frac{\dot{n}_M \, 2 \, \pi \, R_0}{i_k} \right) \tag{12}$$

The meaning of the parameter is given in table 1.

Table 1. Longitudinal model parameters.

Symbol	Parameter
R_0	Wheel Radius, Unloaded
r	Wheel Radius, Loaded
η_k	Degree of Efficiency, Gear Box
i_k	Gear Transmission Ratio
f_R	Rolling Friction Factor
m	Mass
g	Gravity
c_w	Air Resistance Factor
A	Cross Sectional Area
ρ	Air Density
λ	Moulding Bodies Factor

The model is used for the inner loop controller to simulate different control strategies for the longitudinal control. The plant model for the outer loop controller is the transfer function between desired vehicle acceleration and actual vehicle speed. The inner loop is approximated as a PT1 element. In addition, an integral element is needed to integrate the speed from acceleration:

$$P(s) = \frac{1}{s \, (T \, s + 1)} \tag{13}$$

Introducing measured values of the drive chain into the model, leads to a value of $T = 0.6s$ for system lag.

P-PD-Control Controller Cascade. As mentioned above, the longitudinal controller is separated into an outer and inner control loop. The block diagram in Fig. 3 depicts the control structure. $K(s)$ stands for each transfer function of the different controller parts. Different control parameters are used for acceleration and deceleration. While a PD controller is applied for the inner loop, a P controller is introduced for the outer control loop. Control outputs for acceleration and braking are combined via a predefined logic to prevent the system from activating throttle and brake at the same time.

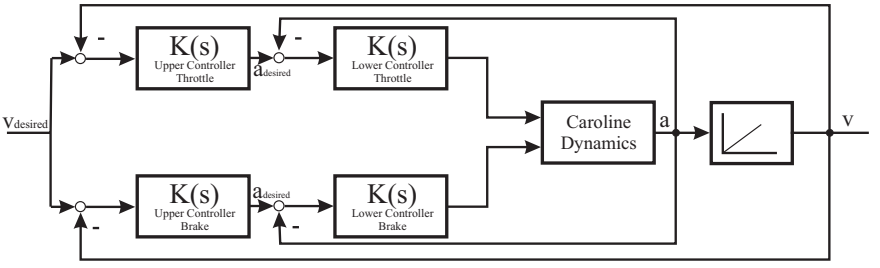


Fig. 31. Block diagram of the longitudinal controller.

In addition, an engine map can be used for direct feed forward of the throttle. Fig. 32 shows a typical implementation of an engine map for longitudinal control.

Performance of the Longitudinal Controller. Figure 33 illustrates the performance of the longitudinal control strategy. Two different examples are shown with two different speed profiles. While in the first example, the desired speed is changed in long and large steps, in the second example the speed is changed in shorter and smaller steps. The desired as well as the actual speed of Caroline are illustrated.

4.4.2 Lateral Control

It is the main goal of the lateral controller to follow a given trajectory with a minimum of track error. Secondly, vehicle driving maneuvers should match certain comfort parameters for smooth driving experience.

Vehicle Dynamics. For simulation of the vehicle as well as design of the controllers it is necessary to describe motion behavior with a mathematical model. In the following the bicycle model is used. The bicycle model is based on the following assumptions:

- The center of mass of the car is located at street level.
- Two wheels of each axle are combined as one wheel in the center of the axles.
- The longitudinal acceleration is zero.
- The wheel load of all wheels is constant.
- Lateral forces at the wheel are proportional to skew angle.

A state space representation within following structure is preferred:

$$\dot{\mathbf{x}}(t) = \underline{\underline{A}}\,\mathbf{x}(t) + \underline{\underline{B}}\,\mathbf{u}(t) + \underline{\underline{E}}\mathbf{z}(t), \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{14}$$

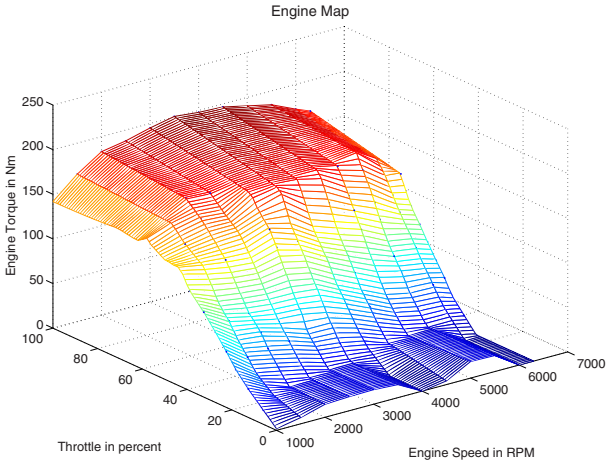


Fig. 32. Engine map.

Track error and track angle deviation have to be described mathematically to take them into consideration. Track angle deviation is defined as the difference between desired and actual orientation of the car. It is assumed that the derivation of the track angle $\zeta_{desired}$ can be calculated as the product of the curvature κ of the track and the current speed v:

$$\zeta_{desired} = \kappa \cdot v \tag{15}$$

Yaw angle ψ_{rel} with respect to the desired track is the difference between absolute yaw angle ψ and desired track angle $\zeta_{desired}$:

$$\psi_{rel} = \psi - \zeta_{desired} \tag{16}$$

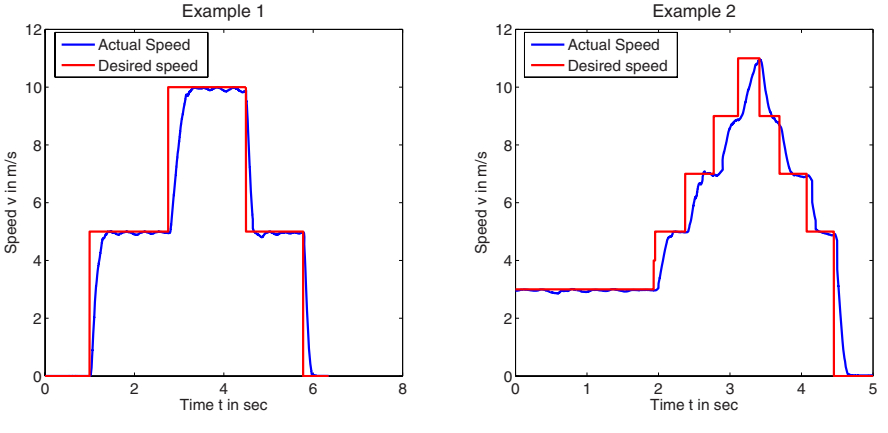


Fig. 33. Performance of the longitudinal controller.

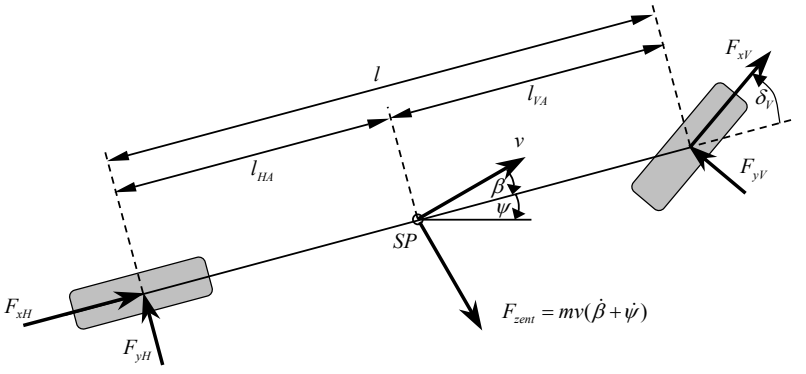


Fig. 34. Bicycle model.

As a result, yaw rate $\dot{\psi}_{rel}$ with respect to the desired track can be determined:

$$\dot{\psi}_{rel} = \dot{\psi} - \kappa\, v \tag{17}$$

Moreover, the derivation of the track error $\dot{d}$ can be formulated based on speed v, attitude angle β and relative yaw angle ψ_{rel}:

$$\dot{d} = v\,(\beta + \psi_{rel}) \tag{18}$$

The state space representation of the bicycle model can be combined with the mathematical representation of the track error, track angle deviation and an additional time delay T_L between commanded and actual steering wheel angle. The state vector consists of yaw rate $\dot{\psi}$, attitude angle β, relative yaw angle ψ_{rel}, track error d and actual steering angle δ. The result is the following state space model with the commanded steering angel $\delta_{desired}$ as the input variable and curvature κ as outer noise:

$$\begin{pmatrix} \ddot{\psi} \\ \dot{\beta} \\ \dot{\psi}_{rel} \\ \dot{d} \\ \dot{\delta} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & a_{15} \\ a_{21} & a_{22} & 0 & 0 & a_{25} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & v & v & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{T_L} \end{pmatrix} \cdot \begin{pmatrix} \dot{\psi} \\ \beta \\ \psi_{rel} \\ d \\ \delta \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{i_L}{T_L} \end{pmatrix} \cdot \delta_{desired} + \begin{pmatrix} 0 \\ 0 \\ -v \\ 0 \\ 0 \end{pmatrix} \cdot \kappa \tag{19}$$

with

$$a_{11} = -\frac{c_{\alpha V}\, l_V^2 + c_{\alpha H}\, l_H^2}{\theta\, v}, \quad a_{12} = -\frac{c_{\alpha V}\, l_V + c_{\alpha H}\, l_H}{\theta}, \quad a_{15} = \frac{c_{\alpha V}\, l_V}{\theta} \tag{20}$$

$$a_{21} = -1 - \frac{c_{\alpha V}\, l_V - c_{\alpha H}\, l_H}{m\, v^2}, \quad a_{22} = -\frac{c_{\alpha V} + c_{\alpha H}}{m\, v}, \quad a_{25} = \frac{c_{\alpha V}}{m\, v} \tag{21}$$

The parameters are described in table 2.

The output of the system is the track error d.

$$\mathbf{y}(t) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \end{pmatrix}^T \mathbf{x}(t) \tag{22}$$

Based on the state space model, the transfer function can easily be determined. The control transfer function is

$$F_c(s) = \frac{i_L}{T_L\, s + 1} \cdot \frac{a_{25} s^2 + (a_{15}\, a_{21} + a_{15} - a_{25}\, a_{11})\, s + (a_{25}\, a_{12} - a_{25}\, a_{12})}{s^2 - (a_{11} + a_{22}) s + (a_{11}\, a_{22} - a_{12}\, a_{21})} \cdot \frac{1}{s} \cdot \frac{v}{s} \tag{23}$$

and the noise transfer function:

$$F_{noise} = -\frac{v}{s} \cdot \frac{v}{s} \tag{24}$$

Table 2. Parameters of the bicycle model.

Symbol	Parameter
$c_{\alpha V}$	Skew Stiffness, Front Wheel
$c_{\alpha H}$	Skew Stiffness, Back Wheel
l_V	Wheel Base Front to Center of Mass
l_H	Wheel Base Back to Center of Mass
θ	Moment of Inertia
m	Mass

Parallel Structure Control. As modeled, the vehicle has three degrees of freedom, which are the x and y position as well as the orientation ψ of the car. Only the steering angle δ is available for controlling the system. As a result, the three degrees of freedom are handled simultaneously. Track error and track angle deviation are used as feedback signals. The working point is chosen at the speed of 30 km/h.

Figure 35 shows the structure of the control strategy used. Again, $K(s)$ stands for each transfer function of the controller. It consists of two parallel control loops for track error and track angle deviation as well as a pilot control taking the curvature of the desired trajectory into consideration. The map-based pilot control algorithm calculates the steering angle that would be needed to follow the desired track based on parameters of the bicycle model.

Performance of the Lateral Controller. Lateral control strategy has to handle different kinds of trajectories. On the one hand, the vehicle has to follow trajectories with a curvature of approximately $\kappa \approx 0$ at higher speeds. On the other hand, the track error in twisting areas is supposed to be as small as possible. Figure 36 shows an example of a trajectory that consists of a long straight part and two sharp curves. On the straight section, the vehicle is accelerated up to a speed of almost $v = 50$ km/h. The curves are driven at a speed of approximately 20 km/h. The speed profile is shown in

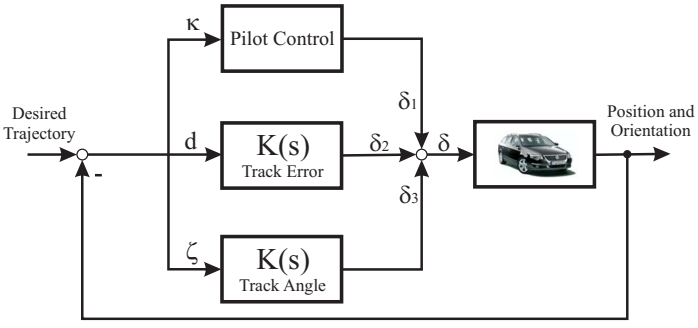


Fig. 35. Lateral control strategy.

Fig. 37. The performance of the control strategy in terms of track error can be seen in the same figure.

The control strategy shown worked well during all tests and missions during the DARPA Urban Challenge. It has always been stable with quite a low track error.

4.5 Safety

The safety systems of Caroline have to ensure the highest possible safety for the car and the environment in both manned or unmanned operation. It has to monitor the integrity of all viable hardware and software components. In case of an error, it has to bring the car to a safe stop. Furthermore, it

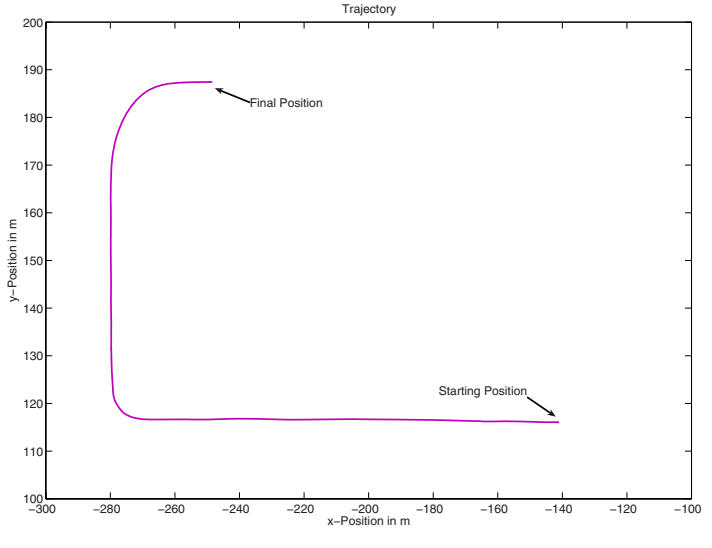


Fig. 36. Trajectory.

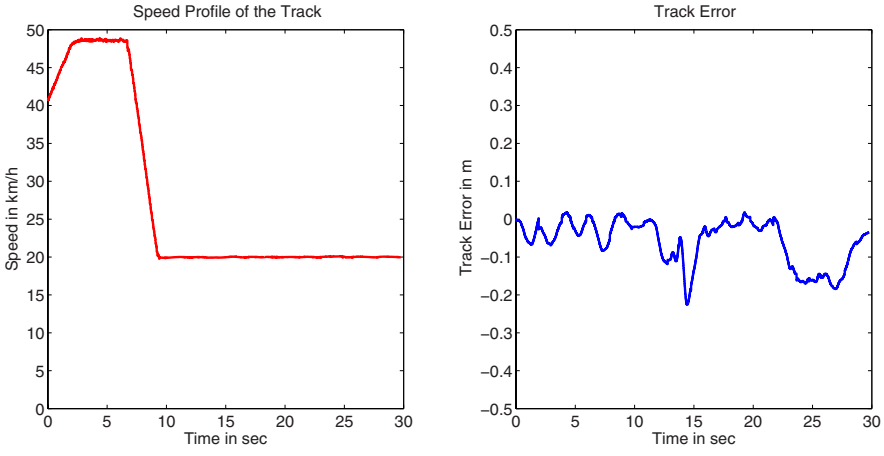


Fig. 37. Speed profile and track error of the trajectory.

must provide an interface for pausing or disabling the car using a remote E-stop controller. We extended these basic features by including the possibility to reset and restart seperate modules independently using hardware and/or software means in order to gain the option of automated failure removal. Figure 38 depicts this watchdog concept.

Caroline is equipped with two separate brake systems. The main hydraulic system and an additional electrical parking brake. The main hydraulic brake is controlled by pressure, usually generated with a foot pedal by the driver. In autonomous mode, this pressure is generated by a small hydraulic brake booster. The parking brake is controlled by a push button in the front console.

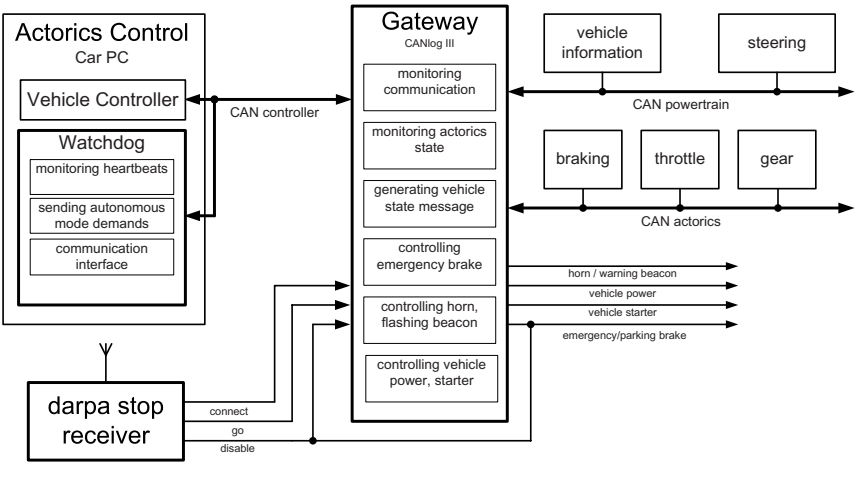


Fig. 38. Watchdog architecture.

This brake is a useful additional feature. If the button is pressed while the car is rolling, the main brake system is activated in addition to the parking brake until the car comes to a complete stop. During autonomous mode, the watchdog gateway, the emergency buttons on the top of the car and the receiver for the remote E-stop controller form a safety circuit, which holds a safety relay open. This relay is connected to the push button for the parking brake. If one of the systems fails, or is activated, the safety circuit is opened, the contact of the relay is closed and the push button of the parking brake is activated. During emergency braking, the lateral controller of the car is still able to hold the car on the given course.

Although the watchdog's main purpose is to assure safety it also increases the system's overall reliability. Caroline is a complex system with custom or pre-production hardware and software modules. These components were developed in a very short time and therefore are not as reliable as off-the-shelf commercial products. For this reason we used devices primarily implemented for all safety-relevant subsystems in order to also provide the means to monitor and reset non-safety relevant subsystems.

Each host runs a local watchdog slave daemon, which monitors all local applications as shown in Fig. 39. A process failing to send periodic heartbeats within a given interval indicates a malfunction, such as memory leakage or deadlocks. Therefore the process and all dependent processes are terminated by the local watchdog slave, to be restarted with respect to the order required by process dependencies.

The slave watchdog itself is monitored by a remote central master watchdog. This approach allows the detection of malfunctions that cannot be resolved by the local slave watchdog, e.g. if a computer freezes. If a computer should freeze, an emergency stop is initiated and the failed system is power-cycled to restart in a stable state. The master watchdog is monitored by the

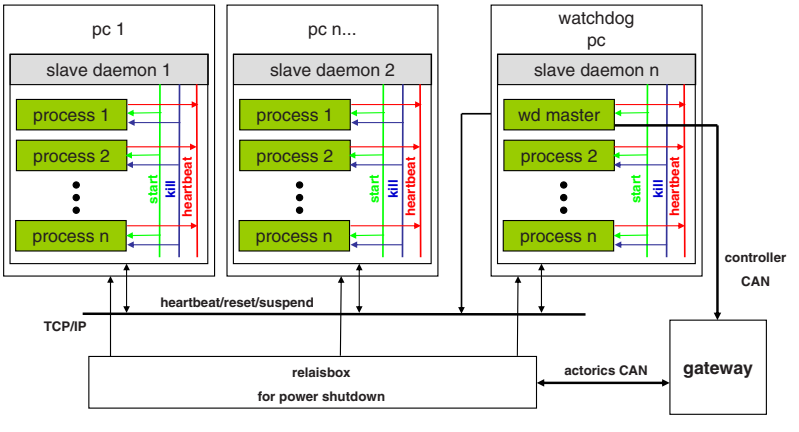


Fig. 39. Software watchdog architecture.

CAN gateway, which initiates an emergency stop on failure of the master watchdog.

5 System Development Process

For developing Caroline's software and ensuring its quality, we implemented a multi-level testing process using elements of extreme programming [Beck, 2005] partly realized in an integrated tool chain shown in Fig. 40. The workflow for checking and releasing software formally consists of five consecutive steps. First the source is compiled to check for syntactical errors. While running the test code, the memory leak checker valgrind [Nethercote and Seward, 2003] checks for existing and potential memory leaks in the source code. After the execution of the test code, source code coverage is computed by simply counting the executed statements. The intent is to implement test cases that completely cover the existing source code or to find important parts of the source code that are still lacking test cases. The last step is for optimization purposes only and executes the code in order to find time-consuming parts inside an algorithm.

The tool chain is executed manually by the developer or by using an integrated development environment such as Eclipse. The tool chain itself can be customized by the developer by selecting only necessary stages for the current run, i.e. skipping test suites for earlier development versions of an algorithm. Nevertheless, the complete tool chain is executed every time a new version of the source code is checked in the revision system Subversion [Collins-Sussmann et al., 2004]. Therefore, an independent bugbuster server periodically checks for new revisions on the server. If a new version is found, it is checked out into a clean and safe environment so that the complete tool chain can be run. The results are collected and a report is automatically generated. The report is easily accessible through the project's web

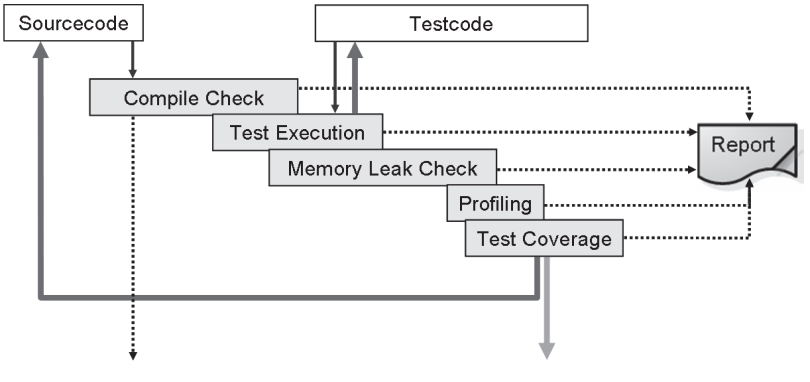


Fig. 40. Workflow for testing and releasing software.

portal [Edgewall Software, 2007] for every developer. For measuring the performance or consulting the results of a previous revision, the history of older revisions is kept and accessible via same the web portal.

The main development process described above mainly covers only unit tests[Liggesmeyer, 2002] for some functions or parts of the complete software system. For the development of Caroline's artificial intelligence, interactive feed back tests using riskless simulations are necessary. Furthermore, the interactive simulations describe different situations for testing the artificial intelligence. After completing the interactive tests, they can be formalized in acceptance tests for automatic execution on another independent server. These test suites are automatically executed after every change to the revision system comparable to the bugbuster server.

The next section describes the simulator development for the CarOLO project. Afterwards, the adoption of the simulator in automatic acceptance tests is explained. This work continues prior work presented in [Basarke et al., 2007a] and [Basarke et al., 2007b].

5.1 Simulator

The simulation of various and partly complex traffic situations is the key for developing a high quality artificial intelligence that is able to handle many different situations with different types of preconditions. The simulator provides appropriate feedback to the other parts of the system, by interpreting the steering commands and changing the Ego State and the surroundings.

The simulator can be used for interactively testing newly developed artificial intelligence functions without the need for real vehicle. A developer can simply, safely and quickly test the functions. Therefore, our approach is to provide a simulator that can reliably simulate missing parts of the whole software system. Furthermore, the simulator is also part of an automatic test infrastructure described in the next section.

Figure 41 shows the main classes of the core simulator. The main idea behind this concept is the use of sets of coordinates in a real world model as

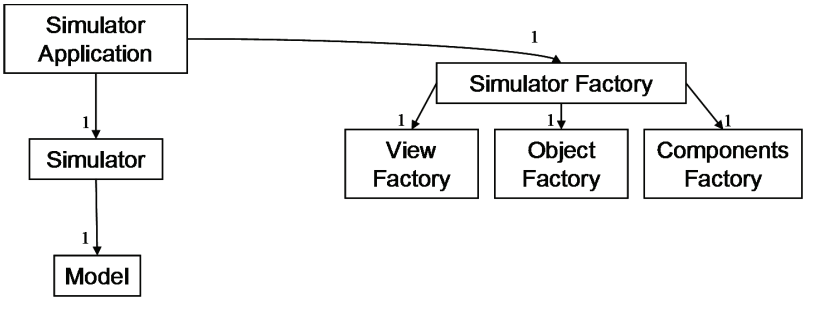


Fig. 41. Main classes of the simulator.

context and input. These coordinates are stored in the model and used by the simulator. Every coordinate in the model is represented by a simulator object position describing the absolute position and orientation in the world. Every position is linked to a simulator object that represents one single object. These objects can have a variety of behaviors, shapes and other information necessary for the simulation. The model is linked with a simulator control that supervises the complete simulation. The simulator application itself controls the instantiation of every simulator component by using object factories.

Figure 42 shows the factories in detail. The simulator view encapsulates a read-only view of an extract of the world model. Every simulator view is linked with a simulator components group. A component represents missing parts of the whole system like an actorics module for steering and braking or a sensor data fusion module for combining measured values and distributing the fused results. Thus, every component in the components group can access the currently visible data of the core data model by accessing the simulator view. As mentioned above, every simulator object position is linked with a simulator object, each of them equipped with its own configuration. Thus, every component can retrieve the relevant data of the owned simulator object.

The main task of the simulator is to modify the world model over time. For simulating the world it is necessary to proceed a step in the simulation. A simulation step is a function call to the world model with the elapsed time step $\delta t_i > 0$ as a parameter that modifies the world model either sequentially or in parallel.

A simple variant is to modify every simulator object sequentially. In this variant, the list of simulator objects is addressed through an iterator and then modified using original object data. Although this is an efficient approach, it is not appropriate when the objects are connected and rely on behaviors from other objects. Another possibility is to use the algorithms as if a copy of the set of simulator object positions were created. While reading the original

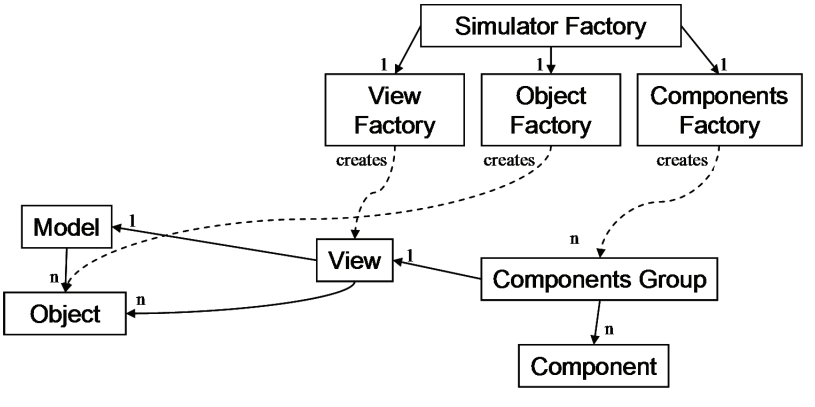


Fig. 42. Object factories creating the simulator's surroundings.

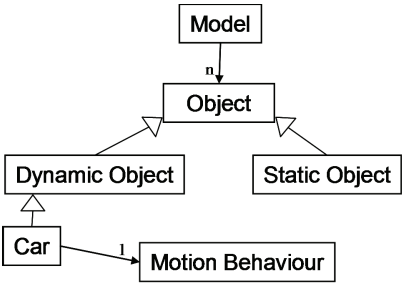


Fig. 43. World's model and motion behavior interface.

data, the modification uses the copy and thus allows a transaction such as a stepwise update of the system, where related objects update their behavior together.

For modifying an object in the world model, every non-static object in the world model uses an object that implements the interface MotionBehavior as shown in Fig. 43. A motion behavior routine executes a simulation step for an individual object. A simulator component implementing a concrete motion behavior registers itself with the simulator object. For every simulation step the simulator object must call the motion behavior and therefore enables the behavior implementation to modify its own position and orientation according to a simulator component. The decoupling of objects and their motion behavior allows us to change the motion behavior during a running simulation, i.e. because of weather influences. Furthermore, it simplifies the implementation of new motion behaviors at development time. For testing Caroline, we have developed additional motion behaviors like MotionBehaviorByKeyboard for controlling a virtual car in the interactive mode by using keys or a MotionBehaviorByRNDF that controls a car in its surroundings by using a predefined route to follow.

The most interesting motion behavior however, is the MotionBehaviorByTrajectory because it communicates directly with the artificial intelligence. For the best imitation of the behavior of the real car, the simulator uses the same code as the vehicle control module based on trajectories expressed as a string of pearls that form consecutive gates. Furthermore, the motion of the simulated car is computed with 3rd order B-splines such as the vehicle controller module. Using a B-spline yields smoother motion in the simulation and a driving behavior sufficiently close to reality - if it is taken into account that for intelligent driving functions it is not necessary to handle the physical behavior in every detail, but in an abstraction useful for an overall correct behavior.

Using motion behaviors, it is possible to compose different motion behaviors to create a new composed motion behavior. For example, it is possible to build a truck with trailer from two related, but only loosely coupled

objects. A composition of the motion behaviors yields a new motion behavior that modifies the position and orientation of the related simulator objects according to inner rules as well as general physical rules.

Getting such a simulator up and running requires quite a number of architectural constraints for the software design. One important issue is that no component of the system being tested tries to call any system functions directly, like threading or communication, but only through an adapter. Depending on whether it is a test or an actual running mode, the adapter decides if the function call is forwarded to the real system or substituted by a result generated by the simulator. Because of the architectural style, it is absolutely necessary that no component retrieves the current time by calling a system function directly. Time is fully controlled by the simulator and therefore knows which time is relevant for a specific software component if different times are used. Otherwise, time-based algorithms will become confused if different time sources are mixed up.

5.2 Quality Assurance

As mentioned at the beginning of this section, the simulator is not only used for interactive development of the artificial intelligence. It is also part of a tool chain that is automatically executed on an independent server for assuring the quality of the complete software system consisting of several modules. In the CarOLO project, we analyzed the DARPA Urban Challenge documents to understand the requirements. These documents contained partly functional and non-functional definitions for the necessary vehicle capabilities. In every iteration a set of tasks consisting of new requirements and bugs from previous iterations is chosen by the development team, prioritized and concretely defined using the Scrum process for agile software engineering [Beedle and Schwaber, 2002]. These requirements are the basis for both a virtual test drive and a real test of Caroline.

After designing a virtual test drive the availability of necessary validators is checked. A validator is part of the acceptance tool chain and responsible for checking the compliance of the artificial intelligence's output with the formal restrictions and requirements. Validators implementing intelligent software functions are used to automatically determine differences in the expected values in the form of a constraint that cannot be violated by the test. A validator implements a specific interface that is called up automatically after a simulator step and right before the control flow returns to the rest of the system. A validator checks, for example, distances to other simulator objects, validates whether a car has left its lane or exceeded predefined speed limits. After an unattended virtual test drive, a boolean method is called upon to summarize the results of all test cases. The results are collected and formatted in an email and web page for the project's web portal.

The set of validators covers all basic requirements and restrictions and can be used for automatically checking the functinality of new software revisions.

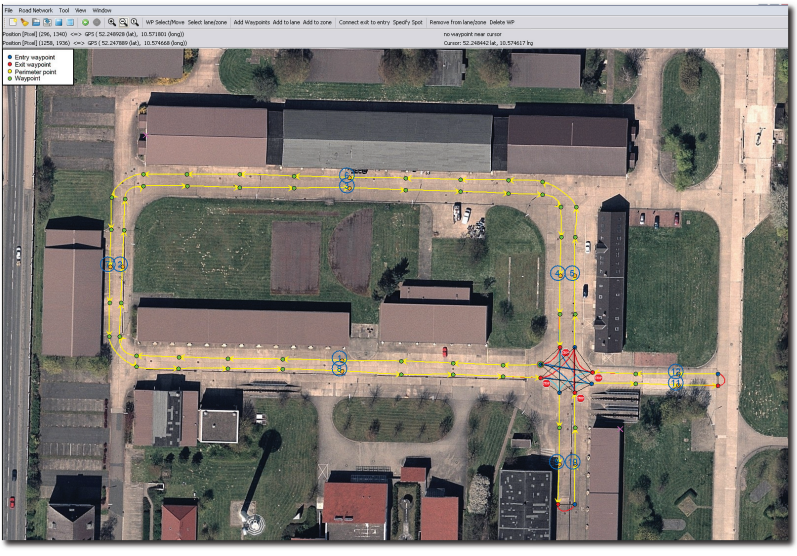


Fig. 44. Screenshot of the GUI tool for constructing RNDFs.

The main benefit is that these high level tests are black-box tests and do not rely on the internal structure of the code. Thus, a subgroup of the CarOLO team was able to develop these high level acceptance tests without a deep understanding of the internal structures of the artificial intelligence. Using this approach, more complex traffic situations could be modeled and repeatedly tested without great effort.

To allow for the quick and convenient creation of test scenarios, various concepts and tools have been developed. The following describes how virtual test drives are defined as well as how certain surroundings such as data fusion objects or drivability data is generated and fed into the simulator. To make this clear we briefly illustrate the proceedings on a basis of an example, which deals with the simple passing maneuver as already described in section 4.3.3. Assume we would like to determine wether the artificial intelligence is able to recognize static obstacles in our travel lane and reacts properly by adhering the required minimum distances.

First, an RNDF must be created that contains information about existing lanes, intersections, parking spots and their connections. As an RNDF provides the basis for every test run, many of those route network definitions had to be created. Therefore we developed a GUI tool to simplify the creation of RNDFs as shown in Fig. 44.

Several features including dragging waypoints, connecting lanes and adding stop signs or checkpoints speed up the construction process. Completed RNDFs could be exported to a text file and used as input for the artificial intelligence as well as for the simulator.

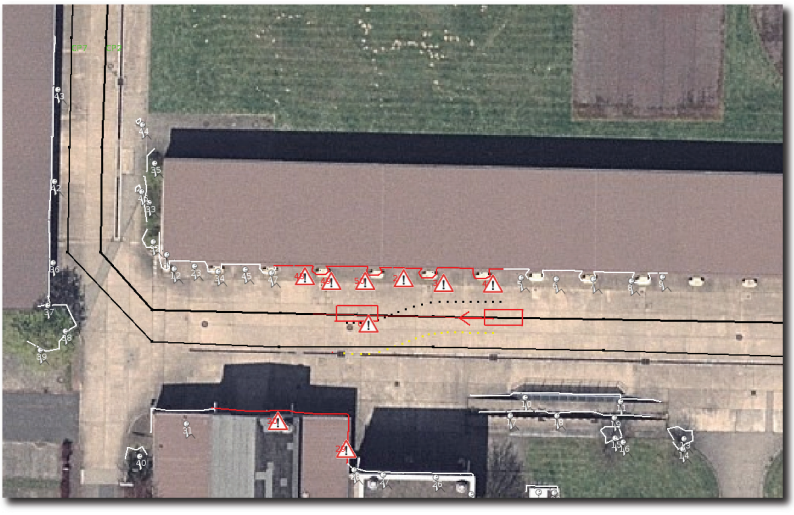


Fig. 45. Screenshot with fusion objects.

The purposes of RNDFs within the simulator vary in different ways. One purpose is to check the behavior of the artificial intelligence concerning the RNDF provided and the actual lane. Therefore a second RNDF can be passed to the simulator. The additional and independent RNDF is used to provide lane data, which is normally detected by the computer vision system. This is especially important if there are major differences between the linear distance and the actual route to the next waypoint.

Another use of RNDFs is to define the behavior of dynamic obstacles during the test run, as mentioned earlier. Thus we are able to check relevant software modules for their interaction with dynamic obstacles. This approach is similar to the one used for providing detected lanes. Dynamic obstacles are interacting on a basis of their individual RNDFs by using the MotionBehaviorByRNDF. This concept can be used for simulating scenarios at intersections and even more complex traffic scenarios.

To extend the example of passing a static obstacle we need to create suitable data, which could be translated to sensor fusion objects. Two principal approaches are available to achieve this goal. Generating scenarios with static obstacles can be accomplished by using our visualisation application, which provides the ability to define polygons or by using a drawing tool. Shapes of fusion objects could be exported to a comma-separated file. The simulator parses the textual representation of polygons and translates them to fusion objects to be processed by the artificial intelligence. The use of a drawing tool implies the use of predefined colors. The positions of static obstacles are computed by scanning the created image for special markers with

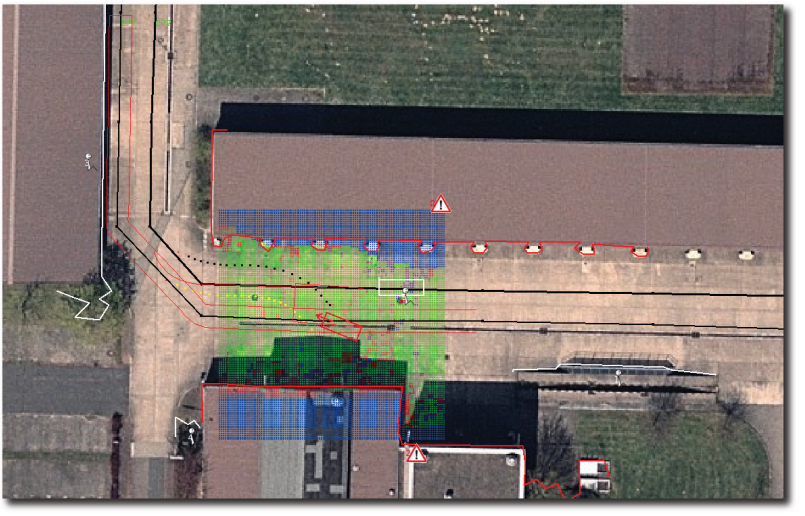


Fig. 46. Sreenshot with additional drivability data.

reference to a known coordinate. Fig. 45 depicts a screenshot of our visualisation application where the corresponding fusion objects are displayed.

For a more realistic simulation, the data fusion objects generated by the simulator could be created with different quality. This is used to simulate sensor noise and GPS drifts and makes fusion objects suddenly disappear or moves them by a tiny offset away from their original location. The sensor visibility range could be specified to affect the range of fusion objects that will be transmitted to the artificial intelligence.

Adding moderate drivability data completes this test run. This could be accomplished by passing an image file to the simulator, which specifies the required information through different colors. Fig. 46 shows the result. The visualisation of drivability grid displays drivable terrain in green, undrivable terrain in red and unknown terrain with blue cells.

6 The Race and Discussion

6.1 National Qualification Event

The National Qualification Event took place from October 26 to October 31 on the former George Airforce Base in Victorville, California as depicted in Fig. 2. The entire area was divided into three major parts named ”Area A”, ”Area B” and ”Area C” as shown in Fig. 47. First of all, Caroline had to demonstrate the proper function of her safety system to participate in the National Qualification Event. As expected Caroline stopped within the necessary range using the E-stop remote controller as well as the emergency stop buttons mounted on her roof.

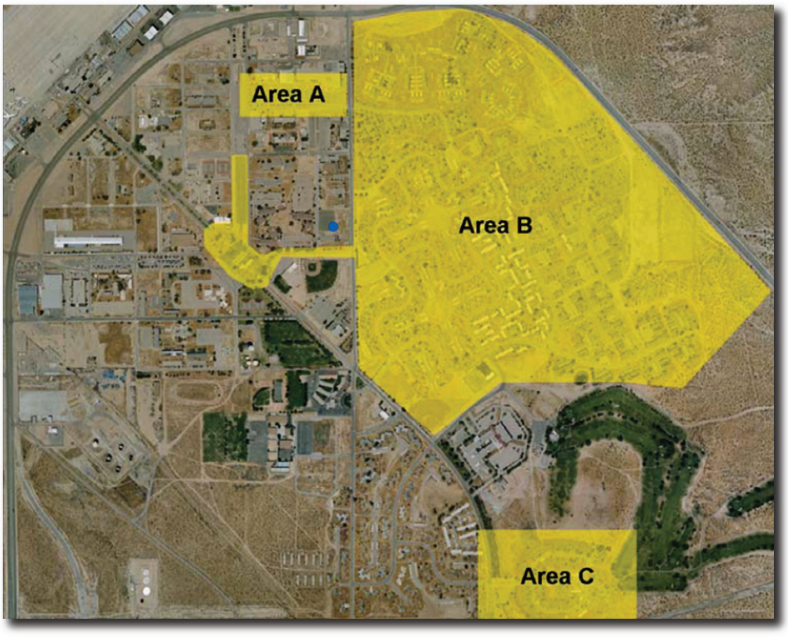


Fig. 47. Layout of the former George Airforce Base for the National Qualification Event. The blue dot indicates the pit area for our team.

6.1.1 Area A

For our team, the National Qualification Event started in "Area A". The main task for Caroline in that part was to merge into and through moving traffic. Therefore, several other vehicles controlled by human drivers drove within predefined speed limits to ensure the 10 seconds time slots as demanded by the DARPA's requirements. Fig. 48 shows the layout of the track. Caroline was placed at checkpoint 2. She had to drive downward to the T-junction, wait for an appropriate time slot and then turn left through the moving traffic. Afterwards, she had to pass checkpoint 1 by following other vehicles and drive to the upper junction. After waiting for an appropriate time slot, she had to turn into the street to pass checkpoint 2 again. The goal was to drive as many rounds as possible in this area.

Compared to other competitors, Caroline had to pass this task several times. The first run in this part let Caroline drive into the opposite lane. Analyzing this obviously strange behavior afterwards using our simulator as depicted in Fig. 49, we figured out that the barriers shown by white lines around the course narrowed the proper lane. Therefore, Caroline, shown as a red rectangle driving downwards to the lower T-junction, interpreted them as stationary obstacles in her way which she tried to overtake which can be seen in the computed trajectory shown by yellow and black pearls that leads into the opposite lane.

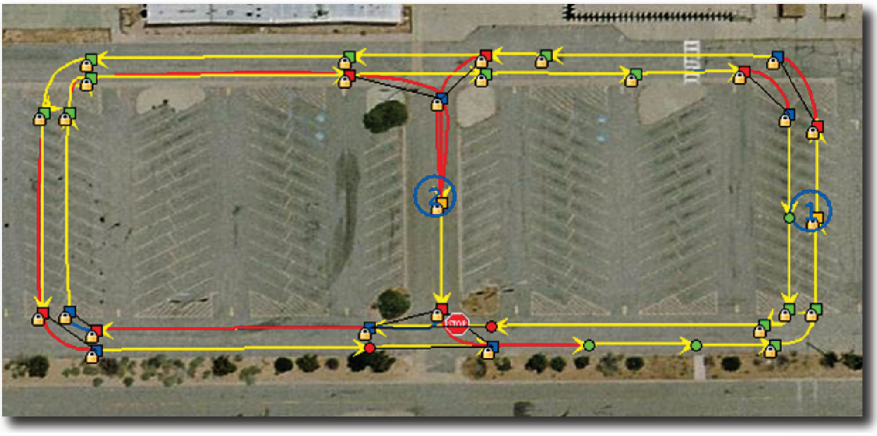


Fig. 48. Layout of "Area A".

After modifying several parameters, we had our second try in "Area A". She drove five rounds, merged into moving traffic correctly, waited at stop lines and followed other vehicles very well. Unfortunately, some problems occurred on the above right corner, when Caroline decided to turn right instead of following the road to the junction. We found out, that Caroline got in trouble with the street surface in that corner. There was a mixture of concrete and tar each with different colors. Thus, Caroline educated that color difference and tried to drive towards areas with a similar surface.

After modifying that behavior, we got another try in that course. Caroline started a perfect first run but waited too long for the second one. Therefore,

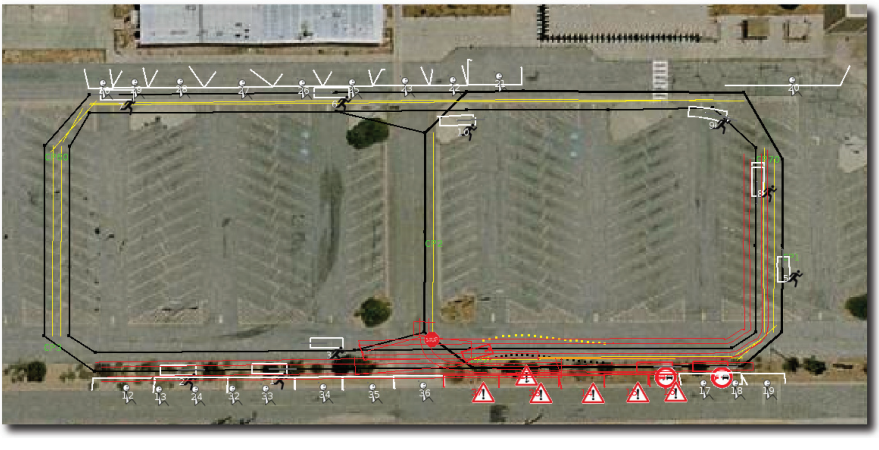


Fig. 49. Analysis of Caroline's behavior in "Area A".

the judges paused our vehicle and demanded a more progressive behavior of Caroline. Tuning again some parameters, we tried the course a fourth time short time later. This time, Caroline drove very swiftly but she did not give way to oncoming traffic. So, we changed the parameters again to get a safer behavior again and convinced the judges in our last try in that area of Caroline's abilities to merge correctly into moving traffic after demonstrating approximately eigth perfect rounds.

6.1.2 Area B

After encountering difficulties in the first task, we were unsure how Caroline would perform in "Area B" since several teams already failed to complete this part. The entire course is shown in Fig. 50. The main task was to overtake stationary obstacles, handle free navigation zones without any lane markings and to park safely inside those zones between other vehicles. The course itself could not be seen completely, so Caroline had to drive for herself without any observation by our team. We only could hear her progress by the team radio and by her siren.

Caroline started within a concrete start chute laid inside a free navigation zone. Many other teams already failed to leave this zone into the traffic circle correctly. She entered smoothly the traffic circle, left the circle and turned into the part on the right hand side of Fig. 50. In the center of the lower circle

Fig. 50. Layout of "Area B".

she had to park between other vehicles. The entry to that zone was very rough and several other teams already damaged the tires of their vehicle. We analyzed the video right after the task and remarked heavy vibration of the camera's picture but she entered the zone smoothly. After finishing the parking she left the zone to proceed the course.

Furthermore, Caroline had to deal with a gate located right at the exit of the upper circle. Due to our sensor layout she had to attempt several times to find the right way for leaving that circle. Returning to the start chutes again, she honked twice to indicate the completion of her mission after passing the last checkpoint. With this successful run, Caroline was one of only three vehicles to accomplish this course completely and in time.

6.1.3 Area C

On the same day, Caroline was faced with "Area C". This area is shown in Fig. 51. The main task was to handle intersections correctly and deal with blocked roads.

Caroline started near checkpoint 30 in the upper left corner on the outer lane. She handled both intersections on the left hand side and the right hand side several times correctly with every combination of other vehicles she was faced. Right in front of checkpoint 30 in the center part of this course, Caroline encountered a road blockage as shown in Fig. 52. We were unsure wether Caroline would detect the barrier since it had no contact to the ground and our sensors could look right through that barrier.

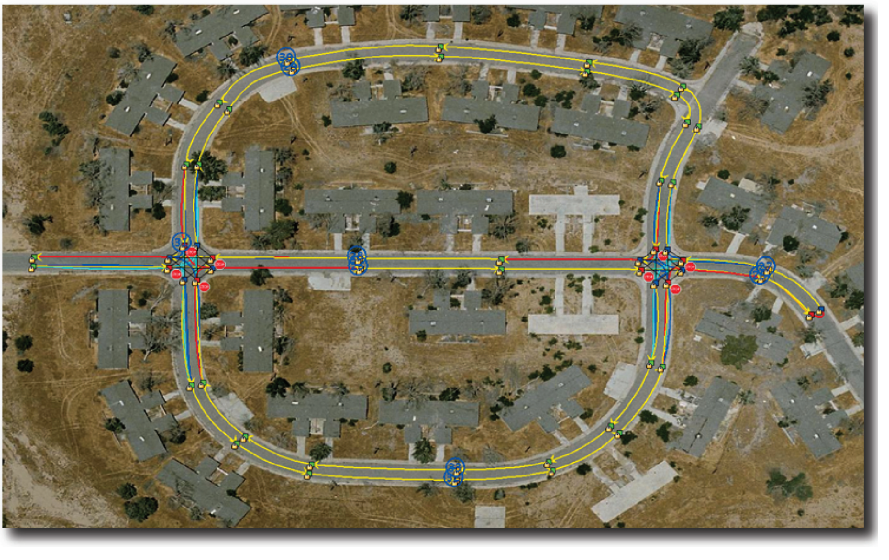


Fig. 51. Layout of "Area C".

Fig. 52. Blocked round in "Area C" by a barrier.

But Caroline detected that barrier properly and initiated the U-turn to choose another route the checkpoint. Afterwards, she passed all further traffic and intersection situations correctly and finished "Area C" finally. With all results achieved in the three areas, Caroline qualified early as a newcomer for the final event besides the well-established team with their experience of the Grand Challenges.

6.2 Mandatory Practice for DARPA Urban Challenge Final Event

The day before the DARPA Urban Challenge Final Event was scheduled, everyone of the eleven finalists had to participate in a practice session. By using this session, DARPA would ensure that every vehicle was able to leave the start chute and turn into the traffic circle. Assuming that this would be an easy task, we put Caroline into autonomous mode and waited for her to begin her run. But she did not leave her start chute and our team failed that practice session. We figured out a problem by parsing the RNDF provided by the DARPA. This issue did not let Caroline understand the road network for the final. After fixing this problem, we got another try. But Caroline still did not leave her start chute. Thus, DARPA placed us in the last of the eleven start chutes and cancelled the practice for our team.

Later analyzing the data we figured out the jitter in the GPS signal while significantly waiting for the "RUN" mode that yielded leaving the calculated

Fig. 53. Layout for the DARPA Urban Challenge Final Event.

trajectory. After fixing this issue we finally prepared Caroline for the DARPA Urban Callenge Final Event on the following day.

6.3 DARPA Urban Challenge Final Event

Figure 53 shows the enlarged "Area B" track for the DARPA Urban Challenge Final Event, including the former "Area A" as a parking lot. The start chutes were the same as for the run in "Area B". Additionally, in the lower-right corner of the map, there was a sandy off-road track located yielding a two-lane road return the inner part of the DARPA Urban Challenge Final Event area.

On November 3, 2007 at 6:53 am PST we loaded the first of three mission files into Caroline and set her into "PAUSE" mode. She calculated the route for the first checkpoint and started her run at 7:27 am PST. Fig. 54 shows the first part of her way during the first mission.

The asterisk in Fig. 54 indicates the location where two members of our team had to accompany the DARPA judges. Caroline had passed approximately 2.5 kilometers until she was paused by the DARPA control vehicle right behind her. Fig. 55 shows the reason for "PAUSE" mode.

Caroline got stuck after she turned into the berms. Fig. 55 (a) and (b) shows Caroline approaching a traffic jam right in front of her. Obviously, she tried to pass the stopped vehicle by interpreting it as a stationary obstacle using the clearance next the last car. The result of this attempt is shown in Fig. 55 (c): Caroline got stuck and could not get free without human intervention.

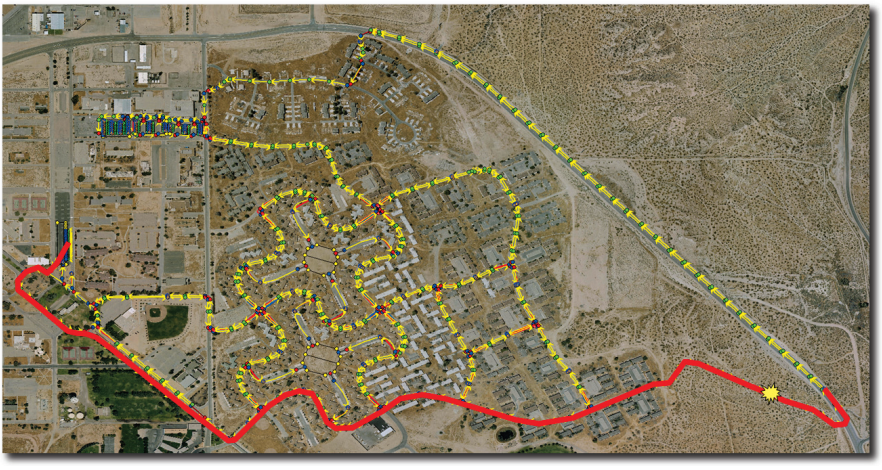


Fig. 54. Passed way before the first problem.

After she got freed and set in "RUN" mode again right at the beginning of the two-lane road, she continued her route and passed several checkpoints. The next incident was after 11.4 kilometers shown as the asterisk in Fig. 57.

At that location Caroline did not yield right of way to Talos, the autonomous vehicle from team MIT. Therefore, the DARPA paused both vehicles and let team members from MIT come to that location. After replacing Talos, both vehicles were sequentially set to "RUN" mode and passed safely each other. Unfortunately, the reason for not yielding right of way to Talos could not be figured out analyzing our log files. Since the situation was a left turn through oncoming traffic, it could be a problem detecting and tracking Talos due to problems either with our front sensors or with the interpretation in the artificial intelligence.

As shown in Fig. 58, Caroline continued her route. Additionally, she parked in the parking lot shown in the upper left picture of Fig. 58. After the parking

(a) Blocked road in front of Caroline.

(b) Caroline is approaching the traffic jam.

(c) Caroline got stuck after turning left into the berms.

Fig. 55. Caroline got stuck after 2.5 kilometers.

Fig. 56. Caroline went on after she got stuck.

maneuver, she returned the second time to the traffic circle and continued her mission 1.

At approximately 9:55 am PST, again two team members from team CarOLO were driven to Caroline, who met Talos from team MIT for the second time in a free navigation zone. This incident is shown as an asterisk in Fig. 59.

Our team members were faced with a twisted carrier rod of the Ibeo laser scanners due to a collision with Talos from team MIT as shown in Fig. 60. Until today it is still unresolved which car was in charge of the accident. Caroline interpreted the situation as described in the technical evaluation criteria [DARPA, 2006] by the section "Obstacle field". Therefore, Caroline tried to pass the oncoming Talos by pulling to the right side. Unfortunately, further interpretation is impossible due to missing detailed log files of that situation. Finally, DARPA retired Caroline as the fourth and last vehicle from the DARPA Urban Challenge Final Event.

Altogether, Caroline drove 16.4 kilometers in total and was retired from the race at 10:05 am PST. At 8:03 am PST, the watchdog module reset the SICK laser scanners mounted on the roof due to communication problems. At

Fig. 57. Next incident including Caroline and Talos from team MIT.

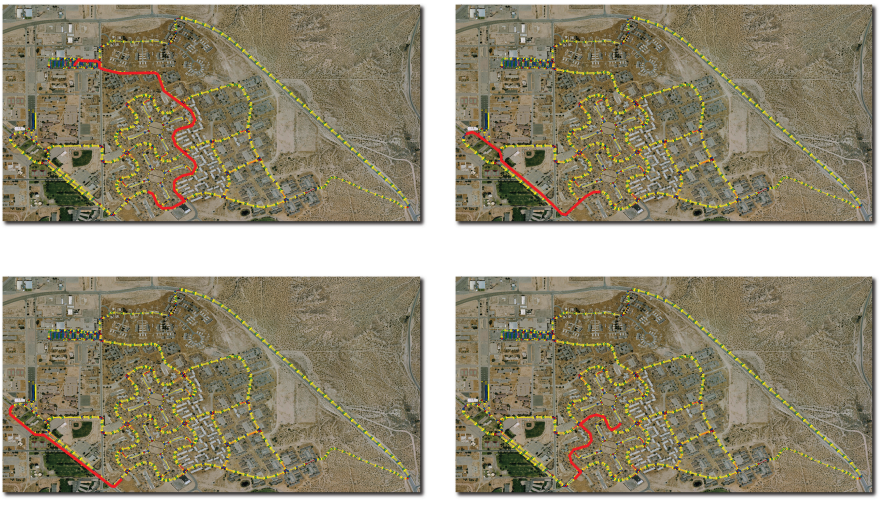
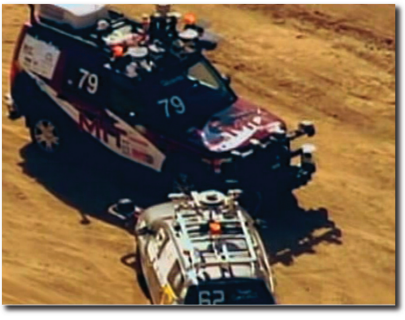


Fig. 58. Caroline went on after not yielding right of way to Talos.

approximately 9:00 am PST, the watchdog missed heartbeats from the IMU, and therefore triggered a reset. Right after the collision with Talos from team MIT, the watchdog observed communication problems with the laser scanners mounted in the front of Caroline. After a reset, the communication was re-established. During the race, computer "Daq1" as shown in Fig. 4 froze two times and had to be reset.

Fig. 59. Passed way before the first problem.



(a) Collision between Caroline and Talos from MIT.

(b) The red circle shows the twisted carrier rod.

Fig. 60. Caroline was retired after the collision with MIT.

7 Conclusion

Team CarOLO is an interdisciplinary team made up of members from the faculties of computer science and mechanical and electrical engineering which is significantly supported by industrial sponsors. Our vehicle Caroline is a standard 2006 Volkswagen Passat station wagon built to European specifications that is able to detect and track stationary and dynamic obstacles at a distance of up to 200 meters. The system's architecture comprises eight main modules: Sensor Data Acquisition, Sensor Data Fusion, Image Processing, Digital Map, Artificial Intelligence, Vehicle Path Planning and Low Level Control, Supervisory Watchdog and Online-Diagnosis, Telemetry and Data Storage

for Offline Analysis. The signal flow through these modules is generally linear in order to decouple the development process. Our design approach uses multi-sensor fusion of lidar, radar and laser scanners, extending the classical point shape based approach to handle extensive dynamic targets expected in urban environments. Image processing detects lane markings along with drivable areas. Artificial intelligence is modeled according to DAMN architecture, redesigned and enhanced to meet requirements of special behavior in urban environments. Our approach is able to handle complex situations and ensure Caroline's proper behavior, e.g. obeying traffic regulations at intersections or performing U-turns when roads are blocked. Decisions of the artificial intelligence are sent to the path planner, which calculates optimal vehicle trajectories with respect to its dynamics in real time. Safety and robustness is ensured by supervisory watchdog monitoring of all vehicle's hardware and software modules. Failures or malfunctions immediately result in a safe and complete stop by Caroline. Since we are a large heterogeneous team with a very tight project schedule, we recognized very early the need for efficient quality assurance during the development process. Thus, we implemented an automatic multi-level test process. Each new feature or modification runs through a series of unit tests or comprehensive simulations before being deployed on the vehicle.

As a competitor in the DARPA Urban Challenge Final Event, Caroline is able to autonomously perform missions in urban environments. She drove approximately 17 kilometers in about three hours in the final.

Acknowledgments

The authors thank their colleagues, students and professors from five institutes of the Technische Universität Braunschweig, who have developed Caroline. As a large amount of effort and resources were necessary to attempt this project, it would not have been successful if not for the many people from the university and local industry that had sponsored material, manpower and financial assistance. Particular thanks go to Volkswagen AG, IAV GmbH and the Ministry of Science and Culture of Lower Saxony. The authors' team also greatly thanks Dr. Bartels, Dr. Hoffmann, Professor Hesselbach, Mr. Horch, Mr. Lange, Professor Leohold, Dr. Lienkamp, Mr. Kuser, Professor Seiffert, Mr. Spichalsky, Professor Varchmin, Professor Wand and Mr. Wehner for their help on various occasions.

References

Basarke et al., 2007a. Basarke, C., Berger, C., Homeier, K., Rumpe, B.: Design and quality assurance of intelligent vehicle functions in the "virtual vehicle". Virtual Vehicle Creation (2007a)

Basarke et al., 2007b. Basarke, C., Berger, C., Rumpe, B.: Software & systems engineering process and tools for the development of autonomous driving intelligence. Journal of Aerospace Computing, Information, and Communication 4 (2007b)

Beck, 2005. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (2005)

Beedle and Schwaber, 2002. Beedle, M., Schwaber, K.: Agile Software Development with Scrum. Prentice-Hall, Englewood Cliffs (2002)

Bilmes, 1997. Bilmes, J.: A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report (1997)

Collins-Sussmann et al., 2004. Collins-Sussmann, B., Fitzpatrick, B.W., Pilato, C.M.: Version Control with Subversion. O'Reilly, Sebastopol (2004)

Cormen et al., 2002. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. (2002)

DARPA, 2006. DARPA, Technical evaluation criteria (2006)

Duda and Hart, 1973. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. John Wiley & Sons Inc., Chichester (1973)

Edgewall Software, 2007. Edgewall Software, Trac. Edgewall Software (2007)

Gary Bradski, 2005. Bradski, G., Adrian Kaehler, V.P.: Learning-based computer vision with intels open source computer vision library, pp. 126–139 (2005)

Heikkil and Silvn, 1996. Heikkil, J., Silvn, O.: Calibration procedure for short focal length off-the-shelf ccd cameras. In: 13th International Conference on Pattern Recognition, Vienna, Austria, pp. 166–170 (1996)

Kalman, 1960. Kalman, R.E.: A new approach to linear filtering and prediction problems. In: Transactions of the ASME-Journal of Basic Engineering, pp. 35–45 (1960)

Liggesmeyer, 2002. Liggesmeyer, P.: Software-Qualitaet: Testen, Analysieren und Verifizieren von Software. Spektrum, Akad. Verl. (2002)

Nethercote and Seward, 2003. Nethercote, N., Seward, J.: Valgrind: A program supervising framework. Theoretical Computer Science 89 (2003)

OpenCV Website, 2007. OpenCV Website, The open cv library (2007)

Pitteway and M.L.V., 1967. Pitteway, M.L.V.: Algorithmn for drawing ellipses or hyperbolae with a digital plotter. Computer Journal 10(3), 282–289 (1967)

Rosenblatt, 1997. Rosenblatt, J.: DAMN: A Distributed Architecture for Mobile Navigation. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (1997)

Shafer, 1976. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton (1976)

Shafer, 1990. Shafer, G.: Perspectives on the theory and practice of belief functions. International Journal of Approximate Reasoning (3), 1–40 (1990)

Thrun et al., 2006. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Winning the darpa grand challenge. Journal of Field Robotics (2006)

Ulrich and Nourbakhsh, 2000. Ulrich, I., Nourbakhsh, I.: Appearance-based obstacle detection with monocular color vision. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Austin, TX, pp. 866–871 (2000)

The MIT – Cornell Collision and Why It Happened

Luke Fletcher[1], Seth Teller[1], Edwin Olson[1], David Moore[1], Yoshiaki Kuwata[1], Jonathan How[1], John Leonard[1], Isaac Miller[2], Mark Campbell[2], Dan Huttenlocher[2], Aaron Nathan[2], and Frank-Robert Kline[2]

[1] Team MIT
Massachusetts Institute of Technology
Cambridge, MA 02139
lukesf@mit.edu

[2] Team Cornell
Cornell University
Ithaca, NY 14853
itm2@cornell.edu

Abstract. Mid-way through the 2007 DARPA Urban Challenge, MIT's robot 'Talos' and Team Cornell's robot 'Skynet' collided in a low-speed accident. This accident was one of the first collisions between full-sized autonomous road vehicles. Fortunately, both vehicles went on to finish the race and the collision was thoroughly documented in the vehicle logs. This collaborative study between MIT and Cornell traces the confluence of events that preceded the collision and examines its root causes. A summary of robot–robot interactions during the race is presented. The logs from both vehicles are used to show the gulf between robot and human-driver behavior at close vehicle proximities. Contributing factors are shown to be: (1) difficulties in sensor data association leading an inability to detect slow-moving vehicles and phantom obstacles, (2) failure to anticipate vehicle intent, and (3) an over-emphasis on lane constraints versus vehicle proximity in motion planning. Finally, we discuss approaches that could address these issues in future systems, such as inter-vehicle communication, vehicle detection and prioritized motion planning.

1 Introduction

On November 3rd, 2007, the Defense Advanced Research Projects Agency (DARPA) Urban Challenge Event (UCE) was held in Victorville, California. For the first time, eleven full-size autonomous vehicles interacted with each other and other human-driven vehicles on a closed course. The aim of the contest was to test the vehicles' ability to drive between checkpoints while obeying the California traffic code. This required exhibiting behaviors including lane keeping, intersection precedence, queuing, parking, merging and passing.

On the whole, the robots drove predictably and safely through the urban road network. None of the robots stressed the (understandably) conservative safety measures taken by DARPA. There were, however, a number of low-speed incidents during the challenge. This paper reviews those incidents and takes an in-depth look at one of them, the collision between Team Cornell's vehicle 'Skynet' and MIT's 'Talos'. This paper scrutinizes why the collision occurred and attempts to draw some lessons applicable to the future development of autonomous vehicles.

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 509–548.
springerlink.com

Fig. 1. The collision. *(left): Skynet*. *(right): Talos*. This paper explores the factors which led to the collision. Despite the mishap, both vehicles when on to complete the race.

The UCE was held on a closed course within the decommissioned George Air-force base. The course was predominantly the street network of the residential zone of the former base. Several graded dirt roads added for the competition. The contest was cast as a race against time to complete 3 missions. The missions were different for each team but were designed to require each team to drive 60 miles to finish the race. Penalties for erroneous or dangerous behavior were converted into time penalties. DARPA provided all teams with a single Route Network Definition File (RNDF) 24 hours before the race. The RNDF is very similar to a digital street map used by an in-car GPS navigation system. The file defined the road positions, number of lanes, intersections, and even parking-space locations in GPS coordinates. A plot of the route network for the race is shown in Figure 2. On the day of the race, each team was provided with a second unique file called a Mission Definition File (MDF). This file consisted solely of list of checkpoints within the RNDF which the vehicle was required to cross.

To mark progress through each mission, DARPA arranged the checkpoints in the mission files to require the autonomous vehicle to return to complete a lap of the oval shaped "Main Circuit" (visible in bottom left corner of Figure 2) at the end of each sub-mission. Each mission was subdivided into 6 or 7 'sub-missions'. The vehicles returned to the finishing area at the end of each mission so the team could recover and reposition the vehicle for the next mission. Most roads were paved with a single lane in each direction, similar to an urban road. Several roads had two lanes of traffic in each direction, like an arterial road or highway. One road, in the southeastern corner of the network, was a raised dirt road constructed especially for the event.

All 11 qualifying robots were allowed to interact in the UCE course simultaneously. Additional traffic was supplied by human-driven Ford Tauruses. To prevent serious crashes during the competition, all autonomous vehicles were followed by an assigned DARPA chase vehicle. The chase-vehicle driver supervised the robot and could 'Pause' or, in extreme cases, 'Disable' the robot via radio link. 'Paused'

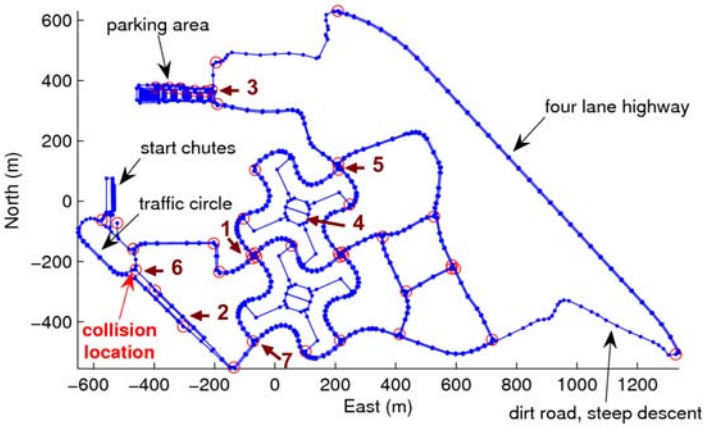


Fig. 2. The UCE road network. Waypoints are designated as blue dots. Traversable lanes and zone boundaries are represented as blue lines. Stop lines are designated as red circles. The *Skynet– Talos* collision happened entering the Main Circuit on the bottom left.

robots could then be 'Un-Paused' to continue a mission when safe. 'Disabling' a vehicle would kill the engine, requiring the vehicle's team to recover it.

The qualifiers and the race provided ample opportunity for damage to the robots on parked cars, concrete barriers, DARPA traffic vehicles and buildings. The fact that the two vehicles were not damaged, other than minor scrapes in the collision, despite hours of driving emphasizes the fact that the circumstances leading to the collision were the product of confounding assumptions across the two vehicle architectures. The robots negotiated many similarly complex situations successfully.

This paper begins with a brief summary in Section 2 of the robot–robot interactions during the 6-hour race. Then, to aid in the collision analysis, summaries of the MIT and Cornell vehicle software architectures are given in Sections 3 and 4 respectively. Section 5 describes the *Skynet–Talos* collision in detail, before branching in Sections 6 and 7 to provide detailed accounts of the robots' software state during the incident. The apparent causes of the incidents are studied here to shed light on the deeper design issues involved. In Section 8, we draw together the insights from the software architecture analysis to summarize the common themes, the lessons learned, and the impediments to using these robots on the real urban roads.

2 Chronology of Robot–Robot Interactions

The following table is a list of robot–robot collisions or close calls during the UCE. The list has been compiled from the race day web-cast and vehicle data logs. The locations of the incidents are marked in Figure 2.

Time (Approx)	Location	Description	Reference
1h00m	Utah and Washington	Cornell's *Skynet* passing with IVS' *XAV-250* and Ben Franklin Racing Team's *Ben* oncoming	Section 2.1
1h30m	George Boulevard	*Ben* and Team UCF's *Knight Rider*	Section 2.2
2h00m	North Nevada and Red Zone	CarOLO's *Caroline* turns across MIT's *Talos*	Section 2.3
3h00m	White Zone	*Caroline* and *Talos* collide.	Section 2.4
4h00m	Carolina Avenue and Texas Avenue	*Talos* swerves to avoid Victor Tango's *Odin*	Section 2.5
4h30m	George Boulevard and Main Circuit	*Skynet* and *Talos* collide	Section 5
5h20m	Utah and Montana	*Talos* turns across *Ben*	Section 2.6

We invited teams with vehicles actively involved the incidents (CarOLO, IVS and Ben Franklin Racing Team) to co-author or comment on the interactions. Received comments are included in the incident descriptions.

A full discussion of the *Skynet– Talos* collision is given Section 5.

Diagrams have been drawn describing each incident. In the drawings, a solid line shows the path of the vehicle, and a dashed line shows the intended/future path of the vehicle. A lateral line across the path indicates that the vehicle came to a stop in this location. DARPA vehicles are driven by DARPA personnel in the roles of either traffic or chase vehicles.

Videos of the log visualization for incidents involving *Talos* can be found in Section 9).

2.1 *Skynet* Passing with *XAV-250* and *Ben* Oncoming at Utah and Washington

The first near-miss occurred at the intersection of Utah and Washington. *Knight Rider* was at the intersection. *Skynet* pulled up behind a traffic vehicle, which was queued behind *Knight Rider's* chase vehicle (the chase vehicle was queued behind *Knight Rider*). The relative positions of the vehicles are shown in Figure 3(b). *Knight Rider* was making no apparent progress through the intersection, so after waiting, *Skynet* elected to pass. *Skynet* was behind three cars, which put it beyond the safety zone in which passing was prohibited. (DARPA, 2007). The rules also stated that vehicles should enter a traffic-jam mode after a prolonged lack of progress at

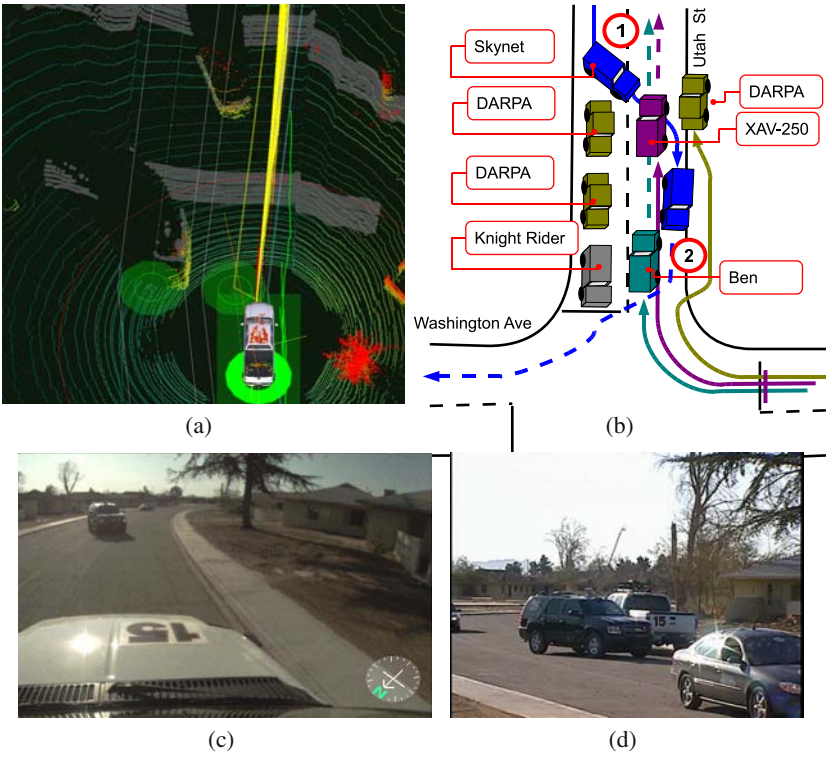
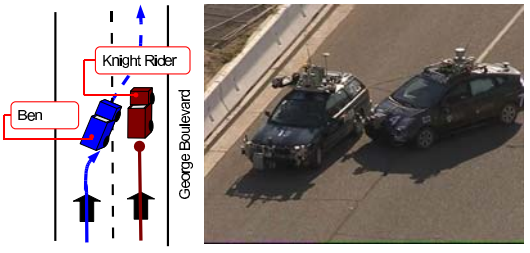


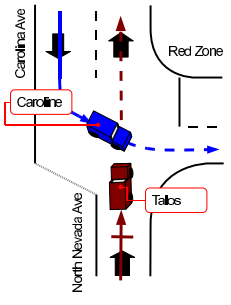
Fig. 3. After queuing behind the stationary *Knight Rider*, *Skynet* passes. (a) Visualization of *XAV-250* log when approaching *Skynet* (Image courtesy of Team IVS). (b) Diagram of incident. *(1):* Vehicle positions when *XAV-250* approaches. *(2):* Vehicle positions when *Ben* approaches. (c) *XAV-250* Camera view (Image courtesy of Team IVS). (d) $Skynet_{(26)}$ once $XAV\text{-}250_{(15)}$ had passed.

an intersection. *Skynet* began to pass. Shortly into the maneuver, the Intelligent Vehicle Systems vehicle *XAV-250* turned right from Washington onto Utah and into the on-coming path of *Skynet*. *Skynet* and *XAV-250* were Paused. *XAV-250* was Un-paused and permitted to drive past *Skynet*, clearing the area. *Skynet* was then also permitted to continue. *Skynet* determined that it could not get back into the correct lane and was too near the intersection, so it pulled over to the curb side of the lane and waited. Next *Ben* also turned onto Utah from Washington, and again, was on-coming to *Skynet*. *Skynet* and *Ben* were Paused . *Ben* was Un-paused and permitted to drive past. Interestingly, *Ben*'s chase vehicle drove onto the curb around to the right to pass the *Skynet* vehicle. This provides an example of the assessment made by a human driver in this scenario. Faced with *Skynet* in the on-coming lane, the chase vehicle driver elected to drive far right onto the curb to accommodate the potential behavior of the *Skynet* vehicle. The passage of *Ben* shows that mounting

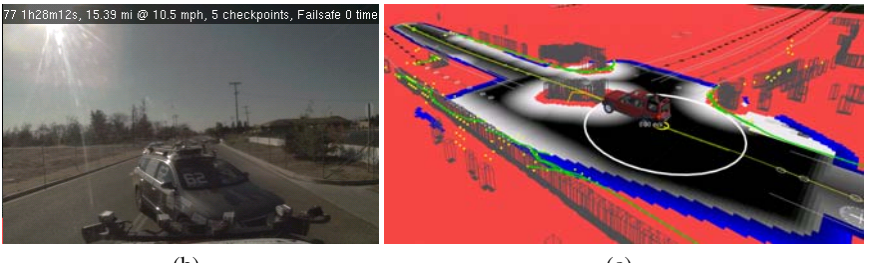


(a) (b)

Fig. 4. (a) Diagram of incident. (b) *Knight Rider*$_{(13)}$ and *Ben*$_{(74)}$ near miss.



(a)



(b) (c)

Fig. 5. (a) Diagram of incident. (b) *Talos'* view of the final pose *Caroline-Talos* turning near-miss. (c) Visualization from *Talos'* log.

the curb was not necessary to physically pass the vehicle. Given a clear intersection, the *Skynet* vehicle was able to negotiate the intersection and continue the mission. A more detailed account of this event from *Skynet*'s point of view is given in (Miller et al., 2008).

2.2 *Ben* and *Knight Rider* on George Boulevard

Figure 4 shows a near-miss featured in the webcast in which *Ben* appeared to be merging into *Knight Rider* on George Boulevard. In this case, *Ben* had been

following *Knight Rider*. *Ben* had been traveling faster than *Knight Rider* so DARPA decided to Pause *Knight Rider* once the vehicles were on George Boulevard, a dual lane road, to permit *Ben* to pass. With *Knight Rider* stopped in the right lane, DARPA expected *Ben* to continue in the left lane and then merge into the right lane after passing *Knight Rider*. At the far end of George Boulevard, the vehicles were required to be in the right lane. Because it involved a lane change at the start of a dual-lane road and the stopped vehicle was in the destination lane, the scenario was different from standard passing maneuvers and hence was not handled in *Ben's* software. Consequently *Ben* performed the lane change without accounting for *Knight Rider*. *Ben* was Paused and stopped in time to prevent a collision.

2.3 *Caroline* and *Talos* at North Nevada and Red Zone

Figure 5 shows the first of two close encounters between *Caroline* and *Talos*. The figure shows the diagram of the incident and how it appeared in the *Talos* software. In this incident *Talos* was driving straight down North Nevada. *Caroline* was approaching in the oncoming direction down Carolina Avenue, then turned left into the Red Zone across the path of *Talos*.

Talos detected the moving object of *Caroline* and found the closest intersection exit to project the assumed trajectory. *Talos'* intended motion plans were then

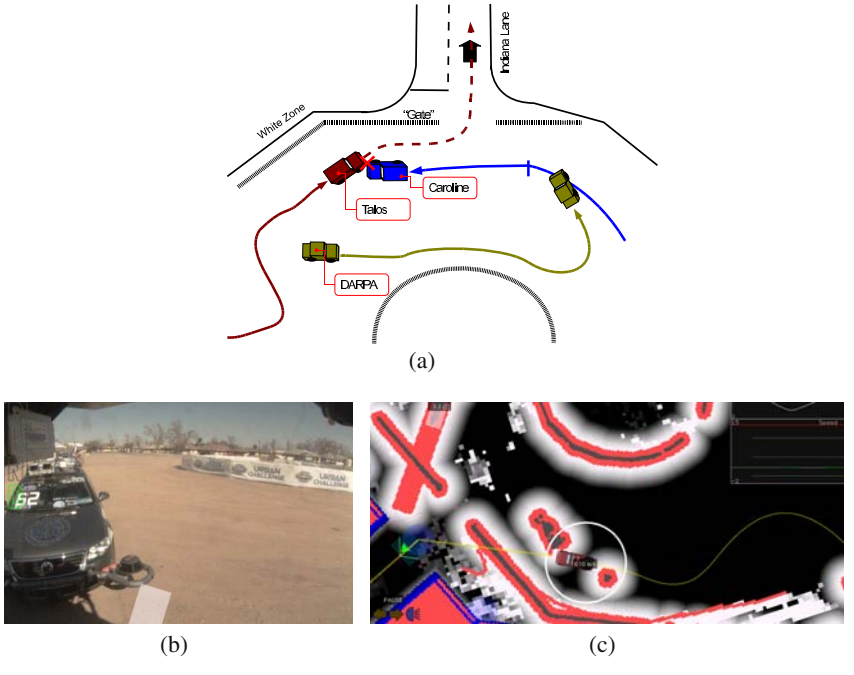
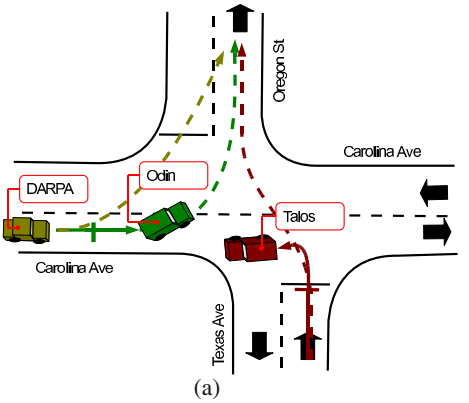
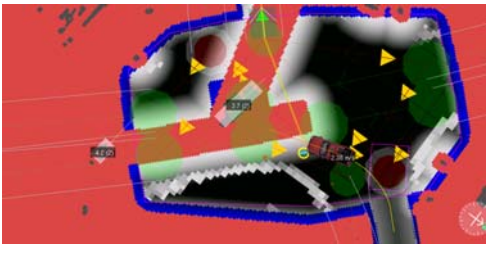


Fig. 6. (a) Diagram of incident. (b) Final pose of *Caroline* and *Talos* collision from *Talos'* front right camera. (c) Visualization from *Talos'* log.



(a)

(b)



(c)

Fig. 7. (a) Diagram of incident. (b) View from *Talos'* front camera. (c) *Talos'* log visualization. *Odin* is turns right. *Talos* brakes and turns hard left to avoid *Odin's* projected motion direction.

severed by *Caroline's* predicted trajectory, so the vehicle commenced an emergency stop. DARPA Paused both vehicles. A full account of this event from *Talos'* view is given in (Leonard et al., 2008).

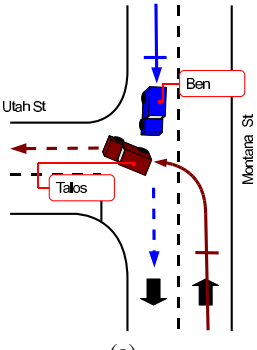
2.4 *Caroline* and *Talos* in White Zone

The second incident between *Caroline* and *Talos* ended in a collision. The *Caroline* vehicle was retired from the race shortly after this event. Figure 6 shows the diagram of the incident and collision between *Caroline* and *Talos* in the White Zone. *Caroline* was near the Indiana Lane exit of the White Zone. *Talos* entered the Kentucky Lane entrance to the White Zone and was en-route to the Indiana Lane exit. Initially, *Talos* planned a route around *Caroline's* chase vehicle to get to the Zone exit on the left. *Caroline's* chase vehicle then drove away from *Talos*. *Talos* then replanned a more direct route to the left, to the zone exit. As *Talos* drove toward the zone exit, *Talos* also approached *Caroline*. Initially *Caroline* was stationary, then *Caroline* drove slowly forward toward *Talos*. *Talos*, with the zone fence to the left and what it perceived as a static obstacle (which was actually *Caroline*) to the right, attempted to negotiate a path in between. *Caroline* advances toward *Talos*.

Talos keeps adjusting its planned path to drive around to the left of what appears to the *Talos* software as a stationary object. Just before the collision *Talos*' motion plans were severed, causing a "planner emergency stop". Due to *Talos'* momentum and *Caroline's* forward movement, the braking failed to prevent physical contact. DARPA then Paused the vehicles. A detailed account of this chain of events from *Talos'* view is given in (Leonard et al., 2008).

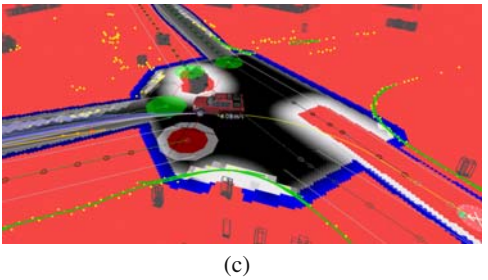
2.5 *Odin* and *Talos* at Carolina and Texas

This incident featured a close call negotiated by the robots without intervention from DARPA. Figure 7 shows the diagram and view from the *Talos* log. *Talos* arrived at a stop line at the intersection of Carolina and Texas. *Talos* was intending to go from Oregon to Texas (from bottom to top in Figure 7(a)). *Talos* yielded to *Odin* approaching. *Odin* arrived at the intersection intending to turn left into Texas Avenue. *Odin* came to a stop entering the intersection. *Talos* detected the time to contact for approaching vehicles had gone to infinity so proceeded across the intersection. *Odin* also proceeded from Carolina Avenue into Texas Avenue. *Odin*, much quicker off



(a)

(b)



(c)

Fig. 8. (a) Diagram of incident. (b) View from *Talos'* right front camera. *Talos'* view of the *Little Ben-Talos* turning near-miss. *Talos* yielded to the velocity track of oncoming $Ben_{(74)}$. *Ben* came to a stop at the intersection. *Talos* began motion. *Ben* began to go through the intersection. *Talos* saw *Ben* as a creeping "static obstacle" and continued. *Talos* completed the turn. *Ben* stopped. (c) *Talos'* log visualization. *Ben* approaching on the right of *Talos*.

the mark, was ahead of *Talos*. *Talos* reacted to *Odin* approaching by braking and replanning an evasive maneuver, turning hard to the left. *Odin* and *Odin's* chase vehicle completed the turn and cleared the intersection. *Talos* then resumed course down Texas Avenue behind the vehicles.

2.6 *Ben* and *Talos* at Utah and Montana

The final incident, a close call, is illustrated in Figure 8. Log data shows that *Talos* turned left from Montana onto Utah. *Talos* arrived at the intersection and yielded to oncoming traffic. *Ben* was approaching, so *Talos* remained stopped. At the intersection *Ben* also came to a stop. Again, *Talos* detected that the time to contact for approaching vehicles had now gone to infinity, so commenced the left-hand turn. As *Talos* crossed in front of *Ben*, *Ben* then also entered the intersection. At this point, *Ben* was quite far to the right of *Talos*, so *Talos'* forward path collision checking was not altered by the vehicle approaching to the side. *Talos* exited the intersection while *Ben* came to a stop. Once the intersection was clear, *Ben* continued the mission.

3 Team MIT's 'Talos'

This section is a summary of the *Talos* software architecture. The purpose of this section is to describe the vehicle software in sufficient detail to understand the vehicle behavior and contributing factors to the collision. A thorough description of the robot architecture is given in (Leonard et al., 2008).

Talos is a Land Rover LR3 fitted with cameras, radar and lidar sensors (shown in Figure 9). Forward, side and rear-facing cameras are used for lane marking detection.

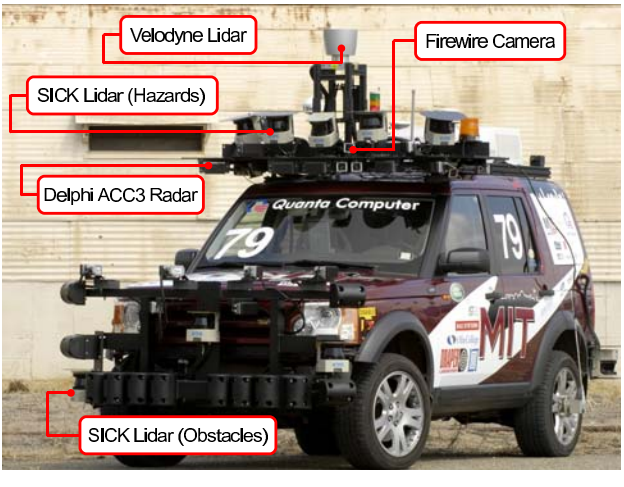


Fig. 9. MIT's 'Talos', a Land Rover LR3 featuring five Point Grey FireFly cameras, 15 Delphi ACC3 radars, 12 Sick LMS-291 lidars and a Velodyne HDL-64 lidar.

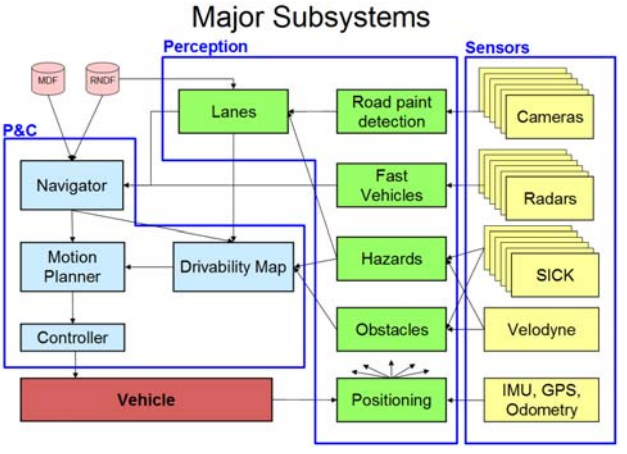


Fig. 10. MIT's *Talos* system architecture.

The Velodyne HDL-64 lidar is used for obstacle detection supplemented in the near-field with 7 horizontal Sick LMS-291 lidars. Five additional downward-facing Sick LMS-291 lidars are used for road-surface hazard detection including curb cuts. 15 Delphi ACC3 millimeter-wave radars are used to detect fast-approaching vehicles.

The system architecture developed for the vehicle is shown in Figure 10. All software modules run on a 40-core Quanta blade server. The general data flow of the system consists of raw sensor data processed by a set of perception software modules: the Position Estimator, Obstacle Detector, Hazard Detector, Fast [approaching] Vehicle Detector and Lane Tracker.

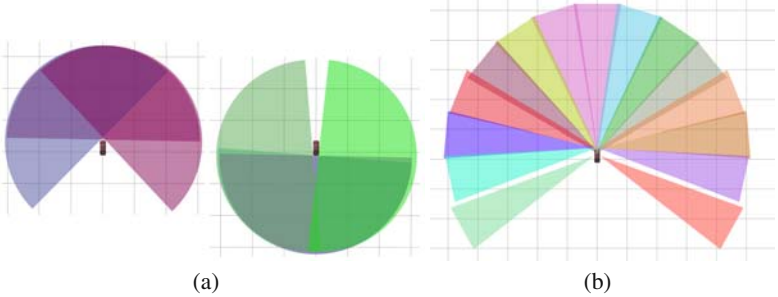
The Navigator process decomposes mission-level decisions into a series of short term ($1m - 60m$) motion goals and behavioral constraints. The output from the perception modules is combined with the behavioral constraints to generate a Drivability Map of the environment. The motion planning to the next short-term goal is done in the Motion Planner module with paths vetted against the Drivability Map. The trajectory created by the Motion Planner is executed by the Controller module. Each module is now discussed in detail.

During the Urban Challenge, the **Navigator** tracked the mission state and developed a high-level plan to accomplish the mission based on the map (RNDF) and the mission data (MDF). The primary output was the next short-term goal to provide to the Motion Planner. As progress was made the short-term goal was moved, like a carrot in front of a donkey, to achieve the mission. In designing this subsystem, the aim was to create a resilient planning architecture that ensured that the autonomous vehicle could respond reasonably and make progress under unforeseen conditions. To prevent stalled progress, a cascade of events was triggered by a prolonged lack of progress. For example, after 10 seconds of no progress queuing behind a stationary vehicle, the Navigator would trigger the passing mode if permitted by the DARPA rules. In this mode the lane center-line constraint was relaxed, permitting the vehicle

to pass. The Drivability Map would then carve out the current and oncoming lanes as drivable. After checking for oncoming traffic, the Navigator would then permit the vehicle to plan a passing trajectory around the stopped vehicle.

The **Obstacle Detector** used lidar to identify stationary and moving obstacles. Instead of attempting to classify obstacles as vehicles, the detector was designed to avoid vehicle classification using two abstract categories: "static obstacles" and moving obstacle "tracks". The output of the Obstacle Detector was a list of static obstacles, each with a location and size, as well as a list of moving obstacle "tracks", each containing position, size and an instantaneous velocity vector. The obstacle tracker integrated non-ground detections over relatively short periods of time in an accumulator. In our implementation, the tracker ran at 15Hz (matching the Velodyne frame rate). At each time step, the collection of accumulated returns were clustered into spatially nearby "chunks". These chunks were then matched against the set of chunks from the previous time step, producing velocity estimates. Over time, the velocity estimates were fused to provide better estimates. The tracking system was able to provide velocity estimates with very low latency, increasing the safety of the system. The reliable detection range (with no false negatives) was about 30m, with good detections out to about 60m (but with occasional false negatives). The system was tuned to minimize false positives.

For detecting vehicles, an initial implementation simply classified any object that was approximately the size of a car, as a car. In cluttered urban scenes this approach quickly led to many false positives. An alternative approach was developed based on the clustering and detection of moving objects in the scene. This approach was much more robust in cluttered environments, however one new issue arose. In particular circumstances, stationary objects could appear to be moving. This was due to the changing viewpoint of our vehicle combined with aperture/occlusion effects of objects in the scene. Due to this effect, a high velocity threshold (3.0m/s in our



(a) (b)

Fig. 11. Sensor fields of view on 20m grid. (a) Our vehicle used a total of seven horizontally mounted 180° planar lidars with overlapping fields of view. Front and rear lidars have been drawn separately to make the overlap more obvious. For ground plane rejection two lidars were required to "see" the same obstacle to register the detection (except in the very near field). (b) 15 radars with 18°-FOV each were fanned to yield a wide (255°) total field of view.

implementation) was used to reduce the frequency at which stationary objects were reported to be moving. Downstream software was written to accomodate that vehicles could appear as a collection of stationary objects or as moving obstacle tracks.

Vehicles were also detected using the radar-based **Fast-Vehicle Detector**. The narrow 18° field of view radars were fanned to provide 255° coverage in front of the vehicle. The raw radar detections were compensated for vehicle ego-motion then data association, and position tracking over time was used to distill the raw returns into a second set of obstacle "tracks". The instantaneous Doppler velocity measurement from the radar returns was particularly useful for detecting distant but fast-approaching vehicles. The information was used explicitly by the Navigator module to determine when it was safe to enter an intersection, or initiate merging and passing behaviors. Figure 11 shows the sensor coverage provided by Sick lidar and radar sensors.

The low-lying **Hazard Detector** used downward-looking planar lidars mounted on the roof to assess the drivability of the road ahead and to detect curb cuts. The module consisted of two parts: a hazard map, and a road-edge detector. The "hazard map" was designed to detect hazardous road surfaces by discontinuities in the lidar data that were too small to be detected by the Obstacle Detector. High values in the hazard map were rendered as high penalty areas in the Drivability Map. The road-edge detector looked for long strips of hazardous terrain in the hazard map. If strips of sufficiently long and straight hazardous terrain were detected, some poly lines were explicitly fitted to these regions and identified as a curb-cut or berm. These road edges were treated as obstacles: if no road paint was detected, the lane estimate would widen, and the road-edge obstacles (curbs) would guide the vehicle.

The **Lane tracker** reconciled RNDF data with lanes detected by vision and lidar. Two different road paint detectors were developed, each as a separate, stand-alone process. The first detector used a matched "Top Hat" filter scaled to the projected ground plane line width. Strong filter responses and the local gradient direction in the image were then used to fit a series of cubic Hermite splines. The second road-paint detector fitted lines to image contours bordering bright pixel regions. Both road-paint detectors produced sets of poly lines describing detected road paint in the local coordinate frame. A lane centerline estimator combined the curb and road paint detections to estimate the presence of nearby lanes. The lane centerline estimator didn't use the RNDF map to produce its estimates. It relied solely on detected features. The final stage of the lane tracking system produced the actual lane estimates by reconciling the RNDF data with the detected lane center lines. The map data was used to construct an *a-priori* estimate of the physical lanes of travel. The map estimates were then matched to the centerline estimates and a minimization problem was solved to snap the RNDF lanes to the detected lane centerlines.

The **Drivability Map** was constructed using perceptual data filtered by the current constraints specified by the Navigator. This module provided an efficient interface to perceptual data for motion planning. Queries from the Motion Planner about future routes were validated by the Drivability Map. The Drivability Map consisted of:

- "Infeasible regions" which were no-go areas due to proximity to obstacles or just undesirable locations (such as in the path of a moving vehicle or across an empty field when the road is traversable).
- "High-cost regions" which would be avoided if possible by the motion planning and
- "Restricted regions" which were regions that could only be entered if the vehicle were able to stop in an unrestricted area further ahead.

Restricted regions were used to permit minor violations of the lane boundaries if progress could be made down the road. Restricted regions were also used behind vehicles to enforce the requisite number of car lengths' stand-off distance behind a traffic vehicle. If there was enough room to pass a vehicle without crossing the lane boundary (for instance if the vehicle was parked on the side of a wide road), then *Talos* would traverse the Restricted region and pass the vehicle, continuing to the unrestricted region in front. If the traffic vehicle blocked the lane, then the vehicle could not enter the restricted region because there was no unrestricted place to stop. Instead, *Talos* would queue behind the restricted region until the traffic vehicle moved or a passing maneuver was commenced. No explicit vehicle detection was done. Instead, moving obstacles were rendered in the Drivability Map with an infeasible region projected in front of the moving obstacles in proportion to the instantaneous vehicle velocity. As shown in Figure 12(c), if the moving obstacle was in a lane the infeasible region was projected along the lane direction. If the moving obstacle was in a zone (where there was no obvious convention for the intended direction) the region was projected in the velocity direction only. In an intersection the obstacle velocity direction was compared with the intersection exits. If a good exit candidate was found, a second region was projected from the obstacle toward the exit waypoint as a prediction of the traffic vehicle's intended route (Shown in Figure 12(d)). The **Motion Planner** identified, then optimized, a kino-dynamically feasible vehicle trajectory that would move the robot toward the goal point. The module was based on the Rapidly exploring Random Tree (RRT) algorithm (Frazzoli et al., 2002), where the tree of trajectories was grown by sampling numerous configurations randomly. A sampling-based approach was chosen due to its suitability for planning in many different driving scenarios. Uncertainty in local situational awareness was handled through rapid replanning. By design, the motion planner contained a measure of safety as the leaves on the tree of potential trajectories were always stopping locations (Figure 12(a)). Shorter trees permitted lower top speeds as the vehicle had to come to a stop by the end of the trajectory. In this way, if for some reason the selected trajectory from the tree became infeasible, another branch of the tree could be selected to achieve a controlled stop. The tree of trajectories was grown towards the goal by adding branches that connected to the randomly sampled points. These were then checked for feasibility and performance. This module then sendt the current best vehicle trajectory, specified as an ordered list of waypoints (position, velocity, headings), to the low-level motion Controller at a rate of 10 Hz.

The **Controller** was a pure pursuit steering controller paired with a PID speed controller. It executed the low-level control necessary to track the desired path and velocity profile from the Motion Planner.

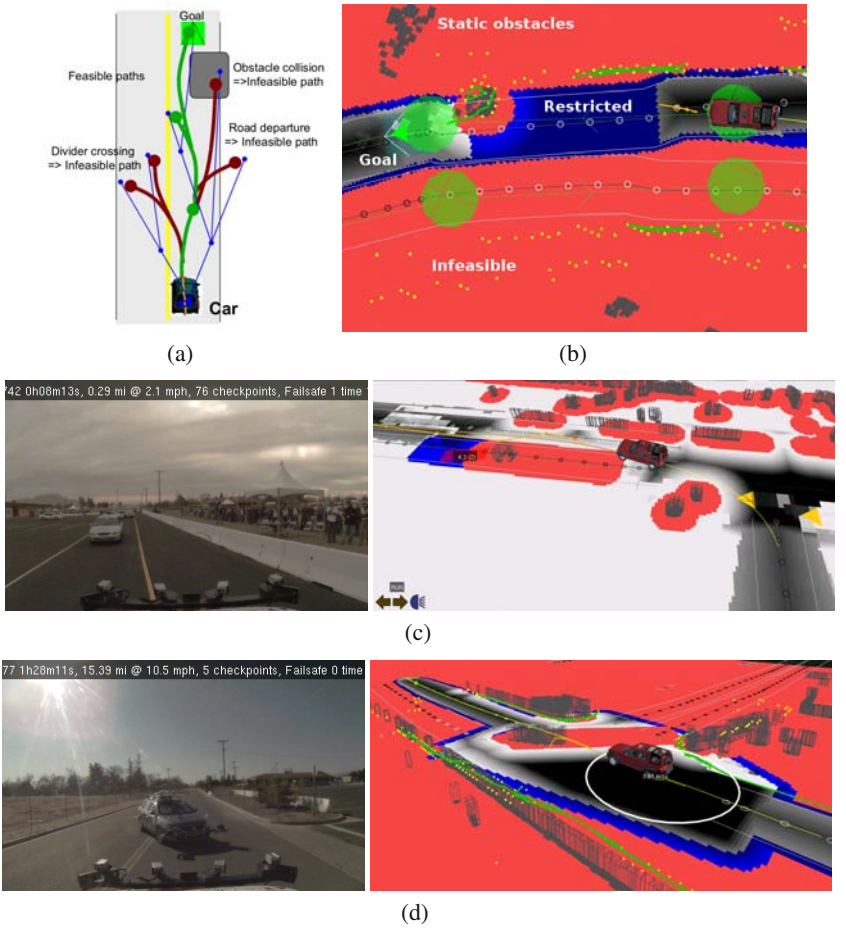


Fig. 12. (a) RRT Motion planning. Each leaf on the tree represented a stopping location. (b) Drivability map explanation. *White arrow on green background:* Short-term goal location. *Red:* Infeasible regions were off-limits to the vehicle. *Blue:* Restricted regions may only be entered if the vehicle could stop in an unrestricted region further on. *White or Gray:* High-cost regions accessible to the vehicle. Dark areas represented low-cost drivable regions. (c) An infeasible region was projected in the moving obstacle velocity direction down lane excluding maneuvers into oncoming vehicle. In this case lane constraints were rendered as [White] High cost instead of [Red] Infeasible due to a recovery mode triggered by the lack of progress through the intersection). (d) Within an intersection an infeasible region was created between a moving obstacle and the intersection exit matching the velocity direction.

4 Team Cornell's 'Skynet'

Team Cornell's 'Skynet,' shown in Figure 13, is an autonomous 2007 Chevrolet Tahoe. *Skynet* was built and developed at Cornell University, primarily by team members returning with experience from the 2005 DARPA Grand Challenge. The

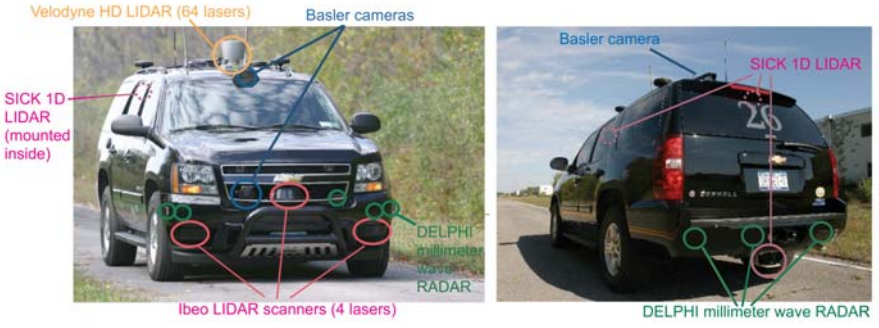


Fig. 13. Team Cornell's 'Skynet.'

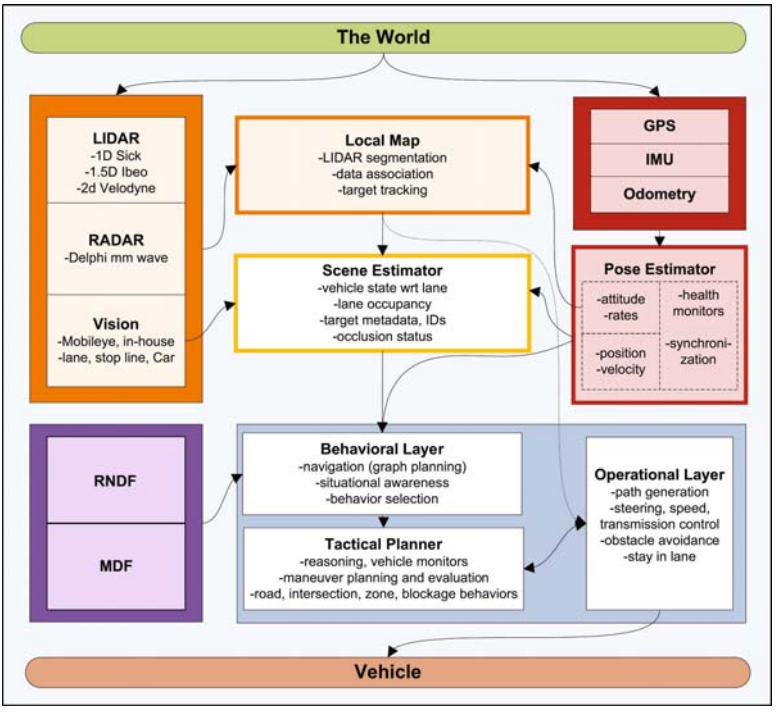


Fig. 14. System architecture of Team Cornell's *Skynet*.

team consisted of 12 core members supported by 9 part-time contributors. Experience levels included professors, doctoral and master's candidates, undergraduates, and Cornell alumni.

The high-level system architecture for Team Cornell's *Skynet* is shown in Figure 14 in the form of key system blocks and data flow. These blocks formed

the multi-layer perception and planning / control solution chosen by Team Cornell to successfully drive in an urban environment. General descriptions of each of these blocks are given below. Detailed descriptions of the obstacle detection and tracking algorithm and the intelligent planning algorithm, both root causes of *Skynet's* behavior during the Cornell – MIT collision, are given in sections 4.2 and 4.3.

4.1 General System Architecture

Skynet observed the world with two groups of sensors. *Skynet's* position, velocity, and attitude were sensed with raw measurements collected from Global Positioning System (GPS) receivers, an Inertial Measurement Unit (IMU), and wheel encoders. These raw measurements were fused in the **pose estimator**, an Extended Square Root Information Filter, to produce robust pose estimates in an Earth-fixed coordinate frame. *Skynet's* external environment, defined in the Urban Challenge as parked and moving cars, small and large static obstacles, and attributes of the road itself, was sensed using a combination of laser rangefinders, radar, and optical cameras.

Skynet used two levels of probabilistic data fusion to understand its external environment. The **Local Map** fused laser, radar, and optical data with *Skynet's* motion estimates to initialize, locate, and track static and dynamic obstacles over time. The **Scene Estimator** then used the local map's tracking estimates, pose estimates, and road cues from processed optical measurements to develop key statistics about *Skynet* and nearby obstacles. Two sets of statistics were generated: those concerning *Skynet*, including location with respect to the road and lane occupancy, and those concerning other obstacles, including position / velocity, an identification number, lane occupancy, car likeness, and whether each obstacle was currently occluded or not.

Planning over DARPA's Route Network Definition File (RNDF) and Mission Definition File (MDF) occurred in three layers. The topmost **Behavioral Layer** combined the RNDF and MDF with obstacle and position information from the Scene Estimator to reason about the environment and plan routes to achieve mission progress. The Behavioral Layer then selected which of four behaviors would best achieve the goal: road, intersection, zone, or blockage. The selected behavior was executed in the **Tactical Layer**, where maneuver-based reasoning and planning occurred. The **Operational Layer**, the lowest level of planning, produced a target path by adjusting an initial coarse path to respect speed, lane, obstacle, and physical vehicle constraints. *Skynet* drove the target path by converting it to a series of desired speeds and curvatures, which were tracked by feedback linearization controllers wrapped around *Skynet's* steering wheel, brake, transmission, and throttle actuators.

4.2 Obstacle Detection and Tracking

Team Cornell's obstacle detection and tracking system, called the **Local Map**, fused the output of all obstacle detection sensors into one vehicle-centric map of *Skynet's* environment. The Local Map fused information from three sensing modalities: laser

Table 1. *Skynet's* obstacle detection sensors

Sensor	Location	Type	Rate	FoV	Resolution
Ibeo ALASCA XT	front bumper left	laser	12.5 Hz	150°	1°
	front bumper center	laser	12.5 Hz	150°	1°
	front bumper right	laser	12.5 Hz	150°	1°
Sick LMS 291	left back door	laser	75 Hz	90°	0.5°
	right back door	laser	75 Hz	90°	0.5°
Sick LMS 220	back bumper center	laser	37.5 Hz	180°	1°
Velodyne HDL-64E	roof center	laser	15 Hz	360°	0.7°
Delphi FLR	front bumper left (2x)	radar	10 Hz	15°	20 tracks
	front bumper center	radar	10 Hz	15°	20 tracks
	front bumper right (2x)	radar	10 Hz	15°	20 tracks
	back bumper left	radar	10 Hz	15°	20 tracks
	back bumper center	radar	10 Hz	15°	20 tracks
	back bumper right	radar	10 Hz	15°	20 tracks
Unibrain Fire-i 520b	back roof center	optical	15 Hz	20° – 30°	N / A

rangefinders, radars, and optical cameras. Mounting positions are shown in Figure 13. Table 1 summarizes *Skynet's* obstacle detection sensors, and Figure 15 gives a top-down view of *Skynet's* sensor coverage. All sensor measurements were fused in the local map at the object level, with each sensor measurement treated as a measurement of a single object. *Skynet's* Delphi radars and MobilEye SeeQ software (run on *Skynet's* rear-facing Unibrain optical camera) fitted easily into this framework, as their proprietary algorithms transmitted lists of tracked obstacles. Data from the laser rangefinders was clustered to fit into this object level framework.

The Local Map formulated obstacle detection and tracking as the task of simultaneously tracking multiple obstacles and determining which sensor measurements corresponded to those obstacles (Miller and Campbell, 2007), (Miller et al., 2008). The problem was cast in the Bayesian framework of estimating a joint probability density:

$$p\left(N\left(1:k\right),X\left(1:k\right)|Z\left(1:k\right)\right) \tag{1}$$

where $N\left(1:k\right)$ were a set of discrete variables assigning sensor measurements to tracked obstacles at time indices 1 through k, $X\left(1:k\right)$ were the continuous states of all obstacles being tracked at time indices 1 through k, and $Z\left(1:k\right)$ were the full set of sensor measurements at time indices 1 through k. The number of obstacles being tracked was also implicitly represented in the cardinality of the measurement assignments and obstacle states, and needed to be estimated by the local map. To do so, equation 1 was factorized to yield two manageable components:

$$p\left(N\left(1:k\right)|Z\left(1:k\right)\right)\cdot p\left(X\left(1:k\right)|N\left(1:k\right),Z\left(1:k\right)\right) \tag{2}$$

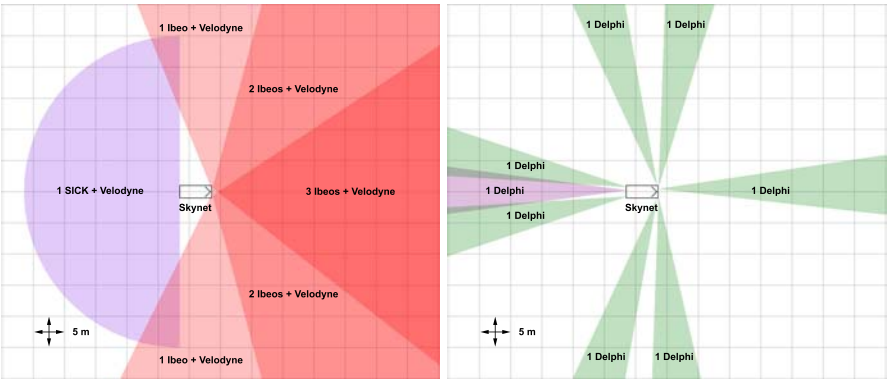


Fig. 15. (left) Laser rangefinder azimuthal coverage diagram for Team Cornell's *Skynet*. (right) Radar azimuthal coverage diagram. *Skynet* faced right in both coverage diagrams. A rear-facing optical camera is not shown, nor are two laser rangefinders with vertical scan planes that detected obstacles immediately to the left and right of *Skynet*.

where, intuitively, $p(N(1:k)|Z(1:k))$ describes the task of determining the number of obstacles and assigning measurements to those obstacles, and $p(X(1:k)|N(1:k),Z(1:k))$ describes the task of tracking a known set of obstacles with known measurement correspondences. In the local map, these two densities were estimated separately using a particle filter to make Monte Carlo measurement assignments and banks of extended Kalman Filters (EKFs) to track obstacles given those assignments (Miller and Campbell, 2007), (Miller et al., 2008). The obstacles were then broadcast at 10 Hz on *Skynet's* data network. A second layer, called the **Track Generator**, combined these obstacles with *Skynet's* position estimates to generate high level obstacle metadata for the planner, including a stable identification number, whether each obstacle was stopped or shaped like a car, and whether each obstacle occupied any nearby lanes.

4.3 Intelligent Planning

Team Cornell's intelligent planning system used *Skynet's* probabilistic interpretation of the environment to plan mission paths within the context of the rule-based road network. The planner's top level behavioral layer combined offline mission information with sensed vehicle and environment information to choose a high level behavioral state given *Skynet's* current situation. The middle level tactical layer then chose contextually appropriate maneuvers based on the selected behavior and the states of other nearby agents. The low-level operational layer translated these abstract maneuvers into actuator commands, taking into account road constraints and nearby obstacles. The following sections describe each of the three primary layers of the planner.

4.3.1 Behavioral Layer

The Behavioral Layer was the most abstract layer in Team Cornell's planner. Its job was to plan the fastest route to the next mission checkpoint, and then to select one of four high-level behavior states to achieve the planned route. The first part of that task, route planning, was solved using a modified version of the A* graph search algorithm (Russell and Norvig, 2003), (Ferguson et al., 2004). First, the DARPA road network was converted from the RNDF format to a graphical hierarchy of segments (Willemsen et al., 2003). The Behavioral Layer planned routes on this graphical hierarchy using dynamically calculated traversal times as costs for road partitions, lane changes, turns, and other maneuvers. After planning a route, the Behavioral Layer selected a high-level behavior state to make progress along the desired path. Four behavioral states were defined for the Urban Challenge: road, intersection, zone, and blockage, each deliberately defined as broadly as possible to promote planner stability. Each of these high-level behaviors executed a corresponding tactical component that drove *Skynet's* actions until the next behavior change.

4.3.2 Tactical Layer

When *Skynet* transitioned to a new behavior state, a corresponding tactical component was executed. All components divided the area surrounding *Skynet* into regions and created monitors to detect events that might have influenced *Skynet's* actions. All components also accessed a common list of intelligent agents, whose behavior was monitored in the planner using estimates from the track generator. Differences between tactical components lay in the types of region monitors they used and in the actions they took in response to nearby events.

The first tactical component was the **Road Tactical**, which controlled *Skynet* when it drove down an unblocked road. This component was responsible for maintaining a desired lane, evaluating possible passing maneuvers, and monitoring nearby agents. At each planning cycle, the road tactical checked agents in front of *Skynet* for speed adjustment, adjacent to *Skynet* for lane changes, and behind *Skynet* for impending collisions and reverse maneuvers (Sukthankar, 1997). Using these checks, the road tactical selected a desired speed and lane to keep. These were passed to the operational layer as a reference path.

The second tactical component was the **Intersection Tactical**, which controlled *Skynet* in intersections. This component was responsible for achieving proper intersection queuing behavior and safe merging. It accomplished these goals by monitoring agent arrival times and speeds at each intersection entry, maintaining a queue of agents with precedence over *Skynet*. When the intersection monitors determined that *Skynet* was allowed to proceed, a target speed, goal point, and a polygon defining the intersection were passed along to the operational layer as a reference path.

The third tactical component was the **Zone Tactical**, which controlled *Skynet* after it entered a zone. This component was responsible for basic navigation in unconstrained zones, including obstacle avoidance and alignment for parking maneuvers. The zone tactical planned over a human-annotated graph drawn on the zone during RNDF preprocessing. The graph imposed wide artificial lanes and directions of travel onto portions of the zone, allowing Skynet to treat zones as if they were roads.

The zone tactical generated the same type of local lane geometry information as the road tactical to send to the operational layer as a reference path.

The final tactical component was the **Blockage Tactical**, which controlled *Skynet* when obstacles blocked forward progress on the current route. This component was responsible for detecting and recovering from road blocks to ensure continued mission progress. Team Cornell's blockage detection and recovery relied on the Operational Layer's constrained nonlinear optimization strategy, described in section 4.3.3, to detect the location of the blockage and any possible paths through it. After initial blockage detection, the blockage tactical component proceeded through an escalation scheme to attempt recovery. First, the blockage was confirmed over multiple planning cycles to ensure that it was not a short-lived tracking error. Second, a reverse or reroute maneuver was executed to find an alternate route on the RNDF, if one were available. If no alternate route existed, *Skynet* reset the local map and scene estimator to remove long-lived mistakes in obstacle detection. If this step failed, planning constraints were relaxed: first the admissible lane boundaries were widened, then obstacles were progressively ignored in order of increasing size. *Skynet's* recovery process escalated over several minutes in a gradual attempt to return to normal driving.

4.3.3 Operational Layer

The Operational Layer converted the Tactical Layer's reference path and speed into steering, transmission, throttle, and brake commands to drive *Skynet* along the desired path while avoiding obstacles. To accomplish this task, the Operational Layer first processed each obstacle into a planar convex hull. The obstacles were then intersected with lane boundaries to form a vehicle-fixed occupancy grid (Martin and Moravec, 1996). The A* search algorithm was used to plan an initial path through the free portion of the occupancy grid (Russell and Norvig, 2003). This initial path was then used to seed a nonlinear trajectory optimization algorithm for path smoothing.

Skynet's nonlinear trajectory optimization algorithm attempted to smooth the initial path to one that was physically drivable, subject to actuator constraints and obstacle avoidance. The algorithm discretized the initial path into a set of n equally spaced base points p_i, $i \in \{1,n\}$. A set of n unit-length 'search vectors' u_i, $i \in \{1,n\}$ perpendicular to the base path are also created, one for each base point. The trajectory optimizer then attempted to find a set of achievable smoothed path points $z_i = p_i + w_i \cdot u_i$, $i \in \{1,n\}$ by adjusting search weights w_i, $i \in \{1,n\}$. Target velocities v_i, $i \in \{1,n\}$ were also considered for each point, as well as a set of variables q_i^l and q_i^r, $i \in \{1,n\}$ indicating the distance by which each smoothed path point z_i violated desired spacings on the left and right of *Skynet* created from the list of polygonal obstacles. Search weights, velocities, and final obstacle spacings were chosen to minimize the cost function J:

$$J\left(w_i, v_i, q_i^l, q_i^r\right) = \alpha_c \sum_{i=2}^{n-1} c_i^2 + \alpha_d \sum_{i=2}^{n-2} (c_{i+1} - c_i)^2 + \alpha_w \sum_{i=1}^{n} \left(w_i - w_i^t\right)^2$$

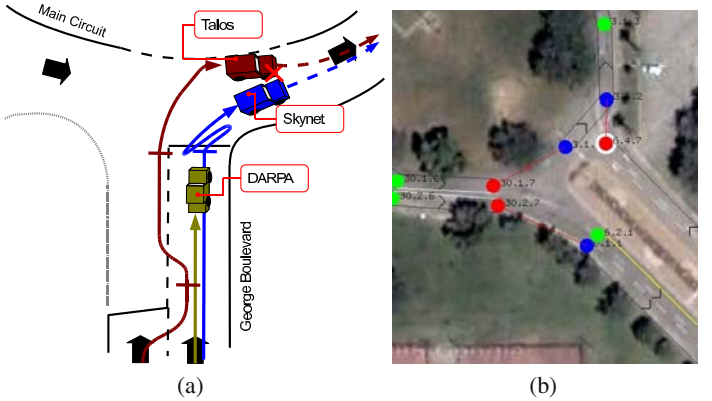
$$+ \alpha_q \sum_{i=1}^{n} \left(q_i^l + q_i^r \right) + \alpha_a \sum_{i=1}^{n-1} a_i^2 - \alpha_v \sum_{i=1}^{n} v_i$$

where α_c, α_d, α_w, α_q, α_a, and α_v are tuning weights, c_i is the approximated curvature at the i^{th} path point, w_i^t is the target search weight at the i^{th} path point, and a_i is the approximated forward vehicle acceleration at the i^{th} path point. This cost function is optimized subject to a set of 6 rigid path constraints:

1. Each search weight w_i cannot push the smoothed path outside the boundary polygon supplied by the tactical layer.
2. Each obstacle spacing variable q_i^l and q_i^r cannot exceed any obstacle's minimum spacing requirement.
3. Curvature at each path point cannot exceed *Skynet's* maximum turning curvature.
4. Total forward and lateral vehicle acceleration at each path point cannot exceed assigned limits.
5. Each search weight w_i and set of slack variables q_i^l and q_i^r must never bring *Skynet* closer to any obstacle than its minimum allowed spacing.
6. The difference between consecutive path weights w_i and w_{i+1} must not exceed a minimum and maximum.

Additional constraints on initial and final path heading were also occasionally included to restrict the smoothed path to a particular end orientation, such as remaining parallel to a lane or a parking spot.

The constrained optimization problem is solved using LOQO, an off-the-shelf nonlinear non-convex optimization library. Two optimization passes were made through each base path to reach a final smoothed path. The first step of the smoothed path was then handed to two independent low-level tracking controllers, one for desired speed and one for desired curvature. The optimization was restarted from scratch at each planning cycle, and was run at 10 Hz.



(a) (b)

Fig. 16. (a) Diagram of the incident. (b) The collision took place while the vehicles traversed the intersection from waypoint (6.4.7) to (3.1.2).

5 The Collision

Undoubtedly the most observed incident between robots during the Urban Challenge was the low-speed collision of *Talos* with *Skynet*. The location of the incident and a diagram of the accident progression are shown in Figure 16.

The collision between *Skynet* and *Talos* occurred during the second mission for both teams. Both vehicles had driven down Washington Boulevard and were attempting to merge on to Main Circuit to complete their latest sub-mission. *Skynet* drove down George Boulevard and was the first to arrive at the intersection. The vehicle paused, moved forward on to Main Circuit (around two car lengths), and then came to a stop. It backed up about three car lengths, stopped, drove forward a car length, stopped again before finally moving forward just as *Talos* was approaching Main Circuit. *Talos* was behind *Skynet* and *Skynet's* chase vehicle on approach to the intersection. *Talos* then passed the queuing *Skynet* chase vehicle on the left. *Talos* then stopped beside the chase vehicle while *Skynet* was backing up back over the stop line. When *Skynet* moved forward again, *Talos* drove up and came to a stop at the stop line of the intersection. *Talos* then drove out to the left of *Skynet* as if to pass. *Talos* was along side *Skynet* in what was looking to be a successful passing maneuver, when *Talos* turned right, pulling close in front of *Skynet*, which was now moving forward.

Next, in Sections 6 and 7, we will branch off and look at the collision from inside the *Skynet* and *Talos* software.

6 The Collision from Inside *Skynet*

UCE spectators characterized *Skynet* as having three erratic maneuvers in the seconds leading up to its collision with *Talos*. First, *Skynet* stuttered through its turn into the south entrance of the traffic circle, coming to several abrupt stops. Second, *Skynet* drove backward after its second stop, returning almost fully to the stop line from which it had just departed. Finally, *Skynet* stuttered through the turn once again, ignoring *Talos* as it approached from behind, around to *Skynet's* driver side, and finally into a collision near *Skynet's* front left headlight. Sections 6.1, 6.2, and 6.3 describe, from a systems-level perspective, the sequence of events causing each erratic maneuver.

6.1 Stuttering through the Turn

Although it did not directly cause the collision, *Skynet's* stuttering through its turn into the traffic circle was one of the first erratic behaviors to contribute to the collision. At its core, *Skynet's* stuttering was caused by a complex interaction between the geometry of the UCE course and its GPS waypoints near the turn, the probabilistic obstacle detection system discussed in section 4.2, and the constraint-based planner discussed in section 4.3.3. First, Team Cornell defined initial lane boundaries by growing polygonal admissible driving regions from the GPS waypoints defining the UCE course. This piecewise-linear interpretation of the lane

worked best when the lane was straight or had shallow curves: sharp turns could yield polygons that excluded significant portions of the lane. The turn at the southern entrance to the traffic circle suffered from this problem acutely, as the turn was closely bounded on the right by concrete barriers and a spectator area. Figure 17 shows that these concrete barriers occupied a large region of the lane polygon implied by the DARPA waypoints. The resulting crowded lane polygon made the turn difficult: *Skynet's* constraint-based Operational Layer, described in section 4.3.3, would not generate paths that drive outside the lane polygon. With space already constrained by *Skynet's* internal lane polygon, small errors in absolute position or obstacle estimates could make the path appear infeasible.

Path infeasibility caused by these types of errors resulted in *Skynet's* stuttering through the south entrance to the traffic circle. At the time leading up to the collision, small variations in clusters of laser rangefinder returns and Monte Carlo measurement assignments in the local map caused *Skynet's* path constraints to change slightly from one planning cycle to the next. In several cases, such as the one shown in Figure 18, the constraints changed to make *Skynet's* current path infeasible. At that point *Skynet* hit the brakes, as the Operational Layer was unable to find a feasible path along which it could make forward progress.

In most cases, variations in the shapes of obstacle clusters and Monte Carlo measurement assignments, like the one shown in Figure 18, cleared in one or two planning cycles: for these, *Skynet* tapped the brakes before recovering to its normal driving mode. These brake taps were generally isolated, but were more deleterious near the traffic circle for two reasons. First, the implied lane polygons forced *Skynet* to drive close to the concrete barriers, making it more likely for small mistakes to result in path infeasibility. Second, *Skynet's* close proximity to the concrete barriers actually made clustering and local map mistakes more likely: Ibeo laser rangefinders and Delphi radars tended to produce more false detections when within 1.5 m of

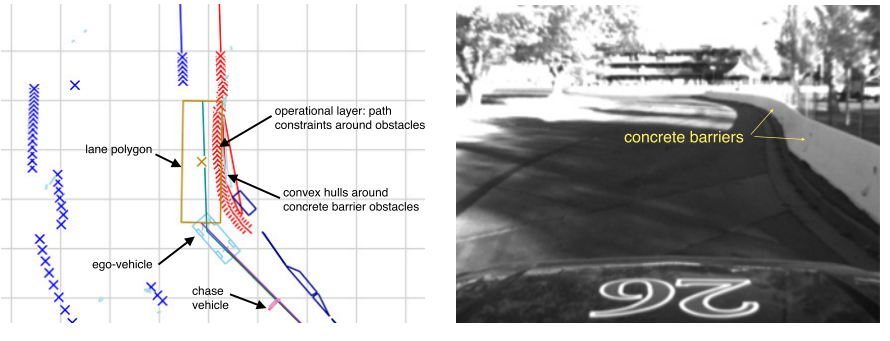


Fig. 17. (left) The lane polygon implied by piecewise-linear interpolation of DARPA waypoints in the turn near the south entrance to the traffic circle. Obstacle constraints from nearby concrete barriers occupied a significant portion of the lane polygon. (right) *Skynet* camera view of the concrete barriers generating the constraints.

an obstacle. The interaction of these factors produced the stuttering behavior, which happened several times at that corner during the UCE.

6.2 Reversing toward the Stop Line

Occasionally, variations in obstacle clusters and poor Monte Carlo measurement assignments in the local map were more persistent: in these cases phantom obstacles may appear in the lane, blocking forward progress for several seconds. In these failures the local map typically did not have enough supporting sensor evidence to delete the phantom obstacle immediately, and allowed it to persist until that evidence was accumulated. When this happened, *Skynet* considered the path blocked and executed the blockage recovery tactical component to deal with the situation. Blockage recovery was activated 10 times over the 6 hours of the UCE.

One of the 10 blockage recovery executions occurred immediately prior to *Skynet's* collision with *Talos*. In this scenario, a measurement assignment mistake caused a phantom obstacle to appear part-way into *Skynet's* lane. The phantom obstacle, shown in Figure 19, caused *Skynet* to execute an emergency braking maneuver. The phantom obstacle was deleted after approximately 2 seconds, but the adjustments to the Operational Layer's constraints persisted long enough for the Operational Layer to declare the path infeasible and the lane blocked. The mistake sent *Skynet* into blockage recovery. In blockage recovery, the Operational Layer recommended the Tactical Layer reverse to reposition itself for the turn. The

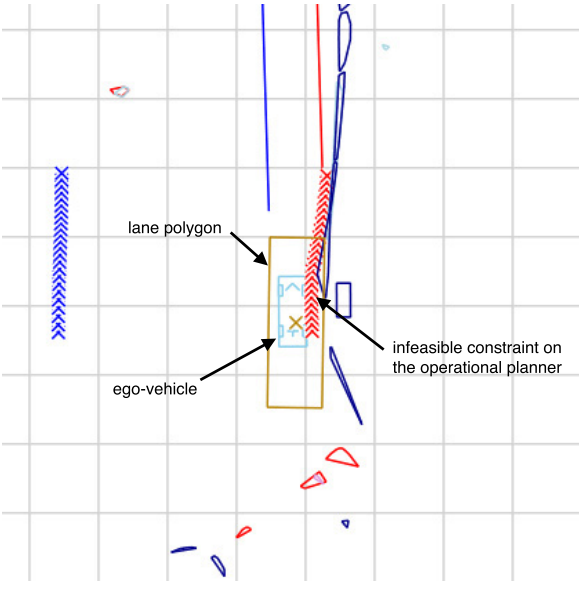


Fig. 18. Small variations in *Skynet's* perception of a concrete barrier cause its planned path to become infeasible.

Tactical Layer accepted the recommendation, and *Skynet* reversed one vehicle length to reposition itself.

6.3 Ignoring *Talos*

After the reverse maneuver described in section 6.2, *Skynet* still had not completed the turn necessary to continue with its mission. The planner therefore remained in its blockage recovery state, though recommendation and completion of the reverse maneuver left it in an escalated state of blockage recovery. In this state the Tactical Layer and Operational Layer once again evaluated the turn into the traffic circle, this time ignoring small obstacles according to the blockage recovery protocol described in section 4.3.2. The Operational Layer decided the turn was feasible, and resumed forward progress. Although the Local Map produced no more phantom obstacles for the duration of the turn, small errors in laser rangefinder returns once again forced the Operational Layer to conclude that the path was infeasible. At this point, the Tactical Layer escalated to its highest state of blockage recovery, removing constraints associated with lane boundaries. Figure 20 shows this escalation from *Skynet's* normal turn behavior to its decision to ignore lane boundaries.

Unfortunately, *Skynet* still perceived its goal state as unreachable due to the nearby concrete barriers. At the highest level of blockage recovery, however, *Skynet* was deliberately forbidden to execute a second reverse maneuver to prevent an infinite planer loop. Instead, it started a timer to wait for the error to correct itself, or

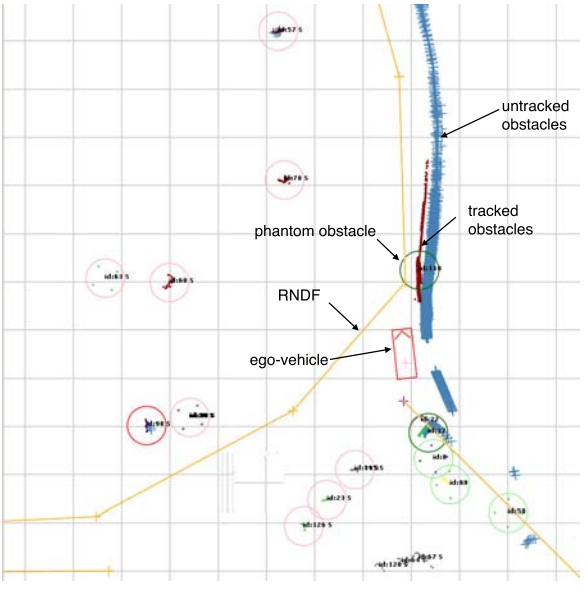


Fig. 19. A measurement assignment mistake causes a phantom obstacle to appear, momentarily blocking *Skynet's* path.

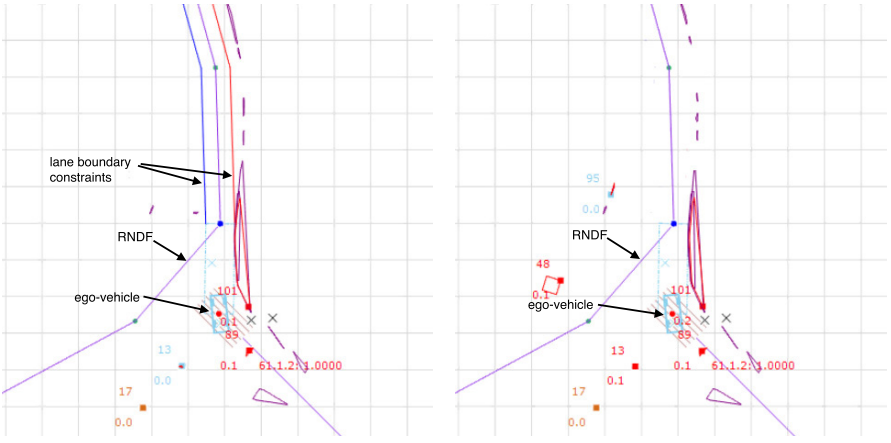


Fig. 20. (left) *Skynet* resumed its turn after a reverse maneuver. (right) Perceiving the turn infeasible a second time, *Skynet* dropped constraints associated with lane boundaries.

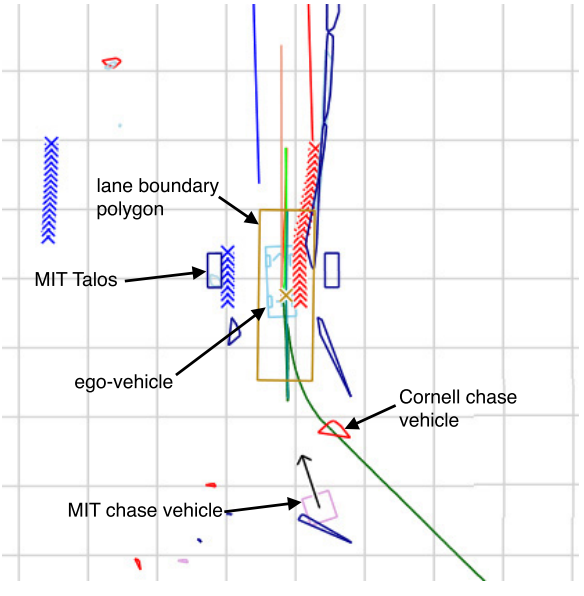


Fig. 21. Skynet ignored *Talos* as it drove outside *Skynet's* polygonal lane boundary.

barring forward progress for several minutes, to reset the local map and eventually the planner itself. Neither of these soft resets would be realized, however, as *Talos* was already weaving its way behind as *Skynet* started its timer.

While *Talos* passed behind and then to the left of *Skynet*, the Operational Layer continued to believe the forward path infeasible. Coincidentally, just as *Talos* pulled out to pass *Skynet* on the left, a slight variation in the obstacle clustering and

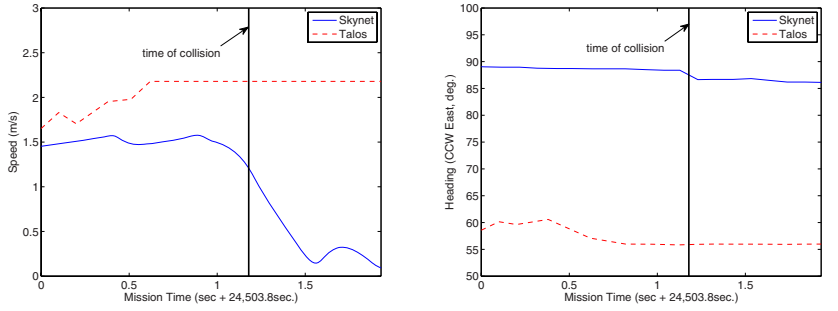


Fig. 22. From *Skynet* logs: Speed (left) and heading (right) for both *Skynet* and *Talos* just before and after collision. Flat line in *Talos'* plot indicates where *Skynet* stopped tracking *Talos*.

measurement assignments accumulated enough evidence in the local map to perceive the path as feasible. With the path momentarily feasible, *Skynet* began to drive forward as *Talos* passed on its left. Here *Skynet's* Tactical Layer ignored *Talos*, because *Talos* drove outside the piecewise-linear polygonal lane boundary, as shown in Figure 21. *Skynet's* Operational Layer also ignored *Talos*, as *Talos* did not constrain the target path in front of *Skynet* in any way. Once *Talos* passed to *Skynet's* left, *Talos* was no longer detected as a moving obstacle; *Skynet's* sideways-facing Sick LMS-291s were mounted with a vertical scan plane and provided only weak position information and no velocity information. The Local Map began tracking *Talos* as a moving obstacle only 1 second before the collision, when it entered into view of *Skynet's* forward-mounted Ibeo ALASCA XTs. Unfortunately, with concrete barriers on *Skynet's* right and *Talos* approaching on its left, no evasive maneuver was available. At that point, given the preceding chain of events, the collision was inevitable.

Figure 22 shows the speed and heading, as estimated on *Skynet*, for both the *Skynet* and *Talos* vehicles. Approximately 0.5 sec before collision, the speed and heading estimates for *Talos* remain constant, which is the time that they are stopped being tracked. *Skynet* did not change its heading or velocity before the collision, indicating that no adjustments were made to the *Talos* movements. Finally, after impact, there was a fast change in *Skynet's* heading, indicating the collision, and its velocity decreases quickly to zero soon after.

7 The Collision from Inside *Talos*

The incident from the *Talos'* viewpoint is shown in Figures 23, 24 and 25. Figure 23 shows that earlier along George Boulevard, the road was dual-lane. *Talos* was going faster along the straight road than the *Skynet* chase vehicle, so *Talos* passed to the left of the chase vehicle (Figure 23(a)). At the end of Washington Boulevard, the road merged (via cones on the left) into a single lane on the right. *Talos* did not have room to merge right in front of the chase vehicle, so *Talos* slowed to a stop while the

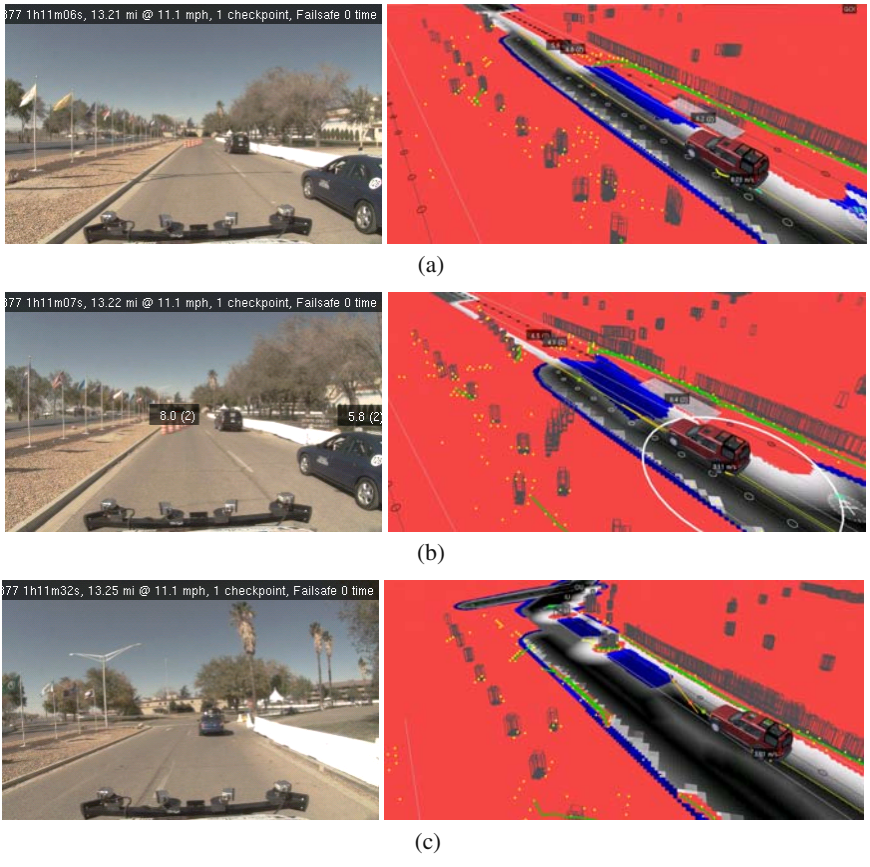


Fig. 23. *Talos'* view of the lead-up to the *Skynet-Talos* incident. (a) *Talos* started to pass *Skynet's* chase vehicle. (b) *Talos* was forced to slow down and merge behind the *Skynet* chase vehicle. (c) *Talos* queued behind the *Skynet* chase vehicle.

Skynet chase vehicle moved ahead. When space was available, *Talos* merged behind the *Skynet* chase vehicle (Figure 23(b)). *Skynet* and the chase vehicle then come to a stop at the intersection (Figure 23(c)).

In Figure 24 we see that at first, *Talos* stopped behind the chase vehicle. However, the lane width was sufficient that *Talos* soon found a path to the left of the chase vehicle (Figure 24(a)). In this case *Talos* was not in a passing mode; it had simply found room on the left-hand side of the current lane to squeeze past the DARPA chase vehicle.

7.1 Wide Lane Bug

The lane was significantly wider to the left because of a Drivability Map construction bug. As described in Section 3, lanes were carved out of the lane-cost map

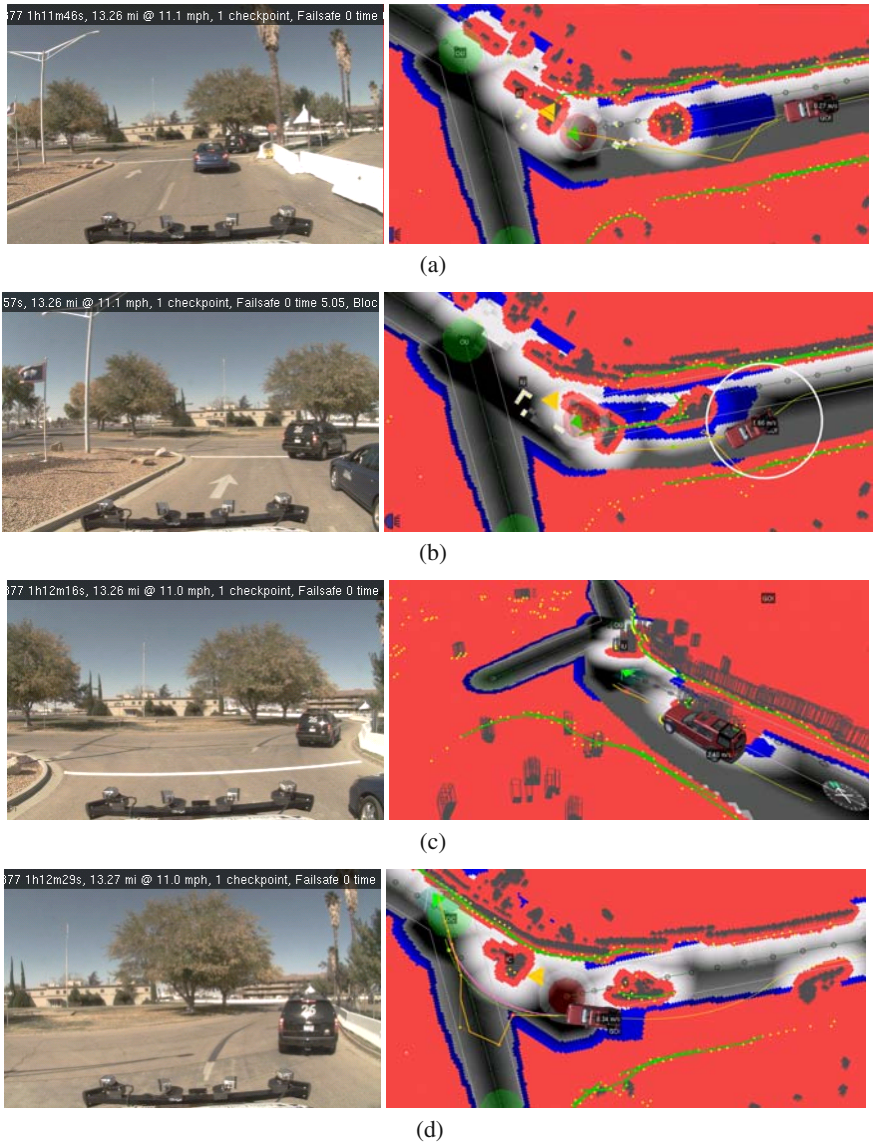


Fig. 24. Lead-up to the *Skynet-Talos* incident. (a) *Talos* found a route around the chase vehicle. (b) *Skynet* backed up onto *Talos'* goal position, *Talos* braked. (c) *Skynet* advanced again. *Talos* passed the chase vehicle. (d) *Talos* yielded at the intersection. There were no moving vehicles nearby, so it proceeded.

like valleys through a plateau. Adjacent lanes carved out often caused small islands remaining between the valleys. These islands were addressed by explicitly planing down the region between adjacent lanes. This strategy worked well in general,

however in this case the road merged down to one lane shortly before the intersection. The adjacent lane was not rendered after the merge, which was correct. However, the planing operation was done all the way along the right lane past the merge point. The effect of the planing alone made the road 3 meters wider on the left than it would otherwise have been. Without the extra width, *Talos* would have been forced to queue behind the DARPA chase vehicle.

7.2 At the Intersection

Figures 24(b) & (c) show how *Talos* pulled out and drove around to the left of the chase vehicle. The robot had a motion plan which was attempting to reach a goal point on the stop line of the intersection. *Talos* was beside the chase vehicle when *Skynet* backed up and occupied *Talos'* goal position. *Talos* came to a stop, unable

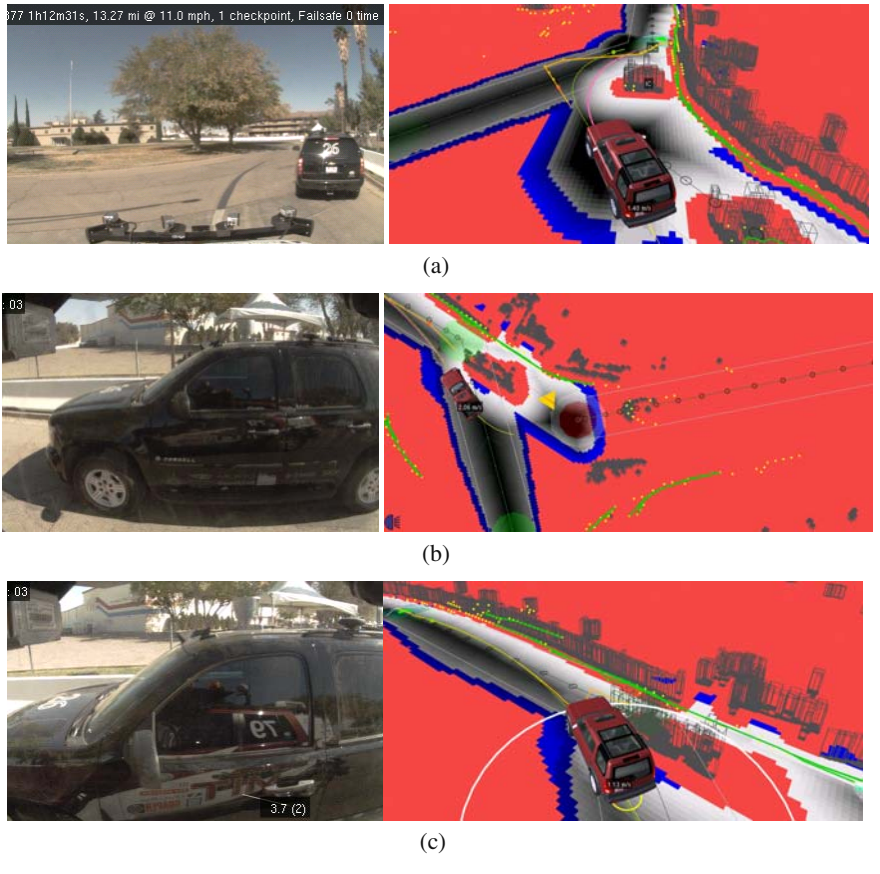


Fig. 25. *Skynet-Talos* incident. (a) *Talos* planned a route around *Skynet*, which appeared as a static object. (b) While *Talos* was passing, *Skynet* began to move. (c) While applying emergency braking, *Talos* turned into *Skynet*.

to drive through the restricted region to get to the goal. In the visualization, *Skynet* did not have a restricted region in front and behind the vehicle. This was because *Skynet* was within the intersection. Obstacles detected inside the intersection did not have restricted regions because the heuristic was that obstacles inside intersections were things like sign posts, traffic islands and encroaching trees. *Skynet* then moved forward again, making *Talos'* goal position clear. *Talos* drove to the stop line. Although now adjacent to *Skynet's* chase vehicle, *Talos* wasn't in a failsafe mode. The artificially widened lane permitted *Talos* to drive up to the intersection as it would a passing parked car or any other object on the side of the road not blocking the path. At the intersection *Talos* followed the standard procedure of giving precedence to obstacle/vehicles approaching on the left. There were no moving or static obstacles in the intersection to the left of *Talos* on Main Circuit, so the software moved the short-term goal point to the exit of the intersection (waypoint 3.1.2) and *Talos* proceeded.

7.3 The Collision

Finally, Figures 24(d) and 25(a) show that *Talos* planned a path out to the left through a region that had a low cost by avoiding *Skynet* (which *Talos* perceived as a static obstacle). *Talos'* goal point moved further down Main Circuit, requiring the robot to find a trajectory that would have an approach angle to the lane shallow enough to drive within the lane boundaries of Main Circuit. *Talos* was now inside the intersection bounding box. *Talos* planned a path around the "*Skynet* static object" and down Main Circuit within the lane boundaries. The path was close to *Skynet* so the route had a high cost but was physically feasible. *Talos* drove between *Skynet* (on the right) and the lane boundary constraint (on the left). *Talos* then pulled to the right so that it could make the required approach angle to enter the lane. Had *Skynet* remained stationary at this point *Talos* would have completed the passing maneuver successfully. In Figure 25(b), we can see that *Skynet* starts moving forward. Had *Skynet* been moving faster (i.e., $>$ 3m/s instead of 1.5m/s), a moving obstacle track would have been initiated in the *Talos* software and a "no-go" region would have been extruded in front of the vehicle. This region would have caused *Talos* to yield to *Skynet* (similar to what occurred with *Odin* and *Talos* in Section 2.5). Instead *Talos* turned to the right to get the correct approach angle to drive down the lane; *Skynet* moved forward; the robots collided (Figure 25(c)).

7.4 Clusters of Static Objects

Talos perceived *Skynet* as a cluster of static objects. The positions of the static objects evolved over time. This may sound strange, however it is not uncommon for a cluster of static obstacles to change shape as the ego-vehicle position moves. It could be due, for instance, to more of an object becoming visible to the sensors. For example, the extent of the concrete barrier detected on the right of *Talos* in Figures 23(a),(b)& (c) varied due to sensor range and aspect in relation to the ego-vehicle. Because the software treated *Skynet* as a collection of static objects instead

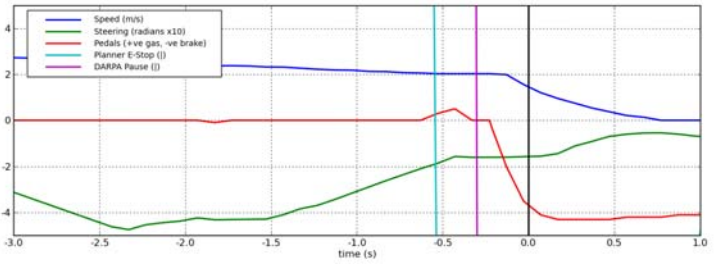


Fig. 26. *Talos'* speed, wheel angle and pedal gas and brake positions during the collision. Time 0.0 is the initial collision. Motion Planner E-Stop was at $-550msec$. DARPA Pause at $-400msec$. The vehicle came to rest after $750msec$.

of a moving obstacle no forward prediction was made on *Skynet's* motion. *Talos* was driving between a lane constraint on the left and a collection of static objects on the right. If *Skynet* had not moved *Talos* would have negotiated the gap just as it had to avoid K-rails and other static objects adjacent to the lanes throughout the day. Instead, unexpectedly for *Talos*, *Skynet* moved forward into the planned path of *Talos*. Without a forward-motion prediction of *Skynet*, by the time *Skynet* was in *Talos'* path, *Talos* was unable to come to a stop before colliding.

Figure 26 shows the vehicle state during the collision. Talos had straightened its wheels to around 9° to the right and was traveling around $2m/s$. The vehicle detected that the motion planning tree had been severed $550msec$ before the collision. It was replanning and no evasive maneuver was performed yet. $150msec$ later the vehicle was coasting and DARPA Paused the vehicle. At the collision *Talos* was moving at $1.5m/s$, dropping to zero $750msec$ after the initial collision. In the log visualization *Talos* was pushed slightly forward and to the left of its heading by the impact (about $0.3m$).

The contributing factors of *Talos'* behavior can be decomposed as: the inability to track slow-moving objects, the use of a moving-obstacle model versus explicit vehicle classification and the dominant influence of lane constraints on motion planning & emergency path diversity. The other contributing factor, the Drivability Map rendering bug which widened the lane to allow *Talos* to attempt to drive around instead of queue, is a test-coverage issue and holds little to analyze further.

7.5 Inability to Track Slow-Moving Objects

At the lowest layer, all objects tracked by the obstacle detection system had a velocity component. However, both sensor noise and changing viewpoint geometry can masquerade as small velocities, making it difficult to reliably determine whether an object is actually moving. The problem of changing viewpoint is especially problematic. If Talos is moving, the visible portion of other obstacles can change; the "motion" of the visible portion of the obstacle is difficult to distinguish from an obstacle that is actually moving.

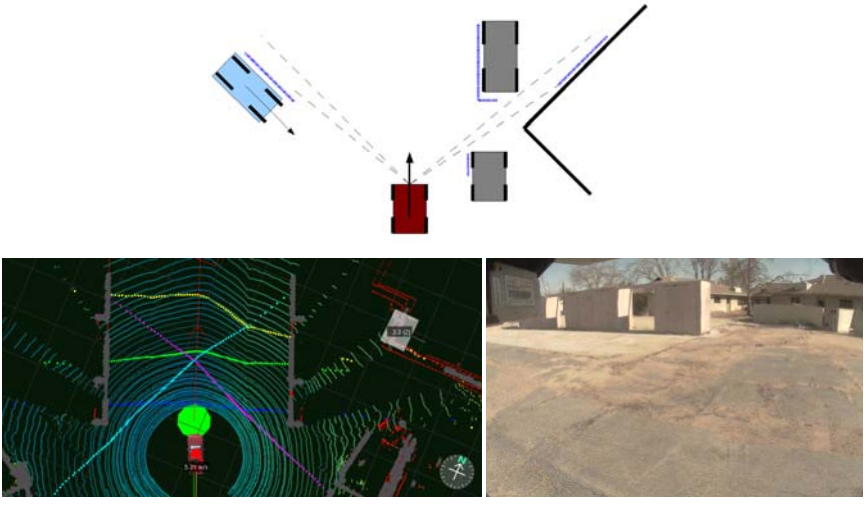


Fig. 27. (a) Illustration of lidar aperture problem. The building on right generates a lidar return indistinguishable from the fast-approaching vehicle on the left. (b) Walls entering the *White Zone* generate phantom moving obstacles from building lidar returns.

Apertures between the sensor and the obstacle present addition complications. Figure 27 shows an example in which a small near-field aperture resulted in a hallucinated car. In this case, only a small patch of wall was visible through the aperture: as Talos moved, an object appeared to be moving in the opposite direction on the other side of the aperture. Several groups have attempted to counter this problem using algorithms to determine the shadowing of distant objects by near-field returns (Thrun et al., 2006). However, with more complex sensor characteristics such as the 64 laser Velodyne sensor, and more complex scene geometries for urban environments, these techniques become difficult to compute. A flat obstacle occlusion map is no longer sufficient since obstacle height must be considered.

7.6 Moving Obstacles versus Explicit Vehicle Classification

As described in Section 3, the MIT vehicle did not explicitly detect vehicles. Instead, objects in the scene were classified either as static or moving obstacles. Moving obstacles were rendered with exclusion regions in the direction of travel along the lane. The decision to use moving obstacles was taken to avoid the limitations of attempting to classify the sensing data into “vehicle” or “non-vehicle” classes. The integrated system was fragile however as classification errors or outages caused failures in down stream modules blindly relying on the classifications and their persistence over time. Up until and including the MIT site visit in June 2007, the software did attempt to classify vehicles based on size and lane proximity. During one mission lane precedence failed due to sensor noise, causing a vehicle to be lost momentarily and then reacquired. The reacquired vehicle had a different ID number,

making it appear as a new arrival to the intersection, so *Talos* incorrectly went next. In reaction to this failure mode, Team MIT migrated to use the concept of static and moving obstacles. The rationale for this was that sensor data classification schemes, by requiring a choice to be made, introduced the potential for false positive and false negative classifications. Much care could be taken in reducing the likelihood of mis-classifications, however classification errors could almost always still occur. Developers have often designed down-stream applications to be over-confident in the classes assigned to objects. Avoiding the assignment of "vehicle"/"non-vehicle" classes to detected objects was an attempt to cut down assumptions made by the down-stream applications interpreting the detected obstacle data. The assumptions were made in relation to the strict definition of "static" and "moving" obstacles instead. On the whole this approach scaled well with the additional complexity of the final race. The apparent gap was in the correct treatment of active yet stationary vehicles. The posture of *Skynet* was not very different from the stationary cars parked in the Gauntlet of Area B during the qualifiers.

7.7 Lane Constraints and Emergency Path Diversity

In the nominal situation, the tree of trajectories ended in stopped states, so that *Talos* always knew how to come to a safe stop. When *Talos* was moving and the planner could not find any feasible safe path from the current configuration (possibly due to a change in the perceived environment caused by sensing noise or dynamic obstacles that changed the constraints) the planner generated a emergency braking plan. This emergency plan consisted of the steering profile of the last feasible path and a speed profile with the maximum allowable deceleration. Before the collision (Figure 25(b)), the tree of trajectories was going towards the target further down the road. When the gap between the left lane boundary and *Skynet* was narrowed as the *Skynet* moved forward, no feasible plan was found that stopped *Talos* safely.When no feasible solution is found, a better approach would be to prioritize the constraints. In an emergency situation, satisfying lane constraints is not as important as avoiding a collision. Therefore, when generating an emergency plan, the planner could soften the lane constraints (still using a relatively high cost) and focus on ensuring collision avoidance with the maximum possible braking.

8 Discussion

Neither vehicle drove in a manner "sensible" to a human driver. On a day of fine weather and good visibility *Skynet* backed up in a clear intersection and started to accelerate when another vehicle was closing in. *Talos* passed a vehicle instead of queuing in a single-lane approach, then pulled in much too close to an active vehicle in an intersection.

To summarize, contributing factors identified in the two vehicles' software were:

- *Talos'* lane-rendering bug permitting *Talos* to pass the DARPA chase vehicle
- *Talos'* inability to track slow-moving objects

- *Skynet's* sensor data associations inducing phantom objects
- *Talos'* failure to anticipate potential motion of an active stationary vehicle
- *Skynet's* failure to accommodate the motion of an adjacent vehicle in an intersection
- *Talos'* overly constrainted motion due to target lane constraints
- *Skynet's* lane representation narrowing the drivable corridor

Apart from the lane-rendering problem, these factors were more than just bugs: they reflected hard trade-offs in road environment perception and motion planning.

8.1 Sensor Data Clustering

Skynet's phantom obstacles and *Talos'* inability to track slow-moving objects represent the downsides of two different approaches to address the same problem of sensor data clustering. Team Cornell chose to estimate the joint probability density across obstacles using Monte Carlo measurement assignments to associate sensor data with objects (Section 4.2). The consequence was that sometimes the associations would be wrong, creating false positives. Team MIT found lidar data clustering too noisy to use for static objects. Instead, relying on its sensor-rich vehicle, the accumulator array with a high entropy presented static objects to motion planning directly. Once the velocity signal was sufficiently strong the clustered features robustly tracked moving objects. A high threshold was set before moving obstacle tracks were reported to suppress false positives. The consequence was that until the threshold was passed, there was no motion prediction for slow moving objects.

8.2 Implicit and Explicit Vehicle Detection

The treatment of vehicles in the road environment must extend past the physics-based justification of obstacle avoidance due to closing rate. For example, humans prefer never to drive into the longitudinal path of an occupied vehicle, even if it is stationary. In Section 2 we mentioned how the DARPA chase vehicle driver preferred to drive on the curb than in front of the Paused *Skynet* vehicle.

Many teams in the contest performed implicit vehicle detection using the object position in the lane and size to identify vehicles(Leonard et al., 2008; Miller et al., 2008; Stanford Racing Team, 2007). Moving objects detected with lidar or using radar Doppler velocity were also often assumed to be vehicles. To prevent identified vehicles being lost, several teams had a "was moving" flag associated with stationary tracked objects, such as queuing vehicles, that had once been observed moving(Tartan Racing, 2007). It is not difficult to imagine a case where a vehicle would not have been observed moving and the vehicle size and position rules of thumb would fail. Some teams also used explicit vehicle detectors such as the Mobileye SeeQ system. However, explicit vehicle detectors struggle to detect all vehicles presented at all aspects. The reconciliation of the two approaches – explicit

(a) (b)

Fig. 28. Results of explicit vehicle detection in the collision. (a) DARPA chase vehicle detected. (b) Last frame: Skynet is detected. Trees and clutter in the background also generate false positives during the sequence. In the intersection there are no lane markings so lane estimate confidence cannot be used to exclude the false detections.

vehicle detection/classification and the location/moving-obstacle approach – seems a promising solution.

Figure 28 shows the result of explicit vehicle detection run on *Talos'* logged data. Both the *Skynet* and the DARPA chase vehicle are detected, though only in a fraction of the frames in the sequence. There were also a number of false detections that would need to be handled. Explicit vehicle detection could have possibly bootstrapped *Talos'* data association and tracking, permitting standoff regions to be placed in front and behind *Skynet*. There still was an apparent gap in the correct treatment of active yet stationary vehicles. The posture of *Skynet* was not very different from the stationary cars parked along the side of a road (such as in "the Gauntlet" of Area B during the national qualifying event). Even with perfect vehicle detection, sensor data and modelling can only recover the current vehicle trajectory. Non-linear motions like the stop-start behaviors require conservative exclusion regions or an additional data source.

8.3 Communicating Intent

Drivers on the road constantly anticipate the potential actions of fellow drivers. For close maneuvering in car parks and intersections, for example, eye contact is made to ensure a shared understanding. In a debriefing after the contest, DARPA stated that traffic vehicle drivers, unnerved by being unable to make eye-contact with the robots, had resorted to watching the front wheels of the robots for an indication of their intent. As inter-vehicle communication becomes ubiquitous, autonomous vehicles will be able to transmit their intent to neighboring vehicles to implement the level of coordination beyond what human drivers currently achieve using eye-contact. This would not help in uncollaborative environments such as defense. There are also many issues such as how to handle incomplete market penetration of the communications system or inaccurate data from equipped vehicles. However, a system where very conservative assumptions regarding other vehicle behavior can be

refined using the intent of other vehicles, where available, seems a reachable objective. We look forward to these synchronized robot vehicle interactions.

8.4 Placing Lane Constraints in Context

Leading up to the collision both *Talos* and *Skynet* substantially constrained their behavior based on the lane boundaries, even though the physical world was substantially more open. *Skynet* lingered in the intersection because the lane was narrowed due to an interaction between the lane modeling and the intersection geometry. Then the vehicles collided due to a funneling effect induced by both vehicles attempting to get the optimum approach into the outgoing lane. The vehicles were tuned to get inside the lane constraints quickly; this behavior was tuned for cases such as the Area A test during the national qualifying event, in which the vehicles needed to merge into their travel lane quickly to avoid oncoming traffic. In test Area A, the robots needed to drive assertively to get into the travel lane to avoid the heavy traffic and concrete barriers lining the course. In the collision scenario, however, the road was one-way, so the imperative to avoid oncoming traffic did not exist. The imperative to meet the lane constraints remained. For future urban vehicles, in addition to perception, strong cues for behavior tuning are likely to come from digital map data. Meta data in digital maps is likely to include not only the lane position and number of lanes but also shoulder drivability, proximity to oncoming traffic and partition type. This a-priori information vetted against perception could then be used to weigh up the imperative to maximize clearance from detected obstacles with the preference to be within the lane boundaries. A key question is how the quality of this map data will be lifted to a level of assured accuracy which is sound enough to base life-critical motion planning decisions on.

9 Conclusion

The fact that the robots, despite the crash, negotiated many similarly complex situations successfully and completed the race after 6 hours of driving implied that the circumstances leading to the collision were the product of confounding assumptions across the two vehicles. Investigating the collision, we have found that bugs, the algorithms in the two vehicles architectures as well as unfortunate features of the road leading up to the intersection and the intersection topology all contributed to the collision.

Despite separate development of the two vehicle architectures, common issues can be identified. These issues reveal hard cases that extend beyond a particular software bug, vehicle design or obstacle detection algorithm. They reflect complex trade-offs and challenges: (1) Sensor data association in the face of scene complexity, noise and sensing "aperture" problems. (2) The importance of the human ability to anticipate the expected behavior of other road users. This requires an estimation of intent beyond the observable physics. Inter-vehicle communication has a good chance of surpassing driver eye-contact communication of intent, which is

often used to mitigate low-speed collisions. However, incomplete system penetration and denial of service for defense applications are significant impediments. (3) The competing trade-offs of conforming to lane boundary constraints (crucial for avoiding escalating problems with oncoming traffic) verses conservative obstacle avoidance in an online algorithm. Map data and meta data in maps about on-coming traffic and road shoulder drivability would be an invaluable data source for this equation. However, map data would need to be accurate enough to support safety-critical decisions.

Multimedia Appendices

Talos' race logs, log visualization software as well as videos of the incidents made from the logs are available at: http://grandchallenge.mit.edu/public/

Acknowledgments

The authors would like to thank all the members of their respective teams namely from Team MIT: Mitch Berger, Stefan Campbell, Gaston Fiore, Emilio Frazzoli, Albert Huang, Sertac Karaman, Olivier Koch, Steve Peters, Justin Teo, Robert Truax, Matthew Walter, David Barrett, Alexander Epstein, Keoni Maheloni, Katy Moyer, Troy Jones, Ryan Buckley, Matthew Antone, Robert Galejs, Siddhartha Krishnamurthy, and Jonathan Williams. From Team Cornell: Jason Catlin, Filip Chelarescu, Ephrahim Garcia, Hikaru Fujishima, Mike Kurdziel, Sergei Lupashin, Pete Moran, Daniel Pollack, Mark Psiaki, Max Rietmann, Brian Schimpf, Bart Selman, Adam Shapiro, Philipp Unterbrunner, Jason Wong, and Noah Zych.

In addition, the authors would like to thank Jim McBride (IVS), Matt Rupp (IVS) and Daniel Lee (Ben Franklin) who helped provide information for the events described in this paper.

The MIT team gratefully acknowledges the sponsorship of: MIT School of Engineering, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), MIT Department of Aeronautics and Astronautics, MIT Department of Electrical Engineering and Computer Science, MIT Department of Mechanical Engineering, The C. S. Draper Laboratory, Franklin W. Olin College of Engineering, The Ford-MIT Alliance, Land Rover, Quanta Computer Inc., BAE Systems, MIT Lincoln Laboratory, MIT Information Services and Technology, South Shore Tri-Town Development Corporation and Australia National University. Additional support has been provided in the form of in-kind donations and substantial discounts on equipment purchases from a variety of companies, including Nokia, Mobileye, Delphi, Applanix, Drew Technologies, and Advanced Circuits.

Team Cornell also gratefully acknowledges the sponsorship of: Singapore Technologies Kinetics, Moog, Septentrio, Trimble, Ibeo, Sick, MobilEye, The Mathworks, Delphi, and Alpha Wire.

The MIT and Cornell teams were sponsored by Defense Advanced Research Projects Agency, Program: Urban Challenge, ARPA Order No. W369/00, Program Code: DIRO.

References

DARPA, 2007. DARPA Urban Challenge rules (2007), http://www.darpa.mil/GRANDCHALLENGE/rules.asp

Ferguson et al., 2004. Ferguson, D., Stentz, A., Thrun, S.: Pao* for planning with hidden state. In: Proceedings of the 2004 International Conference on Robotics and Automation, vol. 3, pp. 2840–2847 (2004)

Frazzoli et al., 2002. Frazzoli, E., Dahleh, M.A., Feron, E.: Real-time motion planning for agile autonomous vehicles. Journal of Guidance, Control, and Dynamics 25(1), 116–129 (2002)

Leonard et al., 2008. Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., Koch, O., Kuwata, Y., Moore, D., Olson, E., Peters, S., Teo, J., Truax, R., Walter, M., Barrett, D., Epstein, A., Mahelona, K., Moyer, K., Jones, T., Buckley, R., Attone, M., Galejs, R., Krishnamurthy, S., Williams, J.: A perception driven autonomous urban robot. Submitted to International Journal of Field Robotics (2008)

Martin and Moravec, 1996. Martin, M., Moravec, H.: Robot evidence grids. Technical Report CMU-RI-TR-96-06, The Robotics Institute, Carnegie Mellon University, Pittsburgh (1996)

Miller and Campbell, 2007. Miller, I., Campbell, M.: Rao-blackwellized particle filtering for mapping dynamic environments. In: Proceedings of the 2007 International Conference on Robotics and Automation, pp. 3862–3869 (2007)

Miller et al., 2008. Miller, I., Campbell, M., Huttenlocher, D., Nathan, A., Kline, F.-R., Moran, P., Zych, N., Schimpf, B., Lupashin, S., Kurdziel, M., Catlin, J., Fujishima, H.: Team cornell's skynet: Robust perception and planning in an urban environment. Submitted to International Journal of Field Robotics (2008)

Russell and Norvig, 2003. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Pearson Education, Inc., Upper Saddle River, New Jersey (2003)

Stanford Racing Team, 2007. Stanford Racing Team, Stanford's robotic vehicle Junior: Interim report (2007), http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Stanford.pdf

Sukthankar, 1997. Sukthankar, R.: Situational Awareness for Tactical Driving. PhD thesis, The Robotics Institute, Carnegie Mellon University (1997)

Tartan Racing, 2007. Tartan Racing, Tartan racing: A multi-modal approach to the DARPA urban challenge (2007), http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Tartan_Racing.pdf

Thrun et al., 2006. Willemsen, P., Kearney, J.K., Wang, H.: Ribbon networks for modeling navigable paths of autonomous agents in virtual urban environments. In: Proceedings of IEEE Virtual Reality 2003, pp. 22–26 (2003)

Willemsen et al., 2003. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics 23(9), 661–692 (2006)

A Perspective on Emerging Automotive Safety Applications, Derived from Lessons Learned through Participation in the DARPA Grand Challenges

J.R. McBride[1,*], J.C. Ivan[1], D.S. Rhode[1], J.D. Rupp[1], M.Y. Rupp[1], J.D. Higgins[2], D.D. Turner[2], and R.M. Eustice[3]

[1,*] Ford Motor Company – Research and Advanced Engineering
Research and Innovation Center
2101 Village Road, MD 2141, Room 2113D
Dearborn, MI 48121
Phone: 313-323-1423
jmcbride@ford.com
[2] Delphi Corporation
[3] University of Michigan

Abstract. This paper reports on various aspects of the Intelligent Vehicle Systems (IVS) team's involvement in the recent 2007 DARPA Urban Challenge, wherein our platform, the autonomous "XAV-250", competed as one of the eleven finalists qualifying for the event. We provide a candid discussion of the hardware and software design process that led to our team's entry, along with lessons learned at this event and derived from participation in the two previous Grand Challenges. In addition, we also give an overview of our vision, radar and lidar based perceptual sensing suite, its fusion with a military grade inertial navigation package, and the map-based control and planning architectures used leading up to and during the event. The underlying theme of this article will be to elucidate how the development of future automotive safety systems can potentially be accelerated by tackling the technological challenges of autonomous ground vehicle robotics. Of interest, we will discuss how a production manufacturing mindset imposes a unique set of constraints upon approaching the problem, and how this worked for and against us, given the very compressed timeline of the contests.

1 Introduction and Overview

The narrative presented in this paper is derived from experiences gained by the various authors through their participation in the series of DARPA Grand Challenges (DGC1, DGC2, DGC3, also variously referred to by year, etc.). During the inaugural edition of the Grand Challenge, two of the authors worked as volunteers for DARPA, and performed tasks such as reviewing the participating teams' technical proposals, conducting safety evaluations of the entrants' robots, and serving as chase vehicle crew members. In the process, they were able to gain

* Corresponding author.

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 549–593.
springerlink.com

Fig. 1. The XAV-250 poses for the camera at the 2007 National Qualifying Event (NQE).

insight into the existing state-of-the-art of ground vehicle robotics, and established an invaluable array of personal contacts, ranging from government officials to university researchers to corporate manufacturers. By the end of the event, the consensus opinion was that many of the problems that were vexing ground vehicle robotics were of the same technological nature as the hurdles impeding the rapid development of advanced safety systems in the automotive industry.

With this conclusion in mind, in 2005 a collaborative effort was formed between Delphi, Ford, Honeywell, and PercepTek, with the goal of conducting joint research directed toward the creation of safe and robust intelligent ground vehicle systems for production-intent commercial and military applications. Participating under the collaborative name "Intelligent Vehicle Safety Technologies", or IVST, they entered the 2005 DGC, fielding the autonomous Ford F-250 dubbed the "*Desert Tortoise*". In their first attempt at ground vehicle robotics, the team impressively made it all the way through the selection process, earning the 5th starting pole position at the finals (IVST, 2005a, 2005b).

Upon announcement of the 2007 DARPA Urban Challenge (DUC / DGC3), three core members from the previous effort formed a new collaboration, this time known as "Intelligent Vehicle Systems" (IVS), which was initially comprised primarily of employees from Delphi, Ford, and Honeywell. However, as the project evolved, the collaboration expanded to include contributions from a variety of external organizations, examples including Cybernet, the University of Michigan (UMich), and the Massachusetts Institute of Technology (MIT). The team once again used the F-250 as its base platform, but significant modifications were made to the sensing suite and computing architecture. The "XAV-250"

(eXperimental Autonomous Vehicle - 250) as it was now called, might seem to be a surprising choice for an urban driving environment, given its size and mass, and particularly given that its long wheel base resulted in a turning radius in excess of 8m. However, the truck had already proved its merits in the DGC2, and moreover precisely represents the type of vehicle that would likely be employed in a realistic autonomous mission capable of carrying any sort of significant cargo. If humans could keep it centered in their lane on the road, we saw no reason to believe computers could not do the same. The team was one of only eleven to achieve the finals, and one of only six to have appeared consecutively in the last two DGC final events.

It should be noted that the striking difference between the first Grand Challenges and the latest is the dynamic nature of the urban event, which introduced moving targets, intermittently blocked pathways, and extended navigation in regions of denied-GPS reception. The team's intent was to build upon the lessons learned from the *Desert Tortoise*, and specifically to evolve from a reactive, arbiter-based methodology to an architecture that dynamically senses the 3D world about the vehicle and performs complex, real-time path planning through a dense global obstacle map populated from multiple types of fused sensor inputs.

The remainder of this article is organized as follows. In Section 2 we parallel how advances in ground robotics research can lead to advances in automotive active safety applications, thereby motivating our corporate research participation in the DARPA challenges. In Section 3 we provide an overview of the hardware and software architecture in place prior to our post DARPA Site-Visit redesign which entailed a migration to the MIT software architecture (described in Section 4). Section 5 reviews our performance during the National Qualifying Event (NQE) and Finals while Section 6 offers a post-DGC reflection on engineering lessons learned through our participation. Finally, Section 7 provides some concluding remarks.

2 The Connection between Robotics Research and Automotive Safety

In common with other long-term, visionary research projects, we are often asked to explain the relevance of our work. The question of "just how does playing with robots deliver benefits to the company, its stakeholders, and customers?" is not uncommon. Our opinion is that by solving the complex challenges required to make a vehicle capable of autonomous driving, we will simultaneously enable and accelerate the development of technologies that will eventually be found on future accident mitigation and avoidance systems throughout the industry.

As we will discuss later in the text, we do not anticipate that mass-production, fully-autonomous automobiles (*"autonomobiles"*) will appear on the market anytime in the foreseeable near-future. However, we do envision a steady and systematic progression of improvements in automotive convenience, assistance, and safety features, wherein the vehicle becomes capable of assuming an ever-increasing

role in the shared human-machine driving experience. The general trend in the automotive industry is evolving from merely providing value-added driver information, to assisting with mundane driving tasks, warning and helping the driver to choose a safe solution when presented with potential hazards, and ultimately assuming increasing control when it becomes indisputably determined that the driver is incapable of avoiding or mitigating an imminent accident on their accord.

The bulleted list below provides a generic overview of several of the upcoming features being touted by automotive OEMs (Original Equipment Manufacturers), and which will undoubtedly benefit from many of the algorithms derived from autonomous vehicle research. Roughly speaking, the first four examples fall under the categories of braking, throttle, steering, and vehicle dynamics. The end point for each of these evolutionary systems converges with the others to provide a comprehensive collision mitigation strategy. The remaining items on the list involve infrastructure and human-machine interactions.

- Anti-lock brake systems (ABS), imminent collision warning, panic brake assist, collision mitigation by braking (CMbB)
- Cruise control, adaptive cruise control (ACC), ACC plus stop-and-go capability, urban cruise control (UCC – recognizes stop signs and traffic signals)
- Lane departure warning (LDW), lane keeping assistance (LKA), electronic power assist steering (EPAS), adaptive front steering (AFS), active steer (EPAS + AFS), emergency lane assist (ELA)
- Traction control, electronic stability control (ESC), roll stability control (RSC), active suspension
- Integration of pre-crash sensing with occupant protection equipment (airbags, seat belts, pretensioners)
- Collision mitigation by integrated braking, throttle, steering and vehicle dynamics control
- Vehicle to vehicle and infrastructure integration (VII / V-V), intelligent vehicle highway systems (IVHS)
- Blind spot detection, pedestrian detection, parking assistance, night vision
- Driver drowsiness and distraction monitoring
- Total accident avoidance / autonomous vehicle control

2.1 The Magnitude of the Problem

Although we are frequently reminded of the annual impact to society caused by vehicular accidents (NHTSA, 2005), it is nevertheless worth summarizing the statistics for the United States (world-wide statistics, although less well-documented, are more than an order of magnitude worse):

- 43,000 deaths
- 2.7 million injuries
- $230 billion in economic costs

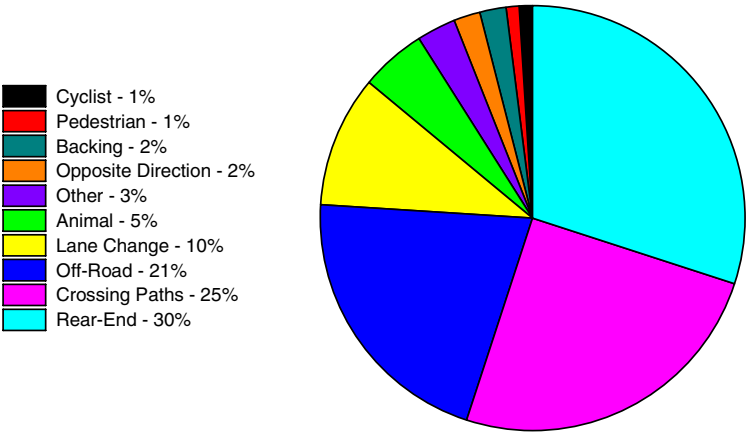


Fig. 2. Distribution of light vehicle crashes by type (Volpe, 2007).

To put this in perspective, this is roughly the equivalent of one major airline crash per day. While we somehow seem to accept the inevitability of traffic accidents, it is quite unlikely that the American public would board airplanes if they knew one of them would crash every day, killing all occupants aboard.

Figure 2 shows a distribution of light vehicle crashes by type, as compiled in "Pre-Crash Scenario Typology for Crash Avoidance Research" (Volpe, 2007). While many of these scenarios are well understood and safety systems are either in place or under development to address them, a noteworthy percentage of the crash scenarios are presently not well-covered by emerging near to mid-term automotive safety technologies. To expand a bit upon this point, the rear-end collision prevention scenario is the closest to production deployment, since a bulk of the hardware and algorithms required to achieve this application are logical extensions of Adaptive Cruise Control systems, which are presently available on selected OEM models. While the off-road and crossing path scenarios, which comprise the largest share of unresolved safety challenges, could benefit greatly from the results of the DGC research efforts, solutions to these issues could be developed even faster if technologies that were explicitly omitted from the DGCs were incorporated, notably vehicle-to-vehicle and vehicle-to-infrastructure communications, and enhanced roadway maps replete with ample metadata. We conclude that there are ample research opportunities for delivering improvements in automotive safety to members of our society, and. it can be argued that these gaps in coverage obviously represent scenarios that require a greater degree of situational awareness of the world around the vehicle, advanced sensors, and more sophisticated algorithms.

2.2 How Does the Reliability of a Present-Day Robot Compare with a Human?

Before we can make any sort of intelligent comments about the impending appearance of the autonomobiles that popular science writers have been promising

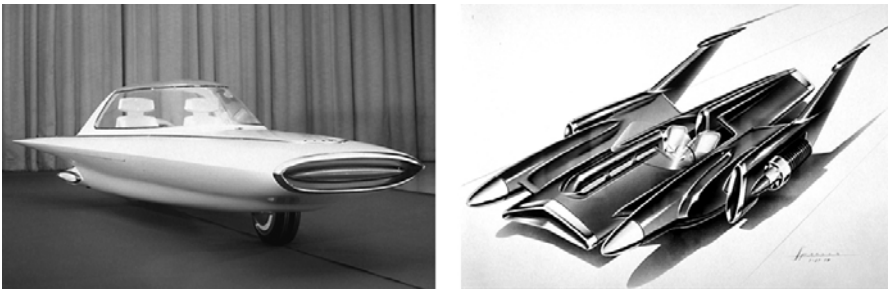


Fig. 3. Yesterday's "Cars of Tomorrow". A couple of fairly typical concept cars from the dawning of the Space Age – a photograph of the 1961 Ford Gyron (left) and pre-production sketch of the 1954 Ford *FX*-Atmos (right).

for the past half century (e.g. Figure 3); or for that matter, any number of the advanced active safety features enumerated in the bulleted list above, we need to have a rough idea of how the performance of present-day autonomous vehicles compare with human drivers. Given that the primary aim of this paper is not to scientifically analyze robotic safety, but rather to discuss our experiences at the 2007 DARPA Urban Challenge, we present instead a "back-of-the-envelope" calculation which serves to initiate the discussions to follow.

For small distances, we can assume the probability of a "failure" is approximately given by

$$P_f \approx \alpha \delta x ,$$

where α is the mean failure rate per distance, and δx is the incremental distance traveled. Conversely, the probability of "success" is given by:

$$P_s = 1 - P_f .$$

If we wish to repeatedly achieve success, and note that the total distance traveled is simply $x = n\,\delta x$, where n is the number of path segments (i.e., trials) then the overall probability becomes:

$$P_s = \left(1 - P_f\right)^n = \lim_{n \to \infty} \left(1 - \frac{\alpha x}{n}\right)^n = e^{-\alpha x} .$$

Of course, this derivation makes some simplifying assumptions, most notably that the events are randomly distributed and independent of one another so that they can be modeled as a Poisson process. If, for example, our vehicle operated flawlessly except whenever a train passed by, this equation would undoubtedly fail to hold. Nevertheless, it makes a good starting point for a provocative discussion.

If we examine the most recently published data from NHTSA's *"Traffic Safety Facts 2005"* (NHTSA, 2005), we observe that there are roughly:

3.0×10^{12} miles driven per year, and
2.0×10^{8} registered drivers, translating to
1.5×10^{4} average annual miles driven per person.

We also find there are:

14.5 fatalities per billion miles, and
900 injuries per billion miles.

Expressed in another manner, on average this is:

68.8 mean million miles between fatality, and
1.1 mean million miles between injury.

Using these NHSTA statistics and our derived equation, we can estimate the lifetime odds of suffering a vehicular injury or fatality. Given that life expectancy, at birth, for a middle-aged American was roughly 68 years (it is longer for children born today), let us for the sake of simplicity assume that the average person drives for 50 years of their lifetime. We now find that:

$$P_{injury} \approx 1 - e^{-(9.0\times10^{-7}\, i/mi)(1.5\times10^{4}\, mi/yr)(50\, yr)} = 49.1\% \text{, and}$$

$$P_{fatality} \approx 1 - e^{-(1.45\times10^{-8}\, f/mi)(1.5\times10^{4}\, mi/yr)(50\, yr)} = 1.1\% .$$

While most government sources state that the lifetime odds of being involved in a "serious" or "major" vehicle accident are about 1 in 3, they do not uniformly define what the metrics are for this categorization, nor do they comment on minor injuries. Fatalities, on the other hand, are not ambiguous, and numerous sources such as the National Safety Council (Mortality Classifications, 2004), put the lifetime odds of death due to injury in a vehicular accident at 1 in 84, or 1.2%. This is in excellent agreement with our simple estimation.

We can also apply this line of reasoning to the behavior of the robots in the DGCs. Although we did not explicitly acquire statistics on the mean time or distance between failures, particularly since we frequently tested and worked out one behavioral bug at a time, we did on occasion conduct some long, uninterrupted test runs, and during these outings we generally felt that in the absence of grossly anomalous road features, we could probably travel on the order of 100 miles between significant failures. Of course this value is highly variable, being dependent upon the type of road conditions being traversed. If one were lane tracking on a freeway, the results would likely be an order of magnitude better than those observed while negotiating dense urban landscapes. Nonetheless, this average value of $\alpha \sim 0.01$ mean failures per mile led one of our team members to speculate that our chances of completing the DGC2 course would be:

$$P_{succcess} \approx e^{-(0.01 f/mi)(132 mi)} = 26.7\% .$$

Now what makes this interesting is that we can turn this argument inside-out and consider the implications upon the other vehicles that participated in the finals of the last two events. If we solve for the mean failure rate α, we find

$$\alpha = -\frac{\ln(P_s)}{x}.$$

At the 2005 DGC2, 5 of the 23 finalists successfully finished the 132 mile course, while at the 2007 UCE, 6 of the 11 finalists finished (albeit with a few instances of helpful human intervention) a 60 mile course (DARPA, 2008). Let us see what this suggests.

$$\alpha_{DGC2} = -\frac{\ln\left(5/23\right)}{132 miles} = 0.012 \text{ mean failures per mile,}$$

$$\alpha_{UCE} = -\frac{\ln\left(6/11\right)}{60 miles} = 0.010 \text{ mean failures per mile.}$$

Remarkably, the observational values between the two DARPA finals events not only agree with one another, but also agree with the crude estimate we had formulated for our own vehicle. To be clear, we do not in any manner wish to suggest that the quality of any team's accomplishments was simply a matter of statistical fortune, nor do we want to make any scientific claims with regard to the accuracy of these observations, given the very small statistics and large assumptions made.

However, we do want to put the present-day capabilities of fully autonomous vehicles in perspective with regard to human abilities. In this context, it would appear that humans are 4 orders of magnitude better in preventing minor accidents, and perhaps 6 orders of magnitude better in avoiding fatal (mission ending) accidents. Therefore, the undeniable and key message is that the robotics community has abundant challenges yet to be solved before we see the advent of an autonomobile in each of our garages. To be fair, one can make a valid case for very specific applications in which semi-autonomous or autonomous vehicles would excel, either presently or in the near future, but bear in mind that these are far outside the scope of the general automotive driving problem as framed above. Examples of near-term applications would likely include a range missions from robotic mining operations to platooning of vehicles in highway "road trains".

2.3 Observations Regarding Customer Acceptance and Market Penetration

Despite the availability of a technology or its proven effectiveness, this is by no means a guarantee that the customer will actually use it. Although present-day

attitudes in society have tipped in favor of not only accepting, but demanding more safety applications and regulations, there is nonetheless a sizeable portion of the population who view the driving experience as something that should be unencumbered by assistance (or perceived intrusions) from the vehicle itself. For some, it's simply a matter of enjoying the freedom and thrill of driving, while for others, it can amount to a serious mistrust of technology, especially that which is not under their direct control. The reader will undoubtedly recall the public commotions made over the introduction of anti-lock brake systems, airbags, and electronic stability control. However, once drivers became sufficiently familiar with these features, their concerns tended to subside. On the other hand, it is yet to be seen how the public will react to convenience and safety features that employ semi- or fully-autonomous technologies.

To illustrate this point, consider the case of seat belts, arguably one of the simplest and most effective safety technologies invented. Although they were patented at essentially the same time as the automobile was invented (1885), 70 years passed before they were offered as optional equipment on production automobiles. Furthermore, it took a full century, and legislative actions beginning in the mid 1980s before customers began to use them in any significant numbers (refer to Figure 4). Even at the present time, nearly 20% of Americans still refuse to wear them (NHTSA, 2005).

Another issue confronting new technologies is the speed at which we can expect penetration into the marketplace. For some features, this is not a big concern, whereas for others, the utility of the technology depends upon universal acceptance. For example, it does not matter very much if one customer chooses not to purchase a premium sound system, but on the other hand, the entire traffic system would fail to work if headlamps were not a required feature. Many factors enter into how fast a new technology is implemented, including customer acceptance, availability, cost, regulatory requirements, etc.

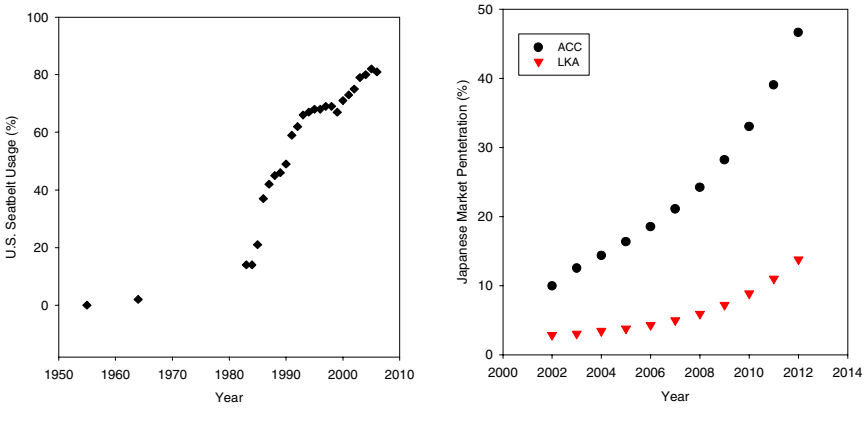


Fig. 4. Seat belt usage in the United States as a function of time (left). Note the dramatic rise in acceptance following legislation passed in 1984. Actual and projected Japanese market penetration for Adaptive Cruise Control (ACC) and Lane Keeping Assist (LKA) automotive driving features (right).

For the case of semi- or fully-autonomous driving, success will undoubtedly depend on having as many vehicles as possible equipped with these features. It has often been suggested that some portion of roadways, the federal interstate system for example, may in the future be regulated so as to exclude vehicles that do not conform to a uniform set of equipment requirements. Whereas having individual autonomous vehicles on the roadway may improve safety, having an entire fleet on the roadway could also increase vehicular density, improve throughput, and by platooning could also reduce drag and improve fuel economy.

While statistics regarding the customer "take rate" of optional features on a new car are often closely guarded by individual OEMs, Figure 4 above (Frost and Sullivan, 2005) also presents a prediction of what "fast adopters", such as the Japanese market, are expected to do with regard to two of the basic robotic building blocks leading toward autonomous operation – ACC and LKA. It should be noted that while present-day acceptance for ACC in Japan and parts of Europe is ~15%, in the U.S. it is a mere 4%. Perhaps more telling, the market penetration for mature systems that merely provide informational content (not vehicular control), such as GPS-based navigation devices, is still limited to a small subset of new vehicles.

2.4 Concluding Remarks Regarding Implementation of Autonomy in Production Vehicles

While the accident statistics and relative reliability of robots vs. human drivers clearly indicate ample opportunities for future autonomous research solutions, we have also illustrated that a number of factors, including customer acceptance and delivery speed to the marketplace will ultimately determine when fully-autonomous passenger vehicles become a commonplace reality. In this regard, the data refutes optimistic projections of production-level autonomobiles by Model Year (MY) 2015 as some have claimed (military and industrial robotic applications obviously constitute a separate conversation), but rather indicates a slower and continual progression of semi-autonomous driver support and safety features. In this regard, we feel that the field of active safety will ultimately lead the way, with robotics and autonomous vehicle research becoming the key enablers for these future systems.

3 Vehicle Architecture

One of the guiding principles in the design of our entry was to assure that the hardware, sensors, and algorithms developed not only addressed the mission goals of the DUC, but also offered a practical path toward future production integration into other active safety applications. In many phases of the project, this philosophy implied that the wisest choice was to employ production components, whereas there were certain aspects of the DUC for which no technical solution presently exists, requiring that risks be taken exploring novel sensing, hardware, and algorithms. With limited time and resources, success depended upon choosing the right balance

between status quo and innovation. In the following, we provide an overview of our vehicle hardware platform (Section 3.1), our perceptual sensing suite (Section 3.2), and our software architecture (Section 3.3) prior to our DARPA Site Visit.

3.1 Platform

Based upon the success of and lessons learned from the *Desert Tortoise* used in the DGC2, the 2005 model year Ford F250 again served as the platform for the IVS research efforts. This truck series has been extensively used as a rugged all-purpose workhorse, operating on roads of all sizes and surface conditions around the world, making it an ideal platform for a realistic autonomous mission – commercial or military. We fabricated two identical trucks for the DUC dubbed the "XAV-250"s, models A and T (XAV for eXperimental Autonomous Vehicle).

Overarching the theme of simplicity stated earlier, safety was always at the forefront of our efforts. Each of the by-wire systems (throttle, brakes, steering, and transmission) operated with redundant mechanical interfaces, enabling the XAV-250 to easily transition from human-controlled, street-legal to fully-autonomous operation by the flip of a switch. Occupant and bystander safety was further enhanced by the use of redundant, fully-independent, remote wireless e-Stop systems. When a pause or disable command was initiated, there were multiple means by which it was obeyed.

3.1.1 Vehicle Control Unit (VCU)

The VCU was implemented using a dSPACE AutoBox rapid control prototyping system, and was primarily used to coordinate the by-wire systems that control vehicle speed and steering. Commonly used by automotive OEMs and suppliers for algorithm development, it uses MATLAB/Simulink/Stateflow and Real-Time Workshop for automatic code generation. The VCU algorithms ran a position control algorithm to control steering wheel (hand wheel) position, a speed control algorithm that coordinated the throttle and brake systems, and also processed launch timing and pause requests.

3.1.2 Throttle-by-Wire

The throttle on the production engine communicates with the engine ECU (electronic control unit) via three voltage signals. A mixing circuit inserted between the throttle pedal and ECU essentially added the by-wire control commands to the outputs from the throttle pedal itself. A dedicated microprocessor with a watchdog timer was used for this interface; nominally the throttle interface commands are issued every 10ms by the VCU and if a valid command is not received within 83ms, the by-wire commands default to zero. This approach did not require modifications to the ECU or throttle pedal and has proven to be simple, safe, and effective.

3.1.3 Brake-by-Wire

Modern active safety systems such as ABS, AdvanceTrac, ESC, and RSC control brake pressures to increase vehicle stability. The XAV-250 has production-representative ESC/RSC hydraulic brake hardware and ECUs, with modified

software and hardware containing an additional CAN interface. These systems are regularly used by OEMs and suppliers to develop and tune vehicle stability algorithms. Through this interface, the VCU can command individual brake pressures at each corner of the vehicle with the full capability of the braking system. By using this production-proven hardware, our vehicle robustness and reliability has been very high.

The parking brake was automated to improve safety and durability, and to provide redundancy to the main braking system. Keeping with the desire to use production proven parts, a MY2000 Lincoln LS electronic parking brake actuator was used to activate the parking brake. To prevent overheating of the brake modulator, the vehicle was shifted into park whenever the vehicle was at zero speed for an extended period of time, and the hydraulic pressure was released on the main brakes. This allowed the brake valves and disks to cool when they were not needed. In the event of an unmanned e-stop disable, the parking brake was actuated by a relay circuit independent of all computing platforms.

3.1.4 Steer-by-Wire

The steering system was actuated by a permanent-magnet DC motor, chain-coupled to the upper steering column. The gear ratio (3.5:1) and inertia of the motor are low enough that manual operation is not affected when power is removed. By coupling to the upper steering column, the hydraulic power steering system aids the DC motor. For production vehicles, the maximum driver torque needed to command full power steering assist is ~10N-m. The motor can deliver this torque at approximately 20% of its maximum capacity. To drive it, an off-the-shelf OSMC H-bridge was used to create a high current 12V PWM signal. Using a 12V motor and drive electronics simplified the energy management and switching noise issues within the vehicle.

3.1.5 Shift-by-Wire

Transmission control was accomplished using a linear actuator to pull or push the transmission shift cable. The position was determined using the production sensor located inside the transmission housing. A microprocessor controls the linear actuator and provides the interface to the manual shift selection buttons and VCU. It also senses vehicle speed from the ECU and the vehicle state (run, disable, pause) from the e-stop control panel, and affords simple push-button manual operation with appropriate safety interlocks. This approach did not require any modification of the transmission or engine ECU and resulted in a robust actuation that provided the same retro-traverse capability that the *Desert Tortoise* had in the DGC2.

3.1.6 E-Stop Interface System

The e-stop interface system connects the radio controlled e-stop system and the various by-wire subsystems to control the operating modes of the vehicle. This system was implemented using automotive relays to increase reliability and reduce complexity. The interface has two modes, Development and Race, and two states, Run and Disable. In the development mode, used when a safety driver is in the vehicle, the disable state allows for full manual operation of the vehicle and the

run state provides full autonomous operation. In the race mode, the vehicle is unmanned and the by-wire systems conform to DGC rules. Pause requests are handled by the VCU to bring the vehicle to a gradual stop while still obeying steering commands. Communication faults were monitored within dSPACE and a signal to actuate an e-stop was issued when a fault was detected.

3.1.7 Navigation

Integration and support for the XAV-250 navigation system was provided by Honeywell, as described in previously published reports (IVS, 2006, 2007a, 2007b). The system incorporated a commercially available NovAtel GPS receiver using OmniSTAR HP satellite corrections, and was coupled with Honeywell's internally proprietary PING (Prototype Inertial Navigation Gyro) package. The PING has a high degree of flexibility, being capable of using a variety of IMUs (Inertial Measurement Units), and can input various state observations besides GPS, such as wheel speed odometry derived from the vehicle. The navigation algorithms from the PING exported position, attitude, acceleration and rotation rates at 100Hz, with the pose information remaining stable and accurate even during GPS outages of up to 20 or 30 minutes, owing to the high quality of ring laser gyroscopes and accelerometers used in their IMUs.

3.2 Sensors

Changes in the mission specifications, as well as the transition from a desert environment to an urban environment, required that many alterations be made to the sensing philosophy and implementation on the XAV-250. In the earlier DGC2, obstacle detection and path planning was essentially constrained to a narrow corridor closely aligned with a pre-defined dense set of GPS waypoints. In the DUC, the game became wide open, with sparse waypoints, large urban expanses such as parking lots and intersections, and more importantly, moving traffic surrounding the vehicle. Clearly this dictated a new solution, capable of sensing the dynamic world surrounding the vehicle, but also able to process the wealth of data in a real-time manner.

The introduction of the revolutionary Velodyne HDL-64E lidar seemed to be the answer to it all, with 64 laser beams, a huge field of view (FOV), 360° in azimuth and 26.5° in elevation, 120m range, and ability to capture one million range points per second at a refresh rate of up to 20Hz. However, this sensor had yet to be field tested, and with the known limitations associated with previous lidar systems, there was certainly some degree of hesitancy to rely too heavily upon this device. As such, it was apparent that we would need to provide a redundant set of coverage for the vehicle. This would not only offer added confirmation of obstacle detections, but would also serve to enhance the robustness of the system to the failure of a single sensor.

While one could take the approach of adding as many sensors as possible to the vehicle (and some teams did in fact do so), this adds an unwieldy burden to computational, electrical power, and thermal management requirements. A better solution was to determine where the XAV-250 most frequently needed to "look"

in order to satisfy the required DUC mission maneuvers (Figure 7), and to place sensors of the appropriate modality accordingly to fulfill these needs. We conducted a detailed study to optimize this problem, which included considerations such as:

- mission requirements of the DUC (GPS waypoint density, road network geometry, route re-planning, merging, passing, stopping at intersections, dealing with intermittent GPS reception)
- infrastructure (intersections, traffic circles, parking lots, dead-end roads)
- roadway design guidelines (line of sight requirements, minimum and maximum road grade and curvature, pavement vs. other roadway surfaces)
- highway driving rules (observance of lane markings, intersection precedence, spacing between vehicles)
- closing velocity of traffic (following time and look-ahead requirements)
- potential obstacles to be encountered (vehicles, curbs, buildings, signs, power-line poles, fences, concrete rails, construction obstacles, foliage)

Based upon our analysis, we initially settled upon a sensing suite that included (in addition to the Velodyne lidar) 8 Delphi Forewarn ACC radars, 4 Delphi dual-beam Back Up Aid (BUA) radars, 2 Cybernet cameras, 1 Mobileye camera, and 2 Riegl lidars. The overall sensor placement is shown in the truck collage in Figure 5, and is also described in more detail below.

Fig. 5. Collage of XAV-250 images revealing key elements of the external hardware. Truck overview (top left); protective frontal exoskeleton with 3 embedded in-line ACC radars and single centered BUA radar underneath (top center); rear view showing GPS mast on rooftop and protected pair of ACC and BUA radars (top right); side exoskeleton with ACC and BUA radars (bottom left); climate controlled and shock isolated box containing computing cluster (bottom center); and lidar and vision systems (bottom right),with Velodyne at apex, Riegls left and right, and cameras hidden within the Velodyne tower and behind the windshield glass.

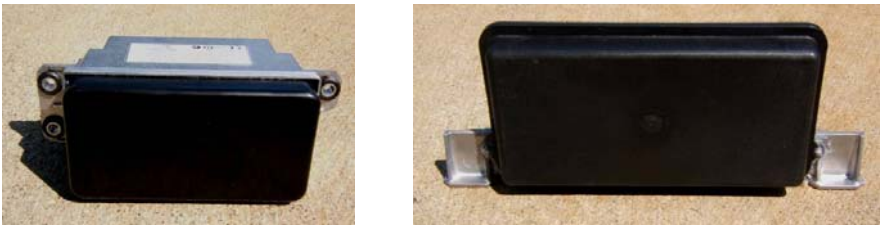


Fig. 6. Delphi radars: 76 GHz ACC (left) 24 GHz BUA (right).

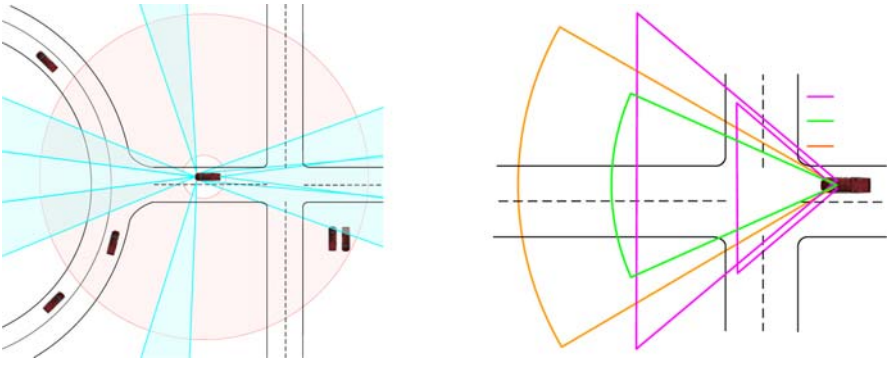


Fig. 7. Depiction of the long-range (left) and mid-range (right) sensing FOVs for typical roadways. The light blue triangles depict ACC radars, the shaded pink circle the Velodyne lidar. Note that the various depictions are not necessarily drawn to scale.

Delphi's 76 GHz ACC radars (Figure 6) are long-range, mechanically scanning radars that have been in production for years on Jaguars and Cadillacs. This radar senses targets at 150m with a 15° FOV and updates its tracking list every 100ms. A grouping of three of these sensors were coupled together to form a combined 45° forward FOV, enabling multi-lane, curved road coverage. Three more ACC units were strategically placed to also create a wide rearward FOV, with one on the rear bumper and two placed on the front outboard corners of the truck to provide rear and adjacent lane obstacle tracking. Additionally, two radars were mounted in a transverse direction on the front corners to provide coverage of obstacles at intersection zones.

In the forward center FOV, a Mobileye camera was used, primarily to provide confirmation to the ACC radars that targets were vehicles, but also to aid with lane tracking. The other cameras were dedicated to detect roadway lane markings and curbs, and to log visual data. The two Riegl LMS Q120 lidars each had an 80° FOV with very fine (0.02° per step) resolution. They were nominally set at distances of 12.5m and 25m, and used to detect curbs and aid the Velodyne

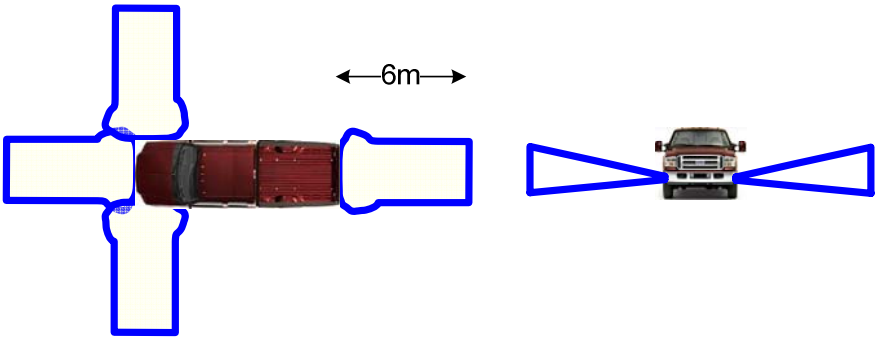


Fig. 8. Short-range sensor map illustrating the coverage of the Delphi BUAs.

in placing static obstacles in the map. For close proximity sensing scenarios, e.g., U-turns, backing up, maneuvering into a parking space, passing a close-by vehicle, etc., multiple Delphi BUA radars were used (Figure 8). These sensors have an effective range of ~5m; however, they do not return azimuth information on the target.

3.3 Software

3.3.1 DGC2: An Arbiter-Based Design

In the prior DARPA Grand Challenge (DGC2), the IVST team employed a situational dependent, arbitrating behavior-based solution (Figure 9) as reported in (IVST, 2005a, 2005b). The arbiter combined the outputs from the current set of contextual behaviors to produce a resultant steering and speed response. Each positive behavior would send its desired steering response to the arbiter in the form of a vector that represents the full range of steering, with the value at each vector element being the degree to which that specific behavior believes the vehicle should steer. Negative behaviors sent a response over the full steering range that represents steering directions not to go. The arbiter produced a weighted sum of all positive behaviors where the weight of a behavior is the product of an assigned relative weight of the behavior to other behaviors in a specific environmental context and the confidence of the behavior. The superposition of the sum of negative behaviors was used to cancel out hazardous directions and then the peak of the final response was used as the desired steering direction. Those behaviors that control speed, also provided a speed vector over the full steering range, where the value of a vector element represents the speed the behavior wants to go for that steering direction. The arbiter took the minimum speed over all behaviors for the chosen steering direction.

While this framework proved effective at NQE, during the final event a situation occurred were multiple competing behaviors with high confidence came into play at one time. These included, but were not necessarily limited to: vision-based road following (a very successful feature) locked onto the wide forward path while having an obstructed view toward the turn, a navigation request to turn

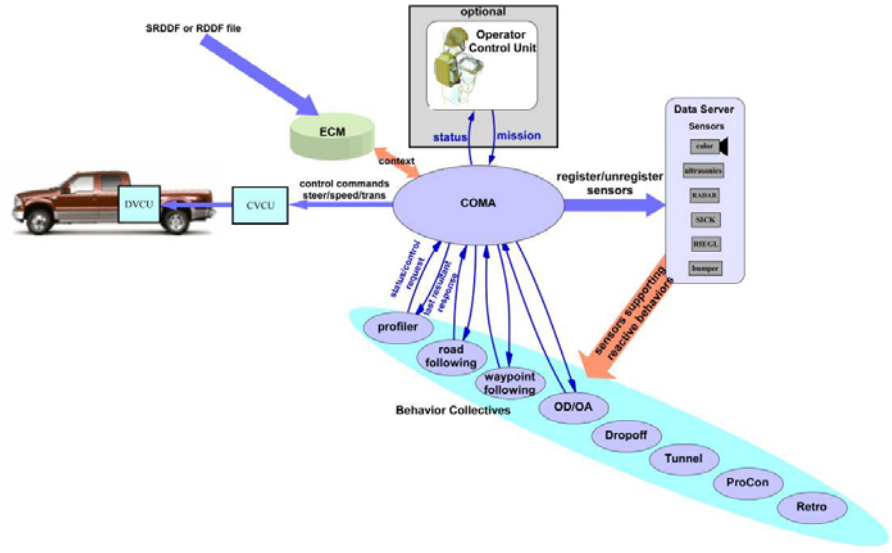


Fig. 9. Schematic of the arbiter software architecture used by IVST during DGC2 (IVST, 2005a, 2005b).

Fig. 10. Location on the DGC2 course where the IVST *Desert Tortoise* departed corridor boundaries. Multiple competing behaviors within the arbiter framework caused the vehicle to delay at a fork in the road. The correct route is the narrow hidden pathway on the left, as indicated by our team member standing where a DARPA photographer had been filming. Site-visit software architecture

sharply left onto a very narrow road, and apparent in-path obstacle detections of a pedestrian as well as from a cloud of dust created by the rapid deceleration to the intersection speed limit. Though the correct behavior did eventually prevail, the arbitration delay initiated a chain of events (Figure 10) that led to a minor departure from the course, and ultimately the team's disqualification. Learning from this lesson, the IVS team decided to pursue a map-based sensor fusion strategy for the DUC that was neither situational dependent nor arbitrated.

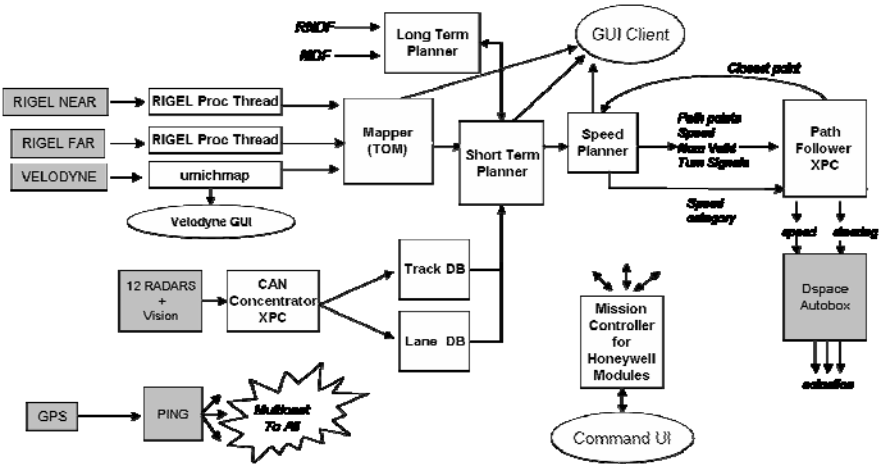


Fig. 11. Schematic drawing of the software architecture employed on the IVS vehicle at the 2007 DARPA Site Visit. White boxes indicate major software components and gray boxes embedded hardware.

3.3.2 Site-Visit Software Architecture

At the time of the DUC DARPA Site Visit, the bulk of the XAV-250 software was provided by Honeywell, and is described in detail in previously published reports (IVS 2006, 2007a, 2007b). We briefly review some key elements of the architecture, as depicted in the Figure 11.

Mission controller: The function of the mission controller is to encapsulate the high-level logic needed to accomplish a mission and to distribute that information to the other components in the system. Each software element is responsible for producing certain events which can trigger state changes and react to changes in state.

Mapper: The One Map (TOM) accepts classification data (unseen, empty, road, line, curb, obstacle) from each sensor, fuses it spatially and temporally, and provides a single map that contains the probabilities of each classification per sensor per cell. Updates to the map are asynchronous per sensor. The map is comprised of 1024×1024, $0.25\text{m} \times 0.25\text{m}$ grid cells, centered on the vehicle. The map is implemented as a doubly circular buffer (or torus) and TOM scrolls this map with the movement of the truck (determined from the PING). Each cell contains every sensor's estimated probability of the classification as being unseen, empty, road, line, curb or obstacle. A fusion algorithm is run at 10Hz across the center 512×512 cells ($128\text{m} \times 128\text{m}$) to derive a single probability of classification for each cell. TOM feeds this data to both the short term planner and to the Graphical User Interface (GUI).

Long term planner: The long term planner is responsible for determining the route the truck will take through its mission. It reads the Route Network Definition File (RNDF) and Mission Definition File (MDF), then based upon the

GPS position of the truck, determines where it is on the route network and creates a high-level plan consisting of a sequence of RNDF waypoints to reach the checkpoints specified by the MDF. This plan is devised using Dijkstra's Shortest Path Algorithm, with edge weights in the constructed graph determined based on a variety of factors, including estimated distance, speed limit specified in the MDF, and whether or not the system has recently experienced that road to be blocked. The long term planner provides the planned route to the short term planner.

Short term planner: The short term planner takes as input the position of the truck, the next goal from the long term planner, and the fused map from TOM, and produces a path to that next goal. The planning problem is formulated as a gradient descent in a Laplacian potential field.

Speed planner: The speed planner takes as input the obstacle map and the path computed by the short term planner and calculates safe speed along that path by considering a number of issues, including intersecting tracks, proximity to obstacles, curvature, speed limits and zones.

Path follower: The path follower takes as input the position, heading and speed of the truck, and a list of path points from the speed planner, and calculates a goal point on the desired path just ahead of the front axle using a vehicle model. The position and curvature of the path at the goal point is used to calculate a steering wheel position command which is passed to the vehicle control unit (VCU), implemented in a dSPACE AutoBox.

4 Transition to the IVS/MIT Vehicle Architecture

An internal assessment of the state of the project was conducted after the IVS team took a few weeks to digest the results of the DARPA Site Visit. Although the demonstrated functionality was sufficient to satisfactorily complete all the required Site-Visit milestones, it had become obvious that there were a number of problems with our approach that would preclude completing the final system development on schedule. These issues included unexpected delays introduced by both hardware and software development, and were compounded by team staffing limitations. While we will not elaborate on the details, we will point out a couple of examples to help the reader understand our subsequent and seemingly radical shift in plans. At the time of the Site Visit, our middle-ware employed a field-based, Laplacian path planner, which had been demonstrated in other robotic applications (IVS 2007a and references therein), notably with Unmanned Air Vehicles (UAVs). In the context of the DUC, successful implementation of the Laplacian planner required the inclusion of pseudo obstacles to prevent "undesired optimal" paths; a simple example being a 4-way intersection. Without painted lane markings existing within the intersection itself, the "un-aided" Laplacian planner would calculate the best path as one passing directly through the center of the intersection, obviously causing the vehicle to depart its lane. While these limitations could be overcome in principle, in practice, the myriad of topological possibilities made this algorithmic approach time consuming and cumbersome.

Also of major concern at the time of Site Visit were infrequent, but significant positional errors exported by the INS. Although the PING IMU was undeniably orders of magnitude more sensitive than the commercial units employed by most of the other teams, this also resulted in complications with tuning its prototype software (Kalman Filter parameters) to accommodate a commercial GPS input as one of the state estimators. Given that the functionality of virtually everything on the vehicle relied on having accurate pose information, Honeywell focused its efforts on this system, and it was decided that additional external collaborative resources would be solicited to complete final system development on schedule.

MIT was a logical candidate in this regard, owing largely to the pre-existing, well-established Ford-MIT Research Alliance. A fair number of projects falling under this umbrella included students and professors whom were also part of MIT's DUC team. Furthermore, the teams had been in contact beforehand, as Ford had provided them prior support, including helping with the acquisition of their Land Rover LR3 platform. As a side benefit, expanded contact between the teams would allow both sides to assess options for future joint autonomous vehicle research.

By mid-August, an agreement in principle had been made to work together. After clearing this proposal with DARPA, an implementation plan was devised. Although IVS had an existing set of code, albeit with gaps, it was instantly apparent that it would be far quicker to simply migrate to the MIT middle-ware code (Leonard, 2008), than it would be to try to merge disparate pieces of software. On the positive side, the MIT software was already successfully running on their platform, and the code structure was generic enough to incorporate most sensor types into their mapper. On the negative side, we would be dealing with a vast amount of code for which we had no inner knowledge, the transition would require a large rip up of hardware and software architecture, and we would no longer have a second identical truck for development.

4.1 Undeployed IVS Sensor Technology

Another one of the negatives of transitioning to the MIT architecture was that several novel sensing systems that were being developed by IVS (and partners) had to be put on hold so that all personnel resources could be reallocated to ensure successful completion of the ambitious software transition. The remainder of this section illustrates a few examples, not only to point out that they could have been migrated to the MIT platform given enough time, but also because they are still under consideration and/or development, and moreover there may be useful new information for the reader (particularly concerning results from the Velodyne lidar effort).

4.1.1 Lane Detection

Redundant vision systems were under development to aid in the fault tolerance of lane detection. A customized Mobileye vision system with a primary focal point of 40m and a Cybernet vision system with a primary focal point of 25m were in development prior to Site Visit. Each system was capable of detecting lane markings and vision gradient differences between the roadway and side of the road.

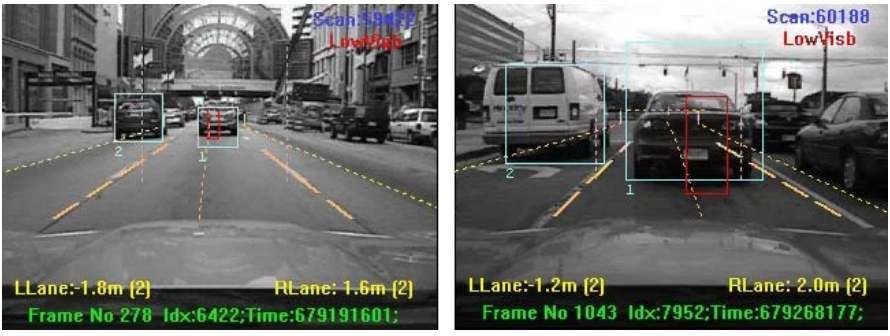


Fig. 12. Delphi/Mobileye radar/vision fusion with lane detection and in-path targets.

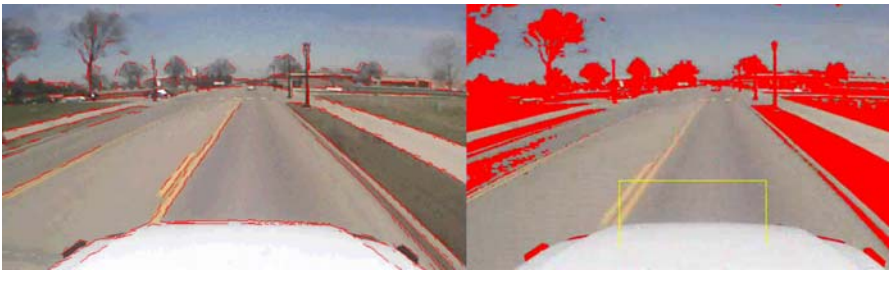


Fig. 13. Cybernet vision system roadway and traversability detection example.

Figure 12 is an example of urban driving data acquired from the Delphi/Mobileye fused radar and vision application taken on a surrogate vehicle using a production intent system comprised of a single radar and camera. The XAV-250 implementation was being developed to use three radars, effectively increasing the radar FOV for forward object detection by a factor of three. Although the vision system has a fairly wide FOV, it was initially intended to only confirm radar targets from the center channel. Figure 13 illustrates two alpha-version feature-extraction applications derived from the Cybernet vision system as actually installed on the XAV-250. The algorithm running on the left identifies lane markings and other sharp-edged features, such as curbs and sidewalks. The algorithm on the right searches for traversable surfaces, based upon contrasts in color and texture, and is heavily influenced by the sample roadway immediately in front of the vehicle (yellow box).

4.1.2 Velodyne Processing

The Velodyne HDL-64E was the primary perceptual sensor used by team IVS, both before and after the transition to the MIT code base. It provided 360° FOV situational awareness and was used for both static and dynamic obstacle detection and tracking, curb and berm detection, and preliminary results also suggested

painted-line lane detection on asphalt was possible via the intensity channel. The HDL-64E operates on the premise that instead of a single laser firing through a rotating mirror, 64 lasers are mounted on upper and lower blocks of 32 lasers each and the entire unit spins. This design allows for 64 separate lasers to each fire thousands of times per second, providing far more data points per second and a much richer point cloud than conventional *"pushbroom"* lidar designs. Each laser/detector pair is precisely aligned at predetermined vertical angles, resulting in an effective 26.8° vertical FOV. By spinning the entire unit at speeds up to 900rpm (15Hz), a 360° FOV is achieved (resulting in 1 million 3D points per second).

Sampling characteristics: Each Velodyne data frame consists of 12 "shots" and is acquired over a period of 384μs. This data frame is packetized and transmitted over the network via UDP at a fixed rate. To properly decode and transform points into the world-frame requires compensating for the rotational effect of the unit, as all of the lasers within a block are not fired coincidentally. Each shot consists of data from one block of 32 lasers; however, the Velodyne only fires 4 lasers at a time with a 4μs lapse between firings during collection of a full shot. Therefore, 32 lasers per block divided by 4 lasers per firing yields 8 firings per shot, with a total elapsed time of 8×4μs = 32μs (thus 32μs×12 shots = 384μs to acquire 1 data frame.) This means that per shot, the head actually spins a finite amount in yaw while acquiring one shot's worth of data. The reported yaw angle for each shot is the sampled encoder yaw at the time the *first* group of 4 lasers are fired, so that in actuality, the yaw changes by $\delta yaw = spin\_rate \times 4\mu s$ between the groups of 4 firings, such that on the 8th firing the unit has spun by $spin\_rate \times 28\mu s$. For example, on a unit rotating at 10Hz (i.e., 600rpm), this would amount to 0.1° of motion intra-shot, which at a range of 100m would result in 17.6cm of lateral displacement.

Coordinate transformation: Figure 14 illustrates the HDL-64E sensor geometry used for coordinate frame decomposition. The unit is mechanically actuated to spin in a clockwise direction about the sensor's vertical z-axis, z_s. Encoder yaw, Ψ, is measured with 0.01° resolution with respect to the base and positive in the

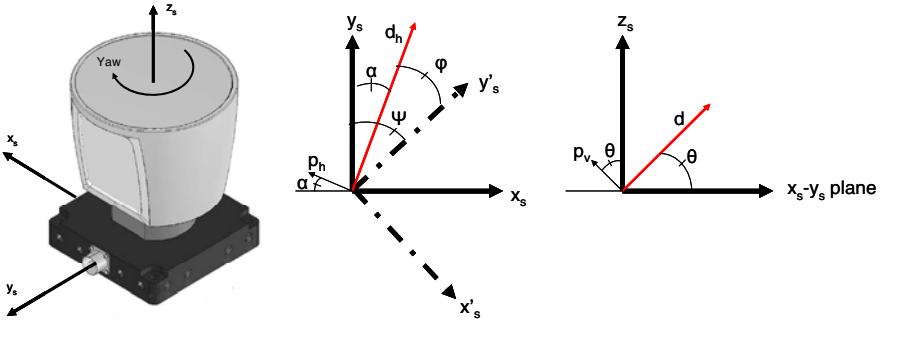


Fig. 14. Velodyne HDL-64E sensor coordinate frame used for decomposition (laser beam depicted in red.)

direction shown. Each laser is individually calibrated and parameterized by its azimuth, φ, and elevation, θ, angle (measured with respect to the rotating x'_s-y'_s sensor head coordinate frame) and by two parallax offsets, p_v and p_h, (measured orthogonal to the laser axis), which account for the non-coaxial laser/detector optics. Thus, a time-of-flight range return can be mapped to a 3D point in the sensor coordinate frame as:

$$\begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} = d \underbrace{\begin{bmatrix} \cos\theta\sin\alpha \\ \cos\theta\cos\alpha \\ \sin\theta \end{bmatrix}}_{\hat{e}} + \underbrace{p_v \begin{bmatrix} -\sin\theta\sin\alpha \\ -\sin\theta\cos\alpha \\ \cos\theta \end{bmatrix} + p_h \begin{bmatrix} -\cos\alpha \\ \sin\alpha \\ 0 \end{bmatrix}}_{\vec{p}} \tag{1}$$

where d is the measured range and $\alpha=\Psi-\varphi$. For real-time computation, this mapping can be pre-cached into a look-up-table indexed by laser ID, i, and encoder yaw (i.e., a 64 by 3600 LUT) so that only three multiplies and three additions are needed per laser for decoding to the sensor frame:

$$\vec{x}_s = d\ \hat{e}(i,\psi) + \vec{p}(i,\psi)\,. \tag{2}$$

Points can then subsequently be mapped to the world-frame based upon vehicle pose. In our system, navigation updates were provided at a rate of 100Hz, therefore, we causally interpolated vehicle pose to the timestamp of each shot using forward Euler integration with a constant velocity kinematic model.

Site-visit technology – UMichMap: An independent alliance between Ford and the University of Michigan was responsible for developing UMichMap, the software package that interfaced and processed the raw data from the HDL-64E. At the time of the Site Visit this software comprised two well tested algorithms: obstacle/traversable area detection and lane marking detection. A third algorithm, dynamic obstacle tracking, was undergoing alpha testing. All algorithms classified 0.25m grid cells within a sliding tessellated map that was output to a global map for fusion with other sensor maps.

Obstacle detection involved firstly eliminating overhead features (bridges, trees, etc.) that did not obstruct the safe passage of the vehicle. An object within a cell was then classified as an obstacle if it vertically exceeded an above-ground-plane threshold of 0.25m and there was confirmational evidence from neighboring cells. During the demonstration runs at the actual Site Visit, this threshold was temporarily increased to 1m due to a ranging problem in the presence of retro-reflective tape (discussed below). Preliminary results of lane detection were positive. The algorithm was based upon thresholding the Velodyne intensity channel returns and then fitting a contour to the remaining data points using a RANSAC (Fischler, 1981) framework. Simply speaking, if the return amplitude ranged from a value of 200 up to the maximum of 255, it was considered to be a

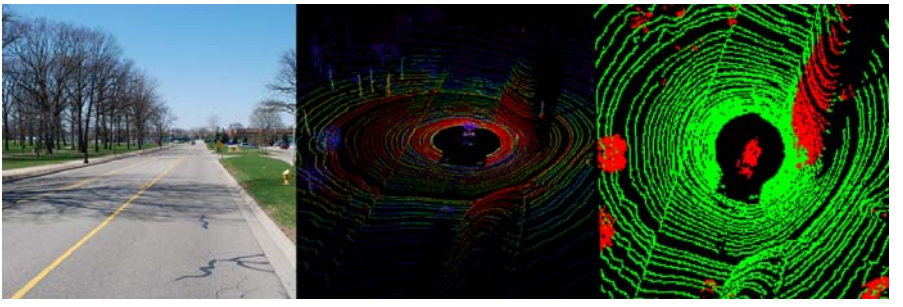


Fig. 15. UMichMap Velodyne LIDAR interface.

candidate lane marking (for comparison, typical range intensities for asphalt were well below a value of 100). Of those remaining thresholded points, those with a sufficient RANSAC model consensus were deemed actual lanes. Lastly, a graphical user interface using GLUT (OpenGL Utility Toolkit) was developed for real-time display, data log playback, analysis, and calibration.

Figure 15 is a composite image illustrating how data captured from the Velodyne is processed in the UMichMap system architecture. The photo on the left is a typical road segment at Ford's Dearborn campus. The center image depicts a single data frame captured from the Velodyne HDL-64E, with each LIDAR beam color-coded by the intensity of the return. One million range measurements per second are transformed into Earth frame coordinates, used to map the ground plane, and determine where the XAV-250 can physically drive. This "driveability" index (one of several exported attribute features) is shown on the right for each 0.25m x 0.25m cell surrounding the vehicle. It should be noted that as the vehicle moved, the driveability map would rapidly fill in as the laser beams swept the entire region in front of the truck.

Site-visit lesson – lidar issues associated with highly reflective materials: The software developed for the Velodyne lidar proved to be a huge success, especially after incorporating the latest firmware upgrades that were designed to correct some vexing hardware issues. However, one item that had yet to be perfected, and for that matter is still unresolved, involves the intensity channel information from each of the 64 beams in the HDL-64E model. The last firmware upgrade enabled our unit to detect obstacles to beyond 100m, and when our vehicle was parked anywhere within the Site Visit course, we were easily able to detect all of the painted lines.

Furthermore, the center of the lines registered in our map to within one pixel or less of where they were calculated to be from the geometry of the course as determined by the GPS survey markers (for the record, it is not known how accurately the lane marking tape was actually laid with respect to these points).

Fig. 16. Michigan Proving Grounds vehicle dynamics test Area, precision steering course, and Site Visit Course. Lane markings for the Site Visit course were laid out using reflective highway construction tape. The intersection area contained in the red rectangle is discussed in the next figure.

Surprisingly, when we processed this data through the obstacle detection algorithm, we saw phantom obstacles appearing on the lines, growing in size with distance from the Velodyne. A brief discussion of lidar is needed in order to explain this effect. Most lidar manufacturers calculate the range based upon the time at which the back-scattered (returned) beam intensity reaches its maximum. Under normal circumstances, this would physically correspond to the brightest spot of the projected beam, typically the center of the dispersed spot. The problem is that laser returns from highly reflective materials, such as retro-reflecting paint found on traffic signs (in our case, road construction tape) are orders of magnitude brighter than from normal materials. As such, the return will immediately saturate the detector, and the peak signal will occur at the time when the beam first strikes the reflector, and not necessarily at the center of the beam spot. As such, the circular sweeping beam from the Velodyne produces a zigzag discontinuity in the range measurements upon crossing a retro-reflecting line. Just prior to and after striking the line, the range is correct. However, when the beam first strikes the line, the perceived range is too short, and at the point where the beam exits the line, the range is perceived as too long. The magnitude of the discontinuity increases with distance owing to the divergence of the projected beam, and over the span of the Site Visit course, could exceed 0.5m. The implications for obstacle detection algorithms are obvious.

There were at least three methods we considered to deal with this. One was to simply leave the lines as real obstacles in the map. The problem with this approach is that the planner needed to ignore lines when passing stalled vehicles, and additional logic would have been required to establish that these were in fact lines. Additionally, declaring them as physical objects would slightly narrow the lane widths (by at least 2 pixels, each 0.25m wide), something we could ill afford

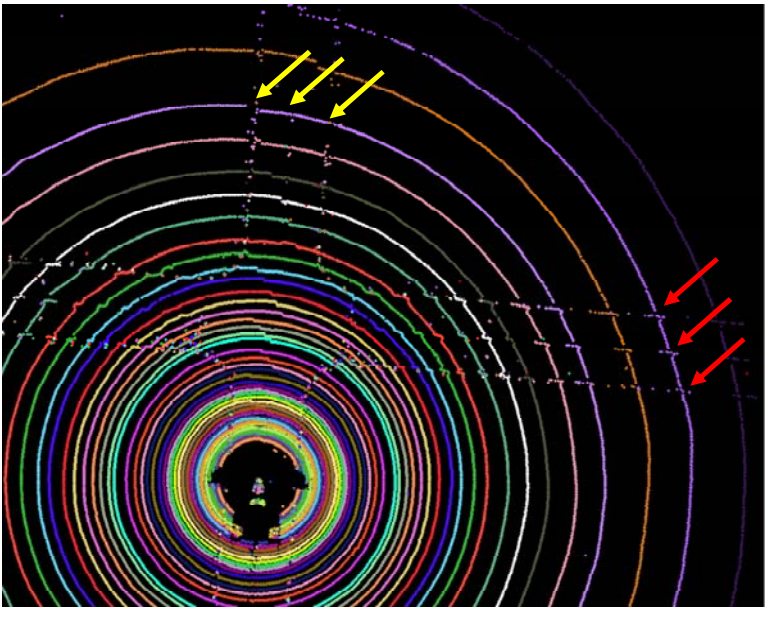


Fig. 17. We observed large range variations as the beams swept over road lane marking tape at our Site Visit course. These delta ranges ended up causing our obstacle detection algorithm to put spurious pixels on the map in these locations. Note the 4-fold symmetry in the lane "zigzag" about the lateral and longitudinal axes. The red arrows highlight that this effect is more pronounced as the laser incidence angle to the line increases; the yellow arrows show that this effect is less so when the sweeping angle of incidence is small.

with a vehicle the size of an F250. Secondly, we had characterized the intensity of laser returns for each of the 64 beams when scattering off asphalt (Figure 19); hence we could easily identify the lines from the pavement. While we did in fact demonstrate this method as means to easily discriminate lane markings, it required more logic to implement than the temporary solution we settled upon – simply raising the threshold for what was declared an obstacle.

Interestingly, a somewhat similar phenomenon was observed while the vehicle was parked in the garage at our Site Visit location, which had a smooth cement floor. A gaping hole appeared in the range map in front of the vehicle. Upon further investigation, we noted a large retro-reflecting sign on the wall several meters in front of the vehicle. The back-scattered laser return from the floor was obviously far weaker than the forward-scattered signal off the floor and back again from the bright sign. The sign was thus observed twice – once in the correct location by the higher elevation beams which hit it directly, and secondly "beneath the floor" by the beams that reflected off the cement and then struck the sign. Unexpected artifacts of this nature plague nearly all sensors, and as such explains the need for redundant sensing systems and conformational data in production

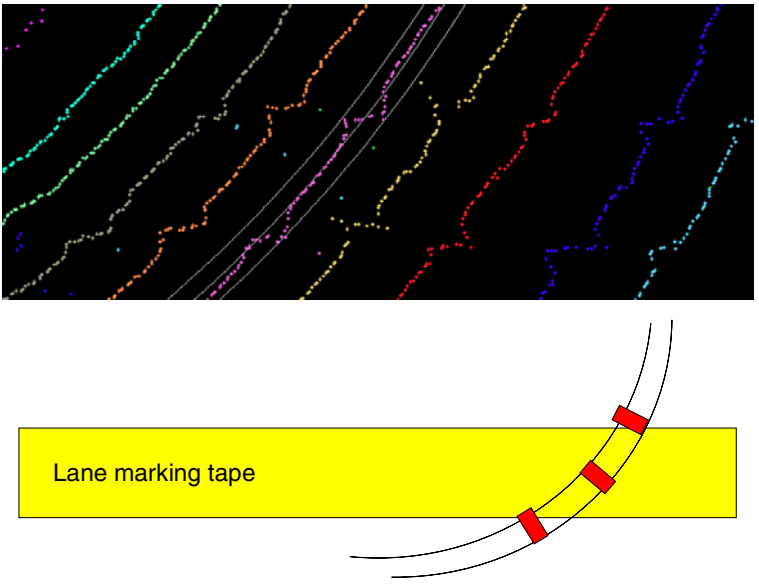


Fig. 18. Screen capture (top) of some of the beams crossing a single piece of lane marking tape, ~10cm wide. The lilac colored one is laser #0, and in our configuration falls on the ground at the 20m horizontal range. The concentric circles are ± 0.25m from the nominal arc everywhere else. Conceptually, a simple minded explanation (bottom) would be provided by the following observation – if the maximum intensity in a range sample return defines the range, then we could see something like that which occurs in the real data

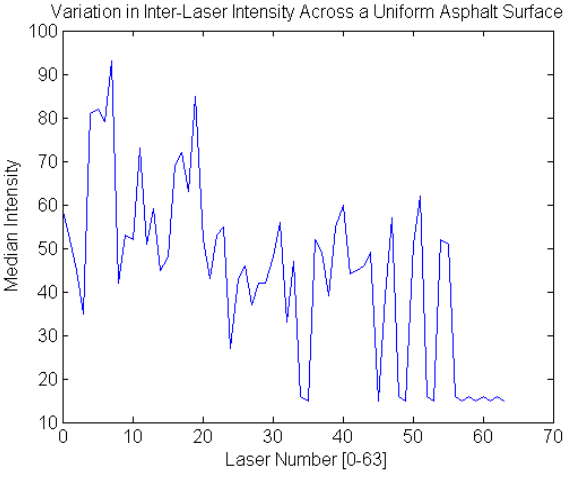


Fig. 19. Inter-laser intensity variation within a single scan across a uniform asphalt surface.

products like automobiles. While we were concerned that this might be an issue in the race, as far as we could ascertain, traffic signs and license plates were never aligned so as to produce these ghost obstacles.

4.2 Making the XAV-250 "Look" Like MIT's LR3

At the start of the transition to the MIT architecture (Figure 20), the MIT team members were actively engaged with tasks to add advanced navigation traffic capabilities to their platform. To minimize distractions to their efforts, it was decided to change the XAV-250 actuation, infrastructure, and sensing suite to match MIT's LR3 as closely as possible. With these alterations, it was also necessary for Ford to write several software and interface modules, as illustrated by the schematic in Figure 21. Some of the notable major changes are listed below:

- In the Site Visit configuration of the IVS vehicles, the compute cluster used a mixture of Advantech and Dell servers. The Dell servers were smaller and had higher-speed CPUs, while the physically larger Advantech servers had increased external I/O card capacity and were compatible with the MathWorks xPC rapid prototyping software. To increase computational power, the Advantech computers were removed and the Dell computers from both IVS vehicles were combined into the race vehicle, resulting in a compute cluster with 24 compute cores. Although less than the 40 cores used by MIT, it was sufficient to run their core software with our reduced sensor set.
- With the removal of the Advantech computers, the CAN concentrator was replaced with an array of EthCAN modules. These modules were based upon a Keil software evaluation board (model MCB2370) using an ARM9 processor and were programmed to pass CAN messages to the compute cluster via Ethernet. Each module supported two CAN networks. For each CAN message received, the EthCAN module would transmit the CAN header information and data bits using one Ethernet packet. Similarly, an Ethernet message could be sent to the EthCAN module and it would repackage the information to produce one CAN message. The EthCAN array was used to interface the radars and VCU (dSPACE AutoBox) to the main compute cluster. It should be noted that the MIT software architecture does not take advantage of the pre-processing that resides within the ACC radar units (e.g. closest in path target identification). Relying solely on raw radar data, the MIT development team created their own radar processing software. The reader is directed to the MIT documentation (Leonard, 2008) for details related to this data processing.
- The EthCAN modules and the PING had difficulties supporting high speed Ethernet traffic. In the final configuration, two additional Ethernet switches were added to form low (10Mb/s), medium (100Mb/s) and high speed (1Gb/s) networks.

- In the ADU command interface, control messages sent from the MIT core software were repackaged and sent via CAN messaging to a dSPACE AutoBox for by-wire execution. An EthCAN module performed the conversion between Ethernet and the AutoBox CAN network. The AutoBox contained the VCU software which controlled the low-level functions of the by-wire systems, and monitored signals for fault detection. Hardware-based monitoring was also implemented if the CAN connections were broken or the network failed. If a hardware, software or out-of-range signal was detected, an emergency stop was requested. In a similar fashion, the vehicle by-wire states were sent back to the main controller module running in the MIT core software for state information and for fault monitoring.
- To avoid camera interface issues, the team decided the quickest way to implement the MIT lane detection algorithms on the IVS vehicle would be to add 3 roof mounted Point Grey Firefly MV cameras. The number of cameras was limited by computational capability. Unfortunately, the Mobileye system was abandoned due to its incompatibility with the MIT software architecture. On a similar note, the Cybernet algorithms, which had been operational well in advance of Site Visit, were also never integrated.
- In contrast to the LR3, the XAV-250 was equipped with Delphi BUA radars which gave nearby obstacle information out to a range of five meters. This improved reliability in detecting low-lying, close-by obstacles, which fell within the Velodyne vehicle shadow. In order to take advantage of the BUA units, Ford developed a sensor interface function that allowed BUA data to be processed by the MIT software. This software function first read the BUA messages from the CAN bus in real time and then transformed the range returns into map coordinates based on the BUA calibration and vehicle pose within the map. If sufficient returns had accumulated in a particular location, that position, with the inclusion of a dilatational radius, was classified as a high-weighted obstacle in the map.
- The LR3 incorporated a large number of SICK line scanning lidars, which have a nominal resolution of 1° per step. The XAV-250 used two high resolution Riegl lidars for the same "pushbroom" functionality, acquiring range returns at 0.02° per step. The Riegl interface function generically repackaged the data into laser scan packets that the MIT software could use.
- The MIT software expected input from an Applanix POS LV 220 INS, thus an emulator was written by Ford to pass the Honeywell PING data in the same format.

Much of the effort to adapt the MIT software to the IVS platform was spent changing calibrations concerning vehicle parameters and sensor reference locations. In most cases, the changes could be made rather easily via calibration files; however, in some instances, these values were hard-coded constants that

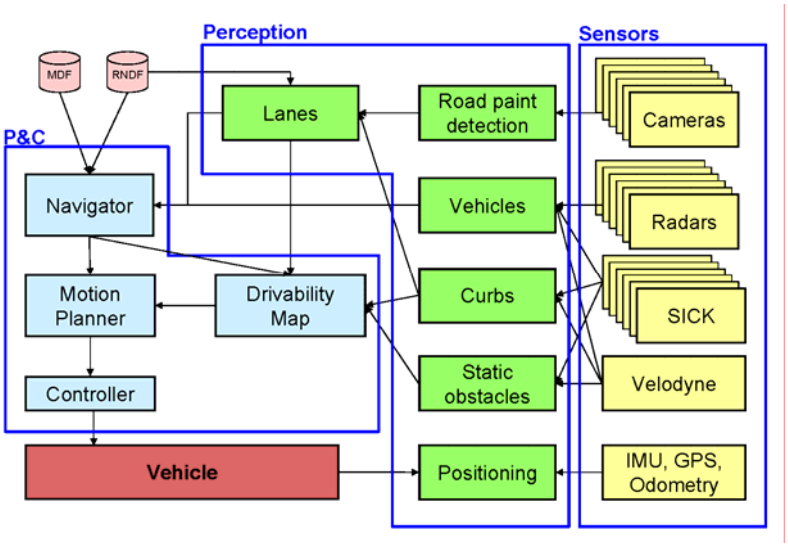


Fig. 20. Schematic drawing of the MIT base code software architecture (Leonard, 2008).

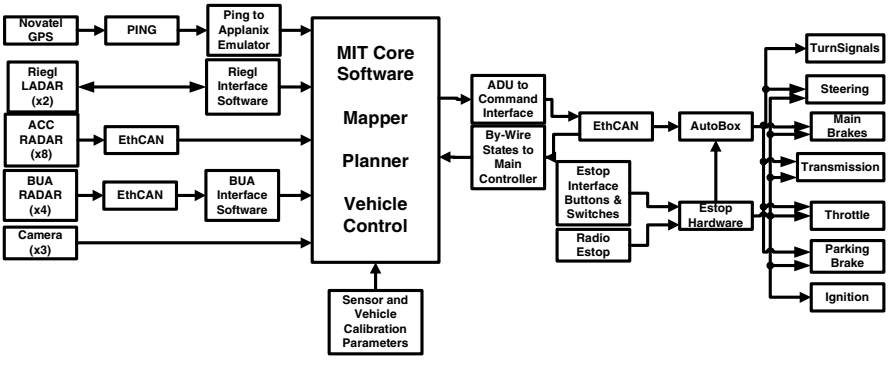


Fig. 21. Schematic drawing of the software architecture employed on the IVS vehicle after the transition to the MIT code.

needed to be identified and changed within the code itself. Within three weeks after deciding to reconfigure the XAV-250, testing began using the MIT software and toolset.

Once testing was underway, it was determined that some additional physical, electrical and software changes were needed to accommodate the "denser" computing cluster, including the installation of larger battery capacity, redistribution of electrical loads between the front and rear electrical systems, and redirection of the cooling air flow in the rear environmental computer enclosure.

5 Performance Analysis

5.1 Testing at El Toro with MIT

MIT team members visited Dearborn in early October, with the primary objective to help fine-tune the parameters which are used by the vehicle prediction model portion of the planner code. Prior to this time, we were having limited success in operating their code on our vehicle. However, once this exercise was complete, we quickly thereafter were able to demonstrate a variety of autonomous behaviors on the XAV-250, many of which exhibited peculiarities such as MIT was reporting from their LR3. It was at this point that the potential utility of collaborative testing was fully realized, and MIT suggested that we join them at a test facility on the El Toro Marine Corps base. With some last minute alterations to our schedule, we were able to divert the truck to southern California and achieved approximately a week of joint testing prior to NQE.

Testing with two different robotic vehicles on the course at the same time proved to be very productive, with both teams learning from each other. MIT had been at El Toro for a couple of weeks prior to our arrival and had constructed a RNDF of the road network, as well as a variety of MDFs. On our first attempt at their course, we had serious difficulties staying in our lane due to a constant bias in position. We had witnessed this before, watching a number of elite teams exhibit this behavior at the first two Grand Challenges. The problem was obvious to us – MIT used an Applanix INS, which by default exports coordinates in the NAD83 datum, whereas our system used the WGS84 datum, the same as DARPA uses. In southern California, these happen to differ by approximately 1.5m, or roughly half a lane width. After a code fix by MIT and a translation of coordinates in the RNDF, we were soon driving robotically past one another without issues.

With some cooperative help from MIT, we were able to successfully demonstrate operational capability of each of the sensors and processes (specific task algorithms, such as curb detection) that the MIT code would support on our platform. A highlight of the testing was the ability to validate all of the intersection precedence scenarios with both robots and traffic vehicles involved. Numerous consecutive runs were made to assure consistent behavior, and the success of this effort was later apparent at NQE and UCE, where (to our knowledge) neither MIT nor IVS ever made a traffic-related driving error. The only real downside of traveling to El Toro was the interruption caused by the Los Angeles wildfires, which shortened our available test time and introduced some hardware problems associated with the fallout of very fine ash.

As a final note, we would like to clarify that when IVS and MIT finished testing at El Toro, there was a code split and no further technical interaction occurred between the teams until after the race. We felt strongly that there should be no advantage afforded to either team, relative to the field of contenders, based upon any prior knowledge gained while undergoing testing at NQE.

5.2 NQE and UCE

At various points during the NQE and/or UCE we successfully demonstrated each of the sensor modalities and software algorithms that were capable of being supported by the MIT code. As it turned out, it was not always possible to operate the full sensing and software suite simultaneously, and as such, in the end we converged upon the simplest stable configuration possible. This consisted of GPS waypoint following, the Velodyne lidar, and a small set of Delphi ACC radars, notably including the front forward unit. Throughout our vehicle evaluation on the NQE sites, and during additional testing, we encountered and solved numerous problems, both with the hardware and software. There were, however, some bugs for which no near-term solution existed, and therefore this impacted what we could reliably run. Even though some of the observed anomalies occurred on a rare basis and we could have likely operated more of our system, we chose not to, as we did not understand the root causes, and moreover because the same functionality could be obtained with a simpler solution. To reiterate, although some sensors were not used for autonomous decision making, the sensor hardware itself was operational, and in many cases data from these systems was recorded for later re-simulation studies.

5.2.1 NQE – Area C, the "Belt Buckle"

Our first test session occurred in Area C, referred to by many as the "belt buckle". This test was presumably designed to evaluate navigation, intersection logic and traffic precedence, and route re-planning when presented with a blocked path.

For both of our runs in Area C, we demonstrated flawless execution of intersection logic and traffic precedence, with the truck stopping precisely at the painted stop lines and no errors occurring in any of the intersection scenarios. In each run, we accurately navigated the course via GPS waypoint tracking and by utilizing the curb detection process fed from both types of lidar – the two pushbroom Riegls and the Velodyne. Although video was recorded for data logging purposes, the lane detection process was not employed for navigational guidance. This decision was made primarily in light of the abundance of curbs and the faintness of painted lines in this neighborhood, but to some extent by issues we were experiencing with our vision hardware and software. At the time of the first run, we had not had an opportunity to validate the camera calibration (following transport from El Toro), and on the second run, we did not want to introduce changes to what had what had worked successfully the first time.

For us, the route re-planning proved to be among the most problematic of any of the NQE tasks, and we would spend the majority of our remaining free time at the event in an effort to solve this issue. On our first run in Area C, the truck was issued a DARPA pause command after it attempted to circumnavigate the road blockage by cutting between the construction barrels and a large tree in the adjacent yard. We were allowed to re-position the vehicle on the road, and on the second attempt it executed a U-turn; however, it did so by departing the street again and completing the maneuver on a lawn. When the truck reached the second

Fig. 22. Area C – intersection logic and dynamic re-planning. Aerial photo of Area C course (top); red arrow indicates the intersection depicted in figures below. XAV-250 successfully exhibits advanced traffic behavior (bottom left). MIT viewer rendering of the RNDF and tracked cars at the intersection (bottom right).

blockage constructed from stop signs on gated arms, it immediately recognized them as obstacles, stopped for several seconds, and again appeared as if it was going to seek a route around them. Coincidentally, our test time ran out at this moment, so the final outcome in this scenario remains uncertain.

The behavior exhibited here was initially unexpected, and to explain it, requires a discussion of the MIT planner code. The blockage occurred immediately in front of a checkpoint on what was essentially a circular loop. In this case, it is topologically impossible, following highway rules of the road, for the vehicle to re-plan a route reaching the checkpoint. The only way this point could be achieved would be to a priori assume that the same blockage would exist after the vehicle had circled the course in the opposite direction; and furthermore, that the vehicle could and would execute another U-turn so as to be in the correct lane in the correct orientation. Unfortunately, this scenario was overlooked in the planning logic, and had not been discovered in our limited prior testing, as there had always been an intersecting roadway, providing a valid alternative route, between blockages and the next checkpoint.

5.2.2 NQE – Area A, the "Circles of Death"

The second test site we visited was Area A, a place we personally called the "Circles of Death". By our account, there were about a dozen traffic cars traveling bi-directionally around an outer oval of roughly 300m in circumference. Our task was to make left hand loops on a subsection of the course, yielding through gaps in the oncoming traffic, and pulling out from a stop sign onto a very narrow section of road abutted on one side by a solid concrete barrier.

We felt that our performance here was very good, despite receiving a fair number of honks from the traffic vehicles. In each of our two attempts, we completed more than a dozen laps, many of them perfect. Our robot always recognized traffic and precedence and never came close to hitting a moving object. The difficulty in this task stemmed not only from the density of traffic, but also from our interpretation of the rules, in which we assumed a requirement of allotting several vehicle lengths of spacing between our truck and the traffic vehicles. Given that our vehicle is roughly 7m in length and we had to allow spacing in both directions when exiting the stop sign, this left very little room or time for traffic gaps along the 60m stretch we were merging onto. Adding to the challenge was the 9 seconds it took for the XAV-250 to accelerate from a stop to 10mph through the tight 90° turn. There were quite a number of cases in which our vehicle would determine it was clear to go, and then balk as a traffic vehicle would make the far turn from the transverse segment of the course onto the segment we were merging onto. Although the Velodyne was identifying the traffic vehicles across the entire span of the test site, our intersection algorithm did not classify these vehicles as obstacles of concern and assign them a track file until they entered a pre-defined zone surrounding the intersection itself.

On our first attempt at the course, we used GPS waypoint tracking and vision-based detection of lane markings for navigation, and left the curb detection algorithm off. There were few curbs on the loop, and we had also recently discovered the potential for a software bug to appear when the vision lane tracker and curb detection algorithm reported conflicting estimates of the lane width or position. This approach worked on all but one lap. In that instance, a group of traffic vehicles passed very close to the F250 on the corner furthest from the stop sign, the truck took the corner perhaps a meter wide, and struck or drove atop the curb. We were generally very pleased, however, with the registration of our sensors (lidar and radar) with respect to ground truth, as the vehicle maintained the center of its lane while tracking very close to the concrete barriers.

We did make a change to our sensing strategy on the second run, however. In this case, we chose to run the curb detection algorithm and turn the vision-based lane tracking off. This decision was prompted in part by some bugs that had cropped up in the vision software, as well as by performing a re-simulation of our previous run. This simulation showed that the curb function provided excellent guidance along the concrete barriers and on the curbs on the back side of the loop, with the GPS waypoints on the other two segments being sufficient to easily maintain lane centers. With this configuration, all loops were navigationally perfect.

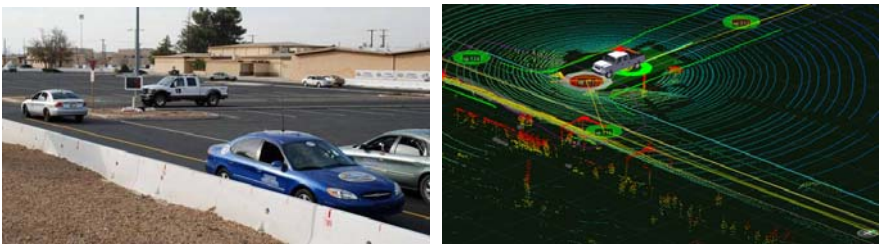


Fig. 23. Area A "circles of death." XAV-250 waits to merge into traffic (left). MIT viewer showing the RNDF course with obstacles derived from the radars and Velodyne point cloud (right).

5.2.3 NQE – Area B, the "Puzzle"

Area B was the last of the sites in our testing order. This area was very representative of the final event, with the exception that it had an abundance of stalled vehicles littering the puzzle-patterned streets. It was also very similar to the roads we tested on at El Toro, and hence we expected to perform well. To the contrary, we experienced our worst outings on this portion of the course, with most of our failures the result of some bewildering hardware failures, software bugs and a bit of misfortune. We were not able to fully complete the mission on either of the runs, and as a result, ended up having scant data to analyze in order to prepare for the finals. It would have been greatly beneficial to our team if DARPA had provided a practice site resembling this area.

On our first attempt at Area B, we choose to navigate using the same sensor set successfully employed in Area C – GPS waypoint tracking and curb detection derived from lidar. Absent from our sensing suite was the forward facing radar cluster, as we had been observing a fair number of false detects from ground clutter on these radars, and we feared this would be a bigger concern in Area B than in Areas A and C. It should also be reminded that the MIT code utilized the raw radar signals, as opposed to filtered output that normally is exported from the Delphi production radars. Given that the Velodyne lidar had been reliably detecting all obstacles of interest, this was deemed an acceptable solution. The vehicle demonstrated the ability to execute parking maneuvers, navigate around stalled obstacles, and again performed without flaw at intersections and in the presence of traffic. However, we did experience occasional issues with the curb detection algorithm, and in some cases missed curbs that existed, resulting in behaviors such as cutting corners between waypoints. In other cases, we misclassified obstacles that were not curbs as curbs, resulting in failsafe modes being invoked, in which case curbs could again be ignored and similar driving behaviors would result. After jumping a curb about midway through the course, the Velodyne process crashed due to the physical unseating of a memory chip. Presented with no Velodyne obstacles in the map, the vehicle drifted at idle speed and was DARPA paused just as it was about to strike a plastic banner located at the interior of one of the puzzle pieces. Although it is highly improbable we would

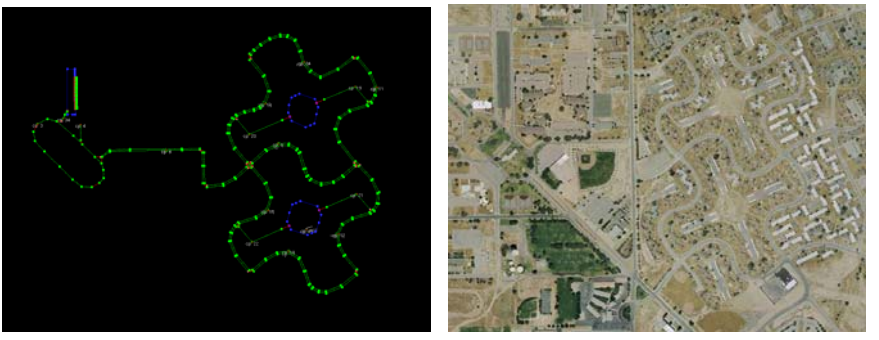


Fig. 24. Site B course RNDF and aerial photo.

have completed the course without the Velodyne, had the front radar been on, we would have likely detected the banner and stopped. Shortly thereafter, we re-aligned the front radars to a higher projection angle above the ground plane. In order to preclude the possibility of getting a false return from an overhead object, such as an overpass or low hanging tree branch, we filtered the data to reject returns from ranges in excess of ~20m.

Our second attempt at Area B came after sitting for 7 hours in the sun on one of the hottest days of the event. We were scheduled to begin promptly at 0700; however, at each of the areas where we were tested, we were continually leapfrogged in the schedule by other teams in the field. Given that this was our last run before the finals, we decided to run all sensors and processes, including the vision lane tracking and lidar curb detection, having felt we had resolved the conflict between these two processes, and wanting to acquire a complete data set. Upon launch, the vehicle proceeded nominally for a few hundred meters, then began to stutter, starting and stopping abruptly. DARPA immediately halted the test and sent us back to the start chute for another opportunity. Upon examining the data logs, it was found that we were flooding the computer network with traffic, and as a result navigation and pose messages were being periodically dropped, causing the stuttering motion observed in the vehicle. We terminated the lane detection function and re-launched the robot. It proceeded normally, until reaching a stop sign at an intersection exiting the interior of one of the puzzle pieces, and at this location a significant discontinuity in the ground plane was formed by the crown of the facing road and/or the rain gutter between the two perpendicular roads. This was perceived to be a potential curb, causing the planner to keep the vehicle halted while it attempted to resolve the situation. After a brief stoppage, the truck idled across the street and into a lawn. A DARPA pause command was ineffectual, and we were subsequently disabled. The data logs revealed that the brake controller module indicated an over temperature warning and refused to command brake pressure, which is consistent with the observed behavior with regard to the DARPA pause vs. disable (which commands the parking brake) commands.

At least three serious issues were raised from testing in this area:

Vision: Our vehicle employed three cameras, two forward-facing with a combined FOV of ~100°, and one center rearward facing. While MIT had significantly more visual coverage, their lane detection algorithm was designed to accept a variable number of camera inputs, and had been shown during testing at El Toro to work acceptably with our configuration when clear lane markings were present. However, during the NQE, we were unable to demonstrate reliable functionality from the vision system on our platform, and are presently not certain whether hardware or software contributed to these shortcomings.

Curb detection: The primary functions of the lidars were to detect obstacles and to determine the topography of the ground around the vehicle, which was used to determine traversable terrain, as well as to infer the existence of curbs. Generally speaking, the algorithms which perform these functions do so by making a comparison of the elevation changes or slopes of neighboring map grid cells. Declaring something an obstacle is much easier than declaring something a curb, especially in the case of the F250, where the ground clearance is more than 0.25m. On the other hand, curbs are often less than 0.10m in height relative to the surrounding terrain. The trick is to tune the thresholds in the software to maximize detection while minimizing false positives. An additional complication is that some features will correctly trigger a detection, yet not be an obstacle of concern. Examples of this would include speed bumps, or other ground discontinuities such as grated sewer covers, rain gutters, etc. At present, when this type of detection arises, the MIT code relies on failsafe modes to make forward progress. Given additional time to develop more sophisticated algorithms, and given redundant sensing corroboration, this problem could be addressed in other manners, but that was beyond the containable scope of the DUC effort.

While these false positives were infrequent, we seemed to be more sensitive to them than MIT's LR3, again presumably due to the differences between our platforms. While MIT operated far more lidars – which could provide a greater degree of redundancy – we feel that the issue was more likely related to the fidelity of the lidar data. MIT's pushbroom scanners, produced by SICK, sampled at 1° intervals, and their Velodyne, which they operated at 15Hz, sampled at 0.15° intervals. On the other hand, our pushbroom Riegls acquired data at 0.02° intervals, and our Velodyne, rotating at 10Hz, sampled at 0.09° intervals. All things being equal, our sensing set would be more prone to elevation noise with an algorithm that compares neighboring cells. Once we realized this, we went back and re-simulated all of our prior data sets, tuning the available parameters until we no longer saw these false positives. In the process, we recognized that the Velodyne alone was sufficient to detect curbs, and to avoid potential noise from the Riegls, we did not incorporate them in this algorithm in the UCE.

Brake Controls: The IVS vehicle braking system was implemented by the use of a brake by-wire system furnished by TRW, one of Ford's primary brake suppliers. A production level ABS module with the addition of custom hardware and software modifications allowed for independent dynamic braking of each wheel of the

vehicle with superior braking accuracy and increased braking bandwidth, as compared with brake pedal displacement devices such as employed by some of the competing vehicles. This module is capable of providing smooth control of vehicle velocities down to 0.5m/s. Another advantage of this system is its quick recovery time, which significantly enhances safety for development engineers occupying the vehicle during autonomous test runs. The main disadvantage of this prototype system was the inability to hold the vehicle at a complete stop for more than ten minutes. As with all production ABS systems, the control module's heat dissipation characteristics are primarily specified for intermittent use, and therefore for our application, the continuous braking utility of the ABS module was limited. To protect the brake module from potential physical damage under these continuous braking applications, an internal software integrator timer was employed.

If the brake module actually overheats and/or the internal software integrator times out, all primary braking for the IVS vehicle is lost, and the vehicle would begin to roll at the idle speed of the engine, with an e-stop pause command being ineffectual. A similar time out failure had occurred during Site Visit, and at that time, DARPA suggested shifting to park position and releasing the brakes during an extended pause condition. This suggestion was an idea we had also contemplated, but had not yet implemented due to time constraints. Once the Site Visit was over, we did follow this approach.

5.2.4 UCE – The Finals

When we started the truck on the morning of the race, one of the servers was inoperative. After some inspection, we discovered that the power supply to this server was not functional, and we had to remove all the servers in the rack to access it. Fortunately, we had a spare, and reassembled the system just in time to make it into the queue. While we were going through our checklist in preparation for the start, we discovered that the low-level controller was exporting bad data. This was quickly traced to a faulty connector, which had likely resulted from the pandemonium in replacing the power supply. This connection was fixed and the low-level controller subsequently seemed to be behaving correctly. Given that we were only two vehicles from the starting chute, and that it takes about 20 minutes for the INS to stabilize, we opted not to power down and perform a complete reboot of the system. This was a clear mistake, as corrupt data remained in the system, causing our steering controller to command a lock left turn upon launch. We were re-staged, and during this time, did reboot the system from start. The second launch proceeded with nominal behavior. This incident points out a clear problem which needs to be resolved before autonomous systems ever reach production product viability – as has been shown with numerous other systems, the customer is unwilling to wait for even a few seconds after ignition before driving, much less the minutes it presently takes for computers (and INS systems) to initialize or re-initialize.

Based upon the lessons we had learned during NQE, we decided to run UCE with the simplest stable configuration of sensors and algorithms possible. This consisted of GPS waypoint following, curb detection using only the Velodyne lidar, and obstacle detection using both lidar types and a small set of Delphi ACC

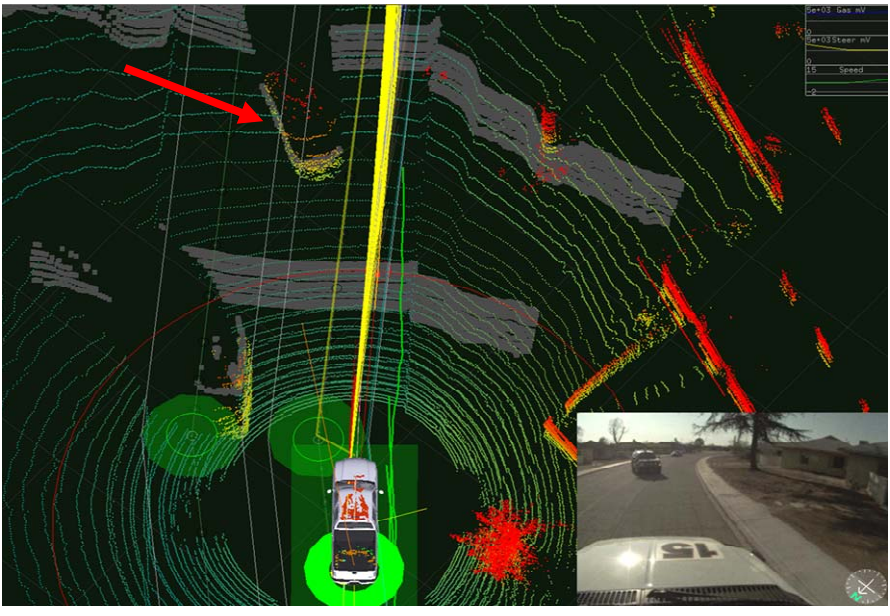


Fig. 25. Incident where the Cornell team tried to pass into XAV-250's lane requiring evasive maneuvering on XAV-250's part to avoid collision. The red arrow denotes the Cornell vehicle as seen in the Velodyne point cloud, while the overlaid camera image to the lower right clearly shows the Cornell team in our lane.

radars, notably including the front forward unit and the 90° intersection scanning units. Because we were somewhat handicapped by not being able to run the vision system (aside from data logging purposes), we did insert a limited number of additional waypoints into the RNDF in regions we deemed might present issues.

During the time our vehicle was operational, it navigated well, obeyed the rules of the road, passed numerous robots and traffic vehicles, displayed correct intersection logic and traffic precedence, and successfully demonstrated parking maneuvers. Furthermore, it exhibited intelligent behavior when presented with the scenario of an oncoming robot approaching us in the wrong lane, slowing down and taking evasive actions to avoid a collision (Figure 25).

The failure mode for our vehicle occurred when we again detected a false positive upon exiting the interior of one of the puzzle pieces. While at the stop sign between this small road and the main road, the curb detection process incorrectly perceived either the crown in the facing road or the sharp discontinuity formed by the rain gutter to be a potential curb (Figure 26). Under normal circumstances, the vehicle would have waited for a short period (nominally 90 sec) and invoked a failsafe mode which would have relaxed the curb constraint. However, following the difficulties we had with the topological conundrum in Area C, the timer for this process had been increased by an order of magnitude to

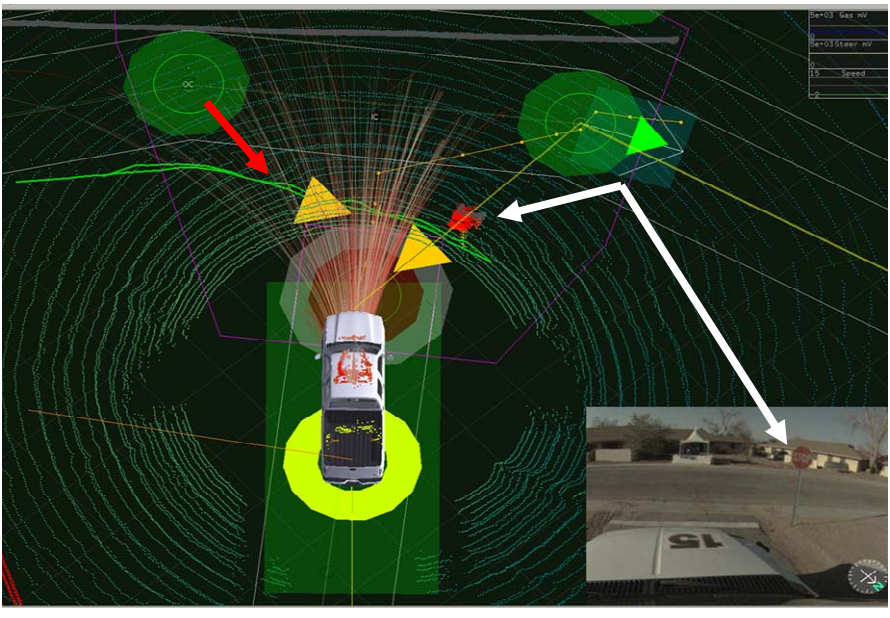


Fig. 26. Failure mode of the XAV-250 during the UCE. The red arrow indicates a false detect of an in-path curb at an intersection. For reference, the white arrow indicates the stop sign in both the Velodyne intensity channel and camera imagery.

rigidly enforce curb constraints while we evaluated a potential fix for this issue. Unfortunately, through oversight on our part, the timer had not been restored to its default value and we were subsequently and fairly disqualified for excessive delay on the course. When we rescued the truck, the planner was indicating a valid path, waiting for the failsafe timer to expire. While we can not say what would have happened for the remainder of the course, we do know that this oversight prevented us from ever finding out.

6 General Observations and Lessons Learned

This section presents, in no particular order, a variety of the remarks contributed by team members during the writing of the DARPA Final Report and this article. While these comments obviously pertain to our perception of the DUC experience, we suspect that many of these general observations and lessons learned will be shared by other teams as well.

- We expect that all teams will complain to some extent about having inadequate developmental and testing time between the announcement of the DUC program and the UCE. It is a very ambitious goal to create a test vehicle within a year, much less one that autonomously drives itself in simulated urban traffic. Complicating the challenge is the large number of

intermediate milestones. While we can certainly understand DARPA's need to assess interim performance, the Track A Funding Proposal, video submission, kick-off meeting, informal interim reports, Technical Paper, Site Visit, multiple revisions to rules and procedures, etc., are nevertheless distractions, especially for teams with few members.

- Many of us felt that the DARPA Site Visit and NQE did not adequately represent the final events at any of the three Grand Challenges. In some sense they are actually a bit of a detour – the Site Visit because it requires an integrated, fully-functional system too early in the development timeline, and the NQE because it demands performance objectives that are not actualized again in the Finals. From our discussions with other teams at the event, we found that a significant number had designed specifically for the Site Visit – often with surrogate platforms, sensors or algorithms – knowing in advance that they would operate different systems if they were invited to participate at NQE.
- From our perspective, we would encourage any potential future event to create mission goals that are both clearly defined and realistic. Conversely, we do understand the opposing perspective, in that specifying requirements too succinctly can result in less innovative solutions. While we felt DARPA did an excellent job of conveying goals at this Challenge, we also feel that they were not entirely representative of a practical mission application. Our assumption is that maps, with ample metadata, will exist for both automotive and military applications. Referring back to an example shown at the Washington briefing, it seems highly improbable to expect a robot to stop within 1m registration of a stop line at one intersection, when the neighboring intersection is completely devoid of GPS coordinates and requires the vehicle to execute a 90° turn through it. Corporate entrants, such as IVS, are driven by a production mindset demanding system reliability, redundancy and robustness, and as such are already prone to over-design for the Challenge, without the added burden of trying to correctly guess in advance what the metrics for success will be.
- Similarly, corporate teams are often disadvantaged with respect to universities or military contractors, wherein the metrics for success are very different. Universities can draw upon a vast pool of inexpensive, talented, and highly-motivated labor, and there is very little downside to not performing well, as they are after all, "just a bunch of students". On the other side of the coin, corporate teams must justify the high costs of (very long-range) internal research and development, and carefully weigh the potential rewards vs. the numerous risks, ranging from liability to negative publicity. Given that major corporations, and not universities, are ultimately going to deliver military and commercial hardware solutions, we would encourage DARPA to consider how to better engage their participation without making all but the winner appear to be a loser.
- Testing in a realistic environment is absolutely critical to uncovering system weaknesses ranging from flaws in logic to bugs in algorithms. A thousand laps in a parking lot is no match for a mere few blocks of urban

roadway. Unfortunately, finding safe and secure test facilities requires connections, time and money. The IVS team was fortunate to have tested at more than half a dozen locations prior to NQE, yet one of our most critical bugs was not realized until we attempted Area C. It was difficult to test potential fixes to this flaw, however, as the practice areas at NQE did not contain representative features of the UCE, one of the very few disappoints we had with DARPA's execution of this event. It would have also been useful if DARPA could have allowed teams a couple of days in which to attempt the courses after the UCE was complete, so as to close the loop on the learning process. Based upon our mutual testing with MIT prior to NQE, we are convinced that if DARPA could arrange for a common testing venue for all teams, autonomous ground vehicle technologies would advance at a much faster pace.

- One of the lessons we learned, and not necessarily by choice, was that the vehicle system does not need to be too complex to accomplish an amazing amount of autonomous behaviors. While we did drive several portions of the course with only the INS and Velodyne lidar, we would not necessarily advocate implementing a system without added redundancy. It should further be noted that we did not even come close to fully exploiting the capabilities of the lidar, particularly in light of the incomplete developmental work on the intensity channel data from the Velodyne HDL-64E. If this hardware/firmware were reliably functioning, one could essentially derive black and white vision simultaneously from the unit and apply the wealth of existing image processing algorithms to the data to significantly expand sensing capabilities. We are looking forward to pursuing this area of research in the near future.
- When the IVS team initially started testing the Velodyne lidar, we frequently lost GPS reception, and hence the INS pose information that was necessary for correcting sensor data in our map. Given our close working relationship with Velodyne, we were able to rapidly validate that the HDL-64E was indeed generating sufficient EMI to jam the electronics on our NovAtel GPS receiver. To solve this, it was deemed necessary to mount our GPS antennas above the Velodyne, and to use a choke-ring design which minimized multipath interference from below the antenna phase center. Without data it is impossible to prove, but we believe that many of the difficulties encountered by other teams during the pre-final practice sessions were due to EMI emanating from the many Velodyne units. There was some anecdotal evidence that other electronic devices could also interfere with our system, including 802.11 wireless communications from laptop computers in the test vehicle. Although it was not employed at NQE, our secondary e-stop system was known to fail if more than one 2.4GHz device (hand-held radios) was keyed simultaneously, something we actually encountered during Site Visit. In a similar vein, the hand-held radios used by DARPA during NQE/UCE were powerful enough to cause the Velodyne units to drop Ethernet packets (this was first observed by Stanford and later verified in the lab by Velodyne). If we are to allow the fate of the vehicle to rely on a

stack of electronics and not a human driver, it is clear that more care must be taken in the future to properly address EMI issues.

- During our pre-race testing, particularly when we were collaborating with MIT, we came to appreciate the importance and power of customized software toolsets. There were several notable tasks, which one team or another could do within minutes, while it would take the other team hours to accomplish. Lots of time can be expended laying waypoints on maps, creating RNDFs or visualizing data, to cite but a few examples. Perhaps DARPA could solicit contributions from the participating teams in this regard, and create a public domain repository of available tools, so that each subsequent effort is not slowed by these mundane tasks. On a similar note, we would like to extend kudos to DARPA for supplying aerial imagery of the Victorville facility in advance of NQE, and for allowing us to preview the UCE course prior to race day.
- An inspection of the entrants from the three Grand Challenges reveals that, with rare exception (most notably the first-generation Velodyne lidar), most of the hardware and sensors utilized were essentially off-the-shelf technologies. It is clear that the cutting edge of autonomous vehicle research really lies in the algorithms and software architecture. As such, the customized construction of a by-wire vehicle platform could be viewed as an unnecessary distraction, and an interesting future twist for a DARPA Challenge might be to outfit each of the teams with an identical platform and see what they could accomplish by virtue of innovative software alone. This places the competitors on even ground, and is somewhat akin to the DARPA PerceptOr program. (Of course, this is the converse of what IVS and MIT did this year, i.e. run common code on vastly different platforms.) Given the success of the Gray Team at the last Challenge and VTU at the DUC, (and with some biased self-promotion) we might suggest the Ford Hybrid Escape as a platform which is by-wire capable with minimal modifications.

7 Conclusion

In conclusion, we have demonstrated the successful operation of an autonomous vehicle capable of safely maneuvering in simulated urban driving conditions. Moreover, we have achieved this, to varying degrees of driving complexity, with the implementation of two very different computer and software architectures. Our switch to MIT's architecture, which included a substantial amount of hardware reconfiguration, was accomplished in a span of less than two months, and not only demonstrated the versatility of their code, but also our resolve and devotion to completing the mission. While we have only partially explored the bounds of what is possible via autonomous vehicle operations, we have learned a great deal and have ample reason for optimism. Although we have estimated, based upon our performance and that of the other contenders, that the capabilities of present-day robots are still orders of magnitude inferior to those of human drivers, we have witnessed a rapid progression in autonomous technologies, despite the relatively short developmental time afforded to the teams that participated in the Urban

Challenge. As such, we anticipate that this general trend will continue, and foresee that many of the lessons learned from this and similar endeavors will soon find their way into commercially available automotive safety features.

Acknowledgements

We would also like to thank our colleagues at the University of Michigan and at the Massachusetts Institute of Technology. Without the incredible support of these organizations, we would not have been able to complete our project goals. It is a testament to the quality of the MIT code that we were able to install it on a completely different platform, with a completely different sensor set, and demonstrate advanced autonomous driving behaviors within months of implementation. Together, we learned a great deal, and hopefully planted the seeds for many future collaborative efforts.

Finally, we would sincerely like to thank DARPA for conducting what we believe to be the best Challenge to date. It has been a privilege and honor to participate in this event. Although we dealt with an inordinate amount of adversity and ultimately may not have produced the results we had hoped for, we were nonetheless thrilled to once again make it all the way to race day. The understanding and support we received on behalf of the DARPA personnel no doubt contributed significantly to our success. We look forward to productive future interactions with DARPA as opportunities become available.

References

DARPA, various archived data and written accounts found on the Grand Challenge website (2008), http://darpa.mil/grandchallenge

Fischler, M., Bolles, R.: Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. Comm. Assoc. and Computing Machine 24, 381–390 (1981)

Frost, Sullivan: Japanese Passenger Car and Passive Safety, Systems Markets, Report #4B79-18 (2005)

Intelligent Vehicle Systems (IVS) Team Proposal for the DARPA Urban Challenge, Proposal for BAA 06-36, submitted by Honeywell Laboratories, June 26 (2006)

Intelligent Vehicle Systems (IVS) DARPA Urban Challenge Technical Paper, submitted on behalf of the Intelligent Vehicle Systems (IVS) Team by J. McBride, April 13 (2007a), http://www.darpa.mil/grandchallenge/TechPapers/Honeyell_IVS.pdf

McBride, J.: Intelligent Vehicle Systems (IVS) DARPA Urban Challenge Final Report, December 22 (2007b) (Submitted)

Klarquist, W., McBride, J.: Intelligent Vehicle Safety Technologies 1 – Final Technical Report, August 29 (2005a), http://www.darpa.mil/grandchallenge05/TechPapers/IVST.pdf (Submitted)

Intelligent Vehicle Safety Technologies (IVST) User Manual, Mark Rosenblum, PercepTek Robotics, Inc., November 11 (2005b)

Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., Koch, O., Kuwata, Y., Moore, D., Olson, E., Peters, S., Teo, J., Truax, R., Walter, M., Barrett, D., Epstein, A., Maheloni, K., Moyer, K., Jones, T., Buckley, R., Antone, M., Galejs, R., Krishnamurthy, S., Williams, J.: A perception driven autonomous urban vehicle. J. Field Robotics (2008) (Submitted) (Under Review)

NHTSA (National Highway Traffic Safety Administration, U.S. Department of Transportation), Traffic Safety Facts (2005), http://www-nrd.nhtsa.dot.gov/Pubs/TSF2005.PDF

Mortality Classifications, from the National Safety Council (2004), http://www.nsc.org/lrs/statinfo/odds.htm

Volpe, Pre-Crash Scenario Typology for Crash Avoidance Research, Volpe National Transportation Systems Center, Project Memorandum, DOT-VNTSC-NHTSA-06-02, DOT HS 810 767 (April 2007), http://www-nrd.nhtsa.dot.gov/departments/nrd-12/pubs_rev.html

TerraMax: Team Oshkosh Urban Robot

Yi-Liang Chen[1], Venkataraman Sundareswaran[1], Craig Anderson[1], Alberto Broggi[2], Paolo Grisleri[2], Pier Paolo Porta[2], Paolo Zani[2], and John Beck[3]

[1] Teledyne Scientific & Imaging, Thousand Oaks, CA
{ylchen,sundar,canderson}@teledyne.com
[2] VisLab - University of Parma, Parma, Italy
{broggi,grisleri,portap,zani}@ce.unipr.it
[3] Oshkosh Corporation, Oshkosh, WI
jbeck@oshkoshcorp.com

Abstract. Team Oshkosh, comprised of Oshkosh Corporation, Teledyne Scientific and Imaging Company, VisLab of the University of Parma, Ibeo Automotive Sensor GmbH, and Auburn University, participated in the DARPA Urban Challenge and was one of the eleven teams selected to compete in the final event. Through development, testing, and participation in the official events, we have experimented and demonstrated autonomous truck operations in (controlled) urban streets of California, Wisconsin, and Michigan under various climate and traffic conditions. In these experiments TerraMax™, a modified Medium Tactical Vehicle Replacement (MTVR) truck by Oshkosh Corporation, negotiated urban roads, intersections, and parking lots, and interacted with manned and unmanned traffic while observing traffic rules. We have accumulated valuable experience and lessons on autonomous truck operations in urban environments, particularly in the aspects of vehicle control, perception, mission planning, and autonomous behaviors which will have an impact on the further development of large-footprint autonomous ground vehicles for the military.

In this article, we describe the vehicle, the overall system architecture, the sensors and sensor processing, the mission planning system, and the autonomous behavioral controls implemented on TerraMax™. We discuss the performance of some notable autonomous behaviors of TerraMax and our experience in implementing these behaviors, and present results of the Urban Challenge National Qualification Event (NQE) tests and the Urban Challenge Final Event (UCFE). We conclude with a discussion of lessons learned from all of the above experience in working with a large robotic truck.

1 Introduction

Team Oshkosh entered the DARPA Urban Challenge with a large footprint robotic vehicle, TerraMax™, a modified Medium Tactical Vehicle Replacement (MTVR) truck. By leveraging our past experience and success in previous DARPA Challenges, the combined multi-faceted expertise of the team members, and the support of a DARPA Track A program award, we demonstrated various autonomous vehicle behaviors in urban environments with excellent performance, passed through many official tests at the National Qualification Event (NQE), and qualified for the Urban Challenge Final Event (UCFE). TerraMax completed the

M. Buehler et al. (Eds.): The DARPA Urban Challenge, STAR 56, pp. 595–622.
springerlink.com

first four sub-missions in Mission 1 of the UCFE before being stopped after a failure in the parking lot due to a software bug. We brought TerraMax to UCFE test site in Victorville in December 2007 where TerraMax completed successfully three missions totaling over 78 miles in 7 hours and 41 minutes.

Team Oshkosh is comprised of Oshkosh Corporation, Teledyne Scientific and Imaging Company, VisLab of the University of Parma, Ibeo Automotive Sensor GmbH, and Auburn University. Oshkosh provided the vehicle, program management, and overall design direction for the hardware, software and control systems. Oshkosh integrated all the electrical and mechanical components, and developed the low and mid-level vehicle control algorithms and software. Teledyne Scientific and Imaging Company developed the system architecture, mission and trajectory planning, and autonomous behavior generation and supervision. University of Parma's VisLab developed various vision capabilities. Ibeo Automotive Sensor GmbH provided software integration of the LIDAR system. Auburn University provided evaluation of the GPS/IMU package.

Although there are substantial hurdles that must be overcome in working with large vehicles such as TerraMax™, we feel that large autonomous vehicles are critical for enabling autonomy in military logistics operations. Team Oshkosh utilized a vehicle based on the U.S. Marine Corps MTVR which provides the majority of the logistics support for the Marine Corps. The intention is to optimize the autonomous system design such that the autonomy capability can be supplied in kit form. All design and program decisions were made considering not only the Urban Challenge requirements, but eventual fielding objectives as well.

Our vehicle was modified to optimize the control-by-wire systems in providing a superior low-level control performance based on lessons learned from the 2005 DARPA Grand Challenge (Braid, Broggi, & Schmiedel, 2006, Sundareswaran, Johnson, & Braid, 2006). Supported by a suite of carefully selected and military practical sensors and perception processing algorithms, our hierarchical state-based behavior engine provided a simple yet effective approach in generating the autonomous behaviors for urban operations. Through the development, testing and participation in official events, we have experimented and demonstrated autonomous truck operations in (controlled) urban streets of California, Wisconsin, and Michigan under various climate conditions. In these experiments, TerraMax negotiated urban roads, intersections, and parking lots, and interacted with manned and unmanned traffic while observing traffic rules.

In this article, we present our experience and lessons learned from autonomous truck operations in urban environments. In Section 2 we summarize the vehicle and hardware implementation. In Section 3 we present the overall system architecture and its modules. In Section 4 we describe TerraMax's sensor and perception processing. In Section 5 we present TerraMax's autonomous behavior generation and supervision approach. In Section 6 we discuss TerraMax's field performance and experience in the NQE and the UCFE. We comment on lessons learned in Section 7.

2 TerraMax: The Vehicle and Hardware Systems

2.1 Vehicle Overview

The TerraMax™ vehicle (see Figure 1) is a modified version of a standard Oshkosh Medium Tactical Vehicle Replacement (MTVR) Navy Tractor[1], which comes with a rear steering system as standard equipment. The MTVR platform was designed for and combat-tested by the U.S. Marine Corps. We converted the vehicle to a 4X4 version by removing the third axle and by shortening the frame rail and rear cargo bed. The TAK-4™ independent suspension allowed rear axle steering angles to be further enhanced to deliver curb to curb turning diameters of 42 feet, equivalent to the turning diameter of a sport utility vehicle. In addition to the enhancement of turning performance, Oshkosh developed and installed low-level controllers and actuators for "by-wire" braking, steering, and powertrain control. Commercial-off-the-shelf (COTS) computer hardware was selected and installed for the vision system and autonomous vehicle behavior functions.

2.2 Computing Hardware

We opted for ruggedized COTS computing platforms to address the computing needs of TerraMax. Two A-Plus Mobile A20-MC computers with Intel Core Duo processors running Windows XP Pro were used for autonomous vehicle behavior

Fig. 1. TerraMax: the vehicle.

[1] Oshkosh MTVR. http://www.oshkoshdefense.com/pdf/Oshkosh_MTVR_brochure_07.pdf.

generation and control. The four Vision PCs use SmallPC Core Duo computers running Linux Fedora. One PC is dedicated to each vision camera system (i.e. trinocular, close range stereo, rearview, and lateral). Low-level Vehicle Controller and Body Controller modules are customized Oshkosh Command Zone® embedded controllers and use the 68332 and HC12X processors, respectively. To meet our objectives of eventual fielding, all the computing hardware was housed in the storage space beneath the passenger seat.

2.3 Sensor Hardware

2.3.1 LIDAR Hardware

TerraMax™ incorporated a LIDAR system from Ibeo Automobile Sensor, GmbH that provides a 360° field of view with safety overlaps (see Figure 2). Two ALASCA XT laserscanners are positioned on the front corners of the vehicle and one ALASCA XT laserscanner is positioned in the center of the rear. Each laserscanner scans a 220° horizontal field. Outputs of the front scanners are fused at the low level; the rear system remained a separate system. The LIDAR system native software was modified to operate with our system architecture messaging schema.

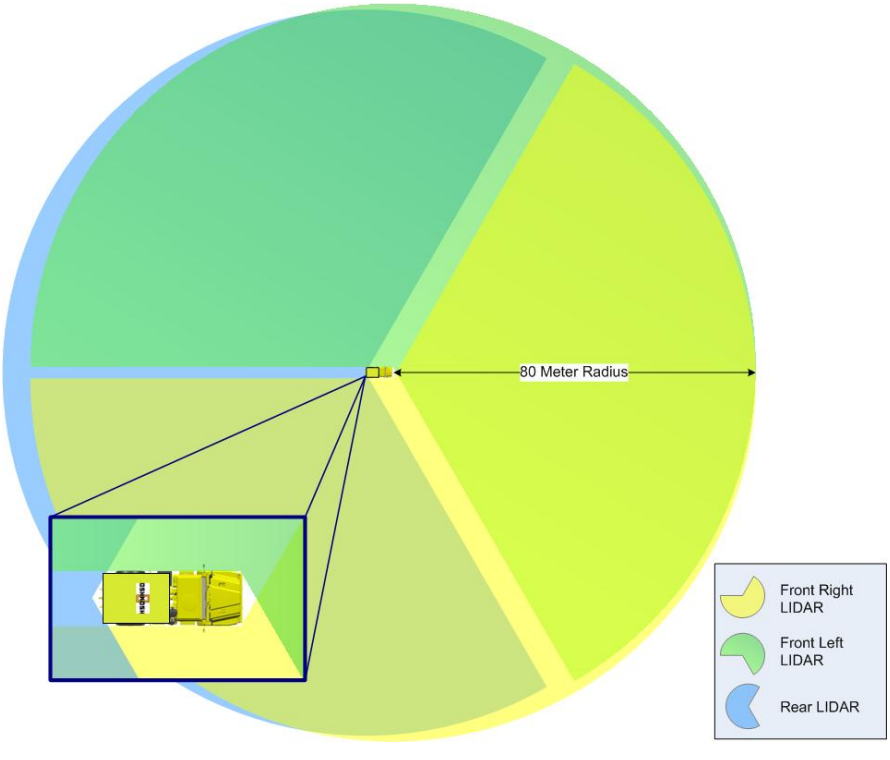


Fig. 2. LIDAR Coverage of TerraMax (truck facing right).

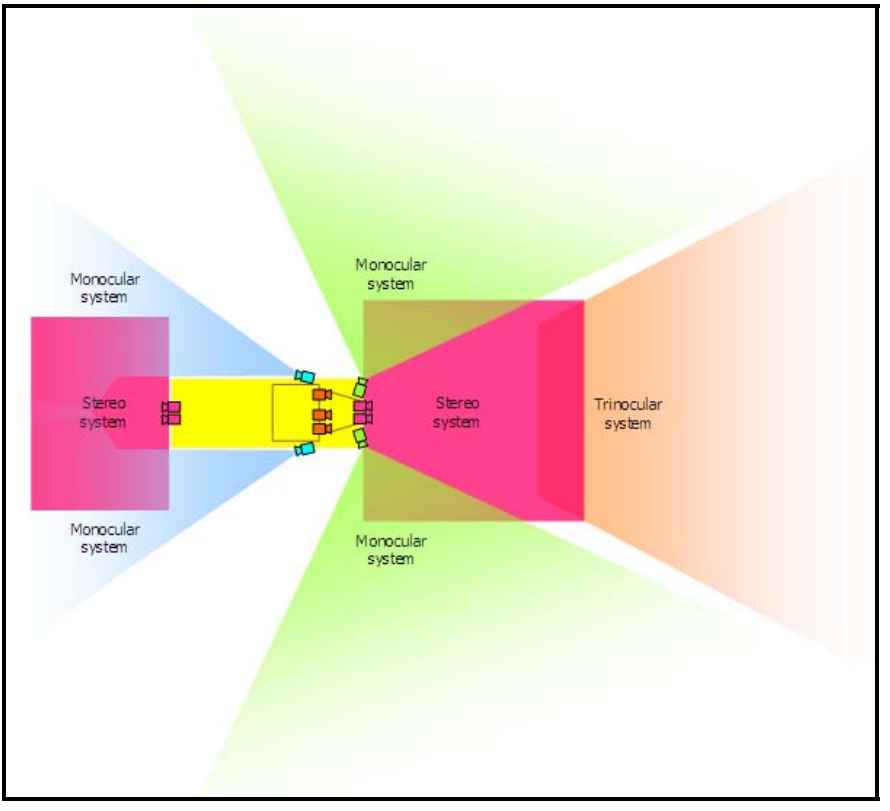


Fig. 3. Vision Coverage of TerraMax (truck facing right). Systems displayed: Trinocular (Orange) looking forward from 7 to 40m, Stereo Front and Stereo Back (Purple) monitoring a 10x10m area on the front of the truck and a 7x5m in the back, RearView (Blue) monitoring up to 50m behind the truck, and Lateral (Green) looking up to 130m.

2.3.2 Vision Hardware

There are four vision systems onboard: trinocular, stereo, rearview, and lateral. Figure 3 depicts the coverage of these vision systems. Table 1 summarizes the functions and components of these vision systems.

Each vision system is formed by a computer connected to a number of cameras and laserscanners, depending on the application. Each computer is connected through an 800Mbps, FireWire B link to a subset of the 11 cameras (9 PointGrey Flea 2, sensor: CCD, 1/3", Bayer pattern, 1024x768 (XGA) and 2 Allied Vision Technologies Pike 2. Sensor: 1", Bayer pattern, 1920x1080 pixels (HDTV)) mounted on the truck, depending on the system purpose.

2.3.3 GPS/INS

Using a Novatel GPS receiver with Omnistar HP corrections (which provides 10 cm accuracy in 95% of cases) as a truth measurement in extensive tests under

Table 1. Vision System Components.

Vision System	TRINO	STEREO	LATERAL	REARVIEW
Cameras	3x PtGrey Flea2 (XGA)	4x PtGrey Flea2 (XGA)	2x Allied Vision Technologies Pike 2 (HDTV)	2x PtGrey Flea2 (XGA)
Cameras Position	Upper part of the windshield, inside the cab	2 on the front camera-bar, two on the back of the truck, all looking downwards	On the sides of the front camera-bar	External, on top of the cab, looking backwards and downwards, rotated by 90°
Linked Laser scanner	Front	Front, back	Not used	Back
Algorithms	Lane detection, stereo obstacle detection	Lane detection, stop line detection, curb detection, short-range stereo obstacle detection	Monocular obstacle detection	Monocular obstacle detection
Range	7 to 40m	0 to 10m	10 to 130m	-4 to -50m
Notes	3 stereo systems with baselines: 1.156 m, 0.572 m, 1.728 m	2 stereo systems (front and rear)	Enabled when the truck stops at crossings	Overtaking vehicles detection

normal and GPS-denied conditions, we selected Smiths Aerospace Inertial Reference Unit (IRU) as our GPS/INS solution based on its more robust initialization performance and greater accuracy in GPS-denied conditions.

3 System Architecture

Based on a layered architecture design pattern (Team Oshkosh DARPA Urban Challenge Technical Report, 2007), we designed the software modules as services that provide specific functions to the overall system. These services interact with each other through a set of well-defined asynchronous messages. Figure 4 illustrates these software modules and the relations among them. As illustrated, there are two main types of services: Autonomous Services, whose modules provide functionalities for autonomous vehicle behaviors and System Services, whose modules support the reliable operations of the vehicle and the mission. We summarize the main functionalities of these software modules in the following description.

3.1 Autonomous Services

Autonomous Vehicle Manager

The Autonomous Vehicle Manager (AVM) manages the high level autonomous operation of the vehicle. It is primarily responsible for performing route planning, trajectory planning, and behavior management. The AVM receives perception updates from the World Perception Server (WPS) and uses this information to track the current vehicle state and determine the current behavior mode. The AVM continuously monitors perceived obstacle and lane boundary information and issues revised trajectory plans to the Autonomous Vehicle Driver (AVD) through Drive Commands.

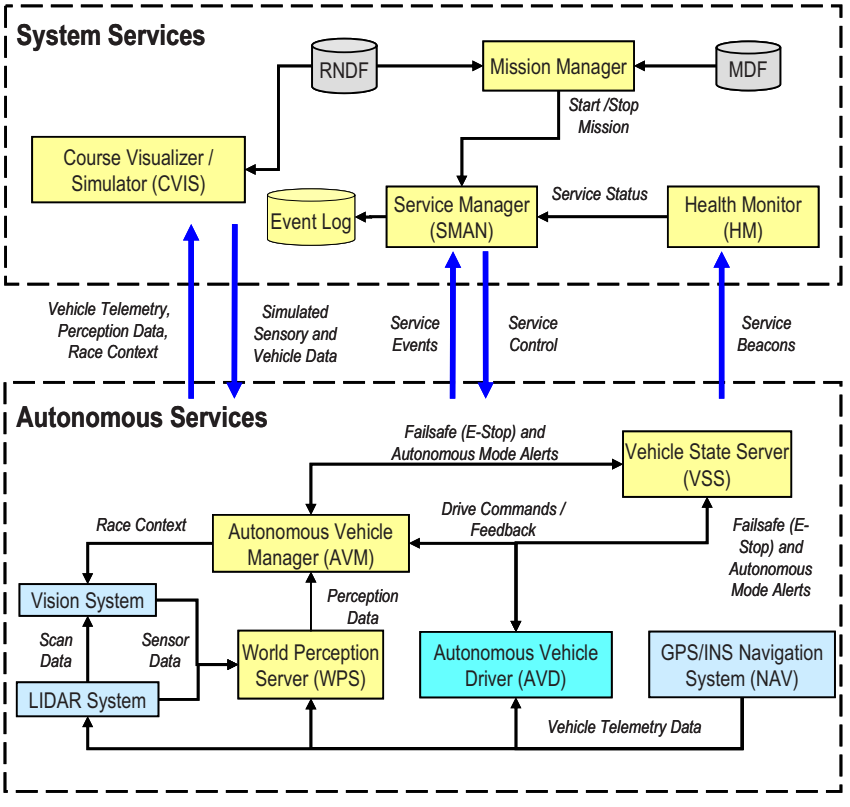


Fig. 4. Software deployment architecture.

Autonomous Vehicle Driver

The Autonomous Vehicle Driver (AVD) provides vehicle-level autonomy, such as waypoint following, lateral, longitudinal and stability control by accepting messages from the AVM and commanding the lower level control-by-wire actuations.

World Perception Server

The World Perception Server (WPS) publishes perception updates containing the most recently observed vehicle telemetry, obstacle, and lane/road boundary information. The WPS subscribes to sensory data from the LIDAR and VISION systems. The WPS combines the sensory data with the vehicle telemetry data received from the navigation service (NAV). Obstacles detected by the LIDAR and VISION systems are further fused in order to provide a more accurate depiction of the sensed surroundings. The AVM consumes the perception updates published by the WPS and uses this information to determine the next course of action for the vehicle.

Vision System
The Vision System (VISION) publishes processed sensory data and meta-data from different groups of cameras. The meta-data may contain information such as detected driving lane/path, lane boundary and curb marking, and/or obstacles. These sensory data and meta-data are sent to WPS for distribution.

Other autonomous services include: Vehicle State Server (VSS), which monitors and manages low level control for transitions from manual to autonomous operations, detects any low level faults, and attempts to recover the system into failsafe mode, if needed; LIDAR System (LIDAR), which fuses and publish obstacle information provided by the native obstacle detection and tracking functionalities from different laser scanners; and the NAV Service that manages communications to the GPS/INS.

3.2 System Services

Course Visualizer
The Course Visualizer is the prime interface to allow human operators/developers to observe the internal operations and status of the autonomous systems during a run. During an autonomous run, it provides real-time two-dimensional visualization of the course data (i.e., Road Network Definition File, RNDF), vehicle telemetry data, meta-data from sensors (e.g., lane updates, obstacles, etc.), and status/ results of autonomous behavior generation (e.g., internal logics of a particular autonomous mode, results of a particular behavior algorithm). It can also serve as the main playback platform to enable post-operation analysis of the data log. Incorporated with a simplistic vehicle model, it also serves as a rough simulation platform to allow early testing and verification for developing or adjusting behavioral algorithms.

Other system services include: Mission Manager, which provides the user interface for configuring autonomous services, loading mission files, and starting the autonomous services/ modes; Health Monitor, which monitors service beacons from other services and alert the Service Manager if an anomaly occurs; and Service Manager, which manages the initialization, startup, restart, and shutdown of all autonomous services.

4 Sensor Processing and Perception

In previous Grand Challenge efforts we used a trinocular vision system developed by VisLab at the University of Parma for both obstacle and path detection, coupled with laser scanning systems for obstacle detection. We used several SICK laser scanners and one IBEO laser scanner for obstacle detection. The Grand Challenge generally involved only static obstacles, so sensing capabilities focused on the front of the vehicle. Urban driving introduces a new dynamic—obstacles move (i.e. other moving vehicles) and the vehicle must respond to these moving obstacles, resulting in a much greater need for sensing behind and to the sides of the vehicle. The autonomous vehicle manager needs more information

about the obstacles, requiring their velocity as well as their location, and it also needs great precision in detecting stop lines and lane boundaries. To meet these goals, we enhanced capabilities in both laser scanning and vision.

IBEO provided three of their advanced ALASCA XT laser scanners and fusion algorithms for an integrated 360º view. The new TerraMax™ vehicle utilizes multiple vision systems, with perception capabilities in all the critical regions.

4.1 LIDAR

The IBEO laser scanners have two roles on TerraMax. First, they provide processed object data (Wender, Weiss, et. al., 2006, Kaempchen, Bühler, & Dietmayer, 2005) to the World Perception Server. Second, they provide scan-level data used by the vision system to improve its results. The onboard External Control Units (ECUs) fuse the data from the two front LIDARs2 acting as a single virtual sensor in the front.

4.2 Vision System

4.2.1 Software Approach

All the vision computers run the same software framework (Bertozzi, Bombini, et. al., 2008), and the various applications are implemented as separate plug-ins. This architecture allows hardware abstraction, while making a common processing library available to the applications, thus making algorithm development independent of the underlying system. Each vision system controls its cameras using a selective auto-exposure feature. Analysis of each image is focused on a specific region of interest.

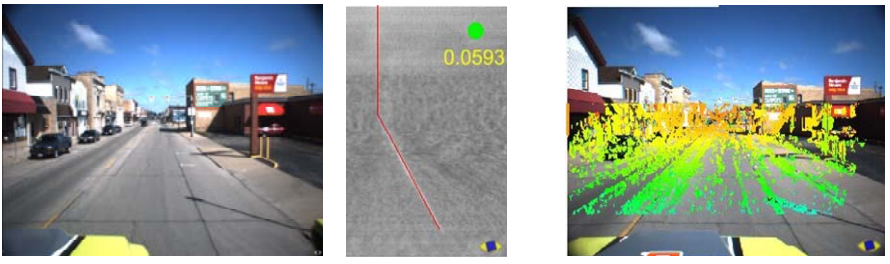
All the vision systems are fault tolerant with respect to one or more, temporary or permanent, sensor failure events. The software is able to cope with FireWire bus resets or laser scanner communication problems, and to reconfigure itself to manage the remaining sensors.

4.2.2 The Trinocular System

Driving in urban traffic requires detailed perception of the environment surrounding the vehicle: for this, we installed in the truck cabin a trinocular vision system capable of performing both obstacle and lane detection up to distances of 40 meters, derived from (Caraffi, Cattani, & Grisleri, 2007). The stereo approach has been chosen since it allows an accurate 3D reconstruction without requiring strong a-priori knowledge of the scene in front of the vehicle, but just correct calibration values, which are being estimated at run-time.

The three cameras form three possible baselines (the baseline is the distance between two stereo cameras), and the system automatically switches between them depending on the current vehicle speed; at lower speeds it is thus more convenient to use the shorter (and more accurate) baseline, while at higher speeds the large baseline permits the detection of obstacles when they are far from the vehicle.

[2] Ibeo laserscanner fusion system. http://www.ibeo-as.com/english/technology_d_fusion.asp.



(a) (b) (c)

Fig. 5. (a) A frame captured from the right camera; (b) Corresponding V-Disparity map, where the current pitch is shown in yellow text, and the detected ground slope is represented by the slanted part of the red line; and (c) Disparity map (green points are closer, orange ones are farther away).

Images are rectified, so that the corresponding epipolar lines become horizontal, thus correcting any hardware misalignment of the cameras and allowing for more precise measurements. The V-Disparity map (Labayrade, Aubert, & Tarel, 2002) is exploited to extract the ground slope and current vehicle pitch, in order to compensate for the oscillations that occur while driving (Figure 5(a), (b)).

The next step is to build a disparity map from the pair of stereo images: this operation is accomplished using a highly optimized incremental algorithm, which takes into account the previously computed ground slope in order to produce more accurate results and to reduce the processing time (Figure 5(c)).

The disparity map, along with the corresponding 3D world coordinates, is used to perform obstacle detection. After a multi-step filtering phase aimed at isolating the obstacles present in front of the truck, the remaining points are merged with the ones from the front LIDAR, and are initial values for a flood-fill expansion step, governed by each pixel disparity value, in order to extract the complete shape of each obstacle. This data fusion step ensures good performance in poorly textured areas, while ensuring robustness of the vision-based obstacle detection algorithm against LIDAR sensor failures. Previously identified obstacles are removed from the full-resolution image used by the lane detection algorithm, lowering the possibility of false positives, such as those introduced by poles or vehicle parts. Figure 6 shows a typical scene and the lanes detected by the algorithm.

4.2.3 Stereo System

Navigation in an urban environment requires precise maneuvers. The trinocular system described in the previous section can only be used for driving at medium to high speeds, since it covers the far range (7-40m). TerraMax includes two stereo systems (one in the front and one in the back, derived from (Broggi, Medici, & Porta, 2007)), which provide precise sensing at closer range. Using wide-angle (fisheye, about 160°) lenses, these sensors gather information over an extended area of about 10x10 meters; the stereo systems are designed to detect obstacles and lane markings with high confidence on the detection and position accuracy.

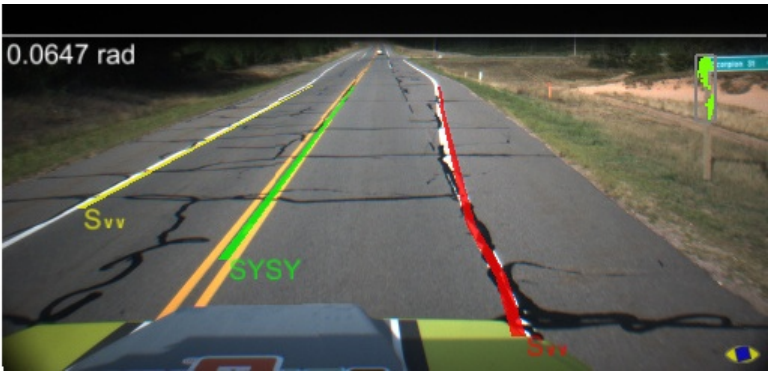


Fig. 6. Results of lane detection algorithm projected on the original image. From right to left, the red line represents a right boundary, green a left boundary, and yellow a far left boundary. In this image, the right line is detected although it is not a complete line.

Obstacle detection is performed in two steps: first the two images, acquired simultaneously, are preprocessed in order to remove the high lens distortion and perspective effect, a thresholded difference image is generated and labeled (Bertozzi, Broggi, & Medici, et. al., 2006), and then a polar histogram-based approach is used to isolate the labels corresponding to obstacles (Bertozzi & Broggi, 1998, Lee & Lee, 2004). Data from the LIDARs are clusterized so that laser reflections in a particular area can boost the score associated with the corresponding image regions, thus enhancing the detection of obstacles.

The chosen stereo approach avoids explicit computation of cameras intrinsic and extrinsic parameters, which would have been impractical, given the choice of using fisheye lenses to cover a wide area in front of the truck. The use of a lookup table (generated using a calibration tarp) to remap the distorted input images to a bird's-eye view of the scene thus results in improved performance and reduced calibration time.

Short-range line detection is performed using a single camera, to detect lane markings (even along a sharp curve), stop lines, and curbs. As the camera approaches the detected obstacles and lane markings, the position accuracy increases, yet the markings remain in the camera field of view due to the fisheye lens. A precise localization of lane markings enables the following: lane-keeping despite large width of the vehicle; stopping of the vehicle at close proximity to the stop line at intersections; accurate turning maneuvers at intersections; and precise planning of obstacle avoidance maneuvers. Figure 7 shows a frame with typical obstacle and lane detection.

4.2.4 Lateral System

We employ lateral perception to detect oncoming traffic at intersections. During a traffic merge maneuver, the vehicle is allowed to pull into traffic only when a gap of at least 10 seconds is available. For this, the vehicle needs to perceive the presence of oncoming traffic and estimate vehicle speeds at range. The intersecting

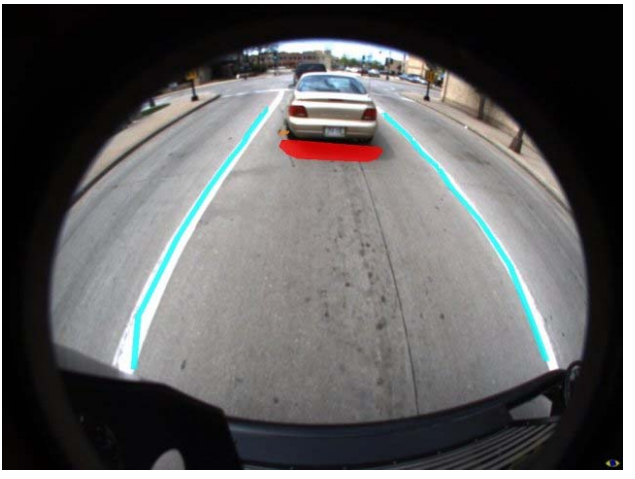


Fig. 7. A sample frame showing typical obstacle and lane detection.

Fig. 8. Lateral system algorithm results (detected vehicles are marked red).

road might be viewed at an angle other than 90°; therefore the lateral system must be designed to detect traffic coming from different angles. We installed two high resolution AVT Pike cameras (1920x1080 pixels) on TerraMax™ – one on each side – for lateral view, together with 8 mm Kowa lenses. With this configuration each camera can cover a 90° angle, and is able to see objects at high resolution up to distances over 130m.

The lateral camera image is processed using a robust, ad-hoc background subtraction based algorithm within a selected region of interest, with the system being triggered by the Autonomous Vehicle Manager when the vehicle stops at an

intersection, yielding to oncoming traffic. This approach allows us to handle the high-resolution imagery with a simple, computationally effective approach by leveraging the semantic context of vehicular motion.

4.2.5 Rearview System

When driving along a road, in both in urban and rural environments, lane changes and passing may occur. The Rearview System is aimed at detecting passing vehicles. This solution has proven to be very robust, while keeping processing requirements low; the onboard camera setup (with cameras placed on top of the cab, looking backwards) assures good visibility, since oncoming traffic is seen from a favorable viewing angle. The Rearview system processing is based on color clustering and optical flow. The first stage of processing performs data reduction in the image: a category is assigned to each pixel depending on its color, resulting in blobs that represent objects or portion of objects of uniform color. In the second (optic flow) stage, blobs of uniform color are analyzed and tracked, to estimate their shape and movement.

Obstacles found using optical flow are then compared with those detected by the LIDAR: since the latter has higher precision, the position of obstacles estimated by vision is refined using the LIDAR data, if available. The fusion algorithm thus performs only position refinement and does not create/delete obstacles, in order to isolate the detection performance of the vision system from that of the LIDAR.

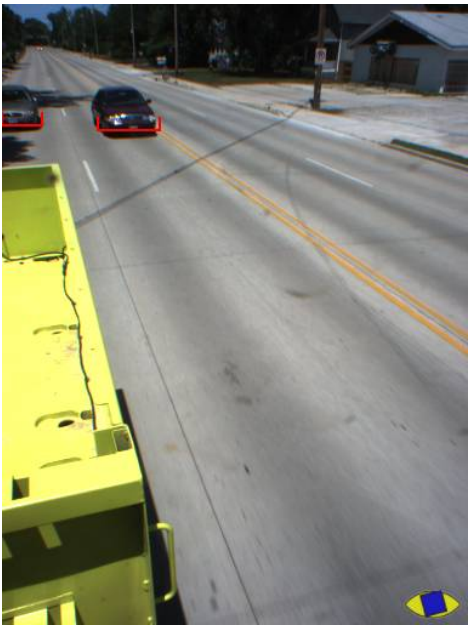


Fig. 9. Rear view system algorithm results (detected vehicles are marked in red).

4.3 Obstacle Fusion and Tracking

We adopted a high-level approach for fusing (both dynamic and static) obstacle information. The high-level fusion approach was favored for its modularity and rapid implementation. It was also well-suited for our geographically dispersed development team.

In this approach, obstacles are detected locally by the LIDAR and vision systems. Detected obstacles are expressed as objects which contain relevant information such as outline points, ID, velocity, height (vision only), and color (vision only). The obstacles from LIDAR and vision are fused in the WPS based on their overlap and proximity.

Similarly, we relied on the native functions of the LIDAR (Wender, Weiss, et. al., 2006) and vision systems for obstacle tracking (through object IDs generated by these systems). This low-level only tracking approach proved to be effective for most of the situations. However, it was inadequate in more complex situations where a vehicle is temporarily occluded by another (see discussions in Sections 6.2 and 7).

To predict the future position of a moving vehicle, the WPS applies a non-holonomic vehicle kinematic model (Pin & Vasseur, 1990) and the context of the vehicle. For example, if the vehicle is in a driving lane, the WPS assumes that it will stay in lane. If the vehicle is not in a lane, the WPS assumes it will maintain its current heading.

5 Planning and Vehicle Behaviors

In this section, we describe our approach for vehicle behavior generation and route/ trajectory planning.

5.1 Overview of Vehicle Behaviors

We adopted a goal-driven / intentional approach to mission planning and generation of vehicle behaviors. The main goal for the mission and behavior generation is to navigate sequentially through a set of checkpoints as prescribed in the DARPA supplied Mission Definition Files (MDF). Functionality related to autonomous vehicle behaviors is implemented in the Autonomous Vehicle Manager (AVM).

Main components in the AVM (as depicted in Figure 10) include: Mission / Behavior Supervisor (MBS), which manages the pursuit of mission goal (and sub-goals), selects and supervises the appropriate behavior mode for execution; Mission/ Route Planner, which generates (and re-generates as needed) high-level route plans based on the road segments and zones defined in the Road Network Definition File (RNDF); Behavior Modes & Logic, which contains a set of behavior modes, the transitional relationship among them, and the execution logic within each behavior mode; Event Generators, which monitor the vehicle and environment state estimation from the WPS and generate appropriate events for

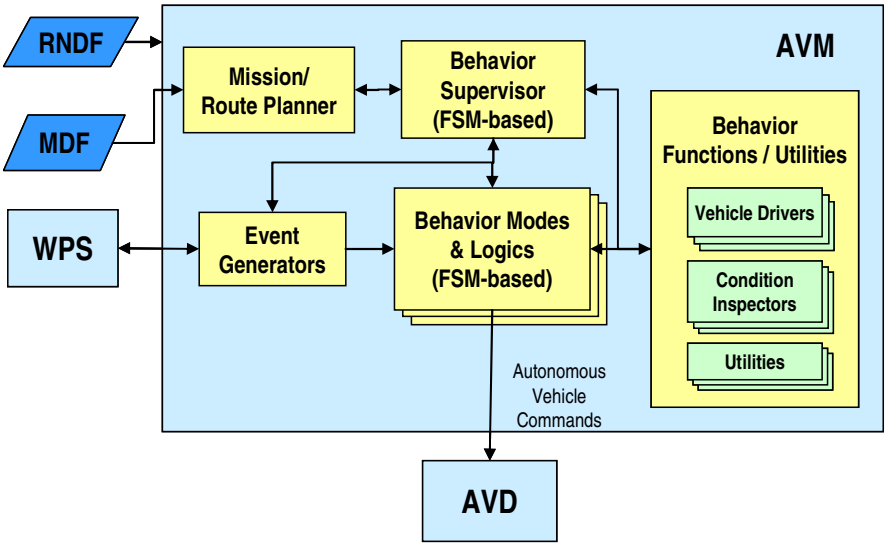


Fig. 10. Major function blocks in the Autonomous Vehicle Manager (AVM).

the behavioral modes when a prescribed circumstance arises (e.g. an obstacle in the driving lane ahead, etc.); and behavior functions/utilities, which provide common services (e.g. trajectory generation, etc.) for different behavior modes.

5.2 Behavioral Modes and Supervision

We adopted a Finite-State-Machine (FSM) based discrete-event supervisory control scheme as our primary approach to generate and execute autonomous vehicle behaviors. The FSM based scheme provides us with a simple, yet structured, approach to effectively model the race rules/constraints and behaviors/tactics, as opposed to the conventional rule-based approaches or behavior-based approaches (Arkin, 1998). This scheme allows us to leverage existing supervisory control theories and techniques (Ramadge & Wonham, 1987, Cassandras & Lafortune, 1999, Chung, Lafortune, & Lin, 1992, Chen & Lin, 2001/CDC) to generate safe and optimized behaviors for the vehicle.

We model autonomous behaviors as different behavior modes. These behavior modes categorize potential race situations and enable optimized logic and tactics to be developed for the situations. We implemented seventeen behavior modes, shown in Figure 11, to cover all the basic and advanced behaviors prescribed in the Urban Challenge. Examples of the behavior modes include: Lane Driving, where the vehicle follows a designated lane based on the sensed lane or road boundaries; and Inspecting Intersection, where the vehicle observes the intersection protocol and precedence rules in crossing intersections and merging with existing traffic.

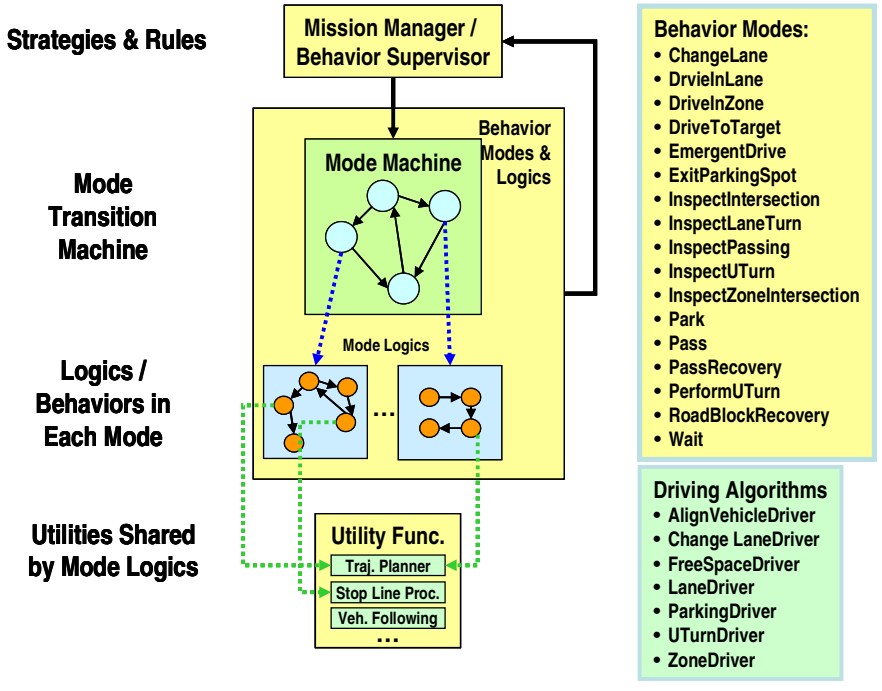


Fig. 11. Components for behavior generation and execution.

For each behavior mode, a set of customized logic is modeled as an extended finite state machine (e.g., a Finite State Machine with Parameters (FSMwP) (Chen & Lin, 2000)), which describes the potential behavior steps, conditions, and actions to be taken. The behavior logic may employ different utility functions (e.g., trajectory planners/ driving algorithms, stop-line procedure, etc.) during execution.

Transitions among behavior modes are modeled explicitly as an FSMwP (Chen & Lin, 2000), named Mode Transition Machine (MTM), where guard conditions and potential actions / consequences for the transitions are expressed. The Mode Transition Machine is used by the Behavior Supervisor in MBS in determining the appropriate behavior mode to transition to during execution.

The generation and control of the vehicle behavior may be formulated as a supervisory control problem. We adopted the concepts of safety control (Chen & Lin, 2001/ACC) and optimal effective control (Chen & Lin, 2001/CDC) for FSMwP where the traffic rules and protocols are formulated as safety constraints and current mission sub-goal (e.g., check point) as the effective measure to achieve. However, to improve the real time performance during execution, we manually implemented a simplified supervisor that does not require explicit expansion of supervisor states (Chung, et. al., 1992) by exploiting the structure of the MTM.

Our finite state machine-based behavior generation scheme is intuitive and efficient. However, it may suffer from several potential drawbacks. Among them are the reduced robustness in handling unexpected situations and the lack of "creative" solutions/behaviors. To mitigate the potential concern in handling unexpected situations, we included an unexpected behavior mode (RoadBlockRecovery mode) and instituted exception-handling logic to try to bring the vehicle to a known state (e.g., on a known road segment, or zone). Through our field-testing and participation at official events, we found that this unexpected behavior mode to be generally effective in ensuring the robust autonomous operation of the vehicle. A more robust unexpected behavior mode based on "non-scripted" techniques, such as behavior-based approaches (Arkin, 1998) may be introduced in the future to handle the unexpected situations. This hybrid approach would strike the balance between simplicity/consistency and flexibility/robustness of behaviors.

5.3 Route Planning

The objective of the route planning component is to generate an ordered list of road segments among the ones defined in RNDF which enables the vehicle to visit the given set of checkpoints, in sequence, at the least perceived cost of the current sensed environment. We implemented the route planner as a derivative of the well-known Dijkstra's Algorithm (Cormen, Leiserson, & Rivest, 1990) which is a greedy search approach to solve the single-source shortest path problem.

We used the estimated travel time as the base cost for each road segment, instead of the length of the road segment. This modification allowed us to effectively take into account the speed limit of the road segments (both MDF-specified and self-imposed due to the large vehicle's constraints) and the traffic conditions the vehicle may experience through the same road segment previously traveled (during the same mission run). Meanwhile, any perceived road information, such as road blockage, and (static) obstacles are also factored into the cost of the road.

5.4 Trajectory Planning

The trajectory planner generates a sequence of dense and drivable waypoints, with their corresponding target speeds, for the AVD to execute, given a pair of start/end waypoints and, possibly, a set of intermediate waypoints. Alternatively, the trajectory planner may also prescribe a series of driving commands (which include steering direction and travel distance). Instead of using a general purpose trajectory/motion planner for all behavior modes, we implemented the trajectory planning capabilities as a collection of trajectory planner utility functions that may be called upon by different behavior modes depending on the current vehicle and mission situation. Our approach exploited specific driving conditions in different behavior modes for efficient and consistent trajectory planning.

We implemented four different types of trajectory planners:

- **Lane-following trajectory planner** utilizes detected/estimated lane and road boundaries to generate waypoints that follow the progression of the lane/road. Targeted speed for each waypoint is determined by considering the (projected) gap between TerraMax and the vehicle in front, if any, dynamics of TerraMax (e.g., current speed, limits of acceleration/deceleration), and the speed limit of the road segment. Curvature among waypoints are further checked, adjusted, and smoothed using a spline algorithm (Schoenberg, 1969) to ensure the waypoints are drivable within TerraMax's dynamic constraints.
- **Template-based trajectory planners** are a set of trajectory planners that can quickly generate trajectory waypoints based on instantiation of templates (Horst & Barbera, 2006) with current vehicle and environment state estimates for common maneuvers such as lane changing, passing, swerving, and turning at intersections. A template-based trajectory planner determines first the targeted speed for (each segment of) the maneuver, using the method similar to that for the lane-following trajectory planner, and apply the targeted speed to the parameterized trajectory template in generating the waypoints.
- **Rule-based trajectory planners** utilize a set of simple driving and steering heuristic rules (Hwang, Meirans, & Drotning, 1993) that mimic the decision process of human drivers in guiding the vehicle into a certain prescribed position and/or orientation, such as U-turns or parking. Since the rule-based trajectory planners are usually invoked for precision maneuvers, we configured the targeted speed to a low value (e.g. 3 mph) for maximum maneuverability.
- **Open-space trajectory planner** provides general trajectory generation and obstacle avoidance in a prescribed open-space where obstacles may be present. We adopted a two-level hierarchical trajectory planning approach where a lattice/A* based high-level planner (Cormen, et. al., 1990) provides a coarse set of guiding waypoints to guide the trajectory generation of the low-level Real Time Path Planner (RTPP), which is based on a greedy, breadth-first search algorithm augmented with a set of heuristic rules. Similar to that for the rule-based trajectory planner, we configure the target speed of the open-space trajectory planner to a low value (e.g. 5 mph) for maximum maneuverability.

6 Field Performance and Analysis

In this section, we discuss TerraMax's performance during testing and participation in the Urban Challenge and related events.

6.1 Basic and Advanced Behaviors

TerraMax successfully demonstrated all the basic and advanced behavior requirements set forth by DARPA. In the following, we comment on our experience in implementing some key autonomous behaviors.

Passing: Initially, the trajectory generation of the passing behavior was handled by a template-based passing trajectory planner, which guided the vehicle to an adjacent lane for passing the detected obstacle in its current lane and returned the vehicle back to its original lane after passing. We added a *swerve* planner to negotiate small obstacles (instead of *passing* them). This modification resulted in smooth and robust passing performance.

U-Turn and Parking: Through testing, we found that a rule-based approach outperformed a template-based approach for U-turns and parking. Therefore we employed a rule-based parking maneuver, which performed flawlessly in the NQE and UCFE.

Merge and Left Turn: We adopted a simple approach to inspect traffic in the lanes of interest and determine if there is a safe gap for executing Merge or Left Turn behaviors. We employ a simple vehicle kinematic model (Pin & Vasseur, 1990) to predict the possible spatial extent that a moving vehicle in the lane may occupy in the near future (e.g. in the next 10 seconds) and apply an efficient geometry-based algorithm to check if the spatial extent intersects with TerraMax's intended path. To reduce false (both positive and negative) detections of traffic in a lane, we further fine-tuned LIDAR sensitivity, employed a multi-sample voting scheme to determine whether a vehicle is present based on multiple updates of the obstacles reported by the LIDAR, and verified our modifications through an extended series of controlled live-traffic tests. The enhancements resulted in near perfect merging and left-turn behaviors.

6.2 Performance at the National Qualification Events

TerraMax participated in multiple runs in Test Areas A, B, and C during the National Qualification Events. During these runs, TerraMax successfully completed all the key autonomous maneuvers as prescribed in each Test Area. Specifically, in Test Area A, TerraMax demonstrated merging into live traffic and left turn maneuvers; in Test Area B, TerraMax completed leaving the start chute, traffic circle, zone navigation and driving, parking, passing, and congested road segment maneuvers; in Test Area C, TerraMax demonstrated intersection precedence, queuing, roadblock detection, U-turn, and re-planning behaviors.

In Table 2, we summarize the performance issues we experienced during the early runs in the NQE and our actions in resolving/ mitigating these issues for the subsequent runs in the NQE and the UCFE. Following the table, we discuss each of the performance items in detail.

Obstacles protruding from the edge of a narrow road could interfere with safety spacing constraints and cause path deviation: During our first run of Test Area A, TerraMax did not stay in the travel lane but drove on the centerline of the road segment. The K-rails on the road closely hugged the lane boundary, appearing to be inside the lane. This prompted obstacle avoidance, and due to our earlier safety-driven decision to never approach an obstacle closer than 0.5 meters, the AVM decided to cross the centerline to pass.

Table 2. Performance issues and resolutions in NQE.

Performance issue	Cause	Impact on performance	Mitigating action(s)	Lessons learned
Minor obstacles (k-rails) causing planned path deviation	Safety parameter setting	Vehicle rode centerline	Reduce clearance to obstacle Widen path by moving k-rails	Narrow roads are harder to navigate for a large truck
Path updates incorrectly processed	False positives for road edges	Vehicle got stuck in driveway	Corrected path updates and readjusted path determination method	Road edge detection and processing can be complex and unreliable
Parking twice in the same spot	Dust clouds behind vehicle perceived as obstacles	Total time taken increased	None required	Temporary dust clouds need to be treated differently than other obstacles; high level obstacle fusion desirable
Traveling too close to parked cars	Real-time path planner not activated in time	Brushing a parked car	Modified trajectory planning transition logic	Navigating congested narrow road segments needs sophisticated supervisory logic
Entering intersection prematurely	Timer bug	Incorrect precedence at intersection	Fixed timer bug	Simple bugs could cause large-scale performance impact
Incorrect tracking at intersections	Vehicle hidden by another	Incorrect precedence at intersection	Implemented obstacle caching; better sensor fusion and tracking	Simple sensor processing is inadequate in complex settings
Queuing too close / erratic behaviors	LIDAR malfunction	Nearly ran into the vehicle in front	Fixed bug in sensor health monitoring	Critical system services have to be constantly monitored

Unreliable path/lane updates could cause incorrect travel lane determination: During the first run of Test Area B, TerraMax pulled into a driveway of a house and, after a series of maneuvers, successfully pulled out of that driveway but led itself right into the driveway next-door where it stalled and had to be manually repositioned. This time-consuming excursion was primarily triggered by an incorrect path update and subsequent incorrect travel lane selection. We resolved the faulty path update and re-adjusted the relative importance of the different information sources used to determine the travel lane.

Dust clouds and other false positive "transient" obstacles could result in unexpected behaviors: Unexpected behaviors observed in dirt lots of Test Area B can be attributed to dust clouds having been sensed as obstacles. These behaviors included parking-twice in both runs, a momentary stop, and "wandering around" in the dirt lot. On all of these occasions, the system recovered well and did not result in failures other than the superfluous excursions.

More sophisticated free-space trajectory planners (other than simple template-based ones) are needed to negotiate highly congested areas: While navigating a segment where many obstacles were set up on a curvy road during the first run of Test Area B, TerraMax slowly knocked down several traffic cones and brushed a parked vehicle with the left corner of its front bumper. Our post-run study revealed that the RTPP was triggered much less often than desired. To prevent TerraMax from hitting obstacles, especially in the congested road segments as we experienced, we revised AVM's transition logic and processes between normal driving modes and RTPP. With the revisions, the Supervisor invoked RTPP in more situations.

Persistent vehicle tracking needed for complex situations at intersections: During the second run of Test Area C, TerraMax entered the intersection prematurely on one occasion. This was due to the lack of proper tracking of the obstacles/vehicles in our overall system. When the first vehicle entered the intersection, it momentarily occluded the second vehicle. The AVM could not recognize that the newly observed obstacle was in fact the vehicle that was there before. We mitigated this problem by refining the obstacle caching and comparison mechanism.

Proper response should be designed to prepare for subsystem malfunction: In the second run of Test Area C, TerraMax started to behave erratically after one-third of the mission was completed. It drove very close to the obstacles (and K-rails) at the right side of the lane and it almost hit a vehicle that queued in front of it at the intersection. In analysis, we discovered that the front right LIDAR had malfunctioned and did not provide any obstacle information. This, combined with the fact that the Stereo obstacle detection had not been enabled meant that TerraMax could not detect obstacles in the front right at close range. Fortunately, there was some overlap coverage from the front left LIDAR, which was functioning correctly. This overlap coverage from the front left LIDAR picked up the vehicle queued in front of TerraMax at the intersection so that the Supervisor stopped TerraMax just in time to avoid hitting this vehicle. Stereo obstacle detection was not turned on since the team did not have time to tune the thresholds before the start of this run.

6.3 Performance at the Urban Challenge Final Event

TerraMax completed the first four sub-missions in Mission 1 of the UCFE with impressive performance. However, TerraMax was stopped after a failure in the parking lot (Zone 61). Table 3 summarizes TerraMax's performance statistics prior to the parking lot event.

Table 3. Performance statistics for UCFE prior to parking lot

	Mission 1 (first 4 sub-missions)	
Total Time	0:40	
Top Speed	21 mph	
Oncoming vehicles encountered	47	
Oncoming autonomous vehicles encountered	7	
	Pass	Fail
Int. precedence	9	0
Vehicle following	0	0
Stop queue	1	0
Passes	1	0
Road blocks/Re-plan	0	0
Zone	1	0
Park	1	0
Merge[3]	1	1

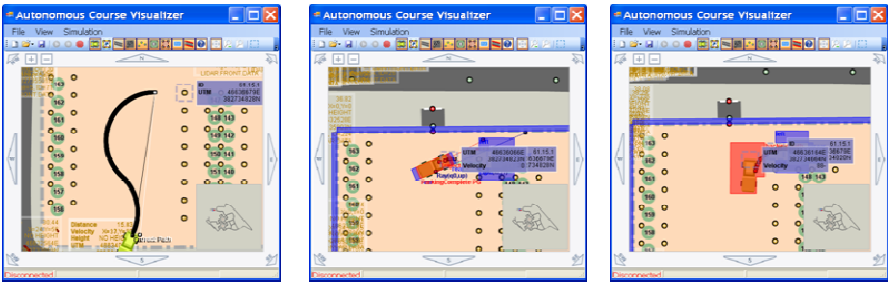


Fig. 12. UCFE Parking Lot: Successful parking maneuvers

6.3.1 Analysis of the Failure

The arrival into the parking lot was normal. There were no detected obstacles in the path of TerraMax and therefore an s-shaped (farmer) turn was issued that brought TerraMax into alignment with the target parking space (Figure 12(a)). TerraMax continued the parking maneuver successfully and pulled out of the parking space without error (Figure 12(c)). TerraMax backed out of the parking space to the right so that it would face the exit of the zone when finished exiting the parking spot.

[3] Failed cases indicate that traffic flow was impeded by TerraMax during merge.

There were two separate problems in the parking lot. The first problem was that TerraMax stalled in the parking lot for a very long time (approx. 17 minutes). The second problem was that TerraMax eventually continued, but no longer responded to commands from the AVM and eventually had to be stopped.

Stall Condition
The Real-Time Path Planner (RTPP) produced paths that contained duplicate waypoints while driving in a zone, resulting in a stall. This was a bug was introduced in the RTPP during pre-race modifications. This bug was quickly corrected after the UCFE and tested when we re-tested at Victorville in December 2007.

Unresponsive Vehicle
TerraMax recovered after stalling for more than 17 minutes. GPS drift "moved" the position of TerraMax to where the RTPP returned 4 points (2 pairs of duplicate waypoints). The Open-space trajectory planner then commanded TerraMax to drive at 5 mph in a straight path toward the parking lot boundary. However, the order of the commanded duplicate waypoints in the Drive command caused the AVD service to fault, at the point where the vehicle had already accelerated to approximately 1 mph. At this point, TerraMax became unresponsive to subsequent commands, and continued to drive in a straight line towards a building until an E-stop PAUSE was issued by DARPA officials.

Table 4. Performance statistics for Victorville test missions.

	Mission 1		**Mission 2**		**Mission 3**	
Total Distance	24.9 miles		19.5 miles		33.8 miles	
Total Time	2:28		1:55		3:18	
Average Speed	10.09 mph		10.17 mph		10.24 mph	
Oncoming vehicles encountered in opposite lane	42		92		142	
Intersections	84		108		193	
	Pass	Fail	Pass	Fail	Pass	Fail
Int. precedence[4]	17	3	8	0	20	2
Vehicle following	3	0	1	0	2	0
Stop queue	2	0	1	0	1	0
Passes	5	0	3	0	1	0
Road blocks/Re-plan	0	0	1	0	2	0
Zone	5	0	4	0	7	0
Park	3	0	2	0	2	0
Merge[5]	7	1	5	0	10	1

[4] In all cases, failed intersection precedence was due to the test vehicle(s) being more than 0.5m behind the stop line.

[5] Merge results include left turns across opposing lane, and left and right turns into traffic from a stop with other vehicles present. Failed cases indicate that traffic flow was impeded by TerraMax during merge.

6.4 Return to Victorville

We were unable to acquire full performance metrics during the UCFE due to the premature finish. Therefore we brought TerraMax back to Victorville on December 13, 2007 for a full test. Although we were not able to use the entire UCFE course, we used the UCFE RNDF and limited the missions to the housing area and added a parking spot in Zone 65. The MDF files created for these missions used the speed limits from the MDF for the first mission at the UCFE. TerraMax ran with the software version used in the UCFE. No revisions were made. The team ran three missions totaling over 78 miles in 7 hours and 41 minutes for an average speed of 10.17 mph. Six test vehicles driven by team members acted as other traffic. One safety vehicle followed TerraMax with a remote e-stop transmitter at all times during autonomous operation. We list performance statistics for the three missions in Table 4.

7 Concluding Remarks

Team Oshkosh entered the DARPA Urban Challenge with the intention of finishing the event as a top contender. Despite not completing the final event, the team believes that TerraMax had performed well and safely up to the point the vehicle was stopped. TerraMax proved to be a very capable contender and is arguably the only vehicle platform in the competition that is relevant for military logistics missions.

Through the development, testing, and official events, we experimented and demonstrated autonomous truck operations in (controlled) urban streets of California, Wisconsin, and Michigan under various climate conditions. In these experiments, TerraMax exhibited all the autonomous behaviors prescribed by DARPA Urban Challenge rules, including negotiating urban roads, intersections, and parking lots, interacting with manned and unmanned traffic while observing traffic rules, with impressive performance. Throughout this endeavor, we learned valuable experience and lessons, which we summarize in the following.

Course Visualizer / Simulator Efforts Truly Paid-off
Learning from our experience in DARPA Grand Challenge 2005, we invested time and effort upfront to develop a graphic tool with a 2-D capability for visualizing the RNDF and MDF. We later expanded the functionality of the tool to include mission data log playback, sensor information display, built-in debugging capability that displays results of various autonomous mode status, logic and calculations in the AVM, and simple simulation of TerraMax operations.

This tool served as a true “force-multiplier” by allowing team members in widely dispersed geographic locations to verify different functionality and designs, experiment with different ideas and behavioral modes, pre-test the software implementation prior to testing onboard TerraMax, and perform post-run analysis to resolve issues. The tool not only sped up our development and testing efforts, but also enabled us to quickly identify the causes for issues encountered during NQE runs and to promptly develop solutions to address them.

Simplicity Worked Well in U-Turn and Parking
Instead of using a full-fledged trajectory planner/generator for maneuvers such as U-Turn and parking, we opted to search for simple solutions for such situations. Our experiments with U-Turn prior to the Site Visit clearly indicated the performance, simplicity, elegance, and agility superiority of a rule-based approach over a template-based one. The rule-based approach, which mimics a human driver's actions and decision process in performing those maneuvers, was our main approach for both U-Turn and parking maneuvers and performed flawlessly in all our Site Visit, NQE and UCFE runs.

Better Persistent Object Tracking Capability Is Needed
In the original design object tracking was to be performed at a high level. However, due to time constraints, the object tracking responsibility was delegated to the processing modules of the individual sensor elements. As demonstrated in our first two runs of Test Area C, a persistent object tracking capability is required to handle situations where objects may be temporarily obstructed from observations. However, this persistent object tracking mechanism should only focus on the objects of both relevance and importance for efficiency and practicality. Though we implemented a less-capable alternative to maintain persistent tracking of vehicles at the intersection that yielded satisfactory results, a systematic approach to address this shortfall is needed.

Our sensor technology proved capable, but additional work is required to meet all the challenges
The use of passive sensors is one of the primary goals of our vehicle design. LIDAR sensors produce highly accurate range measurements, however vision allows cost effective sensing of the environment without the use of active signals (Bertozzi, Broggi, & Fascioli, 2006, Bertozzi, Broggi, et. al., 2002, Broggi, Bertozzi, et. al., 1999) and contains no moving parts which are less desirable in operational environments. The calibration of vision systems needed special care since many of them were based on stereo technology whose large baseline precluded attachment of the two cameras to the same rig. Nevertheless, the calibration of many of them turned out to be absolutely robust and there was no need to repeat the calibration procedure in Victorville after it was performed a few months before. The calibration of the trinocular system, which is installed into the cabin, survived a few months of test and many miles of autonomous driving. The front stereo system was uninstalled for maintenance purposes a few times and a recalibration was necessary, including in Victorville. The back stereo system did not need any recalibration, while the lateral and the rear view system, being monocular, relied only on a weak calibration which was just checked before the race. This is a clear step forward in usability and durability of vision systems for real-world unmanned systems applications.

Performing low-level fusion between vision and LIDAR data has brought considerable improvements in distance measurement for the vision systems especially at further distances. Robustness and persistence of results are additional improvements realized by means of this technique. Despite these improvements, LIDAR was used as the primary sensor for obstacle detection and vision the primary sensor for road boundary detection during the Final Event.

Lane detection provided consistent data and was able to localize most of the lane markings. Some problems were encountered when lane markings were too worn out and in situations in which the red curb was misinterpreted as a yellow line. The specific algorithm used to identify yellow markings was not tested in correspondence to red curbs, which showed the same invariant features that were selected for yellow lines.

Improved Road Boundary Interpretation Is Needed
Although the sensor system detected lanes and curbs in most situations, problems were encountered in situations where sensed data differed significantly from the expected road model obtained by interpreting the RNDF data. As a result TerraMax would cross the centerline, cut corners or drive off the road in order to reach the next RNDF waypoint.

Test for Perfection
The team had tested TerraMax extensively in a variety of environments and scenarios; the NQE and UCFE differed from our testing situations sufficiently however, that we were required to make last minute revisions to the system that were not extensively tested. Unfortunately, these last minute revisions for the NQE adversely impacted performance in the UCFE.

The DARPA Urban Challenge, as all DARPA Grand Challenges, has proven to be an excellent framework for the development of unmanned vehicles for Team Oshkosh. We have experimented and developed many elegant solutions for practical military large-footprint autonomous ground vehicle operations in urban environments. The system developed by Team Oshkosh for the Urban Challenge has been shown to be robust and extensible, an excellent base to which additional capabilities can be added due to the modular architecture.

References

Arkin, R.C.: Behavior-based robotics. MIT Press, Cambridge (1998)

Bertozzi, M., Bombini, L., Broggi, A., Cerri, P., Grisleri, P., Zani, P.: GOLD: a complete framework for developing artificial vision applications for intelligent vehicles. IEEE Intelligent Systems 23(1), 69–71 (2008)

Bertozzi, M., Broggi, A.: GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection. IEEE Transactions on Image Processing 1(7), 62–81 (1998)

Bertozzi, M., Broggi, A., Cellario, M., Fascioli, A., Lombardi, P., Porta, M.: Artificial vision in road vehicles. Proc. of the IEEE - Special issue on Technology and Tools for Visual Perception 90(7), 1258–1271 (2002)

Bertozzi, M., Broggi, A., Fascioli, A.: VisLab and the evolution of vision-based UGVs. IEEE Computer 39(12), 31–38 (2006)

Bertozzi, M., Broggi, A., Medici, P., Porta, P.P., Sjögren, A.: Stereo vision-based start-inhibit for heavy goods vehicles. In: Proc. IVS 2006, pp. 350–355 (2006)

Braid, D., Broggi, A., Schmiedel, G.: The TerraMax autonomous vehicle. J. of Field Robotics 23(9), 655–835 (2006)

Broggi, A., Bertozzi, B., Fascioli, A., Conte, G.: Automatic vehicle guidance: the experience of the ARGO vehicle. World Scientific, Singapore (1999)

Broggi, A., Medici, P., Porta, P.P.: StereoBox: a robust and efficient solution for automotive short range obstacle detection. EURASIP Journal on Embedded Systems - Special Issue on Embedded Systems for Intelligent Vehicles (June 2007) ISSN 1687-3955

Caraffi, C., Cattani, S., Grisleri, P.: Off-road path and obstacle detection using decision networks and stereo. IEEE Trans. on Intelligent Transportation Systems 8(4), 607–618 (2007)

Cassandras, C., Lafortune, S.: Introduction to Discrete Event Systems, 2nd edn. Kluwer, Dordrecht (1999)

Chen, Y.-L., Lin, F.: Modeling of discrete event systems using finite state machines with parameters. In: Proc. 9th IEEE Int. Conf. on Control Applications, September 2000, pp. 941–946 (2000)

Chen, Y.-L., Lin, F.: Safety control of discrete event systems using finite state machines with parameters. In: Proc. 2001 American Control Conf. (ACC), June 2001, pp. 975–980 (2001)

Chen, Y.-L., Lin, F.: An optimal effective controller for discrete event systems. In: Proc. 40th IEEE Conf. on Decision and Control (CDC), December 2001, pp. 4092–4097 (2001)

Chung, S.-L., Lafortune, S., Lin, F.: Limited lookahead policies in supervisory control of discrete event systems. IEEE Trans. on Automatic Control 37(12), 1921–1935 (1992)

Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT Press, Cambridge (1990)

Horst, J., Barbera, A.: Trajectory generation for an on-road autonomous vehicle. In: Proc. SPIE: Unmanned Systems Technology VIII. vol. 6230 (2006)

Hwang, Y.K., Meirans, L., Drotning, W.D.: Motion planning for robotic spray cleaning with environmentally safe solvents. In: Proc. IEEE Intl. Workshop on Advanced Robotics, Tsukuba, Japan (November 1993)

Kaempchen, N., Bühler, M., Dietmayer, K.: Feature-level fusion for free-form object tracking using laserscanner and video. In: Proc. 2005 IEEE Intelligent Vehicles Symposium, Las Vegas (2005)

Labayrade, R., Aubert, D., Tarel, J.P.: Real time obstacle detection in stereo vision on non flat road geometry through V-disparity representation. In: Proc. IEEE Intell. Veh. Symp., vol. II, pp. 646–651 (2002)

Lee, K., Lee, J.: Generic obstacle detection on roads by dynamic programming for remapped stereo images to an overhead view. In: Proc. ICNSC 2004, vol. 2, pp. 897–902 (2004)

Pin, F.G., Vasseur, H.A.: Autonomous trajectory generation for mobile robots with non-holonomic and steering angle constraints. In: Proc. IEEE Intl. Workshop on Intelligent Motion Control, August 1990, pp. 295–299 (1990)

Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control and Optimization 25(1), 206–230 (1987)

Schoenberg, I.J.: Cardinal interpolation and spline functions. Journal of Approximation theory 2, 167–206 (1969)

Sundareswaran, V., Johnson, C., Braid, D.: Implications of lessons learned from experience with large truck autonomous ground vehicles. In: Proc. AUVSI 2006 (2006)

Team Oshkosh DARPA Urban Challenge Technical Report. Oshkosh Corp. (April 2007), http://www.darpa.mil/grandchallenge/TechPapers/Team_Oshkosh.pdf

Wender, S., Weiss, T., Dietmayer, K., Fuerstenberg, K.: Object classification exploiting high level maps of intersections. In: Proc. 10th Intl. Conf. on Advanced Microsystems for Automotive Applications (AAMA 2006), Berlin, Germany (April 2006)

Author Index

Springer Tracts in Advanced Robotics

Edited by B. Siciliano, O. Khatib and F. Groen

Further volumes of this series can be found on our homepage: springer.com

Vol. 55: Stachniss, C.
Robotic Mapping and Exploration
196 p. 2009 [978-3-642-01096-5]

Vol. 54: Khatib, O.; Kumar, V.;
Pappas, G.J. (Eds.)
Experimental Robotics:
The Eleventh International Symposium
579 p. 2009 [978-3-642-00195-6]

Vol. 53: Duindam, V.; Stramigioli, S.
Modeling and Control for Efficient Bipedal
Walking Robots
211 p. 2009 [978-3-540-89917-4]

Vol. 52: Nüchter, A.
3D Robotic Mapping
201 p. 2009 [978-3-540-89883-2]

Vol. 51: Song, D.
Sharing a Vision
186 p. 2009 [978-3-540-88064-6]

Vol. 50: Alterovitz, R.; Goldberg, K.
Motion Planning in Medicine: Optimization
and Simulation Algorithms for
Image-Guided Procedures
153 p. 2008 [978-3-540-69257-7]

Vol. 49: Ott, C.
Cartesian Impedance Control of Redundant
and Flexible-Joint Robots
190 p. 2008 [978-3-540-69253-9]

Vol. 48: Wolter, D.
Spatial Representation and
Reasoning for Robot
Mapping
185 p. 2008 [978-3-540-69011-5]

Vol. 47: Akella, S.; Amato, N.;
Huang, W.; Mishra, B.; (Eds.)
Algorithmic Foundation of Robotics VII
524 p. 2008 [978-3-540-68404-6]

Vol. 46: Bessière, P.; Laugier, C.;
Siegwart R. (Eds.)
Probabilistic Reasoning and Decision
Making in Sensory-Motor Systems
375 p. 2008 [978-3-540-79006-8]

Vol. 45: Bicchi, A.; Buss, M.;
Ernst, M.O.; Peer A. (Eds.)
The Sense of Touch and Its Rendering
281 p. 2008 [978-3-540-79034-1]

Vol. 44: Bruyninckx, H.; Přeučil, L.;
Kulich, M. (Eds.)
European Robotics Symposium 2008
356 p. 2008 [978-3-540-78315-2]

Vol. 43: Lamon, P.
3D-Position Tracking and Control
for All-Terrain Robots
105 p. 2008 [978-3-540-78286-5]

Vol. 42: Laugier, C.; Siegwart, R. (Eds.)
Field and Service Robotics
597 p. 2008 [978-3-540-75403-9]

Vol. 41: Milford, M.J.
Robot Navigation from Nature
194 p. 2008 [978-3-540-77519-5]

Vol. 40: Birglen, L.; Laliberté, T.; Gosselin, C.
Underactuated Robotic Hands
241 p. 2008 [978-3-540-77458-7]

Vol. 39: Khatib, O.; Kumar, V.; Rus, D. (Eds.)
Experimental Robotics
563 p. 2008 [978-3-540-77456-3]

Vol. 38: Jefferies, M.E.; Yeap, W.-K. (Eds.)
Robotics and Cognitive Approaches to
Spatial Mapping
328 p. 2008 [978-3-540-75386-5]

Vol. 37: Ollero, A.; Maza, I. (Eds.)
Multiple Heterogeneous Unmanned Aerial
Vehicles
233 p. 2007 [978-3-540-73957-9]

Vol. 36: Buehler, M.; Iagnemma, K.;
Singh, S. (Eds.)
The 2005 DARPA Grand Challenge – The Great
Robot Race
520 p. 2007 [978-3-540-73428-4]

Vol. 35: Laugier, C.; Chatila, R. (Eds.)
Autonomous Navigation in Dynamic Environments
169 p. 2007 [978-3-540-73421-5]

Vol. 34: Wisse, M.; van der Linde, R.Q.
Delft Pneumatic Bipeds
136 p. 2007 [978-3-540-72807-8]

Vol. 33: Kong, X.; Gosselin, C.
Type Synthesis of Parallel Mechanisms
272 p. 2007 [978-3-540-71989-2]

Vol. 30: Brugali, D. (Ed.)
Software Engineering for Experimental Robotics
490 p. 2007 [978-3-540-68949-2]

Vol. 29: Secchi, C.; Stramigioli, S.; Fantuzzi, C.
Control of Interactive Robotic Interfaces – A Port-Hamiltonian Approach
225 p. 2007 [978-3-540-49712-7]

Vol. 28: Thrun, S.; Brooks, R.; Durrant-Whyte, H. (Eds.)
Robotics Research – Results of the 12th International Symposium ISRR
602 p. 2007 [978-3-540-48110-2]

Vol. 27: Montemerlo, M.; Thrun, S.
FastSLAM – A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics
120 p. 2007 [978-3-540-46399-3]

Vol. 26: Taylor, G.; Kleeman, L.
Visual Perception and Robotic Manipulation – 3D Object Recognition, Tracking and Hand-Eye Coordination
218 p. 2007 [978-3-540-33454-5]

Vol. 25: Corke, P.; Sukkarieh, S. (Eds.)
Field and Service Robotics – Results of the 5th International Conference
580 p. 2006 [978-3-540-33452-1]

Vol. 24: Yuta, S.; Asama, H.; Thrun, S.; Prassler, E.; Tsubouchi, T. (Eds.)
Field and Service Robotics – Recent Advances in Research and Applications
550 p. 2006 [978-3-540-32801-8]

Vol. 23: Andrade-Cetto, J,; Sanfeliu, A.
Environment Learning for Indoor Mobile Robots – A Stochastic State Estimation Approach to Simultaneous Localization and Map Building
130 p. 2006 [978-3-540-32795-0]

Vol. 22: Christensen, H.I. (Ed.)
European Robotics Symposium 2006
209 p. 2006 [978-3-540-32688-5]

Vol. 21: Ang Jr., H.; Khatib, O. (Eds.)
Experimental Robotics IX – The 9th International Symposium on Experimental Robotics
618 p. 2006 [978-3-540-28816-9]

Vol. 20: Xu, Y.; Ou, Y.
Control of Single Wheel Robots
188 p. 2005 [978-3-540-28184-9]

Vol. 19: Lefebvre, T.; Bruyninckx, H.; De Schutter, J. Nonlinear Kalman Filtering for Force-Controlled Robot Tasks
280 p. 2005 [978-3-540-28023-1]

Vol. 18: Barbagli, F.; Prattichizzo, D.; Salisbury, K. (Eds.)
Multi-point Interaction with Real and Virtual Objects
281 p. 2005 [978-3-540-26036-3]

Vol. 17: Erdmann, M.; Hsu, D.; Overmars, M.; van der Stappen, F.A (Eds.)
Algorithmic Foundations of Robotics VI
472 p. 2005 [978-3-540-25728-8]

Vol. 16: Cuesta, F.; Ollero, A.
Intelligent Mobile Robot Navigation
224 p. 2005 [978-3-540-23956-7]

Vol. 15: Dario, P.; Chatila R. (Eds.)
Robotics Research – The Eleventh International Symposium
595 p. 2005 [978-3-540-23214-8]

Vol. 14: Prassler, E.; Lawitzky, G.; Stopp, A.; Grunwald, G.; Hägele, M.; Dillmann, R.; Iossifidis. I. (Eds.)
Advances in Human-Robot Interaction
414 p. 2005 [978-3-540-23211-7]

Vol. 13: Chung, W.
Nonholonomic Manipulators
115 p. 2004 [978-3-540-22108-1]

Vol. 12: Iagnemma K.; Dubowsky, S.
Mobile Robots in Rough Terrain – Estimation, Motion Planning, and Control with Application to Planetary Rovers
123 p. 2004 [978-3-540-21968-2]

Vol. 11: Kim, J.-H.; Kim, D.-H.; Kim, Y.-J.; Seow, K.-T.
Soccer Robotics
353 p. 2004 [978-3-540-21859-3]