A Perception-Driven Autonomous Urban Vehicle

John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, Olivier Koch, Yoshiaki Kuwata, David Moore, Edwin Olson, Steve Peters, Justin Teo, Robert Truax, and Matthew Walter
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
e-mail: jleonard@mit.edu

David Barrett, Alexander Epstein, Keoni Maheloni, and Katy Moyer
Franklin W. Olin College
Needham, Massachusetts 02492
e-mail: david.barrett@olin.edu

Troy Jones and Ryan Buckley
Draper Laboratory
Cambridge, Massachusetts 02139
e-mail: tbjones@draper.com

Matthew Antone
BAE Systems Advanced Information Technologies
Burlington, Massachusetts 01803
e-mail: matthew.antone@baesystems.com

Robert Galejs, Siddhartha Krishnamurthy, and Jonathan Williams
MIT Lincoln Laboratory
Lexington, Massachusetts 02420
e-mail: galejs@ll.mit.edu

Received 10 February 2008; accepted 22 July 2008

Journal of Field Robotics 25(10), 727–774 (2008)
Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/rob.20262

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 kinodynamic 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 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 global positioning system–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 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 vehicle was assigned individual missions. These missions were designed by DARPA to require each team to complete 60 miles within 6 h 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 h before the race. The RNDF was very similar to a digital street map used by an in-car global positioning system (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 that 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 trees (RRT)–based planning algorithm, achieving the requirements of driving in lanes, executing three-point turns, parking, and maneuvering through

Figure 1. Talos in action at the NQE.

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 35 teams that participated in the DARPA Urban Challenge NQE and was one of 11 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 (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 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 kinodynamically feasible vehicle trajectory that moves toward 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

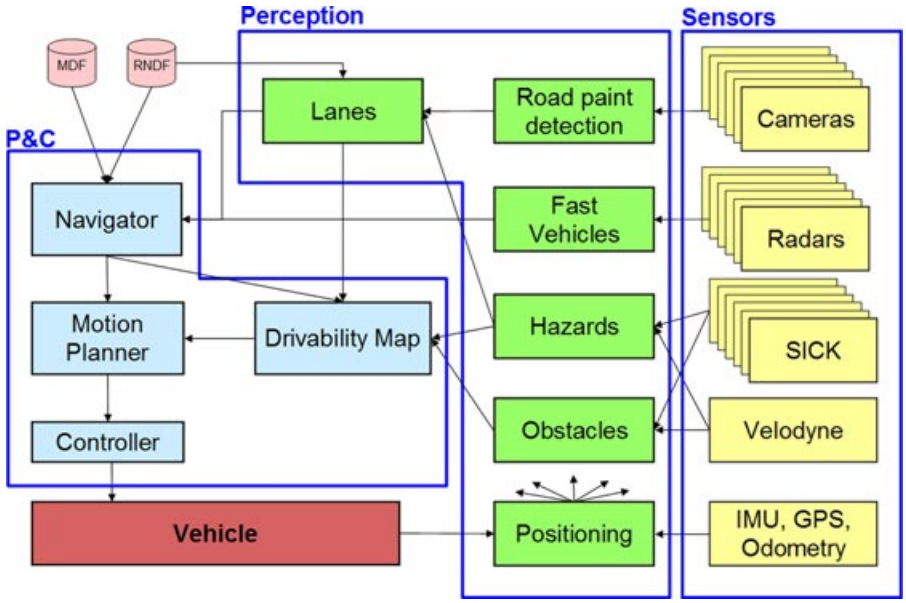


Figure 2. System architecture.

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 Ethernet protocol 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 three-dimensional (3-D) visualization of experimental data.

3. INFRASTRUCTURE DESIGN

Achieving an autonomous urban driving capability is a difficult multidimensional 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; Trepagnier, Nagel, Kinney, Koutsourgeras, & Dooner, 2006;

Urmson 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 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.
- Because 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 multisensor strategy also permits more effective distribution of input/output (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 nonlinear system 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 nonnegligible. 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 and then publishes the sensor data on a shared network. A "drivability map" application programming interface (API) (described more fully below) performs minimal sensor fusion, simply by

(a) (b)

Figure 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.

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. Although 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) (Figure 3) 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 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, 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 EMC in Baton Rouge, Louisiana. They installed computer-controlled servos on the gear shift and 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 240 V as opposed to the standard 120 V and could consume up to 4,000 W, dictating many of the power and cooling design decisions. Primary power for the system came from an internally mounted Honda 6,000-W R/V (recreational vehicle or mobile home) generator. It draws fuel directly from the LR3 tank and produces 120 and 240 V ac 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 240-V ac power is fed to twin Acumentrics rugged UPS 2500 units that 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 noncritical 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) two-dimensional (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. When possible, sensors were connected as Ethernet devices. In contrast to many commonly used standards such as RS-232, RS-422, serial, controller area network (CAN), universal serial bus (USB), or Firewire, Ethernet offers, in one standard, electrical isolation, radio frequency (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 FOV above the SICK LIDAR units and the air conditioner. The Velodyne is a 3-D laser scanner composed of 64 lasers mounted on a spinning head. It produces approximately a million range samples per second, performing a full 360-deg sweep at 15 Hz.

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, and the remaining seven are mounted lower around the sides of the vehicle and project outward parallel to the ground plane.

Each SICK sensor generates an interlaced scan of 180 planar points at a rate of 75 Hz. 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 1-deg 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 FOV of these radars (around 18 deg) required a tiling of 15 radars to achieve the desired 240-deg FOV. The radars require a dedicated CAN bus interface each. To support 15 CAN bus networks, we used eight 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-deg FOV. Each camera was operated at 22.8 Hz and produced Bayer-tiled images at a resolution of 752×480. This amounted to 39 MB/s of image data, or 2.4 GB/min. To support multiple parallel image processing algorithms, camera data were JPEG-compressed and then retransmitted 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. Although 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 an 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 either generated invalid commands or stopped sending commands entirely.

The ADU also implemented the interface to the buttons and displays in the cabin, and the DARPA-provided 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 in which 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.

Whereas 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 among 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 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 time-stamped 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 data sets 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 play back, extracting portions of a logfile to a separate smaller logfile, and the selected play back 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. Although 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 1 min 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 needed only 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 100 Hz.

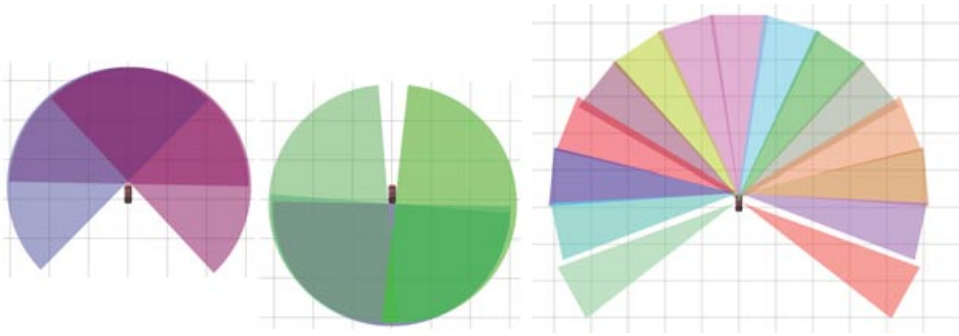
4.2. Obstacle Detector

The system's large number of sensors provided a comprehensive FOV and provided redundancy both within and across sensor modalities. LIDARs provided near-field obstacle detection (Section 4.2.1), and 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, Mano, & Shashua, 2000, 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 & Broggi, 1998).

A notable detection and tracking system for urban traffic from LIDAR data was developed by Wang (2004), who incorporated dynamic object tracking in a 3-D simultaneous localization and mapping (SLAM) system. 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 & 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, 2007) provide alternative examples of successful approaches. Tartan Racing's vehicle tracker built on the algorithm of 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



(a) (b)

Figure 4. Sensor FOVs (20-m grid size). (a) Our vehicle used seven horizontally mounted 180-deg planar LIDARs with overlapping FOVs. The three LIDARs at the front and the four 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-deg radars yield a wide FOV.

obstacles and ground plane measurements, as well as returns from areas outside of the RNDF. The remaining returns are fitted 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 multisensor calibration technique.

Our obstacle detection system combines data from 7 planar LIDARs oriented in a horizontal configuration, a roof-mounted 3-D LIDAR unit, and 15 automotive radars. The planar LIDARs were SICK units returning 180 points at 1-deg spacing, with scans produced at 75 Hz. We used SICK's "interlaced" mode, in which every scan is offset 0.25 deg from the previous scan; this increased the sensitivity of the system to small obstacles. For its larger FOV and longer range, we used the Velodyne "high-definition" LIDAR, which contains 64 lasers arranged vertically. The whole unit spins, yielding a 360-deg scan at 15 Hz.

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. Because each radar has a narrow 18-deg FOV, we arranged 15 of them in an overlapping, tiled configuration in order to achieve a 256-deg FOV.

The planar LIDAR and radar FOVs are shown in Figure 4. The 360-deg FOV of the Velodyne is a ring around the vehicle stretching from 5 to 60 m. A wide FOV 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 fewer 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

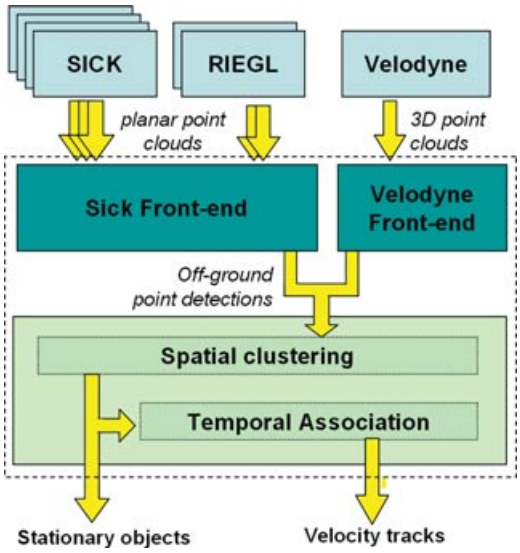


Figure 5. LIDAR subsystem block diagram. LIDAR returns are first classified as "obstacle," "ground," or "outlier." Obstacle returns are clustered and tracked.

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. Because 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 (75 Hz versus the Velodyne's 15 Hz) also makes data association of fast-moving obstacles easier.

Each LIDAR produces a stream of range and angle tuples; these data are 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 offline). The result is a stream of 3-D 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 "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 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 nonflat 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 FOVs 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

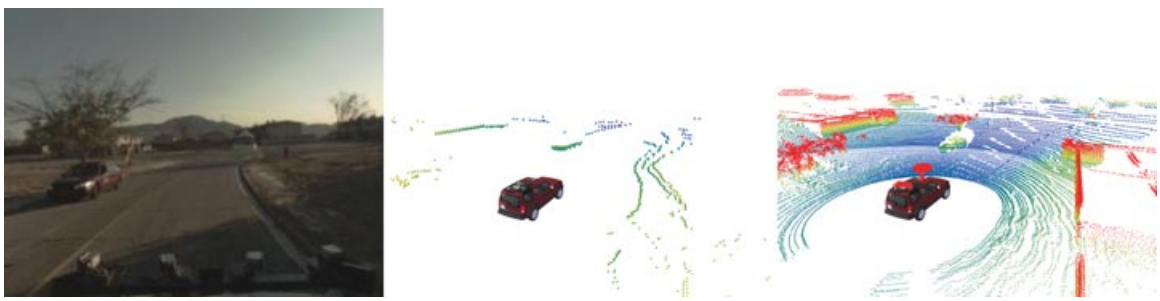


Figure 6. Raw data. Left: camera view of an urban scene with oncoming traffic. Middle: corresponding horizontal planar LIDAR data ("pushbroom" LIDARs not shown for clarity). Right: Velodyne data.

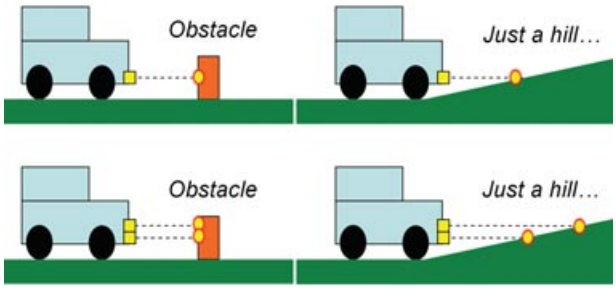


Figure 7. Obstacle or hill? With a single planar LIDAR, obstacles cannot be reliably discriminated from traversable (but hilly) terrain. Multiple planar LIDARs allow appreciable changes in z to be measured, resolving the ambiguity.

highly reflective and cannot be reliably seen by SICK sensors. Even at a distance of under 2 m, 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 deglitch the raw range returns. Any returns that are farther than 1 m 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 2-D grid at 25-cm resolution representing an area of 200×200 m 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 time stamp 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 both returns are passed to the obstacle tracker. As this search proceeds, returns older than 33 ms 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 (because 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 and then fit a nonparametric 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 contain 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. Retroreflectors 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.

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 4 deg of sensor rotation and 3 m 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. Owing to the noise in the Velodyne, the candidate point is not the lowest point;

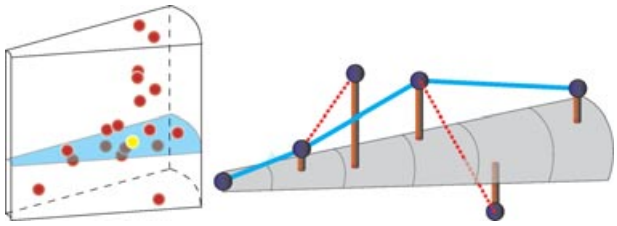


Figure 8. Ground candidates and interpolation. Velodyne returns are recorded in a polar grid (left: single cell is shown). The lowest 20% (in z height) are rejected as possible outliers; the next lowest return is a ground candidate. A ground model is linearly interpolated through ground candidates (right), subject to a maximum slope constraint.

instead, the lowest 20% of points (as measured by z) are discarded before the next lowest point is accepted as a candidate point.

Whereas 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 deg). 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 not only serves 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.

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 15 cm: constantly changing intrinsic variations across lasers make it impossible to reliably measure, across lasers, height changes smaller than

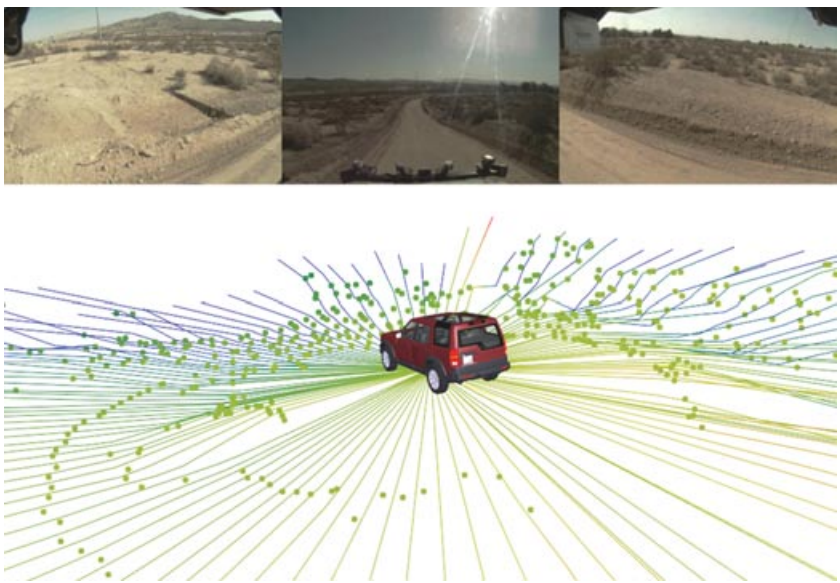


Figure 9. Ground model example. On hilly terrain, the terrain deviates significantly from a plane but is tracked fairly well by the ground model.

this. Thus the overlying algorithm cannot reliably detect obstacles shorter than about 15 cm.

For each polar cell, we tally the number of returns generated by each laser that is above the ground by an "evidence" threshold (nominally 15 cm). Then, we consider each return again: those returns that are above the ground plane by a slightly larger threshold (25 cm) and are supported by enough evidence are labeled 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 (15 cm) and the "obstacle" threshold (25 cm) is designed to increase the sensitivity of the obstacle detector to low-lying obstacles. If we used the evidence threshold alone (15 cm), we would have too many false positives because that threshold is near the noise floor. Conversely, using the 25-cm threshold alone would require obstacles to be significantly *taller* than 25 cm because 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 3 ms (the delay between the first and last laser to scan at a particular bearing) rather than every 66 ms (the rotational period of the sensor). During the saved 63 ms, a car traveling at 15 m/s would travel almost 1 m. 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.

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 grid map.

Chunks are indexed using a 2-D lookup table with about 1-m resolution. Finding the chunk nearest a point p involves searching through all the grid cells that could contain a chunk that contains p. But because 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 2-D 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 (35 cm), 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 reexamined. If a new point has not been assigned to the chunk within the last 250 ms, the chunk expires and is removed from the system.

A physical object is typically represented by more than one chunk. To compute the velocity of obstacles, we must know which chunks correspond to the same physical objects. To determine this, we clustered chunks into groups; any two chunks within 25 cm of one another were grouped 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** $c \in$ 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.

Algorithm 1 requires a relatively small amount of CPU time. The time required to search within a fixed radius of a particular chunk is in fact $O(1)$, because 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 2-D lookup table that stores all chunks. The cost of merging subgraphs, implemented by the Union-Find algorithm (Rivest & 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 15 Hz. 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. Because 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 (Figure 10), 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.)

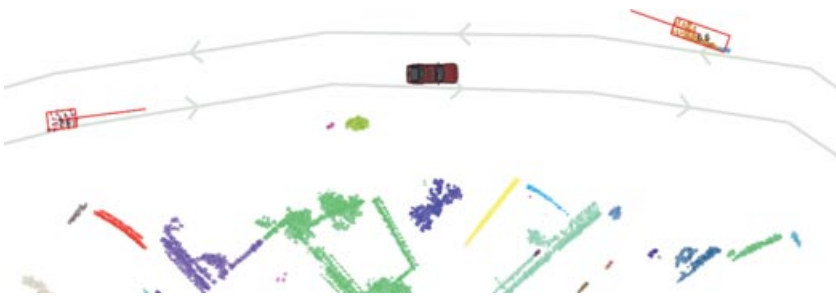


Figure 10. LIDAR obstacle detections. Our vehicle is in the center; nearby (irregular) walls are shown, clustered according to physical proximity to each other. Two other cars are visible as bounding boxes with lines indicating the estimated velocities: an oncoming car ahead and to the left and another vehicle following us (a chase car). The long lines with arrows indicate the nominal travel lanes: they are included to aid interpretation but were not used by the tracker.

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 because 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 3 m/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. Whereas 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.

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. To quantitatively measure the effectiveness of the planar LIDARs (as a set) versus the 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 the following:

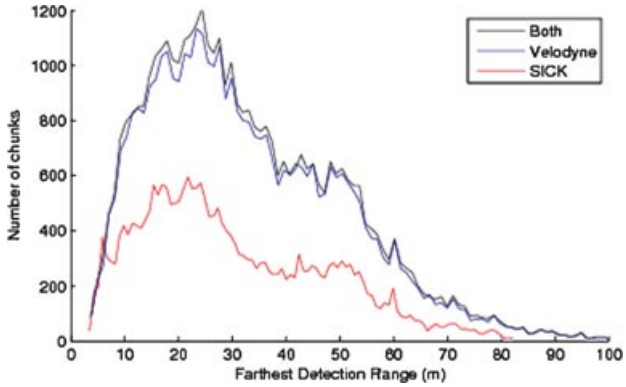


Figure 11. Detection range by sensor. For each of 40,000 chunks, the earliest detection of the chunk was collected for each modality (Velodyne and SICK). The Velodyne's performance was substantially better than that of the SICK's, which observed fewer objects.

- 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: 1.2 m on average.

In complex environments, such as the one used in this data set, the ground is often nonflat. As a result, planar LIDARs often find themselves observing sky or dirt. Whereas 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 FOV, 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 their higher rotational rate (75 Hz versus the Velodyne's 15 Hz), which makes data association easier for fast-moving

(a) (b)

Figure 12. Radar tracking three vehicles. (a) Front right camera showing three traffic vehicles, one oncoming. (b) Points: Raw radar detections with tails representing the Doppler velocity. Rectangles: Resultant vehicle tracks with speed in meters per second (rectangle size is simply for visualization).

obstacles. If another vehicle is moving at 15 m/s, the Velodyne will observe a 1-m displacement between scans, whereas the SICKs will observe only a 0.2-m 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 FOV of 18 deg. To achieve a wide FOV, 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 2 m/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 6.0 m 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 because the target may be accelerating. For simplicity, we fitted the constant-velocity model using just the (x, y) points; whereas 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 were 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 (such as a bolt lying on the ground, or a sewer grate) and large objects (such as cars). To avoid false positives, we used the radars only to detect moving objects.

4.3. Hazard Detector

We define hazards as objects that we *shouldn't* drive over, even if the vehicle probably could. Hazards include potholes, 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 downward-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 grid map that records the largest slope observed in every cell. This grid map is slowly built up over time as the sensors pass over new ground and extended for about 40 m in every

direction. Data that "fell off" the grid map (by being more than 40 m away) were forgotten.

The Velodyne sensor, with its 64 lasers, could observe a large area around the vehicle. However, hazards can be detected only 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 heights 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 reobserved over time, the hazard detector is significantly more stateful because 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 samples the change in height only 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 that 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 grid map 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 because 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 100-ms integration periods are stored in the grid map. Next, 3×3 neighborhoods of the grid map 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 FOV, 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 later in Figure 26(a). The hazard map shows lighter regions on the road ahead and behind the vehicle indicating smooth terrain. Along the edge of the road darker regions reflect uneven terrain such as curb cuts or berms.

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, because 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.

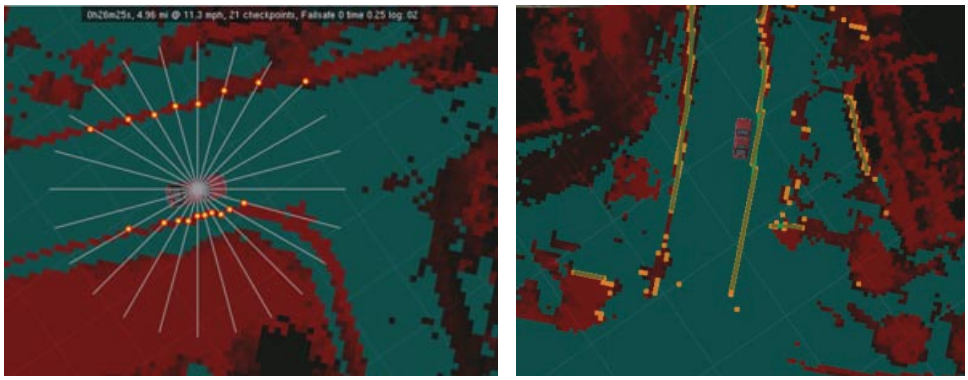
4.3.3. Road-Edge Detector

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 grid map [see Figure 13(a)]. This results in a number of road-edge point detections; these are segmented into chains based on their physical proximity to each other. A nonparametric 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



(a) (b)

Figure 13. Hazard map: Light regions on the road indicate smooth terrain. Dark regions along the road edge indicate uneven terrain such as curb cuts or berms. (a) Rays radiating from vehicle used to detect the road edge. (b) Polylines fitted to road edge.

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 nonparametric 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 allow us to project the expected location of the ground plane into the image, providing a useful prior for the paint-detection algorithms.
- 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

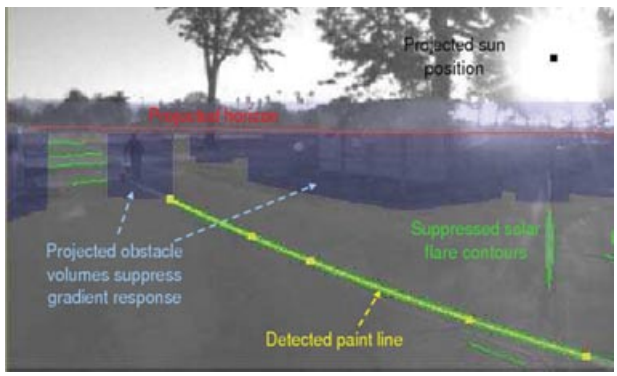


Figure 14. Use of absolute camera calibration to project real-world quantities into the image.

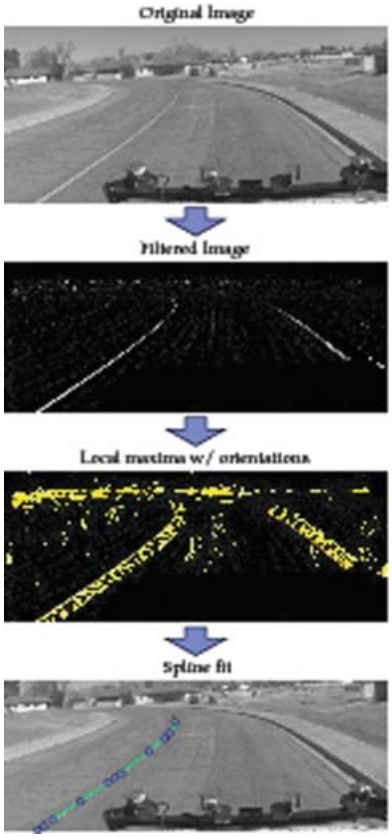


Figure 15. The matched filter–based detector from start to finish. The original image is convolved with a matched filter at each row (horizontal filter shown here). Local maxima in the filter response are enumerated and their dominant orientations computed. The figure depicts orientation by drawing the perpendiculars to each maximum. Finally, nearby maxima are connected into cubic hermite splines.

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.

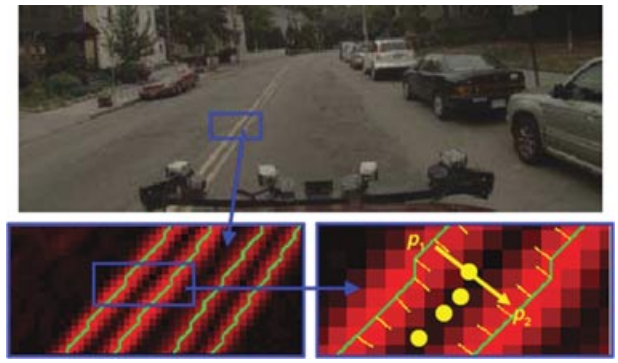


Figure 16. Progression from original image through smoothed gradients, border contours, and symmetric contour pairs to form centerline candidate.

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 nonmaximal 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 a lane. Second, parabolic segments are fitted 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 one-half lane width from a strip of road paint or a curb. Second, 88% of federally managed lanes in the United States are between 3.05 and 3.66 m wide (USDOT Federal Highway Administration, Office of Information Management, 2005). Third, a curb gives us information about the presence of a lane different from that of road paint. From these observations and the characteristics of our road-paint and curb detectors, we define two functions $f_{\text{rp}}(x)$ and $f_{\text{cb}}(x)$, where x is the Euclidean distance from d_i to $\mathbf{p}$:

$$f_{\text{rp}}(x) = -e^{-x^2/0.42} + e^{-(x-1.83)^2/0.14}, \quad (1)$$

$$f_{\text{cb}}(x) = -e^{-x^2/0.42}. \quad (2)$$

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

$$g(d_i, \mathbf{p}) = \begin{cases} 0 & \text{if } \mathbf{c} \text{ is an endpoint of } d_i \\ f_{\text{rp}}(||\mathbf{p} - \mathbf{c}||) & \text{if } d_i \text{ is road paint} \\ f_{\text{cb}}(||\mathbf{p} - \mathbf{c}||) & \text{if } d_i \text{ is a curb} \end{cases},$$

This last condition is introduced because road paint and curbs are observed only 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 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 & 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 & 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 refitted 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 1-m 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:

$$\text{score} = \sum_{\mathbf{p} \in \text{L}} \frac{1}{1 + \mathbf{p}^T \mathbf{C} \mathbf{p}}.$$

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, it is discarded.

Figure 17. Our system constructs a centerline evidence image using road-edge and road-paint detections. Lane centerline candidates (lines) are identified by fitting parabolic segments to the ridges of the image. Front-center camera is shown in top left for context.

After a number of RANSAC iterations (we found 200 to be sufficient), the parabola with the greatest score is selected as a candidate lane centerline. Its inliers are removed from the set of ridge points, and all remaining parabolas are refitted and rescored using this reduced set of ridge points. The next best-scoring parabola is chosen, and this process is repeated to produce at most five 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 75 m. The centerline of each lane is modeled as a piecewise linear curve, with control points spaced approximately every 2 m. 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 whether 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 on only the closest segment of the candidate and not on the entire candidate, on the basis of 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 fail-safe 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 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-m horizon. It then becomes the motion planner's responsibility to avoid obstacles and 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, Nilsson, & 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. Because this search is run continuously at 2 Hz, dynamic replanning comes "for free" as conditions change because 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 because 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.

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.

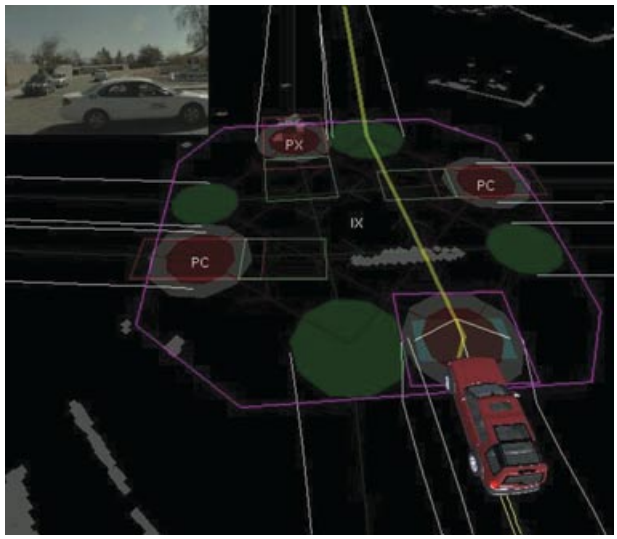


Figure 18. The Navigator's view of intersection precedence. PX means that there is a car with precedence at that entrance, and PC means that there is no car or that the car at the stop line does not have precedence. IX means that there is a moving object in the intersection. Talos is clear to proceed when all PX states have transitioned to PC and IX has transitioned to IC.

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 4 s, 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 1 s. 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 s, Talos yields to the moving traffic. Talos came to a full stop whenever the vehicle was on an RNDF "exit" that crossed another RNDF exit and both did 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 Fail-Safe Modes

To handle unexpected or unpredictable events that could occur in an urban environment, we use fail-safe and blockage timers. The *fail-safe timer* ticks upward from zero whenever the vehicle is not making forward progress. Upon making progress, the timer resets to zero. Upon reaching 80 s, we increment the *fail-safe mode* and reset the timer back to zero. Thus, normal operation is fail-safe mode 0. Once Talos is in fail-safe mode 1, the vehicle has to traverse a predetermined distance in the RNDF before the mode is decremented back to zero. This combination of timer and mode ensures that the fail-safe 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 fail-safe mode and timer to encourage the vehicle to get "un-stuck."

The following summarizes the multiple layers of fail-safe logic implemented in various modules:

- Fail-safe mode 0:
  - 10 s: Relax the center line constraint of the road to allow passing.
  - 10 s: Shrink the margin retained around obstacles from 30 to 15 cm.
  - 15 s: Enable reverse gear motion plans.
  - 20 s: Shrink the margin retained around obstacles from 15 to 0 cm.
  - 30 s: Make curbs drivable with a high penalty instead of impassable.
  - 35 s: Unrestrict the area around the goal.
  - 80 s: Go to fail-safe mode 1.
- Fail-safe mode 1:
  - 0 s: Do not observe standoff distances from stationary obstacles.
  - 0 s: Allow crossing of zone boundaries anywhere.
  - 80 s: Go to fail-safe mode 2.
- Fail-safe mode 2:
  - 0 s: Do not observe standoff distances from moving obstacles.
  - 0 s: Drop lane constraints completely and navigate as if the area were an obstacle field.
  - 0 s: 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 s: Skip the next MDF checkpoint.
  - 80 s: Go to fail-safe mode 3.
- Fail-safe mode 3:
  - 0 s: Restart all the processes except the logger, autonomous drive unit (ADU), and process manager. Because the navigator is restarted, Talos will be in fail-safe mode 0 after the restart.

In addition, detection parameters for the underlying obstacle detectors are relaxed in higher fail-safe modes, although never to the point that Talos would drive into a clearly visible obstacle.

The *blockage timer* behaves similarly to the fail-safe timer but ticks upward only if Talos is on a two-way road where a U-turn is possible. If the timer reaches 50 s 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. To ensure that the maneuver being executed is not interrupted, the fail-safe 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 are 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 be entered only if the vehicle can stop in an unrestricted area farther ahead. Restricted regions are used to permit minor violations of the lane boundaries if progress is made 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.

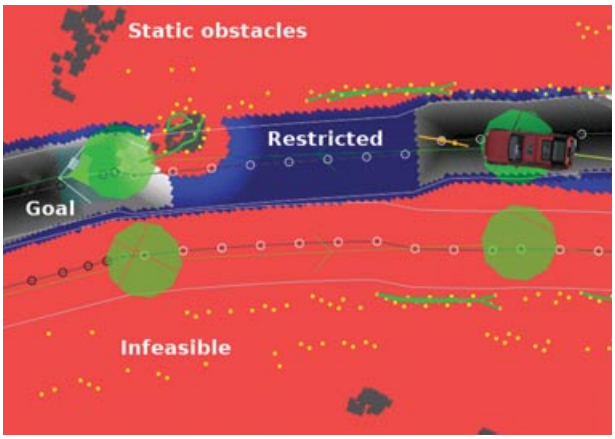


Figure 19. Drivability map visualization Arrow (left): Short-term goal location. Light regions (adjacent to the road): Infeasible regions are off limits to the vehicle. Dark region (in front of the vehicle): Restricted regions may be entered only if the vehicle can stop in an unrestricted region farther ahead. Shades of gray: High-cost regions indicate regions accessible to the vehicle.

As discussed previously, the navigator contains a cascade of events triggered by a prolonged lack of progress. For example, after 10 s of no progress queuing behind a stationary vehicle, the navigator will enter the passing mode. This mode amounts to a relaxation of the lane centerline constraint. The drivability map will 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 30 cm 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.

Object tracks where the intended path was not known, such as in intersections or zones, were propagated by 3 s using a constant-velocity model.

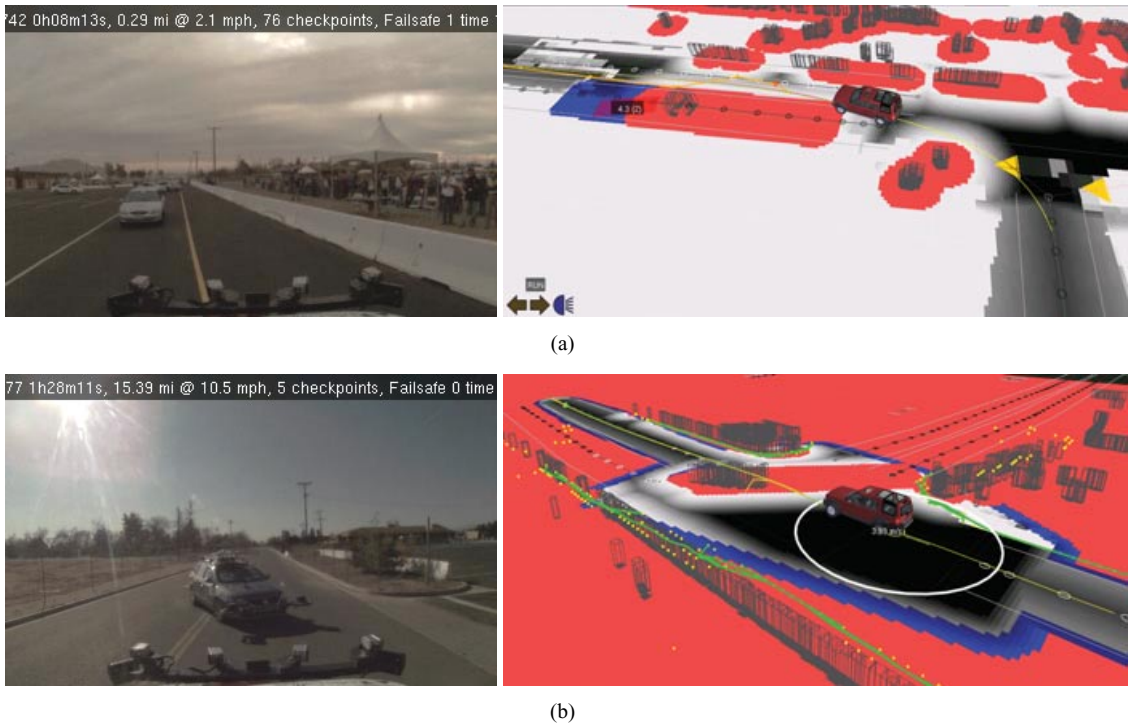


Figure 20. (a) An infeasible region is projected down the lane excluding maneuvers into oncoming vehicles. (b) Within an intersection an infeasible region is created between a moving obstacle and the exit matching the velocity direction.

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 in which 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.

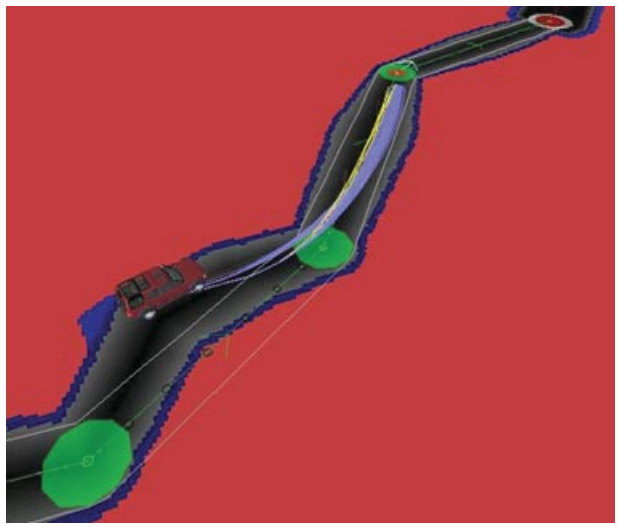


Figure 21. "Stretchy" lane adjustment. When the vehicle is off the lane, the lane is adjusted so that the vehicle does not brake as a result of crossing the lane boundary.

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.

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 RRT (LaValle & Kuffner, 2001), where the tree of kinodynamically 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. 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.

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 Drivability Map
21:   if The best trajectory is infeasible then
22:      Remove the infeasible portion from the tree and
         Go to line 2
23:   end if
24:   Send the best trajectory to the controller
25: until Vehicle reaches the target.
```

5.3.1. Planning over Closed-Loop Dynamics

The first extension is to sample the input to the controller and run closed-loop simulations. 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 that samples be taken at a very high rate or that an arbitrary smoothing process be used. By first closing the loop on the vehicle with a 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 (set of line-connected points) and the predicted trajectory (complementary set of curved paths).

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, because 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.

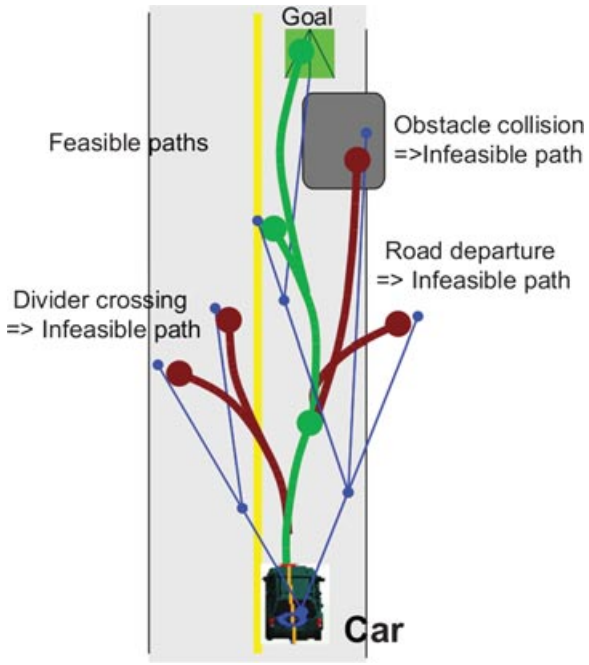


Figure 22. Illustration of RRT motion planning. Each leaf of the tree represents a stopping location. The motion control points (line-connected points) are translated into a predicted path. The predicted paths are checked for drivability (curved paths).

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 an indefinite period of time and remain safe (Schouwenaars, How, & Feron, 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

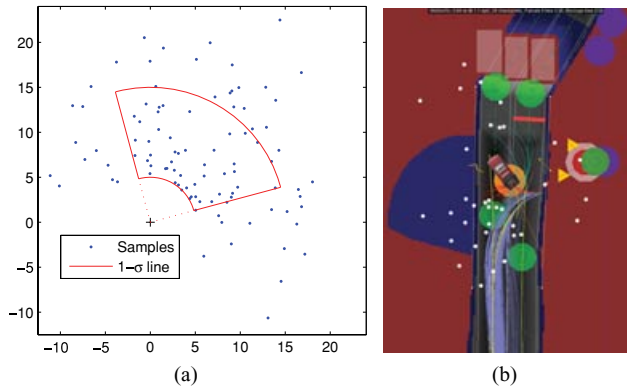


Figure 23. (a) Biased Gaussian samplings. The x, y axes are in meters. (b) Biased sampling for three-point turns: Talos shown in position after the first forward leg of the turn.

to bias the sampling. The samples are taken in two dimensions, 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 $1-\sigma$ 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 $1-\sigma$ 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, $\sigma_{r1} = 8$, $\sigma_{\theta 1} = \pi/10$, $r_{01} = 3$, and $\theta_{01} = 4\pi/9$; $\sigma_{r2} = 10$, $\sigma_{\theta 2} = \pi/10$, $r_{02} = 5$, and $\theta_{02} = -\pi/4$; and $\sigma_{r3} = 12$, $\sigma_{\theta 3} = \pi/10$, $r_{03} = 7$, and $\theta_{03} = \pi$. Samples for the first leg of the U-turn maneuver are positioned relative to the vehicle. Samples for each subsequent leg of the U-turn maneuver are drawn with respect to the previous sample position.

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 Reevaluation

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 does not reevaluate 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 piecewise 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 & Stentz, 1997; Park, Deyst, & How, 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 & Stentz, 1997; Park et al., 2007) to steer the vehicle along the desired path. The steering control law is given by

$$\delta = -\tan^{-1}\left(\frac{L\sin\eta}{L_1/2 + l_a\cos\eta}\right),$$

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. 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 the 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.

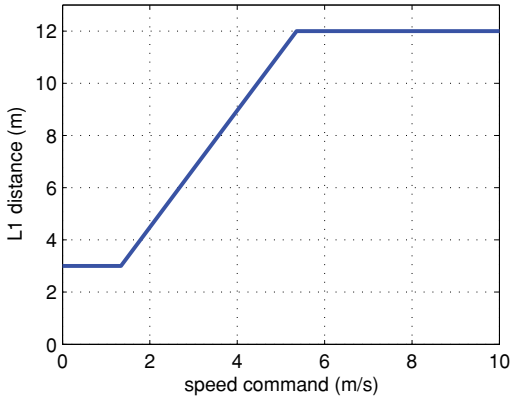


Figure 24. L_1 distance as a function of the commanded speed.

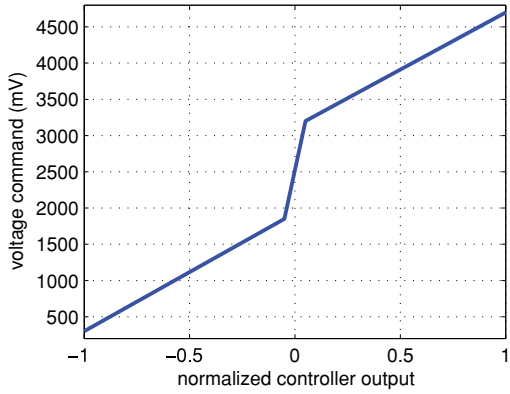


Figure 25. Conversion from the speed controller output to the ADU command voltage.

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 1,850 and 3,200 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 deadband, 3,200 mV, and the small negative output ($u = -0.05$) corresponds to the lower limit of the deadband, 1,850 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 a 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 NQE and then the race itself. This section reviews the vehicle's performance in these events.

6.1. 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 and 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 1 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 3 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:

- no progress due to a road blockage versus a perception failure (such as a misdetected curb cut)

Table I. Results for all of Talos's NQE tests.

Day	Date	NQE schedule	Outcome
1	Sat. 27th Oct.	Area B 1st trial	Completed
2	Sun. 28th Oct.	Area C 1st trial	Completed, but went around the roadblock
3	Mon. 29th Oct.	Area A 1st trial	Completed: safe, but slow (7 laps in 24 min)
4	Tues. 30th Oct.	Area B 2nd trial	Still progressing, but ran out of time
		Area C 2nd trial	Went off-road after 2nd K-turn at blockage
		Area A 2nd trial	Completed: safe and faster (10 laps in 12 min)
		Area B 3rd trial	Completed
		Area C 3rd trial	Completed after recovery from K-turn at first blockage
5	Wed. 31st Oct.	—	

- no progress due to a vehicle to queue behind and pass versus a perception failure (such as 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 the challenge lane was in fact a 6-in. (15 cm) 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 fail-safe logic eventually relaxed the constraint that had been avoiding the ditch. The vehicle then drove down the 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 because of either a close GPS waypoint or lane tracking.

During Talos's 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.

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. Because Talos could then not make progress, it would wait 10 s to determine whether 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 fail-safe 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 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

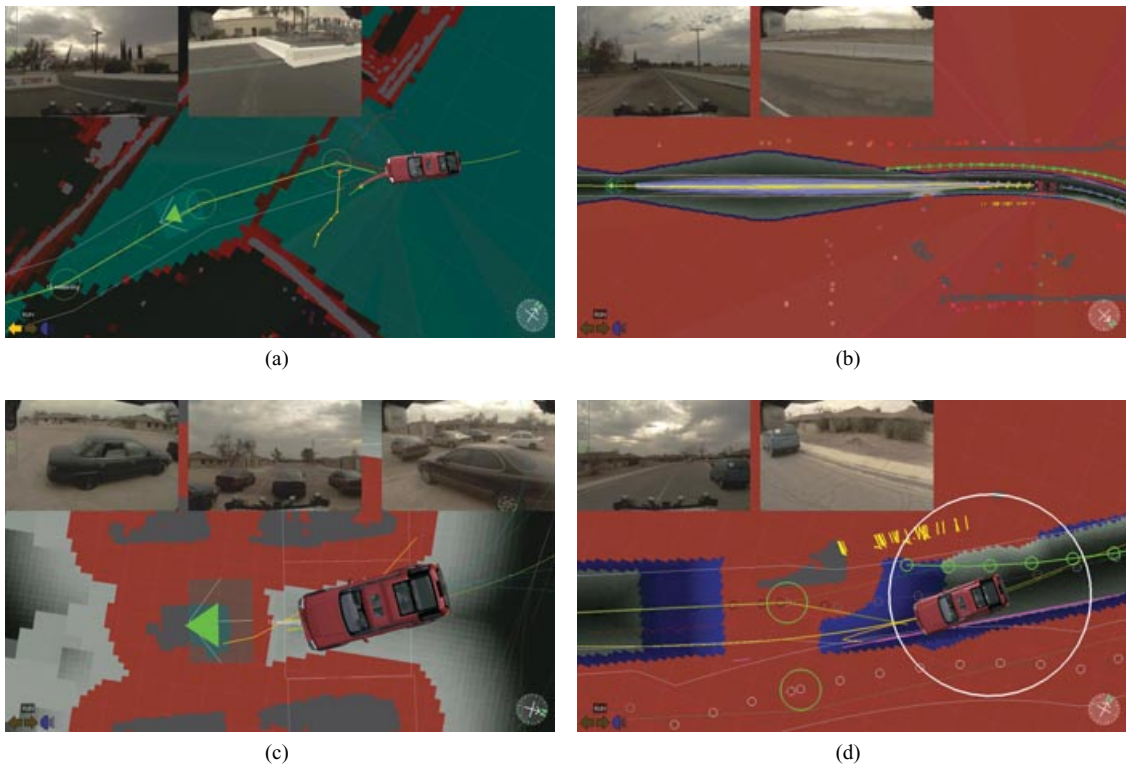


Figure 26. Area B first trial highlights. (a) Road-hazard map showing a line of hazardous terrain across the road at the end of the zone. The drop-off onto the challenge lane was detected as a ditch. (b) Lane position uncertainty between control points reflected by lane dilation. The road past where 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.

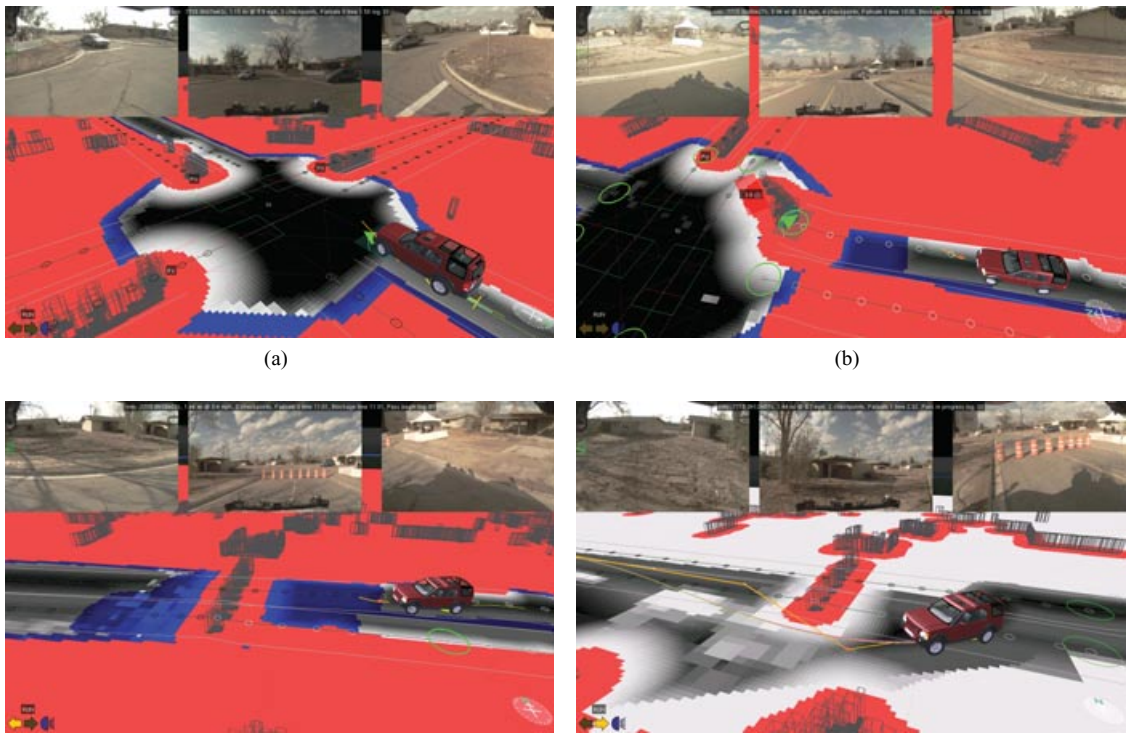
Talos queueing behind a traffic vehicle before giving precedence at the intersection.

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 during which no progress was made, the system would relax 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 fail-safe 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's safety margin for merging from an 8-s window to a 13-s window. As a result, during the Area A trial, Talos performed safely, but very cautiously, as it was waiting for a 13-s window in the traffic, and such a window rarely appeared. In the 24-min trial, Talos completed only seven laps.

Figure 28(a) shows the vehicle track on the right approaching at 3.5 m/s despite being occluded by a vehicle in the closer lane tracked by the radar and LIDAR as traveling at 3.9 and 4.3 m/s, respectively.

Although the fail-safe 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

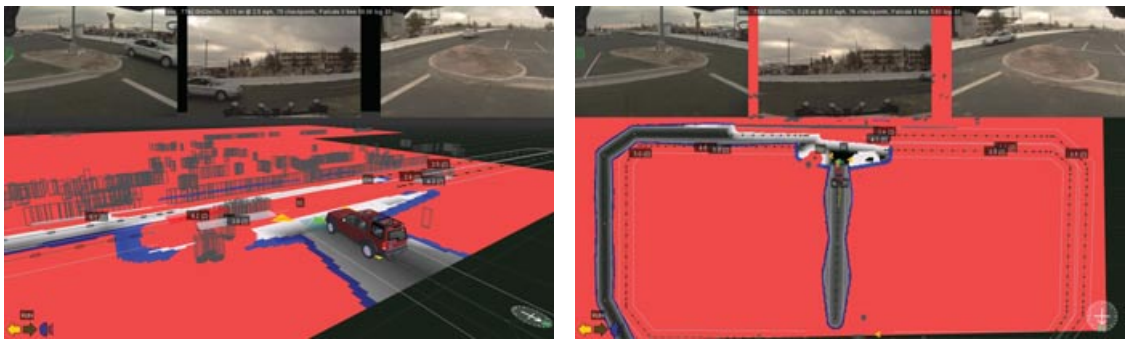


(a) (b)

(c) (d)

Figure 27. Area C first trial highlights. (a) Intersection precedence with four traffic vehicles. (b) Queuing before an intersection. (c) Attempting to go around a blockage. (d) Fail-safe mode 1 permits the vehicle to go around the blockage.

and use it only 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.



(a) (b)

Figure 28. Area A first 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.

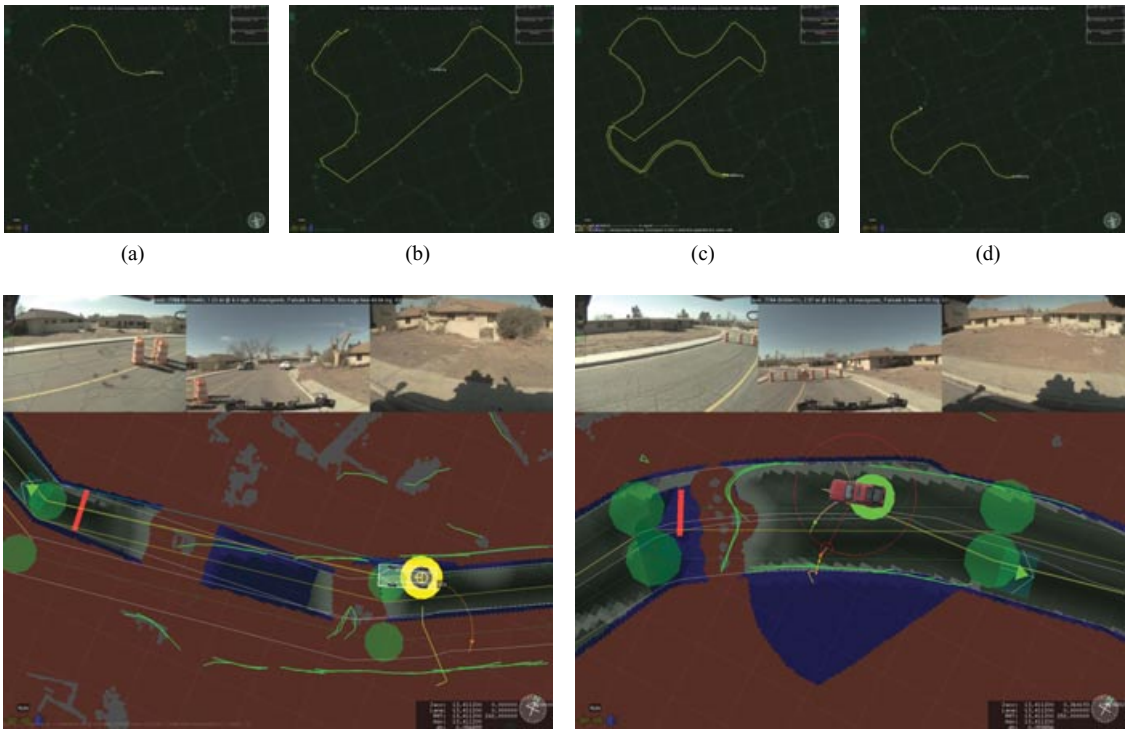
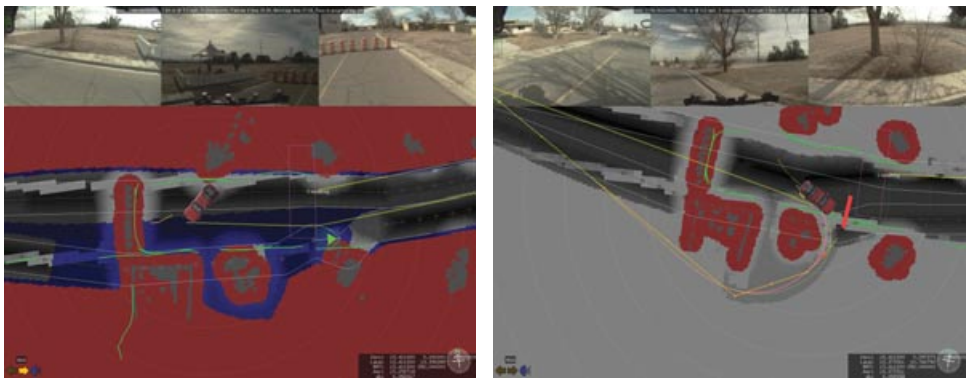


Figure 29. Area B second trial highlights. (a), (b), (c), and (d) show a sequence of navigator plans. After a blockage is declared in the gauntlet, Talos attempts to reach the checkpoint from the opposite direction. (e) Talos gets stuck and cannot go into passing mode due to a poor lane estimate resulting 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.

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 29(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 because 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 fail-safe 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 Talos was enabled to drive across curbs, behind the



(a) (b)

Figure 30. Area C second trial highlights. (a) Progress is stalled during the second K-turn by a poor lane estimate and a restricted region. (b) Fail-safe 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.

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 s, had caused a significant amount of waiting in the first trial) was reduced to 9 s. 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 min. Figure 31 shows a photo of Talos and a screenshot of the viewer output for this mission.

Figure 32 illustrates Talos's performance in its Area B third 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

(a) (b)

Figure 31. Area A second trial highlights. (a) Talos is looking for smaller gaps than in the first trial. (b) The RRT planner is working while waiting for oncoming traffic to clear.

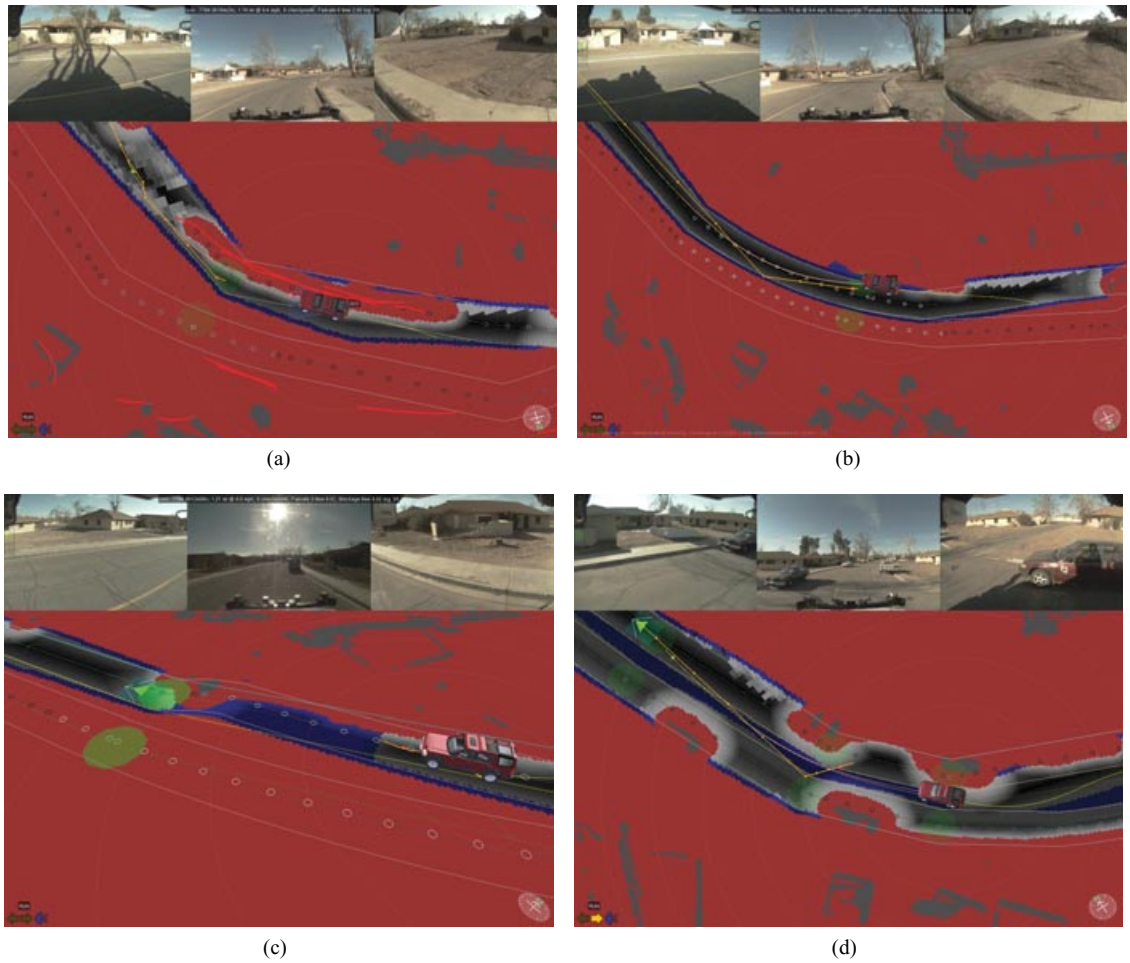
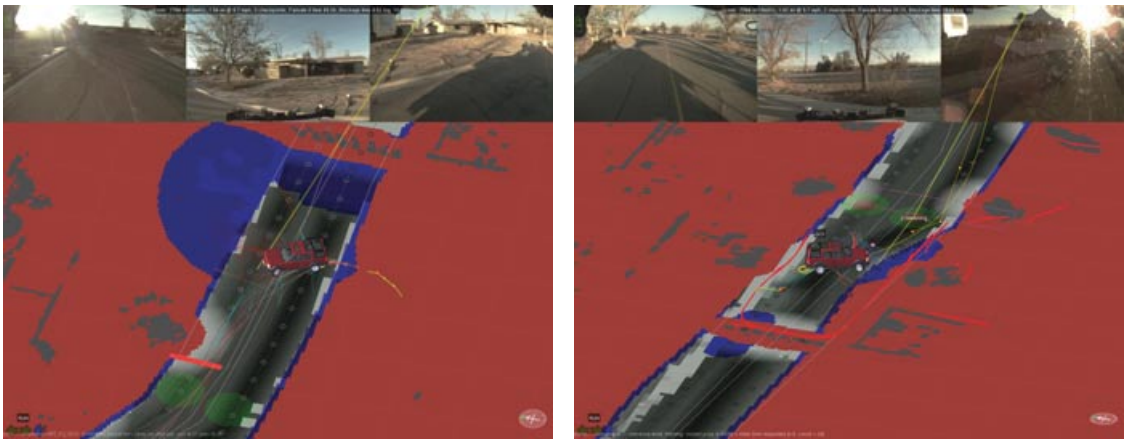


Figure 32. Area B third trial highlights. (a) Without visual lane tracking, a curb-free space algorithm localizes the lane. (b) Visual lane tracking often recovers, providing an improved road estimate. (c) Without a virtual obstacle, passing will still occur as long as the object occupies enough of the lane. Here the parked car still induces a passing behavior. (d) Once in passing mode, parked cars are easily maneuvered around.

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 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 32(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.

Figure 33 illustrates Talos's performance in its Area C third 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.



(a) (b)

Figure 33. Area C third trial highlights. After correctly detecting the blockage, Talos begins a K-turn. The navigator's plan changes. Talos enters fail-safe mode, drives off-road, and is recovered by the pit crew. (b) After the intervention, Talos resumes, and its second K-turn goes well.

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's performance during the UCE. Figure 34(b) shows Talos driving down Phantom East. The vehicle speed was capped at 25 mph as this was the highest speed for which we had validated our vehicle model. The radar in the picture reads 24 mph. 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 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 more than 35 mph. The early detection by Talos's 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, fail-safe mode 1 was entered twice, once at 750 s due to being stuck on the wrong side of an artificial zone perimeter fence. Figure 36 shows how the lane next to the parking zone is narrow in the RNDF (12 ft)—the actual lane is over 30 ft and extends into the zone. The lane perception snaps to the real lane edge, trapping the vehicle against the zone boundary virtual fence. The second fail-safe mode change came at 7,200 s due to a bad lane estimate on a 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 s until the curb constraints were relaxed. In mission 2 the zone perimeter fence bug occurred at 4,000 s. The third mission required three traversals of the gravel road, which caused a large number of 30-s intervals of no progress until curb constraints were relaxed. Once at 7,200 s a bad lane estimate on the gravel road caused Talos to enter fail-safe mode 1.

Outback Road was the Achilles' heel of Talos's UCE performance. We now examine the cause.

6.2.1. Outback Road

As noted above, Talos's third mission during the UCE required three traversals of the steep gravel road known as Outback Road. Figure 37 illustrates why three traversals 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,

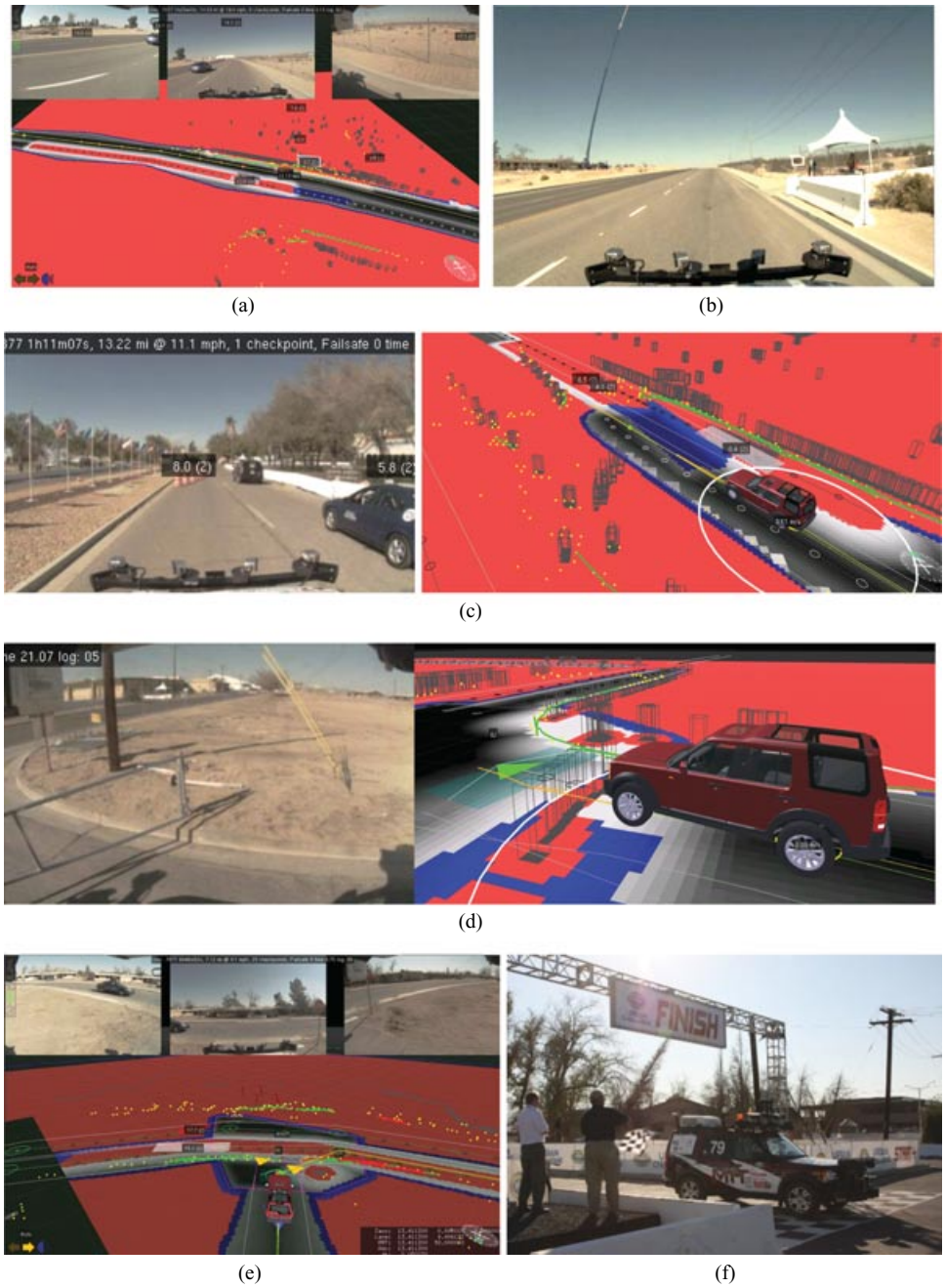
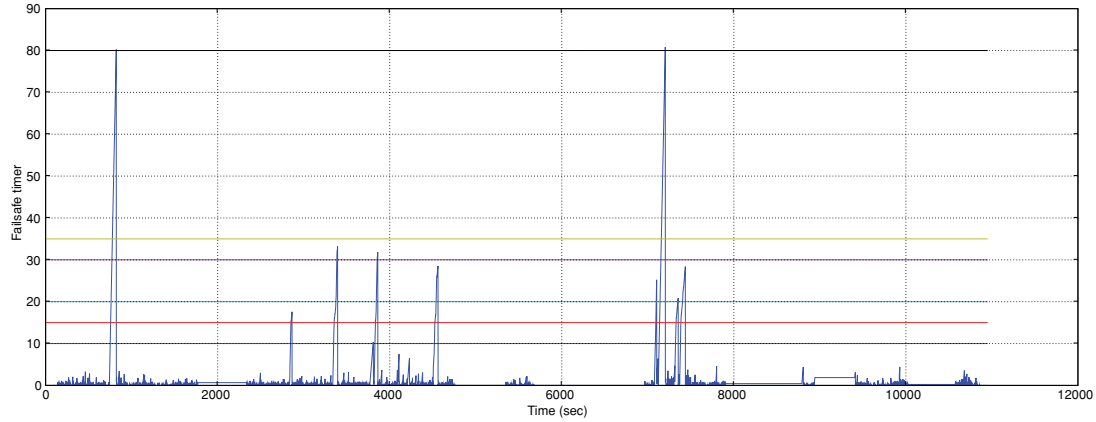
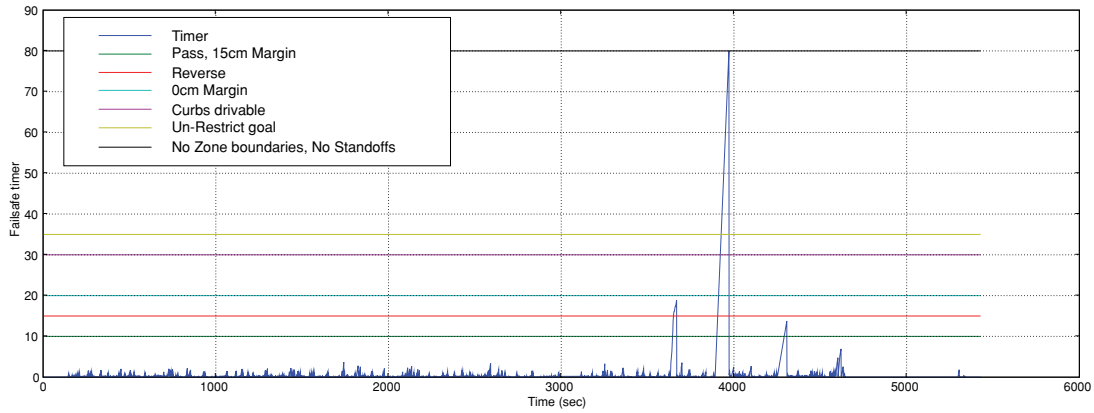


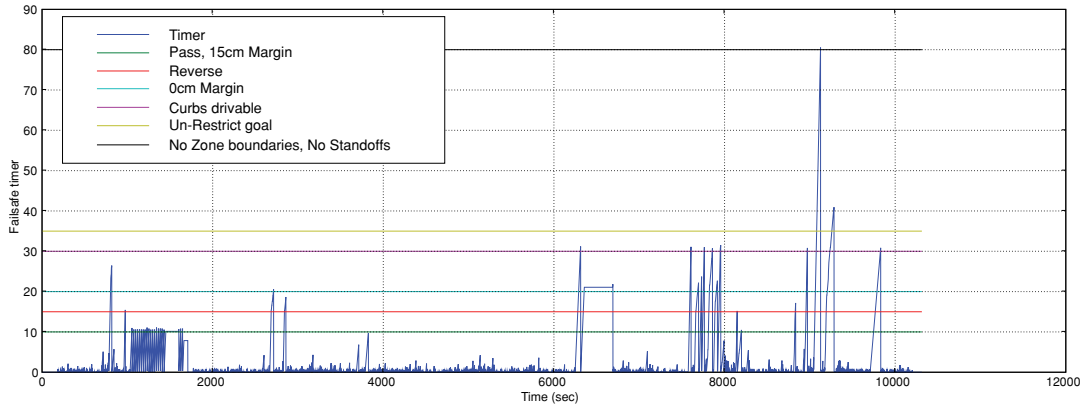
Figure 34. UCE highlights. (a) Talos overtaken by traffic vehicle. (b) Talos reaches its target speed of 25 mph (24 shown on the sign) traveling up Phantom East. (c) Talos slows to merge safely behind the Cornell chase vehicle. (d) Talos waits patiently behind a gate that was blown across an intersection 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

Figure 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, fail-safe mode (80 s of no progress) is entered twice. 750 s: Zone perimeter fence bug. 7,200 s: Bad lane estimate on gravel road. Other times Talos was "ship wrecked" and made no progress for 30 s until curb constraints were relaxed. (b) Mission 2: 4,000 s; zone perimeter fence bug. (c) Mission 3: 7,200 s; bad lane estimate on gravel road. Many times during the three traversals of the gravel road section, no progress was made for 30 s until the curb constraints were relaxed.

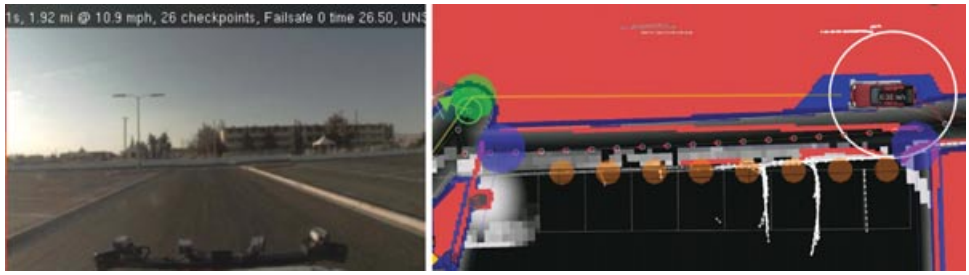


Figure 36. Lane perception snaps the narrow (12 ft) RNDF lane to the (30 ft) actual lane boundary to the left (gray circles, lane centerline; white lines; lane boundary detections). The road would be drivable except that the zone virtual boundary (line to the left of the vehicle) extends into the physical lane blocking the road.

other teams were not required to complete this circuit more than once. Figure 38 provides snapshots of Talos's 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 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 culled. We expected this feature to solve this problem, but it did not. A flat road that curved only 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), which 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 farther 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 time-out elapsed (at which point the road edges were no longer treated as obstacles).

Again, as described earlier, the intended approach of using the curb data in the lane estimate, if mature, would have gone a long way toward addressing this problem.

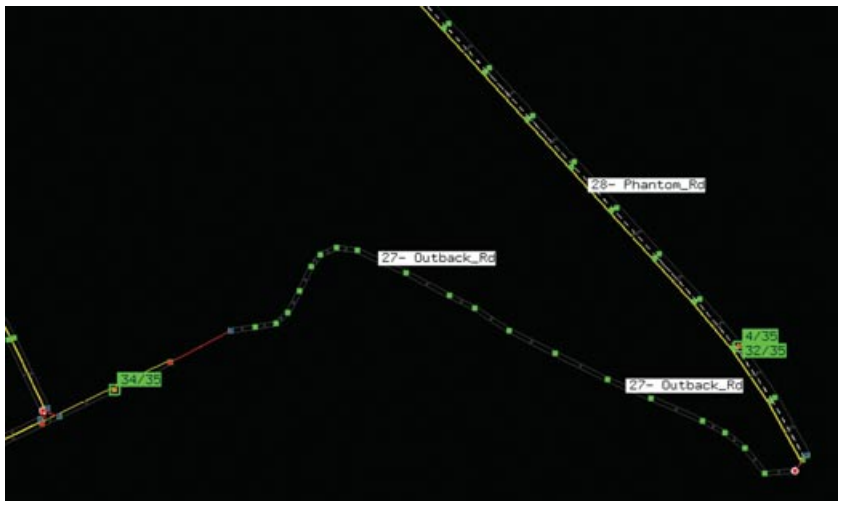


Figure 37. To complete the third mission, three traversals of the Outback Road were required.

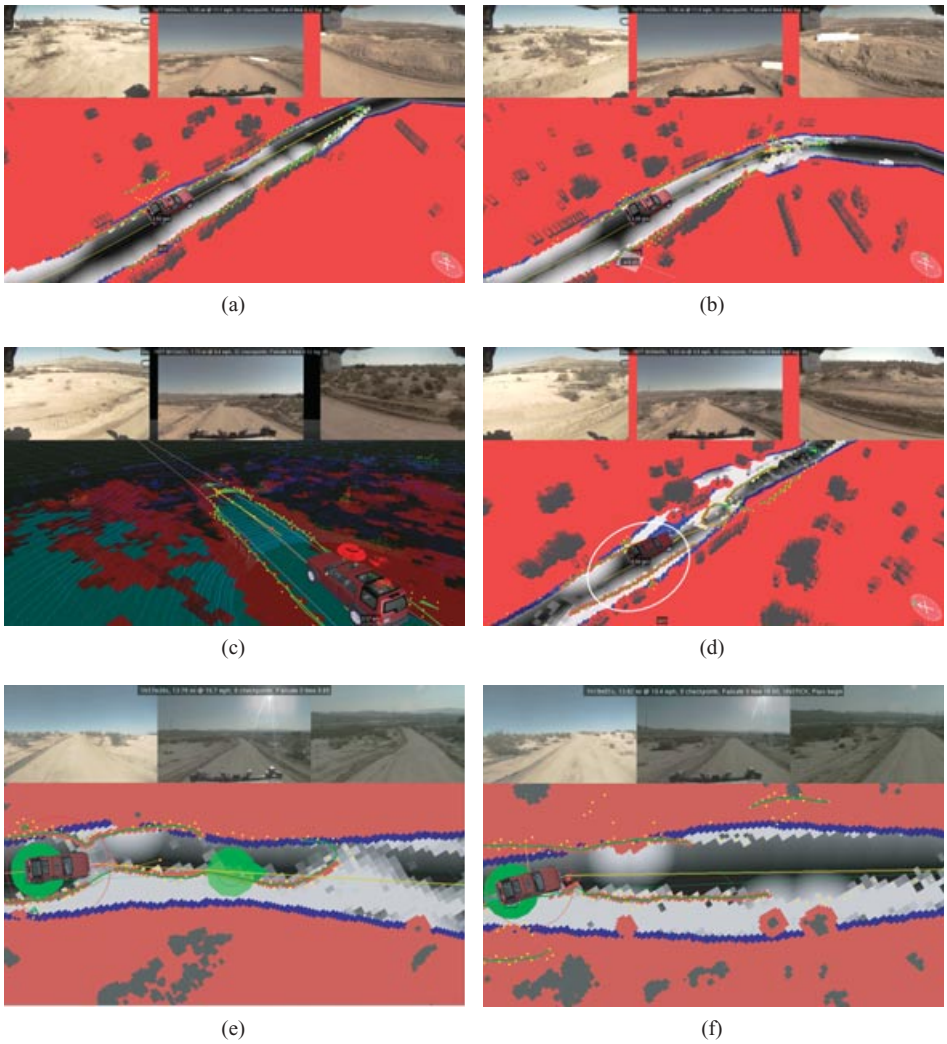


Figure 38. Steep gravel road. (a) and (b) Talos had no problem with gravel roads of gentle slope. (c) Roll-off of the gravel road appears hazardous in the curb hazard map. (d) A phantom curb is detected at the road crest. (e) and (f) Phantom curb detections contort the road corridor, choking the drivable region.

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 an 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's 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.

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 a collision. We do not know why Caroline did not choose a path through the free space to Talos's 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

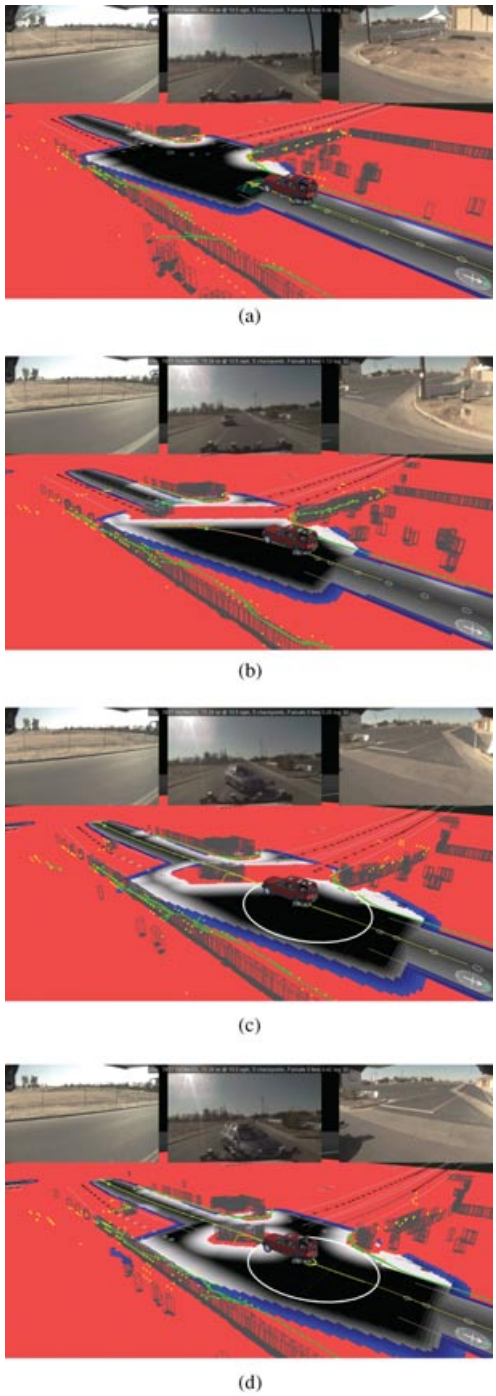


Figure 39. First Team CarOLO's Caroline–Talos near miss. (a) Talos stops upon entering intersection. (b) Talos detects the moving object across its path and begins to plan a path around it. (c) Talos begins an emergency stop. (d) Talos comes to a stop. Caroline no longer appears to be moving and instead is viewed as a static object.

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's 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's 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's 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 toward 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 that 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

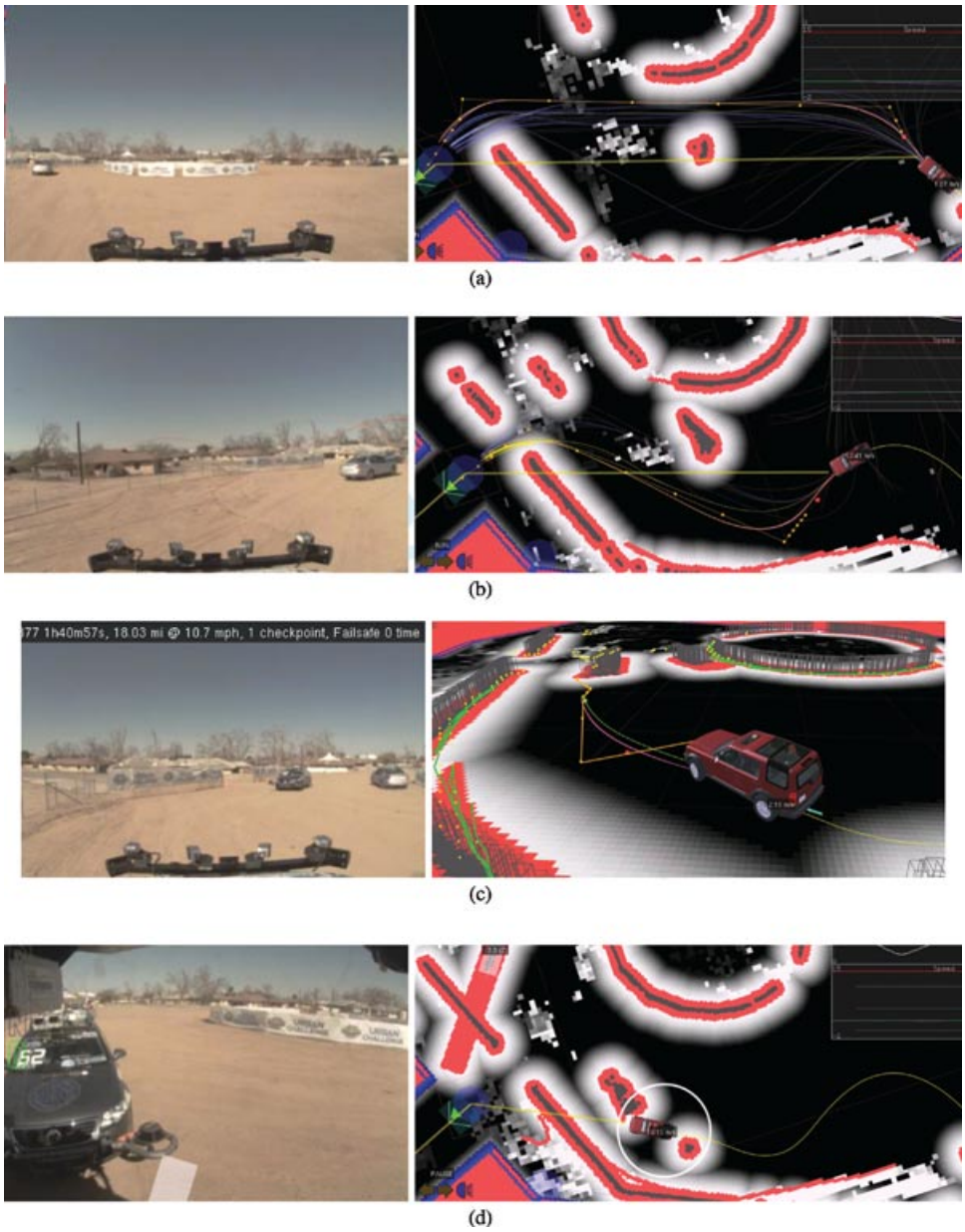


Figure 40. Second Caroline–Talos incident. (a) Talos drives around the Caroline chase vehicle. (b) Talos drives around Caroline, which is moving sufficiently slowly to appear as a static object. (c) Talos continues to replan around Caroline, which was perceived as an static object in a different location. (d) Talos continues to drive around Caroline and then initiates an emergency stop, but cannot stop in the space left and collides with Caroline.

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 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.

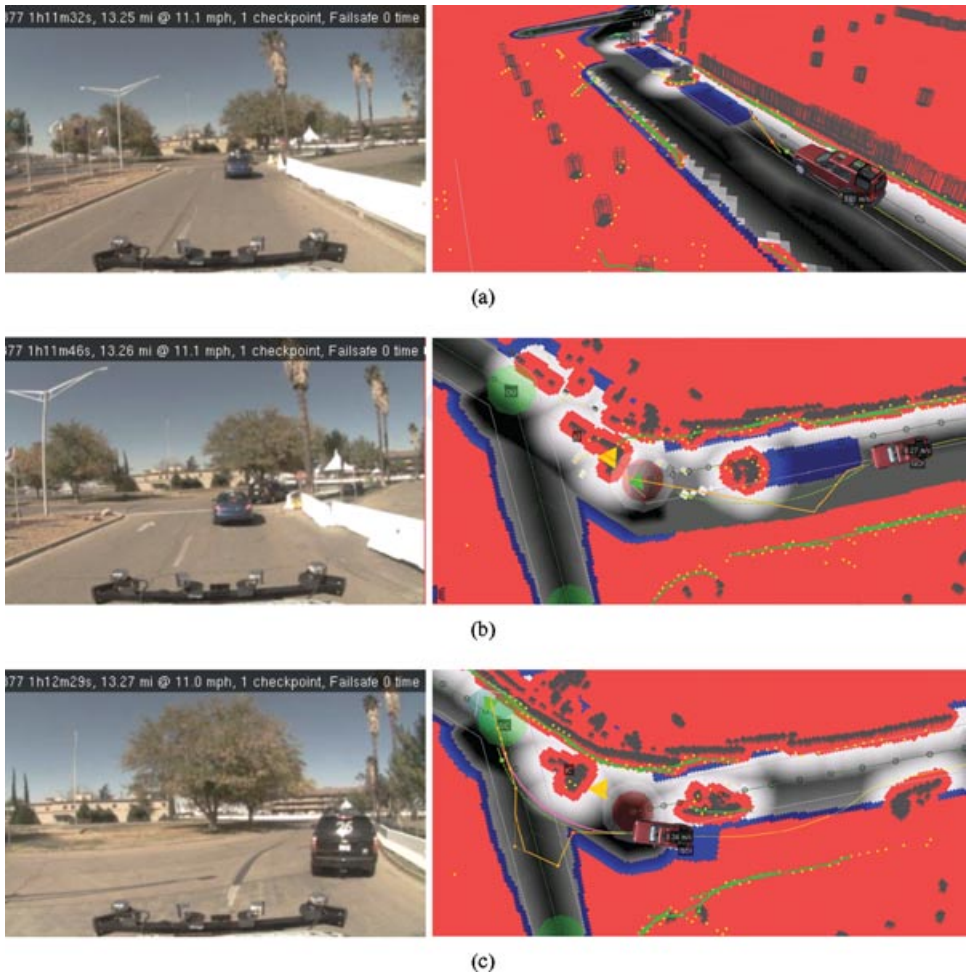


Figure 41. Lead-up to Skynet–Talos incident. (a) Talos initially queues behind the Skynet chase vehicle. (b) Lane position is low, so Talos finds a route around the chase vehicle. (c) Talos yields at the intersection. There are no moving vehicles, so it proceeds through.

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. Although this strategy worked well during the NQE, in the race, the collisions or near misses involving Talos often came about due to the

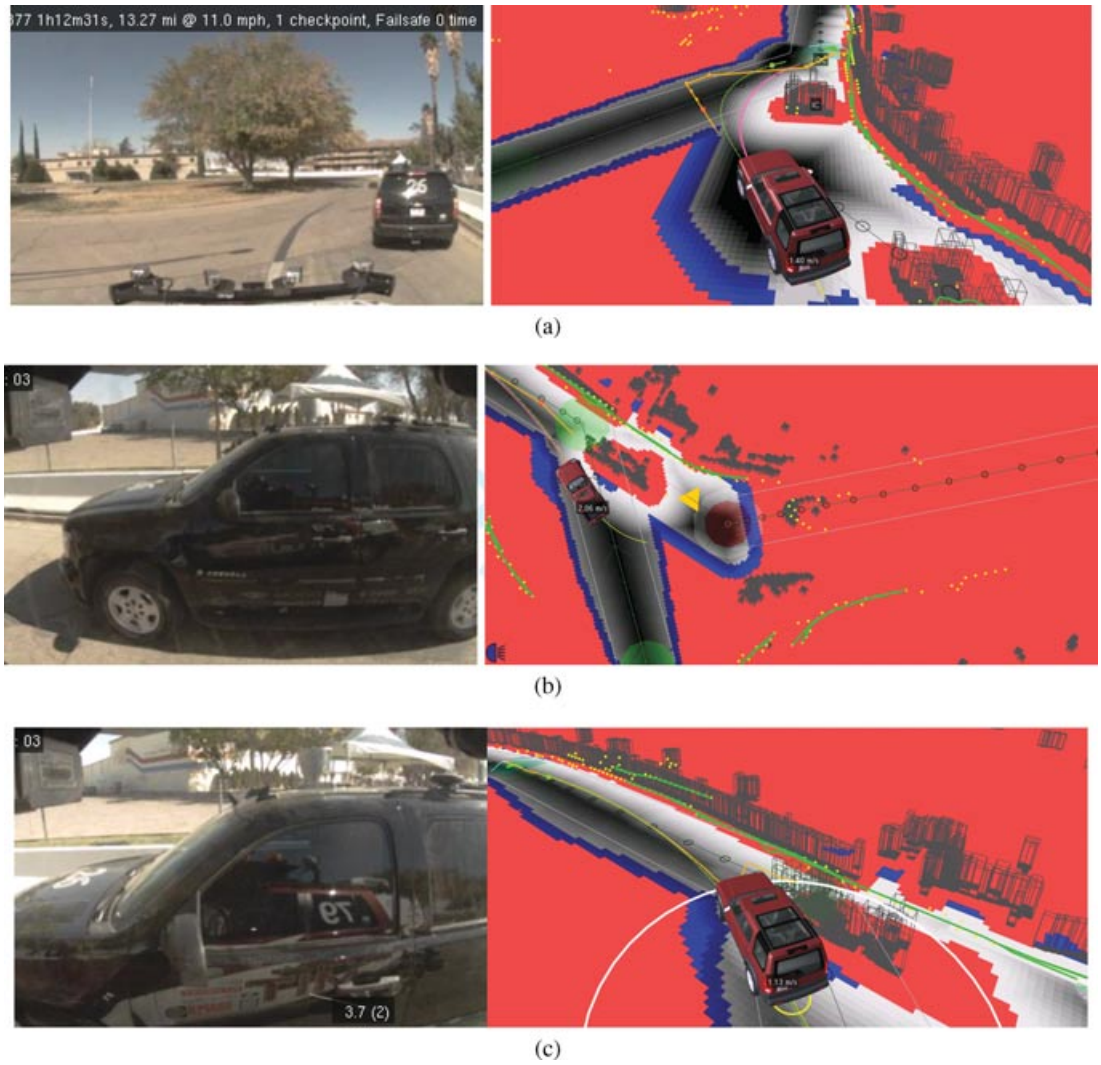


Figure 42. Skynet–Talos incident. (a) Talos plans a route around Skynet, which appears as a static object. (b) While Talos is passing, Skynet begins to accelerate. (c) While applying emergency braking, Talos turns into the accelerating Skynet.

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, is an avenue for future research.

7.3. Improved Simulation

Further investment in simulation tools for complex multirobot interactions is warranted. For this project, we developed a useful simulation for a *single* robotic vehicle in a complex environment (including traffic vehicles following predefined 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.

Whereas the vehicle generally operated in the vicinity of human-driven traffic without incident, problems were encountered when interacting with other autonomous vehicles at low speeds. 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 Fail-Safe Approaches

A key capability for long-term autonomous operation was the creation of comprehensive fail-safe modes. The judicious use of fail-safe 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 interest 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 kinodynamic 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 h. A key novel aspect of our system, in comparison to that of many other teams, is that autonomous decisions were made based on locally sensed perceptual data in preference to prespecified map data whenever 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 that 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 the 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 the 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

Bertozzi, M., & Broggi, A. (1998). Gold: A parallel real-time stereo vision system for generic obstacle and lane detection. IEEE Transactions on Image Processing, 7(1), 62–81.

Blom, H., & Bar-Shalom, Y. (1988). The interacting multiple model algorithm for systems with Markovian switching coefficients. IEEE Transactions on Automatic Control, 33(8), 780–783.

DARPA (2007). DARPA Urban Challenge rules. http://www.darpa.mil/GRANDCHALLENGE/rules.asp.

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), 381–385.

Frazzoli (2001). Robust hybrid control for autonomous vehicle motion planning. Ph.D. thesis, MIT, Cambridge, MA.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems, Science, and Cybernetics, 4(2), 100–107.

Hartley, R. I., & Zisserman, A. (2001). Multiple view geometry in computer vision. Cambridge, UK: Cambridge University Press.

Kelly, A., & Stentz, A. (1997). An approach to rough terrain autonomous mobility. In 1997 International Conference on Mobile Planetary Robots, Santa Monica, CA.

LaValle, S. M., & Kuffner, J. J. (2001). Randomized kinodynamic planning. International Journal of Robotics Research, 20(5), 378–400.

Mertz, C., Duggins, D., Gowdy, J., Kozar, J., MacLachlan, R., Steinfeld, A., Suppe, A., Thorpe, C., & Wang, C. (2005). Collision warning and sensor data processing in urbban areas. In Proceedings of the 5th International Conference on ITS Telecommunications, Brest, France (pp. 73–78).

Newman, P. M. (2003). MOOS—A mission oriented operating suite (Tech. Rep. OE2003–07). Cambridge, MA: MIT Department of Ocean Engineering.

Park, S., Deyst, J., & How, J. P. (2007). Performance and Lyapunov stability of a nonlinear path following guidance method. Journal of Guidance, Control, and Dynamics, 30(6), 1718–1728.

Rivest, R. L., & Leiserson, C. E. (1990). Introduction to algorithms. New York: McGraw-Hill.

Schouwenaars, T., How, J., & Feron, E. (2004). Receding horizon path planning with implicit safety guarantees. In Proceedings of the IEEE American Control Conference, Boston, MA (Vol. 6, pp. 5576–5581). IEEE.

Stanford Racing Team (2007). Stanford's robotic vehicle Junior: Interim report. http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Stanford.pdf.

Stein, G., Mano, O., & Shashua, A. (2000). A robust method for computing vehicle ego-motion. In Proceedings of the IEEE Intelligent Vehicles Symposium, Dearborn, MI (pp. 362–368).

Stein, G. P., Mano, O., & Shashua, A. (2003). Vision-based ACC with a single camera: Bounds on range and range rate accuracy. In Proceedings of the IEEE Intelligent Vehicles Symposium, Columbus, OH (pp. 120–125).

Tartan Racing (2007). Tartan Racing: A multi-modal approach to the DARPA Urban Challenge. http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Tartan_Racing.pdf.

Thorpe, C., Carlson, J., Duggins, D., Gowdy, J., MacLachlan, R., Mertz, C., Suppe, A., Wang, B., & Pittsburgh, P. (2005). Safe robot driving cluttered environments. In Eleventh International Symposium of Robotics Research, Siena, Italy (pp. 271–280). Springer.

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

Trepagnier, P., Nagel, J., Kinney, P., Koutsourgeras, C., & Dooner, M. (2006). KAT-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain. Journal of Field Robotics, 23, 509–526.

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

USDOT Federal Highway Administration, Office of Information Management (2005). Highway statistics 2005. Washington, DC: U.S. Government Printing Office.

Wang, C.-C. (2004). Simultaneous localization, mapping and moving object tracking. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.