MIT-CSAIL-TR-2007-058 December 14, 2007

Team MIT Urban Challenge Technical Report

John Leonard, David Barrett, Jonathan How, Seth Teller, Matt Antone, Stefan Campbell, Alex Epstein, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Troy Jones, Olivier Koch, Yoshiaki Kuwata, Keoni Mahelona, David Moore, Katy Moyer, Edwin Olson, Steven Peters, Chris Sanders, Justin Teo, and Matthew Walter

Team MIT Urban Challenge Technical Report

April 13, 2007

John Leonard[†], David Barrett[†], Jonathan How[†], Seth Teller[†], Matt Antone, Stefan Campbell, Alex Epstein, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Troy Jones, Olivier Koch, Yoshiaki Kuwata, Keoni Mahelona, David Moore, Katy Moyer, Edwin Olson, Steven Peters, Chris Sanders, Justin Teo, Matthew Walter

This technical report describes Team MIT's approach to the DARPA Urban Challenge. We have developed a novel strategy for using many inexpensive sensors, mounted on the vehicle periphery, and calibrated with a new cross-modal calibration technique. Lidar, camera, and radar data streams are processed using an innovative, locally smooth state representation that provides robust perception for real-time autonomous control. A resilient planning and control architecture has been developed for driving in traffic, comprised of an innovative combination of well-proven algorithms for mission planning, situational planning, situational interpretation, and trajectory control.

These innovations are being incorporated in two new robotic vehicles equipped for autonomous driving in urban environments, with extensive testing on a DARPA site visit course. Experimental results demonstrate all basic navigation and some basic traffic behaviors, including unoccupied autonomous driving, lane following using pure-pursuit control and our local frame perception strategy, obstacle avoidance using kino-dynamic RRT path planning, U-turns, and precedence evaluation amongst other cars at intersections using our situational interpreter. We are working to extend these approaches to advanced navigation and traffic scenarios.

[†] *Principal Investigator; others in alphabetical order*

Technical Report — DARPA Urban Challenge

Team MIT

April 13, 2007

Executive Summary

This technical report describes Team MIT's approach to the DARPA Urban Challenge. We have developed a novel strategy for using many inexpensive sensors, mounted on the vehicle periphery, and calibrated with a new cross-modal calibration technique. Lidar, camera, and radar data streams are processed using an innovative, locally smooth state representation that provides robust perception for real-time autonomous control. A resilient planning and control architecture has been developed for driving in traffic, comprised of an innovative combination of well-proven algorithms for mission planning, situational planning, situational interpretation, and trajectory control.

These innovations are being incorporated in two new robotic vehicles equipped for autonomous driving in urban environments, with extensive testing on a DARPA site visit course. Experimental results demonstrate all basic navigation and some basic traffic behaviors, including unoccupied autonomous driving, lane following using pure-pursuit control and our local frame perception strategy, obstacle avoidance using kino-dynamic RRT path planning, U-turns, and precedence evaluation amongst other cars at intersections using our situational interpreter. We are working to extend these approaches to advanced navigation and traffic scenarios.

Acknowledgments: Our team gratefully acknowledges the sponsorship of: MIT School of Engineering, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), MIT Department of Aeronautics and Astronautics, MIT Department of Electrical Engineering and Computer Science, MIT Department of Mechanical Engineering, The C. S. Draper Laboratory, Franklin W. Olin College of Engineering, The Ford-MIT Alliance, 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, Navcom, Delphi, Applanix, Drew Technologies, and Advanced Circuits.

I. INTRODUCTION AND OVERVIEW

Since May 2006, Team MIT has worked to develop a vehicle capable of safe, autonomous driving in an urban (pedestrian-free) environment according to the stated goals of the DARPA Grand Challenge (DGC). We have focused on identifying the key challenges facing our effort, and addressing these challenges sufficiently early in the eighteen months of development to maximize the likelihood of success during the June, October, and November events.

Our team is comprised primarily of faculty, staff, graduate students and undergraduates from the MIT School of Engineering, combined with one faculty member and a team of six undergraduates from the Franklin W. Olin College of Engineering and staff from the C. S. Draper Laboratory, BAE Systems, MIT Lincoln Laboratory, and Australia National University. Our team is organized into four subteams: (1) Vehicle Engineering & Safety; (2) Planning & Control; (3) Perception; and (4) Software/Infrastructure/Testing. Many team members have substantial roles on more than one subteam.

A. Overview of Our Solution

Achieving an autonomous urban driving capability is a difficult multi-dimensional problem. A key element of the difficulty is that significant uncertainty occurs at multiple levels: in the environment, in sensing, and in actuation. Any successful strategy for meeting this challenge must address all of these sources of uncertainty. Moreover, it must do so in a way that is scalable to spatially extended environments, efficient enough for real-time implementation on a rapidly moving vehicle, and robust enough for extended operation.

The difficulties in developing a solution that can rise to these intellectual challenges are compounded by the many unknowns for the system design process. Despite DARPA's best efforts to define the rules for the Urban Challenge in detail far in advance of the race, clearly there is significant potential variation in the difficulty of the final event. It was difficult at the start of the project (and remains difficult now) to conduct a single analysis of the system that can be translated to one static set of system requirements (for example, to predict how different sensor suites will perform in actual race conditions). For this reason, our team chose to follow a spiral design strategy, developing a flexible vehicle design and creating a system architecture that can 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 by successful participants in the 2005 Grand Challenge [1], [2], [3]. As newcomers to the Grand Challenge, it was imperative for our team to bring up 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 have gained experience from continuous testing with our prototype vehicle, the lessons learned have been incorporated into the overall architecture of our final race vehicle.

Our 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 *Talos I*, a Ford Escape, at the outset of the project, to permit early autonomous testing with a minimal sensor suite. Over time we increased the frequency of tests, added more sensors, and brought more software capabilities online to meet a growing set of requirements according to our timeline. In parallel with this, we have procured and fabricated our race vehicle *Talos II*, a Land Rover LR3. Our modular and flexible architecture allows rapid transition from one vehicle to the other, as our system evolves to its final race-ready state.

From an algorithmic perspective, our system employs innovative approaches for a variety of aspects of perception, planning, and control. One of our key innovations for perception, described

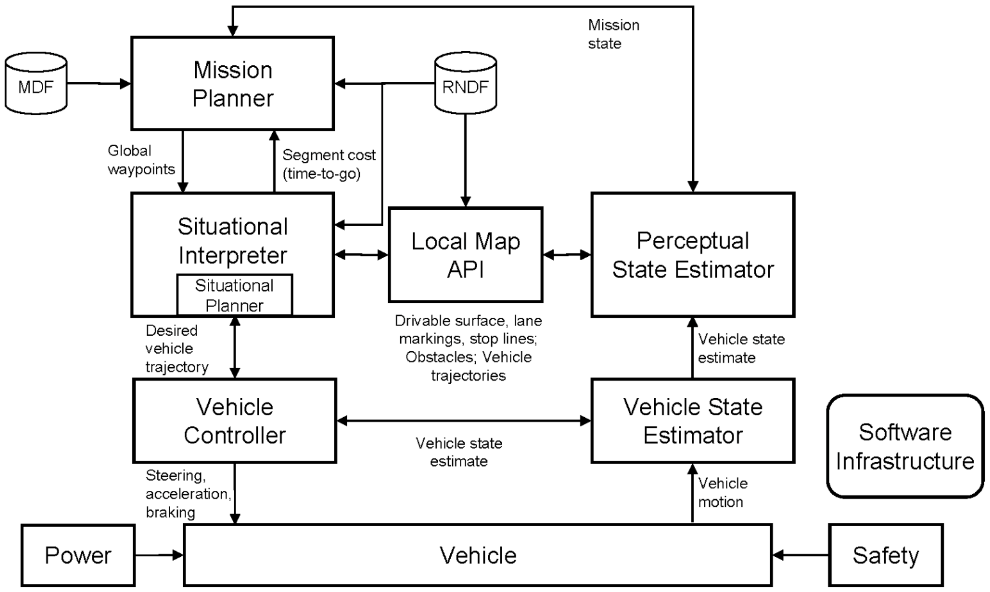


Fig. 1. System Architecture for Team MIT

in Section II-E.3, is the use of a locally smooth state representation for nearly all perceptual processing, providing improved robustness to GPS outages. Our planning and control system, described in Section II-F, incorporates novel integration of kino-dynamic motion planning, based on RRTs, with situational interpretation and high-level mission planning.

B. Architecture

Our overall system architecture (Figure 1) includes the following subsystems:

- The **Mission Planner** 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 an ordered list of RNDF waypoints that are provided to the *Situational Interpreter*. In designing this subsystem, our goal has been to create a resilient planning architecture that ensures that the autonomous vehicle can respond reasonably (and make progress) under unexpected conditions that may occur on the challenge course.
- The **Perceptual State Estimator** comprises algorithms that, using lidar, radar, vision and navigation sensors, detect and track cars and other obstacles, delineate the drivable road surface, detect and track lane markings and stop lines, and estimate the vehicle's pose.
- The **Local Map API** provides an efficient interface to perceptual data, answering queries from the *Situational Interpreter* and *Situational Planner* about the validity of potential motion paths with respect to detected obstacles and lane markings.
- The **Situational Interpreter** uses the mission plan and the situational awareness embedded in the *Local Map API* to determine the mode state of the vehicle and the environment. This information is used to determine what goal points should be considered by the *Situational Planner* and what sets of rules, constraints, and performance/robustness weights should be applied. The Situational Interpreter provides inputs to the *Mission Planner* about any inferred road blockages or traffic delays, and controls transitions amongst different system operating modes.
- The **Situational Planner** identifies and optimizes a kino-dynamically feasible vehicle trajectory that moves towards the RNDF waypoint selected by the *Mission Planner* and *Situational Interpreter* using the constraints given by the *Situational Interpreter* and the situational awareness embedded in the *Local Map API*. Uncertainty in local situational awareness is handled through rapid replanning and constraint tightening. The *Situational*

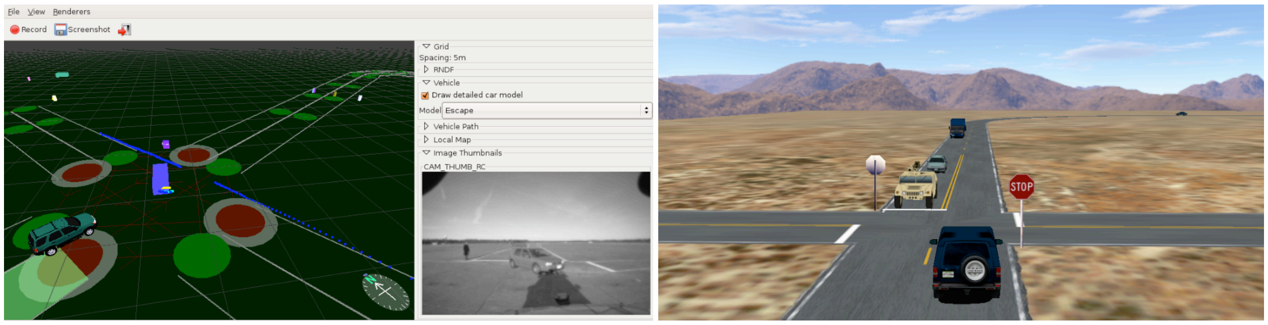


Fig. 2. (a) Screenshot of the real-time visualization tool running "live" for an intersection testing scenario, showing RNDF and vehicle navigation information (white, green, red), lidar (blue, yellow) and camera (lower right) data, and vehicle tracker output (blue solid in intersection). (b) Screenshot of SimCreator simulation environment, including MIT vehicle and traffic vehicles.

Planner also accounts explicitly for vehicle safety, even with moving obstacles. The output is a desired vehicle trajectory, specified as an ordered list of waypoints (each with position, velocity, and heading) that are provided to the *Vehicle Controller*.

- The **Vehicle Controller** uses the inputs from the *Perceptual State Estimator* to execute the low-level control necessary to track the desired paths and velocity profiles issued by the *Situational Planner*.
- **The Safety Module** monitors sensor data, overriding vehicle control as necessary to avoid collisions. This module addresses safety pervasively through its interactions with vehicle hardware, firmware, and software, and careful definition of system operating modes.

To support these modules, we have developed a powerful and flexible software architecture based on a new lightweight UDP message passing system (described in Section II-D). Our system 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. The codebase incorporates sophisticated capabilities, including data logging, replay, and 3-D visualization of experimental data (Figure 2(a)). Our development environment is fully coupled to two different simulation environments, one developed in-house based on simplified vehicle models, and one commercial package (SimCreator) that incorporates detailed car dynamics and other cars to simulate traffic scenarios (Figure 2(b)).

II. ANALYSIS AND DESIGN

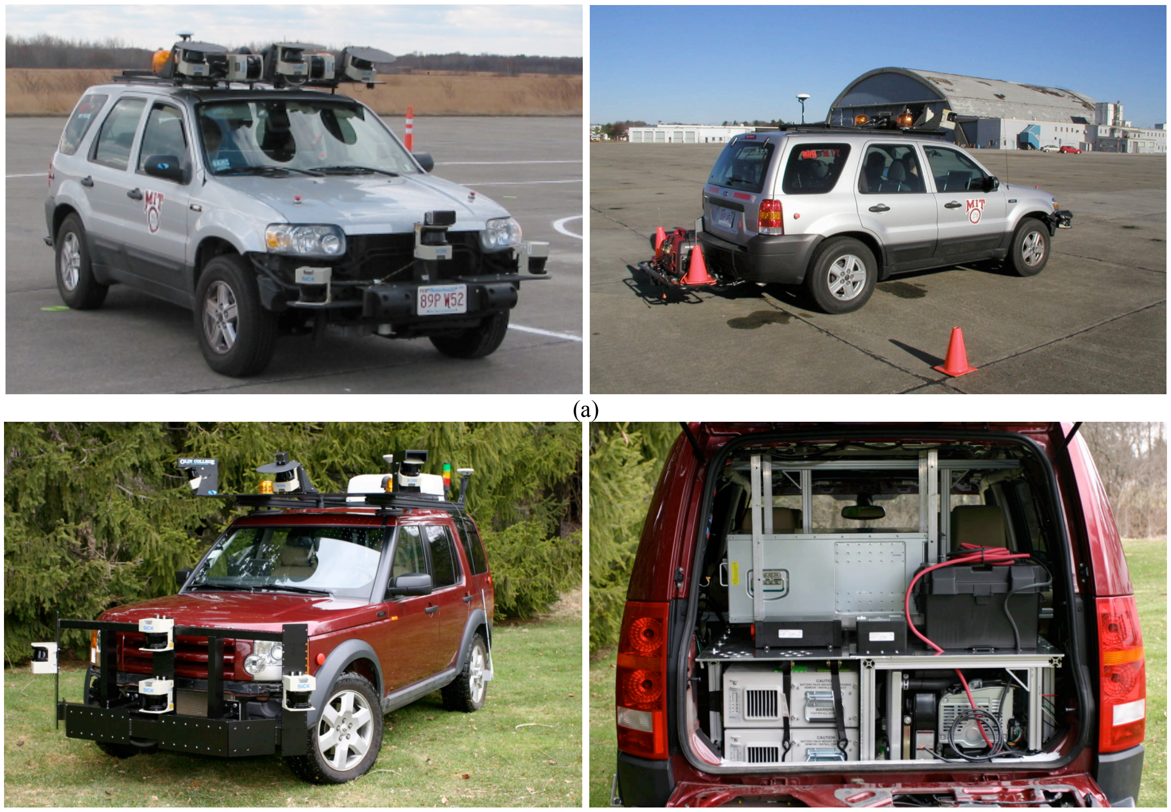
This section describes our analysis of the problem, the conclusions we drew in the form of design principles for our system and its various subsystems, and their implications for our actual design.

A. Design Considerations

In designing our system, we employed several principles.

Many inexpensive sensors. We chose to use a large number of relatively cheap, unactuated sensors, rather than a smaller 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, connector or firmware malfunction, or software failure. Eschewing actuation also simplified the system's mechanical, electrical and software architecture.
- Since each of our many sensors can be positioned at an extreme point on the car, more of the car's field of view (FOV) can be observed. A single sensor, by contrast, would have to be placed at a single position, and would have a more limited FOV due to unavoidable occlusion by the vehicle itself. Deploying many single sensors also gives us increased


(a)

(b)

Fig. 3. Team MIT's two autonomous vehicles. (a) *Talos I*, a Ford Escape. (b) *Talos II*, a Land Rover LR3.

flexibility as designers. For example, we have arranged the sensors so that most points in the car's surround are observed by at least one exteroceptive sensor of each type. Finally, our multi-sensor strategy also permits more effective distribution of I/O and CPU bandwidth across multiple processors.

Minimal reliance on GPS. We observed from our own prior research efforts, 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 our design choice is our "local frame" situational awareness strategy, described more fully below in Section II-E.3.

Fine-grained, high-bandwidth CPU, I/O and network resources. Given the short time available for system development, our main goal was simply to get a first pass at every needed method 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, load balancing, etc. This led us to the conclusion that we should have many CPUs, and that we should lightly load each machine's CPU and physical memory (say, at half capacity) to avoid non-linear systems effects such as process or memory thrashing. A similar consideration led us to use a fast network interconnect, to avoid operating regimes with significant network contention. The downside of our "CPU-rich" strategy was a high power budget, which required an external generator on the car. This has added mechanical and electrical complexity to the system, but we feel that the computational flexibility that we have gained has justified this effort.

Asynchronous sensor publish and update; minimal sensor fusion. Our vehicle has many distinct sensors of six different types (odometry, inertial, GPS, lidar, radar, vision), each type generating data at a different rate. Our architecture dedicates a software driver to each individual sensor. Each driver performs minimal processing, then publishes the sensor data on a shared network. Our local map API (described below) performs minimal sensor fusion, simply by depositing interpreted sensor returns into the map on an as-sensed (i.e. just in time) basis. Finally, the situational interpreter uses a synchronous query mechanism to evaluate proposed motion paths based upon the most recent information in the map.

Bullet-proof low-level control. To ensure that the vehicle is 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 is critical, and it is important that the situational planner and situational interpreter can always predict the state of the controller/vehicle with high confidence.

Simulation, testing, and visualization. While field testing is paramount, it is time consuming and not always possible. Thus we have developed multiple simulation environments that interface directly with the vehicle code to perform extensive testing of the software and algorithms prior to on-site testing. Finally, we make extensive use of algorithm visualization and inspection tools.

B. Vehicle Engineering

1) Key Design Decisions: The key decisions in designing our vehicle related to size, power and computation. For tasks such as parking and execution of U-turns, our team wanted a small vehicle with a tight turning radius. Given the difficulty of the urban driving task, and our desire to use many inexpensive sensors, our team chose to use a relatively powerful computer system. This led us beyond the power requirements of aftermarket alternator solutions for our class of vehicles, mandating the use of an external generator for power.

Our choice to use two different types of cars (a Ford Escape and a Land Rover LR3) has entailed some risk, but given our time and budgetary constraints, this development path has provided significant benefits to our team. The spiral design approach has enabled us to move quickly up the learning curve, testing software to accomplish many of our milestone 2 site visit requirements well before completion of the mechanical fabrication of our race vehicle.

To mitigate some of the complexity inherent in using two different vehicles, we have developed a modular design and made the vehicles very similar. Our modifications to the vehicles are minimally invasive, enabling a change in vehicles with minimal effort should our primary vehicle become damaged. Table I compares the main elements of the design of each vehicle. Shared elements between the vehicles include: (a) similar power systems; (b) identical computer systems; (c) common models of cameras, lidars, and radars; (d) identical sensor mounting hardware; (e) common software infrastructure; and (f) identical safety systems. The two most important distinctions between the two vehicles are: (a) *Talos I* presently uses an external generator, whereas Talos II uses an internally mounted recreational vehicle generator, and (b) *Talos I* uses a Navcom GPS combined with an XSens inertial measurement unit, whereas *Talos II* has an Applanix POSLV 220 GPS/Inertial navigation system. As described in more detail below in Section II-E, our perception system design incorporates many inexpensive lidars, radars, and cameras. In its final race configuration, *Talos-II* will have as many as twelve SICK LMS291 lidars, twelve Delphi ACC3 automotive radars, and ten Point Gray Firefly MV cameras. We are also evaluating more capable (and costlier) lidars for possible future use.

TABLE I

SUMMARY OF VEHICLE COMPONENTS FOR PROTOTYPE AND RACE VEHICLES, *Talos I* AND *Talos II*.

	Talos-I	*Talos-II*
Type of car	Ford Escape	Land Rover LR3
Fly-by-wire conversion	AEVIT/EMC	AEVIT/EMC
IMU/GPS System	XSens /Navcom	Applanix POS LV220
Computer system	Fujitsu-Siemens BX600 Blade Cluster	Fujitsu-Siemens BX600 Blade Cluster
Power generation	Honda EU2000i/Honda EU3000is	Honda EVD6010 internal RV generator
Power conditioning	APC UPS	Dual Acumentrics 2500 220V Ruggedized UPS
Lidars	SICK LMS-291 S05 lidars	SICK LMS 291-S05 lidars
Cameras	Pt. Grey Firefly MV	Pt. Grey Firefly MV
Radars	Delphi ACC3	Delphi ACC3

2) Prototype Vehicle Design: Our prototype vehicle *Talos I*, a Ford Escape, was acquired in July, 2006 and has undergone extensive testing since delivery. A key early decision was to obtain an AEVIT fly-by-wire conversion of each of our vehicles from Electronic Mobility Controls in Baton Rouge, LA. The EMC conversion performed well on the Gray Team's entry KAT-5 in the 2005 race [3]. EMC's detailed prior knowledge of the Escape expedited the conversion process, enabling us to receive delivery of *Talos I* in August, 2006. After installation of an interim computer system and an initial sensor suite, our first autonomous operations were performed in October, 2006. Since that time, the sensor suite and computational capabilities of *Talos I* have evolved over time, in accordance with our spiral design approach.

A major step in the evolution of *Talos I* was the installation of its computer system, which became available in January, 2007. Quanta Computer, Inc. donated two Fujitsu-Siemens Primergy BX600 blade cluster computers, one for each vehicle. Each cluster can have up to ten BX620 blade computers, each with a quad-core processor (four 2.3GHz, 64-bit CPUs).

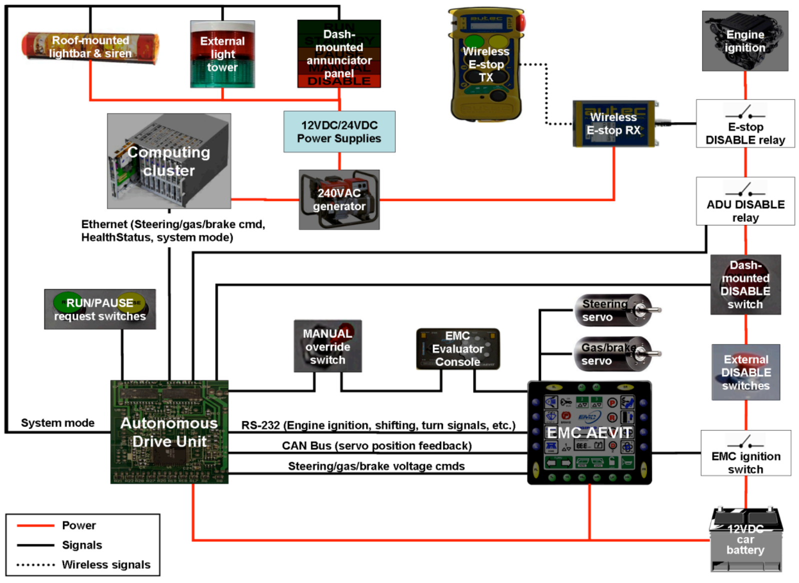
3) Race Vehicle Design: For our race vehicle, *Talos II*, we chose a maneuverable and robust platform, the Land Rover LR3, which has been provided on loan to us for this project from the Ford Motor Corporation. We chose this vehicle for its excellent maneuverability and small turning radius, large payload capacity and unmatched all-terrain driving capabilities in comparison to other vehicles in its class. At present, the vehicle design is complete, all major subsystems are mounted in the vehicle, and all wiring and system integration will be completed by the end of April, 2007.

C. Safety

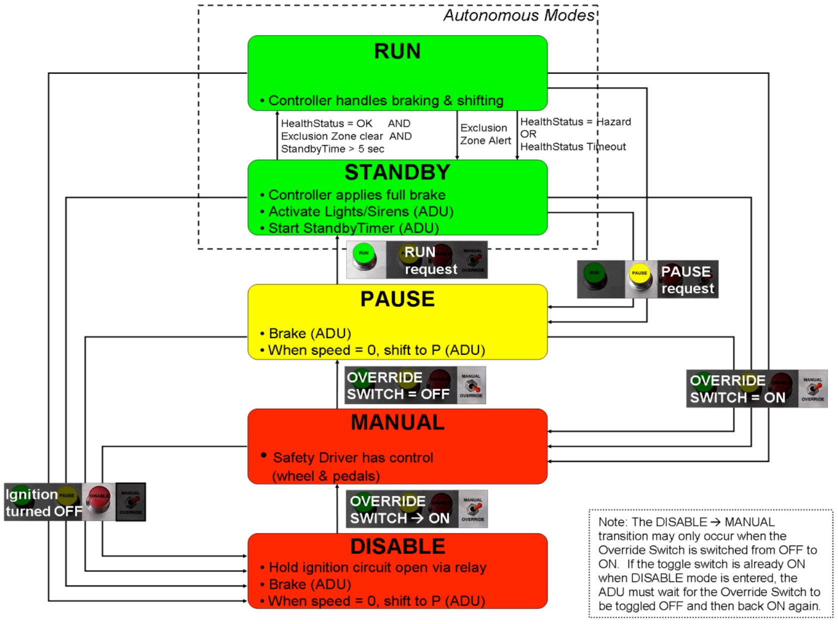
Figure 4(a) presents the components of the safety system of our vehicles. Our design uses a layered safety architecture with four layers: (1) software: perception, planning and trajectory control modules; (2) firmware: Autonomous Drive Unit (ADU) and EMC drive-by-wire system; (3) electrical: E-stop buttons and remote E-stop; and (4) mechanical: vehicle bumpers, restraints for human occupants, firewall separating the generator from the passenger compartment, etc.

The use of a well-defined set of vehicle modes, with clearly described mode transitions, is absolutely vital for safety. Figure 4(b) shows the five vehicle operating modes and all possible transitions among them.

In the DISABLE mode, the vehicle ignition has been disabled and the ADU stops the vehicle in place via brake application and shifting into park. The vehicle can be disabled from any other mode by activating the dash-mounted, remote E-stop, or any of the external kill switches. The only valid mode transition out of DISABLE is to MANUAL, which occurs when the manual override switch is switched from OFF to ON. While in DISABLE mode, the ADU holds the


Roof-mounted lightbar & siren
External light tower
Dash-mounted annunciator panel
DISABLE
Wireless E-stop TX
Engine ignition
Wireless E-stop RX
E-stop DISABLE relay
Computing cluster
12VDC/24VDC Power Supplies
240VAC generator
ADU DISABLE relay
Ethernet (Steering/gas/brake cmd, HealthStatus, system mode)
Dash-mounted DISABLE switch
RUN/PAUSE request switches
MANUAL override switch
EMC Evaluator Console
Steering servo
Gas/brake servo
External DISABLE switches
System mode
Autonomous Drive Unit
RS-232 (Engine ignition, shifting, turn signals, etc.)
CAN Bus (servo position feedback)
Steering/gas/brake voltage cmds
EMC AEVIT
EMC ignition switch
12VDC car battery
Power
Signals
Wireless signals

(a)


Autonomous Modes
RUN
• Controller handles braking & shifting
HealthStatus = OK AND
Exclusion Zone clear AND
StandbyTime > 5 sec
Exclusion Zone Alert
HealthStatus = Hazard OR HealthStatus Timeout
STANDBY
• Controller applies full brake
• Activate Lights/Sirens (ADU)
• Start StandbyTimer (ADU)
RUN request
PAUSE request
PAUSE
• Brake (ADU)
• When speed = 0, shift to P (ADU)
OVERRIDE SWITCH = OFF
OVERRIDE SWITCH = ON
MANUAL
• Safety Driver has control (wheel & pedals)
Ignition turned OFF
OVERRIDE SWITCH → ON
DISABLE
• Hold ignition circuit open via relay
• Brake (ADU)
• When speed = 0, shift to P (ADU)
Note: The DISABLE → MANUAL transition may only occur when the Override Switch is switched from OFF to ON. If the toggle switch is already ON when DISABLE mode is entered, the ADU must wait for the Override Switch to be toggled OFF and then back ON again.

(b)

Fig.4. (a) Safety system components.The main components include:1) ADU, which is a microprocessor responsible for tracking the operating mode of the vehicle and for commanding steering, gas, and brake via analog outputs; 2) EMC drive-by-wire system, which provides a low-level interface to the vehicle steering, gas/brake actuation, and engine ignition; 3) External kill switches; 4) Dash-mounted switch panel; 5) Remote E-stop; 6) Vehicle mode annunciators; 7) Light tower and lightbar. (b) Vehicle operating modes.

vehicle ignition circuit open such that the vehicle can not be restarted until transitioning into MANUAL mode.

In MANUAL mode, the safety driver has turned the manual override switch to ON and, in turn, the EMC has relinquished control of its servomotors and the safety driver has full control of the vehicle. The ADU is no longer breaking the ignition circuit, so the safety driver may start the vehicle and drive it. Flipping the manual override switch ON from any other mode will transition to MANUAL mode. Once in manual mode, switching the override switch to OFF will always put the vehicle into PAUSE mode.

In PAUSE mode, the ADU commands the EMC servomotors to hold the brake and shift the vehicle into park. This creates a safe mode for temporarily stopping the vehicle during a mission or for the safety driver to egress and ingress the vehicle before and after unmanned operation. PAUSE mode is entered by flipping off the manual override switch when in MANUAL mode, or by pressing the dash-mounted or remote E-stop PAUSE request button while in RUN or STANDBY mode.

In STANDBY mode, the safety driver requests the vehicle to go fully autonomous (RUN mode) via the dash-mounted buttons or via the remote E-stop. The vehicle first goes into STANDBY mode. During STANDBY mode, the vehicle is held in place via ADU-commanded braking while the roof-mounted light bar and audible alerts are activated. After five seconds, if the ADU is receiving good system health information from the blade computing cluster, it will automatically transition to RUN mode. If any critical health and status faults are reported to the ADU by the blade cluster, the ADU switches the vehicle into STANDBY mode until the faults are cleared.

RUN mode is reachable only by first transitioning through STANDBY mode. In RUN mode, the vehicle becomes fully autonomous, perceiving its environment, awaiting a mission plan, planning the mission, and executing the planned mission. Vehicle control commands are generated by the software on the blade cluster and passed to the EMC servo motors via the ADU.

D. Software Infrastructure

Although a powerful computing infrastructure affords us greater freedom in designing our software and algorithms, it also threatens us with the overhead of managing a highly complex software system with intricate interdependencies. The design of our software infrastructure has been driven largely by the desire for simplicity, modularity, and robustness.

Each blade in our compute cluster is configured with an identical operating system and software image, with a few exceptions such as computer name and IP address. Management scripts allow us to quickly propagate code changes from one computer to the rest, maintaining a coherent software state. Combined with the fact that most of our sensors are network accessible and not bound to a specific computer, this gives us the ability to update and reconfigure our system with little effort.

Software modules are made as independent as possible. Each module is an independent process, and different modules can run on different CPUs. Most modules communicate via UDP multicast over Ethernet, which has low packet overhead, low latency, and scales well.

A carefully-defined message-passing protocol allows certain modules to serve as drop-in replacements for other modules without requiring configuration or code changes in any other module. For example, we are able to test different mission planning techniques by simply swapping out the mission planner, while leaving the other modules untouched. Similarly, we are able to test in simulation by replacing only the perception and actuation modules. We have tested a variety of GPS units by changing only the receiver hardware and corresponding driver software.

Every module is designed to satisfy two conditions: it can be killed, then restarted, and will resume normal function; and it can tolerate a brief absence of the message stream expected from any other module. There are a few critical modules, such as the motion controller and pose estimation modules, for which the system can not tolerate indefinitely long downtime.

Separating modules into individual processes also allows us to choose the best programming language for a particular task. Latency-sensitive modules where speed is of the essence, such as the low-level controller, are written in C. Modules that do not have stringent timing demands, such as the log file playback and analysis modules, are written in Java or Python. Modules such as the obstacle detector have also been rapidly prototyped and debugged in Java or Python for evaluation before being ported to C when needed. By taking this approach, we gain the flexibility and development speed associated with high-level languages, while retaining a simple and straightforward migration path for performance-sensitive modules.

1) Message Passing: We have developed a Lightweight Communications (LC) system and a Lightweight Communications Marshaller (LCM) to address limitations of existing message-passing systems. Our system uses a publish-subscribe idiom, making both module-to-module communication and broadcast announcements possible.

We rejected existing methods as not well-suited to our application. IPC (used by Carmen), MOOS, and other systems are based on TCP interconnect. This guarantees delivery of all messages, but if a message is dropped, future messages are delayed until the lost message is detected and retransmitted. After evaluating these existing systems (which have enormous code bases), we determined that it would be infeasible to modify them appropriately for our use.

Our system uses UDP multicast for its transmission medium, providing low-latency communications. Dropped packets do not result in future communication delays, as would be the case in a TCP system.

Data packing and unpacking (also called marshalling) is a notoriously error-prone process. A number of existing marshalling schemes exist; the IPC toolkit includes one scheme, while XDR has a long history in applications like RPC and NFS. These systems, however, do not support multiple languages well, and do not provide the type-checking and run-time semantic checking that we desired in order to detect errors. Consequently, we developed our own data marshalling system, LCM. Like XDR, it employs a formal data format specification language which is then compiled into data marshalling code for our various platforms and languages. LCM permits us to easily modify message types, while ensuring that modules are compiled with compatible marshalling code.

LC and LCM are serving us very well; they are already finding use in other research projects, and we plan on eventual public distribution.

2) Simulation Environments: To ensure the robustness of our planning and control algorithms, we make extensive use of simulation, using two different simulation tools: (1) an in-house vehicle simulator that generates simulated GPS, IMU, and odometry data using a no-slip kinematic model; and (2) a commercial vehicle modeling package, SimCreator, which uses more detailed car dynamics but also includes other static/dynamic traffic vehicles. Using the equivalent of the local map API, the control and planners are tested with interactive scenarios such as intersection and passing with oncoming traffic. This tool also enables deterministic and stochastic simulations with scripted planning scenarios for numerous test cases. Each of these simulation modules receives the same gas/steer voltage commands as the actual car, and publishes the same message types used by the on-board sensors, so that all other modules can operate exactly the same way in simulation and in the actual car.

E. Perception

The Perception module maintains the vehicle's awareness of its location with respect to the active RNDF/MDF, and its immediate surround (Figure 5): drivable (smooth) surface; fixed hazards (such as curbs, buildings, trees, etc.); and other stopped or moving vehicles. Some aspects of our system (e.g., the navigation filter) are standard. The perception module's novel aspects include a "local frame" representation used to reduce sensor uncertainty, and a "local map API" that enables us to avoid building a data-heavy terrain map of the vehicle's surround. Both innovations are described below.

1) Sensor Selection and Positioning: The perception module incorporates the principle that many cheap sensors are preferable to a few, expensive sensors. We situated (i.e., positioned and

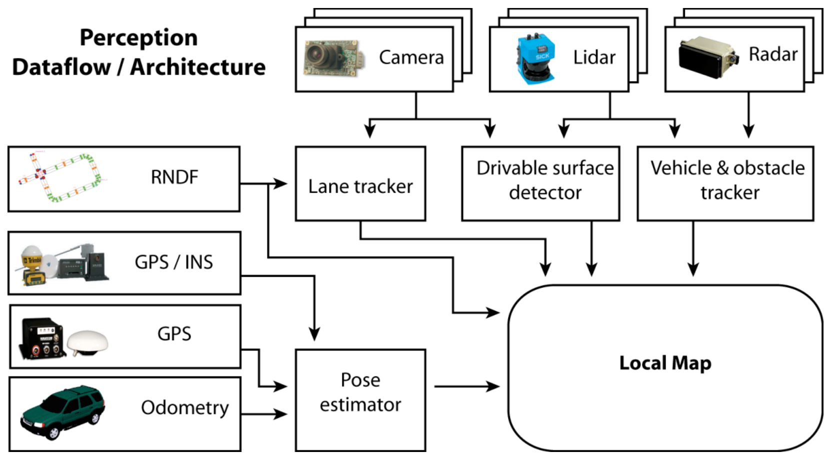


Fig. 5. Dataflow within the perception subsystem. Data from the sensors and RNDF are accumulated into the local map.

oriented) the sensors so that most points in the vehicle's surround would be observed by at least one sensor of each type: lidar, radar, vision. This redundant coverage gives us robustness against both type-specific sensor failure (due for example to unfavorable sun angle or obstacle viewing angle) and individual sensor failure (due for example to mechanical insult or wiring damage).

We selected the sensors themselves with several specific tasks in mind. For close-range obstacle detection, we use "skirt" (horizontal) lidars at a variety of heights, and automotive radar for redundancy. For drivable surface detection, we use "pushbroom" (downward-canted) lidar. For road profile and vehicle pitch estimation, we use vertical lidar. For lane and stop-line detection, we use high-rate forward video, with a planned additional rear video channel for higher confidence lane detection.

To check for safe passage across an active lane when initiating an intersection traversal or a lane-change maneuver, we must determine that the lane to be traversed is vehicle-free at greater than lidar range (about 55 meters under optimal conditions). For this task we use automotive radar, which gives us reliable vehicle detection to at least 120-meter range. Critically, the radars provide a relatively large vertical field-of-view, permitting robust vehicle tracking even when our vehicle pitches.

2) Vehicle State Estimation: We use a hybrid particle filter and Extended Kalman filter with odometry, inertial, and GPS inputs to estimate the absolute 6-DOF pose of the vehicle in a globally-registered coordinate frame. Since GPS is unreliable in many situations (due to poor satellite geometry, urban canyons, foliage obscuration, jamming, etc.), and since GPS is used in the DGC only for waypoint specification, we depend on it only for gross decision-making, such as determining which RNDF segment the vehicle is traversing, and occasional updates of the global-to-local coordinate transform. For fine-grained tasks such as lane following and obstacle avoidance, we use a "local frame" method described in the next section.

3) Local Frame: A key innovation in our approach is the use of a local frame for sensor fusion and trajectory planning. Unlike the global coordinate frame, the local frame is always consistent with vehicle motion over short time scales. GPS data has no effect on the vehicle's position in the local frame; instead, GPS data is used to determine the relationship between the local and global coordinate frame. Since the integrated vehicle motion results in accumulated error over time, the relationship between the local and global frame varies slowly over time.

The vehicle's current position is always known precisely in the local frame: it can be viewed as a vehicle-centric coordinate frame. The map of obstacles, lane markings, and drivable road surface is maintained in the local frame (see Figure 6). Any abrupt change in the GPS receiver output (due e.g. to multipath or a GPS outage) has no impact on the local frame, and thus has no

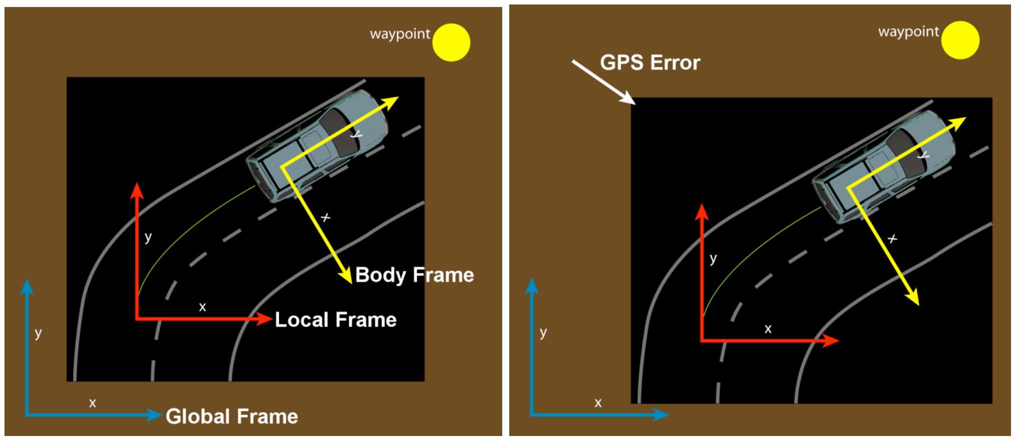


Fig. 6. Our three coordinate frames: The global frame (registered to GPS); the local frame (registered to sensor data); and the vehicle body frame. (a) Normal operation when GPS quality is high. (b) A GPS error corrupts the transformation between the local and global frame. However, perceived environment elements (e.g lane markings, obstacles) remain properly registered to the vehicle, allowing it to continue driving normally.

adverse effect on sensor fusion or trajectory planning. The perception module combines sensor data into a "semi-persistent" map effectively maintained only within the vehicle's sensor range.

Our local frame approach has been validated by our development efforts to date: the milestones described in this report all made use of separate local and global coordinate systems. In fact, the bulk of our testing to-date has been performed with an $80 consumer-grade GPS receiver, demonstrating our resilience to GPS data of limited quality.

4) LIDAR: The vehicle has a number of lidar sensors, positioned and oriented so as to accomplish three separate tasks: obstacle detection and tracking; road surface drivability classification; and road profile estimation.

The *skirt* (horizontal) lidars are mounted at several distinct heights on the front and rear bumpers, providing low-latency (75Hz) detection of any sufficiently tall or low-hanging obstacles in the vehicle's path. Under ideal conditions, our SICK lidars produce object returns to a range of about 55 meters. Under more typical conditions, we have observed a detection range of about 30 meters. For two vehicles approaching each other at the maximum speed (under DGC rules) of 30 mph or about 15 m/s, our detection range gives us about one second of warning, sufficient time for evasive maneuvering and/or emergency braking. (Longer-leadtime detection is provided by automotive radars, discussed below.)

The obstacle detection and tracking system comprises a ring of overlapping LIDAR sensors affording 360 degree coverage. Off-groundplane objects within range, whether stationary or moving, are tracked. This produces an object trajectory from which the tracking system can extrapolate future locations and the uncertainty associated with those predictions. Object detections and trajectories are passed to the situational interpreter and planner. We can reliably detect and track static obstacles out to the limit of our sensor range; moving obstacles (vehicles) are readily tracked at shorter ranges (under 20m). We are aggressively working to extend the functional range of our dynamic obstacle detection. Distinguishing motionless cars from stationary obstacles is also a difficult problem. In addition to algorithmic solutions, we are considering additional sensors.

The *pushbroom* lidars sweep the road surface forward of the vehicle at a variety of ranges (we use 5, 10, and 15 meters at present). Another noteworthy innovation of the perception module is that it does not build an explicit elevation terrain mesh: doing so would be vulnerable to hallucinated obstacles (e.g., humps) when the vehicle's state estimate is noisy. Instead, our implementation produces a grid-based "hazard map" using range discontinuities extracted from

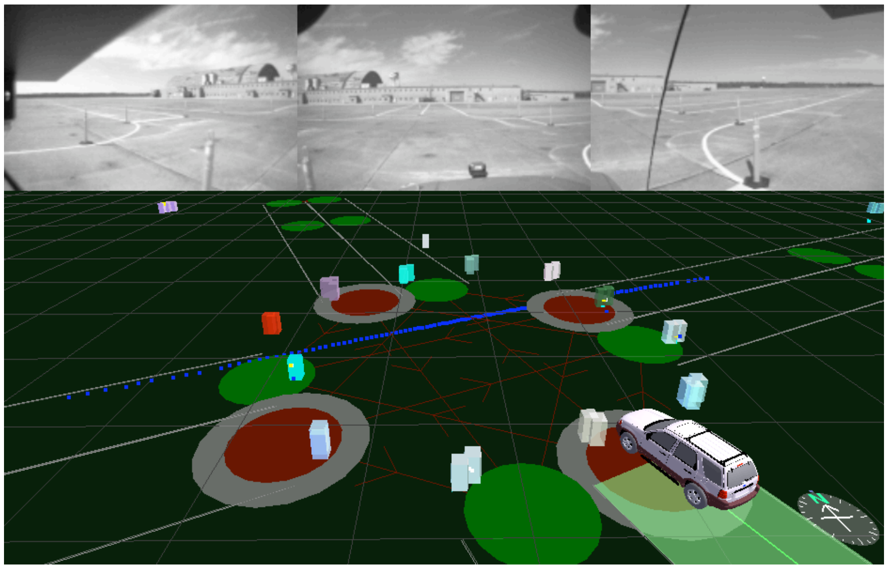


Fig. 7. Co-registered video from three cameras mounted on the roof of *Talos I*.

individual scans. Provided that we can register scans across short time and spatial scales, the deposition of hazards into a common map provides an effective "sensor fusion" capability.

Finally, the *vertical* lidars observe the road surface profile just in front of the vehicle; they provide an independent estimate of vehicle pitch with respect to the road surface (rather than with respect to gravity, as does the IMU) and provide a range return from the vehicle ahead of us. This also provides an estimate of the road's vertical curvature, useful to our vision processing module.

5) Radar: We use a number of Delphi automotive radars (several at each corner of the vehicle, aligned horizontally) for long-range vehicle detection. Long-range detection is critical in two situations: at intersections, when the vehicle must determine if a lane to be traversed is free of oncoming traffic; and when the vehicle must determine whether it is safe to initiate a passing maneuver. Since our vehicle stops at intersections, and the top speed of any vehicle under DGC rules is 30 mph (about 15 m/s), five seconds to traverse an intersection corresponds to about 75 m of vehicle-free area. This is easily achievable with these radars (for this task, the radars are mounted so as to look directly to the left and right). The task is harder at speed when our vehicle must briefly occupy an oncoming lane and the closing speed of two vehicles is 60 mph or about 30 m/s. In this case, we must observe 150 meters of vehicle-free area to execute a .five-second passing maneuver. This range is approximately at the limit of existing automotive radars.

6) Vision: Our vision system consists of a number of high-rate (60 Hz), monochrome, wide field of view cameras. Three such cameras are mounted (with overlap to produce a *tiled* field of view) on the forward portion of the vehicle to perform lane and stopline detection. We have determined that the long-range camera described in our proposal (to check for oncoming traffic) is not necessary, since we can perform this task with radar. Our camera calibration (see below) is sufficiently accurate to support tiled lane perception (see Figure 7).

Detection of road markings such as lane markers and stop lines is a major new technical requirement of the Urban Challenge. Our vision system uses a matched filter tuned to detect lane marking fragments, which are then clustered subject to continuity constraints. This approach handles uncertainty well, and draws upon our team's expertise in data clustering [4]. We are currently developing a novel lane tracker targeted to high-curvature lanes, to complement the output of a commercial lane tracking system from Mobileye, which is designed for highway driving and hence is optimized for low-curvature roads and lane markings.

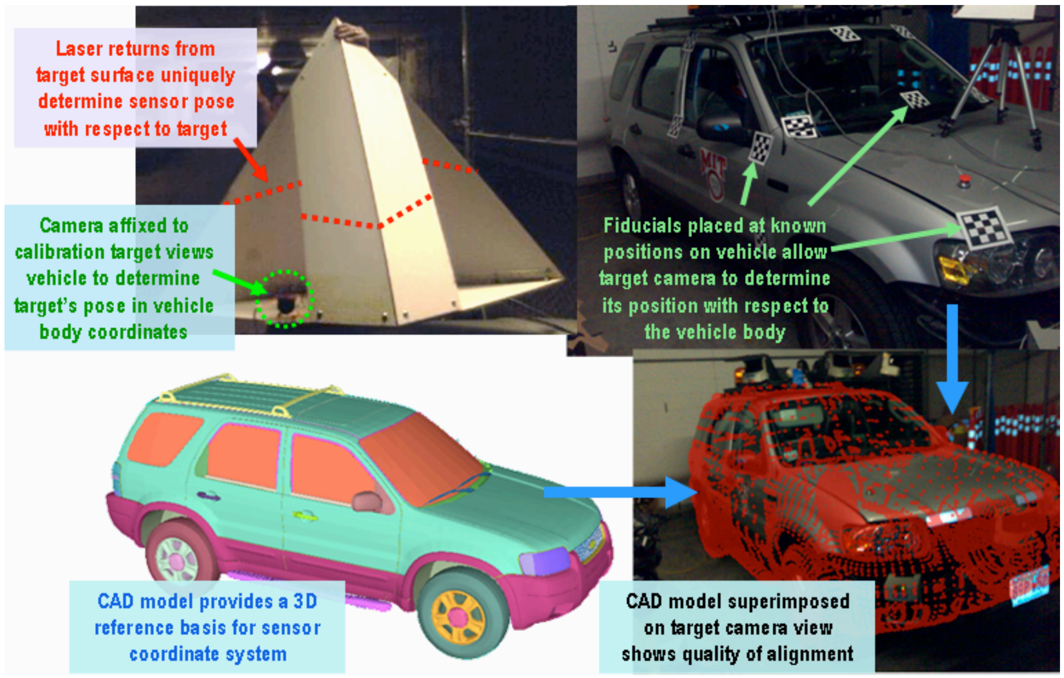

Laser returns from target surface uniquely determine sensor pose with respect to target
Camera affixed to calibration target views vehicle to determine target's pose in vehicle body coordinates
Fiducials placed at known positions on vehicle allow target camera to determine its position with respect to the vehicle body
CAD model provides a 3D reference basis for sensor coordinate system
CAD model superimposed on target camera view shows quality of alignment

Fig. 8. Components and algorithms for rapid field calibration of multiple sensors of several types.

7) Multi-Sensor Calibration: We have developed a rapid exterior calibration method for multiple sensors of three distinct types: video, lidar, and radar. We have dozens of sensors attached to the vehicle. A key issue is how to recover the location and orientation of each sensor in the vehicle's body frame. This issue is especially important because we are still experimenting with various sensor configurations, under complex and not completely understood constraints. For example, we must position and orient the lidars in order to scan the road surface at various front ranges, but must also prevent these bulky devices and their protective hoods from occluding the FOVs of other sensors. Without a fast, accurate exterior calibration method, we would get locked in early to a suboptimal placement. With such a method, we are free to experiment with many placements. A separate design goal is that, in the (hopefully highly unlikely) event of a minor collision during competition, we will have a rapid field calibration method that can be executed in at most five minutes.

Our prototype calibration method involves a 3D reference object that can be detected and localized by all three sensor types (Figure 8). The object, a kind of rigid, open pyramid, also has a calibrated camera attached which can look back at fiducial points on the vehicle in order to determine its own pose. Thus we have a set of observables from each body sensor, often from pairs or triples of sensors simultaneously, along with a carefully validated estimate of the reference object's freespace pose. From this we can recover and sanity-check all required body-frame sensor poses.

8) Local Map API: The local map combines prior information (such as RNDF waypoints and stoplines) and sensor-derived data (including lane markings, drivable surface delineations, and obstacle locations) into a comprehensive representation of the car's surround. The local map is kept current through asynchronous updates by various sensor handlers. To avoid publishing the data-intensive local map over the network, the local map API includes an efficient, synchronous query mechanism that classifies proposed motion trajectories according to the map contents, tagging each trajectory appropriately if it will encounter an obstacle or cross a lane boundary.

9) Vehicle Tracking: One critical component of the perception module is robust tracking of vehicles over time, so that the situational interpreter can be aware of: which obstacles the car believes to be vehicles; whether those vehicles have moved recently; and if so what their motion trajectories have been. We use a three-level hierarchy based on individual skirt lidar returns to accomplish vehicle tracking. During each time slice, we collect all raw lidar returns and cluster them into sub-meter-sized *chunks* of nearby returns. This data-reduction step preserves the sensitivity afforded by many lidars, but spares us the cost of maintaining redundant spatial information. Typically the method reduces approximately 7000 lidar returns (per 100 ms integration window) to a few hundred chunks.

Chunks that occupy approximately the same physical space at approximately the same time are almost always part of the same physical object. By exploiting this spatio-temporal adjacency assumption, we can form *groups* of chunks. For each group, we record the chunks (over space and time) that were assigned to it. From a group's chunks, we can compute a trajectory for the group. We use a simple holonomic motion model and least-squares fit to extrapolate the group's position and velocity. This trajectory estimate is used to initialize a vehicle tracking hypothesis.

Vehicle tracking hypotheses are constantly updated given new observations. In many cases, the data association is performed using the spatio-temporal grouping. When this fails (due to very fast or hard-to-see objects), group-to-group data associations are added to the hypothesis list. We then consider the active hypotheses and look for the subset of them that best explains the data, finally making this hypothesis subset visible to other local map modules.

Our vehicle tracker also records historical information about tracked vehicles. Currently, this is limited to whether the vehicle has moved in the past: this is useful in distinguishing a parked car from a car waiting at an intersection. In the future, we may attempt to infer more information: for example, whether the car might be preparing to turn (as indicated by deceleration while nearing an intersection at which no stop is required).

10) Ground Truth Labeling Tool: We have developed a tool with which a human user can efficiently annotate hundreds of video frames with *ground truth* (human-judged) indications of drivable road surface, road hazards, lane markings, stop lines, and static and moving obstacles. The user also provides an additional "depth below or height above road surface" attribute for hazards and obstacles. The resulting annotations, although derived (for human convenience) from raw video, include implicit metrical range and bearing information (since the camera is calibrated, and a ground plane estimate is available), so can be used to validate the algorithmic outputs of all three sensor types: video, lidar and radar. We will make the ground-truth video labeling tool and labeled datasets available to the robotics and machine vision communities after the competition.

F. Planning and Control

As shown in Figure 1, planning and control are performed at three levels: mission, situational, and vehicle. Strategic questions such as "what sequence of segments should be followed in the RNDF?" are handled by the mission planner. More detailed issues, such as "what is the best path through this intersection?", are handled by the situational planner. Finally, the lowest level controller is designed to ensure that the vehicle precisely follows the trajectories computed by the situational planner. Our particular focus is on *resilient planning*, ensuring that the vehicle makes progress even in the presence of large uncertainties about the environment. This is exemplified in the way that the communication between the planning and control blocks/algorithms in the architecture is two-way, and if it encounters an infeasible problem or situation, each lower block can request that the blocks above modify the current plan by revising their problem statements and recomputing the result. Furthermore, the Mission Planner uses exponentially forgetting edge costs. This enables the planner to re-explore previously rejected plans; thus the vehicle can continue to explore the environment as its situational awareness is updated from fresh sensor data.

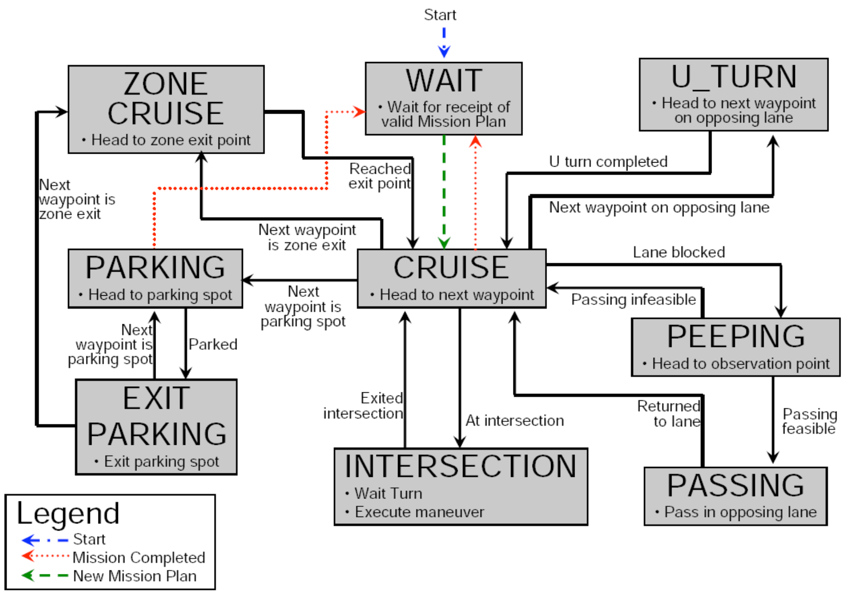


Fig. 9. Situational Interpreter State Transition Diagram. All modes are sub-modes of the system RUN mode (Fig 4(b)).

1) Mission Planner: The Mission Planner module works in the global frame, generating the RNDF-based route plan that minimizes the expected mission completion time. Every planning and control test starts by reading the MDF and RNDF and running the A* algorithm in the Mission Planner. Each route segment is weighted based on *a priori* information such as intersections, turns, and speed limits, but to account for environmental uncertainty, these weights are updated as the mission proceeds. Variability in mission time due to traffic is handled readily using robust real-time optimization techniques [5].

2) Situational Interpreter: The Situational Interpreter module keeps track of the current vehicle status and, when running autonomously, decides the next course of action. Its inputs are (i) a mission plan from the Mission Planner; (ii) current local map state through the Local Map API; and (iii) vehicle state from the Vehicle State Estimator (VSE).

The primary function of the Situational Interpreter is to handle the mode switching for the planner, so its decisions are primarily logical; thus it is implemented as a Finite State Machine (FSM). The top level of the state transition diagram is shown in Figure 9. All major modes in Figure 9 (e.g., `WAIT` and `CRUISE`) are sub-modes of the system RUN mode (c.f. Figure 4(b)). Each major mode has several sub-modes to handle different operating aspects of the major mode.

Other functions of the Situational Interpreter are to modify the behavior of the Situational Planner according to the applicable traffic rules and constraints to guide the trajectory generation process towards achieving the mission objectives. The Situational Interpreter also converts the global waypoints of the mission plan into the local frame coordinates using transformations provided by the VSE. These are eventually passed on as inputs to the Situational Planner.

Referring to Figure 9, when invoked to start (corresponding to the transition from the STANDBY mode to the RUN mode), the Situational Interpreter first determines the location of the vehicle within the route network based on the RNDF and vehicle location estimate from VSE. Once determined, this RNDF location information is sent to the Mission Planner and the Situational Interpreter enters the `WAIT` mode.

On receipt of a valid mission plan, the Situational Interpreter switches to a default run mode of `CRUISE`, which includes basic functions such as lane following and passing on multi-lane roads with the same direction of travel. With the next checkpoint well-defined, the Situational Interpreter then estimates the time to reach it. The RNDF location information and expected time to the next checkpoint are constantly updated and sent to the Mission Planner. The RNDF location anchors the start point for new mission plans while the estimated time to next checkpoint provides information for Mission Planner to update the edge costs used for mission planning. In addition, when a route to the next checkpoint is determined to be blocked or impassable, the Situational Interpreter updates the Mission Planner with this information and triggers the Mission Planner to re-plan.

The first step in `CRUISE` is to determine if the vehicle should switch to any other run mode. This decision is based on the estimated vehicle location in the route network and the first waypoint of the mission plan. For example, if the first mission waypoint is in the opposing lane, the Situational Interpreter will switch to `U-TURN` mode. Also, when approaching a vehicle/obstacle in the lane of traffic from behind, it is likely that the available perceptual information will be insufficient to determine if it is safe or legal to pass. Thus, after stopping, a `PEEPING` mode is used to position our vehicle in a better vantage point for observing past the obstacle. If from this observation position, the Situational Planner finds a safe trajectory that returns to the lane of travel, then the Situational Interpreter infers that it is legal and safe to perform the passing maneuver. In such a situation, the Situational Interpreter will switch to `PASSING` mode, modifying the planning problem (active constraints) in the Situational Planner to enable passing in the opposite lane. If passing is neither feasible nor legal, the Situational Interpreter reports this information to the Mission Planner to trigger a re-plan. While waiting for a new mission plan, the Situational Interpreter will switch back to `CRUISE` mode and will respond accordingly if the vehicle/obstacle that was blocking the lane eventually moves.

The `INTERSECTION` mode is used to enforce precedence at intersections and to execute appropriate maneuvers to get to the next entry waypoint. Within a parking zone, the Situational Interpreter will switch to either `PARKING`, `EXIT PARKING`, or `ZONE CRUISE` modes depending on the next mission waypoint. As the names suggest, `PARKING` is used to enter a parking spot, while `EXIT PARKING` is used to back out of it. `ZONE CRUISE` is the mode used to head towards a parking zone exit point. The Situational Interpreter makes no distinction between cruising and passing in the parking zone.

Safety is handled by the Situational Interpreter within each of the major modes. The role of the Situational Planner is to find feasible trajectories that clear all known obstacles and threats. The Situational Interpreter sends these trajectories to the controller for execution, and constantly monitors the threats as reflected in the local map. If, for example, an errant vehicle strays into the trajectory currently being executed, the Situational Interpreter first requests the Situational Planner to generate a short, possibly sub-optimal but safe trajectory. If any such trajectory is found, it will be executed and normal execution resumes. If no such trajectories are found, the Situational Interpreter relaxes all *artificial* constraints (e.g., lane boundary constraints), and requests the Situational Planner to redo the search for a safe trajectory. If a trajectory is found, it will be executed and the Situational Interpreter will switch to a `RECOVERY` sub-mode until all relaxed *artificial* constraints can be re-enforced, upon which normal execution resumes. If no safe trajectories can be found after relaxing all artificial constraints, the Situational Interpreter enters an `ESTOP` sub-mode, which corresponds to the switching from system mode RUN to mode STANDBY. Recovery from an `ESTOP` will require invocation from the system-level module, through transitions from STANDBY to RUN, as shown in Figure 4(b).

3) Situational Planner: The Situational Planner module generates trajectories that are guaranteed to be collision-free, dynamically feasible, and consistent with the applicable traffic rules. The inputs from the Situational Interpreter include the target point, the speed limit, the standoff distances, the allowable drive direction(s), the enforcement/relaxation lane/zone boundary constraints, the enforcement/relaxation road divider constraint, and the biased path center. The output of the Situational Planner is a list of waypoint and the speed limit used by the controller.

Our motion planner is a modification of a sampling-based method known as Rapidly-exploring Random Tree (RRT) [6]. In this method, a tree of feasible trajectories is incrementally constructed, starting from the current configuration of the car. This process is driven by a biased random sampling strategy, which allows the tree to efficiently explore the region reachable by the vehicle. This approach has numerous advantages: (1) it has provable probabilistic completeness in a static environment – if a feasible path exists, the probability of finding it approaches 1 as the number of generated samples increases; (2) it can easily handle complex (e.g., nonlinear, non feedback-linearizable) vehicle dynamics; (3) the algorithm provides and maintains a large number of possible routes, allowing for efficient replanning in an uncertain environment, in case new obstacles and/or vehicles are detected; and (4) RRTs work just as well in open spaces such as parking lots as they do in cluttered environments.

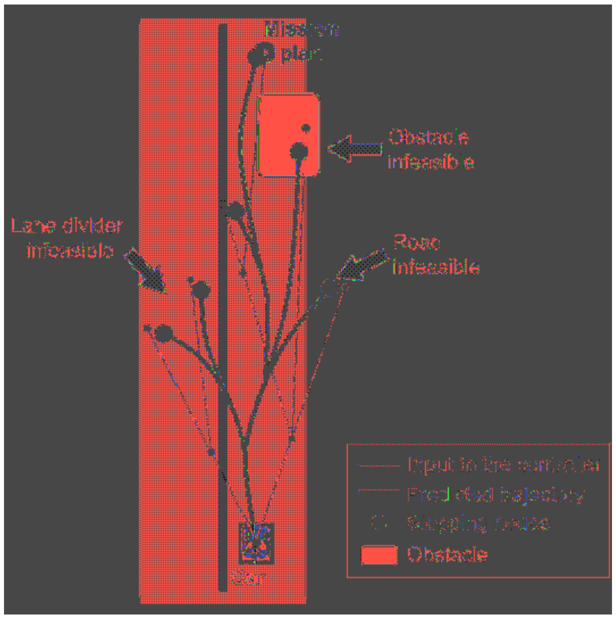


Fig. 10. RRT-based motion planning approach.

In particular, in our implementation, the trajectory generation process is based on a closed-loop model of the car (using the low-level controller) that is designed to track simple paths [7]. The basic step to grow the tree of feasible trajectories is as follows: (i) sample a reference trajectory for the car, e.g., a point and a direction identifying a straight line; (ii) simulate the closed-loop behavior of the car when such reference is fed to the low-level controller; (iii) check the resulting trajectory for feasibility, with respect to obstacles, and to the traffic rules; (iv) if the trajectory is feasible, add it to the tree; (v) iterate. Using the closed-loop model of the car ensures that the trajectories are not only dynamically feasible for the vehicle, but also can be easily followed by the low-level vehicle controller. This use of the closed-loop model also significantly reduces the decision space and enables real-time RRT-based motion planning for vehicles with complex and unstable open-loop dynamics, as is the case for a car.

The Situational Planner uses the local map API in step (iii), which provides a simple interface to access the very complex perceived environment. The query answer includes the feasibility of the trajectory, a confidence level on the perceived information, and the structure of the urban environments that can be used to bias the sampling for more efficient trajectory generation. The local map API also predicts the future trajectory of moving objects, and the moving obstacles are handled in the same way as static obstacles.

The modified RRT approach is made robust to uncertainty in the dynamic environment in several ways. The frequent replanning rate of 10 Hz ensures the Situational Planner can quickly react to any sensed changes in the environment. Uncertainty in the local map is handled by systematically tightening the constraints of the predicted states at future plan times [8], [9]. This margin retained in the constraints is then made available to the optimization as time progresses. As a further measure against this uncertainty, all leaves of the tree are vehicle stopping nodes (see

Figure 10). As a result, if no further tree expansion occurs in an excessively constrained scenario, the vehicle will come safely to a complete stop.

One challenging aspect of this problem is that the environment is dynamic (other cars are moving), so stopping nodes are not safe indefinitely. The modified RRT introduces a time buffer τ to check for potential future collisions at the stopping nodes, so that the vehicle will stop only at a place where it can safely wait for the new motion plan. This ensures that the algorithm has always at least τ seconds to compute the next motion plan, and this property is maintained over the course of the mission. The τ value reflects the sensing radius and speed of other cars, and is currently set at several seconds. Combined with the probabilistic completeness of RRT, the safe stopping node enables the Situational Planner to always find a dynamically feasible trajectory to make mission progress in highly constrained situations.

4) Vehicle Control: The low-level vehicle controller generates the gas, brake, and throttle commands required to track the trajectory specified by the situational planner. In designing this control system, speed and steering control were treated separately. Moreover, by ensuring that the vehicle comes to a complete stop before entering reverse, the steering control problem was subdivided into separate forward and reverse controllers.

The speed is controlled using a linear PID controller with a first order filter incorporated into the derivative term.

$$u_c = K_p(v - v_{ref}) + K_d s\left(\frac{1}{1+Ts}v - v_{ref}\right) + \frac{K_i}{s}(v - v_{ref})$$

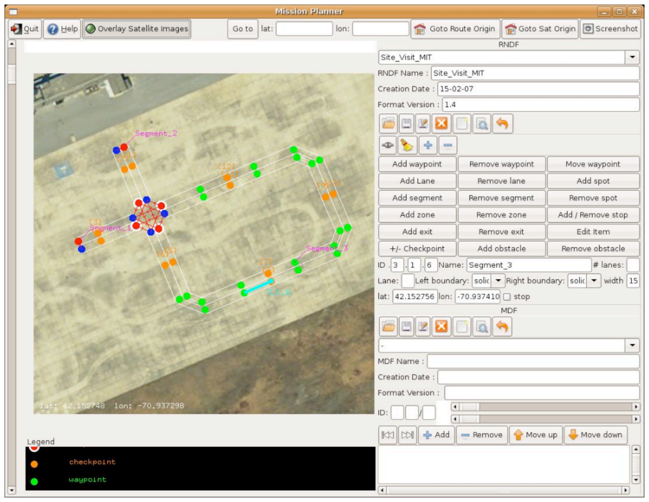
The gains for this controller were tuned to produce minimal overshoot for step inputs using a simple linear model of an internal combustion engine. For braking, these gains were scaled by a proportional term K_b to account for dynamic differences between braking and accelerating.

The EMC actuation system, accepting an input signal of 0-5V, places a single servo-mechanism on both the gas and brake pedal. Using the EMC system specifications, models of acceleration/braking, and the above described compensator, a piecewise-linear mapping from u_c to the voltage V_{app} was developed. This mapping accounts for the deadband in actuation when the controller switches from gas (acceleration) to braking (deceleration), the saturation limits in each of the two pedals, and the different gains/actions associated with each of the pedals. This setup enables us to use one dynamic compensator for the low-level controller.

In implementing this simple control strategy, several issues arose with the use of an integral term. Specifically, it proved necessary to incorporate anti-windup logic that would help keep the integral term from growing under actuator saturation. This logic had the added feature that it significantly improved system overshoot when tracking speed step commands. Experimentation then revealed that resetting the integral term for deceleration maneuvers helped improve performance and the rapidity of the vehicle's response. With these features, stopping maneuvers are executed sufficiently accurately without having to add position feedback to the compensator.

Forward steering control is achieved using a variant of the relatively mature pure-pursuit algorithm. The pure-pursuit algorithm has demonstrated excellent tracking performance for both ground and aerial vehicles over many years of implementation as a result of an inherent anticipation of the path [10]. This controller can robustly stabilize the system even when a large cross-track error is present, and is well suited to tracking non-smooth piece-wise linear paths, simplifying trajectory planning. In our implementation, a circle of radius L_a is centered at the center of mass of the car and the intersection of this circle with the desired path defines the pure-pursuit goal point. The pure-pursuit algorithm then specifies a constant curvature path for the vehicle to travel from the current location to the goal point. A simple vehicle model [11] can then be used to specify the desired steering wheel angle

Testing Area ~ 44 Acres



Site_Visit_MIT

RNDF Name : Site_Visit_MIT

Creation Date : 15-02-07

Format Version : 1.4

ID . 3 . 1 . 6 Name: Segment_3 # lanes:

Lane: Left boundary: solic Right boundary: solic width 15

lat: 42.152756 lon: -70.937410 ☐ stop

MDF Name :

Creation Date :

Format Version :

ID:

lat: 42.152748 lon: -70.937298

(a) (b)

Fig. 11. (a) Overhead view of test site. (b) Screenshot of a powerful RNDF/MDF authoring tool created for the DGC project.

$$\ddot{a} = \left(L_c + \frac{K_{us} v^2}{g} \right) \hat{e}$$

where g is the acceleration due to gravity, L_c is the wheel-base of the car, and K_{us} is the common understeer coefficient, which was determined empirically. The δ command is then converted to a voltage input to the EMC. As illustrated in [12], the pure-pursuit algorithm can be linearized as a PD controller on cross-track error. This linearization then provides insight into selecting the radius L_a of the virtual circle as the gains of the linearized controller are functions of the vehicle's speed and the pure-pursuit radius. As a result, the implementation of this control strategy requires that L_a be selected as a function of the vehicle speed to ensure that the effective bandwidth of the closed-loop system is approximately constant. However, one must select a minimum L_a value (i.e., $L_{a\,min}$) to ensure that the effective controller gains do not grow significantly at low speeds. Good results were achieved with an $L_{a\,min}$ value on the order of 3 to 5 meters. For intermediate speeds, L_a was taken as Kv, where the constant K was determined from a root locus analysis of a linearized version of the pure pursuit algorithm [12], a second order model of the steering actuator dynamics, and a single track model of the vehicle dynamics. Pure pursuit was investigated for driving in reverse, but slightly better tracking was achieved with an alternative controller [13].

III. RESULTS AND PERFORMANCE

Securing a safe and affordable test site in the Boston area was one of the most difficult challenges so far. Our team is grateful to the South Shore Tri-Town Development Corporation for providing extensive access to the former South Weymouth Naval Air Station, which has served as our primary test site. The team chose not to deploy to a warm weather test site, a considerable gamble given typical New England winter weather. In general, we were fortunate to have good weather; this paid off for our team as we were able to frequently perform autonomous vehicle operations. The SSTTDC site is ideal for testing, and will serve as the host of our Milestone2 Site Visit Demonstration in June of 2007. Figure 11 shows an overhead view of the test site, as well as a screenshot of a powerful RNDF/MDF authoring tool that we created for the project.

A. Test Procedures

In the short timeline of the project, our team adopted a strategy of testing early and often. To organize this effort we produced a program schedule which was based on achieving sets of

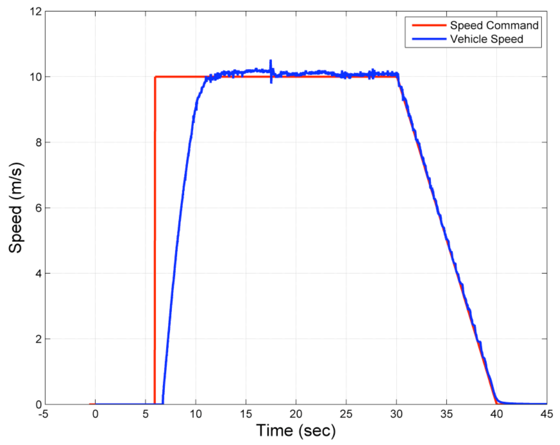


Fig.12. Step Input and Ramp Down to Speed Controller

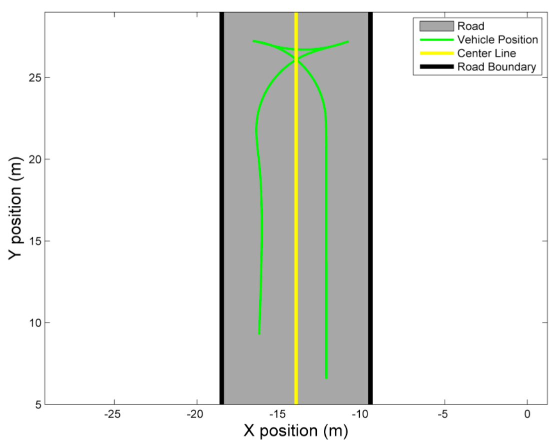


Fig.13. Vehicle Executing a 3-PointTurn

integrated testing milestones derived from performance requirements set forth by DARPA. Each subteam provided input into the schedule and now works to meet the milestones.

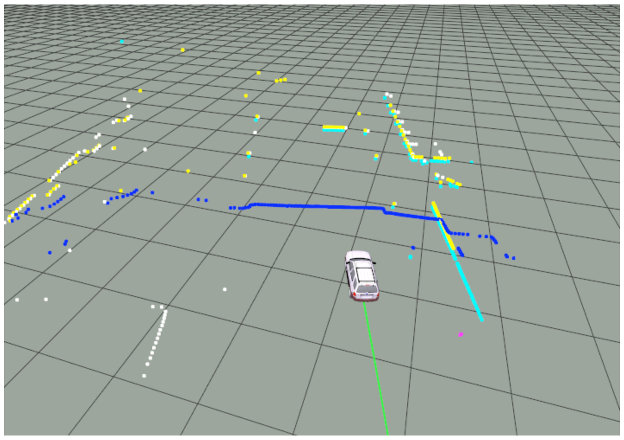
To reach our end goal of creating a system that can drive in a human-like manner, we broke the capabilities of the system down into smaller pieces and compared those capabilities to the DARPA Rules. We then mapped out our test plans for the project to ensure that none of the DARPA requirements were allowed to slip though the cracks of the larger development schedule. We have focused almost exclusively on basic navigation and basic traffic requirements thus far; following the Milestone2 Demonstration, the test plans will be updated and expanded to include all required capabilities for the National Qualifier Event.

A testing sequence always begins with a test planning meeting where the subteam leads review their test needs with the project management and testing engineers. The key personnel for each test are: (1) the *operator*, who is the software developer responsible for the code under test; (2) the *safety driver*, who drives the vehicle (when in manual mode) and actuates the emergency stop(s) if needed; and (3) the *test coordinator*, who assists with detailed test planning during the day, keeps notes during testing, and makes an independent record of test results after testing.

The first run of each test day verifies the in-car safety systems. For occupied missions, the safety driver can at anytime actuate the PAUSE button, which will bring the car to a safe controlled stop, or the DISABLE button, which terminates the engine ignition power and applies braking to create a controlled stop. For unmanned operations, the safety system is first verified with a safety driver present. The driver then exits the car and takes control of the handheld E-Stop remote which affords E-Stop PAUSE and DISABLE functionality equivalent to that of the in-car switches. Once safety tests are completed, the test team begins executing the planned tests.

B. Vehicle Controller

This section provides representative results for the vehicle controller, including speed control, steering control, and U-turn test results. Figure 12 illustrates the vehicle's response to a step command of 10 m/s and a subsequent ramp down at a deceleration of 1 m/s^2. The vehicle's response to step inputs results in less than 5% overshoot. Note that following this speed profile yields excellent results with the vehicle stopping within 1 m of the desired point on the path. Because of the size of *Talos I* and the width of the road, a U-turn must be implemented as a 3-point turn. This turn can be specified by the Situational Planner as a sequence of piecewise-linear path segments. The experimental results are illustrated in Figure 13, where three piecewise-linear path segments were used to make the vehicle successfully execute a 3-point turn within the road. Similar approaches can be used for constructing more complicated maneuvers such as U-turns and 5-point turns.


(a)

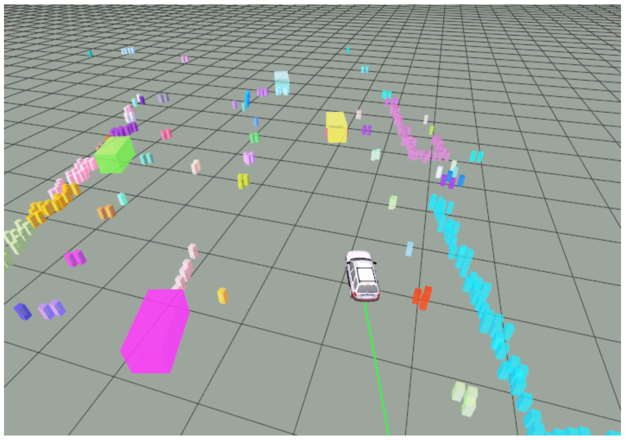
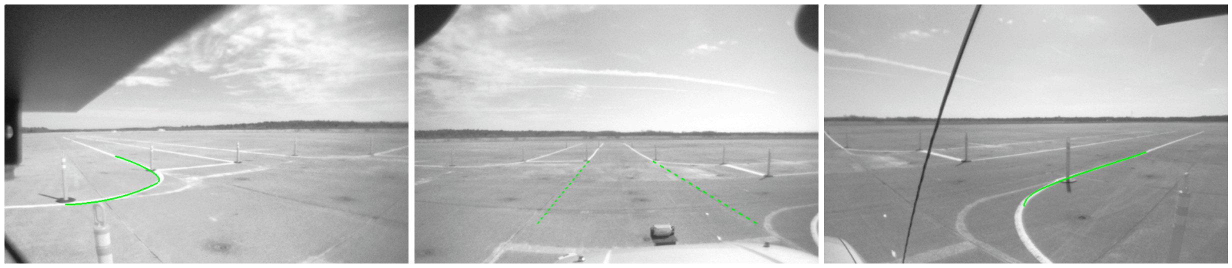

(b)

Fig. 14. Lidar Processing. Individual laser range measurements (a) are combined and grouped (b) based on spatio-temporal proximity. The resulting trajectory data is used to estimate the positions and velocities of other vehicles in the environment.


(a)

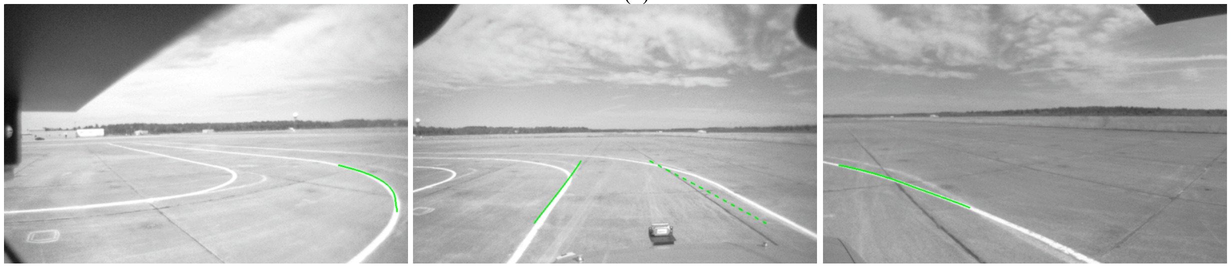

(b)

Fig. 15. Experimental results for multi-camera lane finding. (a) at intersection; (b) on curved RNDF segment. The left and right images show the output of our current MIT lane tracker; the center image shows the output of the Mobileye lane tracking system.

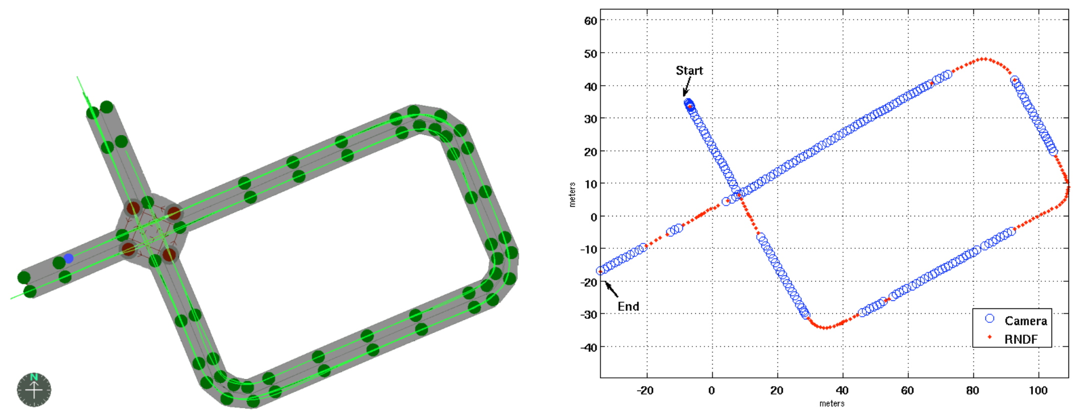
C. Perception

Figure 14(a) shows raw lidar returns for a complex traffic scenario with other vehicles in the environment. Figure 14(b) shows the output of the tracking algorithm described in Section II-E.9. Figure 15 shows results for coordinated tracking of multiple lanes detected across multiple cameras, combining a novel MIT lane tracking algorithm (for high curvature road segments) and the commercial Mobileye lane tracking system.

D. High-Level Performance

This section summarizes our quantitative performance on a number of higher-level tasks, including GPS waypoint following and repeated loop traversal, vision-based lane following, intersection precedence, and passing a stopped vehicle.

Lane following using the pure-pursuit algorithm is very tight, with the cross-track deviations from path specified to the low-level control being less than 50 cm around the site visit course. The worst-case error occurs in the corners, but this can be well predicted by the Situational Planner using simple linear models. Figure 16(a) shows the results of the car autonomously traversing the MIT Site Visit course many times in both clockwise and counter-clockwise directions. The results



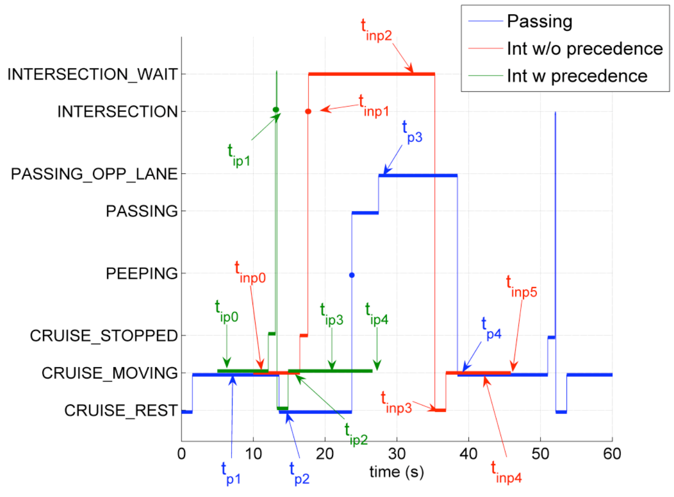
(a) (b)

Fig. 16 Lane following results on site visit course. (a) Results for five repeated traversals of the MIT Site Visit course, using RNDF lane information fed through the local map API; (b) A second experiment that demonstrates the use of the local frame representation to enable smooth transitions between perception-aided lane following (with the Mobileye system during straight

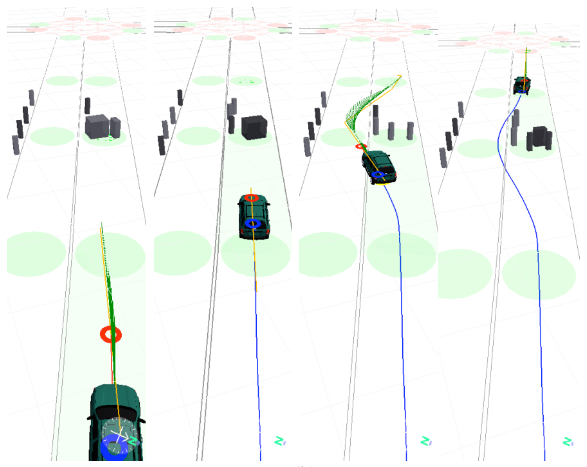
demonstrate a successful integration of the Mission Planner, Perceptual State Estimator, Situational Interpreter, Situational Planner, and Vehicle Controller, and the excellent performance and robustness of the overall planning and control system. The vehicle came to a full stop at each stop line, and the maximum speed reached in the straight segments was 17 mph, which is fast even for a human driver given the short length of these segments and the very tight turns.

Figure 16(b) shows the results for a different experiment to demonstrate the properties of the local frame perceptual strategy. The Mobileye system, designed for highway driving, provides excellent output on straight line segments, but encounters difficulties on the course's sharp turns. With our local frame perception strategy, the system is able to smoothly transition between states with and without Mobileye data in traversing the course. This capability was one of the key design goals of our perceptual system, as it is expected to provide smooth vehicle trajectories when GPS dropouts occur. The Situational Planner ensures that the car stays in the lane by querying the local map API about the feasibility of the predicted trajectory. By default, the local map uses the lanes visually detected by the Mobileye system, but it switches to RNDF-based lane-following when the confidence level of the detected lane is low. This capability demonstrates the benefits of our local frame processing strategy, in which locally-registered perceptual data takes precedence over GPS when the former has high confidence.

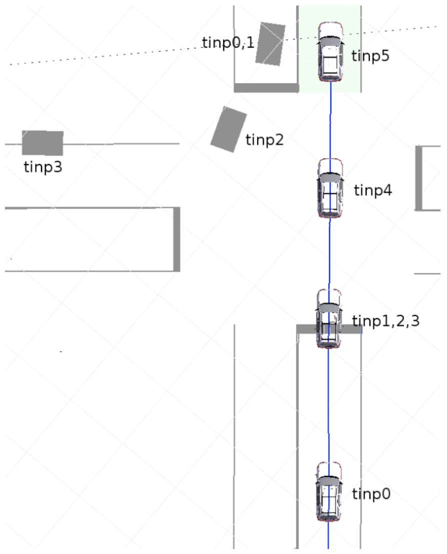
The time history of the Situational Interpreter mode switching is shown in Figure 17(a). Figure 17(b) shows the result of the car performing a passing maneuver. The obstacles were detected using the lidar and reported in the local map API. When in `CRUISE` mode (t_{p1}, t_{p2} –the first and the second slice of Figure 17(b)), the road divider constraint is active, and Situational Planner will not generate trajectories that cross the lane divider. Since in this situation, there are no feasible trajectories within standoff distance of the obstacle (set at two car lengths, the maximum queuing distance), the car will come to a stop at time t_{p2}. After a period of time (in this example, 10 seconds), the Situational Interpreter infers that the path is blocked and switches to `PEEPING` followed by `PASSING`, where the Situational Interpreter relaxes the road divider constraint. The Situational Planner then designed a trajectory around the detected obstacles. The third slice of Figure 17(b) shows the car crossing the road divider at t_{p3}, and the last slice shows the car returning to the right lane at t_{p4}, at which point the Situational Interpreter properly re-enables the road divider constraint. This demonstrates the mode switching of Situational Planner based on the different traffic rules and constraints enforced.



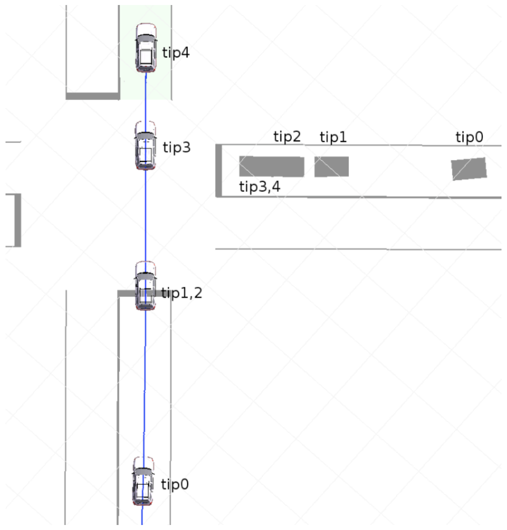
(a)



(b)



(c)



(d)

Fig. 17. Experimental results for three scenarios that illustrate high-level behavior of *Talos I*. (a) Mode time history for the three scenarios; (b) Snapshots of a passing maneuver. From left to right: cruising (t_{p1}), stopped (t_{p2}), leaving the lane to pass (t_{p3}), and returning to lane (t_{p4}); (c) Intersection traversal when *Talos I* does not have precedence; (d) Intersection traversal with precedence.

Numerous intersection scenarios have been generated to investigate vehicles arriving from various directions and times. For example, Figure 17(c) shows data for a case where the traffic vehicle has precedence over *Talos I*, and Figure 17(d) shows data for a case where *Talos I* has precedence. Figure 17(a) shows the mode transitions for both cases, with the time of the k^{th} snapshot denoted as t_{ipk} for the case that *Talos I* has precedence, and t_{inpk} for the other case. The plots show that our vehicle identified that another car was at the intersection and took the appropriate action in response.

IV. CONCLUSION

In summary, we have created two new robotic vehicles equipped for autonomous driving in urban environments using an innovative system architecture that has undergone extensive testing on a DARPA site visit course. Our team has developed novel approaches to sensing, perception, planning, and control that are well-suited to address the DGC challenges. The experimental results demonstrate all basic navigation and some basic traffic behaviors, including unoccupied autonomous driving, lane following using pure-pursuit control and our local frame perception strategy, obstacle avoidance using kino-dynamic RRT path planning, U-turns, and precedence evaluation amongst other cars at intersections using our situational interpreter. We are extending and refining these approaches for advanced navigation and traffic scenarios, and look forward to the site visit in June, the NQE in October, and the race in November.

REFERENCES

[1] S. Thrun *et al.* "Stanley: The robot that won the DARPA Grand Challenge," *J. Robot. Syst.*, vol. 23, no. 9, 2006.
[2] C. Urmson, *et al.* "A robust approach to high-speed navigation for unrehearsed desert terrain," *Journal of Field Robotics*, vol. 23, pp. 467–508, August 2006.
[3] P. Trepagnier, J. Nagel, P. Kinney, C. Koutsourgeras, and M. Dooner, "KAT-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain," *Journal of Field Robotics*, vol. 23, pp. 467–508, August 2006.
[4] E. Olson, M. Walter, J. Leonard, and S. Teller, "Single cluster graph partitioning for robotics applications," in Proceedings of *Robotics Science and Systems*, 2005, pp. 265–272.
[5] C. Tin, "Robust Multi-UAV Planning in Dynamic and Uncertain Environments," SM Thesis, MIT, Aug. 2004.
[6] S. M. LaValle and J. J. Kuffner, "Randomized kino-dynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
[7] E. Frazzoli, "Robust Hybrid Control for Autonomous Vehicle Motion Planning," Ph.D. dissertation, MIT, June 2001.
[8] A. Richards, "Robust Constrained Model Predictive Control," Ph.D. dissertation, MIT, November 2004.
[9] Y. Kuwata, "Trajectory Planning for Unmanned Vehicles using Robust Receding Horizon Control," Ph.D. dissertation, MIT, December 2006.
[10] A. Kelly and A. Stentz, "An approach to rough terrain autonomous mobility" in *International Conference on Mobile Planetary Robots*, Jan 1997.
[11] J. Y. Wong, Theory of Ground Vehicles. John Wiley, 2001.
[12] S. Park, "Avionics and control system development for mid-air rendezvous of two unmanned aerial vehicles," Ph.D. dissertation, MIT, Feb 2004.
[13] S. Patwardhan, H. Shue, J. Guldner, and M. Tomizuka, "Lane following during backward driving for front wheel steered vehicles," in Proceedings of the *American Control Conference*, June 1997.
[14] R. Mason et al., "The Golem Group/UCLA autonomous ground vehicle in the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 8, pp. 527–553, August 2006.
[15] K. Iagnemma and M. Buehler, "Special issue on the DARPA Grand Challenge: Editorial," *Journal of Field Robotics*, vol. 23, no. 9, pp. 655–656, 2006.
[16] D. Braid, A. Broggi, and G. Schmiedel, "The TerraMax autonomous vehicle," *J. Robot. Syst.*, vol. 23, no. 9, pp. 693–708, 2006.
[17] B. Leedy, J. Putney, C. Bauman, S. Cacciola, J. Webster, and C. Reinholtz, "Virginia Tech's twin contenders: A comparative study of reactive and deliberative navigation," *J. Robot. Syst.*, vol. 23, no. 9, pp. 709–727, 2006.
[18] Q. Chen and U. Ozguner, "Intelligent off-road navigation algorithms and strategies of team desert buckeyes in the DARPA Grand Challenge 2005," *J. Robot. Syst.*, vol. 23, no. 9, pp. 729–743, 2006.
[19] A. R. Atreya, B. C. Cattle, B. M. Collins, B. Essenburg, G. H. Franken, A. M. Saxe, S. N. Schiffres, and A. L. Kornhauser, "Prospect Eleven: Princeton university's entry in the 2005 DARPA Grand Challenge," *J. Robot. Syst.*, vol. 23, no. 9, pp. 745–753, 2006.
[20] L. B. Cremean, T. B. Foote, J. H. Gillula, G. H. Hines, D. Kogan, K. L. Kriechbaum, J. C. Lamb, J. Leibs, L. Lindzey, C. E. Rasmussen, A. D. Stewart, J. W. Burdick, and R. M. Murray, "Alice: An information-rich autonomous vehicle for high-speed desert navigation," *J. Robot. Syst.*, vol. 23, no. 9, pp. 777–810, 2006.
[21] R. Grabowski, R. Weatherly, R. Bolling, D. Seidel, M. Shadid, and A. Jones, "MITRE meteor: An off-road autonomous vehicle for DARPA's grand challenge," *J. Robot. Syst.*, vol. 23, no. 9, pp. 811–835, 2006.