# Alice: An Information-Rich Autonomous Vehicle for High-Speed Desert Navigation

Lars B. Cremean Tully B. Foote Jeremy H. Gillula George H. Hines
Dmitriy Kogan Kristopher L. Kriechbaum Jeffrey C. Lamb Jeremy Leibs
Laura Lindzey Christopher E. Rasmussen Alexander D. Stewart
Joel W. Burdick Richard M. Murray

Submitted, *Journal of Field Robotics*
3 June 2006

**Abstract**

This paper describes the implementation and testing of *Alice*, the California Institute of Technology's entry in the 2005 DARPA Grand Challenge. Alice utilizes a highly networked control system architecture to provide high performance, autonomous driving in unknown environments. Innovations include a vehicle architecture designed for efficient testing in harsh environments, a highly sensory-driven approach to fuse sensor data into speed maps used by real-time trajectory optimization algorithms, health and contingency management algorithms to manage failures at the component and system level, and a software logging and display environment that enables rapid assessment of performance during testing. The system successfully completed several runs in the National Qualifying Event, but encountered a combination of sensing and control issues in the Grand Challenge Event that led to a critical failure after traversing approximately 8 miles.

## 1 Introduction

Team Caltech was formed in February of 2003 with the goal of designing a vehicle that could compete in the 2004 DARPA Grand Challenge. Our 2004 vehicle, Bob, completed the qualification course and traveled approximately 1.3 miles of the 142-mile 2004 course. In 2004-05, Team Caltech developed a new vehicle to participate in the 2005 DARPA Grand Challenge. Through a Caltech course in multi-disciplinary project design, over 50 undergraduates participated in conceiving, designing, implementing and testing our new vehicle, named "Alice" (Figure 1). The team consisted of a broad range of students from different disciplines and at different skill levels, working together to create a sophisticated engineering system. The final race team completed the implementation and optimization of the system over the summer as part of the Caltech Summer Undergraduate Research Fellowship (SURF) program.

Alice's design built on many standard techniques in robotics and control, including state estimation using Kalman filters, sensor-based terrain estimation and fusion, optimization-based planning through a "map" of the terrain, and feedback control at multiple levels of abstraction. A novel aspect of the design compared with many robots built prior to the grand challenge was the high-speed nature of the system: Alice was designed to travel through unstructured environments at

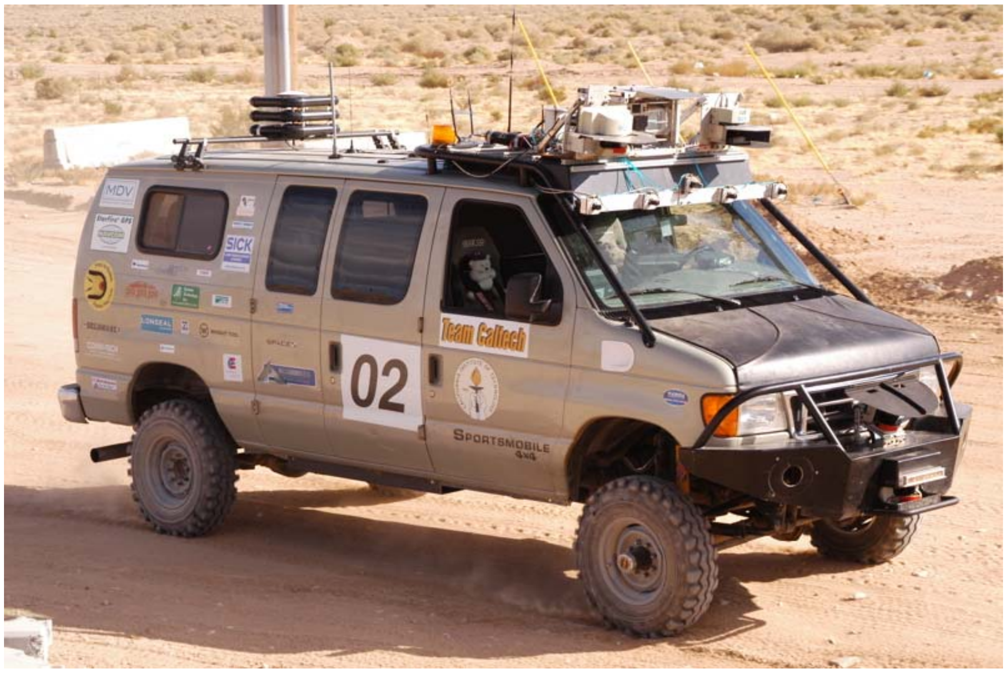


Figure 1: Caltech's 2005 DARPA Grand Challenge Entry, Alice.

speeds of up to 15 m/s (35 mph) using multiple cameras and LADARs across a network of high performance computers. The raw data rates for Alice totaled approximately 350 Mb/s in its race configuration and plans were computed at up to 10 Hz. This required careful attention to data flow paths and processing distribution, as well as the use of a highly networked control architecture. In addition, Alice was designed to operate in the presence of failures of the sensing and planning systems, allowing a high level of fault tolerance. Finally, Alice was designed to allow efficient testing, including the use of a street legal platform, rapid transition between manual and autonomous driving and detailed logging, visualization and playback capabilities.

This paper describes the overall system design for Alice and provides an analysis of the system's performance in desert testing, the national qualification event, and the 2005 Grand Challenge race. We focus attention on three aspects of the system that were particularly important to the system's performance: high-speed sensor fusion, real-time trajectory generation and tracking, and supervisory control. Data and measurements are provided for a variety of subsystems to demonstrate the capabilities of the component functions and the overall system.

Alice's design built on many advances in robotics and control over the past several decades. The use of optimization-based techniques for real-time trajectory generation built on our previous experience in receding horizon control for motion control systems (Milam, 2003; Murray *et al.*, 2003) and extended that work to include efficient methods for cost evaluation along a terrain with sometimes sparse data (Kogan, 2005). Our sensor fusion architecture and the use of speed maps built on work at JPL (Goldberg et al., 2002) and we benefited greatly from the work of Dickmanns (Dickmanns, 2004). The supervisory control architecture that we implemented also relied heavily on concepts developed at JPL (Rasmussen, 2001).

The design approach for Alice was shaped by the lessons learned from fielding a team for the 2004 Grand Challenge race, and by the shared experiences of other teams in that event, notably the technical report published by the Red Team (Urmson, 2005; Urmson et al., 2004) and the relative overall success of path-centric versus behavior-based approaches.

The deliberative approaches to solving the Grand Challenge centered on building a grid-based or obstacle-based map of the environment, and performed a search through that map for an optimal path. The field of finalists for the 2005 race partially reflected a convergence of system-level architectures to this approach; 17 of the 23 team technical papers (including those from the five vehicles that completed the course) describe various deliberative implementations (Defense Advanced Research Projects Agency, 2005).

Based on the technical papers, three teams (Axion, Virginia Tech's Cliff, and IVST) implemented a primarily behavior-based navigation architecture, and Princeton University implemented a purely reactive architecture. These alternative approaches are a source of valuable experience and experimental data, and might provide some insight into the relative merits of different approaches.

The description of Caltech's approach proceeds as follows: Section 2 describes our system architecture, from the vehicle and associated hardware to the software design. Section 3 details the specifics of the vehicle actuation and trajectory-following controller. Our mapping and planning algorithms are explained in Sections 4 and 5, and our higher-level control and contingency management is described in Section 6. Experimental results that illustrate the performance of our system are presented in Section 7.

## 2 System Architecture

Caltech's 2004 entry in the DARPA Grand Challenge utilized an arbiter based planning architecture (similar to that in Rosenblatt (1998)) in which each terrain sensor "voted" on a set of steering angles and speeds, based on the goodness determined by those sensors. These votes were then combined by an arbiter, resulting in a command-level fusion. This approach had advantages in terms of development (each voter could be developed and tested independently), but it was decided that this approach would not be able to handle complex situations involving difficult combinations of obstacles and terrain Cremean (2006). Hence a more sophisticated architecture was developed for the 2005 race, based on optimization-based planning. In addition, a high level system specification was used to guide the operation of each component in the system.

### 2.1 System Specification

Team Caltech's strategy for the race was embedded in its overall system specification, which described the performance characteristics for Alice and the team's operations. This system specification was used to drive the specifications for individual components. The final system specification contained the following requirements:

S1) 175 mile (282 km) range, 10 hours driving, 36 hours elapsed (with overnight shutdown and restart).

S2) Traverse 15 cm high (or deep) obstacles at 7 m/s, 30 cm obstacles at 1 m/s, 50 cm deep water (slowly), 30 cm deep sand and up to 15 deg slopes. Detect and avoid situations that are worse than this.

S3) Operate in dusty conditions, dawn to dusk with up to 2 sensor failures.

S4) Average speed versus terrain type:

| Terrain type | Distance (%) | | Speed (mph) | | | Expected | | Time (hr) |
|---|---|---|---|---|---|---|---|---|
| | Min | Max | Min | Max | Exp | mi | % | |
| Paved road | 1 | 10 | 20 | 40 | 30 | 18 | 10% | 0.6 |
| Dirt road | 40 | 60 | 10 | 40 | 25 | 132 | 75% | 5.3 |
| Trails | 20 | 30 | 5 | 15 | 10 | 18 | 10% | 1.8 |
| Off-road | 5 | 20 | 1 | 5 | 5 | 5 | 3% | 1 |
| Special | n/a | n/a | 2 | 2 | 2 | 2 | 1% | 1 |
| Total | | | 1 | 40 | 25 | 175 | 100% | 9.7 |

S5) Safe operation that avoids irreparable damage, with variable safety factor.

S6) Safety driver w/ ability to immediately regain control of vehicle; 20 mph crash w/out injury.

S7) Commercially available hardware and software; no government supported labor.

S8) \$120K total equipment budget (excluding donations); labor based on student enrollment in CS/EE/ME 75abc (multi-disciplinary systems engineering course) and 24 full-time SURF students.

S9) Rapid development and testing: street capable, 15 minute/2 person setup.

The speed versus terrain type specification (S4) was updated during the course of development. Expected mileages by terrain type were updated from analyses of the 2004 Grand Challenge course, and expected speeds were selected to find a "sweet spot" that balances the trade-off between chance of completing the course (whose trend generally decreases with increased speed, and favors lower speeds) and minimizing completion time (which favors higher speeds).

One of the most important specifications was the ability to do rapid development and testing (S9). This was chosen to to take advantage of being within a few hours drive of desert testing areas: our goal was to test as much as possible in race-like conditions. Our vehicle from the 2004 Grand Challenge, Bob, had to be towed by trailer to and from test sites. We had also removed Bob's steering wheel, which meant that he had to be driven using a joystick. This added unnecessary complication and effort to the testing process. With Alice, the transition from a human driving to autonomous testing only required the operator to insert a single pin to engage the brake mechanism and then flip a few switches to turn on the other actuators. This meant that we could transition from human driving to an autonomous test in only 5 minutes. Alice also supports a complement of four connected but independent interior workstations for development and testing, a vast improvement over Bob's design.

## 2.2 Vehicle Selection

To help achieve ease of development, a Ford E350 van chassis was obtained and modified for off-road use by Sportsmobile West of Fresno, CA. A diesel engine, augmented by a 46-gallon fuel tank, was selected since it was well suited to the conventional operating envelope of generally slow speeds and long idle times during periods of debugging.

Sportsmobile's conversion is primarily intended for a type of driving known as rock-crawling: low-speed operation over very rough terrain. A four-wheel-drive system distributes power through a geared transfer case (manufactured by Advanced Adapters, Inc.) and the suspension rests in

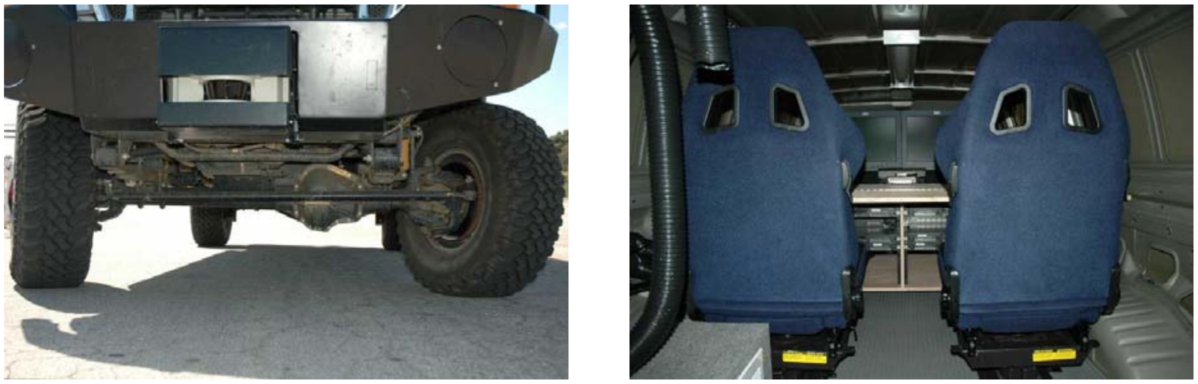

(a) (b)

Figure 2: Vehicle features: (a) front differential, suspension, and front bumper/LADAR mount and (b) the rear workstation area with server array visible (as viewed through the rear doors).

deep compression, allowing the unsprung components to drop very quickly over negative terrain discontinuities (see Figure 2(a)). Another key suspension feature is the use of a reverse shackle design in the implementation of the front leaf spring suspension. As opposed to traditional designs, the reverse shackle method places the extensible end of the leaf spring toward the rear of the vehicle. This provides better tracking at speed. The high-knuckle front axle design decreases minimum turning circle by 6% to 45 feet in diameter.

The vehicle was purchased as a "stripped chassis," which means that the vehicle was bare behind the front seats. This allowed a custom interior design that included a central enclosure for the server array and four racing seats, two of which replaced the stock Ford captain's chairs in the front of the cabin. A removable table was placed in front of the two rear seats, and five terminals were positioned throughout the vehicle: one for each seat and one at the side door, accessible from outside the vehicle. Figure 2(b) shows a view of the interior of Alice from the rear. Thus equipped, the vehicle is capable of supporting three developers and safety driver while mobile, or four developers while stationary.

The electrical system consists of a 3 kilowatt generator mounted to the rear of the vehicle, producing 120 VAC. This power is directed to two 1500 watt inverter/chargers. Power is passed through these inverters directly to the loads without loss when the generator is running, with the remaining power being diverted to charge the auxiliary battery bank. The battery bank consists of four 12 volt marine gel cell batteries rated at 210 amp-hours each, positioned beneath the floor at the rear of the vehicle to keep the center of gravity low. Power from the charged gel cell batteries is diverted back through the inverters to power AC loads when the generator is not functioning, or to the 12 volt systems powering the LADARs, IMU, GPS and actuators. The power system was tested and verified to operate in high-shock, high-vibration environments for up to 22 hours between generator refuelings.

## 2.3 Computing and Networking

One of the initial specifications for Team Caltech's second vehicle was to be able to survive the failure of one or more computers. The design of the vehicle computing and networking systems was developed based on this specification, even though this functionality was not implemented during

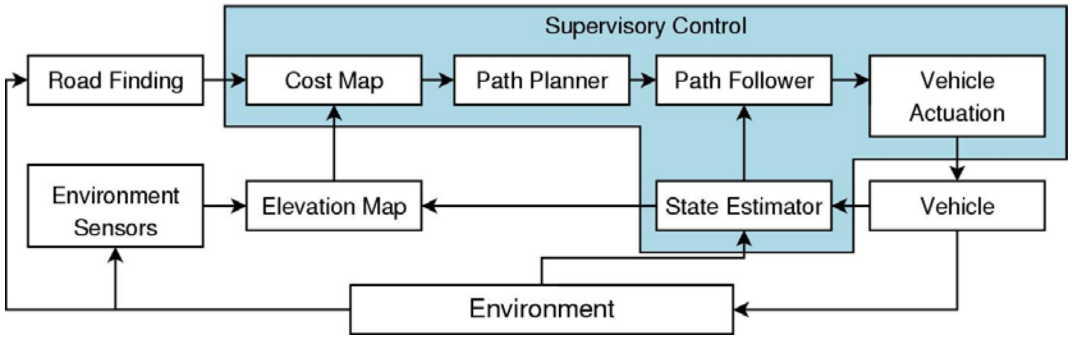


Figure 3: Overall System Architecture for Alice.

the race (due to shortness of time and the strong reliability of the hardware in testing).

The computing platform consisted of 6 Dell PowerEdge 750 servers with 3 GHz, Pentium 4 processors and a single IBM eServer 326 with dual 2.2 GHz dual-core AMD 64 processors. All machines were configured to run Linux; the Gentoo distribution (Vermeulen et al., 2005) was selected based on performance testing early in the system design. Each machine had two 1 Gb/s Ethernet interfaces, although only one interface is used in the race configuration. Originally a serial device server was used to allow any machine to talk to any actuator (all of which had serial interfaces), but the serial device server had reliability problems and was eventually removed from the system. A wireless bridge and wireless access point were also used in the system during testing to allow remote access to the vehicle network.

To allow software components to communicate in a machine-independent fashion, a custom messaging and module management system called "Skynet" was developed. Skynet was specified to provide inter-computer communication between programs on different computers or on the same computer completely transparently. The code run on Alice is therefore broken down into discrete functional modules, each derived from a Skynet class to allow Skynet to start and stop modules. This is required because Skynet was also designed be able to start, stop, and redistribute modules between computers based on computational resources available, assuming hardware failures of any type were possible. Skynet's communication capabilities are built on top of the Spread group communication toolkit (Amir et al., 2004).

In the race configuration, the ability to redistribute and run modules on different computers was not implemented and module restart was accomplished using runlevel functionality provided by Linux (Vermeulen et al., 2005). A custom runlevel was created to put each computer into race-ready mode upon entry, by running a series of scripts to start the modules appropriate for each machine. Should a given module exit or die for any reason, it is immediately restarted using the respawn setting in /etc/inittab. Paired with other scripts for recompiling and distributing our modules and configuration files, this proved to be an extremely efficient way of upgrading software and switching to autonomous operations during development and testing.

## 2.4 Software Architecture

A diagram of our general system architecture is shown in Figure 3. Environmental sensors (stereo vision and LADAR) are used to create an elevation map of the terrain around the vehicle. Our range sensor suite consisted of multiple LADAR units and stereo camera pairs. The data from these sensors, in combination with our state estimate, creates an elevation map in the global reference frame. The elevation map consists of a grid of cells, centered on the vehicle, where the value of each cell corresponded to the elevation of that cell. The map moves along with the vehicle, so as cells move some distance behind us, new cells are created in front of the vehicle.

This elevation map is then converted into a cost map by considering aspects such as elevation gradient, quality of data in that cell, etc. The cost map establishes a speed limit for each location in the terrain around the vehicle. In addition to terrain data, the speed limits set in the Route Definition Data File (RDDF) and information from road finding algorithms (Rasmussen and Korah, 2005) are integrated at this point. The speed limit-based cost map allows the vehicle to traverse rough sections of the terrain (slowly) and insures that the vehicle attempts to make forward progress unless there was an insurmountable obstacle (speed = 0) in its path. The elevation and cost mapping algorithms are discussed in more detail in Section 4.

Once the cost map is generated, it was passed onto the planner where a time-optimal path is created that satisfies vehicle and map speed constraints (Kogan, 2005). This path is sent onto the path follower, which in turn computes and sends appropriate actuation commands (brake, throttle, and steering) to the vehicle actuation system. The vehicle control systems are discussed in Section 3 and the path planning algorithm is discussed in Section 5.

The supervisory control module serves to detect and manage higher-level system faults that other individual modules cannot. This includes scenarios such as losing and reacquiring GPS, and being stuck on an undetected obstacle. The supervisory control module is also responsible for maintaining forward progress in unusual conditions, such as the case of running up against barbed wire (the failure mode for Team Caltech's 2004 entry). This system is described in more detail in Section 6.

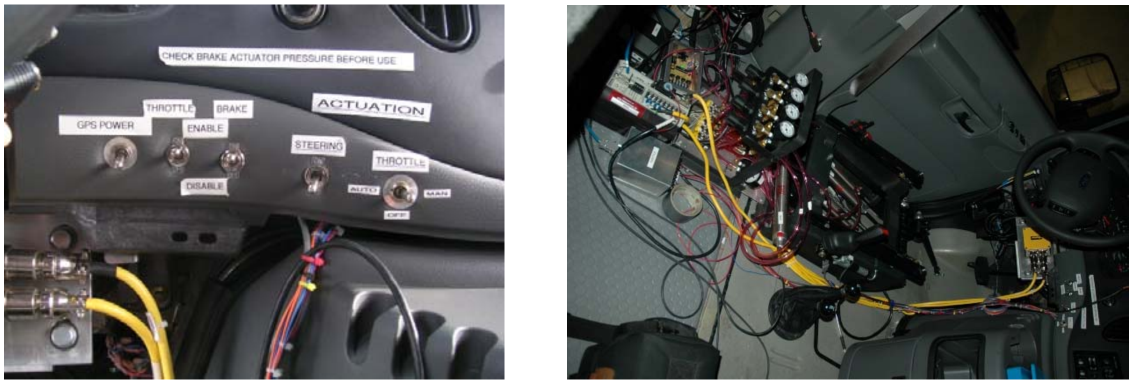
## 3 Control System Design

The lowest level of the vehicle control system consists of the vehicle's actuation system, its state sensing hardware and software, and the trajectory tracking algorithms. Like many teams in the Grand Challenge, these systems were custom designed based on commercially available hardware. This section describes the design decisions and implementation of the control system for Alice.

### 3.1 Vehicle Actuation Systems

There are five components critical to the control and operation of any vehicle: steering, acceleration control (throttle), deceleration control (braking), engine start and stop (ignition) and gear change (transmission). Two auxiliary components, emergency stop and on-board diagnostics round out the control interfaces in use on Alice. Team Caltech developed actuators and interfaces to meet each of the aforementioned needs. Each actuator was designed with safety and performance in mind. To this end, in the case of electrical, mechanical or software failure, critical actuation subsystems automatically bring the vehicle to a quick stop without needing external commands. To achieve the desired overall performance, each actuator is designed by specification to be able to react at least as fast as a human driver.

Each actuator uses its own RS-232 serial interface to communicate with the computing system. The original specification required the ability to survive one or more computer failures which dictated that any actuator had to be available to at least two computers at any time. This led to



(a) (b)

Figure 4: Vehicle actuation systems: (a) dashboard controls and (b) pneumatic braking system.

the use of an Ethernet enabled serial device server. However, through testing it was determined the mean time between failures for the serial device server was shorter than other components in the actuator communication chain. As a result the specification was dropped and direct serial connections were made to the computers.

Each actuation subsystem is controlled through individual switches accessible to a safety driver sitting in the driver's seat (Figure 4(a)). This configuration allows a great deal of flexibility in testing, including allowing the safety driver to control some functionality of the system while the software controlled other functions. The dashboard controls are also used to allow the safety driver to take over control of the vehicle during testing.

**Steering Actuator** The steering system used on Alice was originally developed for Bob and used during the 2004 Grand Challenge. It was adapted and improved for use in the 2005 competition and proved to be a highly reliable component. The basis of this system is a Parker-Hannefin 340 volt servo motor and a GV-6UE controller interfaced to the computing cluster via RS-232. This servo-motor is capable of providing 8 Nm of torque while moving and up to 14 Nm stall torque. Attached to this motor is a 3:1 single stage planetary gearbox which is attached to the steering column by means of steel chain providing a further 1.8:1 reduction. The result is a system capable of providing a continuous torque of more than 40 Nm to the steering column, which is sufficient to turn the wheels should the power steering system of the vehicle fail. The variable speed motor is tuned to allow the wheels to traverse from one end of travel limit to the other in approximately 1.5 seconds, which is slightly faster than is possible for a human. Higher speeds were tested, but despite the reduced delay, they were found to contribute to an overly aggressive behavior in the vehicle control system, as well as causing heavier wear on all components.

**Brake Actuator** Following Team Caltech's less than satisfactory experience with electrical linear actuators for braking in the 2004 Grand Challenge, a pneumatic actuation system was chosen for use this year. This choice provided a faster response while still providing enough force to bring the vehicle to a complete stop. The five piston mechanical portion of the actuator was designed and implemented by Chris Pederson of competing team A. I. Motorvators. As shown in Figure 4(b), the actuator consists of five pistons of incrementally increasing bore arrayed in parallel with their piston rods acting in tension attached to a single pivot point. A one-to-one ratio pivot positioned beneath the driver's seat changes the retracting motion of the pistons to a compressive motion applied to the brake pedal by means of a removable pushrod. Four primary pistons are used for articulate control of braking pressure during autonomous operation. Each piston may be applied and released independently by means of 5 volt active solenoid control valves, each of which is in turn controlled by solid-state relays linked to an Atmel AVR microprocessor, interfaced to the control computer. In case of electrical or air compressor failure, the fifth cylinder is automatically deployed using a separate air reserve. The entire system runs on 70 psi compressed air provided by two 2.4 cfm, 12 volt air compressors. Air is compressed, piped to a reservoir, and then distributed to the brake and sensor cleaning systems. A pressure switch located at the main manifold will close and deploy the reserve piston if the main air system pressure falls below a safe level.

**Throttle Actuation** Caltech's 2005 E-350 van shipped with a 6.0L Powerstroke diesel engine that is entirely electronically controlled, including control of engine acceleration, referred to here as "throttle". The accelerator pedal in this vehicle is simply an electrical interface to the Powertrain Control Module (PCM), which is the Ford vehicle computer. Using specifications provided by engineers from Ford, an interface was designed that closely approximates the response of the accelerator pedal. Using a microcontroller and digital to analog converters, the accelerator pedal was replaced with an effective interface that was controlled via RS-232. The stock pedal was left in the vehicle, and can be used by flipping a switch to disable the aftermarket actuator. Unfortunately, the throttle actuation solution was not perfect. The PCM requires that three sampled input lines agree to within 2.1% otherwise an error condition is declared. In this case the PCM puts the vehicle into a mode called "Limited Operating System" which restricts the engine to idle. The only method to clear the LOS condition is to restart the engine, which requires ignition actuation. Despite these minor problems, the throttle actuation performed admirably, exhibiting no measurable delay between command and actuation, as well as a high degree of operational robustness.

**Ignition and Transmission Actuation** Ignition and transmission actuation, while two completely separate sub-systems on the vehicle, were bundled together in one actuator controller out of convenience, as neither required high communication bandwidth nor was processing intensive for a microcontroller. Ignition control was achieved through the use of three 50 amp solid state relays to control three circuits: powered in run, powered in start, or powered in run and start. The ignition actuator as developed is tri-state: Off, Run or Start. The vehicle protects the ignition system so that the starter motor cannot operate if the engine is already running, and so that the engine cannot be started unless the vehicle is in park or neutral.

Transmission actuation was achieved through the use of an electronic linear actuator connected to the transmission by means of a push-pull cable. A four position controller was used to provide automated shifting into Park, Reverse, Neutral or Drive. The ignition and transmission can also be controlled by a human in one of two ways. Included in the actuator design is a control box that allows the human driver to take control and operate the vehicle by means of a push-button ignition system and a multi-position switch knob for transmission control. Alternatively each can be controlled manually after disabling the actuators.

**On Board Diagnostics** To gather additional real-time data about the vehicle's performance, Alice's CAN bus was accessed using a commercial OBD II reader. Data values such as wheel speed, throttle position, transmission position, and engine torque are gathered to provide more information for contingency management and state estimation. However the data had a large amount of latency due to poor interfaces, with overhead on the order of 10 times the data throughput. To alleviate the bandwidth limitations, asymmetric polling methods are implemented. By prioritizing how often data fields are polled, performance was increased from 0.5 Hz updates to 2 Hz for the time critical value of vehicle speed, which is still significantly less than the desired 10 Hz. Data rates for other fields have been decreased to as slow as once every 8 seconds for the lowest priority information.

**Emergency Stop** Critical to the safety of operators and observers is a method to stop the vehicle when it is autonomous. DARPA provided the basis for this remote safety system in the form of an "E-Stop" system consisting of a transmitter-receiver pair manufactured by Omnitech Robotics. Using a 900 MHz transmitter, an operator outside the vehicle and up to 11 miles (line of sight) away can send one of three commands: RUN, PAUSE, or DISABLE. RUN allows normal operation of the vehicle, PAUSE brings the vehicle to a full and complete stop with all components still functioning, and DISABLE powers down the throttle, turns off the engine, and deploys full braking pressure. The receiver inside the vehicle is connected to a microcontroller interfaced to the computing cluster. This device also takes input from several PAUSE and DISABLE switches located throughout the vehicle, and then passes the most restrictive state to the vehicle interface, called Adrive, which is described in the next section. This system is built with many failsafes to ensure safe operation over a variety of failure modes, including loss of power and communication. The emergency stop system is designed so that even in the event of a full computing system or failure, the vehicle can be remotely brought to a safe stop.

## 3.2 Vehicle Interface

A software module, called Adrive, was specified to provide an abstracted network interface between all the vehicle actuators and computer control. The primary role for the module was to listen for commands on the network then execute them within 50 ms. The second role was to report regularly each actuator's current state (status and position). And a third role of the abstraction was to protect the vehicle from being damaged by control logic or system failures.

All the current state data as well as configuration settings were contained in a hierarchical data structure. Adrive used a multi-threaded design to allow multiple interrupt driven tasks, such as serial communication, to operate concurrently. For optimal performance the command threads would only execute on new commands when the actuator is ready, instead of queuing commands. This allows subsequent commands to have less lag. The maximum delay of the caching system is only

$$\text{Max Delay} = 1/f_{\text{actuator}} + 1/f_{\text{command}},$$

versus the minimum possible delay

$$\text{Optimal Delay} = 1/f_{\text{actuator}},$$

where $f_{\text{actuator}}$ is the frequency of the actuator update and $f_{\text{command}}$ is the frequency of the incoming command. So as long as $f_{\text{command}}$ is much larger than $f_{\text{actuator}}$ this approach approximates optimal.

Within Adrive, two levels of fault tolerance were implemented. At the command level every incoming command was run through a set of rules to determine if the command was safe to execute. For example, when the vehicle is in park and command is received to accelerate, the command will not be executed. In addition to physical protection these low level of rules also encompassed the procedures for PAUSE and DISABLE conditions.

At the actuator level there was a supervisory thread which periodically checks each of the actuators for reported errors or failure to meet performance specifications. If the error occurred on a critical system the supervisory thread will automatically PAUSE the vehicle and attempt to recover the failed subsystem. This can be seen in the case where the engine stalls. The OBD-II will report the engine RPM is below a threshold. The supervisory thread will detect the low RPM as below acceptable and put the vehicle into PAUSE mode, which immediately stops the vehicle. The command will then be sent to restart the engine. When the OBD-II reports the RPM above the threshold the PAUSE condition will be removed and operation will return to normal.

The use of Adrive to abstract simultaneous communication between multiple software modules and multiple actuators worked well. The computer resources required are usually less than 1% CPU usage and required less 4MB of memory. Response times range from 10 ms to 100 ms. However the maximum bandwidth of approximately 10 Hz is limited by the serial communications. The final structure proved to be a relatively adaptable and scalable architecture to allow many asynchronous control interfaces accessible to multiple processes across a network.

## 3.3 State Sensing

The estimate of vehicle state ($X$) is made up of the vehicle's global position (northing, easting, altitude), orientation (roll, pitch, yaw), both the first and second derivatives of these values, as well as the precise time-stamp for when this measurement was valid. The requirements and specifications of tolerances for planning, trajectory tracking and sensor fusion dictate how timely and accurate estimation of Alice's state needs to be. The main specification we believed necessary for "safe driving" was to be able to detect a 20 cm obstacle at 80 m. Doing so requires our system to estimate the terrain with no more than 10 cm of relative error over 80 m of travel, and orientation estimates accurate to within 0.001 radians (based on the approximation $\arctan(0.1/80) \approx 0.001$).

We chose to approach the state estimation problem by combining the outputs of an inertial measurement unit (IMU), global positioning system (GPS), and on-board velocity measurement (OBD II). The output of the IMU ($Z^{imu}$) is integrated forward from the current position estimate according to the standard inertial navigation equations:

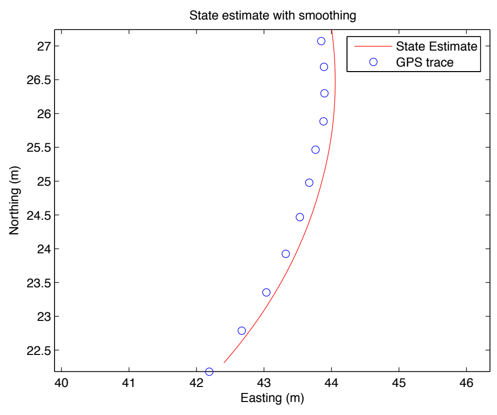
$$X_{t+1}^{imu} = Nav(X_t, Z_{t+1}^{imu}).$$

Unfortunately, while the IMU provides smooth relative position that is accurate over a short time, errors in the initial estimate, IMU biases, and scale factors lead to a solution that drifts from the true position quadratically.

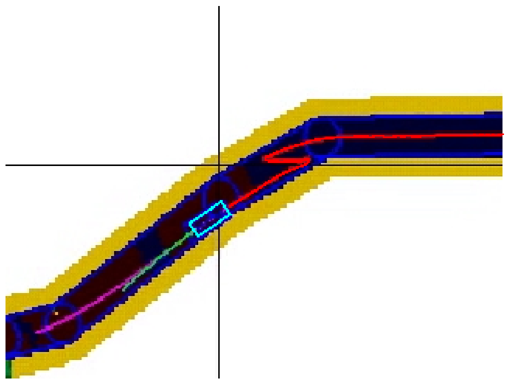
A measurement of the true position or velocity ($Z^{true}$) can be read directly from the GPS unit or OBD II system. In the case of OBD II velocity, time delays are such that only the stationary condition is used, still allowing us to eliminate drift when stationary, even in the absence of a GPS signal. By taking the difference between our IMU and GPS or OBD II based state estimates, we arrive at a measurement of our error ($\Delta X$):

$$Z_t^{\Delta X} = X_t^{imu} - Z_t^{true}.$$

We then use an extended Kalman filter to estimate $\Delta X$, using the self-reported covariance of the GPS measurement as the covariance of $Z^{\Delta X}$. $\Delta X$ is propagated forward in time according to a



(a)



(b)

Figure 5: Vehicle state estimate: (a) smoothing of small GPS signal and (b) sample GPS "jump": the trace shows the estimated vehicle state, with a correction while the vehicle is stopped (from second NQE run, just after the tunnel).

linearization of the equations of motion being integrated in the inertial solution, and an estimate of the errors in the IMU readings. Corrections are then applied to the true state estimate by subtracting these estimated errors from the inertial solution, effectively zeroing out the error values:

$$\begin{aligned} X_t &= X_t^{imu} - \Delta X_t \\ \Delta X_t &= 0. \end{aligned}$$

An additional constraint largely influencing the construction of the state estimator was the fact that our maps are built in a global reference frame. The problem is that previous state errors are cemented into the regions of the map where we are not getting new terrain measurements. This has large ramifications on correcting position errors, since updating our state to the correct location creates a jump discontinuity, which in turn creates a sharp, spurious ridge in the map. As such, there was a trade-off to be made between reporting the most accurate state at a given time, and the most "useful" estimate of state from the point of view of the other modules.

We took two approaches to solving this problem. One solution is to smooth the applied corrections. Rather than applying a correction all at once, a fraction of the correction is applied at each time step, leading to a smoothed out exponential decay in error rather than a sharp discontinuity:

$$\begin{aligned} X_t &= X_t^{imu} - c\Delta X_t \\ \Delta X_t &= (1-c)\Delta X_t. \end{aligned}$$

For small jump-discontinuities this is sufficient to avoid problems with discontinuities, but increases the amount of tracking error, as shown in Figure 5(a).

However, occasionally, when new GPS satellites are picked up or lost (such as when going through a tunnel, or under power-lines), the GPS reading itself can jump by many meters. To deal with this problem we pre-process the GPS data to determine when it is "safe" to incorporate new measurements. As long as the effective jump is below a threshold, GPS data can be continuously applied, making small corrections and keeping the state estimate from ever drifting. If the GPS jumps by a large enough amount (2 meters in the race configuration), we temporarily ignore it, assuming it is more likely a glitch in GPS. During this time, however, the covariance in positional error begins to grow. If the covariance crosses an empirically determined threshold, we deem it necessary to start incorporating GPS measurements again, and bring the vehicle to a pause while we apply the new corrections. The vehicle is allowed to return to forward motion when the state estimator indicates that the estimate has converged, as determined by looking at the magnitude of the differences between subsequent state estimates. Because of the use of a global representation for the cost map (and hence the implicit location of obstacles), the elevation and cost maps for the system are cleared at this point. This "jump" functionality is implemented through the use of a supervisory control strategy as described in Section 6. A sample state jump is shown in Figure 5(b).

## 3.4 Trajectory Tracking

The design specification for the trajectory tracking algorithm is to receive a trajectory from a planning module and output appropriate actuator commands to keep Alice on this trajectory. The inputs to the algorithm are the current state of the vehicle (position and orientation, along with first and second derivatives) and the desired trajectory (specified in northing and easting coordinates, with their first and second derivatives). From these inputs, the algorithm outputs steering and brake/throttle commands to Adrive. Goals for accuracy were +0/-10% for velocity tracking, and $\pm 20$ cm perpendicular $y$-error at 5 m/s, with larger errors allowable at higher speeds. These performance criteria needed to be met on any terrain type found in the system specifications, at speeds up to 15 m/s.

**System Characterization** Before designing the controller, it was necessary to characterize the open-loop dynamics of the system. With this characterization, a mapping from actuator positions to accelerations was obtained. They showed that Alice understeers, and allowed the determination of safe steering commands at various speeds, such that the vehicle would remain in the linear response region. In this region, the feedforward term will be reasonable, and possibly dangerous roll angles/sliding are avoided. Additionally, system delays were determined by examination of the time between commands leaving this module and the resulting vehicular accelerations.

**Control Law Design** Although not entirely independent, the lateral and longitudinal controllers are treated separately in the system design. Longitudinal (throttle and brake) control is executed by a feedback PID loop around error in speed plus a feedforward term based on a time-averaged vehicle pitch, to reduce steady-state error when traveling up or down hills.

The trajectories received as input to the trajectory follower encoded first and second derivative data as well as geometry of the path, so that desired velocity and acceleration are encoded. For the longitudinal controller, we decided not to use a feedforward term associated with acceleration based on the input trajectory. This was determined by experience, as there were occasions where the feedforward term would overpower the feedback, and simultaneous tracking of speed and acceleration was not achievable. For example, the vehicle might not correct error associated with going slower than the trajectory speed if the trajectory was slowing down.

The lateral control loop includes a feedforward term calculated from curvature of the path along with a PID loop around a combined error term.

The error for the lateral PID is a combination of heading and lateral errors:

$$C_{\mathrm{err}} = \alpha \tilde{Y}_{\mathrm{err}} + (1 - \alpha)\theta_{\mathrm{err}},$$

where $\tilde{Y}_{\mathrm{err}}$ is the lateral position error (saturated at some maximum value $Y_{\max}$), $\theta_{\mathrm{err}}$ is the heading error and $\beta$ is a scale factor. This form was motivated by a desire to obtain stability at any distance from the path. Using this error term, the (continuous) vector field in the neighborhood of a desired path will be tangent to the path as $Y_{\mathrm{err}} \to 0$ and will head directly toward the path at distances greater than $Y_{\max}$ away from the path.

Note that the use of this error term requires an accurate estimate of the vehicle heading. Systematic biases in this estimate will result in steady-state error from the desired path.

The lateral feedforward term is generated by determining the curvature required by the input trajectory, and applying the mapping for steering position to curvature, which yields

$$\phi_{\mathrm{FF}} = \arctan\left( L \frac{\dot{N}\ddot{E} - \dot{E}\ddot{N}}{(\dot{N} + \dot{E})^{\frac{3}{2}}} \right), \tag{1}$$

where $N$ is the northing position, $E$ is the easting position and $L$ is the distance between front and rear wheels. For this calculation, it is assumed that Alice behaves as described by the bicycle model (McRuer, 1975).

To avoid oscillations, an integral reset was incorporated in both the lateral and longitudinal controllers, when the relevant error was below some acceptable threshold. In testing, the lateral integral term rarely built up to any significant amount, since performance of the system maintained modest errors comparable to the threshold. For the longitudinal controller, resetting helped to alleviate the overshoot associated with transferring from hilly to flat ground.

To compensate for system delays, a lookahead term was added. This chose the point on the trajectory that lateral feedforward and longitudinal feedback would be computed from.

**System Interfacing** Many of the difficulties in implementing the tracking controller were due not to the actual control algorithm but to interface issues with the trajectory planner and the state estimation software. The controller is relatively simple, but has to make certain assumptions about the data and commands it is receiving from the other modules. Working out these interfaces made the most difference in performance. The trajectory tracking module (trajFollower) interfaced with three other modules: the trajectory planner (plannerModule), the state estimator (Astate) and the software that directly controlled the actuators (Adrive).

Since the planner has no concept of the actual topography of the area, a feasibility evaluator was implemented to check final commands against vehicle state. This is broken down into two components: a "bumpiness" sensor (DBS) and a dynamic feasibility estimator (DFE). The DBS algorithm analyzes the frequency components of the state signal to determine how rough the road is, and adjusts the reference velocity accordingly. In testing on dirt trails, this component choose speed limits comparable to a human's judgment (within 1 m/s). The DFE algorithm places hard limits on safe steering angles, dependent on our speed. These limits were chosen based on where Alice began to understeer.

In the final implementation, the quality of the state estimate provides the most variability in trajectory following performance. Small jumps in state are detrimental to stability, as they lead to an immediate jump in proportional error, and thus steering command. More problematic are the

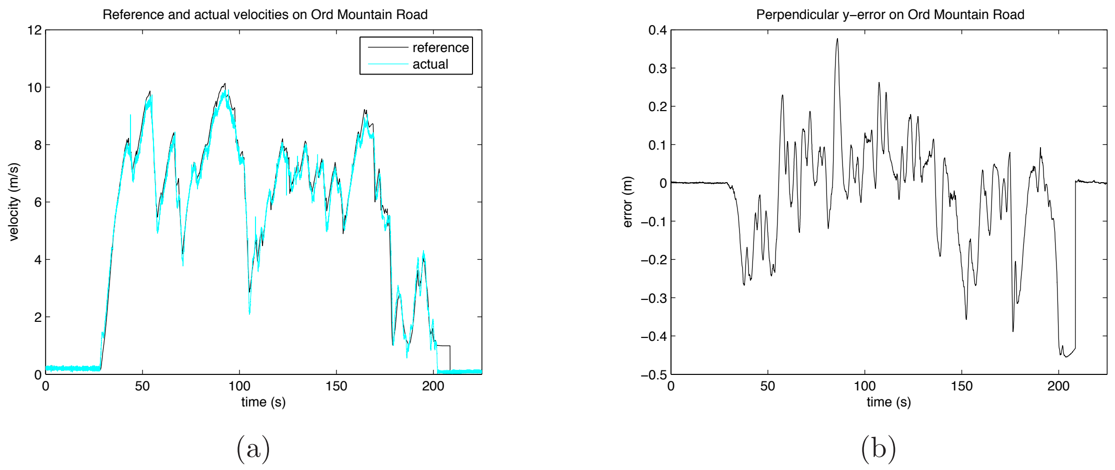


Figure 6: Trajectory tracking results: (a) longitudinal control and (b) lateral control. The resetting of the error at the end of the trajectory is due to the planner replanning from current vehicle state when the vehicle was externally paused.

intermittent difficulties with poor heading estimates, which lead to constant $y$ errors in tracking the trajectory.

The interface between the planner and trajectory following has to make assumptions about how aggressive trajectories may be and what type of continuity they possess. Obviously, the sequential plans need to be continuous for the controller to be stable. However, this leaves open the question of what they are continuous with respect to. Initially the plans originated from the current vehicle state, but this led to oscillations due to the similarity of the rates between the two loops (and the consequent phase delays in the loop transfer function). Instead, the final race configuration used the planner to update plans based on new terrain, and information continuity was then established with the initial portion of the previous trajectory, both in space and in time. One side effect of this approach is that it is possible for situations to occur where the vehicle physically cannot do what is commanded. To handle these cases, it is necessary for the planner to reset and replan from vehicle state whenever thresholds for position or velocity errors are exceeded.

**Tracking Performance** Figure 6 shows representative results of the performance of the trajectory tracking algorithms for longitudinal and lateral motion. The algorithm was tested extensively prior to the qualification event and demonstrated excellent performance in testing and during the multiple qualifying event runs.

## 4 Sensing and Fusion

Alice uses a sensor suite of an IMU, GPS, range sensors and monocular vision to perceive its own state and that of the environment around it (see Table 1 for the full list). Obstacle detection is performed using a combination of several SICK and Riegl LADAR units, as well as two pairs of stereovision cameras. Figure 7 shows Alice's sensor coverage. This combination of disparate sensors was chosen to allow Alice to be robust to different environments as well as to multiple sensor failures—a critical requirement in our specification for autonomous desert driving.

Table 1: Sensors used on Alice.

| Sensor Type | Mounting Location | Specifications |
|---|---|---|
| LADAR (SICK LMS 221-30206) | Roof | 180° FOV, 1° resolution, 75 Hz, 80 m max range, pointed 20 m away |
| LADAR (SICK LMS 291-S14) | Roof | 90° FOV, 0.5° resolution, 75 Hz, 80 m max range, pointed 35 m away |
| LADAR (Riegl LMS Q120i) | Roof | 80° FOV, 0.4° resolution, 50 Hz, 120 m max range, pointed 50 m away |
| LADAR (SICK LMS 291-S05) | Bumper | 180° FOV, 1° resolution, 80 m max range, pointed 3m away |
| LADAR (SICK LMS 221-30206) | Bumper | 180° FOV, 1° resolution, 80 m max range, pointed horizontally |
| Stereovision Pair (Point Grey Dragonfly) | Roof | 1 m baseline, 640x480 resolution, 2.8 mm focal length, 128 disparities |
| Stereovision Pair (Point Grey Dragonfly) | Roof | 1.5 m baseline, 640x480 resolution, 8 mm focal length, 128 disparities |
| Road-Finding Camera (Point Grey Dragonfly) | Roof | 640x480 resolution, 2.8mm focal length |
| IMU (Northrop Grumman LN-200) | Roof | 1–10° gyro bias, 0.3–3 mg acceleration bias, 400 Hz update rate |
| GPS (Navcom SF-2050) | Roof | 0.5 m CEP, 2 Hz update rate |
| GPS (NovAtel DL-4plus) | Roof | 0.4 m CEP, 10 Hz update rate |

To accurately navigate through its environment, it is necessary to provide Alice with a method of fusing the sensor data from its range sensors into a single representation of its environment that can capture information concerning both where it would be safest and fastest for it to drive. The decision of what sort of representation to use was driven primarily by the requirements of the path-planning software: it expected as its input a speed limit map, where each cell in the map contained a maximum speed intended to limit the speed at which the vehicle could drive through that cell. The problem of sensor fusion is thus naturally broken down into two sub-problems: how the incoming sensor data should be fused, and how the speed limit data should be generated from the raw sensor data. The resulting algorithm involves three basic steps (Figure 8):

1. For every range sensor, incoming data is transformed into global coordinates (UTM for $x$ and $y$, and altitude in meters for $z$) using the vehicle's state estimate, and then averaged into the map along with any existing data from that sensor. This creates several individual elevation maps (one per sensor).

2. For every elevation map, a conversion is performed to transform the elevation data into speed limit data, based on a set of heuristic measures of goodness.

3. Matching cells from all of the speed limit maps are averaged together to produce a single speed limit map, which is passed to the path-planning software.

A more detailed description of each stage of the process, as well as an analysis of some of the strengths and weaknesses of our approach, is given below.

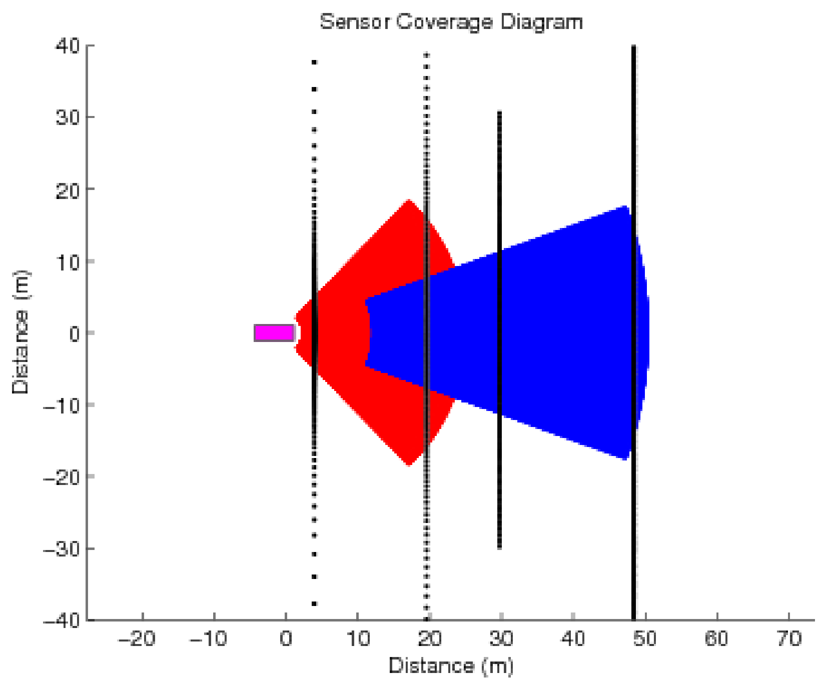


Figure 7: Alice's sensor coverage. Black dotted lines indicate the intersection of LADAR scan planes with the ground, the shorter, wider cone indicates ground covered by the short-range stereovision pair, and the longer, narrower cone indicates coverage by the long-range stereovision pair. The box to the left is Alice, for scale.

## 4.1 Map Data Structure

The map data structure we use is a simple 2.5D grid with fixed cell length and width (40 cm per side for short-range sensors, 80cm for long-range sensors), as well as fixed overall map length and width (200 m per side for all sensors). The map is referenced using global coordinates (UTM coordinates for $x$ and $y$, altitude for $z$), and is scrolled along with the vehicle's movement such that the vehicle always remained at the center of the map. The data structure of the map itself was made flexible, so that cells could contain whatever information was deemed necessary.

The first step of the sensor fusion algorithm is to use the incoming range data, as well as the vehicle's state estimate, to create an individual digital elevation map (DEM) for each sensor. After performing the appropriate coordinate transform on the range data, a simple averaging approach is used to integrate new measurements with old ones, using the equation:

$$\begin{aligned} z_{ij} &\leftarrow (z_{ij} + z_m)/(n_{ij} + 1) \\ n_{ij} &\leftarrow n_{ij} + 1, \end{aligned} \tag{2}$$

where $z_{ij}$ corresponds to the estimate of the height of a given cell in the grid, $n_{ij}$ is a count of the number of measurements that fall into the cell $i, j$, and $z_m$ is a new height measurement for that cell. Due to time constraints, we were not able to take into account error models of either the range sensors or the state estimates when creating our elevation maps. This is one of the reasons we chose to perform sensor fusion in the speed limit domain: it dramatically reduced the sensitivity of

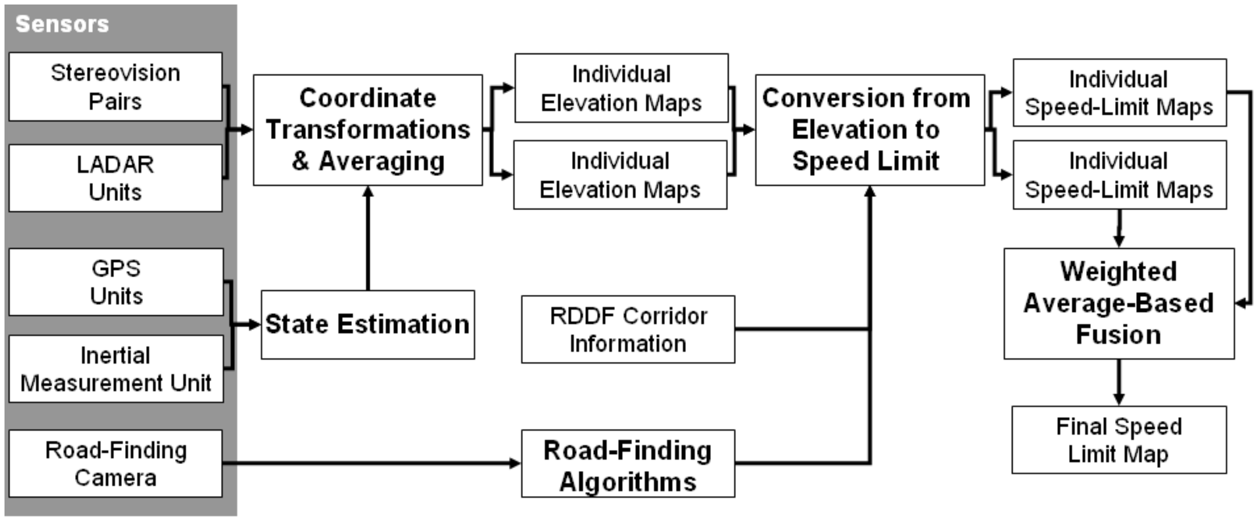

Sensors
Stereovision Pairs
LADAR Units
GPS Units
Inertial Measurement Unit
Road-Finding Camera
Coordinate Transformations & Averaging
State Estimation
Individual Elevation Maps
Individual Elevation Maps
RDDF Corridor Information
Road-Finding Algorithms
Conversion from Elevation to Speed Limit
Individual Speed-Limit Maps
Individual Speed-Limit Maps
Weighted Average-Based Fusion
Final Speed Limit Map

Figure 8: The sensor fusion framework.

the system to cross-calibration errors in the range sensors. Measurements are fused by the system as they come in, and after each scan (or frame) the resulting changes are sent on to the next stage of the algorithm.

### 4.1.1 Elevation to Speed Limit Conversion

In the next stage of the algorithm, the system needs to convert elevation data into speed limit data. We use two independent measures for this task. The first measure is the variance in elevation of the cell: the larger the variance, the larger the spread in elevation measurements that fall into that cell, the more likely that cell contains some sort of vertical obstacle, and thus the more dangerous the cell is to drive through. If the variance is greater than a certain threshold, the cell is identified as an obstacle, and the speed limit $s_1$ of that cell is set to zero.[1] Otherwise, the speed limit is set to some maximum value. Thus the variance acts as a very rough binary obstacle detector, able to detect obstacles that are completely contained within a single cell (e.g., a fence post, which is thinner than a cell width or length, but certainly tall enough to pose a danger to the vehicle).

The second measure we use for determining the speed limit of a cell is a discrete low-pass filter over a window surrounding that cell. The precise equation for the low-pass filter for a cell at row $r$ and column $c$ is

$$l(r,c) = \frac{1}{n} \sum_{\substack{i=-K \\ i\neq 0}}^{K} \sum_{\substack{j=-K \\ j\neq 0}}^{K} |(z_{r+i,c+j} - z_{r,c}) * G_{i,j}(\sigma)| \tag{3}$$

where $K$ is the half-width of the window (e.g., the window has $2K+1$ cells per side), $z_{r,c}$ is the elevation of the cell at row $r$ and column $c$, $n$ is the number of cells used within the window (at most $(2K+1)^2$), and $G_{i,j}(\sigma)$ corresponds to the value at cell $i, j$ of a discrete Gaussian centered at the origin with standard deviation $\sigma$. In essence then, the filter acts as a Gaussian weighted average of the difference in height between the central cell and its neighbors within the window described by

[1]In fact, a small non-zero value is used to avoid numerical difficulties in the path planner.

$K$. Of course, not all of the neighboring cells necessarily have an elevation estimate. For example, a large obstacle may cast a range shadow, blocking the sensor from making measurements behind the obstacle. When such a cell is encountered in the sum, it is discarded and $n$ is reduced by one. Thus we have $n = (2K + 1)^2 - 1 - m$, where $m$ is the number of cells without elevation data inside the window. Additionally, if $n$ is smaller than some threshold, then the output of the filter is not computed, based on the principle that the system did not have enough information about the neighborhood surrounding the center cell to make a reasonable estimate of the area's roughness.

Assuming the output is computed, however, we still need to transform it into a speed limit. To accomplish this transformation, we pass the output through a sigmoid of the form

$$s_2 = h * \tanh(w * (l(r, c) - x)) + y, \tag{4}$$

where $l(r, c)$ is the response of the low-pass filter given above, and the parameters $h$, $w$, $x$, and $y$ are tuned heuristically by comparing sample filter responses to the real-life situations to which they corresponded. Then, to compute the final speed $s_f$, we simply take the minimum of $s_1$, the speed generated using the variance, and $s_2$, the speed generated using this roughness filter.

Of course, the computations described here are not computationally trivial; to do them every time a single new elevation data point is generated would be infeasible given that some of the sensors are capable of generating hundreds of thousands of measurements per second. Additionally, over a small time interval (e.g., less than one second) measurements from a sensor will frequently fall into a small portion of the entire map. To take advantage of this redundancy, and to increase the computational speed of the algorithm, speed limits are generated at a fixed frequency instead of on a per-measurement basis. At some predefined interval the algorithm sweeps through the map, regenerating the speed limit of any cell whose elevation value has changed since the last sweep. Additionally, any cells whose neighbors have changed elevation values are also re-evaluated, as the response of the roughness filter could be different. To speed this process up, the algorithm uses a dynamic list to keep track of which specific cells have changed. Any time a cell receives a new measurement, the algorithm adds that cell to the list of cells to update during the next sweep (if the cell has not been added already). The result is an algorithm that takes advantage of the redundancy of subsequent measurements from a single sensor to transform a digital elevation map into an accurate speed limit map quickly and efficiently.

## 4.2 Speed Limit-Based Fusion

After computing a separate speed limit map for each sensor, all that remains is to fuse these different maps together into a single final speed limit map. We chose to use a simple weighted average, using heuristics to determine the weights for each sensor. Thus the equation for determining the final fused speed limit $s(r, c)$ of a cell at row $r$ and column $c$ is

$$s(r, c) = \frac{\sum\limits_i s_i(r, c) * w_i}{\sum\limits_i w_i}, \tag{5}$$

where $s_i(r, c)$ is the speed limit of the corresponding cell in the speed limit map of sensor $i$, and $w_i$ is the weight associated with the sensor $i$. However, this algorithm has a serious flaw: obstacles occasionally disappear and then reappear as they entered the range of each sensor (see Figure 9). This phenomenon is due to the weights used for fusing the sensors: sensors that are pointed closer

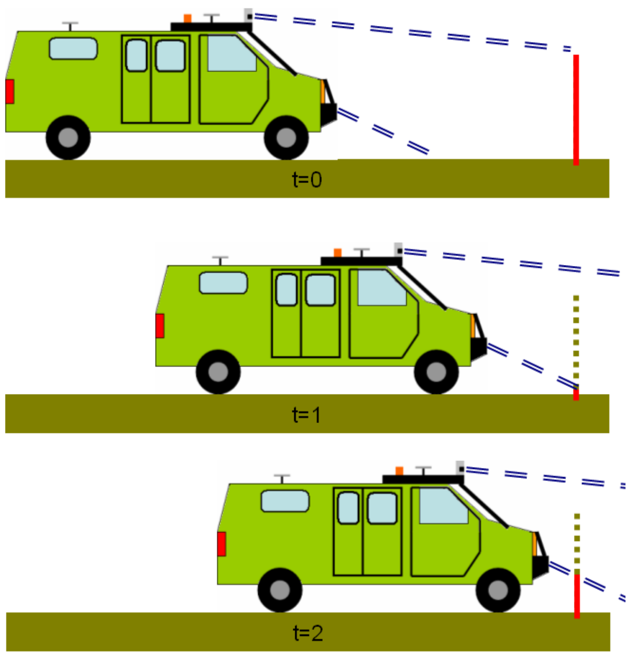


Figure 9: The disappearing obstacle problem.

to the vehicle have higher weights than sensors that are pointed further away, on the principle that the closer a sensor is pointed, the less susceptible its output is to synchronization errors between range measurements and the vehicle's state estimate. As a result, a chain of events can occur in which:

1. Sensor A, a long-range sensor, picks up measurements from an obstacle and assigns the cells that contain it a low speed.

2. The very bottom of the obstacle comes into the view of sensor B, a short-range sensor.

3. The cell containing the leading edge of the obstacle is set to a higher speed than was there before because: sensor B sees only the part of the obstacle near to the ground; seeing only the part near the ground, sensor B perceives only a very small obstacle (if it detects one at all); and finally, sensor B has a higher weight than sensor A.

4. Sensor B sees the entirety of the obstacle, and the cell containing the obstacle is once again set to a low speed. At this point, it is usually too late for the vehicle to react, and a collision occurs.

To solve this problem, we introduce an additional term into the weighted average that served to favor cells that have more measurements (and thus were more complete). The equation becomes

$$s(r,c) = \frac{\sum_i s_i(r,c) * w_i * n_i(r,c)}{\sum_i w_i * n_i(r,c)}, \tag{6}$$

where $n_i(r,c)$ is the number of measurements by sensor $i$ that were contained in the cell at $(r,c)$. As with the cost-generation algorithm, cost-fusion is done at a fixed frequency instead of on a per-cell basis, and only cells whose speed limits have changed since the last sweep are updated.

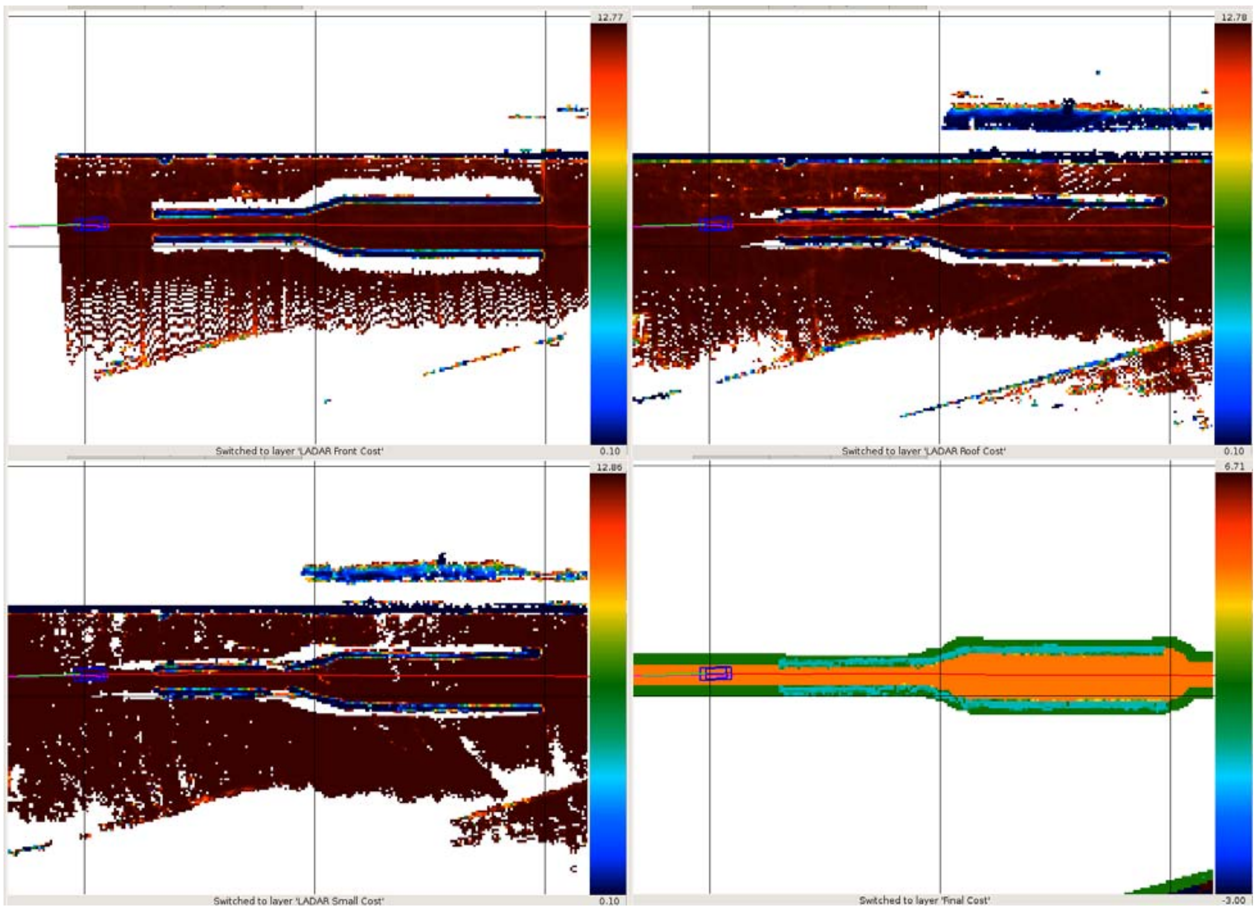


Figure 10: Example input speed limit maps and a final output map. In each case, the small rectangle to the left is the vehicle (which is traveling from right to left) and the line extending behind it is its previous path. (For scale, the grid lines are 40 m apart.) From bottom-left, moving clockwise: speed limit map generated by a midrange LADAR; speed limit map generated by a short-range LADAR; speed limit map generated by a second midrange LADAR; the combined speed limit map. The combined speed limit map has also taken into account information about the race corridor.

## 4.3 Additional Information: Roads, RDDF Corridors, and No Data

Although the sensor fusion problem makes up the bulk of the work, additional information is available for integration into the final speed limit map.

**Road Following** Alice's road following algorithm (Rasmussen, 2006) uses a single calibrated grayscale camera and a bumper-mounted SICK LADAR. The best-fit vanishing point of the road ahead is repeatedly extracted from the pattern of parallel ruts, tracks, lane lines, or road edges present in the camera image, yielding a target direction in the ground plane. This visual information is augmented with estimates of the road width and Alice's lateral displacement from its centerline, derived from the LADAR returns, to populate a "road layer" in the combined cost map used by the planner. Additionally, a failure detection process classifies images as road or non-road to

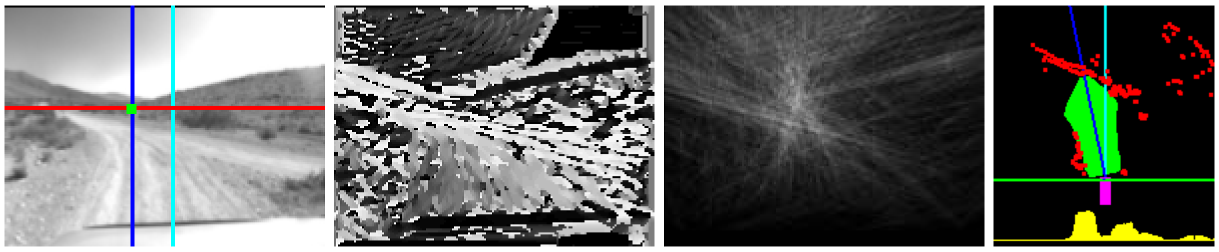

(a) (b) (c) (d)

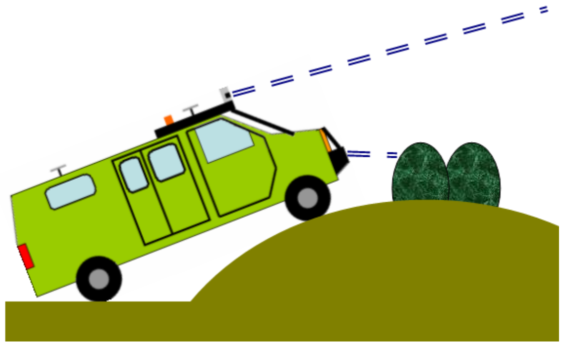
Figure 11: Sample road follower output on test run: (a) computed road direction and Alice direction (image projections of diagonal, vertical lines, respectively, in (d)); (b) dominant orientations ($[0, \pi]$ angle proportional to intensity); (c) vanishing point votes $\mathbf{I}_{VP}$; (d) overhead view of estimated road region (green/lightly shaded area): Alice is the rectangle, bumper LADAR returns are the scattered points, LADAR-derived obstacle function $h(x)$ is shown in at bottom.

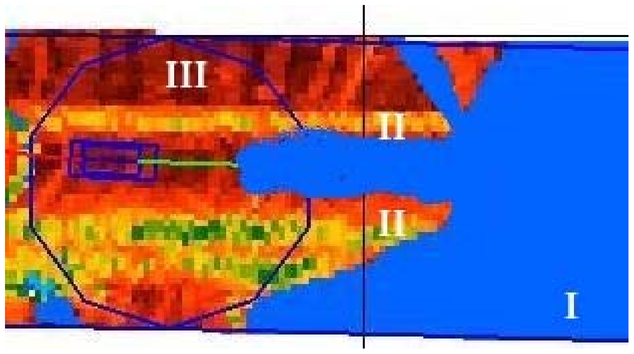
appropriately turn off the road layer and monitors for sun glare or excessive shadow conditions that can confuse the vanishing point finder. Figure 11 illustrates the operation of the algorithm.

To obtain the road vanishing point, the dominant texture orientation at each pixel of a down-sampled $80 \times 60$ image is estimated by convolution with a bank of $12 \times 12$ complex Gabor wavelet filters over 36 orientations in the range $[0, \pi]$. The filter orientation $\theta(\mathbf{p})$ at each pixel $\mathbf{p}$ which elicits the maximum response implies that the vanishing point lies along the ray defined by $\mathbf{r_p} = (\mathbf{p}, \theta(\mathbf{p}))$. The instantaneous road vanishing point for all pixels is the maximum vote-getter in a Hough-like voting procedure in which each $\mathbf{r_p}$ is rasterized in a roughly image-sized accumulation buffer $\mathbf{I}_{VP}$. Using $\mathbf{I}_{VP}$ as a likelihood function, we track the road vanishing point over time with a particle filter.

The road region is computed by projecting LADAR hit points along the vision-derived road direction onto a line defined by Alice's front axle, deweighting more distant points, to make a 1-D "obstacle function" $h(x)$. The middle of the gap in $h(x)$ closest to the vehicle's current position marks Alice's lateral offset from the road midpoint; it is tracked with a separate particle filter. If sufficient cues are available, this road-finding system is able to identify a rectangular section of the area in front of Alice as a road. Using the assumption that roads are generally safer to drive on (and thus the vehicle should drive faster on them), this data is integrated into the existing speed limit map using a heuristic algorithm as a speed bonus. As a result, roads appear as faster sections in the map, enabling the path-planning algorithm to keep Alice on roads whenever possible.

**RDDF Corridor** The final (and perhaps most important) piece of information that is incorporated is the race corridor. Incorporating this information into the map is trivial: if a given cell falls inside the race corridor, then the value of that cell is set to be the minimum of the corridor speed limit and the terrain and road-based speed limit. If a cell falls outside the corridor, its speed is set to zero. In effect, the race corridor acts simply as a mask on the underlying terrain data: no inherent assumptions are made in the mapping software that rely on the corridor information. In fact, testing was frequently performed in which the corridor speed limit was essentially set to infinity, and the corridor's width was set to be dozens of meters wide. Even with these loose bounds, the mapping software was easily able to identify drivable areas and generate accurate speed limit


(a)



(b)

Figure 12: Example no-data problem. (a) As the vehicle crests the hill, its sensors will detect the bushes lining the road before they detect the road itself. (b) A corresponding sample no-data map (with the vehicle traveling from left to right). If area I is no-data, and is a lower speed than area II (corresponding to obstacles) and area III (corresponding to flat terrain), then the vehicle may prefer to drive in no-data areas (I) instead, even if they correspond to dangerous obstacles.

maps.

**No-Data Cells** One final topic that we have alluded to, but not covered in detail, is what we call the no-data problem: in many cases, not all of the cells in a given area of the map will contain elevation measurements. For example, no range measurements will be returned when a sensor is pointed off the side of a cliff, or from behind an obstacle that is obstructing the sensor's field of view. This problem can be partially alleviated by making some assumptions about the continuity of the terrain in the environment, and interpolating between cells accordingly. Our software performs some interpolation (by filling in small numbers of empty cells based on the average elevation of surrounding cells that did have data). Despite this, there are still situations where large patches of the map simply have no data, and interpolating across them could lead to incorrect estimates of the nature of the surrounding terrain. The question, then, is how the mapping software should treat those areas: should it be cautious and set them to have a low speed limit? Or should it make assumptions about the continuity of the surrounding terrain and set them to have a high speed limit? We found that for robust performance, a compromise between these two extremes was necessary.

Clearly, setting no-data cells to a high speed limit would be inviting trouble. For example, the vehicle may decide it prefers to drive off a cliff because the open area beyond the cliff is a high-speed no-data region. However, setting no-data cells to a speed limit of zero can also result in undesirable behavior. Consider a set of steep rolling hills with a dirt road down the center, and bushes or boulders lining the sides. As the vehicle crests a hill, the first thing its sensors will detect are the bushes lining the side of the road (Figure 12(a)). In particular, these bushes may be detected and placed into the map prior to the vehicle detecting the dirt road below, resulting in a speed limit map that looks something like Figure 12(b). In this situation, if the no-data speed is set too low (that is, lower than the speed assigned to the cells containing the bushes) the vehicle may prefer to drive through the bushes instead of stay on the road. Clearly this is undesirable—as a result, a compromise must be made where the no-data speed is low (so as to avoid the vehicle

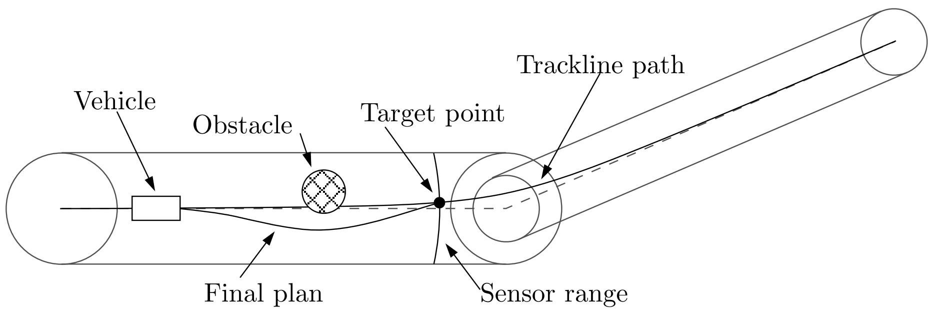


Figure 13: An illustration of the receding horizon framework. A very inaccurate (but easily computable) path is used as the global plan. Here this is a spline based on the trackline. The solver that produces a higher-quality plan performs its computation to reach a point on the lower quality path, some distance ahead. In this figure, that distance (the horizon) is set to the range of the sensors of the vehicle.

barreling across uncharted terrain at high speed) but not lower than the speed associated with most obstacles. The interpretation of no-data cells is performed by the path planning module, described in the next section.

## 5 Navigation and Planning

The navigation system for Alice was designed as two discrete components: a planner that generates trajectories for the vehicle to follow, and a controller that follows those trajectories. The controller was described in Section 3.4; this section will describe the design and implementation of the trajectory planner.

### 5.1 Planner Approach

The top level design goal of the planning system is to be able to traverse the DARPA-specified course while satisfying DARPA's speed limits and avoiding obstacles. An ideal planner will produce trajectories that are "best," in some sense. Thus a numerical optimization method lies at the core of Alice's planning system. Since the DGC race is over 100 miles long, it is both computationally prohibitive and unnecessary to plan the vehicle's trajectory to the finish line. An appropriate solution to deal with this problem is to run the planner in a receding horizon framework. A receding horizon scenario is one where plans are computed not to the final goal, but to a point a set horizon (spatial or temporal) ahead on some rough estimate of the path towards the goal (Figure 13).

The vehicle model used by the planning system is a rear-centered, kinematic model:

$$\begin{aligned}
\dot{N} &= v\cos\theta \\
\dot{E} &= v\sin\theta \\
\dot{v} &= a \\
\dot{\theta} &= \frac{v}{L}\tan\phi,
\end{aligned} \tag{7}$$

where $N$, $E$ are Cartesian spatial coordinates, $\theta$ is yaw (measured from north to east), $v$ is the scalar speed, $a$ is the vehicle acceleration, $L$ is the vehicle wheelbase and $\phi$ is the steering angle.

To apply a numerical optimization scheme to the path planning problem, we have to represent the space of all potential trajectories as a vector space. To cut down on the dimensionality of this vector space, we represent the spatial part of the trajectory as a quadratic spline of $\theta(s)$ where $s$ is length along the trajectory normalized to the total length of the trajectory, $S_f$, and the temporal part as a linear spline of $v(s)$. With this representation, we can represent all variables in the system model (7) as functions of derivatives of $\theta(s)$, $v(s)$, $S_f$:

$$\begin{aligned} N(s) &= N_0 + S_f \int_0^s \cos(\theta(s))ds & a &= \frac{v}{S_f}\frac{dv}{ds} & \tan\phi &= \frac{L}{S_f}\frac{d\theta}{ds} \\ E(s) &= E_0 + S_f \int_0^s \sin(\theta(s))ds & & & & \\ \dot{N}(s) &= S_f \cos(\theta(s))v(s) & \dot{\theta} &= \frac{v}{S_f}\frac{d\theta}{ds} & \dot{\phi} &= \frac{LS_f\frac{d^2\theta}{ds^2}}{S_f^2 + \left(L\frac{d\theta}{ds}\right)^2}. \\ \dot{E}(s) &= S_f \sin(\theta(s))v(s) & & & & \end{aligned} \tag{8}$$

To pose the planning problem as an optimization problem, we need an objective to optimize and constraints to satisfy, all functions of some state vector. Our state is $S_f$ and the spline coefficients of $\theta(s)$ and $v(s)$. We try to find trajectories that are fastest, while keeping the steering and acceleration control effort low. Thus our objective function is

$$J = S_f \int_0^1 \frac{1}{v(s)}ds + k_1 \left\|\dot{\phi}(s)\right\|_2^2 + k_2 \left\|a(s)\right\|_2^2. \tag{9}$$

Besides the initial and end condition constraints, we also satisfy dynamic feasibility constraints:

$$\begin{array}{lccccc} \text{Speed limit :} & & & v & < & v_{\text{limit}} \\ \text{Acceleration limit :} & a_{\min} & < & a & < & a_{\max} \\ \text{Steering limit :} & -\phi_{\max} & < & \phi & < & \phi_{\max} \\ \text{Steering speed limit :} & -\dot{\phi}_{\max} & < & \dot{\phi} & < & \dot{\phi}_{\max} \\ \text{Rollover constraint :} & -\frac{gW}{2h_{\text{cg}}} & < & v^2\frac{\tan\phi}{L} & < & \frac{gW}{2h_{\text{cg}}}. \end{array} \tag{10}$$

In the rollover constraint expression, $W$ is the track of the vehicle (distance between left and right wheels), $h_{\text{cg}}$ is the height of the center of gravity of the vehicle above ground, and $g$ is the acceleration due to gravity. This expression is derived from assuming flat ground and rollover due purely to a centripetal force. In reality, on many surfaces sideslip will occur much before rollover, so this constraint has an adjustment factor.

Obstacle avoidance enters into the problem as the speed limit constraint, with the RDDF and terrain data processed to produce a combined, discrete map that represents a spatially dependent speed limit. What this means is that the areas outside of the RDDF and those that lie inside obstacles have a very low speed limit, and thus any trajectory that goes through those areas is not explicitly infeasible, but *is* suboptimal. This representation allows the obstacle avoidance and quick traversal conditions to be cleanly combined.

The processing of $v_{\text{limit}}$ from the discrete map generated from the sensors as described in Section 4, to a $C_1$ surface was a nontrivial matter. A lot of thought went into the design of this component, but its details are beyond the scope of this paper. More information is available in (Kogan, 2006).

Since the numerical optimizer is only locally optimal, and the problem is only locally convex, choosing a good seed for the solver is an important prerequisite to a successful solution cycle. The seeding algorithm used for Alice's planner consists of a coarse spatial path selector, planning several times beyond Alice's stopping distance. This approach evaluates quickly, introduces a needed element of long-term planning, and works well to select a good local basin of attraction.

## 5.2 System interfacing

During the development of the planning system the challenges were not limited to the planning system itself: a substantial amount of effort went towards interfacing the planner with the other system components.

One issue that needed to be addressed was the interaction of the controller and the planner, and the location of the feedback loop. If a feedback loop is contained in both the controller (through tracking error feedback) and the planner (by planning from the vehicle state), these two feedback loops interact unfavorably and produce spurious oscillations in the closed loop system performance. Since these two feedback loops work on the same time scale, these interactions are natural, and one of the feedback loops has to be eliminated to address the problem. The controller feedback can be eliminated by using only feedforward control input. This creates a very clean system, but requires relatively quick replanning, and a high fidelity model for the trajectory generation. Since there are no theoretical guarantees for the numerical optimizer convergence, this option was rejected in favor of moving all of the feedback into the controller: the controller feeds back on its tracking errors, and the planner plans from a point on its previously computed plan. In this way, the vehicle motion does not directly affect the planner's operation, and the oscillations are eliminated. This can result in a potential issue of runaway poor tracking. To address this concern, the planner *does* plan from the vehicle's state if the tracking errors grow beyond some predefined threshold.

An issue observed during the testing of the planner was an indecisiveness about which way to go to avoid a small obstacle. The local optimality of the planner could repeatedly change the avoidance direction between planning cycles, resulting in an ungraceful, late swerve to avoid an obstacle that has been detected long before. Since the seeding algorithm chooses the basin of attraction, the fix was placed there. By adding a slight element of hysteresis to the seeding algorithm to lightly favor solutions close to the previous solution, the balance between the two avoidance directions was broken, and a consistent direction was favored.

A third interaction challenge that was discovered during the planner development was the treatment of no-data cells in the map. This issue was briefly mentioned in Section 4, but the planning-side issues are described here. While all map cells that the planner gets from the mapping components represent a specific speed limit that the planner should satisfy, this is not as clear for no-data cells. To avoid being overly cautious, or overly brazen around no-data, a dual interpretation of no-data was adopted. First, a threshold distance is computed as the larger of twice the stopping distance or 10 m. Then, any no-data cell closer than the threshold distance to the vehicle is treated as a very slow cell, while no-data beyond the threshold distance is treated as twice-the-current-speed. This representation attempts to avoid or slow down through no-data cells where they could be dangerous (at the vehicle), but does not let no-data negatively influence long-term navigation decisions.

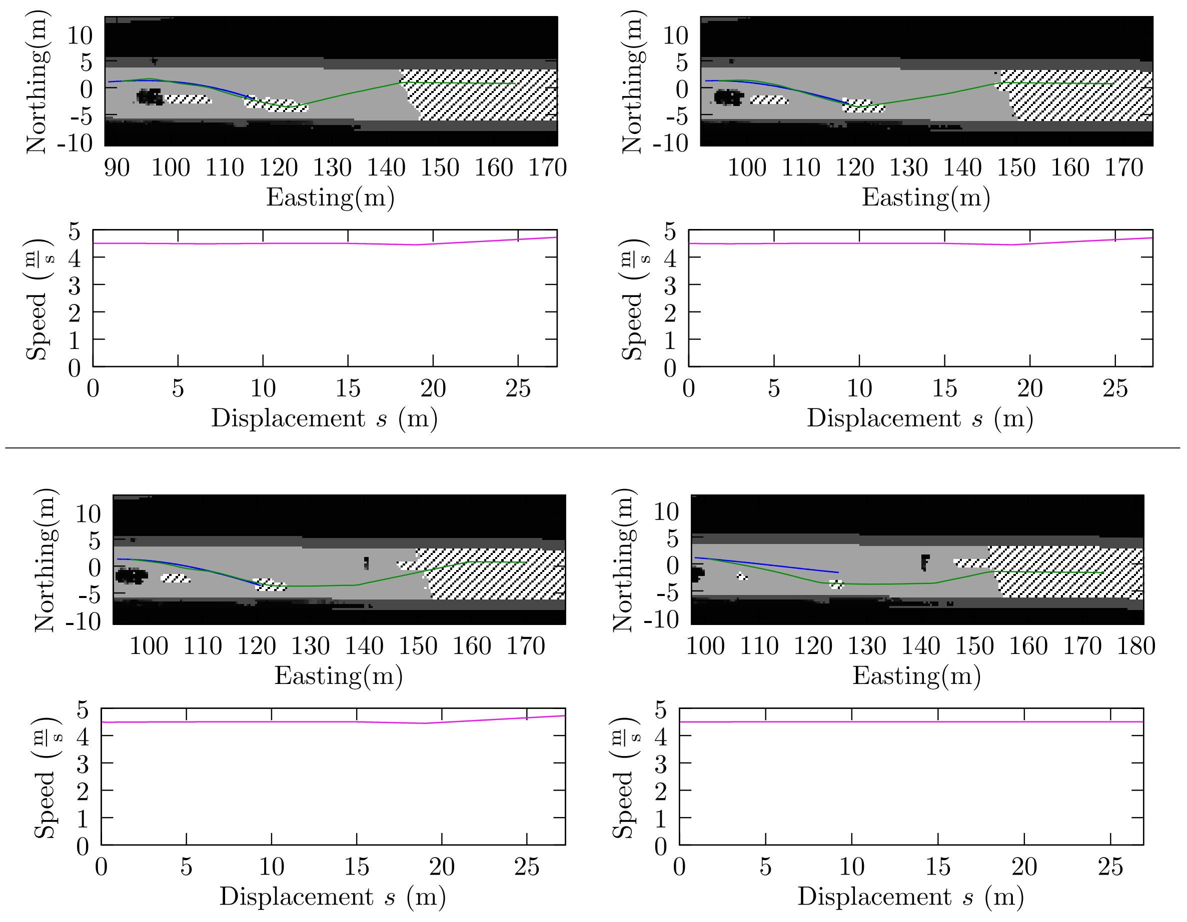


Figure 14: Non-consecutive planner iterations showing Alice avoid a car at the NQE. The vehicle is traveling east. The planner seed is the lighter, long path and the final solution the darker (and shorter) path. Grayscale colors are the speed limits in the map, with hashed regions representing no-data. Cars south and straight ahead of the vehicle are visible, along with the sensor shadow of no-data for the vehicle straight ahead. We can see distant no-data being treated favorably. We can also see a change of plans to avoid the newly detected second car.

### 5.3 Planner results

With these issues taken care of, the planning problem was solved by an off-the-shelf numerical optimizer, SNOPT. This optimization-based approach provides a very clean planning solution, and avoids most heuristic aspects of the planners that were so ubiquitous in the planning systems of most other teams. Optimizer convergence speed is a potential issue, but on Alice's 2.2 GHz Opteron, an average rate of 4.28 plans/second was achieved during the race. Further, this system produces very drivable and precise trajectories, which allows for very tight obstacle avoidance in potentially difficult situations (more than sufficient for the DGC). Some planner performance from the National Qualifying Event is illustrated in Figure 14.

## 6 Contingency Management

Experience has shown that a major challenge in the successful deployment of autonomous systems is the development of mechanisms to allow these systems to continue to operate in unanticipated (non-nominal) conditions. An important subset of these non-nominal conditions is system-level faults. These are situations in which all the individual components of the system are operating correctly, that is to say as detailed in their specifications, yet the system as a whole is unable to achieve its stated goals.

In small-scale, comparatively simple systems, a state machine is often used as the system level controller. The principal advantages of this approach are the ability of the designer to specify through a state-table exactly how the system can evolve so that it responds effectively to the specific faults they envisage. As the state-table defines exactly how the system can evolve, only the stability of the system in the scenarios described by the state table need be considered, reducing the scale of the robustness problem and typically leading to a comparatively short development time. However the state machine approach does not scale well with system complexity or size, and as it cannot execute multiple states simultaneously is not suited to situations in which multiple connected faults may have occurred.

A recent approach combining state-feedback with multiple rule-based filtering stages is demonstrated in JPL's Mission Data System (MDS) (Rasmussen, 2001). MDS seeks to provide a very flexible and expandable architecture for large and complex systems, such as unmanned space missions, that is capable of managing multiple faults simultaneously. However, the disadvantages of this approach are the absence of a known method to verify system stability and robustness, hence significant time is required to develop or enhance the system while maintaining previous empirically proven system robustness.

### 6.1 Problem Specification

The primary objective of the supervisory controller, SuperCon, is to ensure that where physically possible Alice continues to make forward progress along the course as defined by the RDDF. In situations where forward progress is not immediately possible (e.g., intraversible obstacle in front of Alice's current position), SuperCon should take all necessary action to make it possible. This objective encompasses the detection, and unified response to, all system-level faults from which it is possible for the system to recover given its designed functionality, performance, constraints and assumptions. SuperCon is not required to consider specific responses to hardware failure since these are handled at a lower level. It should also be noted that Alice does not carry any backups for its actuators. It is however a requirement of the mapping software that it be resistant to up to two sensor failures. In addition, Alice's computer cluster is configured such that in the event that any machine powers down it will restart (if physically possible), and if any module crashes it will be immediately restarted.

The secondary objective of SuperCon is to modify the overall system configuration/status if required by a module under certain conditions and manage the effects of these changes upon the different components of the system. SuperCon should also verify its own actions, and make decisions based on the accuracy of the information available to it. SuperCon must also be stable, robust, easy to expand to respond to new issues as they are discovered through testing, fast to test and verify and quick to initially assemble.

In essence the presence of SuperCon should only increase the performance, functionality, stability and robustness of the complete system and so increase Alice's chances of winning, it should never lead to a decrease in system performance by impeding correct operation.

## 6.2 Hybrid Architecture

The hybrid architecture developed has a state table and a series of rule-based filtering layers that are re-evaluated each cycle and stored in a diagnostic table. The resulting framework shares the general characteristics with MDS, described above. The state machine is composed of ten strategies, each of which consists of a sequence of stages to configure the system to achieve a specified goal (e.g., reversing requires changing gear in addition to the reconfiguration of several affected software modules). Each stage is composed of a series of conditional tests that must be passed in order for the (single) SuperCon action (e.g., change gear) to be performed, and hence the stage to be marked as complete. If a stage is not completed, and the current strategy is not updated, the next time the state machine steps forward it will re-enter the (same) stage. This allows the state machine to loop until conditions are met while always using the current version of the diagnostic table. If the test was a transition condition (which determines whether SuperCon should move to another strategy) and evaluates to true, the corresponding strategy change is made, for execution in the next cycle of the state machine.

Half of the strategies are stand-alone strategies that are responsible for managing Alice's response to special case scenarios (e.g., the safe reincorporation of GPS after a sustained signal outage). The other half form an interconnected ring that can be used to identify and then resolve any identified scenarios in which Alice would be unable to make forward progress along the course. The hybrid architecture effectively expands the capabilities of the state machine approach through rule-based filtering, and a strategy-driven approach making it easier to design for large, complex systems while retaining its desirable robustness and ease of expansion properties.

The SuperCon hybrid structure consists of two permanent threads, the state update thread maintains the live version of the state table that holds the current and selected history data for all SuperCon state elements and is purely event-driven (by the arrival of new messages). The deliberation thread by contrast is prevented from executing above a defined frequency (10Hz) to prevent unnecessary processor loading.

The SuperCon state machine only attempts to complete one stage each time it "steps forwards". Once it has finished processing a stage (regardless of whether it was successfully completed) the state machine suspends execution and requests a new copy of the state table. Once the new copy of the state table has been received, and the diagnostic rules table has been re-evaluated, the state machine takes another step forward.

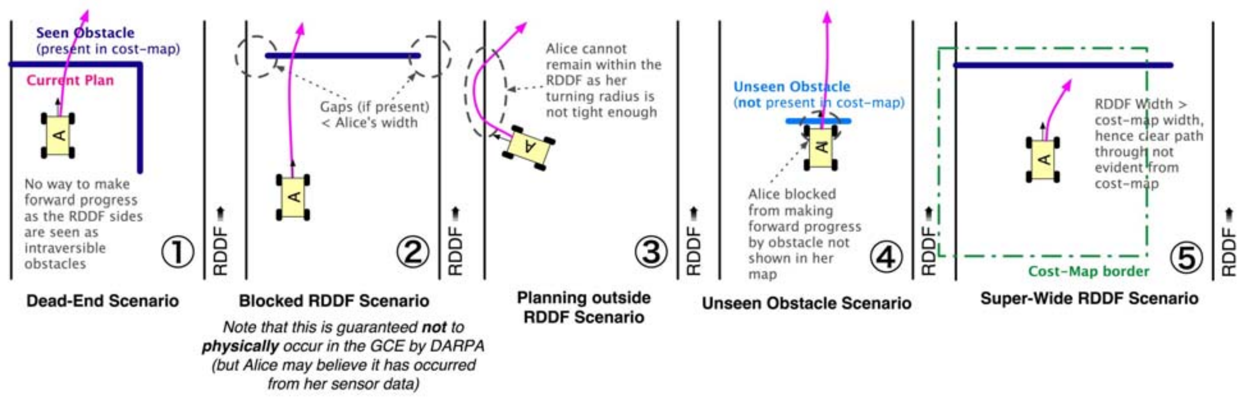

Seen Obstacle
(present in cost-map)
Current Plan
No way to make forward progress as the RDDF sides are seen as intraversible obstacles
①
RDDF
Dead-End Scenario
Gaps (if present) < Alice's width
②
RDDF
Blocked RDDF Scenario
Note that this is guaranteed not to physically occur in the GCE by DARPA (but Alice may believe it has occurred from her sensor data)
Alice cannot remain within the RDDF as her turning radius is not tight enough
③
RDDF
Planning outside RDDF Scenario
Unseen Obstacle
(not present in cost-map)
Alice blocked from making forward progress by obstacle not shown in her map
④
RDDF
Unseen Obstacle Scenario
RDDF Width > cost-map width, hence clear path through not evident from cost-map
Cost-Map border
⑤
RDDF
Super-Wide RDDF Scenario

Figure 15: Diagram showing prototypical examples of the five general NFP (No Forward Progress) scenarios identified.

A significant advantage of the hybrid approach detailed above is that as it is modular, it is comparatively easy to test as the components can be verified independently in sequence.

In order to make SuperCon robust, every time the SuperCon state machine takes an action (e.g., sends a request to change gear), before any additional action is taken SuperCon verifies that any action associated with the previous stage has been successfully completed. Each of these verification stages has an associated time-out. If the time-out is exceeded then SuperCon assumes a fault has occurred, and reverts to the Nominal strategy, from which SuperCon is reset and can re-evaluate which strategy should be active and then transition to it.

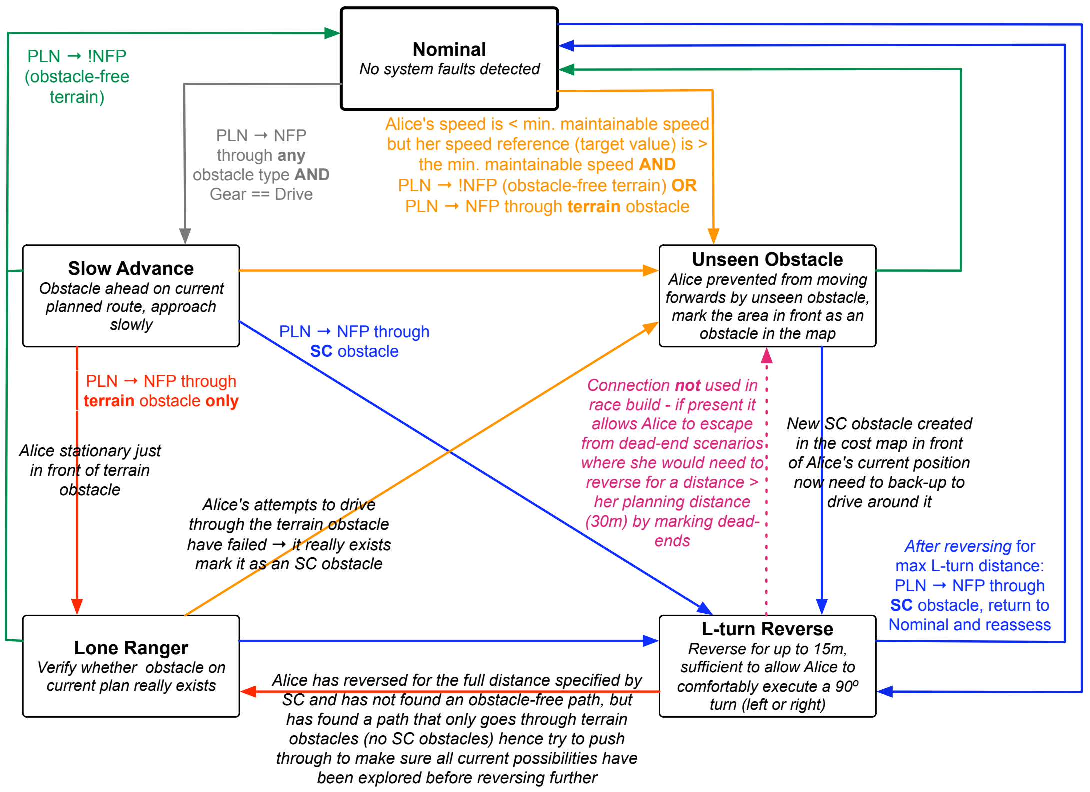
## 6.3 Strategies

The role of the strategies is to create a logical structure that can fulfill the primary and secondary objectives for SuperCon. Corresponding to the primary objective, to ensure that Alice always continues to make forward progress along the RDDF where possible, a list of possible scenarios (from experience and analysis) that could prevent such progress was produced. By definition, this list was coined as No Forward Progress (NFP) scenarios. Analysis of the list produced five distinct families of NFP scenarios, prototypical examples for each of which are shown in Figure 15.

Due to the limited perceptive and cognitive abilities of our system (as compared to a human) it is often not initially possible to distinguish between the five groups shown in Figure 15. Hence in order for SuperCon to be robust an integrated array of strategies that can identify and resolve any arbitrary scenario from any of the groups is required.

After extensive analysis of the responses that a human would take to attempt to resolve the prototypical examples (assumed to be at least on average the best response) and the information they used to do so, the NFP Response Cycle shown in Figure 16 was produced. The cycle consists of five strategies including the nominal strategy, which the system should remain in or revert to if no fault conditions are detected. The following strategies are used in the NFP Response Cycle:

**Nominal** No system faults/requests detected and the current plan does not pass through any obstacles. Verify that the system is correctly configured for nominal operation (gear = drive etc) and correct any exceptions found.



Nominal
No system faults detected
PLN → !NFP (obstacle-free terrain)
PLN → NFP through any obstacle type AND Gear == Drive
Alice's speed is < min. maintainable speed but her speed reference (target value) is > the min. maintainable speed AND PLN → !NFP (obstacle-free terrain) OR PLN → NFP through terrain obstacle
Slow Advance
Obstacle ahead on current planned route, approach slowly
Unseen Obstacle
Alice prevented from moving forwards by unseen obstacle, mark the area in front as an obstacle in the map
PLN → NFP through SC obstacle
PLN → NFP through terrain obstacle only
Alice stationary just in front of terrain obstacle
Connection not used in race build - if present it allows Alice to escape from dead-end scenarios where she would need to reverse for a distance > her planning distance (30m) by marking dead-ends
New SC obstacle created in the cost map in front of Alice's current position now need to back-up to drive around it
Alice's attempts to drive through the terrain obstacle have failed → it really exists mark it as an SC obstacle
After reversing for max L-turn distance: PLN → NFP through SC obstacle, return to Nominal and reassess
Lone Ranger
Verify whether obstacle on current plan really exists
L-turn Reverse
Reverse for up to 15m, sufficient to allow Alice to comfortably execute a 90° turn (left or right)
Alice has reversed for the full distance specified by SC and has not found an obstacle-free path, but has found a path that only goes through terrain obstacles (no SC obstacles) hence try to push through to make sure all current possibilities have been explored before reversing further

Figure 16: No-Forward-Progress (NFP) Response Cycle, showing the constituent strategies and purposes of each of the interconnections between them to produce the cycle response effect. Note that the general conditions under which a transition between strategies occurs (colored arrows) is described by the text of the same color as the arrow. Specific details of individual transitions are given in black text associated with the relevant arrow.

**Slow Advance** The planner's current plan now passes through an obstacle (terrain or SuperCon, including outside the RDDF corridor) when it did not pass through any obstacles previously (nominal strategy). Hence limit the max speed to ensure the sensors can accurately scan the terrain ahead, but allow Alice to continue along the current plan, whose speed profile will bring Alice to a stop in front of the obstacle if it is not cleared from the map by new sensor data.

**Lone Ranger** The planner's current plan passes through a terrain obstacle, which has not been cleared by subsequent sensor data, even after Alice has driven up to it. As Alice would always avoid any terrain obstacles if there was a way to do so (that did not pass through SuperCon obstacles), verify that the terrain obstacle really exists by attempting to push through it.

**Unseen Obstacle** Alice is stationary when it does not intend to be (TrajFollower's reference speed less than minimum maintainable speed), its engine is on and it is actively attempting to move forwards (positive accelerator command and zero brake command) yet an external (assumed) object in front of it is preventing it from doing so. Hence mark the terrain directly in front of its current position as a SuperCon obstacle, as it has been investigated and found to be intraversible.

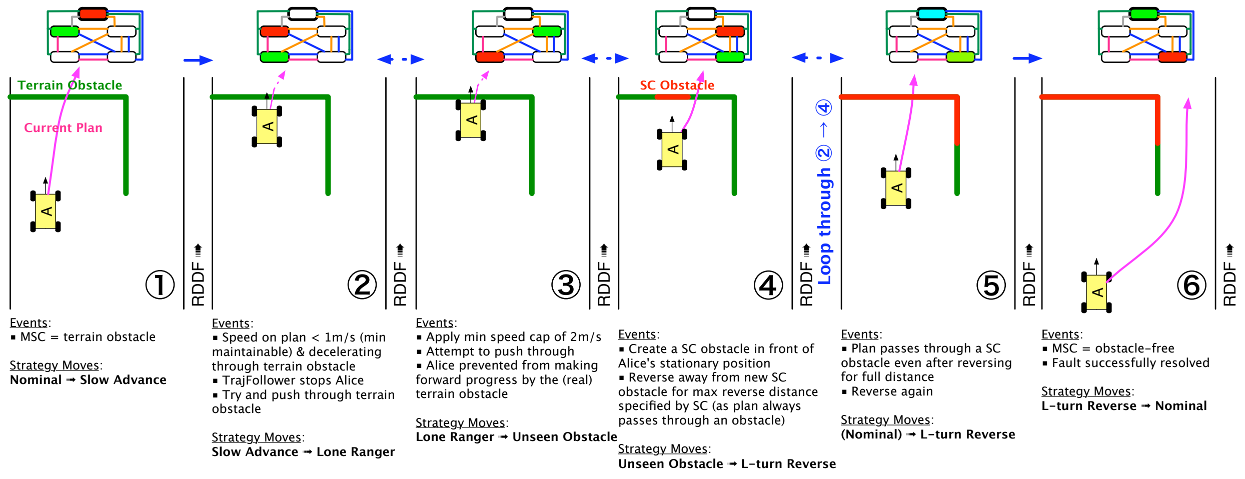
**L-turn Reverse** The planner's current plan passes through a SuperCon obstacle, these are avoided in preference of all other terrain where possible and indicate areas that have been verified as intraversible (or are outside the RDDF). Hence reverse along the previous path (trail of state data points) until the current plan passes through no obstacles, up to a maximum of 15 m (the distance required for Alice to perform a 90-degree left/right turn plus an error margin) per call of the L-turn reverse strategy. If the current plan passes through terrain obstacles after reversing for 15 m then go forwards and investigate them to verify their existence. If it still passes through SuperCon obstacles then recall L-turn Reverse and hence initiate another reversing action.

A significant benefit of the NFP response cycle design is that it only requires a very small amount of additional information to be computed on top of what is already available in the system (prior to SuperCon). The main piece of information used is the value of the speed limit in the lowest speed cell (evaluated over Alice's footprint) through which the current plan evaluated through the most recent version of the cost map speed-layer passes, referred to as the minimum speed cell (MSC). The current plan is defined as the plan currently being followed by the TrajFollower. The planner that Alice uses (as detailed in Section 5) is a non-linear optimizer. As a result, it is not necessary for SuperCon to know the location at which the minimum speed cell occurs, as the plan is optimal for the cost function it represents and the best option available in terms of its obstacle avoidance. There are three ranges (of values) in the speed layer of the cost map: no-obstacle, terrain obstacle and SuperCon obstacle outside the RDDF. The range that the current minimum speed cell value belongs to is used by SuperCon when determining what action (if any) to take. There is a fundamental difference between what is known about terrain and SuperCon obstacles. Terrain obstacles are identified by the sensor suite as having an elevation, and/or gradient outside of Alice's safe limits. SuperCon obstacles represent areas where Alice attempted to drive through (irrespective of whether they were identified as terrain obstacles) and was unable to do so. As such SuperCon obstacles are verified terrain obstacles. Note that all "obstacles" are theoretically intraversible by definition.

The process is analogous to a typical feedback controller, where the planner is the process whose output is the minimum speed cell value, SuperCon is the controller and the reference signal is the range of obstacle-free terrain values. The error signal then becomes non-zero when the current plan passes through an obstacle, which prompts SuperCon to take corrective action through the use of the NFP response cycle.

Figure 17 shows an example of the NFP response cycle enabling Alice to successfully navigate its way out of a dead end scenario, listing the actions taken by SuperCon at each stage, and indicating when each strategy is active. Stages 2–4 in the response are repeated until SuperCon obstacles have been placed as shown in stage 5 (assuming the terrain dead-end obstacle really exists). Note that the RDDF border on the right hand-side of Alice does not need to be verified, as outside the RDDF the terrain is equivalent to an SuperCon obstacle by default. A miniaturized version of the NFP response cycle is also shown for each stage, with the current strategy highlighted in red (darker shading), and the next in green (light shading). In case 5, blue (upper center state) is used to denote the most probable current strategy at the start of the stage).

Figure 18 depicts the evolution of SuperCon strategies that enabled the forward progress made during Alice's first NQE run.



**①**

Events:
- MSC = terrain obstacle

Strategy Moves:
**Nominal ➟ Slow Advance**

**②**

Events:
- Speed on plan < 1m/s (min maintainable) & decelerating through terrain obstacle
- TrajFollower stops Alice
- Try and push through terrain obstacle

Strategy Moves:
**Slow Advance ➟ Lone Ranger**

**③**

Events:
- Apply min speed cap of 2m/s
- Attempt to push through
- Alice prevented from making forward progress by the (real) terrain obstacle

Strategy Moves:
**Lone Ranger ➟ Unseen Obstacle**

**④**

Events:
- Create a SC obstacle in front of Alice's stationary position
- Reverse away from new SC obstacle for max reverse distance specified by SC (as plan always passes through an obstacle)

Strategy Moves:
**Unseen Obstacle ➟ L-turn Reverse**

**⑤**

Events:
- Plan passes through a SC obstacle even after reversing for full distance
- Reverse again

Strategy Moves:
**(Nominal) ➟ L-turn Reverse**

**⑥**

Events:
- MSC = obstacle-free
- Fault successfully resolved

Strategy Moves:
**L-turn Reverse ➟ Nominal**

Figure 17: Example of Alice resolving a dead-end scenario fault, showing the events and strategy transitions at each point.

## 7 Experimental Results

In the previous six sections we have described in detail Team Caltech's approach to designing its unmanned ground vehicle, Alice. Unfortunately, due to the nature of the Grand Challenge, very few controlled experiments were performed on the vehicle as a whole. Although many subsystems were individually tested to make sure they met certain specifications, the measure of success for the end-to-end system was a more qualitative one, focused on making sure that the vehicle could drive autonomously in a wide variety of situations including open desert, parking lots, rainy and wet conditions, dirt roads of varying degrees of roughness, rolling hills, winding roads and mountain passes. The following sections describe the nature and results of the testing of Alice leading up to the National Qualifying Event, for the National Qualifying Event itself, and in the Grand Challenge Event. Although the data presented is somewhat qualitative, we believe it makes clear Alice's capabilities (and weaknesses) as an autonomous vehicle.

### 7.1 Desert Testing

Team Caltech documented over 300 miles of fully autonomous desert driving with Alice from June 2005 to the National Qualifying Event in Fontana in September, all in the Mojave Desert. Figure 19 shows some of the RDDFs used during testing.

Approximately 33 of these miles were driven on the 2004 Grand Challenge race route during the week of June 13th, 2005. Alice traversed these miles with a testing team of four people inside, scrutinizing its performance and making software improvements and corrections. Over the course of three days, Alice suffered three flat tires including a debilitating crash into a short rocky wall that blew out the inside of its front left tire and split its rim into two pieces. This crash was determined to be caused primarily by a lack of accurate altitude estimates when cresting large hills. Along with several related bug fixes, an improved capability to estimate elevation was added to the state estimator.

The majority (approximately 169 miles) of autonomous operation for Alice took place in the

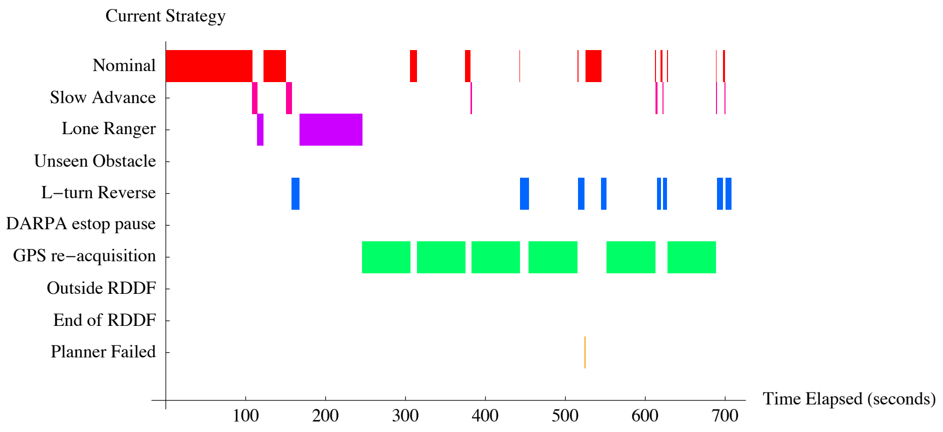


Figure 18: Strategies used during NQE Run #1. The GPS re-acquisition intervals are execution of the design requirement to safely stop the vehicle in the face of large corrections of state estimate errors.

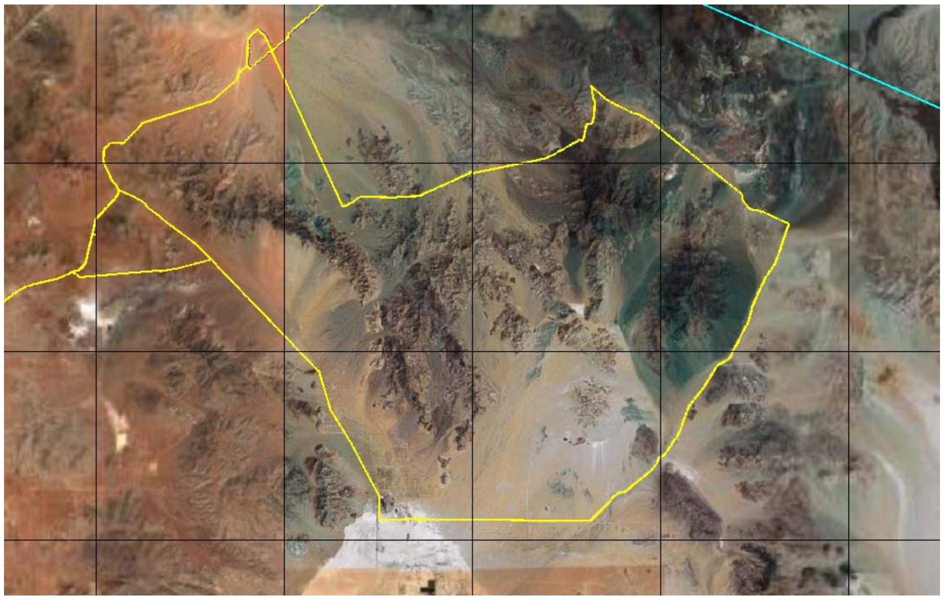


Figure 19: A compilation of several RDDFs used during testing within the Stoddard Wells area of the Mojave Desert. The grid lines are 10 km apart. The RDDF leaving the top of the figure is the 2004 race RDDF, and the line cutting across the top right corner is the boundary of the no-go zone given by DARPA. These RDDFs covered a wide variety of desert terrains, including dirt roads, rocky hills, dry lake beds, bumpy trails, smooth pavement, and mountain passes. During the majority of testing the RDDF speed limit over the entire RDDF was set to be unlimited, and the vehicle's speed was chosen automatically as it traveled.

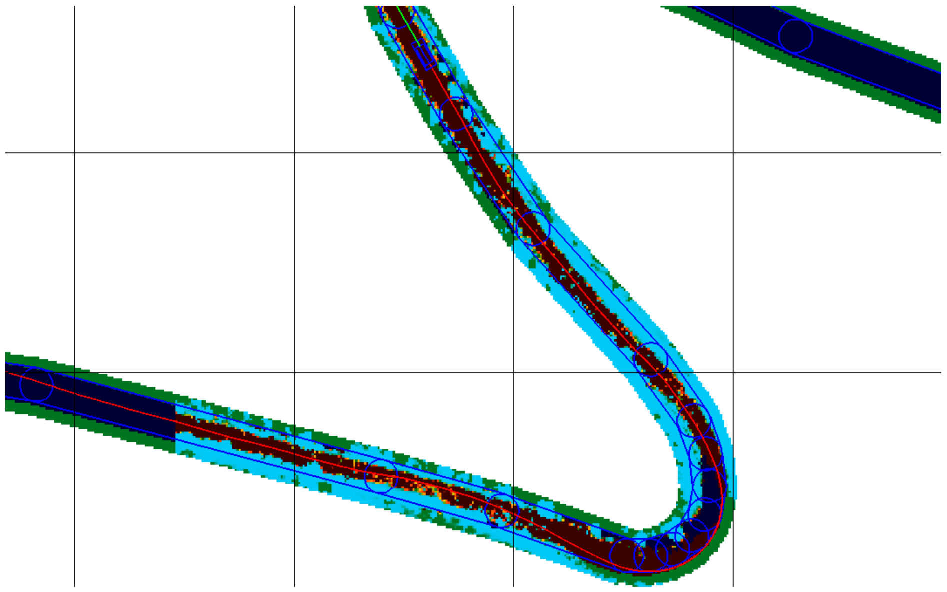


Figure 20: A sample speed limit map taken in Daggett Ridge during testing on the 2004 Grand Challenge course. For scale, the grid lines are 40m apart. The light-colored areas along the sides of the corridor are obstacles (cliff-faces taller than the vehicle, or berms about 0.5m high), the actual road is the darker area in the center, and the confetti-like coloring of some parts of the road indicates bumpier sections. Despite the narrowness of the corridor (around 3-4m wide) and the difficulty of the terrain, Alice (the rectangle near the top of the figure) was able to pick out a safe course (the line extending back down the figure and to the left) at an average speed of 4m/s.

two weeks leading into the National Qualifying Event. This operation included a full traversal of Daggett Ridge at 4 m/s average speed, and significant operation in hilly and mountainous terrain (see Figure 20). The top speed attained over all autonomous operations was 35 mph. The longest uninterrupted autonomous run was approximately 25 miles.

## 7.2 National Qualifying Event

As one of 43 teams at the California Speedway in Fontana, Alice successfully completed three of its four qualifying runs. Several of its runs were characterized by frequent stopping as a result of the performance of the state estimator under conditions of intermittent GPS. Specifically, under degraded GPS conditions its state estimate would drift considerably, partially due to miscalibration of IMU angular biases (especially yaw) and partially due to lack of odometry inputs to the state estimator. However, zero-speed corrections were applied to the Kalman filter when Alice was stopped, which served to correct errors in its state estimate due to drift quite well.

During Alice's first NQE run, zero-speed corrections were not applied in the state estimator. Accordingly, drift accumulating in the state estimator was not corrected adequately when Alice stopped. Travel through the man-made tunnel produced drift substantial enough for Alice's estimate to be outside the RDDF, which at the time resulted in a reverse action. Alice performed a series of reverse actions in this state, going outside the actual corridor, and resulting in DARPA pause and the end of its first run.

| Run | Gates | Obst | Time | Issues |
|---|---|---|---|---|
| 1 | 21/50 | 0/4 | DNF | State estimate problems after tunnel |
| 2 | 44/50 | 4/4 | 12:45 | |
| 3 | 49/50 | 5/5 | 16:21 | Multiple PAUSEs due to short in E-Stop wiring |
| 4 | 44/50 | 5/5 | 16:59 | Frequent stops due to state drift |

Figure 21: Alice on the National Qualifying Event course (left). Table of results from Alice's four runs at the NQE (right).

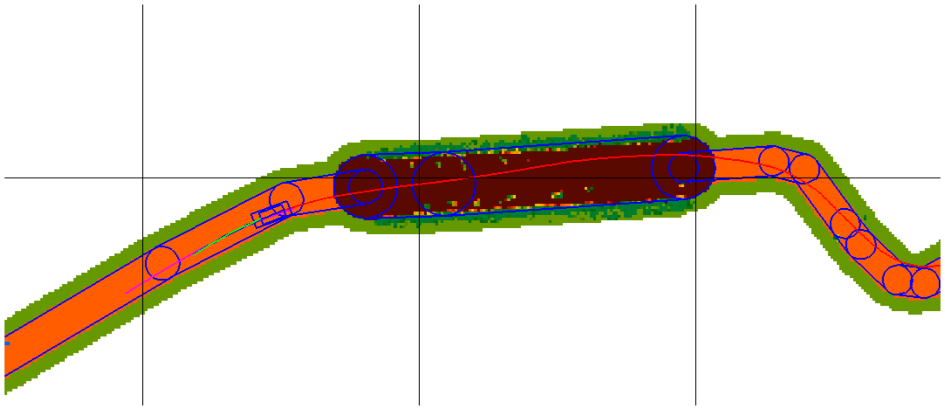


Figure 22: A sample speed limit map taken during the third NQE run. For scale, the grid-lines are 40 m apart. The solidly-colored areas in the center are the intended corridor, the lighter areas above and below the central section are lines of hay bales (around 0.5m tall), and the small spots near those hay bales are additional obstacles laid out by DARPA for Alice to avoid (such as wooden planks and tires). Alice (the rectangle to the left) was easily able to detect and avoid the obstacles (even though it likely could have traversed them easily)—the line extending from Alice to the right of the figure is the path that was followed. (Direction of travel is from right to left.)

Zero-speed corrections were added to the state estimator after Alice's first run, enabling it to successfully complete all subsequent runs, clearing all obstacles and 137 out of a total 150 gates. Completion times were slow for runs 2, 3 and 4 partially due to frequent stopping as a result of state estimator corrections. Figure 21 provides a summary of Alice's NQE run performances. Figure 22 shows a snapshot of Alice's map and followed paths during the third NQE run.

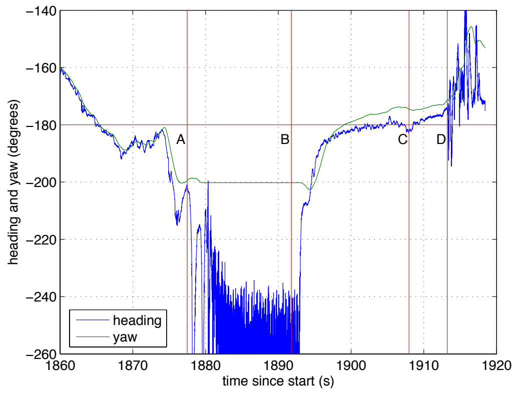
Alice was one of only eleven teams to complete at least three of four NQE runs. As a result of Alice's performance at the NQE, it was preselected as one of ten teams to qualify for a position on the starting line in Primm, Nevada.

## 7.3 Grand Challenge Event

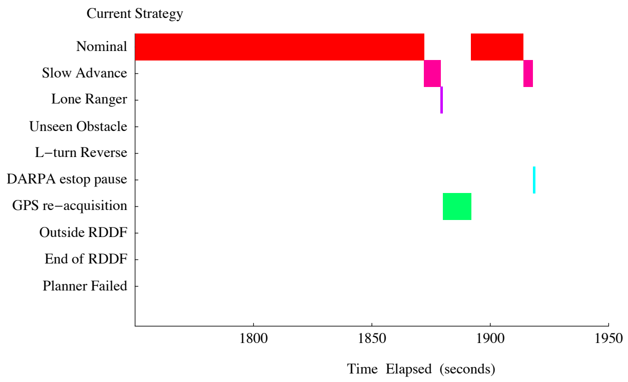
When Alice left the starting line on October 8th, 2005, all of its systems were functioning properly. However, a series of failures caused it to drive off course, topple a concrete barrier and disqualify itself from the race as a result. Although as mentioned above, the system we have described performed well over the course of hundreds of miles of testing in the desert prior to the Grand Challenge, we believe the pathological nature of this particular failure scenario demonstrates a few of the more important weaknesses of the system and exemplifies the need for further ongoing research. We will begin by providing a brief chronological timeline of the events of the race leading up to Alice's failure, followed by an analysis of what weaknesses contributed to the failure.

Alice's timeline of events in the Grand Challenge is as follows:

- Zero minutes into the race, Alice leaves the starting line with all systems functioning normally.
- Approximately four minutes into the race, two of its midrange LADARs enter an error mode from which they cannot recover, despite repeated attempts by the software to reset. Alice continues driving using its long and short-range sensors.
- Approximately 30 minutes into the race, Alice passes under a set of high-voltage power lines. Signals from the power lines interfere with its ability to receive GPS signals, and its state estimate begins to rely heavily on data from its Inertial Measurement Unit (IMU).
- Approximately 31 minutes into the race, Alice approaches a section of the course lined by concrete barriers. Because new GPS measurements are far from its current state estimate, the state estimator requests and is granted a stop from supervisory control to correct approximately 3 meters of state drift. This is done and the map is cleared to prevent blurred obstacles from remaining.
- GPS measurements report large signal errors and the state estimator consequently converges very slowly, mistakenly determining that the state has converged after a few seconds. With the state estimate in an unconverged state, Alice proceeds forward.
- A considerable eastward drift of the state estimate results from a low confidence placed on the GPS measurements. This causes the velocity vector and yaw angle to converge to values that are a few degrees away from their true values. Based on the placement of the north-south aligned row of K-rails in the map by the short-range LADAR (see Figure 24(a)), Alice's actual average yaw for the 5 or so seconds leading into the crash—between times C and D on Figure 23—appears to be about 8 degrees west of south ($-172$ degrees). For the same period, our average estimated yaw was about $-174$ degrees and our average heading (from $\dot{N}$, $\dot{E}$) was about $-178$ degrees. Roughly, as Alice drives south-southwest, its state estimate says it is driving due south, straight down the race corridor (Figure 24(a)).
- Alice's long-range sensors detect the concrete barriers and place them improperly in the map due to the error in state estimate. Alice's mid-range sensors are still in an error mode. Alice picks up speed and is now driving at 10–15 mph.
- At 32 minutes, because it is not driving where it thinks it is, Alice crashes into a concrete barrier. Its short-range sensors detect the barrier, but not until it is virtually on top of it (Figure 24(b) and Figure 25).
- Almost simultaneously, DARPA gives an E-Stop pause, the front right wheel collides with the barrier, the power steering gearbox is damaged, and the driving software detects this and executes its own pause also, independent of that from DARPA. The strong front bumper prevents Alice from suffering any major damage as it drives over the barrier.



(a)



(b)

Figure 23: (a) Alice's estimated heading and yaw, both measured clockwise from north, in its final moments of the GCE. Yaw is from the state estimator and heading is computed directly as the arctangent of the easting and northing speeds of the rear axle. The times at which the state estimator requests a vehicle pause (A), the map is cleared and pause is released (B), Alice speeds up after clearing a region of no data (C) and impact (D) are shown. Between A and B the direction of motion is noisy as expected as Alice is stopped. Between B and D the state estimate is converged around 180 degree yaw, which we know to be about 8 degrees off leading into the crash. (b) The supervisory controller mode (strategy) during the same period.

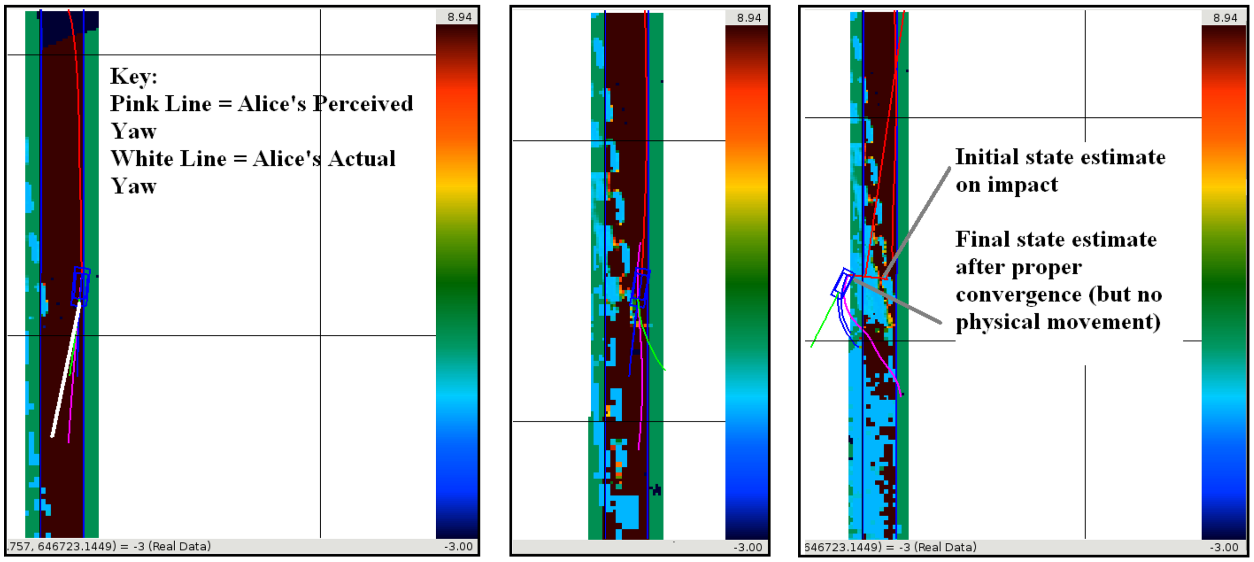
- Once Alice has come to a stop, the state-estimation software once again attempts to reconverge to get a more accurate state estimate—this time it corrects about 9.5 meters and converges close to the correct location, outside the race corridor. Alice is subsequently issued an E-Stop DISABLE (Figure 24(c)).

Figure 23(b) shows the SuperCon state during this final segment, including a Slow Advance through some spurious obstacles, a brief Lone Ranger push to clear them, the GPS reacquisition while stopped, and the final Slow Advance only after Alice has picked up speed and is near to crashing.

It is clear that while Alice's ultimate demise was rooted in its incorrect state estimates (due to poor GPS signals), other factors also contributed to its failure, or could conceivably have prevented it. These include the mid-range LADAR sensor failures, the lack of a system-level response to such failures, and the high speeds assigned to long range sensor data even in the face of state uncertainty. Additionally, in race configuration the forward facing bumper LADAR sensor was only used to assist in detection of the boundaries of the roads for the road following module. This could have helped in assigning appropriate speeds in the map for the row of K-rails.

## 8 Lessons Learned and Future Work

Even after many miles of desert testing and satisfactory performance at the NQE, Alice was not able to demonstrate its full capabilities in the race. Despite this failure, the full capabilities of the system provide an excellent testbed for future research and there are many open challenges that remain to be addressed. Accomplishments of the team include demonstration of a highly networked vehicle architecture with large amounts of sensor data, an optimization-based planner capable of


Key:
Pink Line = Alice's Perceived Yaw
White Line = Alice's Actual Yaw
8.94
-3.00
.757, 646723.1449) = -3 (Real Data)
Initial state estimate on impact
Final state estimate after proper convergence (but no physical movement)
546723.1449) = -3 (Real Data)

(a) (b) (c)

Figure 24: Alice's speed limit maps over the last few seconds of the race, with notation added. (For scale, the grid lines are 40m apart.) For all three diagrams, the center dark area is the road, the light vertically-oriented rectangular areas are the concrete barriers, the hollow rectangle is Alice (traveling downward on the page), and the light-colored line is its perceived yaw (direction of travel). The color-bar on the right indicates the speed limit that was assigned to a given cell, where brown is highest and blue is lowest. The leftmost diagram indicates Alice's expected yaw and Alice's actual yaw during the last few seconds of the race. The center diagram is Alice's speed limit map less than one second before it crashed, indicating the differing placement of the obstacles by the short and long-range sensors—indicated as two parallel lines of concrete barriers (light-colored rectangles). In fact, there was only a single line of them, and that line went directly north-south, not angled as Alice perceived. The rightmost diagram is Alice's final estimate of its location: accurate, but thirty seconds too late.

rapid computation of feasible trajectories that maximize speed along the course, and a supervisory control layer capable of aggressively maintaining forward motion in the presence of complex driving conditions.

A critical short term need is to improve the state estimation capability in the presence of noisy signals. While other teams did not appear to have problems with state estimation, the combination of Alice's reliance on accurate state estimates (for placing obstacles) and supervisory logic for maintaining fault tolerance made it susceptible to poor GPS quality. These problems did not surface during desert testing since most testing was done in the open desert, away from the types of electromagnetic interference experienced at the NQE and the race.

The mapping system was also partly to blame for Alice's "grand finale." One of the major shortcomings that contributed to the failure was that the algorithm did not factor in any measure of confidence in the state estimate when determining the locations of obstacles. This is a major source of error because the state estimate must be used to place the sensor measurements, which are taken in a local coordinate frame, into the map, which is in a global coordinate frame. Thus, when the state estimate is very inaccurate (as was the case in the situation described above) the

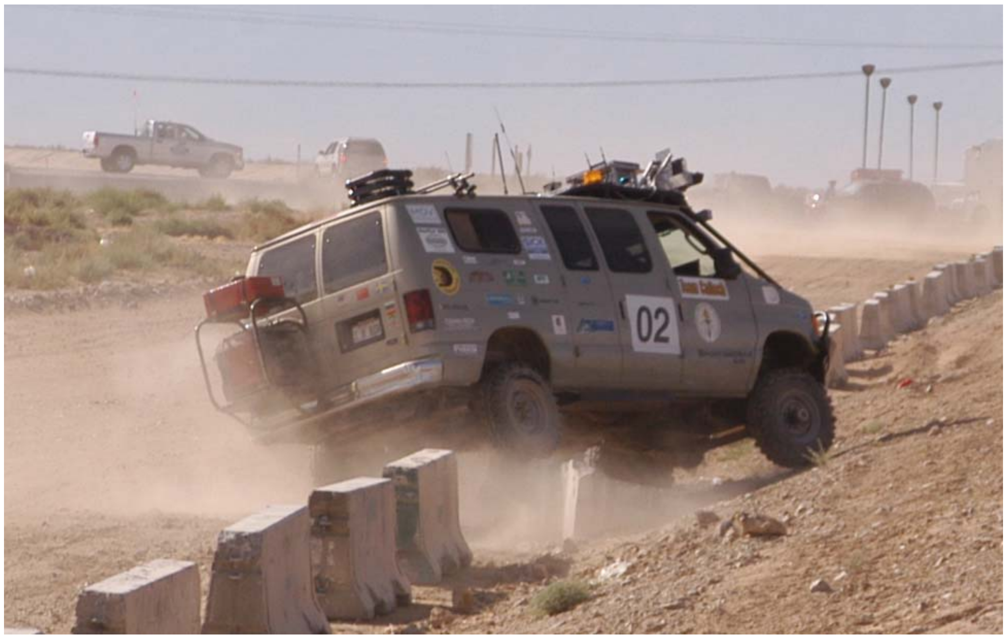


Figure 25: Alice as it "clears" a concrete barrier during the GCE.

resulting map is also inaccurate. If the algorithm had made a probabilistic estimate of the location of obstacles instead of a hard estimate, then it is likely that the resulting map would have made the concrete barriers seem much larger than they were. (This is because the low confidence in the state estimate would have propagated down to become a low confidence in their placement in the map, which would in turn have caused them to occupy a greater portion of the map.) Had this been the case, the vehicle might have avoided a collision.

The second shortcoming of the mapping system was that it did not factor in any measure of the confidence in the sensor measurements when determining the speed of a given cell in the map. In other words, although one sensor was able to outweigh another when placing data, the map still did not reflect which sensor took a given measurement, and how accurate (or inaccurate) that sensor was when it made that measurement. As a result, the map did not reflect that two of the midrange sensors (which usually performed the bulk of the mapping) were inoperative. Instead, the map was filled with semi-inaccurate data from the long-range sensor, which was not differentiated from any other data that might have been present (such as more accurate data from the midrange sensors, had they been functioning correctly). As a result, the path-planning algorithm commanded a speed of between 10 and 15 mph, which prevented the vehicle from reacting in time when the obstacles were more accurately placed by the short-range sensor.

Team Caltech's experience indicates the need for future research in constructing efficient, timely, reliable and actionable models of the world from a rich sensor suite. Of particular interest are ideal sensor coverage algorithms and proper high-level response to loss of particular regions of sensor coverage, as well as formal analysis of the interplay between sensor data longevity (for more complete maps) and degradation of the state estimate (which tends to produce misregistered maps).

Ongoing work on Alice includes demonstration of more sophisticated sensor fusion techniques (Gillula, 2006), a networked architecture of combining sensors for state sensors without the need for offline calibration (Leibs, 2006), and model-based road following using LADAR data (Cremean, 2006;

Cremean and Murray, 2006). Alice will also be used during the summer of 2006 for additional SURF projects in autonomous systems.

**Acknowledgments**

Team Caltech received generous support from a number of organizations and individuals. We would like to thank our primary sponsors—the California Institute of Technology, Northrop Grumman Electronic Systems, and Sportsmobile West, Inc—for their generous support throughout the project. In addition, we received financial support and donations from many sponsors, including AEK Technology, Aluminess, Barry Controls, Blanchard Associates, CommTech, Evolution Robotics, Ford, Horsepower Engineering, IBM, ITU Ventures, Johnson Controls, Lonseal, Mohr Davidow Ventures, Motion Picture Marine, Navcom, Northrop Grumman Space Technology, NovAtel, OmniSTAR, SICK, Santa Anita Park, Space X, Systems Technology Incorporated, Woodhead and Wright Tool.

The team also benefited from the wisdom and experience of many individuals, including Elliott Andrews, Ole Balling, Lynn Burgess, Mani Chandy, Mike Elder, Marionne Epalle, Alex Fax, Tony and Sandie Fender, Will Heltsley, Melany Hunt, John Kalmbach, Maria Koeper, Larry Matthies, Duane McRuer, Kenny Meyer, Bob Rasmussen, Doug Robinson, Jessica Stephens, Claire Tomlin and Dave van Gogh. We would also like to thank Steven Koonin, former Provost at Caltech, for his early support of the project.

Finally, this paper could not have been written without the hard work of the rest of the members Team Caltech, including Sterling Algermisson, Lyudmil Antnov (VPI), Henry Barnor, Ryan Cable, Eric Cady, Kevin Cossel, Adam Craig, Joe Donovan, Kevin Duncklee, Scott Elder, Ryan Farmer, Josh Feingold, Ken Fisher, Laura Fishman, Marie Göransson (Lund), Ike Gremmer, Rob Grogan, Lily Gruenke, Barrett Heyneman, Sue Ann Hong, Haomiao Huang, Cedric Jeanty, Erik Johannesson (Lund), Tony Kelman, Gustov Lindström (Lund), Kyle Miller, Lisa Nyström (Lund), Pietro Perona, Sam Pfister, Ben Pickett, Harmon Pollack, Tami Reyda, David Rosen, Ben Sexson, Alan Somers, Dan Soudek, Phillip Welch, Chris Wetzel, Lusann Yang (Princeton) and Jason Yosinski.

## References

Amir, Y., Danilov, C., Miskin-Amir, M., Schultz, J., and Stanton, J. (2004). The spread toolkit: Architecture and performance. Technical Report CNDS-2004-1, Johns Hopkins University.

Cremean, L. B. (2006). *System architectures and environment modeling for high-speed autonomous navigation*. PhD thesis, California Institute of Technology, Mechanical Engineering.

Cremean, L. B. and Murray, R. M. (2006). Model-based estimation of off-highway road geometry using single-axis ladar and inertial sensing. In *Proc. IEEE International Conference on Robotics and Automation*.

Defense Advanced Research Projects Agency (2005). Grand Challenge 2005 team technical papers. `http://www.darpa.mil/grandchallenge05/techpapers.html`.

Dickmanns, E. D. (2004). Dynamic vision-based intelligence. *AI Magazine*, 25(2):10–30.

Gillula, J. H. (2006). A probabilistic framework for real-time mapping on an unmanned ground vehicle. Senior thesis, California Institute of Technology.

Goldberg, S. B., Maimone, M. W., and Matthies, L. (2002). Stereo vision and rover navigation software for planetary exploration. In *Proceedings of the 2002 IEEE Aerospace Conference*, Big Sky, MT.

Kogan, D. (2005). Realtime path planning through optimization methods. Master's thesis, California Institute of Technology.

Kogan, D. (2006). Optimization-based navigation for the DARPA Grand Challenge. *Conference on Decision and Control (CDC)*. Submitted.

Leibs, J. (2006). Learning positional and conformational state in multi- sensor networks. Senior thesis, California Institute of Technology.

McRuer, D. T. (1975). Measurement of driver-vehicle multiloop response properties with a single disturbance input. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-5(5).

Milam, M. B. (2003). *Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems*. PhD thesis, California Institute of Technology.

Murray *et al.*, R. M. (2003). Online control customization via optimization-based control. In Samad, T. and Balas, G., editors, *Software-Enabled Control: Information Technology for Dynamical Systems*. IEEE Press.

Rasmussen, C. and Korah, T. (2005). On-vehicle and aerial texture analysis for vision-based desert road following. *cvpr*, 3:66.

Rasmussen, C. E. (2006). A hybrid vision + LADAR rural road follower. In *Proceedings of the 2006 International Conference on Robotics and Automation*, Orlando, FL.

Rasmussen, R. D. (2001). Goal-based fault tolerance for space systems using the Mission Data System. In *Proceedings of the 2002 IEEE Aerospace Conference*, Big Sky, MT.

Rosenblatt, J. (1998). Utility fusion: Map-based planning in a behavior-based system. In *Field and Service Robotics*, pages 411–418. Springer-Verlag.

Urmson, C. (2005). Navigation regimes for off-road autonomy. Technical Report CMU-RI-TR-05-23, Robotics Institute, Carnegie Mellon University.

Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J. P., Gowdy, J., Gutierrez, A., Harbaugh, S., Johnson-Roberson, M., Kato, H., Koon, P. L., Peterson, K., Smith, B. K., Spiker, S., Tryzelaar, E., and Whittaker, W. (2004). High speed navigation of unrehearsed terrain: Red team technology for grand challenge. Technical Report TR-04-37, Robotics Institute, Carnegie Mellon University.

Vermeulen, S., Marples, R., Robbins, D., Houser, C., and Alexandratos, J. (2005). Gentoo linux x86 handbook. Available online at `http://www.gentoo.org/doc/en/handbook/handbook-x86.xml`.