DARPA Urban Challenge
Princeton University
Technical Paper

Team Leader:
Professor Alain L. Kornhauser
Department of Operations Research and Financial Engineering
Princeton University
E-407 Engineering Quad
Olden St.
Princeton, NJ 08544

Authors:

Anand Atreya ’07, Bryan Cattle ’07, Safiyy Momen ‘07

Brendan Collins ’08, Alex Downey ’08, Gordon Franken ’08, Jon Glass ’08, Zach Glass ’08, Josh Herbach ’08, Andrew Saxe ‘08

Issa Ashwash ’09, Chris Baldassano ’09, Will Hu ’09, Umar Javed ’09, Jonathan Mayer ’09

David Benjamin ’10, Lindsay Gorman ’10, Derrick Yu ’10

Executive Summary

Princeton University's entry into the DARPA Urban Challenge is a modified 2005 Ford Escape Hybrid SUV, donated by Ford Motor Company. Utilized is a Perception, Cognition, Actuation, Substrate and Environment cycle implemented in a multi-threaded environment of Microsoft Robotics Studio and the Microsoft Windows Server 2003 operating system. Sensing is accomplished primarily by computer vision – both monocular and stereo, with supplemental information from RADAR. A behavior-based navigation scheme generates desired space-time paths which are followed by a set of decoupled longitudinal and crosstrack error controllers. Actuation of the vehicle was accomplished by reverse engineering existing drive-by-wire systems. Design decisions throughout were influenced by the principles of simplicity and cost-effectiveness, held in tension with the goal of completing all of the requirements of the Urban Challenge.

Submitted in partial fulfillment of the requirements for the 2007 DARPA Urban Challenge on June 1, 2007.

Table of Contents

1. Introduction

1.1 Overview

The Princeton University team consists of 20 undergraduate students at Princeton University. Several members of the current team also contributed to Princeton's entry in the DARPA 2005 Grand Challenge, *Prospect Eleven*. Under the direction of Professor Alain Kornhauser, our team is committed to enriching the educational experience of our members by allowing them to contribute to research in the field of autonomous ground vehicles.

1.2 Approach

Our goal is to meet all of the requirements set forth by the Urban Challenge and complete 60 miles of autonomous urban driving in less than 6 hours at the Urban Final Event. Our vehicle platform is a 2005 Ford Escape Hybrid SUV (Figure 1) that was donated by Ford Motor Company. The Hybrid configuration provides substantial range and on-board electrical power, while the SUV body style leaves ample room for our equipment and five passengers. All modifications were performed so as to maintain human operability.

Figure 1: Ford Escape Hybrid SUV.

1.2.1 Issues that drive design choices

Throughout the process of preparing our vehicle for competing in the Urban Challenge, we have relied on the principles of simplicity, cost-effectiveness and reliability to inform and direct all design and implementation decisions.

Enabling a car to operate itself in a dynamic and unpredictable urban environment is inherently a complicated problem. However, that does not necessarily imply the need to gather and process large amounts of data. It is easy to make a problem too difficult to solve; we challenge ourselves to find simple, elegant solutions. In doing so, we intend to accomplish the goals of the Challenge without investing unnecessary resources and time on processing superfluous input.

Our limited budget requires us to search for low-cost solutions by necessity. However, we embrace this constraint, since it forces us to look for innovative system implementations that side-step an otherwise strong temptation to acquire an excessive amount of technology.

In the 2005 Grand Challenge, a small software bug generated a memory leak that caused our vehicle to stall after 9.5 miles. This experience taught us the importance of reliability. For this Challenge we have kept a watchful eye over all of our systems using substantial testing and data logging to ensure that their design and implementation demonstrates quantitative rigor and reliability.

Other guiding principles emerge when examining the requirements for a specific section of our system, but these three prevail throughout. Together, they direct the evolution of our design. We intend to achieve our stated goal while adhering to our commitments to reliability, simplicity and cost-effectiveness.

1.2.2 Implementation

Our approach to the problem is organized according to a continuous five-part cycle we call PCASE: Perception, Cognition, Actuation, Substrate and Environment. These parts interact as shown in Figure 2. The steps of PCAE encapsulate the information flow through the robotic system. The robot receives information about its changing surroundings in the Perception step, computes desired actions in the Cognition step, and carries out these actions during the Actuation step. These motions change the robot's Environment, which also changes independently, so the whole cycle must begin again with Perception.

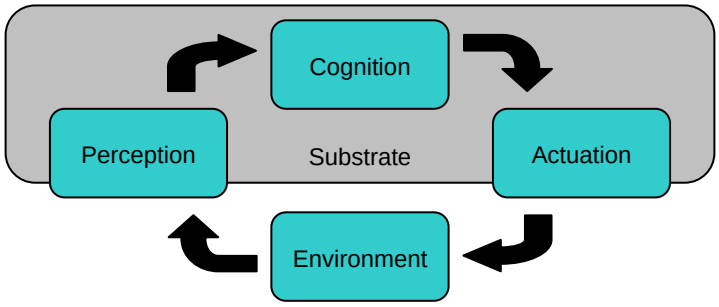


Figure 2: The PCASE System Diagram.

The Urban Challenge Environment is not under our control. Therefore, our implementation focuses on the three remaining steps of the cycle: Perception, Cognition and Actuation (also referred to as Sense, Plan, Act). However, these three steps are sufficient for autonomous operation only in theory. To implement this cyclic behavior in reality, we have identified a necessary fourth process, referred to as Substrate. Our substrate encompasses all of the hardware and software architecture that allows all modules to work asynchronously across the entire system.

Implementation of the remaining three steps divides each into two parts. Perception is split into Environment Sensing and State Sensing - each of which includes its own sensors and processing algorithms. Cognition is a two-step process in which data from multiple sensors is first fused into a unified representation of the surrounding world, and then a desired set of actions is computed. Finally, Actuation consists of Controls, the software controllers designed to implement desired actions, and Actuators, the physical devices that cause the vehicle to move in relation to its surroundings. Our specific design solutions for each of these parts are discussed in the following section.

2. Design & Analysis

2.1. Substrate

Substrate provides the foundation for our system. Therefore, the entire system is only as robust as the substrate implementation, so reliability was a key factor considered when designing our substrate. In addition, we aimed to develop substrate that would facilitate the testing process.

2.1.1 Mounts and Enclosures

Custom housings were built for all external sensors to protect them from weather conditions and allow for easy repositioning if necessary. The housings are constructed of PVC or cast acrylic and sealed with weatherproofing material along their seams. Access ports for

cables are protected with rubber fittings. Each camera housing provides space for lens filters to be attached. Red or orange colored filters are used to increase contrast for the black and white cameras, while clear UV filters are used on all cameras to protect the lenses.

The housings are designed so that they can be easily adapted to attach to the car using either magnetic feet or permanent mountings. The option of using magnetic mounting allows for easy testing of optimal configuration and placement of the sensors.

Our computers are rack-mounted in a steel rack in the rear of the vehicle, shown in Figure 3a. The rack sits on shock-absorbent feet, which significantly dampen shocks and vibrations that could otherwise be fatal to hard drives or other sensitive electronics equipment.

Additional electronics are mounted in a thermally shielded box underneath the hood of the vehicle. The box provides convenient access to the vehicle actuation circuits and data acquisition cards for testing and debugging, while insulating them from the heat of the engine compartment.

2.1.2 Data Acquisition

In order to control the vehicle and auxiliary systems, we use two Labjack UE9 Ethernet Data Acquisition devices, shown in Figure 3b. Each device is encased in a rugged plastic housing to resist surface damage. All terminals are internally protected against shorting and current backflow. The units are also able to withstand temperatures between -40 to +85 °C, making them suitable for operation in the engine compartment. The Labjack devices use the TCP/IP protocol, allowing any of our multiple computers to communicate with either of the devices through the vehicle LAN. Each Labjack has an adequate number of configurable digital and analog I/O ports. Several advanced acquisition features, such as hardware timers and counters have been used to great advantage by our controls and sensing systems.

(a)  (b) 

Figure 3: Computer rack in rear of vehicle (a). Under-hood electronics box (b).

2.1.3 Computer Systems

The vehicle uses a bank of custom-built 1U rack-mount servers to perform high level data processing and send low level actuation commands. All computers are identically equipped with Intel Core 2 Duo processors, 2GB of RAM and an 80GB SATA hard drive running Microsoft Windows Server 2003. Our standard rack allows computers to be easily added as our processing needs increase or swapped out in case of failure. A gigabit Ethernet network switch provides high-bandwidth, low latency communication between computers and other TCP/IP-enabled devices.

Our algorithms take advantage of the servers' dual core technology, leading to faster processing time and an overall increase in system performance. In addition, the Core 2 Duo

processors consume considerably less power than other processor models, which reduces electrical load on the vehicle's power system.

Since all of our code is written in C# and C++ using the Microsoft Visual Studio environment, Windows is the natural choice for our operating system. We chose Windows Server 2003 because it is the most stable of the Windows family and maintains driver compatibility. We can use commercially available drivers with the sensors in use on the vehicle, thereby mitigating compatibility issues during testing. The 2003 Server platform also allows multiple users to modify software on the same machine, via Remote Desktop, resulting in faster code development and a more streamlined testing process.

2.1.4 Software Architecture

Our software architecture is designed to achieve a balance between reliability, flexibility, and speed. This section outlines the various components of our software architecture.

We are using the recently-released Microsoft Robotics Studio (MSRS) as the underlying software framework for the vehicle. Robotics systems in MSRS are composed of independent services which communicate in real time. This modularity allows us to easily write and test individual sections of the vehicle's software. MSRS also provides reliable methods for synchronization of and communication between services using Microsoft's Coordination and Concurrency Runtime (CCR) and Decentralized Software Services Protocol (DSSP).

The services in our vehicle are decomposed into three regimes: Sensing, Planning, and Acting. The individual modules are shown in Figure 4, below, where the arrows depict messaging between services.

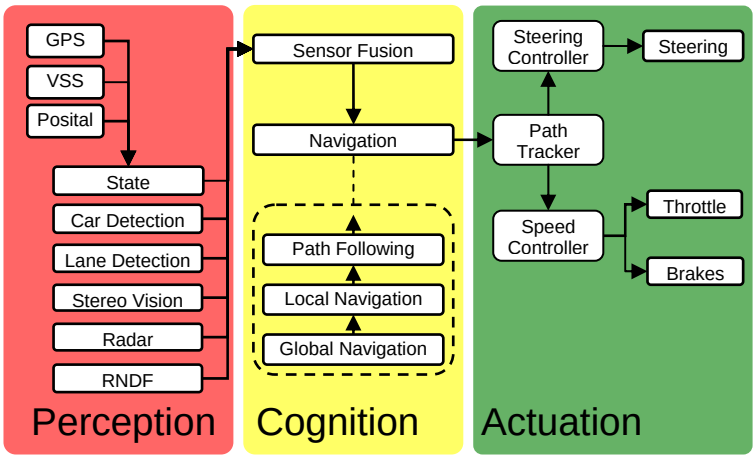


Figure 4: Block Diagram of MSRS Services.

All of our software development is done in C# using the Visual Studio. The sole exception is our vision processing code, which is written in C++ because of performance benefits. Visual Studio offers many tools for developing applications including a powerful debugger, advanced documentation support and integrated source control. C# is a very high level language that offers good performance and an extensive library. Of the languages available for working with Microsoft Robotics Studio, C# is the most powerful.

2.2 Perception

Perception includes both the physical sensors and the software algorithms that process the raw data to extract meaningful descriptors. We have further divided perception into two distinct

tasks: Sensing features of the vehicle's surrounding environment and sensing the vehicle's motion. Both parts are discussed in this section.

2.2.1 Environment Sensing

In designing our environment sensing capabilities, we considered two main factors: cost and performance. Sensors were selected for their ability to meet our specified sensing needs as well as their minimal impact on our budget. We plan to offset reduction in sensor coverage by implementing sensing algorithms that extract as much information as possible from the raw sensor data. Efficient algorithms not only provide information at a faster rate, they allow us to use fewer computers – further reducing cost.

2.2.1.1 Environment Sensors

We use a combination of cameras and RADAR units to perceive the environment, arranged as depicted in Figure 5. This configuration was distilled from the requirements for situational awareness set forth in the *Technical Evaluation Criteria*[1] as summarized in Table 1, below.

Category	**Ability**	**T.E.C. Ref.**
Road Markings		
	Detect painted stop lines	A.8
	Perceive painted lane markings	A.4, C.5, C.6
	Perceive other lane edges in the absence of markings	A.4, C.5
Obstacles		
	Detect when road is completely blocked	C.4
	Perceive obstacles of significance at adequate distances and field of view to allow for evasion	A.7, C.2, D.9
Vehicles		
	Detect distance to vehicle ahead	A.9, D.3, D.8
	Detect distances to vehicles in adjacent lanes	A.9
	Detect vehicles and obstacles behind the vehicle	A.9, A.12, D.3, D.8
Traffic		
	Detect oncoming traffic in opposing travel lanes	A.10, D.4
	Detect all vehicles at or approaching an intersection	B.2
	Detect oncoming perpendicular traffic at distances adequate enough for a safe merge	D.2

Table 1: Overview of Sensor Requirements.

Further evaluation of the rules gave us more detailed guidelines for sensor ability. The need to perceive a 10-second gap in traffic (D.2 & D.4), combined with the 30mph max speed limit implies that the maximum sensor range needed is just over 130m. We were also concerned with having substantial sensor coverage in the 10-20m range for navigating intersections properly.

Forward sensing is accomplished with one monocular camera, three stereo cameras, and one long range RADAR. A Point Grey *Flea2* camera provides color images, which are used for both lane and car detection. The *Flea2* has a field of view of 60 degrees; a one-axis gimbal mount provides full 180 degree coverage. A Point Grey *XB3* stereo camera allows for switching between a 12cm and 24cm baseline for short or medium range obstacle detection. The *XB3* has a horizontal field of view of 50 degrees and maximum detection range of a distance of 60m. Two

[1] DARPA, 2007

Point Grey Bumblebees, oriented at 50 degree angles to the right and left of the vehicle, provide short range obstacle detection and overlapping coverage at intersections. The Bumblebees have a useful detection range of 30m. A background subtraction algorithm will be used detect moving vehicles at greater distances. A forward facing Delphi ACC-3 Forewarn RADAR system provides position and velocity measurements of vehicles and obstacles directly ahead. The system was developed for adaptive cruise control and is specifically tuned for vehicle detection. This device gives range accuracy of +/- 1 m up to a range of 150 m, which meets our full specifications. It has a horizontal field of view of only 15°, so in scenarios such as turns where an obstacle ahead in the lane may lie outside the field of view, supplemental information from stereo vision will be relied upon. The ACC-3 is able to detect visually occluded objects. This ability will prove valuable in obstacle fields and intersection queues.

For rear sensing, our vehicle has two monocular cameras and one short range RADAR. The two Point Grey *Firefly-MV* cameras mounted on the roof over the rear view mirrors and are used for auxiliary lane and car detection before lane change maneuvers. The *Firefly-MV* has a 60 degree field of view. A Delphi Forewarn Dual-Beam Backup Radar system is primarily intended for obstacle detection when the vehicle is backing up. It covers a 6m x 2.75m area behind the vehicle. This low-cost system, employed in many consumer vehicles, is adequate for safe backup operation, since obstacles outside its field of view will either be seen by the rear-facing cameras or remain too far away to worry about at the low speeds involved in reverse maneuvers.

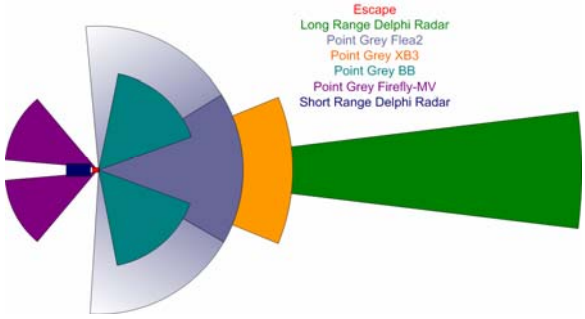


Figure 5: Top-view of sensor coverage

2.2.1.2 Sensing Algorithms

Data from our cameras is processed by three main algorithms: road marking detection, stereo obstacle detection, and car detection. The RADAR units do a substantial amount of processing and filtering on-board, so their output is fed directly into the sensor fusion step.

Lane Detection

Our lane detection algorithm functions in three phases. Initially a pixel-by-pixel search is performed to look for a set of features that identify lanes. Subsequently, these features are fused into a single score for each pixel. Finally, lanes are estimated based on this fused image. The results of each step are shown in Figure 6.

Features are extracted from a raw image (a) using a white filter (b), a yellow filter (c), and a lane marking width filter(d). The two color filters detect pixels which correspond to the probable colors of lane markings, while the width filter is a modified edge filter which responds to edges of the correct width, compensated by vertical location in the image. These features are then fused into a heat map (e) by requiring that pixels be either white or yellow, and have responded to the width filter.

Lane detection is then accomplished by examination of the heat map (f). First, RANSAC is run through several hundred iterations, looking for a parabola which passes through many lane pixels. Since a parabola does not always perfectly describe the geometry of a lane, a greedy search is then performed over the pixels, starting at the parabola's estimate of position. Future work for lane detection includes incorporating more features, including texture, and using more sophisticated methods to combine the features into our fused image.

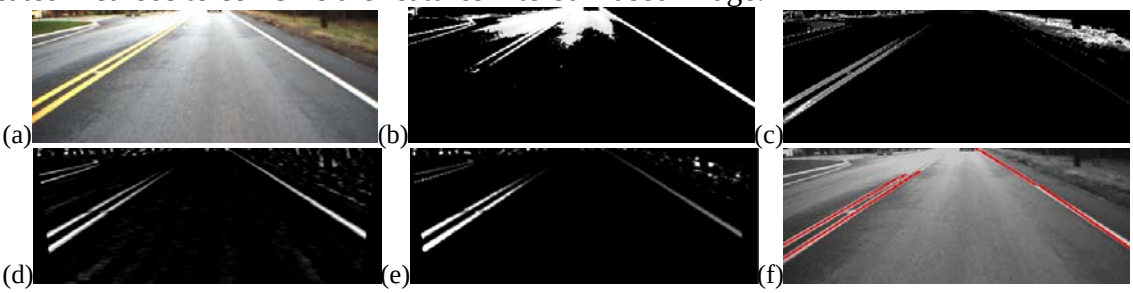


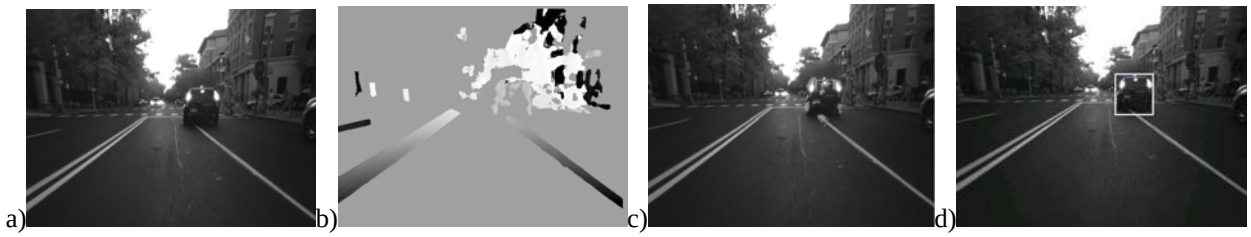
Figure 6: Stages of Lane Detection (a-f).

Our vehicle relies primarily on stereo vision to detect obstacles and other vehicles in and around our lane of travel. The obstacle detection process consists of three steps: depth mapping, point classification, and segmentation, which are shown in Figure 7.

The scene in front (a) is captured simultaneously by two cameras spaced a known distance apart. Using libraries developed by Point Grey, a depth map (b) is computed from the disparity between the two images. Due to the lack of texture in urban environments (as compared to off-road settings) it is more difficult to create a detailed depth map; so improvements such as automatic gain control have been implemented to address this issue.

In order to determine which of the points in the depth map should be classified as obstacles to avoid (c), a C++ implementation of an algorithm proposed by R. Manduchi[2] was developed. An inverted cone region around every point is searched for points at similar depth. If any are found, then these two points are considered obstacles since they are standing roughly vertically. This algorithm has proven effective, both in terms of quality of points selected and processing power required.

To allow for effective navigation and tracking, points must be grouped into discrete obstacles (d). The previous classification step provides some information about what points belong to the same obstacle, and these small groups of points are merged together, if doing so does not significantly change the total volume of the obstacle. The approximate 3D bounding box of the detected obstacles is then passed to sensor fusion to allow for tracking and filtering, and is eventually passed to navigation to allow the vehicle to make informed decisions.



Figures 7: Stages of obstacle detection (a-d).

Car Detection

[2] Manduchi, 2005

Our car detection algorithm uses artificial intelligence techniques to learn a patch-based model of cars. Following Torralba et. al.[3], we begin with a labeled dataset of car images, and use clustering to obtain a dictionary of patches. Our patches are described as Scale Invariant Feature Points[4]. We classify images according to the presence of absence of these patches, using Boosting[5] to select the most salient patches and weight them accordingly. To classify a new image, we simply find SIFT features, match them into our dictionary, and then use the boosted classifier to arrive at a decision.

2.2.2 State Sensing

The most important guiding principle we considered when designing our state sensing systems was the ability to minimize mean square of error in measurements. However, our low-cost approach and limited budget precluded the acquisition of expensive high-precision equipment, so we endeavored to develop innovative ways to extract meaningful data from a minimal set of sensors.

2.2.2.1 State Sensors

Large dependency on reliable and accurate GPS information is a poor assumption in an urban environment. Therefore, it was a poor allocation of our resources to purchase a precision GPS system. A Trimble Ag114, combined with the free OmniStar subscription available to Urban Challenge teams form a low-cost GPS solution that provides sub-meter accuracy at 10Hz.

For an estimation of heading in low-speed or stationary conditions, we are implementing a system consisting of two inexpensive GPS receivers, longitudinally spaced two meters apart on a roof-mounted boom. Heading can be calculated from the position differential between the two receivers. This differential technique improves accuracy, because errors from the transmission of GPS satellite signals through the atmosphere are nearly equal for closely-spaced receivers and therefore cancel when the signals are compared.

Although high-precision inertial sensors are desirable for vehicle motion estimation, we have developed a suitable and more cost-affective approach by taking advantage of existing sensors on our vehicle. In particular, the Vehicle Speed Signal (VSS) is a pulse-width modulated waveform provided by the anti-lock braking system. Using the advanced features of our Labjack data acquisition cards, we are able to parse this signal into a near-instantaneous measurement of vehicle speed.

The only vehicle data that was not readily available was the angle of the front wheels. To measure this, we use an optical encoder mounted on the steering column. Steering angle is converted to wheel angle based on vehicle geometry specifications.

2.2.2.2 State Estimation

An accurate and reliable representation of the vehicle's position and trajectory is crucial for success in navigating an urban environment. As such, our state estimation is designed to minimize error in light of the fact that our low-cost approach provides only for preexisting or imprecise sensors.

---

[3] Torralba, 2004
[4] Lowe, 2004
[5] Freund, 1997

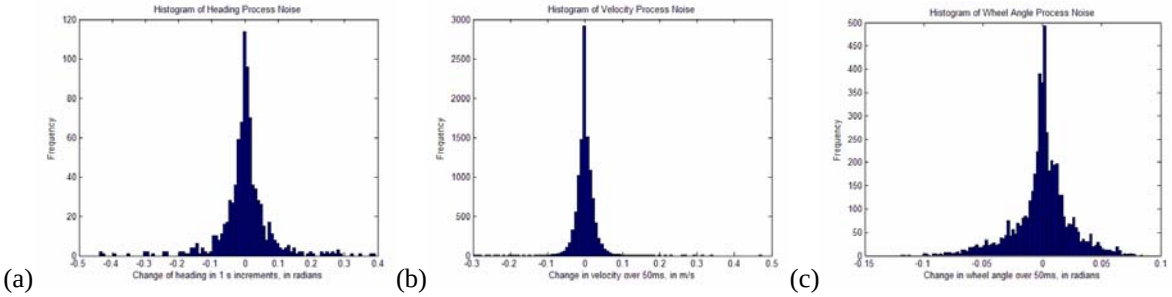
We established two output frames: global and local. In this way, we can isolate situations in which we explicitly rely on GPS positioning and reduce the need for a costly high-accuracy GPS device.

Both frames are updated through the use of an Extended Kalman Filter (EKF) as described by Welch[6]. Each filter maintains a 5 x 1 column vector corresponding to the state of the vehicle, which is defined by northing, easting, heading, speed and wheel angle. The EKF prediction cycle uses an advanced vehicle dynamics model[7] empirically tuned to our vehicle and associated process noise. We receive northing, easting, heading, and velocity measurements from GPS, an additional velocity measurement from the VSS, and a wheel angle measurement from the steering encoder.

In the global frame, the state vector is updated by the EKF upon receiving input from any of the sensors. The local frame, however, relies only on measurement updates from the VSS and steering encoder.

The global frame is used exclusively for interaction with the RNDF. The local frame is subject to drift, so it is useful in situations where only local differences are computed. For instance, when tracking a desired path, it is important to avoid GPS failure modes, provided that a particular path is only followed for a short period of time. It is also used in obstacle or lane tracking, where it is essential to provide reliable motion estimates with associated error.

In order to maintain accuracy and reliability in our state computation, we make two assumptions about the use of the EKF. First, that our prediction equations, derived from the dynamics model, are sufficiently linearizable; and second, that our noise covariances follow a Gaussian distribution. A discussion of the first assumption by St. Pierre and Gingras[8] shows that it is valid for a dynamical system describing vehicular motion. Figure 8, below, demonstrates the validity of the second assumption.



(a) (b) (c)

Figure 8: Gaussian Covariances used for State Estimation of heading (a), velocity (b) and wheel angle (c).

2.3 Cognition

Our software framework divides the Cognition task into two separate processes: sensor fusion and navigation. Sensor Fusion examines data from the vehicle's sensors and probabilistically generates a unified and descriptive representation of the surrounding environment. The Navigation routine then takes this description and analyzes it to determine a desired space-time path through the world.

[6] Welch, 2006
[7] Franken, 2007
[8] St. Pierre, 2004

2.3.1 Sensor Fusion

Sensor fusion produces a description of the perceived world that is both complete and accurate enough for the navigation routine. All of our sensors have measurement error, which we determine from specifications or empirical testing. The best representation of our perceived world comes from properly modeling and accounting for the uncertainty associated with each sensor, which was the primary objective when designing the sensor fusion algorithms.

2.3.1.1 Lane Filtering and Lane Fusion

Lane filtering and fusion algorithms bridge the gap between lane detection and navigation. All potential lanes perceived locally must be filtered for false-positives and combined with knowledge of the RNDF to situate the car globally.

Filtering is performed on a frame-by-frame basis. A Kalman update produces estimates of lane locations in the current frame, based on relative state-estimation and lane locations in the previous frame. These estimates are compared to all observed lane markings through a variety of heuristics. All possible matches between white and yellow lane markings are ranked by these heuristics, and the most probable lanes are extracted.

Lanes are represented as a series of points on the ground plane connected by line segments. Each control point is filtered with the nearest point in the other lane, using a weighted average. Points far from the car are assumed to have more error than closer ones. Centerlines are extracted based on paired lane markings or offsets from single lane markings. The most probable centerlines are passed along to navigation.

The lane fusion step must situate the locally-extracted centerline location within the global RNDF. A bias between GPS coordinates and the RNDF is estimated based on checkpoints and stop lines. Using this information, RNDF lanes are associated with vision-based centerlines. Lane centerlines are not based on RNDF GPS points, as DARPA rules state that RNDF waypoints will be sparse. The resulting description of lanes around the vehicle incorporates both global waypoint information and local lane position information.

At the start of the course, the locally detected and filtered lane is matched with the RNDF based on GPS. A particle filter is maintained for the discrete set of hypotheses which represent the possible RNDF lane locations. Samples of the particle filter are redistributed in each frame after computing the probability of each hypothesis – that is, the probability of being in each lane – according to various indicators, such as lane color, from lane filtering. Consistent local lane observations cause the filter to converge on a sharp distribution around the correct RNDF lane hypothesis. A transition model is applied to the filter during lane changes and intersection-crossings, such that the distribution over hypotheses changes to reflect these actions. In this way, the lane fusion algorithm maintains a probability distribution over all RNDF lane locations. The most likely global estimate of lane location and the filtered local centerline points are passed forward from sensor fusion for global and local navigation, respectively.

2.3.1.2 Obstacle Tracking

Our experience from the previous Challenge taught us the importance of tracking obstacles in the time domain. The greater complexity of an urban environment makes this essential. We implement multiple-model tracking by classifying objects perceived by the stereo system into three categories: stationary, moving, and erratic. Each obstacle is tracked according to all three of these models, and then assigned to the most likely category. Stationary obstacles refer those whose position is not recognized to vary significantly over time; moving obstacles

refer to those whose motion is expected to follow a precise course, most often a lane; and erratic obstacles refer to those whose motion we cannot predict for any significant period of time. It is assumed that such obstacles travel along our best estimate of their velocity.

As a component of sensor fusion, obstacle tracking receives the position of perceived obstacles from stereo cameras, and their velocities from radar. It uses these measurements in the measurement update cycles of three separate Extended Kalman Filters corresponding to the three classes of obstacles, in order that each obstacle may be tracked as a stationary, moving and erratic obstacle. A computation of the Mahalanobis distance between measurements and Kalman states yields a probabilistic classification and expected position of the obstacle.

2.3.2 Navigation

We believe the majority of the Urban Challenge will require only two basic behaviors: lane following and maintaining a constant velocity or following distance. This observation led us to two overarching principles for our navigation scheme: efficiency and simplicity. The urban environment is highly structured; most paths planned through it will require little deviation from the behaviors above and observe a sequential pattern. Consequently, we have chosen to implement a behavior-based expert system. Attempting a cost map approach would have required tuning numerous constants and been computationally intensive. A reactive navigation scheme, such as the one employed by Princeton in the 2005 Grand Challenge, would complicate the task of following traffic rules and enforcing specific behaviors.

The decision to use position-based controllers, as discussed later in Section 2.4.2, greatly simplifies all navigation tasks. Planning is only required to generate a desired space-time path. Lane following is accomplished by propagating points at a constant velocity along the lane centerline. Similarly, vehicle following propagates points at a constant distance behind the lead vehicle. Stopping at a stop line requires a smoothly decelerating path that holds the vehicle's position constant at the stop line. The intuitive nature of the code for these and other behaviors allows for rapid development and debugging. Combined with our simulation systems, we are able to implement and test new features and behaviors rapidly.

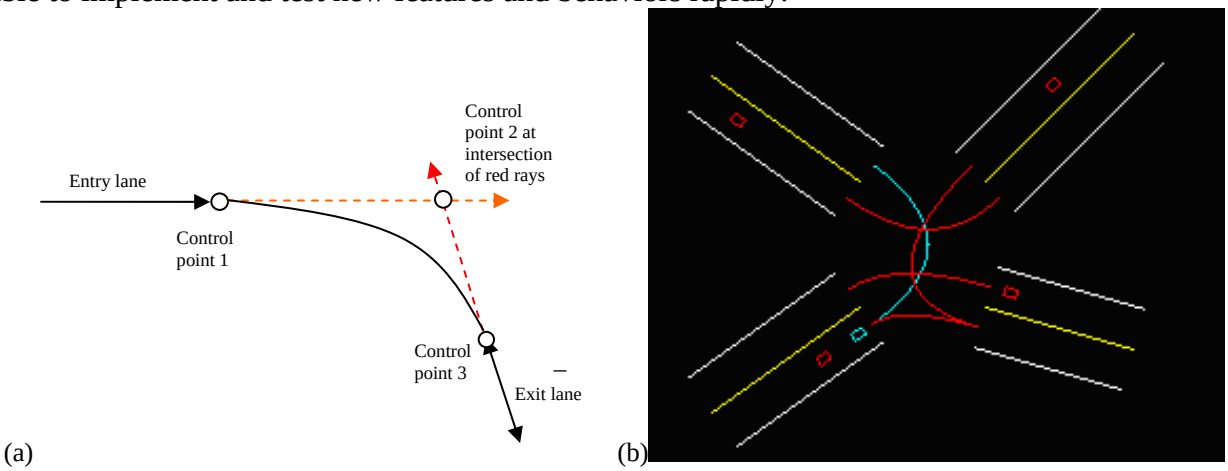


Figure 9: (a) Trajectory generation with Bezier splines. (b) Paths generated by precedence simulator.

An intersection behavior handles queuing and precedence. To form a smooth path to follow through the intersection, a three control point Bezier curve is used. The first and last control points are taken to be the ending and starting points of the entry and exit lanes respectively. At the first and last control points, the derivative of the curve is pointed at the middle control point. Thus, to ensure a continuous derivative path, this middle control point

must lie at the intersection of the lines running through the starting and ending segments, as illustrated in Figure 9a. In order to test the trajectory generation and intersection precedence logic an intersection simulator, Figure 9b, was created.

Our system transitions from the behavior-based expert controller to a rudimentary cost map scheme within parking zones. In consideration of the uncertain nature of parking lots, where predicting obstacle motion is difficult, we believe the best solution is to travel at low speed and focus primarily on obstacle avoidance. Once near a parking spot, an expert system guides the car into the spot and then reverses out.

To accomplish planning through the route network, we perform a reverse Dijkstra graph search using a binary heap. Edge length is determined by travel time between vertices (lane start and end points), and the reverse search provides the shortest path to the desired mission checkpoint. If the checkpoint is in a parking zone, routes are calculated to all entrances to the zone, and the shortest one is used. The entire process from receiving an RNDF to completing the graph search takes approximately 2 seconds with our current implementation, and re-running the graph search for a new mission checkpoint takes approximately $1/10^{th}$ of a second.

Road blockages are handled first by the local navigation algorithm. If navigating around the blockage is impossible, the appropriate edges of the graph are severed and a new graph search is performed. Severed edges are reconnected according to a linear time decay to account for the potential removal of obstructions, though preference is still given to routes containing no known obstructions. If a route cannot be planned given the current set of severed edges, edges are reconnected in the order they were removed until the graph search is successful.

2.4 Actuation

The actuation step is responsible for implementing a desired set of actions for the vehicle to take, so its key principles are reliability and minimizing delay. It is extremely important that these low-level controllers be reliable, as frequent breakdowns can seriously impede the development of high-level behaviors. Fast actuation prevents controls systems from becoming unstable, allowing faster and more accurate responses.

2.4.1 Vehicle Actuators

When designing the vehicle actuation systems, we choose to interface directly with the vehicle's native drive-by-wire systems wherever possible. This allows us to control the vehicle with a minimum of physical alterations, saving us initial development time. We expect the stock electronic actuators to exhibit substantially less delay custom-built ones and we are ensured of their reliability because they have been extensively tested by Ford Motor Co. and are in use on thousands of production vehicles.

Steering, brakes and throttle were all interfaced with electronically. Each system was reverse engineered to find the primary control sensor. Simulating the signal produced by that sensor enables our own control of the drive-by-wire system. Redundancy in these sensor signal lines reduces the chance of failure. In all cases, the speed of actuation not only exceeds that of a human operator, but also that of any actuator we would have designed and built for the same task.

A DC gear-motor mounted to the transmission housing is attached to the transmission through a pair of linkage arms. A hobby motor controller provides a variable voltage to the gear-motor, and the stock transmission position sensor is used for feedback. A gear-motor was chosen over a servomotor because of the high torque required for reliable actuation.

A mechanically and electronically fail-safe emergency stop system has been installed in the vehicle, in compliance with Competition Rules. It consists of a pneumatic cylinder positioned above the brake pedal, a pneumatic solenoid, a compressed air tank, and control circuitry. As long as a voltage is supplied, the cylinder remains unfired (brake pedal up). When the control line is grounded, the cylinder immediately fires, rapidly pressing down the pedal. The vehicle ignition is simultaneously shut off. This active-off configuration ensures fail-safe operation. The control line can be grounded by either of two externally-mounted latching buttons, or a "DISABLE" signal from the DARPA-supplied E-Stop system ("PAUSE" functionality is implemented in software). This system takes advantage of the vehicle's mechanically redundant braking, so even if the Electro-Hydraulic Braking (EHB) system fails, emergency-stop actuation will bring the vehicle to a halt. Pneumatics are a natural choice for this task because the compressed air tank acts as a large potential energy reservoir; even in the case of electrical failure, the cylinder will fire and the brake pedal will be depressed.

2.4.2 Controls

We specifically designed our vehicle controllers to minimize position error to a desired path. This approach is appropriate because it provides adherence to many of the competition requirements,[9] including: Passing over each checkpoint (A.3), stopping at precise locations (A.8 & C.3), maintaining following distances (A.9, B.3 & B.4), and executing precise trajectories (A.4, A.10, A.11, A.12 & D.9).

There are two situations for which guaranteeing position is not advantageous. The first is adhering to speed limits (A.5). We address this by layering our longitudinal controller on top of a speed controller. The second difficulty is maintaining accurate position control during GPS outages (C.6). Reliance on a local reference frame, as discussed in Section 2.2.2, mediates this difficulty.

2.4.2.1 Path tracking Controller

Paths are specified as a desired position over time, $\vec{r}(t)$. Our path tracking controller minimizes two error metrics as shown in Figure 10a. The first is the crosstrack error *e*(*t*), defined as the signed distance from the closest point on the path to the point $\vec{p}(t)$ between the car's front wheels,

$$|e(t)| = \min_n \|\vec{r}(n) - \vec{p}(t)\|.$$

The second is the longitudinal error *l*(*t*), defined as the signed arc length from the closest point on the path $\vec{r}(n)$ to the desired point on the path for the current time,

$$l(t) = \int_n^t \|\vec{r}'(s)\| ds .$$

Hoffmann et al. show that the crosstrack error metric in a kinematic car model evolves according to

$$\dot{e}(t) = v(t)\sin(\psi(t) - \delta(t)),$$

where v(t) is the vehicle speed, $\psi(t)$ is the car's heading relative to the trajectory, and $\delta(t)$ is the wheel angle with respect to the vehicle[10]. Similarly, the derivative of the longitudinal error is

[9] DARPA, 2007
[10] Hoffmann, 2007

$$\dot{l}(t) = -v(t)\cos(\psi(t) - \delta(t)) + \left\|\vec{r}'(t)\right\|.$$

The control law uses the steering control law given in Hoffmann[11], and adds a velocity control law:

$$\delta(t) = \psi(t) + \arctan\frac{k_e e(t)}{v(t)}$$

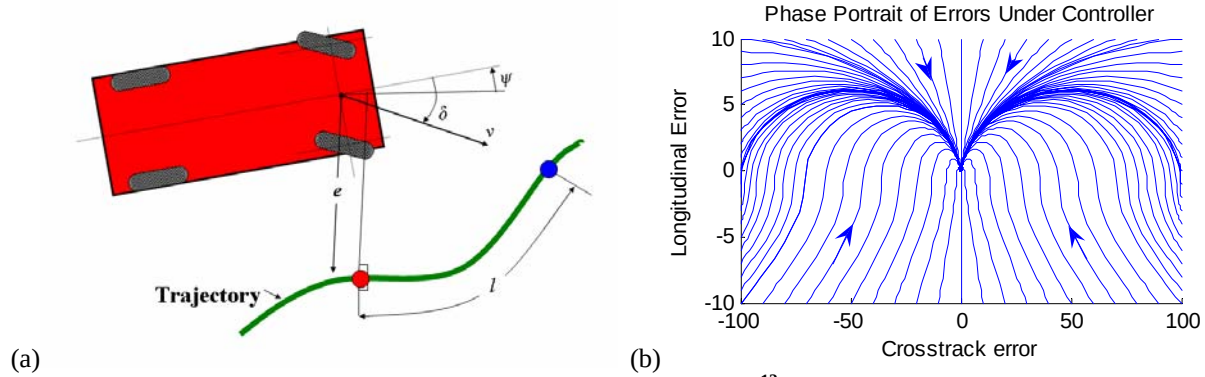
$$v(t) = k_l l(t) + \left\|\vec{r}'(t)\right\|$$

Substituting the control laws into the dynamics gives:

$$\dot{e}(t) = -(k_l l(t) + \left\|\vec{r}'(t)\right\|)\sin\arctan\left(\frac{k_e e(t)}{k_l l(t) + \left\|\vec{r}'(t)\right\|}\right) = \frac{-k_e e(t)}{\sqrt{1 + \left(\frac{k_e e(t)}{k_l l(t) + \left\|\vec{r}'(t)\right\|}\right)^2}}$$

$$\dot{l}(t) = -(k_l l(t) + \left\|\vec{r}'(t)\right\|)\cos\arctan\left(\frac{k_e e(t)}{k_l l(t) + \left\|\vec{r}'(t)\right\|}\right) + \left\|\vec{r}'(t)\right\| = \frac{-k_l l(t) - \left\|\vec{r}'(t)\right\|}{\sqrt{1 + \left(\frac{k_e e(t)}{k_l l(t) + \left\|\vec{r}'(t)\right\|}\right)^2}} + \left\|\vec{r}'(t)\right\|$$

Solving for steady state yields one fixed point at $e(t)=0$ and $l(t)=0$. The linearized state dynamics matrix is $\begin{bmatrix} -k_e & 0 \\ 0 & -k_l \end{bmatrix}$. The Eigenvalues, $-k_e$ and $-k_l$ are both negative so the system is locally stable. A phase portrait for k_e=1, kl=1, $\left\|\vec{r}'(t)\right\| = 1$, is given in Figure 10b. This analysis is approximate because it assumes a simplified car model, no steering or velocity constraints, instantaneous velocity control, and a constant path velocity. However, it has been found to be stable on the competition vehicle as discussed in results.



(a) (b)

Figure 10: (a) Crosstrack and longitudinal error definitions[12] (b) Phase portrait of convergence

2.4.2.2 Low Level Controllers

Longitudinal control is implemented by sending the desired speed from the longitudinal error control law $v(t) = k_l l(t) + \left\|\vec{r}'(t)\right\|$ to a speed controller that manipulates the throttle and brakes. To design the speed controller the overall system was linearized by empirically finding

[11] Hoffmann, 2007

[12] This diagram is adapted from Hoffmann07

the vehicle's acceleration as a function of control input. Then a PI controller was synthesized using classical methods.

Data on the throttle and brake responses was taken and fitted to expected vehicle dynamics. Figure 11, below, shows that brakes induce a linear deceleration, while throttle fits a standard exponential approach model, as predicted by vehicle dynamics.

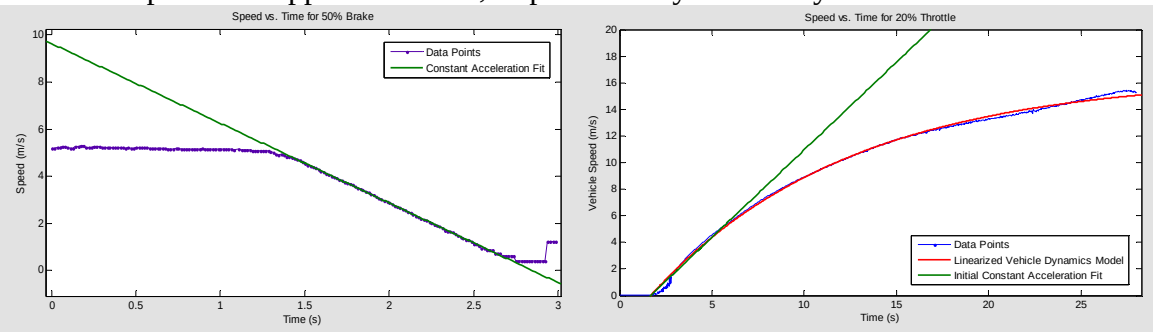


Figure 11: Sample brake (l) and throttle (r) responses.

These responses were analyzed to map control input to acceleration, shown in Figure 12.

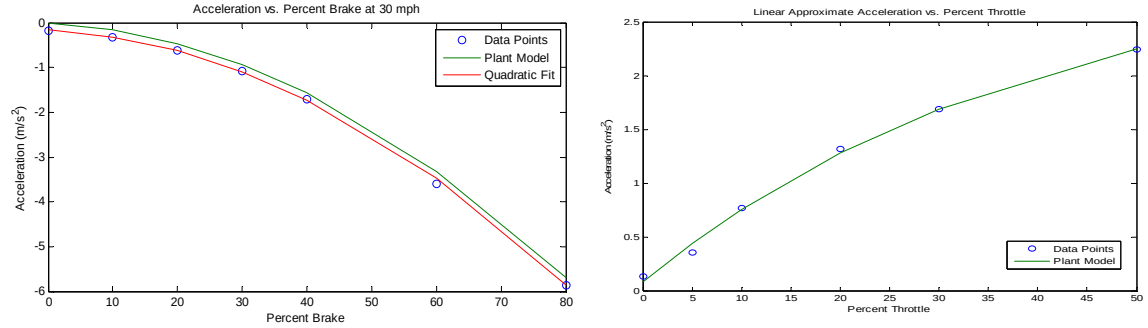


Figure 12: Sample fits for brakes (l) and throttle (r).

The inverses of these curves are used to linearize the overall system, and actuator and sensor delays are modeled with the Padé approximant to a 32 millisecond time delay. A critically damped PI controller was chosen because overshoot is undesirable in an environment with speed limits.

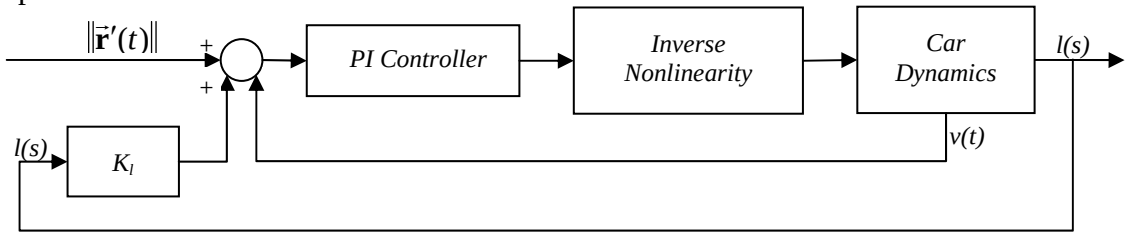


Figure 13: Block Diagram of Speed Controller.

The full system is shown in Figure 13. The instantaneous speed of the path at the current time, $\|\vec{r}'(t)\|$, is fed to the speed control as an open loop input. This causes the vehicle to track the desired trajectory provided that the longitudinal error starts at zero. To correct for disturbances, a term proportional to the longitudinal error, $k_l l(t)$, is also supplied to the speed controller, increasing the desired speed if the longitudinal position is too low and decreasing the desired speed if the longitudinal position is too high. Implementing longitudinal control using speed control as a subcomponent allows for easily enforcing speed limits, and controlling

convergence to the path from large errors by limiting speeds which otherwise would cause the car to skid and invalidate the dynamics model.

Lateral control is accomplished by providing the desired steering angle from the crosstrack error controller $\delta(t) = \psi(t) + \arctan\left(\frac{k_e e(t)}{v(t)}\right)$ to a PI steering controller.

3. Results and Performance

3.1 Testing

This section provides an overview of the various frameworks we use to test our system, from fully autonomous navigation to individual component evaluation.

3.1.1 Facilities

We are fortunate to have several facilities available for testing. Our garage contains necessary equipment to bench-test electronic systems, and the adjacent parking lot is used for testing systems that do not require fully autonomous vehicle operation. We have access to local practice fields which, when not in use, provide us with a large, open area to test our control systems.

We perform autonomous testing of our vehicle at an isolated section of Princeton University's Forrestal campus. The testing area, consisting of elements required for the Video Submission and Site Visit, is isolated during testing to ensure safety and has been marked with road markings consistent with the California Department of Transportation's design standards.

Collection of data for internal testing of software algorithms has included streets, roads and highways in the vicinity of Princeton, NJ. We collect data using the vehicle under human control.

3.1.2 Actuation

Testing of the vehicle actuators began as soon as the electronic interfaces were implemented. The goals of testing were to quantify the speed of actuation and determine reliability. GUI software was written to allow a user to send precise commands to individual actuators. Our logging system captured relevant data. Throttle, brakes and steering were given full-range simulated signals to determine their delay in actuation. The transmission actuation time was measured after the implementation of a simple controller, and the emergency-stop actuator timed with the car stopped. For stress testing, software was written that allows an operator to control the actuators with a wireless gamepad. This was used to test the reliability of the actuators and our interface to them over longer times and more diverse conditions.

3.1.3 Simulation

The development of a simulation environment is integral to our navigation testing strategy. Real-world tests consume a considerable amount of time and team resources. In addition, environmental variation from run to run increases the difficulty of testing specific behaviors in isolation. Consequently, we have developed a simulator that allows developers to test production vehicle code on a standard laptop.

Our simulation engine takes full advantage of the Microsoft Robotics Studio framework by emulating Sensor Fusion and Actuation output while retaining their production interfaces. As

a result, transitioning code from simulation to the autonomous vehicle does not require recompilation. The simulation engine models vehicle motion using experimentally gathered vehicle dynamics and control system responses, allowing system constants to remain largely unchanged between real-world and simulated trial runs.

The simulator features a 3D GUI, shown in Figure 14, which enables easy viewing of vehicle behavior from numerous angles. A virtual dashboard provides immediate control system feedback, and the planned path is displayed ahead of the vehicle. The crosstrack and longitudinal control points currently pursued by our position-based controller are displayed as well. A provided RNDF is parsed and displayed on the ground plane for reference, and interpolated to simulate lane, stop line, and parking spot detection. Stationary and moving obstacles can be easily added to the environment.

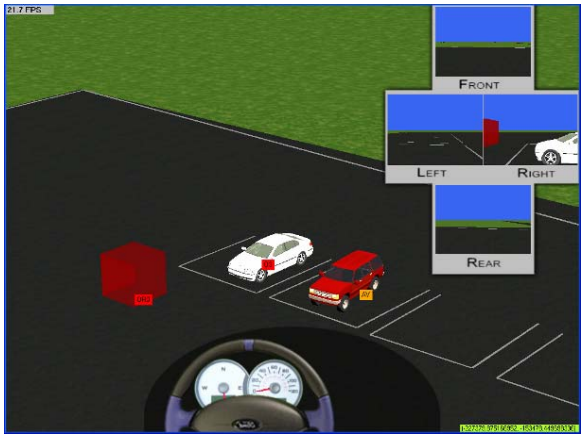


Figure 14: Screenshot of the Simulator

3.1.4 Data Logging

The ability to log data from our tests provides the capability to analyze the vehicle's performance and search for possible bugs in code. Every service automatically logs relevant data while it is running. A single Robotics Studio service is responsible for all data logging, allowing for a standardized logging API and storing of data in a consistent fashion. A Microsoft SQL server database allows us to store data for convenient access and post-processing. During the course of testing in a one month period, over 20GB of data were recorded.

3.1.5 Perception

Perception systems are tested both on logged data, and in live testing. Over the past several months, we have logged over 10GB of test images on roads. These images form the basis of our testing procedures for obstacle detection, lane detection, and car detection. Additionally, live user interfaces allow for a developer to inspect the output of an algorithm in real time.

3.2 Results

In this final section, we examine our systems and analyze their performance to determine if they meet our original specifications. We also identify current weaknesses in our system and propose solutions to implement as we continue developing.

3.2.1 Substrate

The Microsoft Robotics Studio CCR and DSSP allow for extremely efficient communication among services. A benchmark test was performed, where two services bounced 10,000 messages between one another. Across different service nodes, get-based and subscription-based messages averaged 2,974 messages per second and 7,057 messages per second respectively. When services run on the same node, the transmission rates rise to 164,880 and 294,118 messages per second respectively. This implies that message-based communication

in MSRS has extremely low delay and minimal overhead. An estimation of our maximum requirements - 30 services running at 60Hz, each sending/receiving 5 messages per cycle – is only 9000 messages per second. A total of five nodes, one on each computer, means that most traffic will be intra-nodal. We therefore find it extremely unlikely that we will come close to exceeding maximum transmission rates.

3.2.2 Perception

3.2.2.1 Obstacle Detection

The obstacle detection system was tested by placing a variety of objects at specific distances from the car and recording the results, allowing for analysis of both range and precision. As seen in Table 2, larger objects (such as trash cans and pedestrians) were detected over a greater range than smaller objects (such as a cardboard box).

Height (m)	Obstacle Range (m)																
	1.5	3	6.1	7.6	9.1	10.7	12.2	15.2	16.8	18.3	19.8	21.3	22.9	24.4	25.9	27.4
0.22	0	>.95	>.80	>.20	0	0	0	0	0	0	0	0	0	0	0	0
0.37	0	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.80	>.95	>.20	0	0	0	0	0
0.53	0	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	0	0	0	0	0
0.7	0	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.80	>.20	>.20	0
1.85	0	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.95	>.20	>.20	0

Table 2: Detection rates as a function of obstacle range and height.

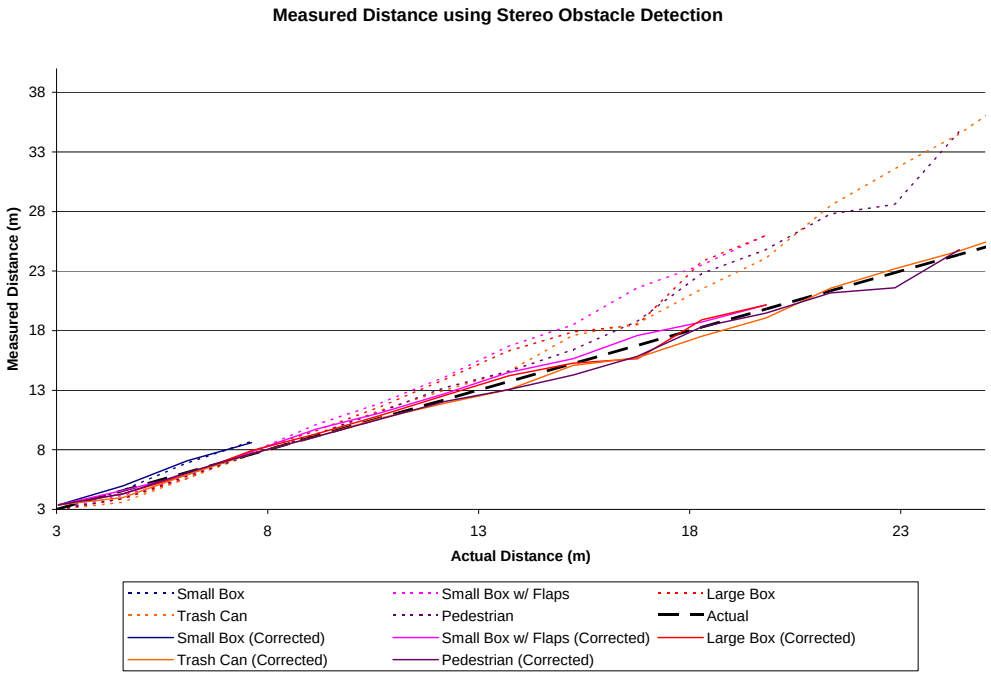


Figure 15: Measured distances for detected obstacles, before and after correction.

The measured distances of the detected obstacles were consistently higher than their actual distances, as demonstrated by the dotted lines in Figure 15. By using a quadratic fitting function and then solving for the actual distance, the appropriate correction was determined to be $20.83 \cdot \left(\sqrt{.6485 + .096d} - .8053\right)$, where d is the measured distance. With this correction, all

the measured distances match the actual distances to within 1m in regions with high detection rates.

3.2.2.2 Lane Detection

Lane detection is effective across numerous environmental and lighting conditions. We evaluated algorithmic performance on 200 images along a highway, with one solid lane marking and several dashed lane markings. Between 2 and 3 lanes are visible in each image.

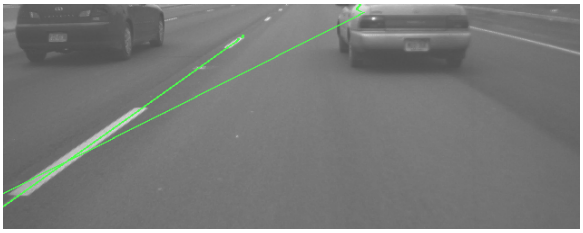


Figure 16: Example of lane detection failure.

In the 200 images, we find that one lane is found in 7% of images, two lanes are found in 88% of images and three are found in 5% of images. In all 200 images, only 5 false positives were observed. One such image is shown in Figure 16. Current limitations include difficulty with high-curvature lanes at long distances, and high levels of clutter. However, we believe that even in its current form, lane detection is sufficient for Urban Challenge navigation.

3.2.2.3 State Estimation

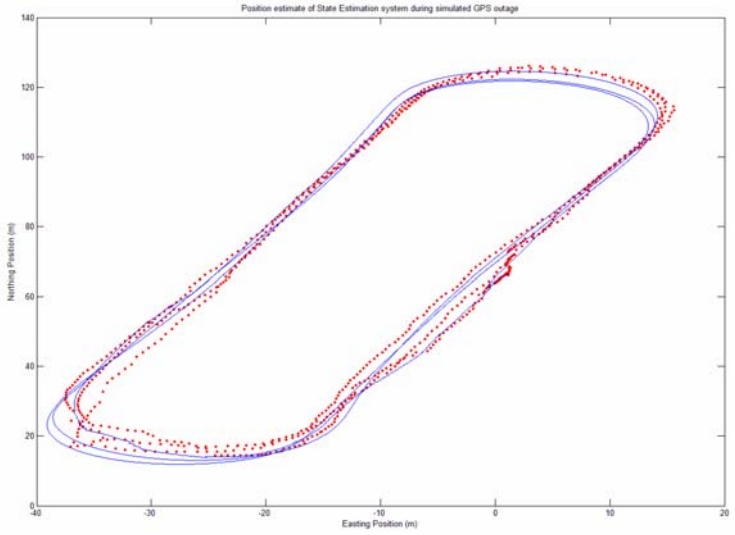


Figure 17: Simulation of State Estimation without GPS (blue).

State estimation is effective at providing global and local position information. Figure 17 shows the result of 3 laps around a 300m course. To evaluate the effectiveness of our algorithm, we simulate a GPS outage, starting halfway through the first lap, and persisting through the rest of the test. Drift over the remaining 750m is less than 2 m. This bodes well for both the global and local frame; for the global frame, it indicates we can withstand extended GPS outages, for the local frame, it means we can estimate local motion vectors with high accuracy. Several steps are planned to increase the accuracy of state estimation. First, we plan to more thoroughly model sensor performance, including latency. Further, we expect arrival of an IMU very soon, which will aid in enhancing our vehicle dynamics model and in withstanding GPS outages.

3.2.3. Cognition

The dividends of simulation have already proven to be extensive in testing navigation and path tracking software. The development cycle for implementation of advanced navigation behaviors, such as car following, lane changing, and intersection precedence, has been greatly reduced, allowing for more time testing both in simulation and in the real world. Currently, our navigation and path tracking algorithms have logged over 100 hours of simulation testing. Figure 18a shows a birds-eye view of the simulator, where the blue line is the desired path, the orange dot represents the crosstrack point and the purple dot is the longitudinal point. The heads-up display shows other relevant information.

(a) 

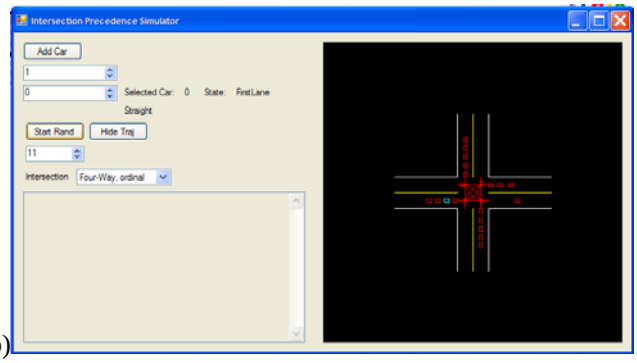
(b) 

Figure 18: Birds-eye view of simulation environment (a). Intersection precedence simulator (b).

Immediately planned improvements to the simulator include the ability to implement behaviors for other vehicles, modeling sensor noise, and more detailed field-of-view estimation. Future additions will include the ability to pause and rewind the simulator, as well as modify the environment without restarting a trial run.

Using our intersection precedence simulator, we have simulated over 10,000 vehicles exhibiting proper queuing, precedence and spline generation behavior. The full console of the precedence simulator is shown in Figure 18b. All variety of intersection configurations can be generated and the behavior of the cars can be adjusted.

3.2.4 Actuation

3.2.4.1 Actuators

Since the final implementation of our actuation scheme, we have observed a high degree of reliability as well as minimal delay. The maximum speed of the steering wheel under autonomous control is faster than a human can turn it. Although it is difficult to measure the actuation speed of the mechanical throttle and brake calipers directly, we are able to set the full range of the accelerator pedal instantaneously, apply full brakes within 280 ms and release them in 560 ms. Furthermore, we have not observed any mechanical or electronic failures of these systems in their final state.

The E-stop brakes actuator provides over 100 lbs of force directly to the brake pedal, fully pressing the pedal in less than 300 ms. The transmission shifter is capable of full-range shifting in under 2s in normal conditions. Its motor has been wrapped in weatherproofing material, which interferes with its air venting and causes it to heat up. During stress testing, we have observed a small decrease in the shifting speed. The motor has never stalled at elevated temperatures, but performance degradation over time due to heat damage is possible. We are

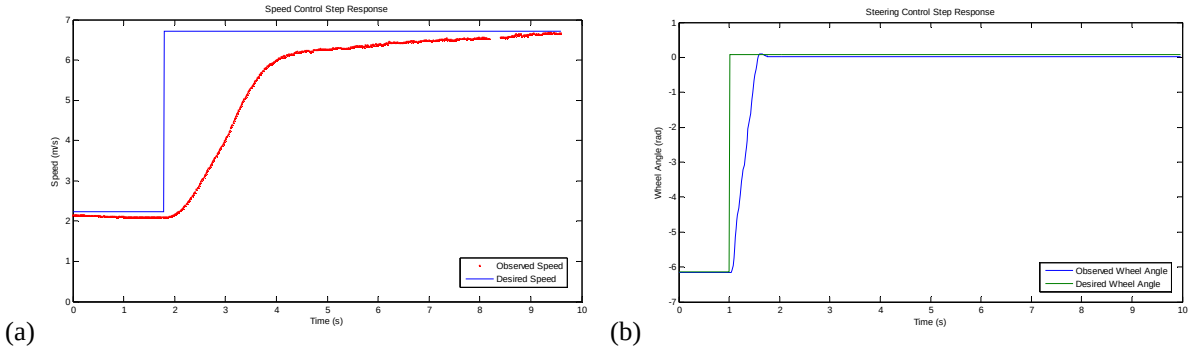
currently investigating weatherproofing solutions that do not interfere with the airflow over the motor.

One of our key concerns is overheating of the Labjacks and circuitry stored under the hood. We have logged temperatures inside the electronics box up to 122 ºF. Although this is lower than the Labjack failure temperature of 185 ºF, we believe that temperature could be reached under extreme conditions. An active cooling system is being designed that will pipe cooled air into the electronics box and exhaust the hotter air.

3.2.4.2 Controls

The speed controller was specifically tuned to have no overshoot. As a result, the system is slightly overdamped. Its 1% settling time is around 7.25 seconds and it has no steady state error. A sample step response from 5 to 15mph is shown in Figure 19a.

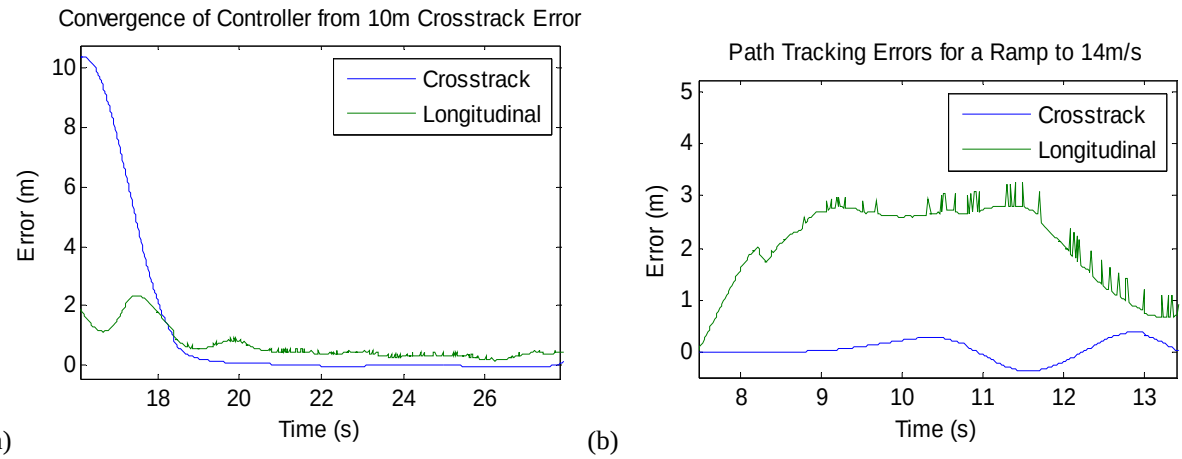
Steering delay adds error to our higher-level lateral control loop. Consequently, the low-level steering controller was tuned to give as fast a response as possible. The resulting system is close to critically damped, although it still exhibits slight overshoot. The rise time is around 0.5 seconds and the 1% settling time is 0.75 seconds. A step response of the steering controller is given in Figure 19b. It is worth noting that this is a step of one full revolution of the steering wheel. This response is faster than the average human driver.



(a) (b)

Figure 19: Speed control response for a 10mph step (a). Steering control response to a step of 360° (b).

The path tracking controller was found to be stable at moderate speeds. Figure 20a shows convergence to a straight path from a ten meter initial error. The longitudinal error first increases, and then decreases as the crosstrack error nears zero. Because the speed controller is slow to attain its steady state, a longitudinal error of .25m persists for the duration of the run. The speed controller is currently under redesign to correct this. Figure 20b shows a current limitation of the controller at higher speeds. At 14m/s, the crosstrack error becomes unstable. This issue is noted in [ref Stanford paper], and corrected for with a yaw damping term which will be added soon. The longitudinal error during the speed ramp is constant as expected. Once the speed ramp ends at t=11.5s, the longitudinal error decreases to a steady value of .69m.



(a) (b)

Figure 20: Path tracking response from a 10m crosstrack error (a) and a ramp to 14 m/s (b).

4. Conclusion

Throughout the design and testing of all our systems, we have been guided by a commitment to simplicity, reliability, and cost-effectiveness—principles demanded of each of our individual systems, their aggregate, and the Challenge itself. Our efforts to uphold these principles have challenged us to explore more innovative solutions, from lane detection and sensor fusion, to harnessing the vehicle's existing drive-by-wire capabilities. They have imbued us with the belief that a cost-effective solution is not an inferior one—that the Urban Challenge can be completed without acquiring costly equipment. Our efforts have focused on turning creative solutions into reliable systems. Thus the reliability stems from our ideas and the robustness of their implementation, instead of from costly, high-precision, and even superfluous equipment.

In our quest to develop elegant, efficient, and cost-effective solutions, we take on the intellectual challenge of engineering a truly urban vehicle—one which relies on the regularity of the urban environment and the rigid rules of urban driving to derive the vehicle's behavior. Given the likelihood of GPS failure in real urban canyons, we chose not to rely too heavily on this sensor, but depend instead on our fusion algorithms to synthesize it with various other sensor inputs. Finally, because we are a team of undergraduates whose objective is to enrich the academic experience of its members, this approach ensures that we remain open to the exploration of all solutions and all sides of the hurdles that await us in the urban challenge.

References

DARPA, 2007	DARPA. *Urban Challenge Technical Evaluation Criteria*. DARPA Urban Challenge Documentation. Arlington, VA. March, 2007. http://www.darpa.mil/grandchallenge/docs/Technical_Evaluation_Criteria_031607.pdf
Franken, 2007	Franken, Gordon and Glass, Zachary. *Advanced State Estimation and Control of an Autonomous Ground Vehicle Using* a priori *Knowledge of Vehicular Dynamics.* Princeton University Junior Independent Project. Princeton, NJ. May 2007. Available at: http://pave.princeton.edu/publications/
Freund, 1997	Freund, Y. and Schapire, R. *A decision-theoretic generalization of on-line learning and an application to boosting*. Journal of Computer and System Sciences, 55(1):119-139. 1997
Hoffmann, 2007	Hoffmann, G., Tomlin, C., Montemerlo, M., and Thrun, S. *Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing.* To appear in the Proceedings of the 26th American Control Conference, New York, NY, July 2007. http://hoffmann.stanford.edu/papers/hoffmann_stanley_control07.pdf
Manduchi, 2005	Manduchi, R. *Obstacle Detection and Terrain Classification for Autonomous Off-Road Navigation.* Autonomous Robots 18:81-102, 2005 http://www.soe.ucsc.edu/~manduchi/papers/AutRobots.pdf
Lowe, 2004	Lowe, D. *Distinctive image features from scale-invariant keypoints.* International Journal of Computer Vision 60,2. pp. 91-110. 2004.
St. Pierre, 2004	St. Pierre, M. and Gingras, D. *Comparison between the unscented Kalman filter and the extended Kalman filter for the position estimation module of an integrated navigation information system.* 2004 IEEE Intelligent Vehicle Symposium,. pp.831-835. June, 2004. http://ieeexplore.ieee.org/iel5/9278/29469/01336492.pdf
Thrun, 2004	Thrun, S., Burgard, W. and Fox, D. *Probabilistic Robotics*. The MIT Press. Massachusetts. 2004.
Torralba, 2004	Torralba, A., Murphy, K. P. and Freeman, W. T. *Sharing features: efficient boosting procedures for multi-class object detection*. Proceedings of the 2004 IEE Computer Society Conference on Computer Vision and Patter Recognition. pp. 762-769. 2004.
Welch, 2006	Welch and Bishop. *An Introduction to the Kalman Filter*. Department of Computer Science. University of North Carolina at Chapel Hill. July, 2006. http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html

Acknowledgements

We would like to thank our team sponsors: Texas Instruments, Valde Systems, Agilent Technologies, Delphi, Ted Todd '67, BookSwim, Shock Tech, FRABA–Posital, The Norman D. Kurtz '58 Fund for Innovation in Engineering Education, and the Princeton University Lion Senior Thesis Fund.

We would like to especially acknowledge Ford Motor Company for donating our competition vehicle for the Urban Challenge.

In addition, we would like to thank Princeton University and the School of Engineering and Applied Science for their continuing support of our research

The students wish to thank our team leader and faculty advisor, Professor Alain Kornhauser, for his support of this project and steadfast dedication to enriching our academic careers.