DARPA Urban Challenge Final Report for Tartan Racing

Submitted to Dr. Norm Whitaker

In partial fulfillment of Milestone 4 contract requirements for contract HR0011-06-C-0142

December 31, 2007

By

Carnegie Mellon University
The Robotics Institute
5000 Forbes Avenue
Pittsburgh, PA 15213

Principal Investigator
William “Red” Whittaker
red@cmu.edu
412-268-6556

Technical POC
Chris Urmson
curmson@ri.cmu.edu
412-268-5901

Sponsored by
Defense Advanced Research Projects Agency
Program: Urban Challenge
ARPA Order No. W363/00, Program Code: DIRO
Issued by DARPA/CMO under Contract No. HR0011-06-C-0142

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.
DARPA Urban Challenge Final Report for Tartan Racing

December 31, 2007

Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Dave Ferguson, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Alonzo Kelly, Nick Miller, Matt McNaughton, Kevin Peterson, Maxim Likhachev, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, Red Whittaker, Ziv Wolkowicki and Jason Ziglar

Carnegie Mellon University

Hong Bae, Thomas Brown, Daniel Demitrish, Jim Nickolaou, and Varsha Sadekar, Wende Zhang

General Motors

Joshua Struble and Michael Taylor

Caterpillar

Michael Darms

Continental AG

Corresponding Author:

Chris Urmson (curmson@ri.cmu.edu)

Executive Summary

Tartan Racing developed Boss (an autonomous 2007 Chevrolet Tahoe) to compete in the 2007 DARPA Urban Challenge. Boss uses on-board sensors (GPS, lasers, radars, and cameras) to track other vehicles, detect static obstacles and localize itself relative to a road model. A three layer planning system combines mission, behavioral and motion planning to address the problems of urban driving. The mission planning layer considers which street to take to achieve a mission goal. The behavioral layer determines when to change lanes, precedence at intersections and performs error recovery maneuvers. The motion planning layer selects actions to avoid obstacles while making progress towards local goals.

The system was developed from the ground up to address the requirements of the DARPA Urban Challenge using a cyclic system development process with a heavy emphasis on regular, regressive system testing. During the national qualification event and the urban challenge final event Boss demonstrated some of its capabilities, qualifying first and completing and winning the DARPA Urban Challenge.
# Table of Contents

Table of Contents ......................................................................................................................... 3

Introduction ..................................................................................................................................... 4

Boss .................................................................................................................................................. 4
   Vehicle and Actuation .................................................................................................................. 4
   Power and cooling ....................................................................................................................... 4
   Controls ....................................................................................................................................... 5
   Sensors ......................................................................................................................................... 6

Urban Challenge Software ............................................................................................................. 6
   Architecture ............................................................................................................................... 7
   Infrastructure .............................................................................................................................. 7
   Perception .................................................................................................................................... 9
   Mission Planning ...................................................................................................................... 16
   Behaviors .................................................................................................................................... 18
   Motion Planning ...................................................................................................................... 26

Testing ........................................................................................................................................... 33
   Cyclic System Development .................................................................................................... 33
   Test Case Design ..................................................................................................................... 33
   System Testing Process ........................................................................................................... 33
   Test Summary Statistics .......................................................................................................... 34

Site Visit ......................................................................................................................................... 34

National Qualification Event ......................................................................................................... 35

Final Event ...................................................................................................................................... 36

Lessons Learned ........................................................................................................................... 37

Conclusions .................................................................................................................................... 39

Acknowledgements ...................................................................................................................... 39

References ....................................................................................................................................... 39
Introduction

Tartan Racing developed Boss (an autonomous 2007 Chevrolet Tahoe) to compete in the 2007 Urban Challenge. Boss uses on-board sensors (GPS, lasers, radars and camera) to track other vehicles, detect static obstacles and localize itself relative to a road model. A three layer planning system combines mission, behavioral and motion planning to address the problems of urban driving. The mission planning layer considers which street to take to achieve a mission goal. The behavioral layer determines when to change lanes, precedence at intersections and performs error recovery maneuvers. The motion planning layer selects actions to avoid obstacles while making progress towards local goals.

The entire system was developed from the ground up to address the requirements of the Urban Challenge using a cyclic system development process with a heavy emphasis on regular, regressive system testing. During the national qualification event and the urban challenge final event Boss showed some of it’s capabilities, qualifying first and completing and winning the Urban Challenge.

Boss

Vehicle and Actuation

Boss is a 2007 Chevrolet Tahoe modified for autonomous driving. Chassis upgrades include an internal roll cage, improved suspension for additional equipment payload, modified steering rack, racing brakes and rotors, wheels, and self-sealing tires. The OEM engine control module (ECM) was modified to accept direct throttle commands via the CAN bus. A commercial off-the-shelf drive-by-wire system was integrated into Boss with electric motors to turn the steering column, depress the brake pedal, and shift the transmission. The standard human driving controls remain intact and functional allowing a “safety driver” to sit in the driver’s seat with a manual override button within easy reach. Boss has its original seats in addition to a custom center console with power and network outlet for developers observing Boss’s performance on their laptops. For unmanned operation a safety radio is used to engage autonomous driving, pause or disable the vehicle. The third-row seats and cargo area were replaced with electronics racks.

Power and cooling

Boss has two independent power busses. The stock Tahoe power bus is intact with its 12VDC battery and harnesses, upgraded with a high-output alternator for extra margin. An auxiliary 24VDC power system consisting of a belt-driven generator, battery bank, and an inverter charger module (ICM) was added to run the autonomous driving equipment. The ICM charges the 24VDC battery bank from the generator and inverts the battery power to supply 120VAC. Battery chargers supplied from shore power enable Boss to remain fully powered up without running the engine. Cooling is done with the stock Tahoe air conditioning with modified control software to optimize heat rejection.
**Controls**

**Drive-by-Wire Actuators**

Low-level control, except throttle-by-wire, is executed by a commercially available Electronic Mobility Controls (EMC) system. These functions include positioning the steering wheel, brake pedal, accelerator pedal, and transmission gear selector. Additionally, the EMC system provides a serial interface to control headlights, turn signals, windshield wipers, and the parking brake.

**Real-time Control Platform**

The mid-level control software runs on a dSpace MicroAutoBox. MicroAutoBox is a real-time system for performing rapid control prototyping for automotive applications. Programs are stored in nonvolatile memory, enabling MicroAutoBox to start up on power-up and behave like, and communicate with, the various stock controllers already integrated into the vehicle as well as the EMC system.

**Speed Control**

The velocity controller has two separate PID modules, one responsible for braking and one for accelerator control. Only one PID module at a time is active enabling their outputs to be combined into one signal. Each module has a single set of gain values for all conditions. The speed controller is provided with a desired speed and a desired acceleration command.

The decision when to switch between the brake and throttle PID controllers is important. The controller uses the desired acceleration value to reason about the current machine state to make fast, intelligent switching choices. A separate acceleration command, as opposed to differentiating the velocity command, enables the planning module to introduce instantaneous changes in the velocity command without corrupting the switching logic.

The Apply Brake function was added to the basic velocity controller to improve low-speed performance since the wheel speed encoder does not allow accurate low-speed measurements. To ensure that the vehicle is stopped, the controller enters a feed-forward braking mode when the velocity command is zero and the actual speed is below a threshold, staying in this mode as long as the command is zero. A simple integrator capping scheme helps to avoid delays associated with the electromechanical system and integrator windup. Additionally, the brake module integrator employs multiple initial values. To prevent overheating of the electromechanical actuator when the vehicle was in “Pause”, the system is placed into Park after 5 seconds and the forceful brake application is stopped until the vehicle exits Park.

The system is able to track velocity profiles within 0.5 m/s with a 0.25m/s steady-state error bound for feasible trajectories (i.e. over accelerations up to $1.8m/s^2$, the maximum bulk acceleration of Boss).

---

For DARPA internal use only
Steering Control
The main structure of the steering control is feedforward control with a small compensation from PID feedback. The commanded steering angle is filtered through a magnitude saturator and a rate limiter to protect the steering hardware as well as avoid instantaneously large error signals.

Sensors
Boss uses a combination of sensors to satisfy both the necessary sensing requirements to navigate safely in an urban environment and to provide some redundancy should sensors fail during operation. The selected sensors are described in Table 1, and their role in the overall perception strategy is outlined in Table 2.

Table 1. A description of the sensors incorporated onto the Tartan Racing Robots.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Characteristics</th>
</tr>
</thead>
</table>
| Applanix POS-LV 220/420 GPS / IMU | • sub-meter accuracy with Omnistar VBS corrections  
                                  • tightly coupled inertial/GPS bridges GPS-outages |
| SICK LMS 291-S05/S14 Lidar  | • 180° / 90° x 0.9° FOV with 1° / 0.5° angular resolution  
                                  • 80m maximum range |
| Velodyne HDL-64 Lidar     | • 360° x 26° FOV with 0.1° angular resolution  
                                  • 70m maximum range |
| Continental ISF 172 Lidar | • 12° x 3.2° FOV  
                                  • 150m maximum range |
| IBEO Alasca XT Lidar      | • 240° x 3.2° FOV  
                                  • 300m maximum range |
| Continental ARS 300 Radar | • 60° / 17° x 3.2° FOV  
                                  • 60m / 200m maximum range |

Table 2. The roles filled by different sensing modalities.

<table>
<thead>
<tr>
<th>Role</th>
<th>Velodyne</th>
<th>SICK</th>
<th>Alaska XT</th>
<th>ISF 172</th>
<th>ARS 300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detecting and mapping static obstacles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detecting and tracking moving vehicles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Localizing to a prior map</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimate road shape and lane locations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The decision to use only active sensing was guided primarily by the team’s skill set, and the belief that, in the urban challenge direct measurement of range and target velocity was more important than getting richer, but more difficult to interpret data from a vision system.

Urban Challenge Software

For DARPA internal use only
Boss replaces the human driver with a virtual one that is comprised of over 100 processes distributed across twenty processors to autonomously drive Boss. These processes are built from over 400,000 source lines of C++ code partitioned into nearly 100 libraries developed over the course of eighteen months.

The Urban Challenge software includes perception, behavioral and planning components that are combined through supporting infrastructural components that glue the algorithms together into a coherent system.

**Architecture**

The task of autonomous urban driving can be decomposed into the following major blocks:

* **Infrastructure** – The infrastructure is the foundation upon which the algorithms are built and provides a toolbox of components for online data logging, offline data log playback, and visualization utilities that aid developers in building and troubleshooting the system.

* **Perception** – Perception components interface sensors, process raw sensor data, and fuse the multiple streams together in order to provide a composite picture of the world to the rest of the system. The composite model is divided into several discrete elements: static obstacle maps, moving obstacle lists, sensor visibility, sensor health, current vehicle pose, and the road’s geometry.

* **Mission Planning** – The Mission Planning component computes the cost of all possible routes given an RNDF, road blockages detected by Perception, and the next checkpoint the vehicle must achieve according to the mission definition as stated in an MDF.

* **Behaviors** – The Behaviors component formulates a problem definition for the motion planning component to solve based on the strategic information provided by the mission planning component. The behaviors component is implemented as a state machine that decomposes the mission task into top-level behaviors and their simpler sub-behaviors in order to complete a mission.

* **Motion Planning** – The Motion Planning component consists of two planning subsystems, each capable of avoiding static and dynamic obstacles while safely achieving a desired goal: structured driving (road following) and unstructured driving (parking lot or jammed intersection).

These areas are addressed in greater detail in the following sections and include key requirements, design, and performance criteria.

**Infrastructure**

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

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

* **Communications Library** – Abstracts around basic interprocess communication over UNIX

  For DARPA internal use only
Domain Sockets, TCP/IP, or UDP; supports the Boost.Serialization library to easily marshal a data structure across the communications link, then unmarshal on the receiving side. A key feature is anonymous publish/subscribe, to disconnect a consumer of data from having to know who is actually providing that data.

**Interfaces Library** – Each interface between two processes in the system is added to this library. Each interface fits into a plugin framework so that a task, depending on its communications configuration, can dynamically load the interfaces required at run-time. For example, consider a perception task that reads from an abstract Lidar interface, and at run-time is configured to either load a SICK or an IBEO implementation of the Lidar interface. Furthermore, interfaces can be built on top of other interfaces in order to produce composite information from multiple sources of data. For example, a pointed lidar interface combines a lidar interface, and a pose interface.

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

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

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

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

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

**Geometry Math Library** – Common routines for performing mathematical operations (rotations, transformations, translations, matrix operations, etc) with various geometric data structures (e.g., point, pose, polygon, quaternion, etc).

**Scrolling Map** – A scrolling two-dimensional map library that can store an arbitrary data type in each cell.
SCons Build System – The SCons build system is used as a replacement to the common make build system. SCons provides nice improvements over make: it automatically calculates dependency information, and is built on the Python scripting language for powerful extensibility.

Perception

Moving Obstacle Detection and Tracking

The moving obstacle detection and tracking subsystem provides a list of moving obstacles and their characteristics to the system. The following design principles guided the implementation:

- No information about the driving context is used inside the tracking module.
- No explicit vehicle classification is performed. The tracking system only provides information about the movement state of object hypotheses.
- Information about the existence of objects is based on sensor information only. The time during which objects are predicted without being detected by sensors reflects typical duration of sensor artifacts only. Detection drop outs caused by noise, occlusions and other artifacts are handled specifically within the Behaviors subsystem.
- Object identifiers are not guaranteed to be stable. A new identifier does not necessarily mean that an object has not been detected before.
- Well defined and distinct tracking models are used to maximize the use of information provided by heterogeneous sensors.
- Predictions are based on logical constraints where possible.
- Sensor specific algorithms are encapsulated in sensor specific modules.

Figure 2 shows the two tracking models, which were used to describe moving obstacles. The box model uses a fixed length and width to represent the shape of a vehicle whereas the point model has no shape information. The box model uses a simple bicycle model for state propagation [1], the point model uses a constant acceleration model [2] with adaptive noise dependent on the length and direction of the velocity vector. A model switching approach selects the model with highest precision currently supported by sensor data [3].

Figure 3 shows the four different movement states into which the tracking system classifies object hypotheses. The Moving flag is set, if the object currently has a velocity which is significantly not equal to zero. The flag Observed Moving is set once the object has been moving for a significant amount of time. The four states act as a well

For DARPA internal use only
defined interface to the other software modules.

Figure 4 shows the architecture of the tracking system. It is divided into two layers, a Sensor Layer and a Fusion Layer [4] [5]. For each sensor type (e.g. radar, scanning laser, etc.) a specialized sensor layer is implemented, which encapsulates all sensor specific algorithms. This enables new sensor types to be added to the system with minimal changes to the existing software.

Each time a sensor has new raw data it requests a prediction of the current tracked object hypotheses from the fusion layer. Features are extracted from the measured raw data (e.g. edges from laser scanner data [6]. Artifacts caused by ground detections or vegetation are suppressed by validating features with sensor specific algorithms (e.g. using the velocity measurements inside a radar module) and over a general validation interface provided by the fusion layer. The validation performed in the fusion layer hereby only uses non sensor specific information and performs checks against the road geometry and against an instantaneous obstacle map, which holds untracked 3D information about any obstacles in the near range.

The validated features are associated to the predicted object hypothesis. For each extracted feature, possible interpretations as a box or point model is generated using a sensor specific heuristic which takes the sensor characteristics into account (e.g. resolution, field of view, detection probabilities). Finally, a measure for the compatibility of the generated interpretations with the associated prediction is computed.

If an interpretation differs significantly from the predicted object, then the sensor module initializes a new object hypothesis which can potentially replace the current model hypothesis used in the fusion layer. This alternative model is called a proposal. An observation is generated for the interpretation which fits best to the prediction. The observation holds all of the data necessary to update the state estimate in the fusion layer. If no interpretation is compatible, then only proposals without observations are generated. This is also performed for features which cannot be associated to any object hypothesis.

In addition to the observation and proposal generation, the sensor module also provides a movement observation with each measurement to the fusion layer. The movement observation is based on the raw sensor data and provides information about whether an object is currently moving or not (e.g. via an evaluation of the velocity measurement inside the radar module).

The best tracking model is selected with a voting algorithm in the fusion layer. The algorithm tracks the alternative proposals provided from the different sensors. Additionally, it uses information about the road shape to bias the selection of the best model on roads. In parking lots

For DARPA internal use only
the best proposal is selected according to a ranked ordering of the proposals provided by the sensor modules.

Once the best model is selected, the state estimate is updated with the observation provided by the sensor layer or the model is switched to the current best alternative. A global classification of the movement state is carried out which is based on the movement observations from the sensors and a statistical test based on the estimated state variables.

For objects which have the *Moving* and *Observed Moving* flag set, a prediction is carried out. The prediction is based on logical constraints for objects on the road. At every point where a driver has a choice to change lanes (e.g. intersections), multiple hypotheses are generated. In zones the prediction is solely based on the estimated states (see Figure 5).

**Static Obstacle Detection and Mapping**

The static obstacle mapping system consists of two primary components - an algorithm to detect obstacles and a second algorithm to detect curbs. The output of these two algorithms is fused to generate a single static obstacle map consisting of both features. Due to space constraints, only the curb detection algorithm is presented in this report.

**Curb Detection and Mapping**

Geometric features present a reliable method for determining road shape in urban and off-road environments. With rich LIDAR data presenting a detailed view of the geometry ahead, geometric features can provide accurate long-range detection. Smooth and safe driving for Boss requires road shape approximately 30 m in front of the vehicle. Detection algorithms need to prove robust to the variation in geometric features found across the many variants of curbs, berms, ditches, embankments, etc. In order to take advantage of LIDAR data, and detect the wide variety of geometric features at range, a detection algorithm based on the Haar wavelet is presented.

Haar wavelets provide an efficient method for performing multi-resolution numerical derivatives, using a mother wavelet as shown in Figure 6 [7]. Haar wavelets can robustly detect edges, particularly when edges occur over different scales [8]. Since edge scale could potentially range from centimeters to meters, the multi-resolution features of the Haar wavelet were appealing. Difficulties in applying the Haar wavelet come primarily from input restrictions; wavelet transforms must have exactly $2^n$ points, and points must have constant

---

For DARPA internal use only
LIDAR data received by the vehicle will contain regions of sparseness. Sparse regions occur due to occlusions by obstacles, intermittent returns through vegetation or fences, and sensor configuration. As an example, the two aft SICKs on Boss point perpendicular to the vehicle and out over the rear wheel which results in dense data near the vehicle, and sparse data further out. This kind of sparseness introduces artifacts in the Haar transform, since the transform assumes even spacing between points for its multi-resolution calculations. In order to handle sparseness due to obstacles and occlusions, preprocessing through distance-based clustering and thresholding is applied. Points are initially clustered by distance between consecutive points – once clustered, groups of points which are too small are removed from the scan. The second pass annotates each point with a dense/sparse classification – points which are considered sparse are used in processing other points but or not considered for labeling as curb points.

After sparseness checking, the remaining points are linearly interpolated. This results in even spacing between points, which eliminates artifacts in the Haar transform. Furthermore, by forcing the output of the linear interpolation to always return $2^n$ points, input length requirements of the Haar transform are met. With data linearly interpolated, the height data are taken and passed through the Haar transform. The data are now the wavelet coefficients, which are equivalent to the average slope for the windows at each resolution level.

The actual feature extraction uses an iterative approach to finding edges. An example of the data is shown in Figure 8, which shows the view from an aft SICK. Feature extraction works by looking at the coarsest resolution level, and thresholding windows into either curb or road candidates. The regions classified as roads are fed into the next resolution level, and used to calculate the mean slope of road candidates. The coefficients at this resolution level are

![Wavelet Based Curb Detection](image)

**Figure 7 Data Flow Diagram of the Curb Detection System**

![Example Data Above, First Step of Feature Extraction Below](image)

**Figure 8 Example Data Above, First Step of Feature Extraction Below**

-12-

For DARPA internal use only
thresholded relative to the calculated road mean, and the algorithm iterates these steps to the final window of interest. Figure 8 shows a sequence of analysis from successive resolution levels, and the resulting classification for each. At the final level, each window classified as non-road selects the two consecutive points creating the steepest slope. All other points are classified as non-curb points.

The algorithm received data from three SICKs and twenty-five of sixty four Velodyne lasers, processing all input data in real-time and proved capable of detection of approximately ≤0.05 m curbs, at ranges up to 40 m.

**Roadmap Localization**

The localization system used by Boss incorporates three components: a high quality GPS/INS system, a road map, and lane marking detection algorithms. These components are integrated to provide an estimate of where Boss is relative to an RNDF. The system also enables boss to operate with intermittent GPS and to correct for potentially significant GPS or map errors. For other perception and planning algorithms to work, the state estimation system must provide locally smooth position estimates while at the same time maintaining global accuracy such that the autonomous vehicle can reliably stay within its lane.

The state estimation system uses the output of an off-the-shelf fused inertial and GPS state estimation system from Applanix. The Applanix unit incorporates differential corrections (both Omnistar *High Precision* and *Virtual Base Station*) to provide an inertially smoothed position estimate with at best 0.1m CEP and expected nominal errors on order of 0.6m CEP. By incorporating inertial and wheel speed sensors the system is able to continue to offer quality position estimates during GPS dropout and in multi-path scenarios.

The localization system incorporates these state estimates and a prior map to ensure that even in situations when the global accuracy of the Applanix system is degraded, the state estimate is sufficient to maintain position within a lane. It is important to note that the localization result is not in a widely accepted coordinate system (e.g. relative to a UTM plane) but in a distorted frame defined by the road model. While this frame will in general be similar to a widely accepted coordinate system, due to errors in the road model it is unlikely they will be the same.

Localization is performed by estimating position error relative to the road model. The location of lane markings relative to the nominal lane edges are measured using the intensity signal of down looking laser scanners. The lateral offset between the measured lane marker location and the nominal lane edge are then averaged over some distance of travel (nominally 1m). The composite lateral offset estimates are then combined in a recursive filter of the form $E_{cur} = (1 - \alpha)E_{prev} + \alpha E_{int \tan \text{yaw}}$. The resultant error estimate is then applied to the current position estimate from the GPS/INS system.

Using this approach Boss is able to maintain a smooth position estimate and localize relative to a given RNDF even in situations where GPS quality is insufficient for it to remain in its own lane.

**Road Shape Estimation**

In order to robustly handle sections of the RNDF where waypoints were sparse, we developed an estimator that measures the curvature, position, and heading of roads near the vehicle. The estimator fuses inputs from a Velodyne, roof-mounted SICKs, IBEO lidars, and cameras to composite a road-relevant model of the world. The estimator is initialized using the RNDF,
where available, and generates a best-guess road location between designated sparse points where a road may twist and turn. The road shape is represented as the Taylor expansion of a clothoid with an offset normal to the direction of travel of the vehicle. This approximation is generated at 10 Hz.

**Sensor Inputs**

Three primary features were used to determine road location.

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

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

*Painted line markers* represent the edge of the lanes. Lanes are centered on the parametric model and are allowed to vary in width relative to the overall width of the road. The estimator is more likely to generate a road shape estimate that aligns with observed lane markers.

Given an input image, pixel values are normalized to be zero mean and unit variance to enhance image contrast. The image is then convolved with oriented first- and second- derivative-of-Gaussian filters to detect lines that may be present. In each orientation, the line points that have high line-filter responses with high edge-filter responses on both sides along the perpendicular orientation are selected. Point detections are then conglomerated into linear features through the use of the Hough Transform [9]. Finally, heuristics are applied to filter out short line segments and shadows resulting from trees or buildings. Sample detection results are shown in Figure 9.

**State Vector**

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

\[ s(t) = (x(t), y(t), \phi(t), C_0(t), C_1(t), W(t)) \]

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

\[ y(x) = \tan(\phi(t))x + C_0 \frac{t}{2} x^2 + C_1 \frac{t}{6} x^3 \]

**Particle filter**

The road estimator uses an SIR (sample importance resample) [10] filter populated by 500 particles. Each particle is an instantiation of the state vector. During the sampling phase, each particle is propagated forward according to the following equations where \(ds\) represents the relative distance that the robot traveled from one iteration of the algorithm to the next.
The final $C_1$ term represents the assumption that the curvature of a road will always tend to head towards zero which helps to straighten out the particle over time. After the deterministic update, the particle filter adds random Gaussian noise to each of the dimensions of the particle in an effort to help explore sudden changes in the upcoming road curvature that aren't modeled by the curve parameters. In addition to Gaussian noise, several more directed searches are performed where the width of the road can randomly increase or decrease itself by a fixed amount. Empirically, this represents the case where a road suddenly becomes wider because a turn lane or a shoulder has suddenly appeared.

**Sensor data processing**

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

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

$$L = \frac{1}{Z} \exp^{-C(\text{shape, data})}$$

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

The filter evaluates the number of obstacles, $N_o$, and number of curb points, $N_c$, encountered inside the road shape; the distance of the edge of the road to the detected curb points, $D_c$; the distance between the observed lane markers and the model’s lane markers, $D_l$; and the presence of blockage across the road, $B$. In order to scale the cost function, counts and distances are normalized. The resulting cost function is:

$$C = \sum_{i=0}^{N} \left( \frac{N_i}{\sigma_o} \right)^2 + \left( \frac{N_i}{\sigma_c} \right)^2 + \left( \frac{D_i}{q_c} \right)^2 + \left( \frac{D_i}{q_l} \right)^2 + \alpha B_i$$

**Fast Convolutions and Distance Transforms**

In order to exactly compute the cost function each road shape must be convolved with the cost map to sum the detection counts and obstacle costs. This requires tens of thousands of convolutions per second for each count. While fast methods exist to exactly compute simple shapes [15], road shapes are too complicated for these approaches. Instead, the road shape is approximated as overlapping discs centered on the road shape Figure 10.

The discs have a radius equal to the width of the road and are spaced at 1.5 meter samplings. While this approach tends to over-count, empirically, it is adequate for the purposes of tracking.
the road, and is more than fast enough.

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

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

To evaluate the distance components of the cost function the distance transform [16] is used. The distances from the nearest curb location or lane marker location to a given sample location is built into a distance map. The distance map can then be examined at sample points and evaluated like the cost counts.

Summing the overall cost function results in a minimum located at the true location of the road.

The performance of the final system can be observed in Figure 11. These were taken on an off-road stretch where some geometric features were visible in terms of berms and shrubbery as obstacles. The top of the figure shows the output from two cameras mounted on the top of Boss. The particle filter stretches forward from the vehicle and the road is represented as 3 lines.

**Mission Planning**

**Basic Route Planning**

To generate mission plans, the data provided in the RNDF are used to create a graph that encodes the connectivity of the environment. Each waypoint in the RNDF becomes a node in this graph, and directional edges (representing lanes) are inserted between a given waypoint and all other waypoints that it can reach. These edges are assigned costs based on a combination of several factors, including expected time to traverse the edge, distance of the edge, known road blockages, and complexity of the corresponding environment.

This graph is searched to compute a minimum-cost path from each position in the graph to a desired goal position, such as the first checkpoint in the mission. Computing minimum-cost paths from every position is useful because it enables the navigation system to behave correctly
should the vehicle be unable to perfectly execute the original path. As the vehicle navigates through the environment, the mission planner updates its graph to incorporate newly-observed information, such as road blockages. Each time a change is observed, the mission planner regenerates a new policy.

**Road Blockage Handling**

The robot's progress can be upset by blockages across a road, such as a large traffic accident or an intentional barricade. When such a blockage is encountered, the robot must plan an alternate path to its goal. It must be able to detect and respond to blockages on one-way roads, two-way roads, and across intersections. The robot should eventually revisit the site of a previously encountered blockage to see if it has been cleared away, thus potentially saving time in the long run. In fact, the robot *must* revisit a blockage if all other paths to the goal have also been found to be blocked.

**Representation of Blockages**

The effect on the road model of a blockage on a two-way road not near an intersection is illustrated in Figure 12. The affected lanes are identified and the extent along them that the blockage occupies. Locations before and after the blockage are identified within a safe margin away from the blockage at which to add a U-turn. Road model elements representing a legal place to make a U-turn are added at the chosen locations, and the corresponding traversal costs are set to a low value. Traversal costs of road elements that cross the blockage are boosted by a large amount. Since the robot follows the lowest-cost path to its goal, the high costs levied on traversing a blockage cause the robot to choose an alternate path. When the blockage is gone, the traversal costs for elements crossing the blockage are restored, and the traversal costs for the added U-turn elements are set high so that the robot will not attempt to make an illegal U-turn. The U-turn elements are not simply removed outright from the road model for reasons to be discussed later.

![Figure 12 Blockage on a Two-Way Road With No Intersection Nearby](image)

The revisiting behavior is induced by gradually reducing the traversal cost increments applied to road elements crossing a blockage. When the cost has dropped below a threshold, the blockage is treated as if it were directly observed to be gone. Note that the U-turn traversal costs are not concomitantly increased. They are changed all at once when the blockage is observed or assumed to be gone. Decreasing the cross-blockage traversal costs encourages the robot to return to check whether the blockages is removed, while *not* increasing the U-turn traversal costs encourages the robot to continue to plan to traverse the U-turn if it would be beneficial to do so.

The cost increment added by the blockage is reduced using an exponential decay rate. The cost increment is $\text{cost} = p \cdot 2^{-a/h}$, where $a$ is the time since the blockage was last observed, $h$ is a half-life parameter, and $p$ the starting cost penalty increment for blockages. To illustrate, if the blockage is new, $a=0$ and $\text{cost}=p$. If the blockage was last observed $h$ time units in the past, $a=h$ and $\text{cost}=p/2$ and the cost continues to decay exponentially as the blockage ages.
**Perception of Blockages**

Blockages are detected along the road infront of Boss by searching the static obstacle map along road segments near the vehicle and checking to see if they contain lethal obstacles blocking the road. The search is restricted to areas that are indicated in the visibility map to be unobscured. Noise in the obstacle map is suppressed by ignoring apparent blockages that have been observed for less than five seconds. Blockages are counted as gone once the obstacle map shows a wide path available through the segment.

There may be conditions on the road ahead that do not appear to the perception layer as a road blockage, but which prevent the robot from making forward progress. In this condition, the behavior error stack may induce a virtual blockage on the road ahead of it, which will cause U-turns to be created as described above. In addition, when space does not become available in the desired exit from an intersection, the Behaviors module may mark that lane as high cost to traverse, causing the robot to naturally pick an alternate route out of the intersection.

Since virtual blockages induced by the Behaviors module are not created due to something directly observed, they cannot be removed by observation either: instead, virtual blockages are removed from the model whenever a checkpoint is achieved. This guarantees forward progress.

**Blocked Intersections**

Blockages across intersections complicate the picture. Intersections have complex shapes and it is not easy to tell how the blockage affects which entry lanes are connected to which exit lanes. Blockages in intersections are therefore handled by allowing the robot to attempt to plan through to its desired exit from the intersection. If it fails to create a plan within a defined amount of time, it declares that exit blocked by setting its successors to high cost and attempts the next exit as determined by the value function. This process continues until the robot successfully makes its way through the intersections or all alternatives have been tried. Once all alternatives have been tried and failed, the robot declares the segment leading into the intersection blocked at the entry, which causes it to turn around.

**Behaviors**

The Behaviors subsystem is responsible for achieving the waypoints specified along the path determined by the Mission Planner; making lane-change, precedence, and safety decisions respectively on roads, at intersections, and at yields; and responding to and recovering from anomalous situations. The Mission Planner provides Behaviors with a cost graph containing the cost-to-goal associated with available path. The resultant cost graph is the baseline for current road and lane decisions by Behaviors, but it can be modified locally and dynamically to account for blocked routes, slow traffic, and other anomalies.
The initial Behaviors design was based on the concept of identifying driving contexts, each of which required the vehicle software to focus on a reduced set of environmental features. At the highest level of this design, the two contexts were “Lane Driving” (structured environment), and “Intersection Handling”. These driving contexts combined with an auxiliary Goal Selection component resulted in the Behaviors architecture illustrated in Figure 13.

The depicted behavior architecture components have the following functions:

**Goal Selection Components**

*StateEstimator:* combines the vehicle’s position with the world model to produce a discrete and semantically rich representation of the vehicle’s logical position with the model.

*GoalSelector:* uses the current logical location as reported by StateEstimator to generate the next series of local goals for execution by the Motion Planner.

**Intersection Handling Components**

*PrecedenceEstimator:* uses the list of known other vehicles and their state information to determine precedence at an intersection.

*PanheadPlanner:* aims the panhead sensors to gain the most relevant information for intersection precedence decisions.

*TransitionManager:* manages the discrete-goal interface between the Behavioral Executive and the Motion Planner, using the goals from GoalSelector and the gating function from PrecedenceEstimator to determine when to transmit the next sequence of goals.

**Lane Driving Components**

*LaneSelector:* uses the surrounding traffic conditions to determine the optimal lane to be in at any instant and requests a merge into that lane if it is feasible.

*MergePlanner:* determines the feasibility of a merge into a lane proposed by LaneSelector and executes that merge when appropriate.

*CurrentSceneReporter:* The Current Scene Reporter distills the list of known vehicles and lane blockages into a few discrete data elements, most notably the distance to and velocity of the nearest vehicle in front of Boss in the current lane.
**DistanceKeeper:** uses the surrounding traffic conditions to determine the necessary in-lane vehicle safety gaps and govern the vehicle’s speed accordingly.

**VehicleDriver:** combines the outputs of DistanceKeeper and LaneSelector with its own internal rules to generate a so-called “MotionParameters” message, which governs details such as the vehicle’s speed, acceleration and desired tracking lane.

**Intersection Handling and Yielding**

The Precedence Estimator is the component most directly responsible for the system’s adherence to the DARPA Urban Challenge rules [11] concerning intersections.

To fulfill these requirements, the Precedence Estimator combines various data from the rest of the system to generate what is effectively a boolean “clear to go” state. This state is used as a gating bit in the Transition Manager and triggers the issuance of the motion goal to proceed through the intersection. The Precedence Estimator derives this bit from two primary sources: the world model and the moving obstacle set.

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

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

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

**Implementation**

Within the world model, an intersection is defined as a group of exit waypoints. That is, an intersection must contain one or more exit waypoints, and each exit waypoint will be a part of exactly one intersection. Precedence between any two exit waypoints is determined first by whether the exit waypoints are stop lines since non-stop exit waypoints automatically have precedence over stopline exit waypoints. Thereafter, precedence is determined by arrival times, where earlier arrivals have precedence over later arrivals.

The robust computation of arrival time is critical to the correct operation of the precedence

\(^1\)Yield lanes are lanes of moving traffic for which the robot must wait for a clear opportunity to execute the associated maneuver.

-20-

For DARPA internal use only
estimator. Given the dynamic, stochastic nature of the moving obstacle set, the algorithm uses a purely geometric and instantaneous notion of waypoint occupancy for computing arrival times. All movement states for objects determined by the Moving Obstacle Tracking are taken into account. An exit waypoint is considered to be occupied when any vehicle’s estimated front bumper is inside or intersects a small polygon around the waypoint, called its occupancy polygon. Boss’s front bumper is added to the pool of estimated front bumpers and is treated no differently for the purposes of precedence estimation.

These polygons are constructed for each exit waypoint and for the whole intersection. The occupancy polygons for each exit waypoint are used to determine precedence, where the occupancy polygon constructed for the intersection is used to determine whether the intersection is clear of other traffic.

Figure 14 shows a typical exit occupancy polygon extending three meters back along the lane from the stop line and with one meter of padding on all sides. This is the configuration that was used during NQE and the UCFE.

The estimated front bumper of a vehicle must be inside the occupancy polygon as shown in Figure 14 to be considered to be an occupant of that polygon.

A given occupancy polygon maintains its associated exit waypoint, its binary (Occupied) state and two pieces of temporal data: the time of first occupancy (which is used to determine precedence ordering), and the time of most recent (last) occupancy (which is used to implement a temporal hysteresis on when the polygon becomes “unoccupied”).

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

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

Beyond interacting with stopped traffic, the precedence estimator is also responsible for merging into or across moving traffic from a stop. To support this, the system maintains a *next intersection goal*, which is invariably a *virtual lane* that connects the target exit waypoint to some other waypoint, nominally in another lane or in a parking or obstacle zone. That virtual lane has an associated set of *yield lanes*, which are other lanes that must be considered for moving traffic in order to take the next intersection action. The yield lanes are computed per virtual lane using plain geometric overlap with any real lanes, with a minimum overlap area of 1m², subject to the constraint that all yield lanes must be real lanes. That is, virtual lanes that overlap other virtual lanes should have a clearly established precedence order via stop lines or else be an ill-formed intersection. Thus, yield cases shall only be considered for merges into or across real lanes. Figure 15 shows an example T-intersection, highlighting the next intersection goal and the associated yield lanes.

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

1. Compute $T_{\text{action}}$ for completing the length of next intersection goal using conservative accelerations from a starting speed of zero.
2. Compute $T_{\text{accelerate}}$ for accelerating from zero up to speed in the destination lane using the same conservative acceleration.
3. Compute $T_{\text{delay}}$ to compensate for accumulated system delays.
4. Specify $T_{\text{spacing}}$ as the minimum required temporal spacing between vehicles, where one second approximates the one-vehicle-length-per-10mph rule.

Using these values, a required temporal window, $T_{\text{required}}$, is computed for each yield lane as: $T_{\text{required}} = T_{\text{action}} + T_{\text{delay}} + T_{\text{spacing}}$. In the case of merging into a lane, the required window is extended to include the acceleration time, if necessary. This temporal window is then used to construct a polygon similar to an exit occupancy polygon backward along the road network for a distance of: $\text{length}_{\text{yield polygon}} = \text{max speed}_{\text{lane}} \times T_{\text{required}} + D_{\text{safety}}$

These yield polygons, shown in Figure 15, are used as a first pass for determining cars that are...
relevant to the yield window computations.

Any reported moving obstacle that is inside or overlaps the yield polygon is considered in the determination of the available yield window. Only obstacles with the moving flag set are taken into consideration. The yield windows are also provided to the Panhead Planner, which performs coverage optimization to point long-range sensors along the yield lanes and thus at oncoming traffic, increasing the probability of detection for these vehicles at long range.

For each such moving obstacle, the time of arrival at the near edge of the overlap area is estimated, called the crash point as follows:

1. Compute a worst-case speed \( V_{\text{obstacle}} \) along the yield lane by projecting the reported velocity vector, plus one standard deviation, onto the yield lane.
2. Compute \( D_{\text{crashpoint}} \) as the length along the road network from that projected point to the leading edge of the overlap area.
3. Compute an estimated time of arrival as \( T_{\text{arrival}} = \frac{D_{\text{crashpoint}}}{V_{\text{obstacle}}} \).
4. Retain the minimum \( T_{\text{arrival}} \) as \( T_{\text{current}} \) over all relevant vehicles per yield lane.

The yield window for the overall intersection action is considered to be instantaneously open when \( T_{\text{current}} < T_{\text{required}} \) for all yield lanes. In order to account for temporal existence uncertainties as in the exit waypoint precedence determination, this notion of instantaneous clearance is protected by a one-second temporal hysteresis. That is, all yield windows must be continuously open for at least one second before yield clearance is passed to the rest of the system.

**Gridlock Management**

With exit precedence and yield clearance in place, the third and final element of intersection handling is the detection and prevention of gridlock situations. Gridlock is determined simply as a vehicle (or other obstacle) blocking the path of travel immediately after the next intersection goal such that the completion of the next intersection goal is not immediately feasible. Gridlock management comes into effect once the system determines that Boss has precedence at the current intersection and begins with a 15-second timeout to give the obstruction an opportunity to clear. If still gridlocked after 15 seconds, the current intersection action is marked as locally high-cost and the mission planner is allowed to determine if an alternate path to goal exists. If so, Boss re-routes along that alternate path: otherwise the system jumps straight into error recovery for intersection goals, using the generalized pose planner to get around the presumed-dead vehicle.

**Lane Driving**

**Distance Keeping**

The distance-keeping behavior uses the lead vehicle information provided by the Current Scene Reporter to derive an instantaneously desired speed and acceleration that will satisfy the rules for minimum spacing in travel areas. It aims to simultaneously zero the difference between Boss’ velocity and that of the relevant vehicle in front of Boss, and the difference between the desired and actual inter-vehicle gaps. Only moving obstacles with the observed moving flag set are taken...
into consideration. The commanded velocity is governed by the equation: \( v_{\text{cmd}} = v_{\text{target}} + K_{\text{gap}}(\text{gap}_{\text{actual}} - \text{gap}_{\text{desired}}) \) where \( v_{\text{target}} \) is the target-vehicle velocity and \( K_{\text{gap}} \) is a settable gain. Experiments showed that the velocity of the target vehicle can be set to zero in the equation for the commanded velocity. The desired gap is computed as: \( \text{gap}_{\text{desired}} = \max((l_{\text{vehicle}}/10) \cdot v_{\text{cmd}}, \text{absMinGap}) \)

where \( l_{\text{vehicle}} \) is the length of a vehicle, \((l_{\text{vehicle}}/10)\) represents the one-vehicle-length-per-10-mph minimum-gap requirement, and \( \text{absMinGap} \) is the absolute minimum gap requirement. When Boss’ velocity exceeds the target vehicle’s, its deceleration is set to a single configurable default value; when Boss’ velocity is less than the target vehicle’s, for safety and smoothness, Boss’ commanded acceleration is made proportional to the difference between the commanded and actual velocities and capped at maximum and minimum values (nominal values are \( a_{\text{max}} = 4.0 \) and \( a_{\text{min}} = 1.0 \) m/sec\(^2\)) according to: \( a_{\text{cmd}} = a_{\text{min}} + K_{\text{acc}}(v_{\text{cmd}} - v_{\text{target}})(a_{\text{max}} - a_{\text{min}}) \)

It is important to note that this speed governing rule will drive the vehicle’s speed to zero on approach to a stopped car or roadblock. If the obstacle subsequently starts moving along the road, the commanded speed will increase again, resulting in a stable queuing behavior.

**Merge Planning**

The merge, or lane-change, planner determines the feasibility of changing lanes. It is relevant not only on a unidirectional multi-lane road, but also on a bidirectional two-lane road in order to handle passing a stopped vehicle after coming to a stop. Feasibility is based on the ability to maintain proper spacing with surrounding vehicles and to reach a checkpoint in the lane to merge into (the “merge-to” lane) while meeting a velocity constraint at the checkpoint. The two-lane unidirectional case is depicted in Figure 16. The merge planner performs the following steps:

1. Check whether it is possible to reach the checkpoint in the merge-to lane from the initial position given velocity and acceleration constraints and the “merge distance”, i.e., the distance required for Boss to move from its current lane into an adjacent lane. For simplicity’s sake, the merge distance was made a constant parameter.

2. Determine the merge-by-distance, i.e., the allowable distance in the current lane in order to complete the merge. This is the smallest of the distance to the: 1) next motion goal (i.e., checkpoint); 2) end of the current lane; 3) closest road blockage in the current lane; 4) projected position of the closest moving obstacle in the current lane. The merge-by-distance in the case of a moving obstacle in front of Boss is \( d_{\text{moving-obst}} = v_0d_{\text{initial}}(v_0 - v_1) \), where \( d_{\text{initial}} \) is the initial distance to the moving obstacle. Note that this reduces to \( d_{\text{initial}} \) if the obstacle is still, i.e. if \( v_1 = 0 \).

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

4. For either a front- or back-merge, first determine whether proper spacing is already met. For a front merge, this means \( x_0 - l_{\text{vehicle}} - x_1 \geq \max((v_1 \cdot l_{\text{vehicle}})/10, \text{absMinGap}) \); for a back merge, \( x_1 - l_{\text{vehicle}} - x_0 \geq \max((v_0 \cdot l_{\text{vehicle}})/10, \text{absMinGap}) \).

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

6. If proper spacing is not met, check whether it is possible and determine the acceleration profile.

For DARPA internal use only

-24-
to accelerate or decelerate respectively to and remain at either the maximum or minimum speed until proper spacing is reached.

7. Check whether it is possible to reach and meet the velocity constraint at the checkpoint in the merge-to-lane starting from the merge point, i.e., the position and velocity reached in the previous step after proper spacing has been met. If so, the merge is feasible.

8. Repeat the above steps for all $n$ obstacles in the merge-to-lane. There are $n+1$ “slots” into which a merge can take place, one each at the front and rear of the line of obstacles, the rest in between obstacles. The feasibility of the front and rear slots is associated with a single obstacle and therefore already determined by the foregoing. A “between” slot is feasible if the following criteria are met: 1) the slot’s front-obstacle back-merge and rear-obstacle front-merge are feasible; 2) the gap between obstacles is large enough for Boss plus proper spacing in front and rear; 3) the front obstacle’s velocity is greater than or equal to the rear obstacle’s velocity, so the gap is not closing; 4) the merge-between point will be reached before the checkpoint. Boss determines whether a merge is feasible in all slots in the merge-to-lane (there are three in the example in Fig. 13: in front of vehicle 1, between vehicles 1 and 2, and behind vehicle 2), and targets the appropriate feasible merge slot depending on the situation.

---

Figure 16 Two-Lane Merging

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

**Error Recovery**

One of the most important functions of the Behavioral Executive is to detect and address errors from the Motion Planner and other aberrant situations. Desirable properties of the recovery system include being able to: generate a non-repeating sequence of recovery goals in the face of repeated failures; and generate different sets of recovery goals to handle different contexts.

**Design Decisions**

Keeping external complexity (that is, of dealing with the rest of the goal selection system) to a minimum, the goal selection system follows the state graph shown in Figure 17. Each edge represents the successful completion (Success) or the failed termination (Failure) of the current goal. Goal failure can be either directly reported by the motion planner, or triggered internal to behaviors by a progress monitoring system that will force a goal failure if sufficient progress is
not made in a configurable span of time. All edge transitions trigger the selection of a new goal and modify an observable datum, RecoveryLevel, as follows: success edges reset RecoveryLevel to zero, and failure edges increment RecoveryLevel by one.

The value of RecoveryLevel along with the type and parameters of the original failed goal are the primary influences on the recovery goal algorithms. The general form of the recovery goals is that an increasing RecoveryLevel yields higher-risk recovery goals, generally farther away from the current position and/or the original goal in an attempt to keep initial recovery attempts low-risk and easy to execute in benign situations while being able to take more drastic measures in more convoluted situations.

It is important to note that the successful completion of any one recovery goal sets the system back to normal operation. This eliminates the possibility of complex multi-maneuver recovery schemes, but it was deemed that situations convoluted enough to require a more complex recovery system were beyond scope.

One limitation of this approach is that it is possible to become stuck in local minimum if a normal goal fails twice in a row with an attempted recovery in between. To address this limitation the RecoveryLevel is cached allowing the system to bypass recovery levels that have already been tried and failed to get the system out of trouble.

**Motion Planning**

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

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

**Model-Based Trajectory Generation**

A model-based trajectory generator originally presented in [12] is responsible for generating dynamically feasible actions from an initial vehicle state to a desired terminal state. In general, this algorithm can be used to solve the problem of generating a set of parameterized controls \((u(p,x))\) that satisfy state constraints \((C(x))\) whose dynamics can be expressed in the form of a set of differential equations \(\dot{x}(p,x) = f(x,u(p,x))\).
State Constraints
For navigating urban environments, we required the satisfaction of position and heading boundary state constraints. We define the constraint equation formula as the difference between the target boundary state constraints \( X_C \) and the integral of the model dynamics (the vehicle trajectory).

Vehicle Modeling
The development of a high-fidelity vehicle dynamics model is important for the accurate prediction of vehicle motion. An accurate model will properly express how the vehicle moves through space and time, an important factor for simple actions such as path tracking and complex behaviors including high-speed lane changing, obstacle avoidance in dynamic environments, and passing maneuvers. In general, there are two components to the vehicle model: the motion model and the motion simulation. The motion model consists of parameterized functions that are fit to vehicle data and consist of controller delays, the curvature limit, and the curvature rate limit. The other component of the vehicle model, the motion simulation, is an application of fixed-timestep Euler integration of the motion model.

Controls Parameterization
The parameterizations of the vehicle controls are important to the performance of the algorithm. It is important to have enough degrees of freedom in the system to satisfy all of the boundary state constraints. On the other hand, it is important to limit the number of free parameters in the system to limit the dimensionality of the search space, limit/prevent local optima, and improve the runtime performance. For Ackermann-steered vehicles, it is advantageous to define the vehicle controls with a time-based linear velocity function \( v \) and an arclength-based curvature function \( \kappa \):

\[
\mathbf{u}(\mathbf{p}, \mathbf{x}) = [v(\mathbf{p}, t) \ \ k(\mathbf{p}, s)]
\]

The linear velocity profile takes the form of a constant profile, linear profile, linear ramp profile, or a trapezoidal profile (see Figure 18). The motion planner selects the appropriate parameterization for particular applications (including parking and distance keeping). Each of these profiles consists of dependent parameters \( (v_0, v_t, v_f, a_0, a_f) \) and the time to complete the profile \( (t_f) \). Since all of the dependent profile parameters are typically known, no optimization is done on the shape of each of these profiles: only the time to complete the profile is optimized.

Several different linear velocity profiles were applied in this system, each with its own parameterization.
and application. Each parameterization contains some subset of velocity and acceleration knot points \((v_0, v_t, v_f, a_0, a_f)\) and the length of the path, measured in time \((t_f)\).

The curvature profile defines the shape of the trajectory, as it is the primary profile upon which the optimization is executed. The profile consists of three dependent parameters \((\kappa_0, \kappa_1, \text{ and } \kappa_2)\) and the trajectory length \((s_f)\). A second-order spline profile was chosen because it contains enough degrees of freedom \((4)\) to satisfy the boundary state constraints \((3)\). The initial command knot point, \(\kappa_0\), is fixed during the optimization process to a value that generates a \textit{smooth} or \textit{sharp} trajectory (as discussed later).

One type of curvature profile was applied in this system. The parameterization includes four degrees of freedom, the three spline knot points \((\kappa_0, \kappa_1, \text{ and } \kappa_2)\) and the length of the path, measured in distance \((s_f)\):

\[
\mathbf{p} = \begin{bmatrix} \kappa_1 \\ \kappa_2 \\ s_f \end{bmatrix}
\]

With the linear velocity profile’s dependent parameters being fully defined and the initial spline parameter of the curvature profile fixed to produce smooth or sharp trajectories, what remains is a system with three parameterized freedoms: the latter two curvature spline knot points and the trajectory length. The duality of the trajectory length \((s_f)\) and time \((t_f)\) can be resolved by estimating the time that it takes to drive the entire distance through the linear velocity profile. Time was used for the independent variable of the linear velocity profile because of the simplicity of computing profiles defined by accelerations (such as the linear ramp and trapezoidal profiles), while arclength was used for the curvature profile because of the relative invariance of solutions to the speed at which it was executed.

\textit{Initialization Function}

As the vehicle model parameters do not adapt to online information, a mapping of state space to input space is pre-computed to seed the optimization process. This mapping enables high performance of the algorithm by placing the initial guess of the control parameters close to the actual solution. Given the high number of degrees of freedom and the non-integrable model dynamics, it is infeasible to pre-compute the entire mapping of state space to input space for any nontrivial system, such as our vehicle model. Instead an approximation of this mapping is generated through a five-dimensional lookup table with varying position, relative heading, initial curvatures, and constant velocities. A trajectory optimization step is then required to account for the error in table interpolation and the constant-velocity assumption (for more complex profiles).

\textit{Trajectory Optimization}

To improve the initial lookup table trajectory, the system of equations is linearized and inverted in order to produce a correction factor for the free control parameters based on the product of the inverted Jacobian and the current boundary state error. The Jacobian is model-invariant because it is determined numerically through central differences of simulated vehicle actions.

\[
\Delta \mathbf{p} = - \left( \frac{\partial \Delta \mathbf{x}_c(\mathbf{p})}{\partial \mathbf{p}} \right)^{-1} \Delta \mathbf{x}_c(\mathbf{p})
\]

The control parameters are modified until the residual of the boundary state constraints is within acceptable bounds or until the optimization diverges. There are situations where divergence is expected, as the boundary states that the trajectory generator is attempting to connect are
infeasible (such as on the edges of curvature or curvature rate limits). The resulting trajectory is returned as the best estimate and is evaluated by the motion planner, as discussed below.

On-road Navigation

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

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

The goals are placed at a fixed longitudinal distance down the centerline path but vary in lateral offset from the path to provide several options for the planner. The trajectory generation algorithm described above is used to compute dynamically feasible trajectories to these local goals. For each goal, two trajectories are generated: a smooth trajectory and a sharp trajectory. The smooth trajectory has the initial curvature parameter fixed to the curvature of the forwards-predicted vehicle state. The sharp trajectory has the initial curvature parameter set to an offset value from the forwards-predicted vehicle state to produce a sharp initial action. The velocity profile used for each of these trajectories is computed based on several factors, including: the maximum velocity bound given from the behaviors layer based on safe following distance to the lead vehicle, the speed limit of the current road segment, the maximum velocity feasible given the curvature of the centerline path, and the desired velocity at the goal (e.g. if it is a stopline).

![Figure 20 Smooth and Sharp Trajectories](image)

Figure 20 provides an example of smooth and sharp trajectories. The leftmost image shows two trajectories (cyan and purple) generated to the same goal pose. The purple (smooth) trajectory exhibits continuous curvature control throughout; the cyan (sharp) trajectory begins with a discontinuous jump in curvature control, resulting in a sharp response from the vehicle. In these images, the initial curvature of the vehicle is shown by the short pink arc. The four center images show the individual sharp and smooth trajectories, along with the convolution of the vehicle along these trajectories. The rightmost image illustrates how these trajectories are generated in practice for following a road lane.
The resulting set of trajectories is then evaluated against proximity to static and dynamic obstacles in the environment, as well as distance from the centerline path, smoothness, and various other metrics. The best trajectory according to these metrics is selected and executed by the vehicle. Since the trajectory generator computes the feasibility of each trajectory using an accurate vehicle model, the selected trajectory can be directly executed by a vehicle controller.

Figure 21 provides an example of the local planner following a road lane. Figure 21a shows the vehicle navigating down a two-lane road (lane boundaries shown in blue, current curvature of the vehicle shown in pink, minimum turning radius arcs shown in white) with a vehicle in the oncoming lane. Figure 21b shows the extracted centerline path from the desired lane (in red). Figure 21c shows trajectories generated by the vehicle given its current state and the centerline path and lane boundaries. Figure 21d shows the evaluation of one of these trajectories against both static and dynamic obstacles in the environment, and Figure 21f shows this trajectory being selected for execution by the vehicle.

**Zone Navigation**

During zone navigation, the motion goal from behaviors is a pose within a zone (such as a parking spot). The motion planner attempts to generate a trajectory that moves the vehicle towards this goal pose. However, unlike driving down a road, there are no driving lanes and thus the movement of the vehicle is far less constrained.

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

To efficiently generate complex plans over large, obstacle-laden environments, the planner relies on an anytime, replanning search algorithm known as Anytime D*, developed by Likhachev et al. [13]. Anytime D* quickly generates an initial, suboptimal plan for the vehicle and then improves the quality of this solution while deliberation time allows. At any point in time, Anytime D* provides a provable upper bound on the sub-optimality of the plan. When new information concerning the environment is received (for instance, a new static or dynamic obstacle is observed), Anytime D* is able to efficiently repair its existing solution to account for the new information. This repair process is expedited by performing the search in a backwards direction, such that updated information in the vicinity of the vehicle affects a smaller portion of the search space and so fewer repairs are required.

To scale to very large zones (up to 0.5 km by 0.5 km), the planner uses a multi-resolution search and action space. In the vicinity of the goal and vehicle, where very complex maneuvering may be required, the search considers states of the vehicles with 32 uniformly spaced orientations. It also uses a dense set of actions that enable the vehicle to transition in between these states. Figure 22 (a) shows the dense set of actions used for two vehicle states. Both states have \( x=0,y=0, \theta = 0 \) but \( v \) is positive for one state and negative for the second. The actions that move the vehicle forward (to the right) correspond to the vehicle state with a positive velocity. The actions that move the vehicle backward (to the left) correspond to the vehicle state with a negative velocity. In the areas that are not in the vicinity of the goal or a vehicle, the search considers only the states of the vehicle with 16 uniformly spaced orientations. It also uses a sparse set of actions that enable the vehicle to transition in between these states. Figure 22 (b) shows the sparse set of actions emanating for the same states as in Figure 22 (a). Because coarse and dense resolution variants both share the same dimensionality and, in particular, have 16 orientations in common, they seamlessly interface with each other and the resulting solution paths overlapping both coarse and dense areas of the space are smooth and feasible.

![Figure 22](image)

*Figure 22 Action spaces used by the planner. The spaces shown are for the vehicle with orientation = 0. For other vehicle states, these action spaces are rotated by the vehicle orientation. (a) is used to connect states with 32 possible orientations (b) is used to connect states with 16 possible orientations.*

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

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

Figure 23 illustrates the replanning capability of the lattice planner. These images were
taken from a parking task performed during the National Qualification Event (the top-left image
shows the zone in green and the neighboring roads in blue). The top-right image shows the
initial path planned for the vehicle to enter the parking spot indicated by the white triangle.
Several of the other spots were occupied by other vehicles (shown as rectangles of varying colors), with detected
obstacles shown as red areas. The trajectories generated to follow the path are shown emanating
from the vehicle. As the vehicle gets closer to its intended spot, it observes more of the vehicle
parked in the right-most parking spot (bottom-left image). At this point, it realizes its current
path is infeasible and replans a new path that has the vehicle perform a loop and pull in
smoothly. This path was favored in terms of time over stopping and backing up to re-position.

The lattice planner is flexible enough to be used in a large variety of cases that can occur during
on-road and zone navigation. In particular, it is used during error recovery when navigating
congested intersections, to perform difficult U-turns, and to get the vehicle back on track after
emergency defensive driving maneuvers. In these error recovery scenarios the lattice planner is
biased to avoid areas that could result in unsafe behavior (such as oncoming lanes when on
roads).
Testing

Tartan Racing developed a rigorous 18-month test program utilizing several feedback mechanisms. With two robots and multiple test sites that spanned three states, the team uncovered key interrelationships between the system’s hardware and software that enabled Boss to win the Urban Challenge. Testing focused on satisfying the critical explicit and implicit requirements. These were derived from the DARPA rules, technical interchanges, internal team analysis, and considerable experience in past competitions. Highlights of the testing program included attention to safety, a strong simulation presence, weekly capability demonstrations, and many endurance runs. The result of testing was a gap analysis that proved especially agile in the final weeks before the national qualification tests.

Cyclic System Development

The Urban Challenge competition required teams to have robust and reliable software. As such, testing was integral to Tartan Racing’s software lifecycle. The model was cyclic and shown in its generalized form in Figure 24:

The Tartan Team’s Cyclic Development Model is patterned after Grady Booch’s 1994 admonition [14] “Design a little, code a little, test a little approach is far superior [to a waterfall approach] In this manner we seek to correct design and coding errors in a much more timely fashion.”

Test Case Design

The design of test cases originated from the team’s requirements document and manifested itself as an extensive list of test scenarios. The scenarios had unique identifiers along with descriptions of success and a rating for the scenario’s probability of being encountered in the Urban Challenge. Collections of scenarios formed a playbook for system test purposes. These were field documents that test team members used to synchronize their interactions with the robot. Each collection within the playbook formed standardized runs that were designed to prove previously tested capabilities. The order of standardized runs focused on increasingly complex capabilities. A typical order of events for regressive testing follows: Vehicle inspection, Navigation, Parking Lot Operations, Intersection Precedence, Traffic and Obstacles.

System Testing Process

A requirement was satisfied by a successful run of a play (scenario) or by a ruling from the Director of Technology when post-test analysis was required to determine success or failure. In addition, the team performed regression testing. Every Friday the robot system had to perform its basic functions plus any already verified capabilities. Performance feedback after each system test was by presentation and an engineering report. The first “Hot Wash” report was published within hours of the test by the robot’s software operator to the entire software development team. The test leader published a report within 48 hours. Whereas the software operator gave first-hand knowledge of code performance, the test leader’s report provided an understanding of how well the system met its mission requirements. Software bugs were electronically tracked and reviewed
weekly. Hardware incidents were logged in a separate incident database and reviewed in the weekly team meeting for root cause and progress toward closure. The test leader published a “gap analysis” report showing the requirements that remained to be verified by test. Essentially the gap analysis document provided a measure of the Team’s readiness to compete at the NQE and UCFE.

**Test Summary Statistics**

Several feedback measurements were helpful during system development. Mission readiness was a direct function of meeting the implicit and explicit requirements. Another readiness measure was the number of failures during a robot’s endurance run. The endurance tests typically lasted 6 hours or 60 miles, the expected parameters of the Urban Challenge event. An indirect performance measurement was the number of software fixes in a given period. The team also tracked the number of hardware failures, the number of engine and computing hours, and the number of autonomous miles driven.

There were 225 requirements in the System Requirements Document. Of these, 23 were implicit and 202 were explicit. All requirements were satisfied either by demonstration or analysis as of one week before the NQE.

The playbook that was generated from these requirements detailed 216 scenarios. 207 of them had been tested with a majority of the untested “plays” pertaining to lane detection with various problematic textures and markings.

From February, 2007 until NQE, the team performed 65 days of formal tests. Combined, the team’s two robots exceeded 7,250 miles through development and testing. The total formal testing miles for February, 2007 was less than 10 autonomous miles while total formal test miles for the first three weeks of October, 2007 exceeded 1,000 autonomous miles.

**Site Visit**

Boss performed well at the June site visit evaluation. Overall the team was very happy with the performance and noted only two defects, as described below:

<table>
<thead>
<tr>
<th>Location:</th>
<th>Site Visit Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result:</td>
<td>Boss should have yielded for a vehicles stopped near a stop line</td>
</tr>
<tr>
<td>Actual Result:</td>
<td>Boss ignored the vehicle, determining that it was not within one meter of the stop line (as required by rule) and proceeded safely through the intersection</td>
</tr>
<tr>
<td>Subsystem:</td>
<td>Behaviors</td>
</tr>
<tr>
<td>Analysis:</td>
<td>Boss correctly followed the rules as specified in the TEC but did not correctly assess the intent of the vehicle stopped near the line. After this event the occupancy polygon was extended from 1.5m to approximate 3.5m back from the stop line, reducing the likelihood of a similar event.</td>
</tr>
</tbody>
</table>

Location: Site Visit Course
<table>
<thead>
<tr>
<th>Expected Result</th>
<th>Actual Result</th>
<th>Subsystem</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boss should have turned right to complete the site visit test</td>
<td>Boss completed extra loops around the site visit course</td>
<td>Behaviors</td>
<td>During the test Boss was paused as it rolled over a mission checkpoint, when in pause the behavioral system did not count the checkpoint as achieved and thus Boss had to go back around to get it. As a result, the system was modified to consider checkpoints achieved if Boss drove over them in pause, but not manual mode.</td>
</tr>
</tbody>
</table>

---

**National Qualification Event**

The Tartan Racing team identified three primary incidents across five NQE test runs. The following charts detail each incident.

<table>
<thead>
<tr>
<th>Location</th>
<th>Expected Result</th>
<th>Actual Result</th>
<th>Subsystem</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>NQE / Area A October 28, 2007 / Commissary loop bottom. (5-15 mph)</td>
<td>Boss drives around unobstructed bend in road with traffic in oncoming lane.</td>
<td>Boss stops abruptly before the bend for 20 seconds before continuing.</td>
<td>Perception + Planning + Pre-race planning</td>
<td>Rough interpolation of GPS waypoints result in moving obstacles in other lanes and Jersey barriers lining road to appear to be in Boss' lane, causing it to stop and plan carefully.</td>
</tr>
<tr>
<td>NQE / Area C October 29, 2007 / Sheppard Lane. (10-10 mph)</td>
<td>Boss encounters stop-sign barrier, makes U-turn and continues with new mission plan.</td>
<td>Boss freezes for a minute in mid-turn.</td>
<td>Perception</td>
<td>Dust in front of Boss is interpreted as a lethal obstacle and enters Boss' blind spot so it cannot be cleared away. A tree behind Boss prevents it from backing up to finish the turn. The dust lethals are eventually cleared.</td>
</tr>
<tr>
<td>NQE / Area B October 29, 2007 / Washington St. (10-20 mph)</td>
<td>Boss drives down road with parked cars and other passable obstacles.</td>
<td>Boss drives backwards 30 meters and is put in pause mode by DARPA.</td>
<td>Behaviors</td>
<td>Boss enters an error recovery state while trying to plan past parked cars on the road. The error recovery state was not properly reset, so Boss plans back to a previous error recovery goal behind it.</td>
</tr>
</tbody>
</table>
## Final Event

The Tartan Racing team identified four significant incidents across all three missions upon post-race analysis of the Final Event footage. During the entire final event, Boss drove 54.3 miles at an average speed of 14 mph. Had these incidents not occurred, Boss would have driven a total of 52.8 miles. The following charts detail each incident.

<table>
<thead>
<tr>
<th>Location:</th>
<th>Race / Mission 1 / Outback Rd. (5-7 mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result:</td>
<td>Driving toward exit of Outback Rd. Wait for traffic to clear, when safe, turn onto Phantom Rd.</td>
</tr>
<tr>
<td>Actual Result:</td>
<td>Waited for several seconds, no traffic present. After excessive delay, Boss turned onto Phantom Rd.</td>
</tr>
<tr>
<td>Subsystem:</td>
<td>Perception</td>
</tr>
<tr>
<td>Analysis:</td>
<td>At the bottom of the dirt road, Boss pitched 6 degrees causing sensors to see the ground and false obstacles were placed into the static obstacle map. Local Planner reported “blocked”, Planner 3D moved vehicle slightly, false obstacles were cleared, Boss resumed normal operations and pulled onto Phantom Rd.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location:</th>
<th>Race / Mission 1 / Sabre Blvd. (10-20 mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result:</td>
<td>Boss passes the IVST chase car.</td>
</tr>
<tr>
<td>Actual Result:</td>
<td>Boss stopped for the chase car, and waited for several minutes before continuing the mission.</td>
</tr>
<tr>
<td>Subsystem:</td>
<td>Perception</td>
</tr>
<tr>
<td>Analysis:</td>
<td>The IVST chase car was momentarily detected to be in Boss’ lane by the moving obstacle detector, when in fact the chase car was not in Boss’ lane. As a result, the Local Planner reported “Blocked”, Boss stopped, and the Planner 3D failed to find a plan. While behaviors attempted to issue different motion goals to the planner, Boss regained its lock on HP, causing its position to move by 2cm, shifting the vehicle’s position with respect to the static obstacle map just enough such that a plan could be safely executed. Boss successfully resumed the mission.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location:</th>
<th>Race / Mission 1 / Texas Ave. (10-25 mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result:</td>
<td>Boss is queued at intersection, as vehicle in front of Boss travels through intersection, Boss should move to stop line, wait for precedence, and continue through the intersection.</td>
</tr>
<tr>
<td>Actual Result:</td>
<td>As the vehicle in front of Boss moved through the intersection, Boss never proceeded to the stop line. After an excessive delay, Boss performed a U-Turn, away from the intersection.</td>
</tr>
</tbody>
</table>
### Subsystem: Perception + Planning

#### Analysis:

Local Planner reports blocked due to the slight delay in removing lethal cells from the static obstacle map due to moving obstacles, in this case generated from the lead vehicle.

Planner 3D is requested, behaviors issues goal right on top of lead vehicle → goal is invalid, so Local Planner does not invoke the 3D planner.

Lead vehicle moves through intersection.

Local Planner does not update its moving obstacle set until just before trajectory evaluation (this is a bug) – and since the initial goal was invalid, and since the Local Planner has a stale copy of the moving obstacle set, all subsequent goals are also invalid, so Boss does not move.

While the Behaviors recovery stack had little freedom to try different goals this close to an intersection, it did eventually detect no-progress, declared a virtual (inferred) blockage, and forced a U-turn.

Boss continued the mission.

---

### Lessons Learned

Through the development of Boss and the Urban Challenge competition, we learned several lessons:

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

* **Road shape estimation may be replaced by estimating position relative to road** - In urban environments the shape of roads change infrequently. There may be local anomalies (e.g. a stopped car or construction) but in general a prior model of road shape can be used for on-road navigation. Several urban challenge teams took this approach, including Tartan Racing, and demonstrated that it was feasible on a small to medium scale. The next step will be to apply the same approach on a large or national scale.

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

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

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

Sliding Autonomy will reduce complexity of autonomous vehicles- In building a system that was able to recover from a variety of failure cases we introduced significant system complexity. In general Boss was able to recover from many failure modes but took considerable time to do so. If instead of attempting an autonomous recovery the vehicle were to request assistance from a human controller, much of the system complexity would be reduced and the time taken to recover from faults would decrease dramatically. The critical balance here is to ensure the vehicle is sufficiently capable that it does not request help so frequently that the benefits of autonomy are lost. In the case of the Urban Challenge, Boss may have requested help three times over the four hour period, an operator may have required somewhat less than a minute of time per incident and would likely have reduced Boss’ overall mission time by on order of fifteen minutes.

Driving is a social activity- Human driving is a social activity consisting of many subtle, and some not-so-subtle cues. Drivers will indicate their willingness for other vehicles to change lanes by varying their speed, and the gap between themselves and another vehicle, by small amounts. At other times it is necessary to interpret hand gestures and eye-contact in situations when the normal rules of the road are violated, or need to be violated for traffic to flow smoothly and efficiently. For autonomous vehicles to seamlessly integrate into our society they would need to be able to interpret these gestures.

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

The dream of the autonomous automobile is near- The Urban Challenge has generated a generation of autonomous vehicles which, while clearly imperfect, are approaching the capabilities needed for a consumer vehicle to drive autonomously. While it is unlikely that a fully autonomous vehicle will be sold in the next decade, piece components are already making it to market in the automotive industry, and there will be more autonomy features in the years to come. In more isolated industries it is likely that we will see autonomous vehicles operating in controlled, but “real world” settings within the next five years.
Conclusions

The Urban Challenge was a tremendously exciting program to be a part of. The aggressive technology development timeline, international competition, and compelling defense and industrial motivations fostered an environment that brought out the best in both our team and the rest of the field.

The development of Boss generated many innovations:

- a coupled moving obstacle and static obstacle detection and tracking system;
- a road navigation system that combines road localization and road shape estimation to enable Boss to drive both on roads it knows about in advance and roads where prior knowledge of road geometry is unavailable;
- a mixed mode planning system that is able to both efficiently navigate on roads and safely maneuver through open areas and parking lots;
- a behavioral engine that is capable of both following the rules of the road and violating them when necessary;
- a development and testing methodology that enables us to quickly develop and test highly capable autonomous vehicles.

While this paper outlines the technology that made Boss capable of meeting the challenge, there is much left to do. Urban environments are considerably more complicated than what the vehicles faced in the urban challenge; pedestrians, traffic lights, varied weather, and dense traffic make the general driving considerably more difficult than what was encountered in the Urban Challenge. As the field advances to address these problems, we will be faced with the secondary problems such as how do we test these systems and how will society accept them?. In some regards the military need may provide the inertia necessary to help pull these promising technologies along before broader society is willing to accept them. While these challenges loom large, it is clear that there is a bright and non-too-distant future for autonomous vehicles.

Acknowledgements

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

References


