Autonomous Driving in Urban Environments: Boss and the Urban Challenge

Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William "Red" Whittaker, Ziv Wolkowicki, and Jason Ziglar
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
e-mail: curmson@ri.cmu.edu

Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, and Wende Zhang
General Motors Research and Development
Warren, Michigan 48090

Joshua Struble and Michael Taylor
Caterpillar, Inc.
Peoria, Illinois 61656

Michael Darms
Continental AG
Auburn Hills, Michigan 48326

Dave Ferguson
Intel Research
Pittsburgh, Pennsylvania 15213

Received 22 February 2008; accepted 19 June 2008

Journal of Field Robotics 25(8), 425–466 (2008)
Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/rob.20255

Boss is an autonomous vehicle that uses on-board sensors (global positioning system, 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 drive in urban environments. The mission planning layer considers which street to take to achieve a mission goal. The behavioral layer determines when to change lanes and precedence at intersections and performs error recovery maneuvers. The motion planning layer selects actions to avoid obstacles while making progress toward local goals. The system was developed from the ground up to address the requirements of the DARPA Urban Challenge using a spiral system development process with a heavy emphasis on regular, regressive system testing. During the National Qualification Event and the 85-km Urban Challenge Final Event, Boss demonstrated some of its capabilities, qualifying first and winning the challenge.

1. INTRODUCTION

In 2003 the Defense Advanced Research Projects Agency (DARPA) announced the first Grand Challenge. The goal was to develop autonomous vehicles capable of navigating desert trails and roads at high speeds. The competition was generated as a response to a congressional mandate that a third of U.S. military ground vehicles be unmanned by 2015. Although there had been a series of ground vehicle research programs, the consensus was that existing research programs would be unable to deliver the technology necessary to meet this goal (Committee on Army Unmanned Ground Vehicle Technology, 2002). DARPA decided to rally the field to meet this need.

The first Grand Challenge was held in March 2004. Though no vehicle was able to complete the challenge, a vehicle named Sandstorm went the farthest, setting a new benchmark for autonomous capability and providing a template on how to win the challenge (Urmson et al., 2004). The next year, five vehicles were able to complete a similar challenge, with Stanley (Thrun et al., 2006) finishing minutes ahead of Sandstorm and H1ghlander (Urmson et al., 2006) to complete the 244-km race in a little under 7 h.

After the success of the Grand Challenges, DARPA organized a third event: the Urban Challenge. The challenge, announced in April 2006, called for autonomous vehicles to drive 97 km through an urban environment, interacting with other moving vehicles and obeying the California Driver Handbook. Interest in the event was immense, with 89 teams from around the world registering interest in competing. The teams were a mix of industry and academics, all with enthusiasm for advancing autonomous vehicle capabilities.

To compete in the challenge, teams had to pass a series of tests. The first was to provide a credible technical paper describing how they would implement a safe and capable autonomous vehicle. Based on these papers, 53 teams were given the opportunity to demonstrate firsthand for DARPA their ability to navigate simple urban driving scenarios including passing stopped cars and interacting appropriately at intersections. After these events, the field was further narrowed to 36 teams who were invited to participate in the National Qualification Event (NQE) in Victorville, California. Of these teams, only 11 would qualify for the Urban Challenge Final Event (UCFE).

This article describes the algorithms and mechanism that make up Boss (see Figure 1), an autonomous vehicle capable of driving safely in traffic at speeds up to 48 km/h. Boss is named after Charles "Boss" Kettering, a luminary figure in the automotive industry, with inventions as wide ranging as the all-electric starter for the automobile, the coolant Freon, and the premature-infant incubator. Boss was developed by the Tartan Racing Team, which was composed of students, staff, and researchers from several entities, including Carnegie Mellon University, General Motors, Caterpillar, Continental, and Intel. This article begins by describing the autonomous vehicle and sensors and then moves on to a discussion of the algorithms and approaches that enabled it to drive autonomously.

The motion planning subsystem (described in Section 3) consists of two planners, each capable of avoiding static and dynamic obstacles while achieving a desired goal. Two broad scenarios are considered: structured driving (road following) and unstructured driving (maneuvering in parking lots). For structured driving, a local planner generates trajectories to avoid obstacles while remaining in its

Figure 1. Boss, the autonomous Chevy Tahoe that won the 2007 DARPA Urban Challenge.

lane. For unstructured driving, such as entering/exiting a parking lot, a planner with a four-dimensional search space (position, orientation, direction of travel) is used. Regardless of which planner is currently active, the result is a trajectory that, when executed by the vehicle controller, will safely drive toward a goal.

The perception subsystem (described in Section 4) processes and fuses data from Boss's multiple sensors to provide a composite model of the world to the rest of the system. The model consists of three main parts: a static obstacle map, a list of the moving vehicles in the world, and the location of Boss relative to the road.

The mission planner (described in Section 5) computes the cost of all possible routes to the next mission checkpoint given knowledge of the road network. The mission planner reasons about the optimal path to a particular checkpoint much as a human would plan a route from his or her current position to a destination, such as a grocery store or gas station. The mission planner compares routes based on knowledge of road blockages, the maximum legal speed limit, and the nominal time required to make one maneuver versus another. For example, a route that allows a higher overall speed but incorporates a U-turn may actually be slower than a route with a lower overall speed but that does not require a U-turn.

The behavioral system (described in Section 6) formulates a problem definition for the motion planner to solve based on the strategic information provided by the mission planner. The behavioral subsystem makes tactical decisions to execute the mission plan and handles error recovery when there are problems. The behavioral system is roughly divided into three subcomponents: *lane driving*, *intersection handling*, and *goal selection*. The roles of the first two subcomponents are self-explanatory. Goal selection is responsible for distributing execution tasks to the other behavioral components or the motion layer and for selecting actions to handle error recovery.

The software infrastructure and tools that enable the other subsystems are described in Section 7. The software infrastructure provides the foundation upon which the algorithms are implemented. Additionally, the infrastructure 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. A run-time execution framework is provided that wraps around algorithms and provides interprocess communication, access to configurable parameters, a common clock, and a host of other utilities.

Testing and performance in the NQE and UCFE are described in Sections 8 and 9. During the development of Boss, the team put a significant emphasis on evaluating performance and finding weaknesses to ensure that the vehicle would be ready for the Urban Challenge. During the qualifiers and final challenge, Boss performed well, but made a few mistakes.

Despite these mistakes and a very capable field of competitors, Boss qualified for the final event and won the Urban Challenge.

2. BOSS

Boss is a 2007 Chevrolet Tahoe modified for autonomous driving. Modifications were driven by the need to provide computer control and also to support safe and efficient testing of algorithms. Thus, modifications can be classified into two categories: those for automating the vehicle and those that made testing either safer or easier. 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 third-row seats and cargo area were replaced with electronics racks, the steering was modified to remove excess compliance, and the brakes were replaced to allow faster braking and reduce heating.

Boss maintains normal human driving controls (steering wheel and brake and gas pedals) so that a safety driver can quickly and easily take control during testing. Boss has its original seats in addition to a custom center console with power and network outlets, which enable developers to power laptops and other accessories, supporting longer and more productive testing. A welded tube roll cage was also installed to protect human occupants in the event of a collision or rollover during testing. For unmanned operation a safety radio is used to engage autonomous driving, pause, or disable the vehicle.

Boss has two independent power buses. The stock Tahoe power bus remains intact with its 12-V dc battery and harnesses but with an upgraded high-output alternator. An auxiliary 24-V dc power system provides power for the autonomy hardware. The auxiliary system consists of a belt-driven alternator that charges a 24-V dc battery pack that is inverted to supply a 120-V ac bus. Shore power, in the form of battery chargers, enables Boss to remain fully powered when in the shop with the engine off. Thermal control is maintained using the stock vehicle air-conditioning system.

For computation, Boss uses a CompactPCI chassis with 10 2.16-GHz Core2Duo processors, each with 2 GB of memory and a pair of gigabit Ethernet ports. Each computer boots off of a 4-GB flash drive, reducing the likelihood of a disk failure. Two of the machines also mount 500-GB hard drives for data logging. Each computer is also time synchronized through a custom pulse-per-second adaptor board.

Boss uses a combination of sensors to provide the redundancy and coverage necessary to navigate safely in an urban environment. Active sensing is used predominantly, as can be seen in Table I. The decision to emphasize active sensing was primarily due to the team's skills 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. The configuration of sensors on Boss is illustrated in Figure 2. One of the novel aspects of this sensor configuration is the pair of pointable sensor pods located above the driver and front passenger doors. Each pod contains an ARS 300 radar and ISF 172 LIDAR. By pointing these pods, Boss can adjust its field of regard to cover crossroads that may not otherwise be observed by a fixed-sensor configuration.

3. 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 through 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 toward the desired goal and then tracks this path by generating a set of candidate trajectories that follow the path to various 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. For more details on all aspects of the motion planner, see Ferguson, Howard, and Likhachev (2008, submitted).

3.1. Trajectory Generation

A model-predictive trajectory generator originally presented in Howard and Kelly (2007) is responsible for generating dynamically feasible actions from an initial state $\mathbf{x}$ to a desired terminal state. In general, this algorithm can be applied to solve the problem of generating a set of parameterized controls $\mathbf{u}(\mathbf{p},\mathbf{x})$

Table I. Description of the sensors incorporated into Boss.

Sensor	Characteristics
Applanix POS-LV 220/420 GPS/IMU (APLX)	• Submeter accuracy with Omnistar VBS corrections • Tightly coupled inertial/GPS bridges GPS outages
SICK LMS 291-S05/S14 LIDAR (LMS)	• 180/90 deg × 0.9 deg FOV with 1/0.5-deg angular resolution • 80-m maximum range
Velodyne HDL-64 LIDAR (HDL)	• 360 × 26-deg FOV with 0.1-deg angular resolution • 70-m maximum range
Continental ISF 172 LIDAR (ISF)	• 12 × 3.2 deg FOV • 150-m maximum range
IBEO Alasca XT LIDAR (XT)	• 240 × 3.2 deg FOV • 300-m maximum range
Continental ARS 300 Radar (ARS)	• 60/17 deg × 3.2 deg FOV • 60-m/200-m maximum range
Point Grey Firefly (PGF)	• High-dynamic-range camera • 45-deg FOV

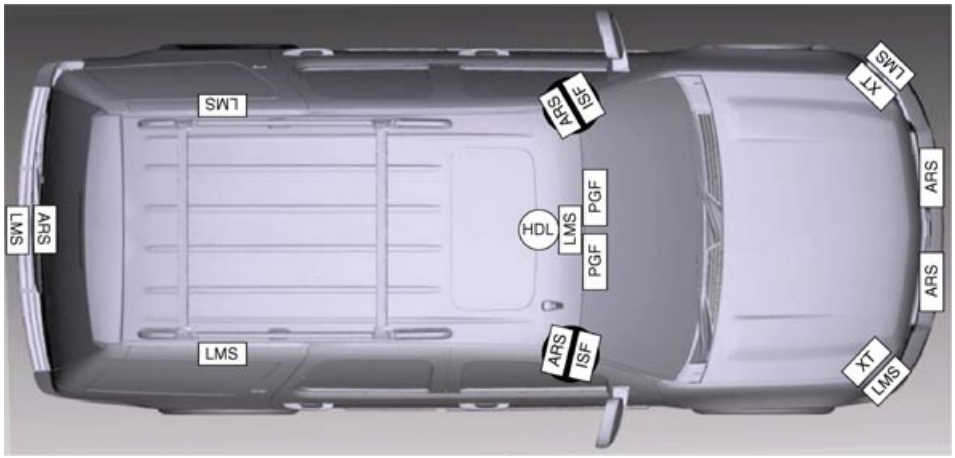

Figure 2. The mounting location of sensors on the vehicle; refer to Table I for abbreviations used in this figure.

that satisfy state constraints **C(x)** whose dynamics can be expressed in the form of a set of differential equations **f**:

$$\dot{\mathbf{x}} = \mathbf{f}[\mathbf{x}, \mathbf{u}(\mathbf{p}, \mathbf{x})]. \tag{1}$$

To navigate urban environments, position and heading terminal state constraints are typically required to properly orient a vehicle along the road. The constraint equation $\mathbf{x}_C$ is the difference between the target terminal state constraints and the integral of the model dynamics:

$$\mathbf{x}_C = [x_C \;\; y_C \;\; \theta_C]^T, \tag{2}$$

$$\mathbf{C}(\mathbf{x}) - \mathbf{x}_C - \int_0^{t_f} \dot{\mathbf{x}}(\mathbf{x}, \mathbf{p})\, \mathrm{d}t = 0. \tag{3}$$

The fidelity of the vehicle model directly correlates to the effectiveness of a model-predictive planning approach. The vehicle model describes

the mapping from control inputs to state response (changes in position, orientation, velocity, etc.). Selecting an appropriate parameterization of controls is important because it defines the space over which the optimization is performed to satisfy the boundary state constraints.

The vehicle model used for Boss combines a curvature limit (the minimum turning radius), a curvature rate limit (a function of the maximum speed at which the steering wheel can be turned), maximum acceleration and deceleration, and a model of the control input latency. This model is then simulated using a fixed-timestep Euler integration to evaluate the constraint equation.

The control inputs are described by two parameterized functions: a time-based linear velocity function v_{cmd} and an arc-length-based curvature function κ_{cmd}:

$$\mathbf{u}(\mathbf{p}, \mathbf{x}) = [v_{\text{cmd}}(\mathbf{p}, \mathbf{t}) + \kappa_{\text{cmd}}(\mathbf{p}, \mathbf{s})]^T. \tag{4}$$

The linear velocity profile takes the form of a constant profile, linear profile, linear ramp profile, or a trapezoidal profile (Figure 3). The local motion planner selects the appropriate parameterization for particular applications (such as parking and distance keeping).

The response to the curvature command function by the vehicle model defines the shape of the trajectory. The profile consists of three dependent parameters (κ_0, κ_1, and κ_2) and the trajectory length s_f. A second-order spline profile was chosen because it contains enough degrees of freedom (four) to satisfy the boundary state constraints (three). The initial spline knot point κ_0 is fixed during the optimization process to a value that generates a *smooth* or *sharp* trajectory and will be discussed later:

$$p_{\text{free}} = [\kappa_1 \;\; \kappa_2 \;\; s_f]^T. \tag{5}$$

As described, the system retains three parameterized freedoms: two curvature command spline knot points (κ_1, κ_2) and the trajectory length s. 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 for the linear velocity command function because of the simplicity of computing profiles defined by accelerations (linear ramp and trapezoidal profiles). Arc length was used for the curvature command function because the trajectory shape is less dependent on the speed at which trajectories are executed.

Given the three free parameters and the three constraints in our system, we can use various optimization techniques to solve for the parameter values that minimize our constraint equation. An initial estimate of the parameter values is defined using a precomputed approximate mapping from state space to parameter space in a lookup table. The parameter estimates are iteratively modified by linearizing and inverting the differential equations describing the equations of motion. A correction factor is generated by taking the product of the inverted Jacobian and the boundary state constraint error. The Jacobian is model invariant because it is determined numerically through central differences of simulated vehicle actions:

$$\mathbf{x}_F(\mathbf{p}, \mathbf{x}) = \int_0^{t_f} \dot{\mathbf{x}}(\mathbf{x}, \mathbf{p})\, \mathrm{d}t, \tag{6}$$

$$\mathbf{C}(\mathbf{x}, \mathbf{p}) = \mathbf{x}_C - \mathbf{x}_F(\mathbf{p}, \mathbf{x}), \tag{7}$$

$$\Delta\mathbf{p} = -\left[\frac{\delta\mathbf{C}(\mathbf{x}, \mathbf{p})}{\delta p}\right]^{-1} \mathbf{C}(\mathbf{x}, \mathbf{p}). \tag{8}$$

The control parameters are modified until the residual of the boundary state constraints is within acceptable bounds or until the optimization diverges.

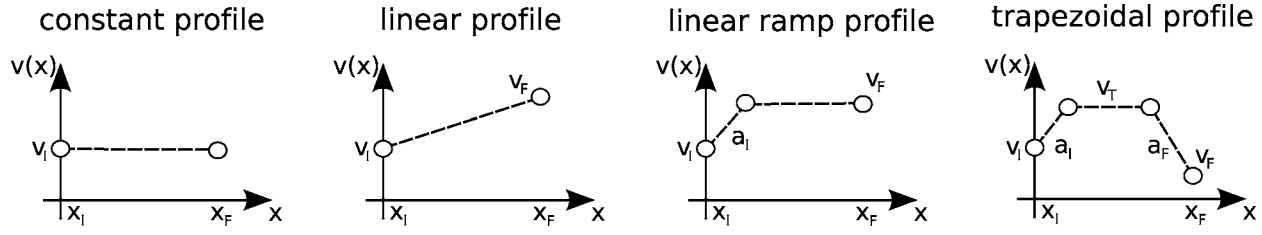

Figure 3. Velocity profiles used by the trajectory generator.

If the boundary state constraints are infeasible to reach given a particular parameterization (e.g., inside the minimum turning radius), the optimization is expected to diverge. The resulting trajectory is returned as the best estimate and is evaluated by the motion planner.

3.2. On-Road Navigation

During on-road navigation, the motion goal from the behavioral system is a location within a road lane. The motion planner then attempts to generate a trajectory that moves the vehicle toward 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 a set of local goals derived from the centerline path. The local 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 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 forward-predicted vehicle state. The sharp trajectory has the initial curvature parameter set to an offset value from the forward-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 behavioral subsystem, 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., zero if it is a stop line).

Figure 4 provides an example of smooth and sharp trajectories (light and dark) generated to the same goal poses. The smooth trajectories exhibit continuous curvature control throughout; the sharp trajectories begin with a discontinuous jump in curvature control, resulting in a sharp response from the vehicle.

The resulting trajectories are then evaluated against their proximity to static and dynamic obstacles in the environment, as well as their distance from the centerline path, their smoothness, and various other metrics. The best trajectory according to these metrics is selected and executed by the vehicle. Because the trajectory generator computes the feasibility of each trajectory using an accurate vehicle model, the selected trajectory can be directly executed by the vehicle controller.

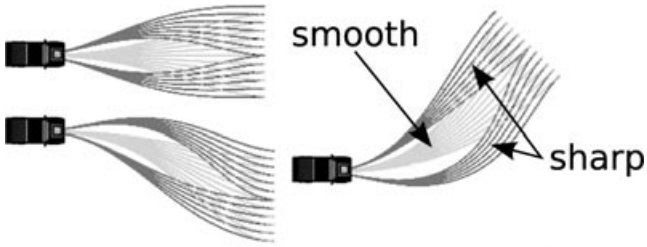

Figure 4. Smooth and sharp trajectories. The trajectory sets are generated to the same endpoints but differ in their initial commanded curvature.

Figure 5 provides an example of the local planner following a road lane. Figure 5(a) 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 5(b) shows the extracted centerline path from the desired lane (in red). Figure 5(c) shows a set of trajectories generated by the vehicle given its current state and the centerline path and lane boundaries. From this set of trajectories, a single trajectory is selected for execution, as discussed above. Figure 5(d) shows the evaluation of one of these trajectories against both static and dynamic obstacles in the environment, and Figure 5(f) shows this trajectory being selected for execution by the vehicle.

3.3. 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 toward this goal pose. However, driving in unstructured environments, such as zones, significantly differs from driving on roads. As mentioned in the preceding section, when traveling on roads the desired lane implicitly provides a preferred path for the vehicle (the centerline of the lane).

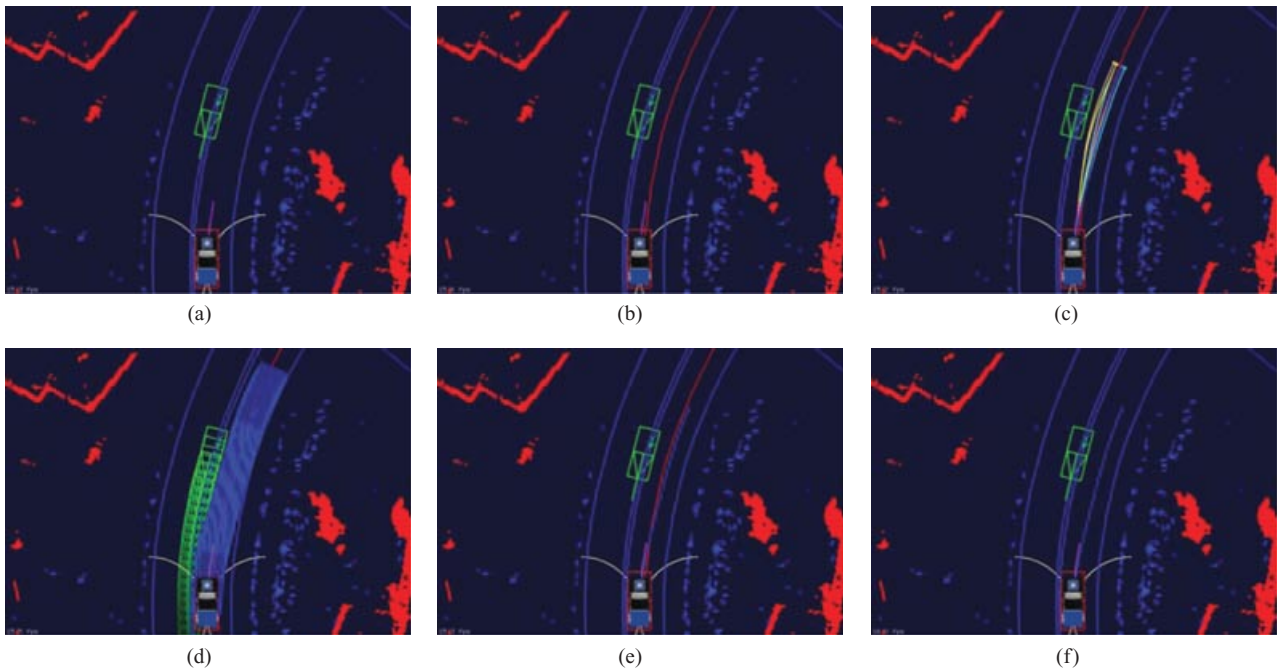

Figure 5. A single timeframe following a road lane from the DARPA Urban Challenge. Shown is the centerline path extracted from the lane (b), the trajectories generated to track this path (c), and the evaluation of one of these trajectories against both static and dynamic obstacles (d and e).

In zones 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, we use a lattice planner that searches over vehicle position (x, y), orientation θ, and speed v. The set of possible local maneuvers considered for each (x, y, θ, v) state in the planner's search space is constructed offline using the same vehicle model as used in trajectory generation, so that it can be accurately executed by the vehicle. This planner searches in a backward 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* (Likhachev, Ferguson, Gordon, Stentz, & Thrun, 2005). 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 subotpimality 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 backward direction, because in such a scenario, updated information in the vicinity of the vehicle affects a smaller portion of the search space so that less repair is required.

To scale to very large zones (up to 0.5×0.5 km), the planner uses a multiresolution 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. 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 allow the vehicle to transition between these states. Because coarse- and dense-resolution variants share the same dimensionality and, in particular, have 16 orientations in common, they seamlessly interface with each other, and the resulting

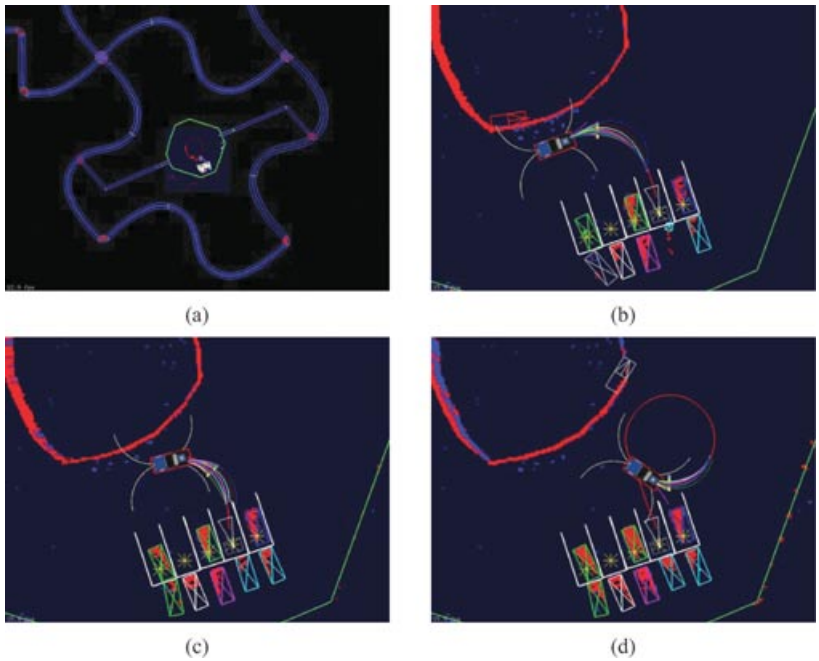

Figure 6. Replanning when new information is received. As Boss navigates toward its desired parking spot (lattice path shown in red, trajectories to track path in various colors), it observes more of one of the adjacent vehicles and replans a path that brings it smoothly into the spot.

solution paths overlapping both coarse and dense areas of the space are smooth and feasible.

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 toward a parking spot, we have a second lattice planner 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 then tracked by the local planner in a similar manner to the paths extracted from road lanes. The motion planner generates a set of 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, we significantly reduce the risk of having the vehicle move itself into a state that is difficult to leave.

Figure 6 illustrates the tracking of the lattice plan and the replanning capability of the lattice planner. These images were taken from a parking task performed during the NQE (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 various colors), with detected obstacles shown as red areas. The trajectories generated to follow the path are shown emanating from our vehicle. (Notice how each trajectory consists of two sections, with the first leaving the path and the second returning to the path.) 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

that 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 reposition.

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 such cases, the behaviors layer issues a goal pose (or set of poses) to the motion planner and indicates that it is in an error recovery mode. The motion planner then uses the lattice planner to generate a path to the set of goals, with the lattice planner determining during its planning which goal is easiest to reach. 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).

4. PERCEPTION

The perception system is responsible for providing a model of the world to the behavioral and motion planning subsystems. The model includes the moving vehicles (represented as a list of tracked objects) and static obstacles (represented in a regular grid) and localizing the vehicle relative to, and estimating the shape of, the roads it is driving on.

4.1. Moving Obstacle Detection and Tracking

The moving obstacle detection and tracking subsystem provides a list of object hypotheses and their characteristics to the behavioral and motion planning subsystems. The following design principles guided the implementation:

- No information about driving context is used inside the tracking algorithm.
- No explicit vehicle classification is performed. The tracking system provides information only about the movement state of object hypotheses.
- Information about the existence of objects is based on sensor information only. It is possible for some objects to be predicted, but only for short time intervals, as a compensation for known sensor parameters. Detection dropouts caused by noise, occlusions, and other artifacts must be handled elsewhere.

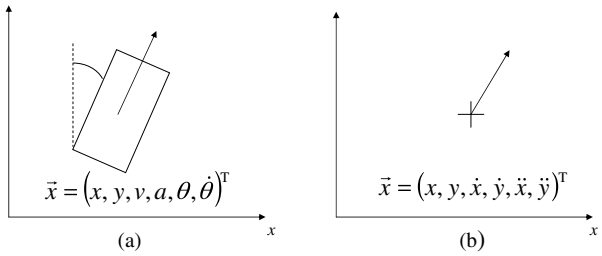

Figure 7. The two models used by the tracking system: a reduced bicycle model with a fixed shape (a) and a point model without shape information (b).

- Object identifiers are not guaranteed to be stable. A new identifier does not necessarily mean that it is a new object.
- Well-defined and distinct tracking models are used to maximize the use of information provided by heterogeneous sensors.
- Motion prediction exploits known road geometry when possible.
- Sensor-specific algorithms are encapsulated in sensor-specific modules.

Figure 7 shows the two tracking models used to describe object hypotheses. The box model represents a vehicle by using a simplified bicycle model (Kaempchen, Weiss, Schaefer, & Dietmayer, 2004) with a fixed length and width. The point model provides no estimate of extent of the obstacle and assumes a constant-acceleration model (Darms, Rybski, & Urmson, 2008a) with adaptive noise dependent on the length and direction of the velocity vector. Providing two potential tracking models enables the system to represent the best model of tracked objects supported by the data. The system is able to switch between these models as appropriate.

The system classifies object hypotheses as either *moving* or *not moving* and either *observed moving* or *not observed moving*, so that each hypothesis can be in one of four possible states. The moving flag is set if the object currently has a velocity that is significantly different from zero. The observed moving flag is set once the object has been moving for a significant amount of time (on the order of 0.4 s) and is not cleared until the vehicle has been stopped for some larger significant amount of time (on the order of 10 s). The four states act as a well-defined interface to the other software modules, enabling classes of tracked objects

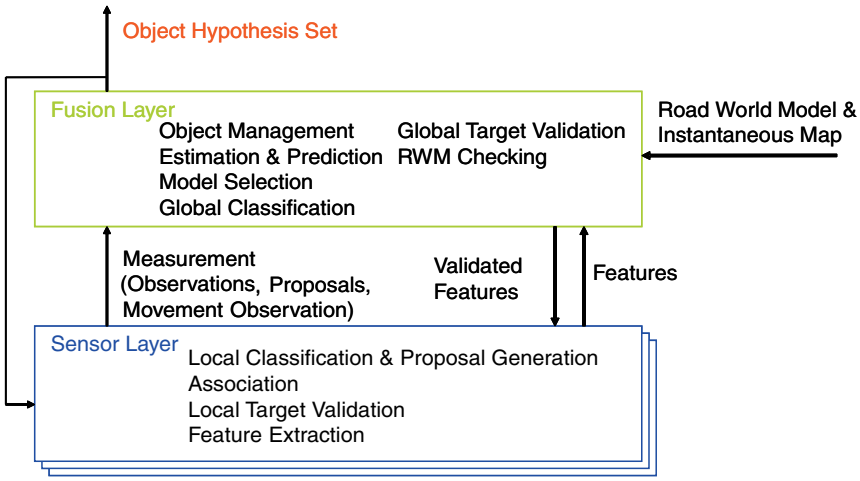

Figure 8. The moving obstacle detection and tracking system architecture.

to be ignored in specific contexts (e.g., not observed moving object hypotheses that are not fully on a road can be ignored for distance-keeping purposes, as they likely represent vehicles parked at the side of the road or other static obstacles (Darms, Baker, Rybski, & Urmson, 2008).

Figure 8 illustrates the architecture of the tracking system. It is divided into two layers, a *sensor layer* and a *fusion layer* (Darms & Winner, 2005). For each sensor type (e.g., radar, scanning laser, etc.), a specialized sensor layer is implemented. For each physical sensor on the robot a corresponding sensor layer instance runs on the system. The architecture enables new sensor types to be added to the system with minimal changes to the fusion layer, and other sensor modules, such as new physical sensors, can be added without any modifications to source code. The following paragraphs describe the path from sensor raw data to a list of object hypotheses.

Each time a sensor receives new raw data, its corresponding sensor layer instance requests a prediction of the current set of object hypotheses from the fusion layer. Features are extracted out of the measured raw data with the goal of finding all vehicles around the robot (e.g., edges from laser scanner data; MacLachlan, 2005). Artifacts caused by ground detections or vegetation, for example, are suppressed by validating features in two steps. In the first step validation is performed with sensor-specific algorithms, e.g., using the velocity measurements inside a radar module to distinguish a static ground return from a moving vehicle. The second step is performed via a general validation interface. The validation performed inside the fusion layer uses only non-sensor-specific information. It performs checks against the road geometry and against an instantaneous obstacle map, which holds untracked three-dimensional (3D) information about any obstacles in the near range. The result is a list of validated features that potentially originate from vehicles.

The validated features are associated with the predicted object hypotheses using a sensor-type-specific association algorithm. Afterward, for each extracted feature (associated or not), multiple possible interpretations as a box or point model are generated using a sensor-type-specific heuristic, which takes the sensor characteristics into account [e.g., resolution, field of view (FOV), detection probabilities]. The compatibility of each generated interpretation with its associated prediction is computed. If an interpretation differs significantly, or if the feature could not be associated, the sensor module initializes a new object hypothesis. In case of an associated feature, a new hypothesis can replace the current model hypothesis (box or point model). Note that for each feature, multiple new hypotheses can be generated. A set of new object hypotheses is called a proposal.

For each associated feature the interpretation that best fits the prediction is used to generate an observation. An observation holds all of the data necessary to update the state estimation for the associated object hypothesis in the fusion layer. If no interpretation

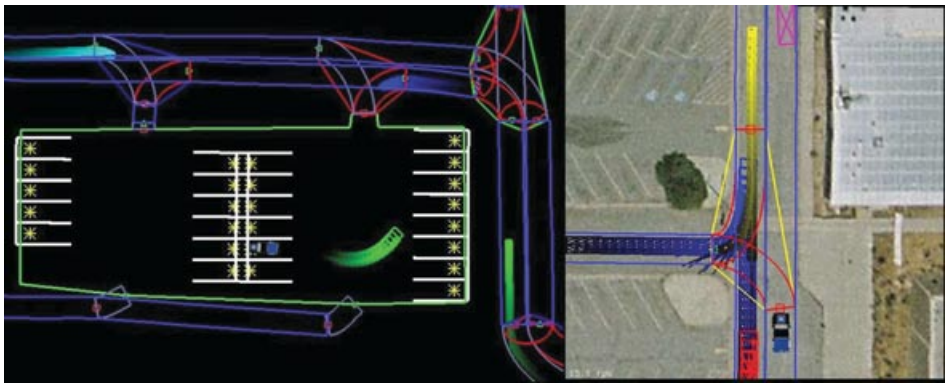

Figure 9. The moving obstacle detection system predicts the motion of tracked vehicles. In parking lots (left) predictions are generated by extrapolating the tracking filter. For roads (right) vehicles are predicted to move along lanes.

is compatible, then no observation is generated and only the proposal exists. As additional information for each extracted feature becomes available, the sensor module can also provide a movement observation. The movement observation tells the fusion layer whether an object is currently moving. This information is based only on sensor raw data (e.g., via an evaluation of the velocity measurement inside the radar module).

The proposals, observations, and movement observations are used inside the fusion layer to update the object hypotheses list and the estimated object states. First the best tracking model (box or point) is selected with a voting algorithm. The decision is based on the number and type of proposals provided from the different sensors (Darms, Rybski, & Urmson, 2008b). For objects that are located on roads, the road shape is used to bias the decision.

Once the best model is determined, the state estimate is either updated with the observation provided by the sensor layer or the model for the object hypothesis is switched to the best alternative. For unassociated features, the best model of the proposal is added to the current list of object hypotheses. With the update process complete, object hypotheses that have not been observed for a certain amount of time are removed from the list.

Finally, a classification of the movement state for each object hypothesis is carried out. It is based on the movement observations from the sensors and a statistical test based on the estimated state variables. The movement observations from sensors are prioritized over the statistical test, and movement observations that classify an object as not moving overrule movement observations that classify an object as moving (Darms et al., 2008).

The result is an updated list of object hypotheses that are accompanied by the classification of the movement state. For objects that are classified as moving and observed moving, a prediction of the state variables is made. The prediction is based on logical constraints for objects that are located on the road. At every point where a driver has a choice to change lanes (e.g., at intersections), multiple hypotheses are generated. In zones (parking lots, for example), the prediction is solely based on the estimated states of the hypothesis (see Figure 9).

4.2. Static Obstacle Detection and Mapping

The static obstacle mapping system combines data from the numerous scanning lasers on the vehicle to generate both instantaneous and temporally filtered obstacle maps. The instantaneous obstacle map is used in the validation of moving obstacle hypotheses. The temporally filtered maps are processed to remove moving obstacles and are filtered to reduce the number of spurious obstacles appearing in the maps. Whereas several algorithms were used to generate obstacle maps, only the curb detection algorithm is presented here.

Geometric features (curbs, berms, and bushes) provide one source of information for determining road shape in urban and off-road environments. Dense LIDAR data provide sufficient information to generate accurate, long-range detection of these relevant geometric features. Algorithms to detect these features must be robust to the variation in features

found across the many variants of curbs, berms, ditches, embankments, etc. The curb detection algorithm presented here exploits the Haar wavelet to deal with this variety.

To detect curbs, we exploit two principle insights into the LIDAR data to simplify detection. First, the road surface is assumed to be relatively flat and slow changing, with road edges defined by observable changes in geometry, specifically in height. This simplification means that the primary feature of a road edge reduces to changes in the height of the ground surface. Second, each LIDAR scan is processed independently, as opposed to building a 3D point cloud. This simplifies the algorithm to consider input data along a single dimension. The curb detection algorithm consists of three main steps: preprocessing, wavelet-based feature extraction, and postprocessing.

The preprocessing stage provides two important features: mitigation of false positives due to occlusions and sparse data, and formatting the data for feature extraction. False geometric cues can result from striking both foreground and background objects or be due to missing data in a scan. Foreground objects are typically detected as obstacles (e.g., cones, telephone poles) and do not denote road edges. To handle these problems, points are initially clustered by distance between consecutive points. After clustering, small groups of points are removed from the scan. A second pass labels the points as dense or sparse based on the distances between them. The dense points are then linearly resampled in order to produce an input sequence of 2^n heights.

The wavelet-based feature extraction step analyzes height data through a discrete wavelet transform using the Haar wavelet (Daubechies, 1992). The Haar wavelet is defined by the mother wavelet and scaling function:

$$\Psi(t) = \begin{cases} 1 & \text{if } 0 \le t < \frac{1}{2}, \\ -1 & \text{if } \frac{1}{2} < t < 1, \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

$$\varphi(2^j t - i) = \begin{cases} 1 \text{ if } 0 \le t < 1, \\ \qquad\qquad\qquad j > 0 \wedge 0 \le i \le 2^j - 1. \\ 0 \text{ otherwise,} \end{cases} \tag{10}$$

The Haar transform results in a sequence of coefficients representing the scaled average slopes of the input signal within various sampling windows (Shih & Tseng, 2005). Because each sampling window is half the size of the previous window, these windows successively subdivide the signal into higher resolution slopes or detail levels.

The feature extraction step (see Figure 10) takes the Haar coefficients, y, and considers them by window sizes, going from largest to smallest window. The algorithm classifies points as road points (class 1) or nonroad points, and works as follows:

1. Collect coefficients for the current detail level, i.
2. Label each coefficient with the label of the coefficient at detail level $i - 1$, which represents the same portion of the signal.
3. Calculate $\hat{y}_{\text{road}}$ using these labels.
4. Relabel coefficients by absolute distance from $\hat{y}_{\text{road}}$, where the distance threshold for detail level i is given as d_i. In other words, points are labeled by the function

$$\text{class, } (y[n], i) = \begin{cases} 1 \text{ if } y[n] - \hat{y}_{\text{road}}| \ge d_i \\ 0 \qquad \text{otherwise} \end{cases}. \tag{11}$$

5. Continue to detail level $i + 1$.

Postprocessing applies a few extra heuristics to eliminate false positives and detect some additional nonroad points. Using the dense/sparse labeling from preprocessing, nonroad labels in sparse sections are moved from the sparse points to the neighboring dense point closest to the vehicle. Because all LIDARs on the vehicle look downward, the closer point corresponds to the higher surface (e.g., berm, wall) creating the geometric cue. Afterward, sparse points are removed from the classification list. The resulting list represents the locations of the likely road and surrounding geometric cues. Figure 11 illustrates the performance of the algorithm in a typical on-road scene from the Urban Challenge.

4.3. Roadmap Localization

Boss is capable of either estimating road geometry or localizing itself relative to roads with known geometry. Most urban roads change shape infrequently, and most urban driving can be thought of as responding to local disturbances within the constraints of a fixed road network. Given that the shape and location of paved roads change infrequently, our approach was

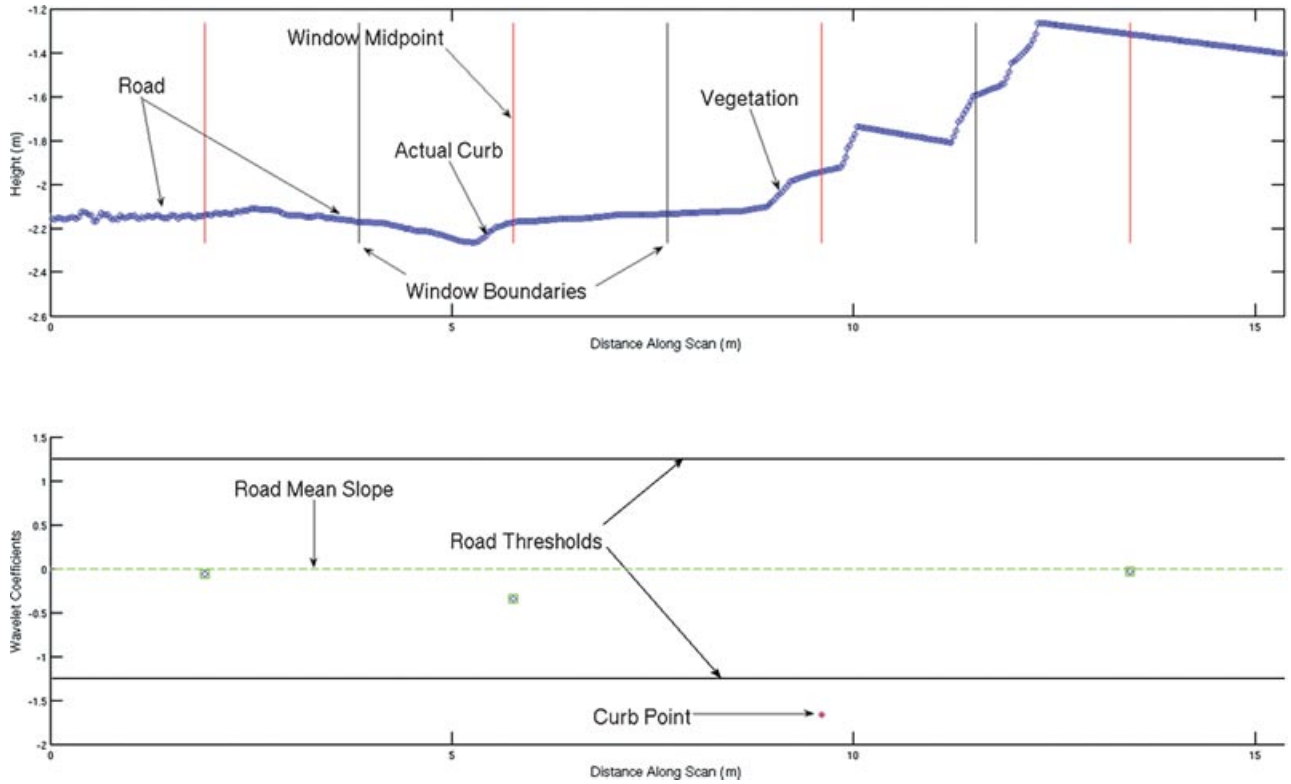

Figure 10. A single frame from the feature extraction algorithm. The top frame contains the original height signal (thick line). The window boundaries are from a single-detail level of the transform. The bottom frame shows the wavelet coefficients for each window.

to localize relative to paved roads and estimate the shape of dirt roads, which change geometry more frequently. This approach has two main advantages:

- it exploits a priori knowledge to eliminate the necessity of estimating road shape in most cases;
- it enables the road shape estimation problem to emphasize geometric cues such as berms and bushes, which are common in environments with dirt roads and easier to detect at long range than lane markings.

This approach led to two independent algorithms, one to provide a smooth pose relative to a road network, and one to estimate the shape of dirt roads. The two algorithms are never operated simultaneously, thus avoiding complex interactions between them. Both the localization and road shape estimation algorithms were heavily tested and proved to be effective. Despite confidence in the road shape estimation system, it was not enabled during the Urban Challenge competition. Based on the way point density and aerial imagery of the UCFE course, the team determined that there was not a need to estimate road shape. A description of the road shape estimation approach is provided in Section 4.4 for completeness because it enables Boss to drive on general roads and was ready to be used in the event, if necessary.

4.3.1. Localization Inputs

The localization process can be thought of as transforming the pose provided by a global positioning system (GPS)–based pose estimation system into a smooth coordinate frame registered to a road network. To do this it combines data from a

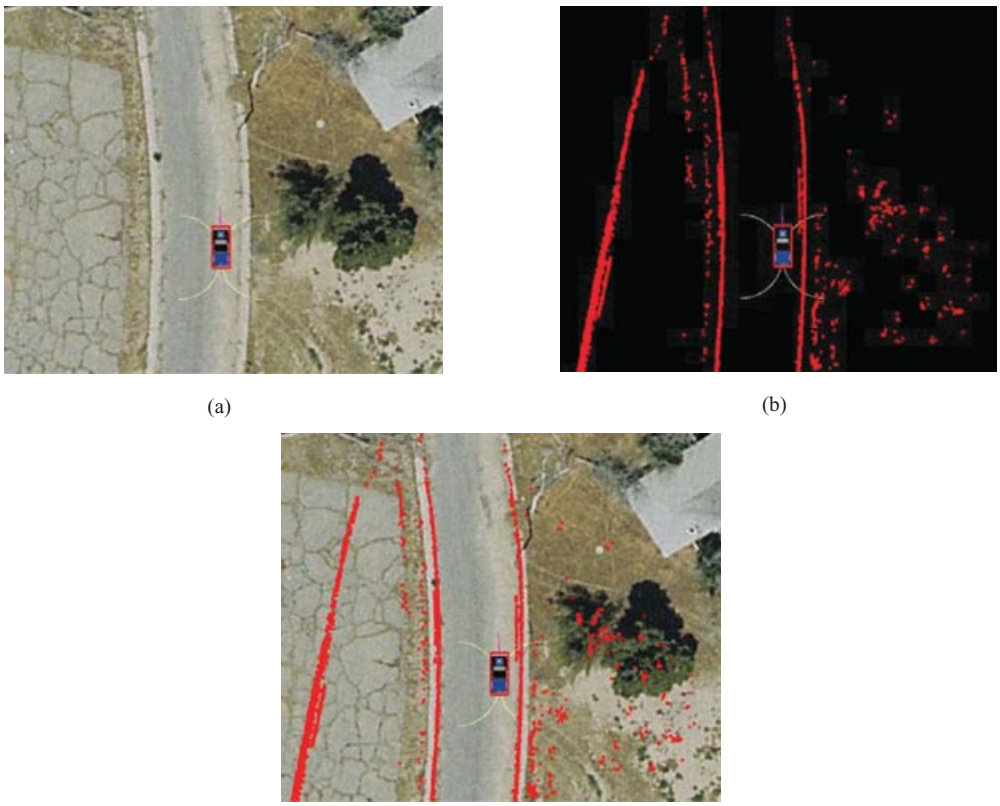

Figure 11. Overhead view of a road section from the Final Event course (a). Points show nonroad points (b). Overlay of nonroad points on imagery (c).

Table II. Error characteristics of the Applanix POS-LV, as reported by Applannix.

	Error with GPS and differential corrections	Error after 1 km of travel without GPS
Planar position, m	0.3	0.88
Heading, °	0.05	0.07

commercially available position estimation system and measurements of road lane markers with an annotated road map.

The initial global position estimate is received from a device (POS-LV) developed by the Applanix Corporation. This system fuses GPS and inertial and wheel encoder data to provide a 100-Hz position estimate that is robust to GPS dropout. Table II describes the nominal performance of this system with and without GPS. The POS-LV is configured to slightly outperform the nominal performance specifications through the use of a combination of Omnistar Virtual Base Station and High Precision services. By incorporating the High Precision data, nominal performance is improved to a 0.1-m planar expected positioning error. Whereas a positioning accuracy of 0.1 m sounds sufficient to blindly localize within a lane, these correction signals are frequently disrupted by even small amounts of overhead vegetation. Once disrupted, this signal's reacquisition takes approximately a half hour. Thus, relying on these corrections is not viable for urban driving. Furthermore, lane geometries may not be known to meter accuracies a priori. It is critically important to be localized correctly relative to the lane boundaries, because crossing over the lane center could have disastrous consequences.

To detect lane boundaries, down-looking SICK LMS lasers are used to detect the painted lane markers on roads. Lane markers are generally brighter than the surrounding road material and are detected by convolving the intensities across a line scan with a slope function. Peaks and troughs in the response represent the edges of potential lane marker boundaries. To reduce false positives, only appropriately spaced pairs of peaks and troughs are considered to be lane markers. Candidate markers are then further filtered based on their brightness relative to their support region. The result is a set of potential lane marker positions.

The road map used for localization encodes both correct local geometry and information about the presence or absence of lane markings. Although it is possible for road geometry to be incorrect globally, the local geometry is important to the estimation scheme, as will be described below. If the road geometry is not well known, the map must indicate this. When the vehicle traverses parts of the map with poor geometry, the road shape estimation algorithms operate and the road map localization algorithms are disabled.

4.3.2. Position Filtering

To transform the measurements provided by the POS-LV to a smooth, road-network-registered frame, we consider three potential sources of position error:

1. *Position jumps.* Despite the availability of inertial information, the POS-LV will occasionally generate position jumps.
2. *Position drift.* The correction signals, variation in satellite constellation, and ionospheric disturbances cause slowly various changes to the position reported by the POS-LV.
3. *Road model errors.* Our approach to creating road maps is to manually extract road shapes from aerial imagery. Modern aerial imagery can provide quarter-meter or better image resolution, but global registration is generally good only to a meter or worse. Distortion in the imagery generally has a low spatial frequency, so that the local shape of the road is accurate, but the apparent global position may be inaccurate.

These three error sources are grouped into two classes; discontinuous errors (such as jumps) and continuous errors (drift and model errors). With every new state measurement, the change in position $\Delta\mathbf{x}$ is checked for validity based on measured wheel speed v, anticipated percentage velocity error ζ, allowed position jitter ε, travel direction θ, and allowable travel direction error τ:

$$\text{reject} = |\Delta\mathbf{x}| > v(1+\zeta)\Delta t + \varepsilon \ \vee \ \left\{ (|\Delta\mathbf{x}| > \varepsilon) \wedge \frac{\Delta\mathbf{x}}{|\Delta\mathbf{x}|} \cdot \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} > \tau \right\}. \quad (12)$$

In Eq. (12), the first term ensures that the reported motion of the vehicle is not significantly greater than the anticipated motion given the vehicle wheel speed. The second term ensures that for any significant motion, the reported motion is in approximately the same direction as the vehicle is pointed (which is expected for the speeds and conditions of the Urban Challenge). If $\Delta\mathbf{x}$ is rejected, a predicted motion is calculated based on heading and wheel speed. The residual between the prediction and the measured change in position is accumulated in a running sum, which is subtracted from position estimates reported by the POS-LV. In practice, values of $\zeta = 0.05$, $\varepsilon = 0.02$, and $\tau = \cos(30 \text{ deg})$ produce good performance.

Correcting for the continuous class of errors is how localization to the road model is performed. The localization process accumulates lane marker points (LMP) generated by the laser lane marker detection algorithms. After a short travel distance, 1 m during the Urban Challenge, each LMP is associated with a lane boundary described in the road model. The distance of the corrected global position p for each LMP from the lane center is calculated, and the projection point onto the lane center p_c is noted. Half of the lane width is subtracted from this distance, resulting in the magnitude of the local error estimate between the lane boundary position and the model of the lane boundary position. This process is repeated for each LMP, resulting in an error estimate:

$$e_{\text{LMP}} = \frac{1}{n_{\text{LMP}}} \sum_{1}^{n_{\text{LMP}}} \left[\left| (p^i - p_c^i) \right| - \frac{w_l^i}{2} \right] \cdot \left(\frac{p^i - p_c^i}{\left| p^i - p_c^i \right|} \right). \quad (13)$$

This represents the error in the current filtered/localized position estimate; thus the e_{LMP} represents how much error there is between the current

combination of the existing error estimate and position. In practice, we further gate the error estimates, discarding any larger than some predetermined maximum error threshold (3 m during the Urban Challenge). Over time, error estimates are accumulated through a recursive filter:

$$e_{\text{cur}} = e_{\text{prev}} + \alpha e_{\text{LMP}}. \tag{14}$$

This approach generates a smooth, road-network-referenced position estimate, even in situations in which GPS quality is insufficient to otherwise localize within a lane. This solution proved to be effective. During prechallenge testing, we performed several tests with GPS signals denied (through the placement of aluminum caps over the GPS antennas). In one representative test, the vehicle was able to maintain position within a lane, while traveling more than 5.7 km without GPS. During this test, the difference error in the POS-LV position reached up to 2.5 m, more than enough to put the vehicle either off the road or in another lane if not compensated for.

4.4. Road Shape Estimation

To robustly drive on roads where the geometry is not known a priori, the road shape estimator measures the curvature, position, and heading of roads near the vehicle. The estimator fuses inputs from a variety of LIDAR sensors and cameras to composite a model of the road. The estimator is initialized using available prior road shape data 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.

4.4.1. Sensor Inputs

Two primary features were used to determine road location.

Curbs represent the edge of the road and are detected using the Haar wavelet (see Static Obstacle 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.

4.4.2. State Vector

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

$$s(t) = [x(t), y(t), \varphi(t), C_0(t), C_1(t), W(t)], \tag{15}$$

where $[x(t), y(t), \phi(t)]$ represents 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

$$y(x) = \tan[\varphi(t)]x + C_0\frac{t}{2}x^2 + C_1\frac{t}{6}x^3. \tag{16}$$

4.4.3. Particle Filter

The road estimator uses an SIR (sample importance resample) (Duda & Hart, 1972) 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 set of equations, where ds represents the relative distance that the robot traveled from one iteration of the algorithm to the next:

$$y = y + \varphi ds + \frac{(ds)^2}{2}C_0 + \frac{(ds)^3}{6}C_1, \tag{17}$$

$$\varphi = \varphi + C_0 ds + \frac{(ds)^2}{2}C_1 - d\varphi, \tag{18}$$

$$C_0 = C_0 + C_1 ds, \tag{19}$$

$$C_1 = (0.99)C_1. \tag{20}$$

The final C_1 term represents the assumption that the curvature of a road will always tend to head toward 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 are not modeled by the curve parameters. In addition to Gaussian noise, several more directed searches are performed, in which the width of the road can randomly increase or decrease itself by a fixed amount. Empirically, this represents the case in which a road suddenly becomes wider because a turn lane or a shoulder has suddenly appeared.

4.4.4. Sensor Data Processing

Because particle filtering requires the evaluation of a huge number of hypotheses (more than 10,000 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:

$$L = \frac{1}{Z} e^{-C(\text{shape, data})}. \tag{21}$$

In Eq. (21), Z is a normalization constant that forces the sum of the likelihoods over all road shapes to be 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 subclasses 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. To scale the cost function, counts and distances are normalized. The resulting cost function is

$$C = \sum_{i=0}^{N} \left(\frac{N_o^i}{\sigma_o}\right)^2 + \left(\frac{N_C^i}{\sigma_C}\right)^2 + \left(\frac{D_C}{\sigma_C}\right)^2 + \left(\frac{D_L}{\sigma_L}\right)^2. \tag{22}$$

4.4.5. Fast Convolutions and Distance Transforms

To exactly compute the cost function, we need to convolve each road shape with the cost map to sum the detection counts and obstacle costs. This requires tens of thousands of convolutions per second for each count. Whereas fast methods exist to exactly compute simple shapes (Viola & Jones, 2001), the shapes that we wish to convolve are too complicated for these approaches. Instead, we approximate the road shape as a set of overlapping disks centered on the road shape (see Figure 12).

The disks have a diameter equal to the width of the road and are spaced at 1.5-m samplings.

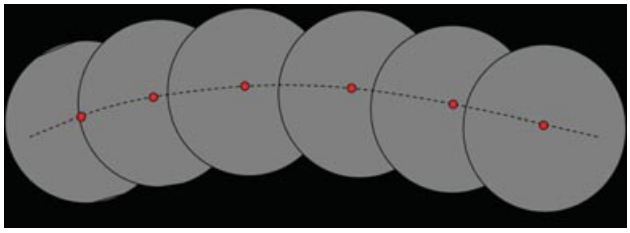

Figure 12. Example illustrating how road shape is approximated by a series of disks.

Although this approach tends to overcount, we have found that it is adequate for the purposes of tracking the road and is more than fast enough for our purposes.

To allow the width of the road to vary, we compute convolutions for different-width disks ranging from 2.2 to 15.2 m 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, and 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 disk 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, and obstacles that have vanished are removed. The result of the convolutions is a set of cost maps that represent the road configuration space for each potential road width.

To evaluate the distance components of the cost function, we employ a distance transform (Huttenlocker & Felzenswalb, 2004). The distances from the nearest curb location or lane marker location to a given sample location are 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.

A snapshot of the overall system performance is illustrated in Figure 13. The example shows on an

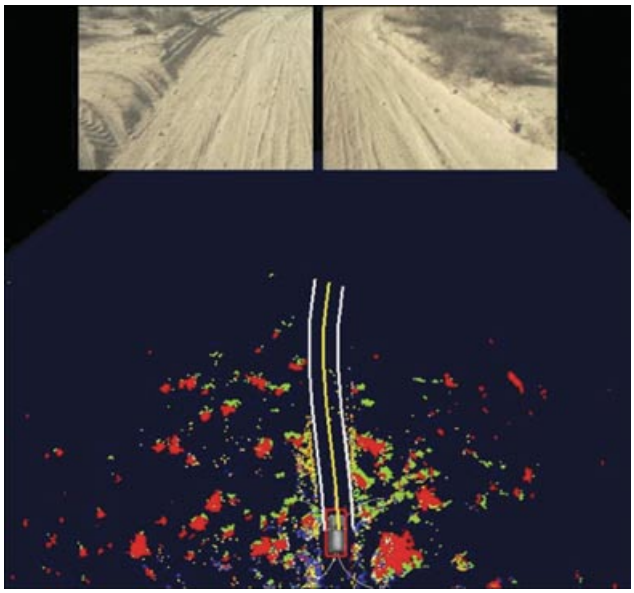

Figure 13. Example showing the road shape estimate (parallel curves) for an off-road scene. Obstacles and berms are illustrated by pixels.

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 three lines.

5. MISSION PLANNING

To generate mission plans, the data provided in the road network definition file (RNDF) are used to create a graph that encodes the connectivity of the environment. Each way point in the RNDF becomes a node in this graph, and directional edges (representing lanes) are inserted between way points and all other way points that they can reach. For instance, a way point defining a stop line at an intersection will have edges connecting it to all way points leaving the intersection that can be legally driven to. These edges are also assigned costs based on a combination of several factors, including expected time to traverse the edge, distance of the edge, and complexity of the corresponding area of the environment. The resultant cost graph is the baseline for travel road and lane decisions by the behavioral subsystem.

A value function is computed over this graph, providing the path from each way point to the current goal (e.g., the first checkpoint in a mission). In addition to providing the executive more information to reason about, computing a value function is useful because it allows the navigation system to react appropriately if an execution error occurs (e.g., if the vehicle drives through an intersection rather than turning, the new best path to take is instantly available).

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. Because the size of the graph is relatively small, this replanning can be performed quickly, allowing for near-immediate response to detected changes.

To correctly react to these occurrences, the robot must be able to detect when a road is impassable and plan another route to its goal, no matter where the goal is. Particularly difficult cases include planning to a goal immediately on the other side of a newly discovered blockage and behaving reasonably when a one-way street becomes blocked. Road conditions are fluid and highly variable, and road blockages may not be permanent. Thus, a robot should eventually revisit the site of a previously encountered blockage to see whether it has been cleared away. In fact, the robot *must* revisit a blockage if all other paths to a goal have also been found to be blocked, hoping to discover that a road blockage has been cleared.

5.1. Detecting Blockages

To determine whether there is a blockage, Boss can either directly detect the blockage or infer it by a failure to navigate a lane. To directly detect a blockage, the road in front of Boss is checked against a static obstacle map to see whether lethal obstacles completely cross the road. Noise in the obstacle map is suppressed by ignoring apparent blockages that have been observed for less than a time constant (nominally 5 s). Blockages are considered to no longer exist if the obstacle map shows a sufficiently wide corridor through the location where a blockage previously existed.

The direct detection algorithm generates obstacles using an efficient but optimistic algorithm; thus, there are configurations of obstacles that effectively block the road but are not directly detected as a road blockage. In these conditions, the on-road navigation algorithm may report that the road is blocked, inducing a *virtual blockage*. Because the behavior generation

module works by picking the lowest cost path to its goal, this is an elegant way for it to stimulate itself to choose an alternate route. Because virtual blockages induced by the behavior generation module are not created due to something explicitly observed, they cannot be removed by observation; thus, they are removed each time the vehicle achieves a checkpoint. Although heuristic, this approach works well in practice, as these blockages are often constructed due to odd geometries of temporary obstacles. By waiting until the current checkpoint is complete, this approach ensures that the vehicle will wait until its current mission is complete before revisiting a location. If the only path to a goal is through a virtual blockage that cannot be detected as cleared, and the situation that caused the blockage to be declared has been resolved, then forward progress will still occur, because the virtual blockages decay in the same manner that explicitly observed blockages do.

5.2. Blockages

Once a blockage has been detected, the extent along affected lanes that the blockage occupies is determined. Locations before and after the blockage are identified where U-turn maneuvers can be performed. At these locations, road model elements representing legal places to make a U-turn are added. Simultaneously, the corresponding traversal costs for the U-turn maneuvers are set to low values, the costs for crossing the blockages are increased by a large amount, and the navigation policy is recomputed. Because Boss follows the policy, the high costs levied on traversing a blockage cause the robot to choose an alternate path. If Boss later detects that the blockage is gone, the traversal costs for elements crossing the blockage are restored to their defaults, and the traversal costs for the added U-turn elements are effectively removed from the graph.

Revisiting of previously detected blockages is implemented by gradually reducing the traversal cost applied to road elements crossing a blockage. If the cost eventually drops below a predefined threshold, the blockage is treated as if it were observed to be gone. The U-turn traversal costs are not concomitantly increased; instead, 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 a blockage is removed, whereas *not* increasing the U-turn traversal costs encourages the robot to continue to plan to traverse the U-turn if it is beneficial to do so.

The cost c increment added by a blockage is decayed exponentially:

$$c = p2^{-a/h}, \tag{23}$$

where a is the time since the blockage was last observed, h is a half-life parameter, and p is the starting cost penalty increment for blockages. To illustrate, if the blockage is new, we have $a = 0$ and $c = p$. If the blockage was last observed h time units in the past, we have $a = h$ and $c = p/2$. The cost continues to decay exponentially as the blockage ages.

An exponential decay rate mitigates the problem of a single blockage being interpreted as multiple blockages (due to incomplete perception), causing a cost of np, where n is the number of blockages. Under this condition, a linear decay rate would cause an unacceptably long delay before revisiting a blockage.

A weakness of this blockage handling approach is that it is possible to waste time making multiple visits to a blockage that never gets removed. A simple solution of incrementing h for the blockage after each new visit would make the traversal costs decay more slowly each time the obstacle is observed.

On one-way roads, U-turn lanes are not created in response to road blockages. However, traversal costs across the blockage are increased, decreasing the likelihood of reusing the road. To respond to one-way road blockages, the zone navigation planner is invoked as an error recovery mode, as discussed in Section 6.3.

6. BEHAVIORAL REASONING

The behavioral architecture is responsible for executing the policy generated 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 behavioral architecture is based on the concept of identifying a set of driving contexts, each of which requires the vehicle to focus on a reduced set of environmental features. At the highest level of this design, the three contexts are road, intersection, and zone, and their corresponding behaviors are, respectively, *lane driving*, *intersection handling*, and *achieving a zone pose*. The achieving a zone pose behavior is meant for unstructured or unconstrained

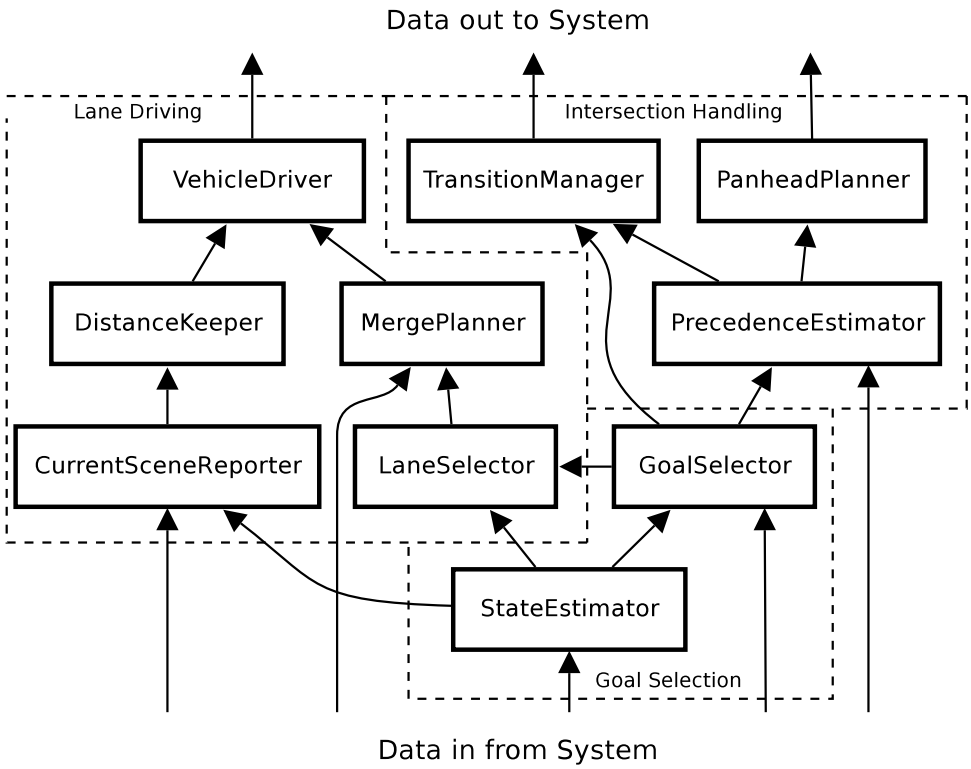

Figure 14. High-level behaviors architecture.

environments, including parking lots and jammed intersections. In practice this behavior's function is performed by the zone planner. Figure 14 shows a diagram of behavioral subsystem architecture with the subbehaviors corresponding to the high-level behaviors, along with two subbehaviors making up the auxiliary goal selection behavior, which plays a crucial role not only in standard operation but also in error recovery. The function of each of these subcomponents is described in Table III.

6.1. Intersections and Yielding

The *precedence estimator* is most directly responsible for the system's adherence to the Urban Challenge rules (DARPA, 2007) including obeying precedence, not entering an intersection when another vehicle is in it, and being able to merge into and across moving traffic. To follow the rules, the precedence estimator combines data from the rest of the system to determine whether it is clear to go. This state is used as a gate condition in the *transition manager* and triggers the issuance of the motion goal to proceed through the intersection.

The precedence estimator uses a combination of the road model and moving obstacle information to determine whether it is clear to go. The road model provides static information about an intersection, including the set of lane exit way points that compose the intersection and the geometry of their associated lanes. The moving obstacle set provides dynamic information about the location, size, and speed of estimated nearby vehicles. Given the problem of spatial uncertainty, false positives and false negatives must be accounted for in the precedence estimation system.

The road model provides important data, including the following:

- The current intersection of interest, which is maintained in the world model as a group of exit way points, some subset of which will also be stop lines.
- A virtual lane representing the action the system will take at that intersection.

Table III. Components of the behavioral subsystem.

Goal selection components	Drive down road	Handle intersection
State estimator: 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 RNDF. ***Goal selector***: uses the current logical location as reported by state estimator to generate the next series of local goals for execution by the motion planner; these will be either lane goals or zone goals.	***Lane selector***: uses the surrounding traffic conditions to determine the optimal lane to be in at any instant and executes a merge into that lane if it is feasible. ***Merge planner***: determines the feasibility of a merge into a lane proposed by lane selector. ***Current scene reporter***: the current scene reporter distills the list of known vehicles and discrete obstacles 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. ***Distance keeper***: uses the surrounding traffic conditions to determine the necessary in-lane vehicle safety gaps and govern the vehicle's speed accordingly. ***Vehicle driver***: combines the outputs of distance keeper and lane selector with its own internal rules to generate a so-called "motion parameters" message, which governs details such as the vehicle's speed, acceleration, and desired tracking lane.	***Precedence estimator***: uses the list of known other vehicles and their state information to determine precedence at an intersection. ***Pan-head planner***: aims the pan-head sensors to gain the most relevant information for intersection precedence decisions. ***Transition manager***: manages the discrete-goal interface between the behavioral executive and the motion planner, using the goals from goal selector and the gating function from precedence estimator to determine when to transmit the next sequence of goals.

- A set of yield lanes[1] for that virtual lane.
- 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 an intersection's geometry.

The moving obstacle set is received periodically and represents the location, size, and speed of all detected vehicles around the robot. In contrast to the information gleaned from the road model, these data are highly dynamic, displaying several properties that must be accounted for in the precedence estimation system: tracked vehicles can flicker in and out of existence for short durations of time; sensing and modeling uncertainties can affect the estimated shape, position, and velocity of a vehicle; and 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.

[1]Yield lanes are lanes of moving traffic for which a vehicle must wait for a clear opportunity to execute the associated maneuver.

6.1.1. Intersection-Centric Precedence Estimation

Within the road model, an intersection is defined as a group of lane exit way points. That is, an intersection must contain one or more lane exit way points, and each lane exit way point will be a part of exactly one intersection. The next intersection is thus determined as the intersection containing the next lane exit way point that will be encountered. This excludes exits that Boss will cross but not stop at (e.g., crossing the top of a tee intersection that has only one stop sign).

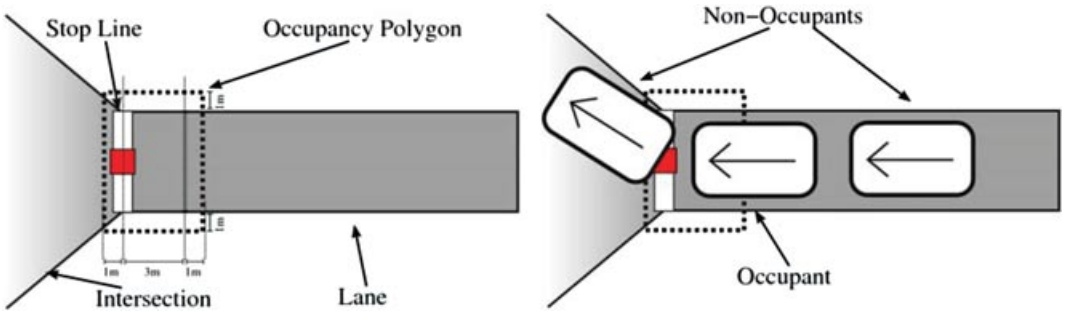

Figure 15. Typical exit occupancy polygon and examples of vehicles at an exit.

Precedence between any two exit way points is determined first by whether the exit way points are stop lines. Nonstop exit way points automatically have precedence over exit way points that have stop lines. Among stop line exit way points, 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 estimator. Given the dynamic and noisy nature of the moving obstacle set, the algorithm uses a purely geometric and instantaneous notion of way point occupancy for computing arrival times. An exit way point is considered to be occupied when any vehicle's estimated front bumper is inside or intersects a small polygon around the way point, 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.

Occupancy polygons are constructed for each exit way point and for the whole intersection. The occupancy polygons for each exit way point 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. The size of the polygon for an exit way point determines several factors:

- The point at which a vehicle gains its precedence ordering, which is actually some length along the lane backward from the stopline.
- The system's robustness to spatial noise, where larger polygons are generally more robust than smaller ones at retaining the precedence order.
- The system's ability to discriminate two cars moving through the intersection in sequence, where larger polygons are more likely to treat two discrete cars as one for the purposes of precedence ordering.

Figure 15 shows a typical exit occupancy polygon extending 3 m back along the lane from the stop line and with 1 m of padding on all sides. This is the configuration that was used on race day.

The estimated front bumper of a vehicle must be inside the occupancy polygon as shown in Figure 15 to be considered to be an occupant of that polygon.

A given occupancy polygon maintains its associated exit way point, its occupancy state, and two pieces of temporal data:

1. The time of first occupancy, which is used to determine precedence ordering.
2. The time of most recent (last) occupancy, which is used to implement a temporal hysteresis around when the polygon becomes unoccupied.

To account for (nearly) simultaneous arrival, 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 exit way point. Exit way points 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-right rule. Similarly, exit way points 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. (Empirically, 0.5 s worked well for this value.) The result is considered to be the exit way point'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

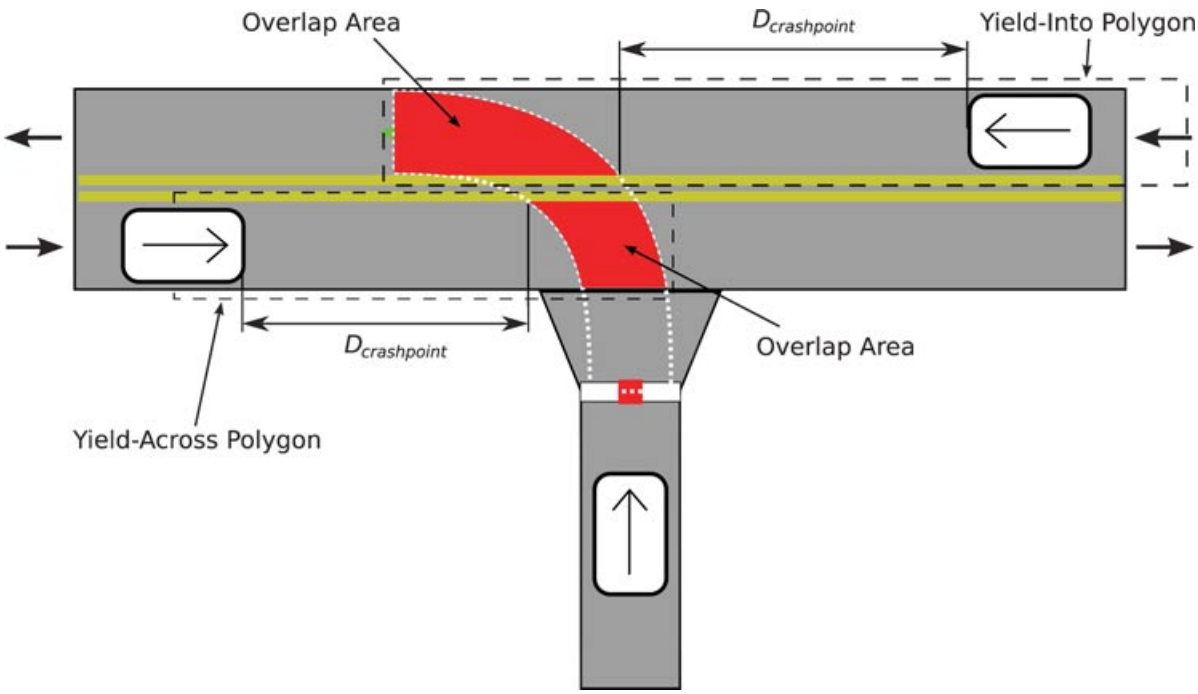

Figure 16. Typical tee intersection with yield lanes.

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 way point, Boss is considered to have precedence at that intersection.

6.1.2. Yielding

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 way point to some other way point, 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 defined as the real lanes that overlap a virtual lane. Overlapped virtual lanes are not considered because they must already have a clearly established precedence order via stop lines. Intersections that fail this requirement (e.g., an intersection with four yield signs) are considered ill formed and are not guaranteed to be handled correctly. Thus, yield cases are considered only for merges into or across real lanes. Figure 16 shows an example tee intersection, highlighting the next intersection goal and the associated yield lanes.

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

1. T_{action} is computed as the time to traverse the intersection and get into the target lane using conservative accelerations from a starting speed of zero.
2. $T_{\text{accelerate}}$ is computed as the time for accelerating from zero up to speed in the destination lane using the same conservative acceleration.
3. T_{delay} is estimated as the maximum system delay.
4. T_{spacing} is defined as the minimum required temporal spacing between vehicles, where 1 s approximates a vehicle length per 10 mph.

Using these values, a required temporal window T_{required} is computed for each yield lane as

$$T_{\text{required}} = T_{\text{action}} + T_{\text{delay}} + T_{\text{spacing}} \tag{24}$$

for lanes that are crossed by the next intersection action. In the case of merging into a lane, the required window is extended to include the acceleration time, if necessary, as

$$T_{\text{required}} = \max(T_{\text{action}}, T_{\text{accelerate}}) + T_{\text{delay}} + T_{\text{spacing}}. \tag{25}$$

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

$$\ell_{\text{yield polygon}} = v_{\text{maxlane}}\ T_{\text{required}} + d_{\text{safety}}. \tag{26}$$

These yield polygons, shown in Figure 16, are used as a first pass for determining cars that are relevant to the yield window computations.

Any reported vehicle that is inside or overlaps the yield polygon is considered in the determination of the available yield window. Yield polygons are also provided to a *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 vehicle in a yield lane, a time of arrival is estimated at the near edge of the overlap area, called the *crash point* and illustrated in Figure 16 as follows:

1. Compute a worst-case speed v_{obstacle} along the yield lane by projecting the reported velocity vector, plus one standard deviation, onto the yield lane.
2. Compute d_{crash} 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

$$T_{\text{arrival}} = \frac{d_{\text{crash}}}{v_{\text{obstacle}}}. \tag{27}$$

4. Retain the minimum T_{arrival} as T_{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. To account for the possibility of tracked vehicles being lost momentarily, as in the exit way point precedence determination, this notion of instantaneous clearance is protected by a 1-s hysteresis. That is, all yield windows must be continuously open for at least 1 s before yield clearance is passed to the rest of the system.

6.1.3. 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 (i.e., a situation that would cause Boss to become stopped in an intersection).

Gridlock management comes into effect once the system determines that Boss has precedence at the current intersection and begins with a 15-s timeout to give the problematic vehicle an opportunity to clear. If still gridlocked after 15 s, the current intersection action is marked as locally high cost, and the mission planner is allowed to determine whether an alternate path to goal exists. If so, Boss will reroute along that alternate path; otherwise, the system jumps into error recovery for intersection goals, using the generalized pose planner to find a way around the presumed-dead vehicle. This is discussed in greater detail in the section describing error recovery.

6.2. Distance Keeping and Merge Planning

The distance-keeping behavior aims simultaneously to zero the difference between Boss's velocity and that of the vehicle in front of Boss and the difference between the desired and actual intervehicle gaps. The commanded velocity is

$$v_{\text{cmd}} = K_{\text{gap}}(d_{\text{actule}} - d_{\text{desired}}), \tag{28}$$

where v_{target} is the target-vehicle velocity and K_{gap} is the gap gain. The desired gap is

$$d_{\text{desired}} = \max\left(\frac{\ell_{\text{vehicle}}}{10} v_{\text{actual}}, d_{\text{mingap}}\right), \tag{29}$$

where the ℓ_{vehicle} term represents the one-vehicle-length-per-10-mph minimum-separation requirement and d_{mingap} is the absolute minimum gap requirement. When Boss's velocity exceeds the target vehicle's, its deceleration is set to a single configurable default value; when Boss's velocity is less than

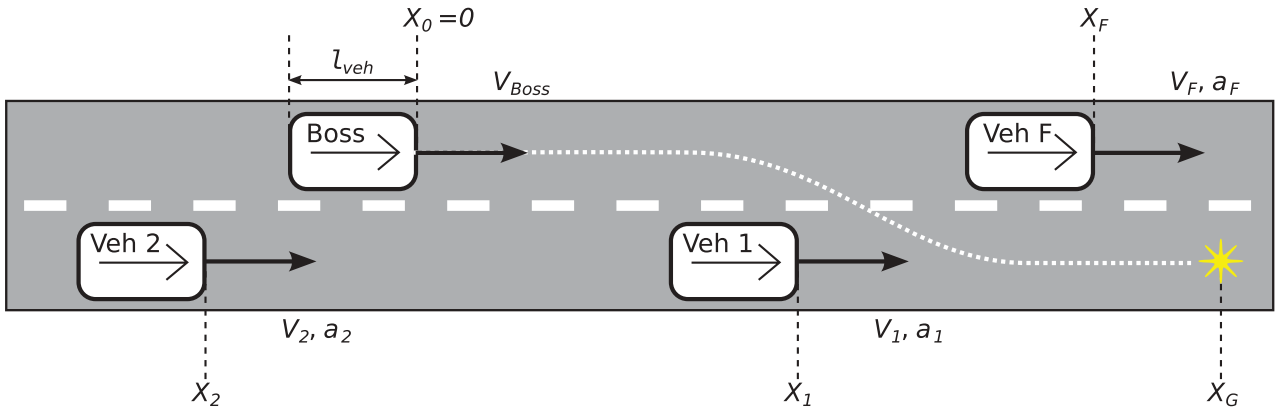

Figure 17. Two-lane merging.

the target vehicle's, for safety and smoothness, Boss's commanded acceleration is made proportional to the difference between the commanded and actual velocities and capped at maximum and minimum values ($a_{\max} = 4.0$ and $a_{\min} = 1.0$ m/s^2 on race day) according to

$$a_{\text{cmd}} = a_{\min} + K_{\text{acc}} v_{\text{vmd}} (a_{\max} - a_{\min}). \tag{30}$$

The merge, or lane-change, planner determines the feasibility of changing lanes. It is relevant not only on a unidirectional multilane 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 17. 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 whose setting based on experimentation was 12 m on race day.
2. Determine the merge-by distance, i.e., the allowable distance in the current lane in order to complete the merge. The merge-by distance in the case of a moving obstacle in front of Boss is

$$d_{\text{obst}} = \frac{v_0 d_{\text{initial}}}{v_0 - v_1}, \tag{31}$$

where d_{initial} is the initial distance to the moving obstacle. Note that this reduces to d_{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.

For either a front- or back-merge, first determine whether proper spacing is already met. For a front merge, this means

$$x_0 - \ell_{\text{vehicle}} - x_1 \geq \max\left(\frac{v_1 \ell_{\text{vehicle}}}{10}, d_{\text{mingap}}\right). \tag{32}$$

For a back merge,

$$x_1 - \ell_{\text{vehicle}} - x_1 \geq \max\left(\frac{v_0 \ell_{\text{vehicle}}}{10}, d_{\text{mingap}}\right). \tag{33}$$

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. Otherwise, determine the acceleration profile to accelerate or decelerate respectively to, and remain at, either the maximum or minimum speed until proper spacing is reached.

4. 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.
5. 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 and the rest 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 front of vehicle 1, between vehicles 1 and 2, and behind vehicle 2) and targets the appropriate feasible merge slot depending on the situation. For a multilane unidirectional road, this is generally the foremost feasible slot.

Once feasibility for all slots is determined, appropriate logic is applied to determine which slot to merge into depending on the situation. For a multilane 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.

Determination of the merge-by distance is the smallest of the distances to the 1) next motion goal (i.e., checkpoint), 2) end of the current lane, 3) closest road blockage in the current lane, and 4) projected position of the closest moving obstacle in the current lane.

6.3. Error Recovery

One of the most important aspects of the behavioral reasoning system is its responsibility to detect and address errors from the motion planner and other aberrant situations. To be effective, the recovery system should

- be able to generate a nonrepeating and novel sequence of recovery goals in the face of repeated failures ad infinitum.
- be able to generate different sets of recovery goals to handle different contexts.
- be implemented with minimal complexity so as to produce as few undesirable behaviors as possible.

To reduce interface complexity, the goal selection system follows the state graph shown in Figure 18.

Each edge represents the successful completion (Success) or the failed termination (Failure) of the current motion goal. Goal failure can be either directly reported by the motion planner or triggered internally by progress monitoring that declares failure if sufficient progress is not made within some time. All edge transitions trigger the selection of a new goal and modify the *recovery level*:

- Success resets recovery level to zero but caches the previous recovery level.
- Failure sets the recovery level to one greater than the maximum of the cached and current recovery level.

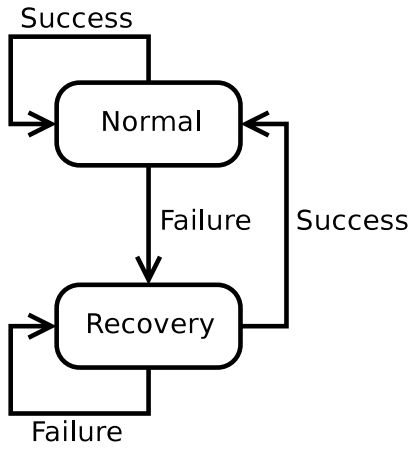

Figure 18. Goal selection state graph.

The recovery level and 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 recovery level results in higher risk attempts to recover, meaning actions that are generally farther away from the current position and/or the original goal. This process ensures that Boss tries low-risk and easy-to-execute maneuvers initially while still considering more drastic measures when necessary. If the recovery process chooses an obviously infeasible goal, the motion planner will signal failure immediately and the recovery level will increment. Otherwise, to complement explicit failure reports from the motion planner, forward progress is monitored such that the robot must move a minimum distance toward the goal over some span of time. If it does not, the current goal is treated as a failure, and the recovery level is incremented. The progress threshold and time span were determined largely by experimentation and set to 10 m and 90 s on race day. Generally speaking, these represent the largest realistic delay the system was expected to encounter during operation.

In general, the successful completion of a recovery goal sets the system back to normal operation. This eliminates the possibility of complex multimaneuver recovery schemes, at the benefit of simplifying the recovery state tracking. In situations in which the vehicle oscillates between recovery and normal operation, the recovery system maintains sufficient state to increase the complexity of recovery maneuvers.

6.3.1. On-Road Failures

The most commonly encountered recovery situation occurs when the on-road planner generates an error while the vehicle is driving down a lane. Any number of stimuli can trigger this behavior, including

- small or transient obstacles, e.g., traffic cones that do not block the entire lane but are sufficient to prevent the planner from finding a safe path through them
- larger obstacles such as road barrels, K-rails, or other cars that are detected too late for normal distance keeping to bring the system to a graceful stop
- low-hanging canopy, which is generally detected late and often requires additional caution and careful planning
- lanes whose shape is kinematically infeasible.

The algorithm for lane recovery goal selection, called *shimmy* and illustrated in Figure 19, selects an initial set of goals forward along the lane with the distance forward described by

$$d_{\text{shimmy}} = d_{\text{initial}} + R d_{\text{incremental}}. \tag{34}$$

Empirically, $d_{\text{initial}} = 20$ m and $d_{\text{incremental}} = 10$ m worked well. The 20-m initial distance was empirically found to give the planner sufficient room to get past stopped cars in front of the vehicle.

These forward goals (Goals 1–3 in Figure 19) are selected out to some maximum distance, roughly 40 m and corresponding to our high-fidelity sensor range, after which a goal is selected 10 m behind the vehicle (Goal 4) with the intent of backing up and getting a different perspective on the immediate impediment.

After backing up, the sequence of forward goals is allowed to repeat once more with slight (less than 5 m) alterations, after which continued failure causes one of two things to happen:

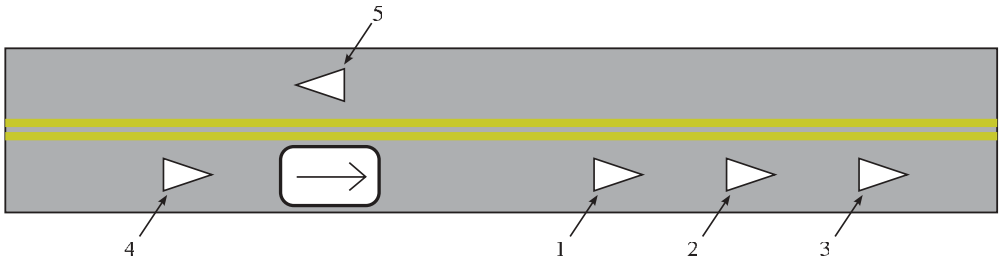

Figure 19. Example shimmy error recovery goal sequence.

1. If a lane is available in the opposing direction, mark the segment as locally blocked and issue a U-turn (Goal 5). This is supplemental to the external treatment of total segment blockages discussed in Section 5.2 and presumes that a U-turn has not yet been detected and generated.
2. If no lanes are available in the opposing direction (i.e., Boss is stuck on a one-way road), then the goal selection process is allowed to continue forward infinitely beyond the 40-m limit with the additional effect of removing an implicit *stay near the lane* constraint that is associated with all previous recovery goals. Removing this constraint gives the pose planner complete freedom to wander the world arbitrarily in an attempt to achieve some forward goal.

6.3.2. Intersection Failures

Error cases in intersections are perhaps the most difficult to recover from. These errors happen when an attempt to traverse an intersection is not possible due to obstacles and/or kinematic constraints or else as part of the gridlock resolution system. A simplified example sequence from this recovery algorithm, called *jimmy*, is show in Figure 20.

The first step in the jimmy algorithm is to try the original failed goal over again as a pose goal, instead of a road driving goal (e.g., goal), giving the motion planner the whole intersection as its workspace instead of just the smooth path between the intersection entry and exit. This generally and quickly recovers from small or spurious failures in intersections as well as compensating for intersections with turns that are tighter than the vehicle can make in a single motion. Should that first recovery goal fail, its associated entry way point is marked as blocked, and the system is allowed an opportunity to plan an alternate mission plan to the goal. If there is an alternate path, that alternate intersection goal (e.g., Goals 2, 3) is selected as the next normal goal and the cycle is allowed to continue. If that alternate goal fails, it is also marked as blocked, and the system is allowed to replan and so forth, until all alternate routes to the goal are exhausted, whereupon the system will select unconstrained pose goals (i.e., pose goals that can drive outside of the intersection and roads) incrementally farther away from the intersection along the original failed goal.

6.3.3. Zone Failures

The case in which specifics do matter, however, is the third recovery scenario, failures in zones. The pose planner that executes zone goals is general and powerful enough to find a path to any specific pose if such a path exists, so a failure to do so implies one of the following:

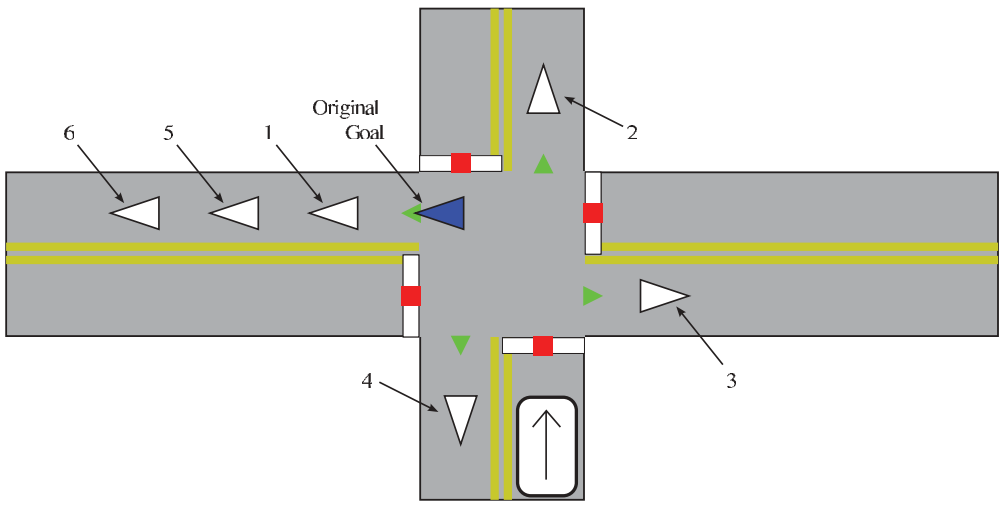

Figure 20. Example jimmy recovery goal sequence.

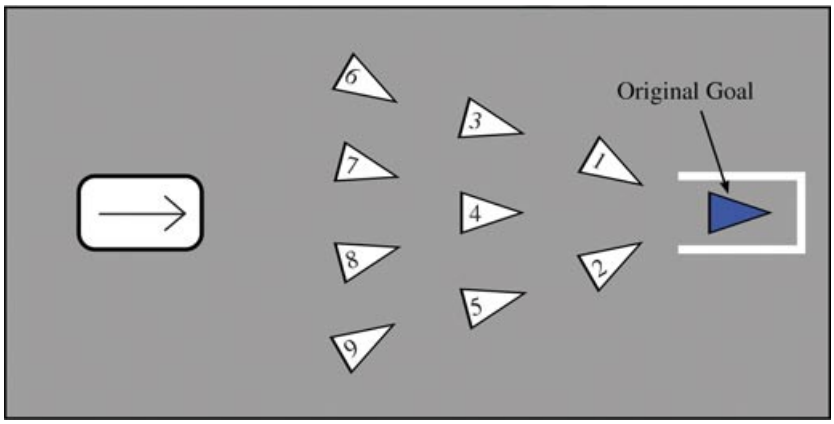

Figure 21. Example shake recovery goal sequence.

1. The path to the goal has been transiently blocked by another vehicle. Although DARPA guaranteed that a parking spot in a zone will be free, they made no such guarantees about traffic accumulations at zone exits or about the number of vehicles between the current position and the target parking spot. In either case, a retry of the same or a selection of a nearby similar goal should afford the transient blockage time to pass.
2. Owing to some sensor artifact, the system believes that there is no viable path to goal. In this case, selecting nearby goals that offer different perspectives on the original goal area may relieve the situation.
3. The path to the goal is actually blocked.

The goal selection algorithm for failed zone goals, called *shake*, selects goals in a regular, triangular pattern facing the original goal, as shown in Figure 21.

On successful completion of any one these goals, the original goal is reattempted. If the original goal fails again, the shake pattern picks up where it left off. If this continues through the entire pattern, the next set of actions is determined by the original goal. Parking spot goals were guaranteed to be empty, so the pattern is repeated with a small incremental angular offset ad infinitum. For zone exit way point goals, the exit is marked as blocked and the system attempts to reroute through alternate exits similar to the alternate path selection in the jimmy algorithm. In the case of no other exits, or no other path to goal, the system issues completely unconstrained goals to logical successors of the zone exit way point in a last-ditch effort to escape the zone.

If these goals continue to fail, then farther successors are selected in a semirandom breadth-first search along the road network in a general last-ditch recovery algorithm called *bake*. Increasing values of recovery level call out farther paths in the search algorithm. The goals selected in this manner are characterized by being completely unconstrained, loosely specified goals that are meant to be invoked when each of the other recovery goal selection schemes has been exhausted. In addition to shake goals at a zone exit, these are selected for shimmy goals that run off the end of the lane and similarly for jimmy goals when all other attempts to get out of an intersection have failed.

Through these four recovery algorithms (shimmy, jimmy, shake, and bake), many forward paths are explored from any single location, leaving only the possibility that the system is stuck due to a local sensor artifact or some other strange local minima that requires a small local adjustment. To address this possibility, a completely separate recovery mechanism runs in parallel to the rest of the goal monitoring and recovery system with a very simple rule: if the system has not moved at least 1 m in the last 5 min, override the current goal with a randomized local goal. When that goal is completed, pretend that there was a completely fresh wakeup at that location, possibly clearing the accumulated state, and try again.

The goals selected by this algorithm, called *wiggle*, are illustrated in Figure 22. The goals are

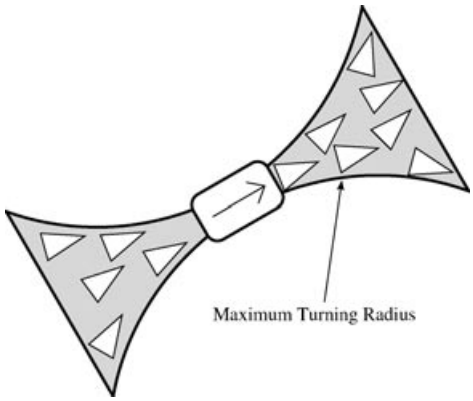

Figure 22. Example wiggle recovery goals.

pseudo-random and approximately kinematically feasible and can be either in front of or behind the vehicle's current pose. The algorithm is, however, biased somewhat forward of the robot's position so that there is statistically net-forward motion if the robot is forced to choose these goals repeatedly over time in a behavior similar to the "wander" behavior described by 0.

The composite recovery system provides a necessary line of defense against failures in the planning and perceptive systems. This robustness was a key element of winning the Urban Challenge.

7. SOFTWARE INFRASTRUCTURE

The software infrastructure is a toolbox 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 list of the tools provided by the infrastructure:

Communications library. Abstracts around basic interprocess communication over UNIX Domain Sockets, TCP/IP, or UDP; supports the boost::serialization library to easily marshal data structures across a communications link and then unmarshal on the receiving side. A key feature is anonymous publish/subscribe, which disconnects a consumer of data from having to know who is actually providing the data, enabling the easy interchange of components during testing and development.

Interfaces library. Each interface between two processes in the system is added to this library. Each interface fits into a plug-in framework so that a task, depending on its communications configuration, can dynamically load the interfaces required at run time. For example, this enables a perception task to abstract the notion of a LIDAR source and at run time be configured to use any LIDAR source, effectively decoupling the algorithmic logic from the nuts-and-bolts of sensor interfacing. 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 such as 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, to provide 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 time stamp. 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. Because the interprocess communication system uses an anonymous publish/subscribe scheme, the consuming

processes will receive the played-back messages, without realizing that data are not coming from a live sensor. This feature is useful in order to replay incidents that occurred on the vehicle for offline analysis.

Tartan Racing Operator Control Station (TROCS). A graphical user interface (GUI) 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 (Figure 23).

8. TESTING

Testing was a central theme of the research program that developed Boss. Over the 16 months of development, Boss performed more than 3,000 km of autonomous driving. The team used time on two test vehicles, a simulation and data replay tool, and multiple test sites in three states to debug, test, and evaluate the system.

Testing and development were closely intertwined, following the cyclic process illustrated in Figure 24. Before algorithm development began, the team assembled requirements that defined the capabilities that Boss would need to be able to complete

Figure 23. The TROCS is an extensible GUI that enables developers to both monitor telemetry from Boss while it is driving and replay data offline for algorithm analysis.

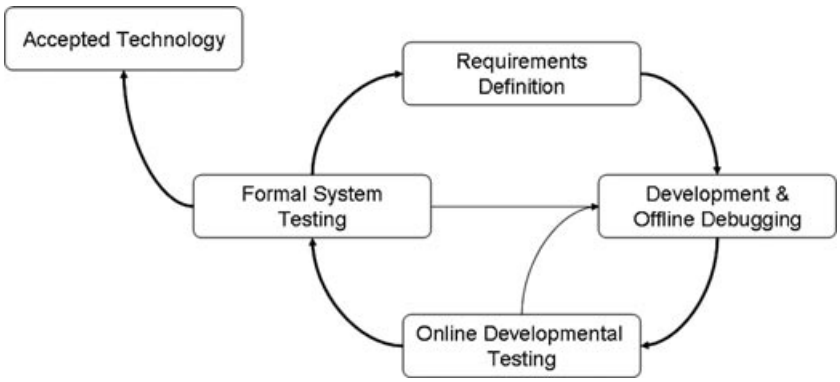

Figure 24. The requirements and testing process used in the development of Boss.

the challenge. Requirements drove the development of algorithms and selection of components. Algorithms and approaches were tested offline either in simulation or by using data replay. Once an algorithm became sufficiently mature, it would move to on-vehicle testing, in which system and environmental interactions could be fully evaluated. These tests would often uncover algorithmic problems or implementation bugs that were not obvious during offline testing, often resulting in numerous cycles of rework and further offline debugging. Once the developers were satisfied that the algorithm worked, testing of the algorithm was added to the formal, regularly scheduled system test. Independent testing of algorithms would often uncover new deficiencies, requiring some rework. In other cases, testing and postanalysis would cause the team to modify the requirements driving the development, limiting or extending scope as appropriate. Eventually, the technology would be deemed accepted and ready for the Urban Challenge.

Regressive system testing was the cornerstone of the development process. Following the cyclic development process, each researcher was free to implement and test as independently of the overall system as possible, but to judge overall capability and to verify that component changes did not degrade overall system performance, the team performed regressive system testing.

System testing was performed with a frequency proportionate to system readiness. Between February and October 2007, the team performed 65 days of system testing. Formal testing time was adjusted over the course of the program to ensure relevance. As an example, during February the team performed less than 16 km of system testing. In contrast, during the first 3 weeks of October, the team performed more than 1,500 km of autonomous testing.

In general, the team tested once a week, but leading up to major milestones (the midterm site visit and NQE) the team moved to daily regressive testing. During regressive testing, the team would evaluate Boss's performance against a standard set of plays (or scenarios) described in a master playbook. The playbook captures more than 250 different driving events that are important to evaluate. Figure 25 illustrates what a page from the playbook looks like. Each play is annotated with priority (ranked 1–3), how thoroughly it has been tested, how it relates to requirements, and a description of how Boss should behave when encountering this scenario. The 250 plays cover the mundane (correctly stopping at a stop sign) to the challenging (successfully navigating a jammed intersection), enabling the team to have confidence that

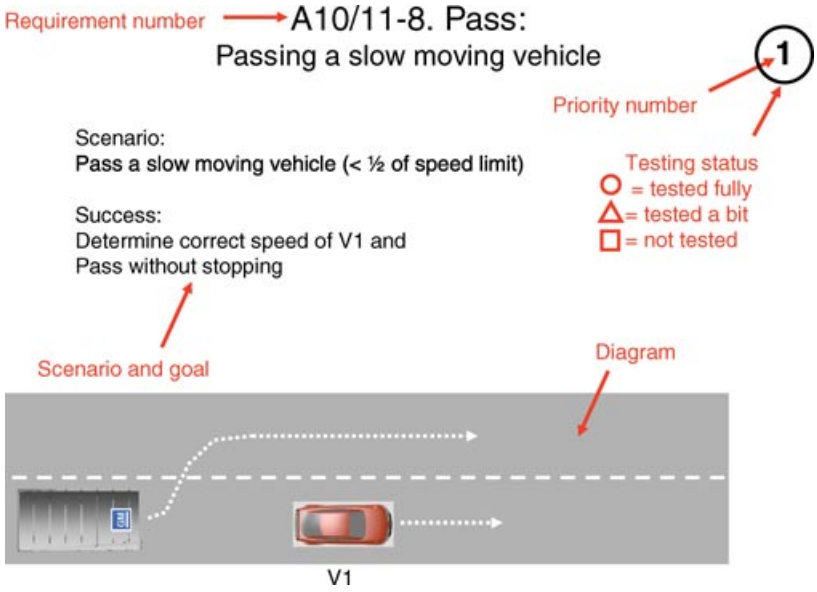

Figure 25. A representative page from the testing playbook.

even the most coupled software changes were not damaging overall system performance.

Feedback from system tests was rapidly passed to the development team through a *hot wash* after each system test, and a test report was published within 48 h. Whereas the hot wash was delivered by the test software operator, who conveyed first-hand experience of code performance, the test report provided a more black-box understanding of how well the system met its mission requirements. Software bugs discovered through this and other testing were electronically tracked, and formal review occurred weekly. The test report included a *gap analysis* showing the requirements that remained to be verified under system testing. This gap analysis was an essential measure of the team's readiness to compete at the Urban Challenge.

In addition to regressive testing, the team performed periodic endurance tests designed to confirm that Boss could safely operate for at least 6 h or 60 miles (96 km) (the stated length of the Urban Challenge). This was the acid test of performance, allowing the team to catch intermittent and subtle software and mechanical defects by increasing time on the vehicle. One of the most elusive problems discovered by this testing process was an electrical shorting problem that was the result of a 2-mm gash in a signal line on the base vehicle Tahoe bus. The problem caused Boss to lose all automotive electrical power, killing the vehicle. Had the team not performed endurance testing, it is plausible that this defect would have never been encountered before the UCFE and would have caused Boss to fail.

9. PERFORMANCE AT THE NATIONAL QUALIFICATION EVENT AND URBAN CHALLENGE FINAL EVENT

The NQE and UCFE were held at the former George Air Force Base (see Figure 26), which provided a variety of roads, intersections, and parking lots to test the vehicles. The NQE allowed DARPA to assess the capability and safety of each of the competitors. The teams were evaluated on three courses. Area A required the autonomous vehicles to merge into and turn across dense moving traffic. Vehicles had to judge the size of gaps between moving vehicles, assess safety, and then maneuver without excessive delay. For many of the vehicles, this was the most difficult challenge, as it involved significant reasoning about moving obstacles. Area B was a relatively long road course that wound through a neighborhood.

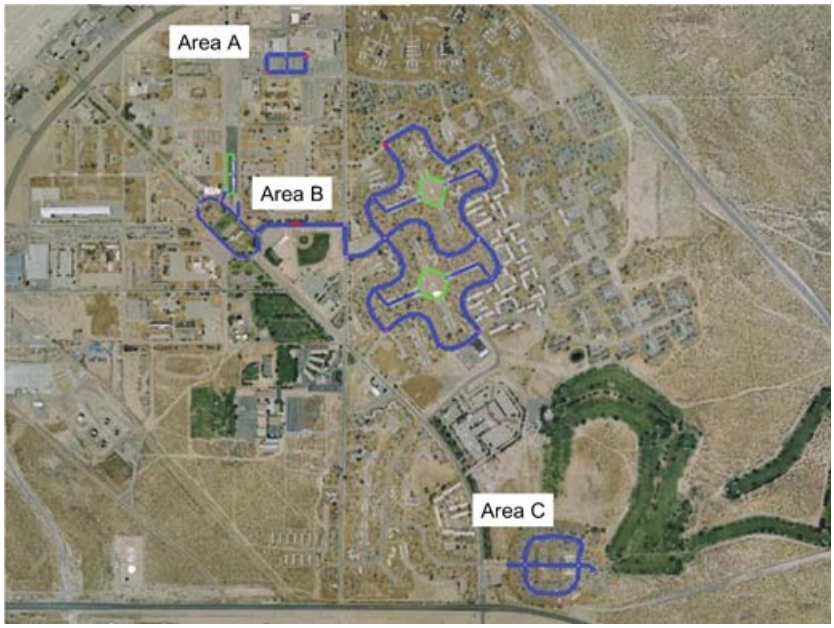

Figure 26. The NQE took place in three areas, each emphasizing different skills. Area A tested merging with moving traffic, area B tested navigation, and area C tested rerouting and intersection skills.

Figure 27. Of the 11 vehicles that qualified for the UCFE, 3 completed the challenge without human intervention. Three additional teams finished the course with minor interventions.

Vehicles were challenged to find their way through the road network while avoiding parked cars, construction areas, and other road obstacles but did not encounter moving traffic. Area C was a relatively short course but required autonomous vehicles to demonstrate correct behavior with traffic at four-way intersections and to demonstrate rerouting around an unexpectedly blocked road.

For the final event, the field was reduced to 11 teams (see Figure 27). Traffic on the course was provided by not only the 11 qualifying vehicles but 50 human-driven vehicles operated by DARPA. While primarily on-road, the course also included a pair of relatively short dirt roads, one of which ramped its way down a 50-m elevation change.

Despite the testing by each of the teams and a rigorous qualification process, the challenge proved to be just that. Of the 11 teams that entered, 6 were able to complete the 85-k course, 3 of them without human intervention. Boss finished the challenge approximately 19 min faster than the second-place vehicle, Junior (from Stanford University) and 26 min ahead of the third-place vehicle, Odin (from Virginia Tech). The vehicles from Cornell, MIT, and the University of Pennsylvania rounded out the finishers.

Overall, the vehicles that competed in the challenge performed admirably, with only one vehicle-to-vehicle collision, which occurred at very low speeds and resulted in no damage. Although the vehicles drove well, none of them was perfect. Among the foibles: Boss twice incorrectly determined that it needed to make a U-turn, resulting in its driving an unnecessary 2 miles (3.2 km); Junior had a minor bug that caused it to repeatedly loop twice through one section of the course; and Odin incurred a significant GPS error that caused it to drive partially off the road for part of the challenge. Despite these glitches, these vehicles represent a new state of the art for urban driving.

The following sections describe a few incidents during the qualifications and final event when Boss encountered some difficulties.

9.1. National Qualification Event Analysis

Boss performed well at each component of the NQE, consistently demonstrating good driving skills and overall system robustness. Through the NQE testing, Boss demonstrated three significant bugs, none of which was mission ending.

The first significant incident occurred in area A, the course that tested a robot's ability to merge with traffic. During Boss's first run on this course, it approached a corner and stopped for about 20 s before continuing, crossing the centerline for some time before settling back into the correct lane and continuing. This incident had two principal causes: narrow lanes and incorrect lane geometry. In preparing the course, DARPA arranged large concrete Jersey barriers immediately adjacent to the lane edges to prevent vehicles from leaving the course and injuring spectators. This left little room for navigational error. When configuring Boss for the run, the team adjusted the geometry defined in the RNDF, with the intention of representing the road shape as accurately as possible given the available overhead imagery. During this process, the team incorrectly defined the shape of the inner lanes. Whereas this shape was unimportant for Boss's localization and navigation, it was used to predict the motion of the other vehicles on the course. The incorrect geometry caused Boss to predict that the other vehicles were coming into its lane (see Figure 28). It

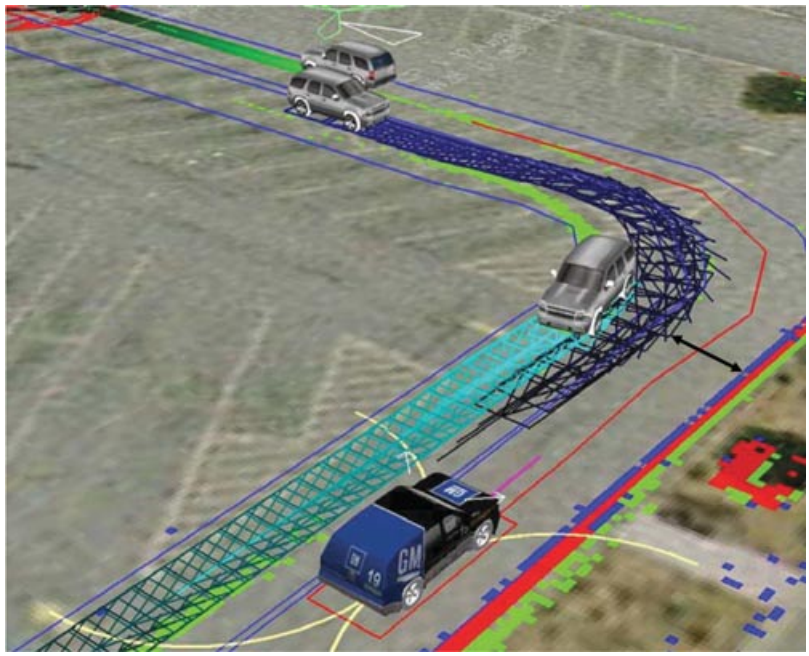

Figure 28. Data replay shows how the incorrectly extrapolated path of a vehicle (shaded rectangles) and the wall (pixels to the right of Boss) create a space that Boss believes is too narrow to drive through (indicated by the arrow).

thus stopped and an error recovery mode (shimmy) kicked in. After shimmying down the lane, Boss reverted to normal driving and completed the test.

The second significant incident occurred during testing in area C. This course tested the autonomous vehicle's ability to handle traffic at intersections and replan for blocked roads. For most of this course, Boss drove well and cleanly, correctly making its way through intersections and replanning when it encountered a blocked road. The second time Boss encountered a blocked road it began to U-turn, getting two of its wheels up on a curb. As it backed down off the curb, it caused a cloud of dust to rise and then stopped and appeared to be stuck, turning its wheels backward and forward for about a minute. Then Boss started moving again and completed the course without further incident.

Posttest data analysis revealed that Boss perceived the dust cloud as an obstacle. In addition, an overhanging tree branch behind Boss caused it to believe there was insufficient room to back up. Normally, when the dust settled, Boss would have perceived that there was no longer an obstacle in front of it and continued driving, but in this case the dust rose very close to Boss, on the boundary of its blind spot (see Figure 29). The obstacle detection algorithms treat this area specially and do not clear obstacles within this zone. Eventually Boss wiggled enough to verify that the cell where it had previously seen the dust was no longer occupied. Once again, Boss's consistent attempts to replan paid off, this time indirectly. After this test, the obstacle detection algorithms were modified to give no special treatment to obstacles within the blind spot.

The third significant incident during qualifications occurred in area B, the course designed to test navigation and driving skills. During the test, Boss came up behind a pair of cars parked along the edge of the road. The cars were not sufficiently in the road to be considered stopped vehicles by the perception system, so it fell to the motion planning system to avoid them. Owing to pessimistic parameter settings, the motion planner believed that there was insufficient room to navigate around the obstacles without leaving the lane, invoking the behavioral error recovery system. Because of a bug in the error handling system, the goal requested by the behavioral engine was located approximately 30 m behind Boss's current position, causing it to back up. During the backup maneuver DARPA paused Boss. This cleared the

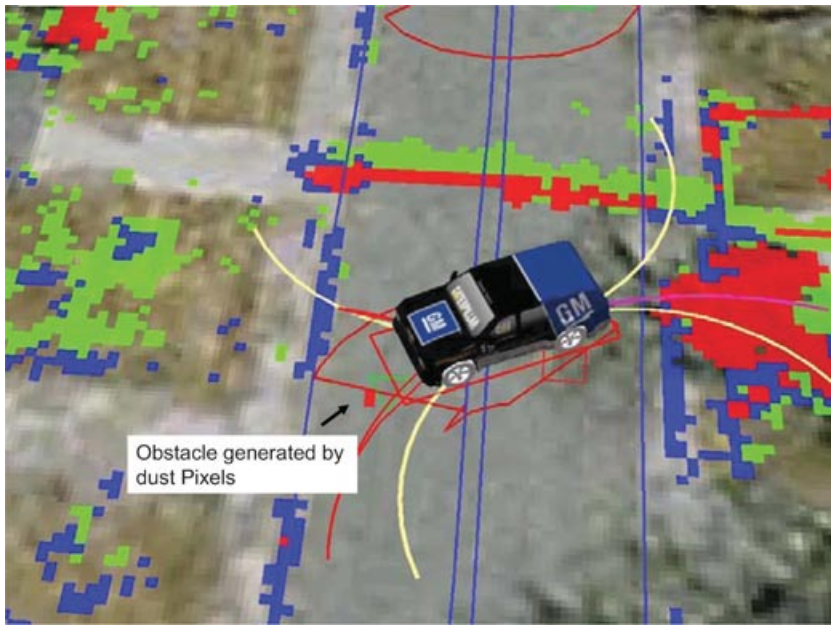

Figure 29. The false obstacles generated by dust and the bush behind Boss prevented Boss from initially completing its U-turn.

error recovery stack, and upon restart, Boss continued along the lane normally until it encountered the same vehicles, at which point it invoked the recovery system again, but due to the pause clearing state in the behavioral system, the new recovery goal was in a correct, down-road location.

Despite these foibles, after completion of the qualification events, Boss was ranked as the top performer and was given pole position for the final event.

9.2. Final Event Analysis

Immediately before Boss was to begin the final event, the team noticed that Boss's GPS receivers were not receiving GPS signals. Through equipment tests and observation of the surroundings, it was determined that the most likely cause of this problem was jamming from a Jumbotron (a large television commonly used at major sporting events) newly positioned near the start area. After a quick conference with the DARPA officials, the Jumbotron was shut down and Boss's GPS receivers were restarted and ready to go. Boss smoothly departed the launch area to commence its first of three missions for the day.

Once launched, Boss contended with 10 other autonomous vehicles and approximately 50 other human-driven vehicles. During the event Boss performed well but did have a few occurrences of unusual behavior.

The first incident occurred on a relatively bumpy transition from a dirt road to a paved road. Instead of stopping and then continuing normally, Boss stopped for a prolonged period and then hesitantly turned onto the paved road. Boss's hesitation was due to a calibration error in the Velodyne LIDAR, used for static obstacle detection. In this case, the ground slope combined with this miscalibration was sufficient for the laser to momentarily detect the ground, calling it an obstacle (see Figure 30). Boss then entered an error recovery mode, invoking the zone planner to get to the road. With each movement, the false obstacles in front of Boss would change position, causing replanning and thus hesitant behavior. Once Boss made it off the curb and onto the road, the false obstacles cleared and Boss was able to continue driving normally.

The next incident occurred when Boss swerved abruptly while driving past an oncoming robot. The oncoming vehicle was partially in Boss's lane but

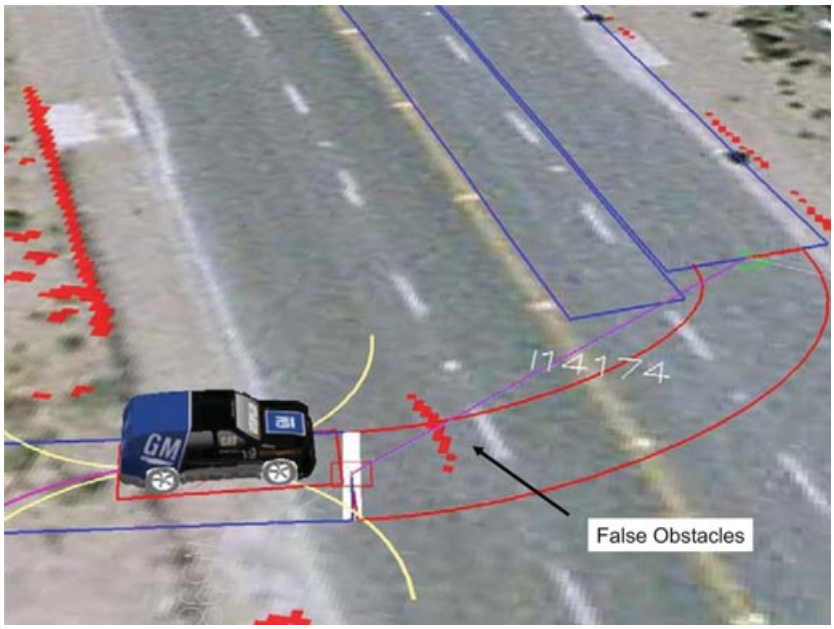

Figure 30. False obstacles that caused Boss to stutter when leaving a dirt road.

not sufficiently that Boss needed to maneuver to avoid it. The oncoming vehicle partially occluded its chase vehicle and momentarily caused Boss's perception system to estimate the vehicle with an incorrect orientation such that it was perceived to be entering Boss's travel lane (see Figure 31). At this point, Boss swerved and braked to try and avoid the perceived oncoming vehicle, moving it very close to the Jersey barrier wall. While this was happening, DARPA ordered a full course pause due to activity elsewhere on the course. This interrupted Boss midmotion, causing it to be unable to finish its maneuver. Upon awakening from the pause, Boss felt it was too close to the wall to maneuver safely. After a minute or so of near-stationary wheel turning, its pose estimate shifted laterally by about 2 cm, just enough that it believed that it had enough room to avoid the near wall. With a safe path ahead, Boss resumed driving and continued along its route.

Later in the first mission Boss was forced to queue behind a vehicle already waiting at a stop line. Boss began queuing properly, but when the lead vehicle pulled forward, Boss did not. After an excessive delay, Boss performed a U-turn and drove away from the intersection, taking an alternative route to its next checkpoint.

Analysis revealed a minor bug in the planning system, which did not correctly update the location of moving vehicles. For efficiency, the zone planner checks goal locations against the location of obstacles before attempting to generate a path. This check can be performed efficiently, allowing the planning system to report that a goal is unreachable in much less time than it takes to perform an exhaustive search. A defect in this implementation caused the planning system to not correctly update the list of moving obstacles prior to performing this check. Thus, after the planner's first attempt to plan a path forward failed, every future attempt to plan to the same location failed, because the planner erroneously believed that a stopped vehicle was in front of it. The behavioral error recovery system eventually selected a U-turn goal, and Boss backed up and went on its way. This reroute caused Boss to drive an additional 2.7 km, but it was still able to complete the mission.

Despite these incidents, Boss was able to finish the challenge in 4 h, 10 min, and 20 s, roughly 19 min faster than the second-place competitor. Boss averaged 22.5 km/h during the challenge (while enabled) and 24.5 km/h when moving. Through offline simulation, we estimated that the maximum average speed Boss could have obtained over the course was

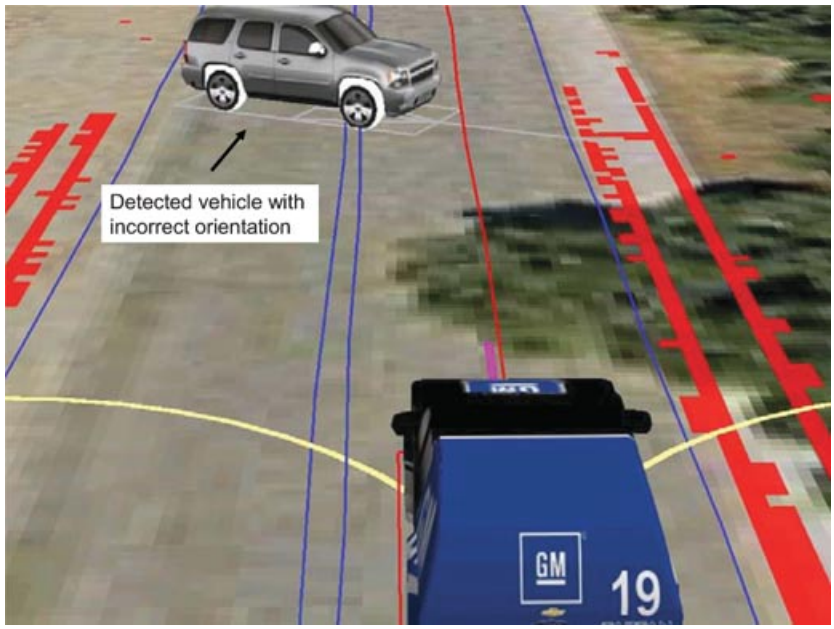

Figure 31. This incorrect estimate of an oncoming vehicle's orientation caused Boss to swerve and almost to become irrevocably stuck.

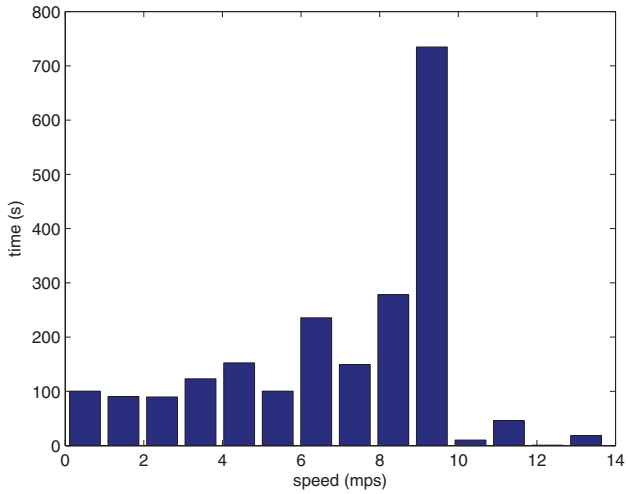

Figure 32. Boss averaged 22.5 km/h during the challenge; this figure shows a distribution of the vehicle's speed while moving.

26.2 km/h. Figure 32 shows the distribution of Boss's moving speeds during the challenge. The large peak at 9–10 m/s is due to the course speed limits. This spike implies that Boss was limited by these speed limits, not by its capability.

The roadmap localization system played an important role during the challenge. For a majority of the challenge, the error estimate in the roadmap localization system was less than 0.5 m, but there was more than 16 min when the error was greater than

0.5 m, with a peak error of 2.5 m. Had the road map localization system not been active, it is likely that Boss would have been either off the road or in a wrong lane for a significant amount of time.

In general, Boss drove well, completing a large majority of the course with skill and precision. As demonstrated in this brief analysis, one of Boss's strengths was its robustness and ability to recover from unexpected error cases autonomously.

10. LESSONS LEARNED

Through the development of Boss and competition in the Urban Challenge, the team learned several valuable lessons:

Available off-the-shelf sensors are insufficient for urban driving. Currently no single sensor is 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 the road. In urban environments, the shape of roads changes 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 our team, and demonstrated that it was feasible on a small to medium scale. Whereas this may not be a viable approach for all roads, it has proven to be a viable method for reducing complexity in common urban driving scenarios. The next step will be to apply the same approach on a large or national scale and automate the detection of when the road has changed shape from the expected.

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 is not a vehicle: if it is observed to move within some small time window and is in 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. Although subsystems 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 marketplace, 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, but unscripted, ways. Although this can increase robustness, it can in no way guarantee that the system is correct.

Sliding autonomy will reduce the 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 that the vehicle is sufficiently capable that it does not request help so frequently that the benefits of autonomy are lost. As an example, if Boss was allowed to ask for help during the 4-h Urban Challenge, there were three occasions on which it might have requested assistance. Human intervention at these times would likely have reduced Boss's overall mission time by approximately 15 min.

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 that 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.

11. CONCLUSIONS

The Urban Challenge was a tremendously exciting program to take part in. The aggressive technology development timeline, international competition, and compelling motivations fostered an environment that brought out a tremendous level of creativity and effort from all those involved. This research effort 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 drive on roads where a priori road geometry both is and is not available
- 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 rapid development and testing of highly capable autonomous vehicles

Although this article outlines the algorithms and 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 all contribute to this complexity.

As the field advances to address these problems, we will be faced with secondary problems, such as, How do we test these systems and how will society accept them? Although defense needs may provide the momentum necessary to drive these promising technologies, we must work hard to ensure our that work is relevant and beneficial to a broader society. Whereas these challenges loom large, it is clear that there is a bright and non-too-distant future for autonomous vehicles.

ACKNOWLEDGMENTS

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

Committee on Army Unmanned Ground Vehicle Technology and the National Research Council. (2002) Technology Development for Army Unmanned Ground Vehicles. Washington, DC: Author.

Darms, M. (2007) Eine Basis-Systemarchitektur zur Sensordatenfusion von Umfeldsensoren für Fahrerassistenzsysteme, Fortschrittb. VDI: R12, Nr. 653. Dusseldorf, Germany: VDI-Verlag.

Darms, M., Baker, C., Rybski, P., & Urmson, C. (2008). Vehicle detection and tracking for the Urban Challenge—The approach taken by Tartan Racing. In M. Maurer & C. Stiller (Eds.), 5. Workshop Fahrerassistenzsysteme (pp. 57–67). Karlsruhe, Germany: FMRT.

Darms, M., Rybski, P., & Urmson, C. (2008a). An adaptive model switching approach for a multisensor tracking system used for autonomous driving in an urban environment. AUTOREG 2008. Steuerung und Regelung von Fahrzeugen und Motoren: 4. Fachtagung, Baden Baden, Germany (pp. 521–530). Dusseldorf, Germany: VDI-Verlag.

Darms, M., Rybski, P., & Urmson, C. (2008b). Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments. In Proceedings of the 2008 IEEE Intelligent Vehicles Symposium, Eindhoven, the Netherlands (pp. 1192–1202). IEEE.

Darms, M., & Winner, H. (2005). A modular system architecture for sensor data processing of ADAS applications. In Proceedings of the 2008 Intelligent Vehicles Symposium, Las Vegas, NV (pp. 729–734). IEEE.

DARPA Urban Challenge. (2007). http://www.darpa.mil/grandchallenge/index.asp.

Daubechies, I. (1992). Ten lectures on wavelets. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Duda, R. O. & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. Communications of the ACM, 15, 11–15.

Ferguson, D., Howard, T., & Likhachev, M. (2008, submitted for publication). Motion planning in urban environments.

Howard, T.M., and Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. International Journal of Robotics Research, 26(2), 141–166.

Huttenlocker, D., & Felzenswalb, P. (2004). Distance transforms of sampled functions (Tech. Rep. TR2004-1963). Ithaca, NY: Cornell Computing and Information Science.

Kaempchen, N., Weiss, K., Schaefer, M., & Dietmayer, K.C.J. (2004, June). IMM object tracking for high dynamic driving maneuvers. In IEEE Intelligent Vehicles Symposium 2004, Parma, Italy (pp 825–830). Piscataway, NJ: IEEE.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2005). Anytime dynamic A*: An anytime, replanning algorithm. In S. Biundo, K. L. Myers, & K. Rajan (Eds.), Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), Monterey, CA. AAAI.

MacLachlan, R. (2005, June). Tracking moving objects from a moving vehicle using a laser scanner (Tech. Rep. CMU-RI-TR-05-07). Pittsburgh, PA: Carnegie Mellon University.

Shih, M-Y., and Tseng, D.-C. (2005). A wavelet-based multi-resolution edge detection and tracking. Image and Vision Computing, 23(4), 441–451.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., & Mahoney, P. (2006). Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics, 23(9), 661–692.

Urmson, C., Anhalt, J., Bartz, D., Clark, M., Galatali, T., Gutierrez, A., Harbaugh. S., Johnston, J., Kato, H., Koon, P.L., Messner, W., Miller, N., Mosher, A., Peterson, K., Ragusa, C., Ray, D., Smith, B.K., Snider, J.M., Spiker, S., Struble, J.C., Ziglar, J., & Whittaker, W.L. (2006). A robust approach to high-speed navigation for unrehearsed desert terrain. Journal of Field Robotics, 23(8), 467–508.

Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J.P., Gowdy, J., Gutierrez, A., Harbaugh, S., Johnson-Roberson, M., Kato, H., Koon, P.L., Peterson, K., Smith, B.K., Spiker, S., Tryzelaar, E., & Whittaker, W.L. (2004). High speed navigation of unrehearsed terrain: Red Team technology for Grand Challenge 2004 (Tech. Rep. CMU-RI-TR-04-37). Pittsburgh, PA: Robotics Institute, Carnegie Mellon University.

Viola, P., & Jones, M. (2001). Robust real-time object detection. In Proceedings of IEEE Workshop on Statistical and Computational Theories of Vision, Vancouver, BC, Canada. Available at: http://www.stat.ucla.edu/~sczhu/Workshops/SCTV2001.html.