Motion Planning in Urban Environments

Dave Ferguson
Intel Research Pittsburgh
Pittsburgh, Pennsylvania 15213
e-mail: dave.ferguson@intel.com

Thomas M. Howard
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
e-mail: thoward@ri.cmu.edu

Maxim Likhachev
University of Pennsylvania
Philadelphia, Pennsylvania 19104
e-mail: maximl@seas.upenn.edu

Received 7 July 2008; accepted 6 September 2008

We present the motion planning framework for an autonomous vehicle navigating through urban environments. Such environments present a number of motion planning challenges, including ultrareliability, high-speed operation, complex intervehicle interaction, parking in large unstructured lots, and constrained maneuvers. Our approach combines a model-predictive trajectory generation algorithm for computing dynamically feasible actions with two higher level planners for generating long-range plans in both on-road and unstructured areas of the environment. In the first part of this article, we describe the underlying trajectory generator and the on-road planning component of this system. We then describe the unstructured planning component of this system used for navigating through parking lots and recovering from anomalous on-road scenarios. Throughout, we provide examples and results from "Boss," an autonomous sport utility vehicle that has driven itself over 3,000 km and competed in, and won, the DARPA Urban Challenge.

1. INTRODUCTION

Autonomous passenger vehicles present an incredible opportunity for the field of robotics and society at large. Such technology could drastically improve safety on roads, provide independence to millions of people unable to drive because of age or ability, revolutionize the transportation industry, and reduce the danger associated with military convoy operations. However, developing robotic systems that are sophisticated enough and reliable enough to operate in everyday driving scenarios is tough. As a result, up until very recently, autonomous vehicle technology has been limited to either off-road, unstructured environments where complex interaction with other vehicles is nonexistent (Carsten, Rankin, Ferguson,

Journal of Field Robotics 25(11–12), 939–960 (2008) © 2008 Wiley Periodicals, Inc.
Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/rob.20265

& Stentz, 2007; Iagremma & Buehler, 2006a, 2006b; Kelly, 1995; Singh et al., 2000; Stentz & Hebert, 1995) or very simple on-road maneuvers such as highway-based lane following (Thorpe, Jochem, & Pomerleau, 1997).

The DARPA Urban Challenge competition was designed to extend this technology as far as possible toward the goal of unrestricted on-road driving. The event consisted of an autonomous vehicle race through an urban environment containing single- and multilane roads, traffic circles and intersections, open areas and unpaved sections, road blockages, and complex parking tasks. Successful vehicles had to travel roughly 60 miles, all in the presence of other human-driven and autonomous vehicles, and all while abiding by speed limits and California driving rules.

This challenge required significant advances over the state of the art in autonomous vehicle technology. In this paper, we describe the motion planning system developed for Carnegie Mellon University's winning entry in the Urban Challenge, "Boss." This system enabled Boss to travel extremely quickly through the urban environment to complete its missions; interact safely and intelligently with obstacles and other vehicles on roads, at intersections, and in parking lots; and perform sophisticated maneuvers to solve complex parking tasks.

We first introduce very briefly the software architecture used by Boss and the role of motion planning within that architecture. We then describe the trajectory generation algorithm used to generate every move of the vehicle. In Section 5, we discuss the motion planning framework used when navigating on roads.

In Section 6, we discuss the framework used when navigating through unstructured areas or performing complex maneuvers. We then provide results and discussion from hundreds of hours and thousands of miles of testing and describe related work in both on-road and unstructured planning.

2. SYSTEM ARCHITECTURE

Boss's software system is decomposed into four major blocks (see Figure 1) and is described in detail in (Urmson et al., 2008). The perception component fuses and processes data from Boss's sensors to provide key environmental information, including the following:

- **vehicle state**, globally referenced position, attitude, and speed for Boss;
- **road world model**, globally referenced geometric information about the roads, parking zones, and intersections in the world;
- **moving obstacle set**, an estimation of other vehicles in the vicinity of Boss;
- **static obstacle map**, a two-dimensional (2D) grid representation of free, dangerous, and lethal space in the world; and
- **road blockages**, an estimation of clearly impassable road sections.

The mission planning component computes the fastest route through the road network to reach the

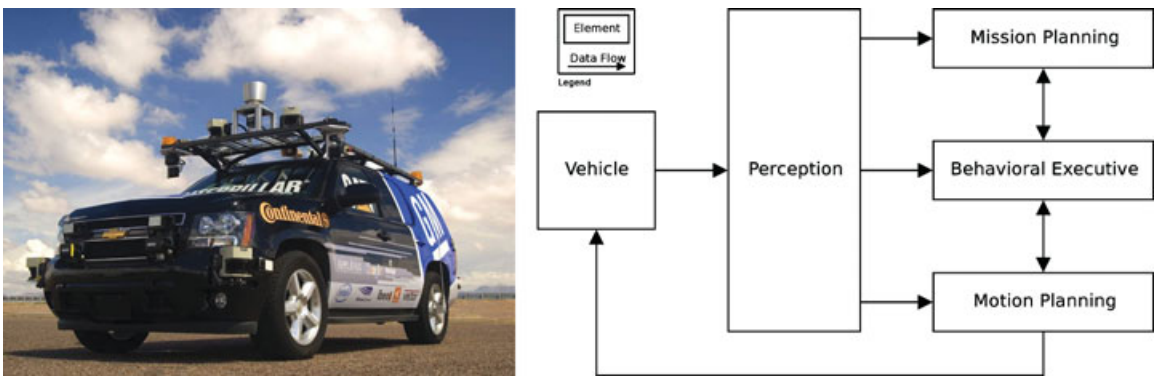

Figure 1. Boss: Carnegie Mellon's winning entry in the Urban Challenge, along with its software system architecture.

next checkpoint in the mission, based on knowledge of road blockages, speed limits, and the nominal time required to make special maneuvers such as lane changes or U-turns.

The behavioral executive combines the strategic global information provided by mission planning with local traffic and obstacle information provided by perception and generates a sequence of local tasks for the motion planner. It is responsible for the system's adherence to various rules of the road, especially those concerning structured interactions with other traffic and road blockages, and for detection of and recovery from anomalous situations. The local tasks it feeds to the motion planner take the form of discrete motion goals, such as driving along a road lane to a specific point or maneuvering to a specific pose or parking spot. The issuance of these goals is predicated on traffic concerns such as precedence among vehicles stopped at an intersection. In the case of driving along a road, desired lane and speed commands are given to the motion planner to implement behaviors such as distance keeping, passing maneuvers, and queuing in stop-and-go traffic.

The motion planning component takes the motion goal from the behavioral executive and generates and executes a trajectory that will safely drive Boss toward this goal, as described in the following section. Two broad contexts for motion planning exist: on-road driving and unstructured driving.

3. MOTION PLANNING

The motion planning layer is responsible for executing the current motion goal issued from the behavioral executive. This goal may be a location within a road lane when performing nominal on-road driving, a location within a parking lot or obstacle field when traversing through one of these areas, or any location in the environment when performing error recovery. The motion planner constrains itself based on the context of the goal and the environment to abide by the rules of the road.

Figure 2 provides a basic illustration of the nature of the goals provided by the behavioral executive. During nominal on-road driving, the goal entails a desired lane and a desired position within that lane (typically a stop line at the end of the lane). In such cases, the motion planner invokes a high-speed, on-road planner to generate a path that tracks the desired lane. During unstructured driving, such as when navigating through parking lots, the goal consists of a desired pose of the vehicle in the world. In these cases, the motion planner invokes a four-dimensional (4D) lattice planner that generates a global path to the desired pose. These unstructured motion goals are also used when the vehicle encounters an anomalous situation during on-road driving and needs to perform a complex maneuver (such as when an intersection is partially blocked and cannot be traversed in the desired lane).

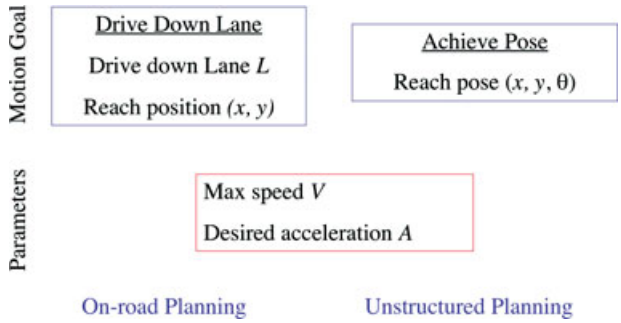

Figure 2. Motion goals provided by the behavioral executive to the motion planner. Also shown are the frequently updated speed and desired acceleration commands.

As well as issuing motion goals, the behavioral executive is constantly providing desired maximum speed and acceleration/deceleration commands to the motion planner. It is through this interface that the behavioral executive is able to control the vehicle's forward progress in distance keeping and intersection precedence scenarios. When the vehicle is not constrained by such scenarios, the motion planner computes desired speeds and accelerations based on the constraints of the environment itself (e.g., road curvature and speed limits).

Given a motion goal, 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 varying degrees and selecting from this set the best trajectory according to an evaluation function. Each of these candidate trajectories is computed using a trajectory generation algorithm described in the next section. As mentioned above, the nature of the generated path being followed differs based on the context of the motion goal and the environment. In addition, the evaluation function differs depending on the context but always includes consideration of static and dynamic obstacles, curbs, speed, curvature, and deviation from the path. The selected trajectory is then directly executed by the

vehicle. This process is repeated at 10 Hz by the motion planner.

4. TRAJECTORY GENERATION

Each candidate trajectory is computed using a model-predictive trajectory generator from Howard and Kelly (2007) that produces dynamically feasible actions between initial and desired vehicle states. In general, this algorithm can be used to solve the problem of generating a set of parameterized controls $[\mathbf{u}(\mathbf{p}, \mathbf{x})]$ that satisfy a set of state constraints whose dynamics can be expressed by a set of differential equations:

$$\mathbf{x} = \begin{bmatrix} x & y & \theta & \kappa & v & \dots \end{bmatrix}^T, \tag{1}$$

$$\dot{\mathbf{x}}(\mathbf{x}, \mathbf{p}) = f(\mathbf{x}, \mathbf{u}(\mathbf{p}, \mathbf{x})), \tag{2}$$

where $\mathbf{x}$ is the vehicle state (with position x, y, heading θ, curvature κ, and velocity v, as well as other state parameters such as commanded velocity) and $\mathbf{p}$ is the set of parameters for which we are solving. The derivative of vehicle state $\dot{\mathbf{x}}$ is a function of both the parameterized control input $\mathbf{u}(\mathbf{p}, \mathbf{x})$ and the vehicle state $\mathbf{x}$ because the vehicle's response to a particular control input is state dependent. In this section, we describe the application of this general algorithm to our domain, specifically addressing the state constraints, vehicle model, control parameterization, initialization function, and trajectory optimization approaches used.

4.1. State Constraints

For navigating both on-road and unstructured areas of urban environments, we generated trajectories that satisfied both target 2D position (x, y) and heading (θ) constraints. We defined the constraint equation formula $[\mathbf{C}(\mathbf{x}, \mathbf{p})]$ as the difference between these target boundary state constraints (denoted $\mathbf{x_C}$) and the integral of the model dynamics (the endpoint of the computed vehicle trajectory):

$$\mathbf{x_C} = \begin{bmatrix} x_C & y_C & \theta_C \end{bmatrix}^T, \tag{3}$$

$$\mathbf{x_F}(\mathbf{p}, \mathbf{x}) = \mathbf{x_I} + \int_0^{t_f} \dot{\mathbf{x}}(\mathbf{x}, \mathbf{p}) dt, \tag{4}$$

$$\mathbf{C}(\mathbf{x}, \mathbf{p}) = \mathbf{x_C} - \mathbf{x_F}(\mathbf{p}, \mathbf{x}). \tag{5}$$

The constrained trajectory generation algorithm determines the control parameters $\mathbf{p}$ that drive Eq. (5) to zero. This results in a trajectory from an initial state $\mathbf{x_I}$ to a terminal vehicle state $\mathbf{x_F}$ that is as close as possible to the desired terminal state $\mathbf{x_C}$.

4.2. Vehicle Modeling

The development of a high-fidelity vehicle predictive motion model is important for the accurate prediction of vehicle motion and thus for the generation of accurate trajectories using our constraint-based approach. Our vehicle model consists of a set of parameterized functions that were fitted to data extracted from human-driven performance runs in the vehicle. The key parameters in our model are the controller delay, the curvature limit (the minimum turning radius), the curvature rate limit (a function of the maximum speed at which the steering wheel can be turned), and the maximum acceleration and deceleration of the vehicle. The controller delay accurately predicts the difference in time between a command from software and the corresponding initial response from hardware and is an important consideration when navigating at high speeds. The curvature, rate of curvature, and acceleration and deceleration limits were essential for accurately predicting the response of the vehicle over entire trajectories. This model is then simulated using a fixed-timestep Euler integration to predict the vehicle's motion. The Appendix provides details of the model used for Boss.

4.3. Controls Parameterization

For Ackermann-steered vehicles, it is advantageous to parameterize the curvature function in terms of arclength $[\kappa(\mathbf{p}, s)]$. This is because, for similar trajectories, the solution values for the curvature profile parameters are less dependent on the speed at which the trajectory is executed. For Boss, we parameterize the vehicle controls with a time-based linear velocity function $[v(\mathbf{p}, t)]$ and an arclength-based curvature function $[\kappa(\mathbf{p}, s)]$:

$$\mathbf{u}(\mathbf{p}, \mathbf{x}) = \begin{bmatrix} v(\mathbf{p}, t) & \kappa(\mathbf{p}, s) \end{bmatrix}^T. \tag{6}$$

We allow the linear velocity profile to take the form of a constant profile, linear profile, linear ramp profile, or trapezoidal profile [see Figure 3(a)]. The motion planner selects the appropriate profile based on the driving mode and context (e.g., maintaining a constant velocity for distance keeping or slowing down for an upcoming intersection). Each of these

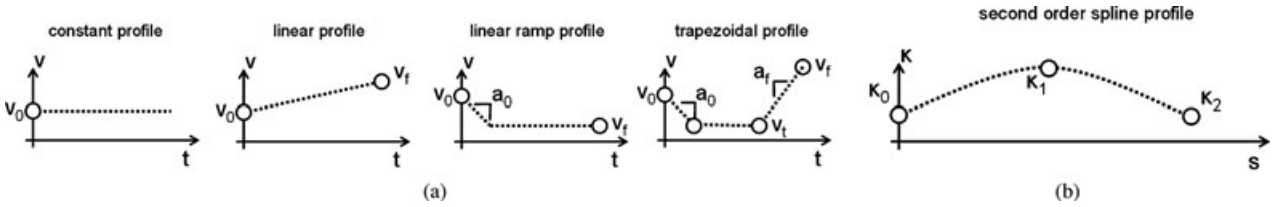

Figure 3. Velocity and curvature profiles. (a) Several different linear velocity profiles were applied in this system, each with its own parameterization and application. Each parameterization contains some subset of velocity and acceleration knot points (v_0, v_t, v_f, a_0, and a_f) and the length of the path, measured in time (t_0, t_f). (b) The curvature profile includes four possible degrees of freedom: the three spline knot points (κ_0, κ_1, and κ_2) and the length of the path (s_f).

profiles consists of a set of dependent parameters (v_0, v_t, v_f, a_0, and a_f) and the time to complete the profile (t_0, t_f), all of which become members of the parameter set $\mathbf{p}$. Because all of the dependent profile parameters are typically known, no optimization is done on the shape of each of these profiles.

The curvature profile defines the shape of the trajectory and is the primary profile over which optimization is performed. Our profile consists of three independent curvature knot point parameters (κ_0, κ_1, and κ_2) and the trajectory length s_f [see Figure 3(b)]. In general, it is important to limit the degrees of freedom in the system to minimize the presence of local optima and to improve the runtime performance of the algorithm (which is approximately linear with the number of free parameters in the system) but maintain enough flexibility to satisfy all of the boundary state constraints. We chose a second-order spline profile because it contains four degrees of freedom, enough to satisfy the three boundary state constraints. We further fix the initial command knot point κ_0 during the optimization process to the curvature at the initial state $\mathbf{x_I}$ to provide smooth controls.[1]

With the linear velocity profile's dependent parameters being fully defined and the initial spline parameter of the curvature profile fixed, we are left with a system with three parameterized freedoms, that is, the latter two curvature spline knot points and the trajectory length:

$$\mathbf{p_{free}} = \begin{bmatrix} \kappa_1 & \kappa_2 & s_f \end{bmatrix}^T . \tag{7}$$

[1]However, this can also be fixed to a different value to produce sharp trajectories, as described in Section 5.2.

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. Arclength was used for the independent parameter for the curvature profiles because the shape of these profiles is somewhat independent of the speed at which they are traveled. This allows solutions with similar parameters for varying linear velocity profiles.

4.4. Initialization Function

Given the three free parameters and the three constraints in our system, we can use various optimization or root-finding techniques to solve for the parameter values that minimize our constraint equation. However, for efficiency it is beneficial to precompute offline an approximate mapping from relative state constraint space to parameter space to seed the constraint optimization process. This mapping can drastically speed up the algorithm by placing the initial guess of the control parameters close to the desired solution, reducing the number of online optimization steps required to reach the solution (within a desired precision). Given the high number of state variables and the fact that the system dynamics cannot be integrated in closed form, it is infeasible to precompute the entire mapping of state space to input space for any nontrivial system, such as the Boss vehicle model. We instead generate an approximation of this mapping through a five-dimensional (5D) lookup table with varying relative initial and terminal position $\triangle x, \triangle y$, relative heading $\triangle\theta$, initial curvature κ_i, and constant velocities v. Because this is only an approximation some optimization is usually required;

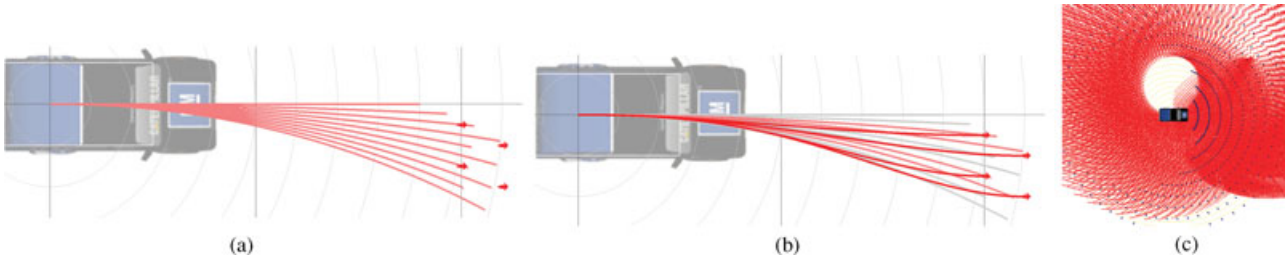

Figure 4. Offline lookup table generation. (a) Some sampled trajectories (in red) and some table endpoints (red arrows) that we wish to generate trajectories to for storage in the lookup table. (b) The closest sampled trajectories to the desired table endpoints are selected and then optimized to reach the desired endpoints. The parameter sets corresponding to these optimized trajectories are then stored in the lookup table. (c) The set of all sampled trajectories (red) and table endpoints (blue) for a single initial vehicle state. In this case the initial vehicle curvature is not zero, so the set of trajectories is not symmetric about the vehicle.

however, the initial seed from the lookup table significantly reduces the number of optimization iterations required from an arbitrary set of parameters.

Figure 4 provides an illustration of the lookup table generation process. First, the five dimensions of interest are discretized into a (5D) table. Next, uniform sampling is used to sample from the set of all possible parameter values and the table positions each of these sample trajectories terminate in are recorded. The 5D table is then stepped through, and for each position in the table the sample parameter values that come closest to this position are found. This parameter set is then optimized (using the optimization technique presented in the next section) to accurately match the table position, and then the resulting parameter set is stored in the corresponding index of the 5D table.

4.5. Trajectory Optimization

Given a set of parameters $\mathbf{p}$ that provide an approximate solution, it is then necessary to optimize these parameters to reduce the endpoint error and "snap" the corresponding trajectory to the desired terminal state.[2] To do this, we linearize and invert our system of equations to produce a correction factor for the free control parameters based on the product of the inverted Jacobian and the current boundary state error. The Jacobian can be determined numerically through central differences of simulated vehicle actions:

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

The control parameters are modified until the residual of the boundary state constraints is within an acceptable bound or until the optimization process diverges. In situations in which boundary states are unachievable due to vehicle limitations, the optimization process predictably diverges as the partial derivatives in the Jacobian approach zero. The optimization history is then searched for the best candidate action (the most aggressive action that gets closest to the state constraints), and this candidate is accepted or rejected based on the magnitude of its error.

Figure 5 illustrates the online trajectory generation approach in action. Given a desired terminal state, we first look up from our table the closest terminal and initial states and their associated free parameter sets. We then interpolate between these closest parameter sets in 5D to produce our initial approximation of the parameter set to reach our desired terminal state. Figure 5(a) shows the lookup and interpolation steps, with the resulting parameter set values and corresponding trajectory. Figures 5(c)–5(e) show the interpolation process for the free parameters.[3] Next, we evaluate the endpoint error of the

[2]Depending on the desired terminal state accuracy, it is sometimes possible to use the approximate parameters from the lookup table without any further optimization.

[3]Note that, for illustration purposes, only a subset of the parameter sets used for interpolation is shown in these figures (the full interpolation is in 5D not 2D), and this is why the interpolated result does not lie within the convex hull of the four sample points shown.

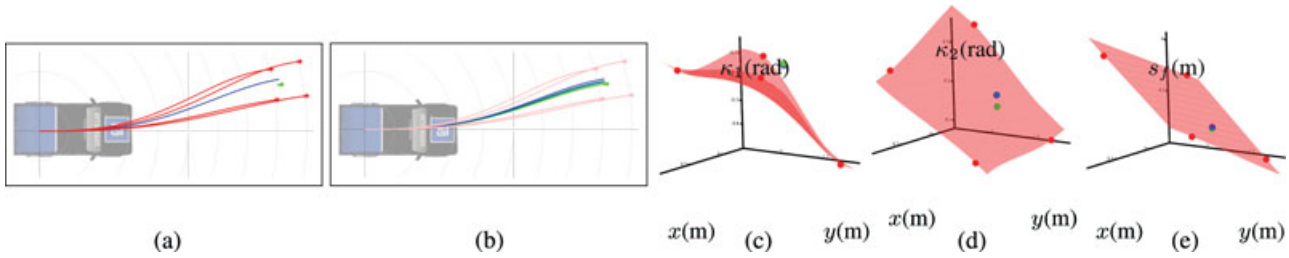

Figure 5. Online trajectory generation. (a) Given an initial state and desired terminal state (relative terminal state shown in green), we find the closest elements of the lookup table (in red) and interpolate between the control parameters associated with these elements (interpolation of the free parameters is shown in graphs c–e) to come up with an initial approximation of the free parameters (resulting corresponding trajectory shown in blue). (b) This approximate trajectory is then optimized by modifying the free parameters based on the endpoint error, resulting in a sequence of trajectories that get closer to the desired terminal state. When the endpoint error is within an acceptable bound, the most recent parameter set is returned (trajectory shown in green). The interpolation over the free parameters κ_1, κ_2, and s_f is shown by the three graphs c–e (interpolated solutions shown in blue, final optimized solutions shown in green).

resulting trajectory, and we use this error to modify our parameter values to get closer to our desired terminal state, using the optimization approach just described. We repeat this optimization step until our endpoint error is within an allowed bound of the desired state [see Figure 5(b)], and the resulting parameters and trajectory are stored and evaluated by the motion planner.

5. ON-ROAD PLANNING

5.1. Path Extraction

During on-road navigation, the motion goal from the behavioral executive 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, representing 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.

5.2. Trajectory Generation

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. Each of these trajectories originates from the predicted state that the vehicle will reach by the time the trajectories are executed. To calculate this state, forward prediction using an accurate vehicle model (the same model used in the trajectory generation phase) is performed using the trajectories selected for execution in previous planning episodes. This forward prediction accounts for both the high-level delays (the time required to plan) and the low-level delays (the time required to execute a command).

The goals are placed at a fixed longitudinal distance down the centerline path (based on the speed of the vehicle) but vary in lateral offset from the path to provide several options for the planner. The trajectory generation algorithm described above is used to compute dynamically feasible trajectories to these local goals. For each goal, two trajectories are generated: a smooth trajectory and a sharp trajectory. The smooth trajectory has the initial curvature parameter κ_0 fixed to the curvature of the forward-predicted vehicle state. The sharp trajectory has this parameter set to an offset value from the forward-predicted vehicle state curvature to produce a sharp initial action. These sharp trajectories are useful for providing quick responses to suddenly appearing obstacles or dangerous actions of other vehicles.

Figure 6 provides an example of smooth and sharp trajectories. The left-most image shows two trajectories (cyan and purple) generated to the same goal pose. The purple (smooth) trajectory exhibits continuous curvature control throughout; the cyan (sharp) trajectory begins with a discontinuous jump in commanded curvature, resulting in a sharp response from

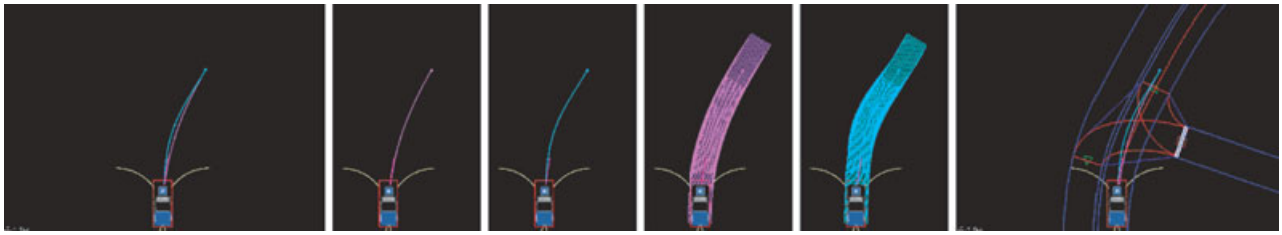

Figure 6. Smooth and sharp trajectories.

the vehicle. In these images, the initial curvature of the vehicle is shown by the short pink arc. The four center images show the individual sharp and smooth trajectories, along with the convolution of the vehicle along these trajectories. The right-most image illustrates how these trajectories are generated in practice for following a road lane.

5.3. Trajectory Velocity Profiles

The velocity profile used for each of the generated trajectories is selected from the set introduced in Section 4.3 based on several factors, including the maximum speed given from the behavioral executive based on safe following distance to the lead vehicle, the speed limit of the current road segment, the maximum velocity feasible given the curvature of the centerline path, and the desired velocity at the local goal (e.g., if it is a stop line).

In general, profiles are chosen that maximize the speed of the vehicle at all times. Thus, typically a linear ramp profile is used, with a ramp velocity equal to the maximum speed possible and a linear component corresponding to the maximum acceleration possible. If the vehicle is slowly approaching a stop line (or stopped short of the stop line), a trapezoidal profile is employed so that the vehicle can both reach the stop line quickly and come smoothly to a stop.

Multiple velocity profiles are considered for a particular trajectory when the initial profile results in a large endpoint error. This can occur when the rate of curvature required to reach the desired endpoint is not possible given the velocity imposed by the initial profile. In such cases, additional profiles with less aggressive speeds and accelerations are generated until either a valid trajectory is found or a maximum number have been evaluated (in our case, three per initial trajectory).

5.4. Trajectory Evaluation

The resulting set of 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, their endpoint error, and their speed. 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 a vehicle controller.

One of the challenges of navigating in urban environments is avoiding other moving vehicles. To do this robustly and efficiently, we predict the future behavior of these vehicles and collision-check our candidate trajectories in state-time space against these predictions. We do this collision checking efficiently by using a hierarchical algorithm that performs a series of intersection tests between bounding regions for our vehicle and each other vehicle, with each test successively more accurate (and more computationally expensive). See Ferguson, Darms, Urmson, and Kolski (2008) for more details on the algorithms used for prediction and efficient collision checking.

Figure 7 provides an example of the local planner following a road lane. Figure 7(b) shows the vehicle navigating down a two-lane road (detected obstacles and curbs shown as red and blue pixels, lane boundaries shown in blue, centerline of lane in red, current curvature of the vehicle shown in pink, minimum turning radius arcs shown in white) with a vehicle in the oncoming lane (in green). Figure 7(c)

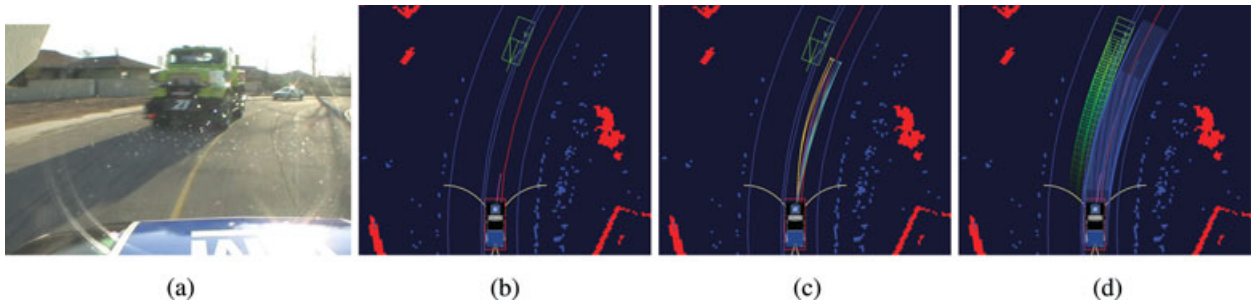

Figure 7. Following a road lane. These images show a single timeframe from the Urban Challenge.

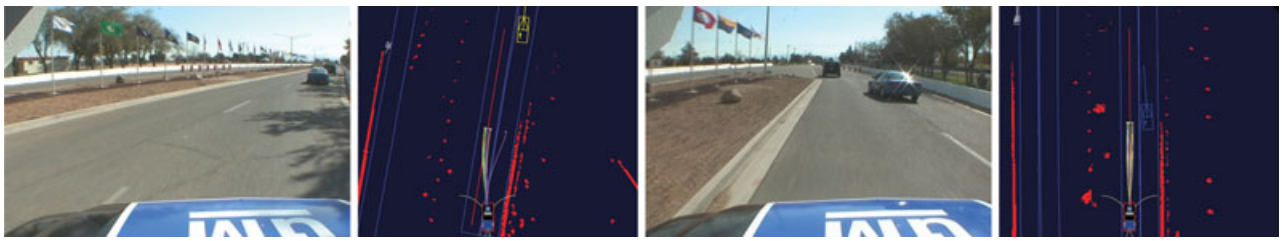

Figure 8. Performing a lane change reliably and safely. Here, Boss changed lanes because another robot's chase vehicle was traveling too slowly in its original lane.

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 7(d) shows the evaluation of one of these trajectories against both static and dynamic obstacles in the environment.

5.5. Lane Changing

As well as driving down the current lane, it is often necessary or desired in urban environments to perform lane changes. This may be to pass a slow or stalled vehicle in the current lane or move into an adjacent lane to prepare for an upcoming turn.

In our system, lane changes are commanded by the behavioral executive and implemented by the motion planner in a way similar to normal lane driving: a set of trajectories is generated along the centerline of the desired lane. However, because it is not always possible to perform the lane change immediately, an additional trajectory is generated along the current lane in case none of the desired lane trajectories is feasible. Also, to ensure smooth lane changes, no sharp trajectories are generated in the direction of the current lane. Figure 8 provides an example lane change performed during the Urban Challenge to pass a chase vehicle.

5.6. U-Turns

If the current road segment is blocked, the vehicle must be able to turn around and find another route to its destination. In this scenario, Boss uses information about the dimensions of the road to generate a smooth path that turns the vehicle around. Depending on how constrained the road is, this path may consist of a single forward segment (e.g., a traditional U-turn) or a three-point turn.[4] This path is then tracked in a fashion similar to the lane centerline paths, using a series of trajectories with varying offsets. Figure 9 provides an example three-point turn

[4]If the road is extremely narrow, even a three-point turn may not be possible. In such cases, a more powerful lattice planner is invoked, as described in Section 7.2.

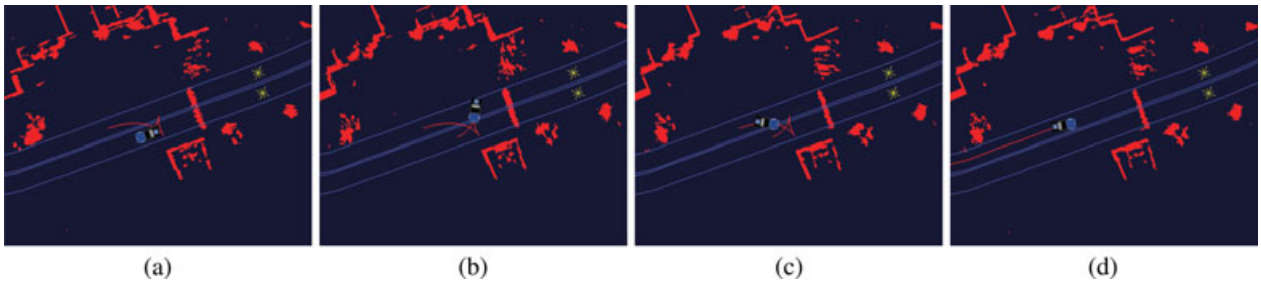

Figure 9. Performing a U-turn when encountering a road blockage. In this case the road was too narrow to perform a single forward action to turn around and a three-point turn was required. (a) Initial plan generated to reverse the direction of the vehicle. (b, c) Tracking the plan. (d) Reverting back to lane driving after the vehicle has completely turned around and is in the correct lane.

performed during one of the qualification events at the Urban Challenge.

5.7. Defensive Driving

One of the advanced requirements of the Urban Challenge was the ability to react safely to aberrant behavior of other vehicles. In particular, if another vehicle was detected traveling the wrong direction in Boss's lane, it was the responsibility of Boss to pull off the road in a defensive driving maneuver to avoid a collision with the vehicle. To implement this behavior, the behavioral executive closely monitors other vehicles and if one is detected traveling toward Boss in its lane, the motion planner is instructed to move Boss off the right side of the road and come to a stop. This is performed in a fashion similar to a lane change to a hallucinated lane off the road but with a heavily reduced velocity so that Boss does not leave the road traveling too quickly and then comes to a stop once it is completely off the road. After the vehicle has passed, Boss then plans back onto the road and continues (see Figure 10).

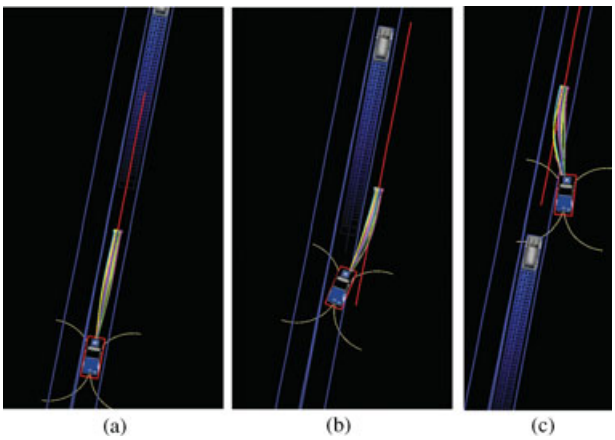

Figure 10. Defensive driving on roads. (a) Boss initially plans down its lane while the oncoming vehicle is far away. (b) When the oncoming vehicle is detected as dangerous, Boss generates a set of trajectories off the right side of the road. (c) After the oncoming vehicle has passed, Boss plans back onto the road and continues.

5.8. Error Detection and Recovery

A central focus of our system-level approach was the detection of and recovery from anomalous situations. In lane driving contexts, such situations usually presented themselves through the motion planner being unable to generate any feasible trajectories to track the desired lane (for instance, if the desired lane is partially blocked and the on-road motion planner cannot plan a path through the blockage). In such cases, the behavioral executive issues a motion goal that invokes the more powerful, yet more computationally expensive, 4D lattice motion planner. If this goal is achieved, the system resumes with lane driving. If the motion planner is unable to reach this goal, the behavioral executive continues to generate new goals for the lattice planner until one is satisfied. We provide more details on how the lattice planner interacts with these goals in the latter part of this article, and more details on the error detection and recovery process can be found in Baker, Ferguson, and Dolan (2008).

6. UNSTRUCTURED PLANNING

During unstructured navigation, the motion goal from the behavioral executive is a pose (or set of

poses) in the environment. The motion planner attempts to generate a trajectory that moves the vehicle toward this goal pose. However, driving in unstructured environments significantly differs from driving on roads. As mentioned in Section 5.1, when traveling on roads the desired lane implicitly provides a preferred path for the vehicle (the centerline of the lane). In unstructured environments 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, we use a lattice planner that searches over vehicle position x, y, orientation θ, and velocity v. The set of possible local maneuvers considered for each (x, y, θ, v) state in the planner's search space are constructed offline using the same vehicle model as used in trajectory generation, so that they can be accurately executed by the vehicle. This planner searches in a backward direction out from the goal pose(s) 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.

This global high-fidelity path is then tracked by a local planner that operates similarly to the on-road lane tracker, by generating a set of candidate trajectories that follow the path while allowing for some flexibility in local maneuvering. However, the nature of the trajectories generated in unstructured environments is slightly different. In the following sections, we describe in more detail the elements of our approach and how it exploits the context of its instantiation to adapt its behavior based on the situation (e.g., parking lot driving vs. off-road error recovery).

6.1. Planning Complex Maneuvers

To efficiently generate complex plans over large, obstacle-laden environments, the planner relies on an anytime, replanning search algorithm known as Anytime Dynamic A* (Anytime D*), developed by Likhachev, Ferguson, Gordon, Stentz, and 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. The algorithm is also able to provide control over the suboptimality bound of the solution at all times during planning. Figure 19 later in this paper shows an initial, suboptimal path converging over time to the optimal solution.

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, as in such a scenario updated information in the vicinity of the vehicle affects a smaller portion of the search space and so Anytime D* is able to reuse a large portion of its previously constructed search tree in

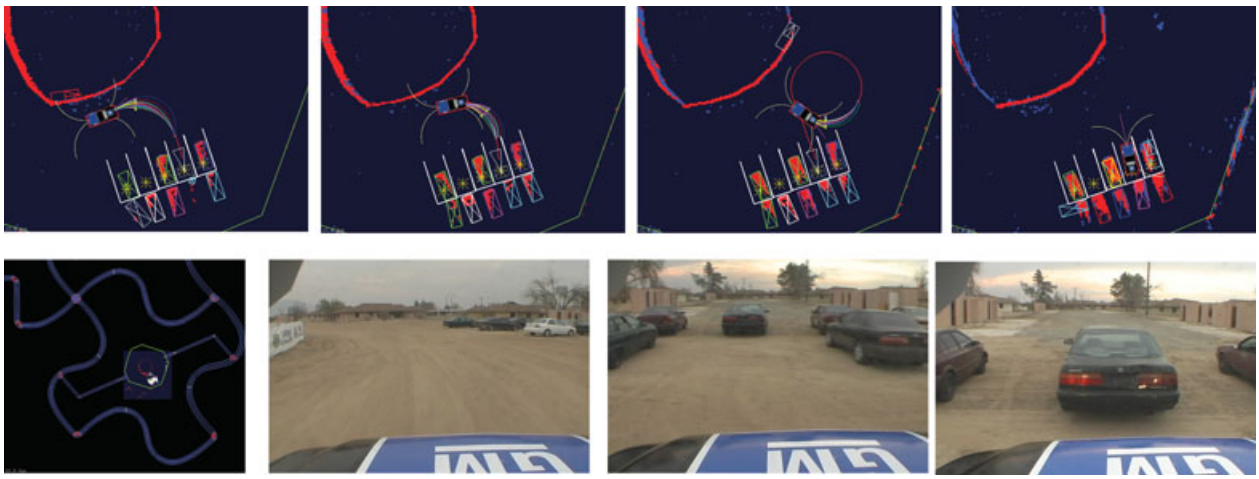

Figure 11. Replanning when new information is received.

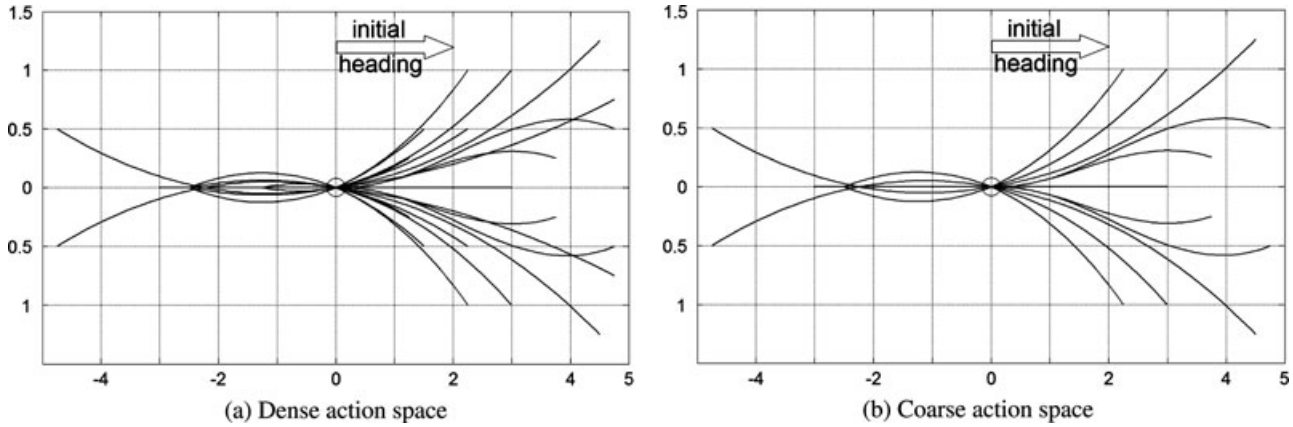

Figure 12. Dense and coarse resolution action spaces. The coarse action space contains many fewer actions (24 versus 36 in the dense action space) with transitions only to states with a coarse-resolution heading discretization (in our case, 16 headings versus 32 in the dense-resolution discretization). In both cases the discretization in position is 0.25 m and the axes in both diagrams are in meters.

recomputing a new path. Figure 11 illustrates this replanning capability. These images were taken from a parking task performed during the National Qualification Event (the bottom-left image shows the parking lot in green and the neighboring roads in blue). The top-left 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 (discussed later). As the vehicle gets closer to its intended spot, it observes a little more of the vehicle parked in the rightmost parking spot (top, second-from-left image). At this point, it realizes its current path is infeasible and replans a new path that has the vehicle perform a loop and pull in smoothly. This path was favored in terms of time over stopping and backing up to reposition.

To further improve efficiency, the lattice planner uses a multiresolution state and action space. In the vicinity of the goal and vehicle, where very complex maneuvering may be required, a dense set of actions and a fine-grained discretization of orientation are used during the search. In other areas, a coarser set of actions and discretization of orientation are employed. However, these coarse and dense resolution areas share the same dimensionality and seamlessly interface with each other, so that resulting solution paths overlapping both coarse and dense areas of the space are smooth and feasible. Figure 12 illustrates how the dense and coarse action and state spaces differ.

The effectiveness of the Anytime D* algorithm is highly dependent on its use of an informed heuristic to focus its search. An accurate heuristic can reduce the time and memory required to generate a solution by orders of magnitude, whereas a poor heuristic can diminish the benefits of the algorithm. It is thus important to devote careful consideration to the heuristic used for a given search space.

Because in our setup Anytime D* searches backward, the heuristic value of a state estimates the cost of a path from the robot pose to that state. Anytime D* requires these values to be admissible (not to overestimate the actual path cost) and consistent (Pearl, 1984). For any state x, y, θ, v, the heuristic we use is the maximum of two values. The first value is the cost of an optimal path from the robot pose to x, y, θ, v assuming a completely empty environment. These values are precomputed offline and stored in a heuristic lookup table (Knepper & Kelly, 2006). This is a very well-informed heuristic function when operating in sparse environments and is guaranteed to be admissible. The second value is the cost of a 2D path from the robot x_r, y_r coordinates to x, y given the actual environment. These values are computed online by a 2D grid-based Dijkstra's search. This second heuristic function is very useful when operating in obstacle-laden environments. By taking the maximum of these two heuristic values, we are able to

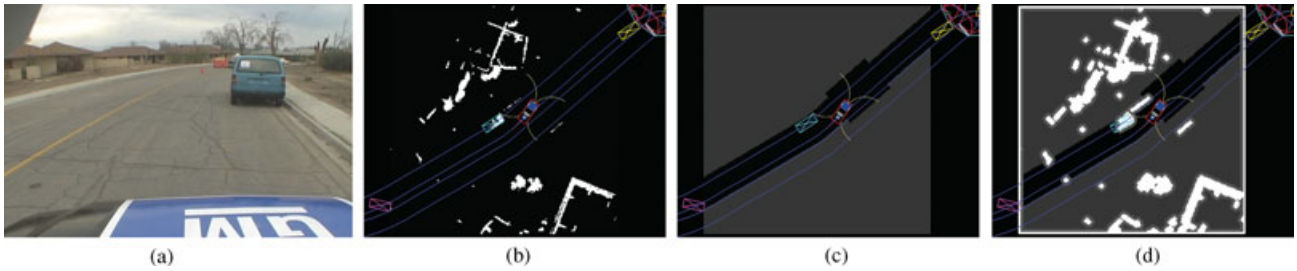

Figure 13. Snapshot from a qualification run during the Urban Challenge, showing (b) the obstacle map from perception (obstacles in white), (c) the constrained cost map based on the road structure (lighter areas are more costly), and (c) the resulting combined cost map used by the planner.

incorporate both the constraints of the vehicle and the constraints imposed by the obstacles in the environment. The result is a very well-informed heuristic function that can speed up the search by an order of magnitude relative to either of the component heuristics alone. For more details concerning the benefit of this combined heuristic function and other optimizations implemented in our lattice planner, including its multiresolution search space and how it efficiently performs convolutions and replanning, see Likhachev and Ferguson (2008) and Ferguson and Likhachev (2008).

6.2. Incorporating Environmental Constraints

In addition to the geometric obstacle information provided by perception, we incorporate context-specific constraints on the movement of the vehicle by creating an additional cost map known as a constrained map. This 2D grid-based cost map encodes the relative desirability of different areas of the environment based on the road structure in the vicinity and, if available, prior terrain information. This constrained cost map is then combined with the static map from perception to create the final combined cost map to be used by the lattice planner. Specifically, for each cell (i, j) in the combined cost map $\mathcal{C}$, the value of $\mathcal{C}(i, j)$ is computed as the maximum of $\mathcal{EPC}(i, j)$ and $\mathcal{CO}(i, j)$, where $\mathcal{EPC}(i, j)$ is the static map value at (i, j) and $\mathcal{CO}(i, j)$ is the constrained cost map value at (i, j).

For instance, when invoking the lattice planner to plan a maneuver around a parked car or jammed intersection, the constrained cost map is used to specify that staying within the desired road lane is preferable to traveling in an oncoming lane and similarly that driving off-road to navigate through a cluttered intersection is dangerous. To do this, undesirable areas of the environment based on the road structure are assigned high costs in the constrained cost map. These can be both soft constraints (undesirable but allowed areas), which correspond to high costs, and hard constraints (forbidden areas), which correspond to infinite costs. Figure 13 shows the constrained cost map generated for an on-road maneuver, along with the expanded perception cost map and the resulting combined cost map used by the planner.

6.3. Incorporating Dynamic Obstacles

The combined cost map of the planner is also used to represent dynamic obstacles in the environment so that they can be avoided by the planner. The perception system of Boss represents static and dynamic obstacles independently, which allows the motion planner to treat each type of obstacle differently. The lattice planner adapts the dynamic obstacle avoidance behavior of the vehicle based on its current proximity to each dynamic obstacle. If the vehicle is close to a particular dynamic obstacle, that obstacle and a short-term prediction of its future trajectory is encoded into the combined cost map as a hard constraint so that it is strictly avoided. For every dynamic obstacle, both near and far, the planner encodes a varying high-cost region around the obstacle to provide a safe clearance. Although these high-cost regions are not hard constraints, they result in the vehicle avoiding the vicinity of the dynamic obstacles if at all possible. Further, the generality of this approach allows us to influence the behavior of our vehicle based on the specific behavior of the dynamic obstacles. For instance, we offset the high-cost region based on the relative position of the dynamic

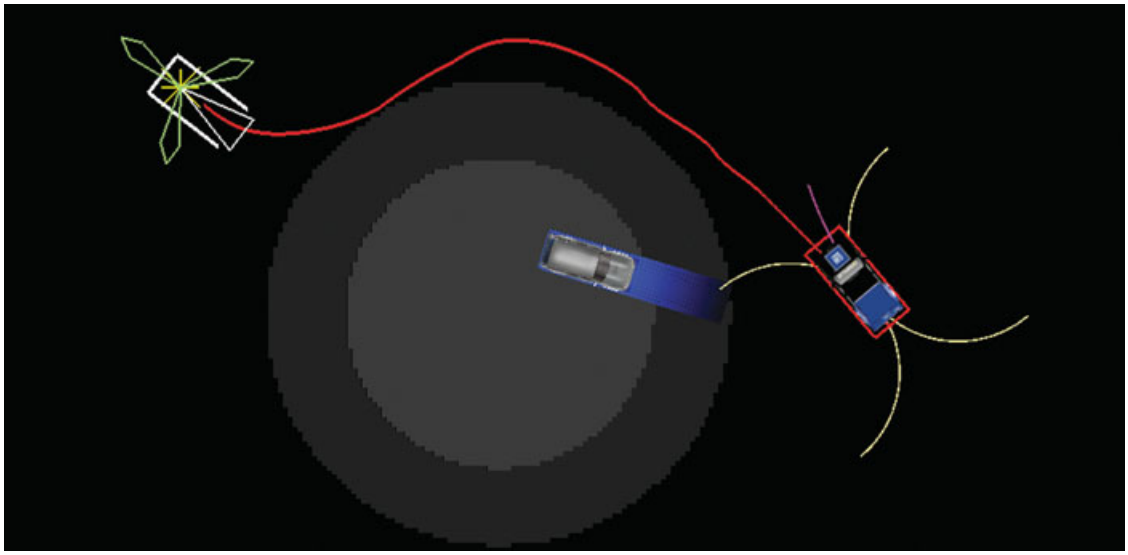

Figure 14. Biasing the cost map for the lattice planner so that the vehicle keeps away from dynamic obstacles. Notice that the high-cost region around the dynamic obstacle is offset to the left so that Boss will prefer moving to the right of the vehicle.

obstacle and our vehicle so that we will favor moving to the right, resulting in yielding behavior in unstructured environments quite similar to how humans react in these scenarios. Figure 14 provides an example scenario involving a dynamic obstacle along with the corresponding cost map generated.

7. TRACKING COMPLEX PATHS

The resulting lattice plan is then tracked in a manner similar 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 lattice 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.

As mentioned earlier, the motion planner generates trajectories at a fixed 10 Hz during operation. The lattice planner also nominally runs at 10 Hz. However, in very difficult planning scenarios the lattice planner may take longer (up to a couple of seconds) to generate its initial solution, and it is for this reason that preplanning is performed whenever possible (as will be discussed later). The motion planner continues to track the current lattice path until it is updated by the lattice planner.

Figure 15 provides an example of the local planner following a lattice plan to a specified parking spot. Figure 15(a) shows the lattice plan generated for the vehicle (in red) toward the desired parking spot (desired pose of the vehicle shown as the white triangle). Figure 15(b) shows the set of trajectories generated by the vehicle to track this plan, and Figure 15(c) shows the best trajectory selected by the vehicle to follow the path.

Both forward and reverse trajectories are generated as appropriate based on the velocity of the lattice path being tracked. When the path contains an upcoming velocity switching point, or cusp point, the local planner generates trajectories that bring the vehicle to a stop at the cusp point. Figure 16 shows reverse trajectories generated to a cusp point in the lattice path.

As mentioned above, one of the desired capabilities of our vehicle was to be able to exhibit

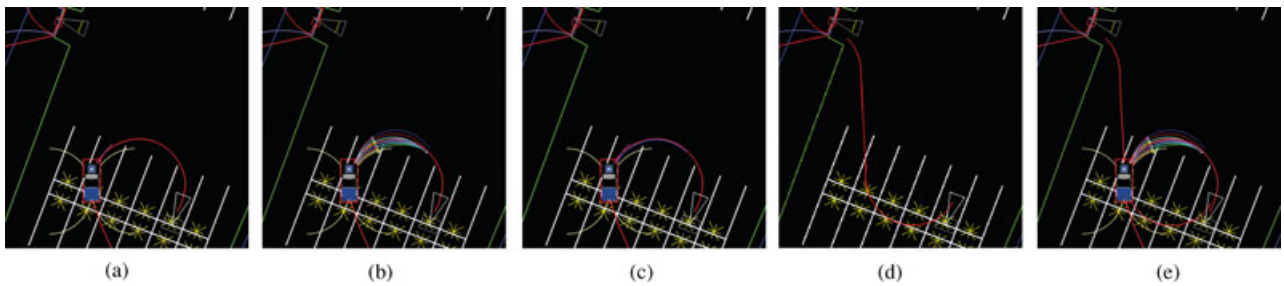

Figure 15. Following a lattice plan to a parking spot. Here, one lattice planner is updating the path to the spot while another is simultaneously preplanning a path out of the spot. The goals are represented by the white (current goal) and gray (next goal) triangles.

human-like yielding behavior in parking lots, to allow for safe, natural interaction with other vehicles. Through our biased cost function, the lattice planner typically generates paths through parking lots that keep to the right of other vehicles. However, it is possible that another vehicle may be quickly heading directly toward Boss, requiring evasive action similar to the on-road defensive driving maneuvers discussed in Section 5.7. In such a case, Boss's local planner detects that it is unable to continue along its current course without colliding with the other vehicle, and it then generates a set of trajectories that are offset to the right of the path. The intended behavior here is for each vehicle to move to the right to avoid a collision. Figure 17 provides an example of this behavior in a large parking lot.

In addition to the general optimizations always employed by the lattice planner, several context-specific reasoning steps are performed in different urban driving scenarios for which the lattice planner is invoked. In the following two sections we describe different methods used to provide optimized, intelligent behavior in parking lots and on-road error recovery scenarios.

7.1. Planning in Parking Lots

Because the location of parking lots is known a priori, this information can be exploited by the motion planning system to improve the efficiency and behavior of the lattice planner within these areas. First, we can use the extents of the parking lot to constrain the vehicle through the constrained cost map. To do this, we use the a priori specified extents of the parking lot to set all cells outside the lot (and not part of entry or exit lanes) in the constrained cost map to be hard constraints. This constrains the vehicle to operate only inside the lot. We also include a high-cost buffer around the perimeter of the parking lot to bias the vehicle away from the boundaries of the lot.

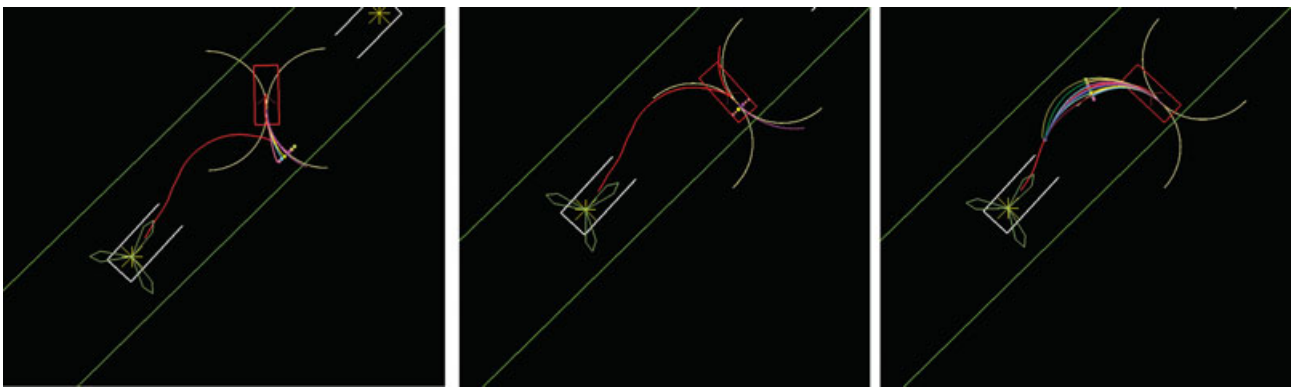

Figure 16. Reversing during path tracking. The goal is represented by the green star inside the white parking spot.

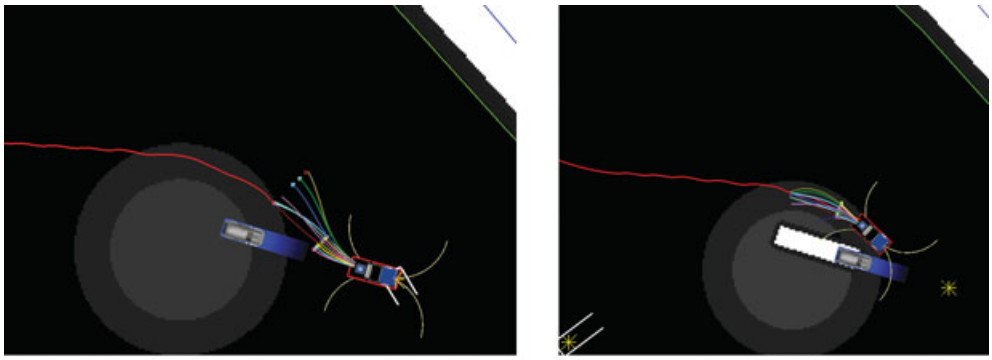

Figure 17. Defensive driving when in unstructured environments. (a) Trajectories thrown to right of path (other vehicle should likewise go to right). (b) New path planned from new position.

When prior terrain information such as overhead imagery exists, this information can also be incorporated into the constrained cost map to help provide global guidance for the vehicle. For instance, this information can be used to detect features such as curbs or trees in parking lots that should be avoided, so that these features can be used in planning before they are detected by onboard perception. Figures 18(a) and 18(b) shows overhead imagery of a parking lot area used to encode curb islands into a constrained cost map for the parking lot, and Figure 18(c) shows the corresponding constrained cost map. This constrained cost map is then stored offline and loaded by the planner online when it begins planning paths through the parking lot. By storing the constrained cost maps for parking lots offline, we significantly reduce online processing as generating the constrained cost maps for large, complex parking lots can take several seconds.

Further, because the parking lot goals are also known in advance of entering the parking lot (e.g., the vehicle knows which parking spot it is intending on reaching), the lattice planner can preplan to the first goal pose within the parking lot while the vehicle is still approaching the lot. By planning a path from the entry point of the parking lot in advance, the vehicle can seamlessly transition into the lot without needing to stop, even for very large and complex lots.

Figure 19 illustrates the preplanning used by the lattice planner. The left-most image shows our vehicle approaching a parking lot (boundary shown in green), with its intended parking spot indicated

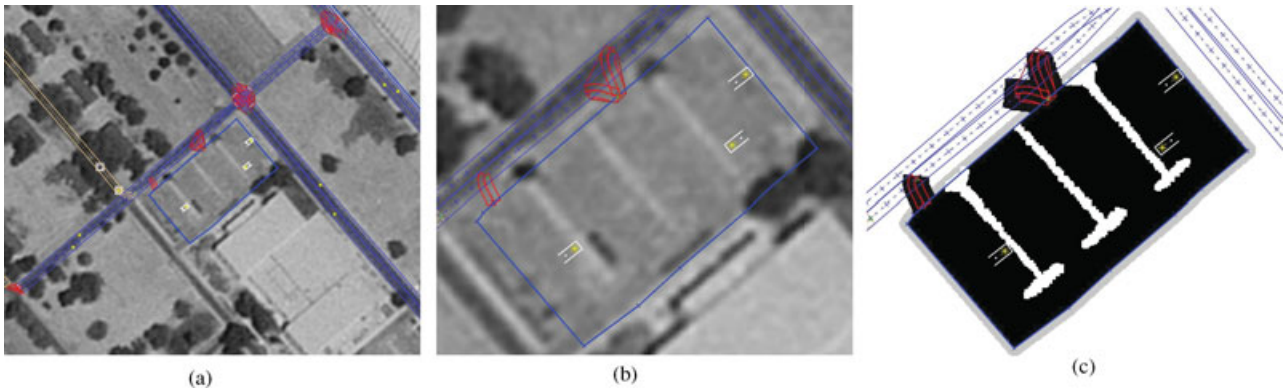

Figure 18. Generating constrained cost maps offline for Castle Commerce Center, California. (a) Overhead imagery showing testing area with road network overlaid. (b) Parking lot area (boundary in blue) with overhead imagery showing curb islands. (c) Resulting constrained cost map incorporating boundaries, entry and exit lanes, and curb islands (the lighter the color, the higher the cost).

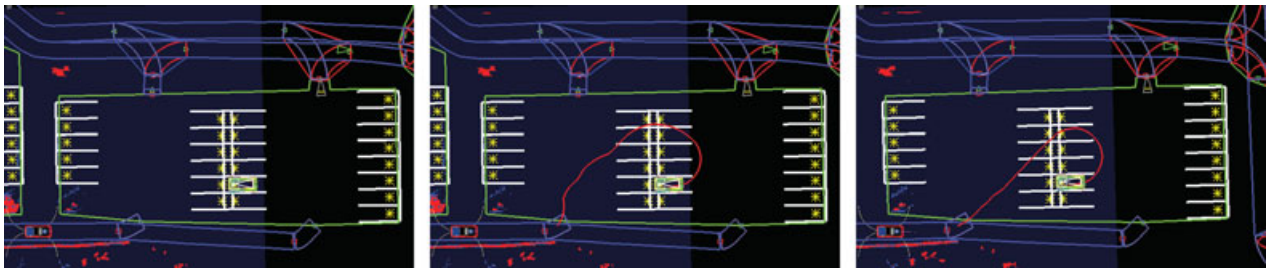

Figure 19. Preplanning a path into a parking spot and improving this path in an anytime fashion. A set of goal poses are generated that satisfy the parking spot, and an initial path is planned while the vehicle is still outside the parking lot. This path is improved as the vehicle approaches, converging to the optimal solution shown in the right-hand image.

by the white triangle (and multicolored set of goal poses). While the vehicle is still outside the lot, it begins planning a path from one of the entries to the desired spot, and the path converges to the optimal solution well before the vehicle enters the lot.

In a similar vein, when the vehicle is in a lot traveling toward a parking spot, we use a second lattice planner to simultaneously plan a path from that spot to the next desired location (e.g., the next parking spot to reach or an exit of the lot). When the vehicle reaches its intended parking spot, it then immediately follows the path from this second planner, again eliminating any time spent waiting for a plan to be generated.

Figure 15 provides an example of the use of multiple concurrent lattice planners. Figure 15(a) shows the lattice plan generated toward the desired parking spot. Figure 15(d) shows the path simultaneously being planned out of this spot to the exit of the parking lot, and Figure 15(e) shows the paths from both planners at the same time.

7.2. Planning in Error Recovery Scenarios

The lattice planner is flexible enough to be used in a large variety of cases that can occur during on-road and unstructured navigation. In particular, it is used during error recovery when navigating congested lanes or intersections and to perform difficult U-turns. In such cases, the nominal on-road motion planner determines that it is unable to generate any feasible trajectory and reports its failure to the behavioral executive, which in turn issues an unstructured 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

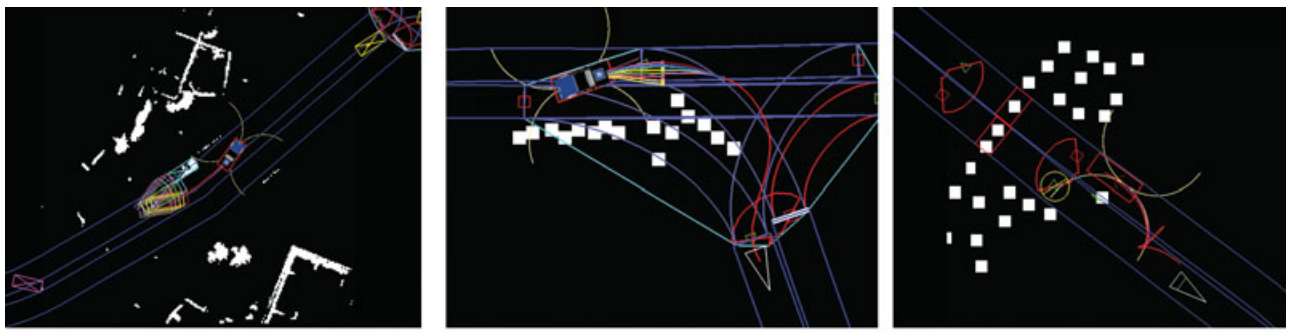

Figure 20. Error recovery examples. (a) Planning around a stalled vehicle in the road (obstacles in white, set of pose goals given to planner shown as various colored rectangles, and resulting path shown in red). Corresponds to scenario from Figure 13. (b) Planning through a partially blocked intersection. (c) Performing a complex U-turn in a cluttered area. (Boss's three-dimensional model has been removed to see the lattice plan underneath the vehicle.)

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) through increasing the cost of undesirable areas in the constrained cost map (see Figure 13).

The ability to cope with anomalous situations was a key focus of Boss's software system, and the lattice planner was used as a powerful tool to maneuver the vehicle to arbitrary locations in the environment. Figure 20 provides several error recovery examples involving the lattice planner.

The lattice planner is also invoked when the road cannot be detected with certainty due to the absence of markers (e.g., an unpaved road with berms). In this case, the lattice planner is used to bias the movement of the vehicle to stay within any detected geometric extents of the road.

8. RESULTS AND DISCUSSION

Our motion planning system was developed over the course of more than a year and tested over thousands of kilometers of autonomous operation in three different extensive testing sites, as well as the Urban Challenge event itself. Through this extensive testing, all components were hardened and the planners were incrementally improved upon.

A key factor in our system-level design was that Boss should never give up. As such, the lattice planner was designed to be general enough and powerful enough to plan in extremely difficult scenarios. In addition, the behavioral executive was designed to issue an infinite sequence of pose goals to the motion planner should it continue to fail to generate plans (see Baker et al., 2008, for details of this error recovery framework). And in the final Urban Challenge event, as anticipated, this error recovery played a large part in Boss's successful completion of the course. Over the course of the three final event missions, there were 17 different instances in which the lattice planner was invoked due to an encountered anomalous situation. (Some of these were getting cut off at intersections, coming across other vehicles blocking the lane, and perception occasionally observing heavy dust clouds as static obstacles.)

Incorporation of an accurate vehicle model was also an important design choice for Boss's motion planning system. This allowed Boss to push the limits of acceleration and speed and travel as fast as possible along the road network, confident in its execution. Combining this model with an efficient on-road planner that could safely handle the high speeds involved was also central to Boss's on-road performance.

In addition, the efficiency and path quality of the lattice planner enabled Boss to also travel smoothly and quickly through parking lot areas, without ever needing to pause to generate a plan. As well as generating smooth paths in (x, y, θ), by also considering the velocity dimension v, the lattice planner was able to explicitly reason about the time required to change direction of travel and was thus able to generate very fast paths even when complex maneuvers were required. Overall, the focus on execution speed and smoothness strongly contributed to Boss finishing the 4-h race 19 min and 8 s faster than its nearest competitor (DARPA, 2008).

One of the important lessons learned during the development of this system was that it is often extremely beneficial to exploit prior, offline processing to provide efficient online planning performance. We used this idea in several places, from the generation of lookup tables for the trajectory generator and lattice planner heuristic function to the precomputing of constrained cost maps for parking lots. This prior processing saved us considerably at run time. Further, even when faced with calculations that cannot be precomputed offline, such as planning paths through novel environments, it can often pay to begin planning before a plan is required. This concept was the basis for our preplanning on approach to parking lots and our concurrent planning to both current and future goals, and it enabled us to produce high-quality solutions without needing to wait for these solutions to be generated.

Finally, although simplicity was central to our high-level system development and significant effort was put into making the interfacing between processes as lightweight as possible, we found that in regard to motion planning, although simple, approximate planning algorithms can work well in most cases; generality and completeness when needed are priceless. Using a high-fidelity, high-dimensional lattice planner for unstructured planning problems proved time and time again to be the right choice for our system.

9. PRIOR WORK

Existing research on motion planning for autonomous outdoor vehicles can be roughly broken into two classes: motion planning for autonomous vehicles following roads and motion

planning for autonomous vehicles navigating unstructured environments including off-road areas and parking lots. A key difference between road following and navigating unstructured environments is that in the former case a global plan is already encoded by the road (lane) itself, and therefore the planner needs to generate only short-term motion trajectories that follow the road, whereas in the latter case no such global plan is provided.

9.1. On-Road Planning

A vast amount of research has been conducted in the area of road following. Some of the best known and fully operational systems include the CMU NavLab project (Thorpe, Hebert, Kanade, & Shafer, 1988), the INRIA autonomous car project (Baber, Kolodko, Noel, Parent, & Vlacic, 2005) in France, the VaMoRs (Dickmanns et al., 1993) and VITA projects (Ulmer, 1992) in Germany, and the Personal Vehicle System (PVS) project (Hattori, Hosaka, Taniguchi, & Nakano, 1992) in Japan. Approaches to road following in these and other systems vary drastically. For example, one of the road following algorithms used by NavLab vehicles is ALVINN (Pomerleau, 1991), which learns the mapping from road images onto control commands using a neural network by observing how the car is driven manually for several minutes. In the VITA project, on the other hand, the planner was used to track the lane while regulating the velocity of the vehicle in response to the curvature of the road and the distance to nearby vehicles and obstacles. Stanford University's entry in the second DARPA Grand Challenge also exhibited lane following behavior through evaluating a set of candidate trajectories that tracked the desired path (Thrun et al., 2006). Our lane planning approach is closely related to theirs; however, to generate their candidate trajectories they sample the control space around a base trajectory (e.g., the trajectory leading down the center of the lane), whereas we sample the state-space along the road lane. Some significant advantages of using a state-space approach include the ability to finely control position and heading at the terminal state of each trajectory (which we can align with the road shape), the ability to impose the requirement that each trajectory terminates at exactly the same distance along the path, allowing for fairer evaluation of candidate actions, and the simplification of generating complex maneuvers such as U-turns and lane changes.

However, several of these existing approaches have been shown to be very effective in road following in normal conditions. A major strength of our approach, on the other hand, is that it can handle difficult scenarios, such as when a road is partially blocked (e.g., by an obstacle, a stalled car, a slow-moving car, or a car driving in an opposite direction but moving out of the bounds of its own lane). Our system can handle these scenarios robustly and at high speeds.

9.2. Unstructured Planning

Roboticists have concentrated on the problem of mobile robot navigation in unstructured environments for several decades. Early approaches concentrated on performing local planning, in which very short-term reasoning is performed to generate the next action for the vehicle (Fox, Burgard, & Thrun, 1997; Khatib, 1986; Simmons, 1996). A major limitation of these purely local approaches was their capacity to get the vehicle stuck in local minima en route to the goal (for instance, cul-de-sacs). To improve upon this limitation, algorithms were developed that incorporated global as well as local information (Brock & Khatib, 1999; Kelly, 1995; Philippsen & Siegwart, 2003; Thrun et al., 1998). Subsequent approaches have focused on improving the local planning component of these approaches by using more sophisticated local action sets that better follow the global value function (Howard & Kelly, 2007; Thrun et al., 2006) and by generating sequences of actions to perform more complex local maneuvers (Braid, Broggi, & Schmiedel, 2006; Stachniss & Burgard, 2002; Urmson et al., 2006). In parallel, researchers have concentrated on improving the quality of global planning, so that a global path can be easily tracked by the vehicle (Knepper & Kelly, 2006; LaValle & Kuffner, 2001; Likhachev, Gordon, & Thrun, 2003; Likhachev et al., 2005; Pivtoraiko & Kelly, 2005; Song & Amato, 2001). However, the computational expense of generating complex global plans over large distances has remained very challenging, and the approaches to date have been restricted to small distances, fairly simple environments, or highly suboptimal solutions. Our lattice-based global planner is able to efficiently generate feasible global paths over much larger distances than previously possible, while providing suboptimality bounds on the quality of the solutions and anytime improvement of the solutions generated.

10. CONCLUSIONS

We have presented the motion planning framework for an autonomous vehicle navigating through urban environments. Our approach combines a high-fidelity trajectory generation algorithm for computing dynamically feasible actions with an efficient lane-based planner, for on-road planning, and a 4D lattice planner, for unstructured planning. It has been implemented on an autonomous vehicle that has traveled more than 3,000 autonomous kilometers, and we have presented sample illustrations and results from the Urban Challenge, which it won in November 2007.

11. APPENDIX: VEHICLE MODEL

Boss's vehicle model predicts the resulting vehicle state $\mathbf{x}_{t+\Delta t}$ after applying a parameterized set of controls $\mathbf{u}(\mathbf{p},\mathbf{x})$ to an initial vehicle state $\mathbf{x}_t$. It does this by forward-simulating the movement of the vehicle given the commanded controls. However, it also constrains these controls based on the physical constraints of the vehicle and safety bounds. Algorithms 1–3 provide pseudocode of this process (the main vehicle model function is *MotionModel* in Algorithm 3). Values for the parameters used in each function are defined in Table A1.

Table A1. Parameters used in vehicle model.

Description	Parameter	Race day values
Maximum curvature	$[\kappa]_{\max}$	0.1900 rad
Minimum curvature	$[\kappa]_{\min}$	−0.1900 rad
Maximum rate of curvature	$[\frac{d\kappa}{dt}]_{\max}$	0.1021 rad/s
Minimum rate of curvature	$[\frac{d\kappa}{dt}]_{\min}$	−0.1021 rad/s
Maximum acceleration	$[\frac{dv}{dt}]_{\max}$	2.000 m/s
Maximum deceleration	$[\frac{dv}{dt}]_{\min}$	−6.000 m/s
Control latency	t_{delay}	0.0800 s
Speed control logic "a" coefficient	a_{scl}	0.1681
Speed control logic "b" coefficient	b_{scl}	−0.0049
Speed control logic threshold	$\lvert v\rvert_{\text{scl}}$	4.000 m/s
Max curvature for speed	$[\kappa v]_{\max}$	0.1485 rad
Speed control logic safety factor	$\text{safety}_{\text{factor}}$	1.000

Algorithm 1 SpeedControlLogic ($\mathbf{x}_{t+\Delta t}$)

Input: $\mathbf{x}_{t+\Delta t}$

Output: $\mathbf{x}_{t+\Delta t}$

$|v|_{cmd} \leftarrow |[v_{t+\Delta t}]_{cmd}|$; //calculate speed

$[|v|_{cmd}]_{max} \leftarrow max\left[|v|_{scl}, \left[\frac{\kappa_{t+\Delta t}-a}{b}\right]\right]$; //compute safe speed

$[\kappa]_{max,scl} \leftarrow min\left[[\kappa]_{max}, a+b|v|_{cmd}\right]$; //compute safe curvature

if $\left[|\kappa_{t+\Delta t}| \geq [\kappa]_{max,scl}\right]$ **then**

  $|v|_{cmd} \leftarrow |safety_{factor} \cdot [|v|_{cmd}]_{max}|$; //check for safe speed

$[v_{t+\Delta t}]_{cmd} \leftarrow |v|_{cmd} \frac{[v_t]_{cmd}}{|[v_t]_{cmd}|}$; //update velocity command

return $\mathbf{x}_{t+\Delta t}$;

Algorithm 2 ResponseToControlInputs ($\mathbf{x}_t$,$\mathbf{x}_{t+\Delta t}$, Δt)

Input: $\mathbf{x}_t$,$\mathbf{x}_{t+\Delta t}$, Δt

Output: $\mathbf{x}_{t+\Delta t}$

$\left[\frac{d\kappa}{dt}\right]_{cmd} \leftarrow \frac{[\kappa_{t+\Delta t}]_{cmd}-\kappa_t}{dt}$; //compute curvature rate command

$\left[\frac{d\kappa}{dt}\right]_{cmd} \leftarrow min\left[\frac{d\kappa}{dt}, \left[\frac{d\kappa}{dt}\right]_{max}\right]$; //upper bound curvature rate

$\left[\frac{d\kappa}{dt}\right]_{cmd} \leftarrow max\left[\frac{d\kappa}{dt}, \left[\frac{d\kappa}{dt}\right]_{min}\right]$; //lower bound curvature rate

$\mathbf{x}_{t+\Delta t} \leftarrow$ SpeedControlLogic $[\mathbf{x}_{t+\Delta t}]$; //speed control logic

$\kappa_{t+\Delta t} \leftarrow \kappa_t + \left[\frac{d\kappa}{dt}\right]_{cmd} \Delta t$; //compute curvature at time $t+\Delta t$

$\kappa_{t+\Delta t} \leftarrow min[\kappa_{t+\Delta t}, \kappa_{max}]$; //upper bound curvature

$\kappa_{t+\Delta t} \leftarrow max[\kappa_{t+\Delta t}, \kappa_{min}]$; //lower bound curvature

$\left[\frac{dv}{dt}\right]_{cmd} \leftarrow \frac{[v_{t+\Delta t}]_{cmd}-vt}{dt}$; //compute acceleration command

$\left[\frac{dv}{dt}\right]_{cmd} \leftarrow min\left[\left[\frac{dv}{dt}\right]_{cmd}, \left[\frac{dv}{dt}\right]_{max}\right]$; //upper bound acceleration

$\left[\frac{dv}{dt}\right]_{cmd} \leftarrow max\left[\left[\frac{dv}{dt}\right]_{cmd}, \left[\frac{dv}{dt}\right]_{min}\right]$; //lower bound acceleration

$v_{t+\Delta t} \leftarrow v_t + \left[\frac{dv}{dt}\right]_{cmd} \Delta t$; //compute velocity at time $t+\Delta t$

return $\mathbf{x}_{t+\Delta t}$;

Algorithm 3 MotionModel ($\mathbf{x}_t$, $\mathbf{u}$ ($\mathbf{p}$,$\mathbf{x}$), Δt)

Input: $\mathbf{x}_t$, $\mathbf{u}(\mathbf{p},\mathbf{x})$, Δt

Output: $\mathbf{x}_{t+\Delta t}$

$x_{t+\Delta t} \leftarrow x_t + v_t \cos[\theta_t]\Delta t$; //compute change in 2D x-position

$y_{t+\Delta t} \leftarrow y_y + v_t \sin[\theta_t]\Delta t$; //compute change in 2D y-position

$\theta_{t+\Delta t} \leftarrow \theta_t + v_t \kappa_t \Delta t$; //compute change in 2D orientation

$[\kappa_{t+\Delta t}]_{cmd} \leftarrow \mathbf{u}[\mathbf{p}, s]$; //get curvature command

```
5  [v_{t+Δt}]_cmd ← u[p, t − t_delay];          //get velocity command
6  [a_{t+Δt}]_cmd ← u[p, t − t_delay];          //get acceleration command
7  x_{t+Δt} ← ResponseToControlInputs(x_t, x_{t+Δt}, Δt);   //estimate response
8  return x_{t+Δt};
```

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

Baber, J., Kolodko, J., Noel, T., Parent, M., & Vlacic, L. (2005). Cooperative autonomous driving: Intelligent vehicles sharing city roads. IEEE Robotics and Automation Magazine, 12(1), 44–49.

Baker, C., Ferguson, D., & Dolan, J. (2008). Robust mission execution for autonomous urban driving. In Proceedings of the International Conference on Intelligent Autonomous Systems (IAS).

Braid, D., Broggi, A., & Schmiedel, G. (2006). The TerraMax autonomous vehicle. Journal of Field Robotics, 23(9), 693–708.

Brock, O., & Khatib, O. (1999). High-speed navigation using the global dynamic window approach. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

Carsten, J., Rankin, A., Ferguson, D., & Stentz, A. (2007). Global path planning on-board the Mars Exploration Rovers. In Proceedings of the IEEE Aerospace Conference.

DARPA (2008). DARPA Urban Challenge official results. Posted at http://www.darpa.mil/GRANDCHALLENGE/mediafaq.asp.

Dickmanns, E., Behringer, R., Brudigam, C., Dickmanns, D., Thomanek, F., & Holt, V. (1993). All-transputer visual autobahn-autopilot/copilot. In Proceedings of the 4th International Conference on Computer Vision ICCV (pp. 608–615).

Ferguson, D., Darms, M., Urmson, C., & Kolski, S. (2008). Detection, prediction, and avoidance of dynamic obstacles in urban environments. In Proceedings of the IEEE Intelligent Vehicles Symposium (IV).

Ferguson, D., & Likhachev, M. (2008). Efficiently using cost maps for planning complex maneuvers. In Proceedings of the Workshop on Planning with Cost Maps, IEEE International Conference on Robotics and Automation.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics and Automation, 4(1).

Hattori, A., Hosaka, A., Taniguchi, M., & Nakano, E. (1992). Driving control system for an autonomous vehicle using multiple observed point information. In Proceedings of Intelligent Vehicle Symposium.

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

Iagnemma, K., & Buehler, M. (Eds.). (2006a). Special Issue on the DARPA Grand Challenge, Part 1. Journal of Field Robotics, 23(8).

Iagnemma, K., & Buehler, M. (Eds.). (2006b). Special Issue on the DARPA Grand Challenge, Part 2. Journal of Field Robotics, 23(9).

Kelly, A. (1995). An intelligent predictive control approach to the high speed cross country autonomous navigation problem. PhD thesis, Carnegie Mellon University, Pittsburgh, PA.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, 5(1), 90–98.

Knepper, R., & Kelly, A. (2006). High performance state lattice planning using heuristic look-up tables. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS).

LaValle, S., & Kuffner, J. (2001). Rapidly-exploring random trees: Progress and prospects. Algorithmic and computational robotics: New directions (pp. 293–308).

Likhachev, M., & Ferguson, D. (2008). Planning dynamically feasible long range maneuvers for autonomous vehicles. In Proceedings of Robotics: Science and Systems (RSS).

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2005). Anytime dynamic A*: An anytime, replanning algorithm. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS).

Likhachev, M., Gordon, G., & Thrun, S. (2003). ARA*: Anytime A* with provable bounds on sub-optimality. In Advances in neural information processing systems. Cambridge, MA: MIT Press.

Pearl, J. (1984). Heuristics: Intelligent search strategies for computer problem solving. Addison-Wesley.

Philippsen, R., & Siegwart, R. (2003). Smooth and efficient obstacle avoidance for a tour guide robot. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

Pivtoraiko, M., & Kelly, A. (2005). Constrained motion planning in discrete state spaces. In Proceedings of the International Conference on Advanced Robotics (FSR).

Pomerleau, D. (1991). Efficient training of artificial neural networks for autonomous navigation. Neural Computation, 3(1), 88–97.

Simmons, R. (1996). The curvature velocity method for local obstacle avoidance. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., & Schwehr, K. (2000). Recent progress in

local and global traversability for planetary rovers. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

Song, G., & Amato, N. (2001). Randomized motion planning for car-like robots with C-PRM. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS).

Stachniss, C., & Burgard, W. (2002). An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS).

Stentz, A., & Hebert, M. (1995). A complete navigation system for goal acquisition in unknown environments. Autonomous Robots, 2(2), 127–145.

Thorpe, C., Hebert, M., Kanade, T., & Shafer, S. (1988). Vision and navigation for the Carnegie-Mellon Navlab. IEEE Transactions on Pattern Analysis and Machine Intelligence, 10(3), 362–373.

Thorpe, C., Jochem, T., & Pomerleau, D. (1997). The 1997 automated highway demonstration. In Proceedings of the International Symposium on Robotics Research (ISRR).

Thrun, S., et al. (1998). Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. Bonasso, & R. Murphy (Eds.), AI-based mobile robots: Case studies of successful robot systems. Cambridge, MA: MIT Press.

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., 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.

Ulmer, B. (1992). VITA—An autonomous road vehicle (ARV) for collision avoidance in traffic. In Proceedings of Intelligent Vehicle Symposium (pp. 36–41).

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

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M.N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.-W., Singh, S., Snider, J., Stentz, A., Whittaker, W., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., & Ferguson, D. (2008). Autonomous driving in urban environments: Boss and the Urban Challenge. Journal of Field Robotics, 25(8), 425–466.