Motion Planning in Urban Environments: Part I

Dave Ferguson
Intel Research Pittsburgh
Pittsburgh, PA
dave.ferguson@intel.com

Thomas M. Howard
Carnegie Mellon University
Pittsburgh, PA
thoward@ri.cmu.edu

Maxim Likhachev
University of Pennsylvania
Philadelphia, PA
maximl@seas.upenn.edu

***Abstract*— We present the motion planning framework for an autonomous vehicle navigating through urban environments. Such environments present a number of motion planning challenges, including ultra-reliability, high-speed operation, complex inter-vehicle 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 this Part I of a two-part paper, we describe the underlying trajectory generator and the on-road planning component of this system. We provide examples and results from "Boss", an autonomous SUV that has driven itself over** 3000 **kilometers and competed in, and won, the Urban Challenge.**

I. 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 non-existent [1], [2], [3], [4], [5], [6], or very simple on-road maneuvers such as highway-based lane following [7].

The Urban Challenge competition was designed to extend this technology as far as possible towards the goal of unrestricted on-road driving. The event consisted of an autonomous vehicle race through an urban environment containing single and multi-lane roads, traffic circles and intersections, open areas and unpaved sections, road blockages, and complex parking tasks. Successful vehicles had to travel roughly 90 kilometers, 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 into 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.

In Part I of the paper we 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 V we discuss the motion planning framework used when navigating on roads.

In Part II of the paper 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 kilometers of testing, and describe related work in both on-road and unstructured planning.

II. System Architecture

Boss' software system is decomposed into four major blocks (see Figure 1). The **Perception** component fuses and processes data from Boss' sensors to provide key environmental information, including:

- **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 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 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

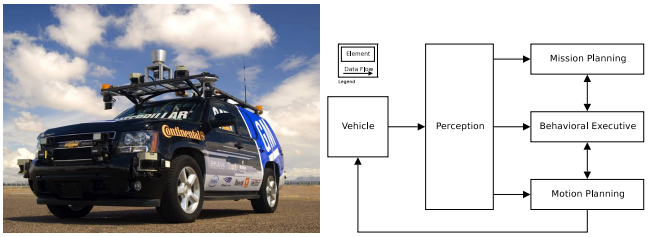


Fig. 1. "Boss": Tartan Racing's winning entry in the Urban Challenge, along with its software system architecture.

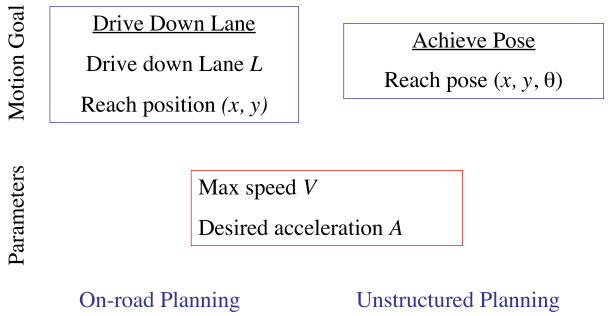


Fig. 2. Motion goals provided by the Behavioral Executive to the Motion Planner. Also shown are the frequently updated speed and desired acceleration commands.

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 queueing 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 towards this goal, as described in the following section. Two broad contexts for motion planning exist: on-road driving and unstructured driving.

III. 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.

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 lane-based 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 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 through in the desired lane).

As well as issuing motion goals, the Behavioral Executive is constantly providing 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 towards the desired goal 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. As mentioned above, the nature of the path generated 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.

IV. Trajectory Generation

Each candidate trajectory is computed using a model-predictive trajectory generator from [8] 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)) 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.

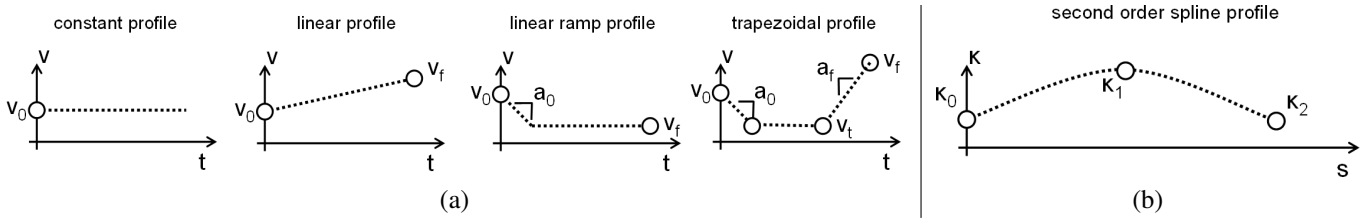


Fig. 3. Velocity and curvature profiles. (a) Several different linear velocity profiles were applied in this system, each with their own parameterization and application. Each parameterization contains some subset of velocity and acceleration knot points (v_0, v_t, v_f, a_0, and a_f) and the length of 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).

A. State Constraints

For navigating both on-road and unstructured areas of urban environments we generated trajectories that satisfied both target two-dimensional 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 Equation 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 $\mathbf{x_C}$.

B. Vehicle Modeling

The development of a high fidelity vehicle dynamics 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 fit 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, 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.

C. Controls Parameterization

For Ackermann steered vehicles, it is advantageous to define 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 a 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 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 $\mathbf{p}$. Since 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 enough degrees of freedoms (4) to satisfy the boundary state constraints (3). 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: the latter two curvature spline knot points and the trajectory length:

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

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

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

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

D. 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 pre-compute 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 non-integrable model dynamics, it is infeasible to pre-compute the entire mapping of state space to input space for any nontrivial system, such as the Boss vehicle model. We instead generate an approximation of this mapping through a five-dimensional 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, 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. We first discretize our five dimensions of interest into a 5D table. Next, we uniformly sample from the set of all possible parameter values and record which table position each of these sample trajectories terminates in. We then step through the 5D table and for each position in the table we find the sample parameter values that came closest to this position. We take that parameter set and optimize it (using the optimization technique presented in the next section) to accurately match the table position, and then store the resulting parameter set in the corresponding index of the 5D table.

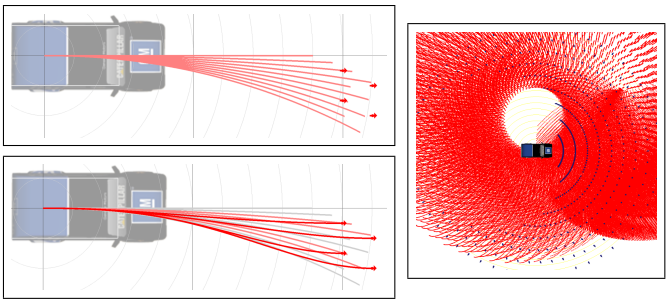


Fig. 4. Offline lookup table generation. *Top-left*: Some sampled trajectories (in red) and some table endpoints (red arrows) that we wish to generate trajectories to for storage in the lookup table. *Bottom-left*: The closest sampled trajectories to the desired table endpoints are selected and then optimized to reach the desired endpoints. The parameter sets corresponding to these optimized trajectories are then stored in the lookup table. *Right*: The set of all sampled trajectories (red) and table endpoints (blue) for a single initial vehicle state.

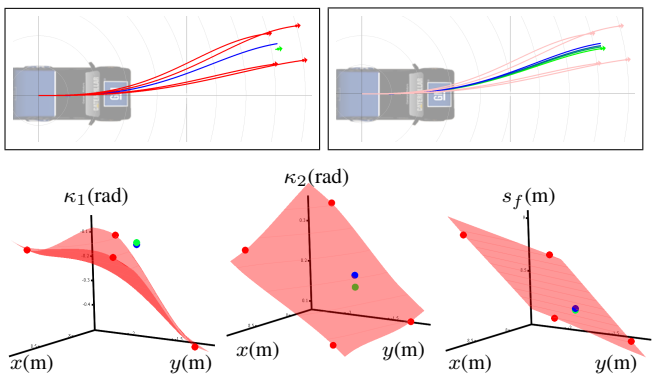


Fig. 5. Online Trajectory Generation. *Top-left*: Given an initial state and desired terminal state (relative terminal state shown in green), we find the closest elements of the lookup table (in red) and interpolate between the control parameters associated with these elements (interpolation of free parameters shown at bottom row) to come up with an initial approximation of the free parameters (resulting corresponding trajectory shown in blue). *Top-right*: 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 on the bottom row (interpolated solutions shown in blue, final optimized solutions shown in green).

E. 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. 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 is model-invariant since it is 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 where 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 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 lookup 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(*top-left, bottom*) shows the lookup and

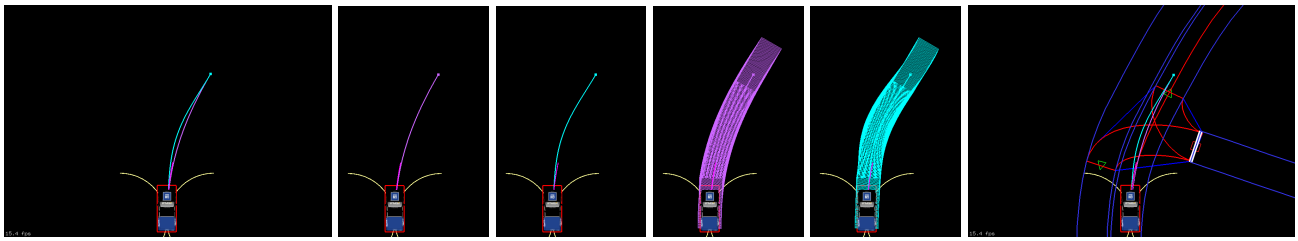
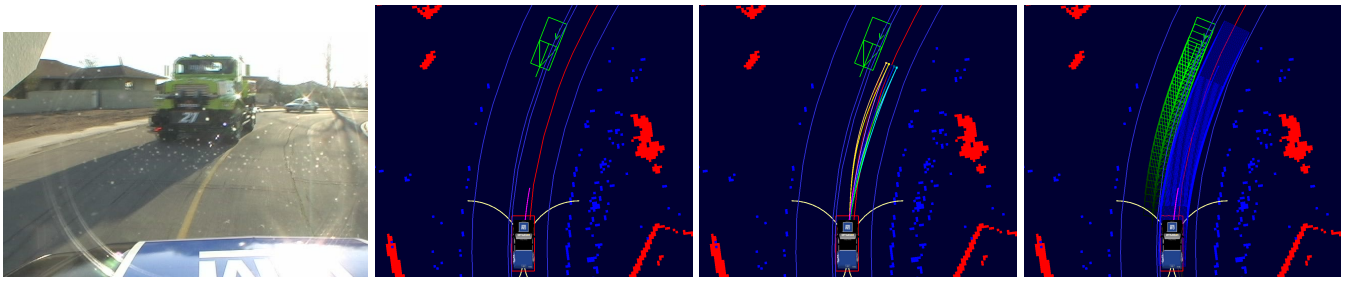


Fig. 6. Smooth and sharp trajectories



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

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

interpolation steps, with the resulting parameter set values and corresponding trajectory. Next, we evaluate the endpoint error of the 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(*top-right*)), and the resulting parameters and trajectory are stored and evaluated by the motion planner.

V. On-road Planning

A. 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 towards 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.

B. 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 will be executed. To calculate this state, forwards-prediction using an accurate vehicle model (the same model used in the trajectory generation phase) is performed using the trajectories selected for execution in previous planning episodes. This forwards-prediction accounts for both the high-level delays (the time required to plan) and the low-level delays (the time required to execute a command).

The goals are placed at a fixed longitudinal distance down the centerline path (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 forwards-predicted vehicle state. The sharp trajectory has this parameter set to an offset value from the forwards-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 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.

C. Trajectory Velocity Profiles

The velocity profile used for each of the generated trajectories is selected from the set introduced in Section IV-C based on several factors, including: the maximum speed given from the Behavioral Executive based on safe following

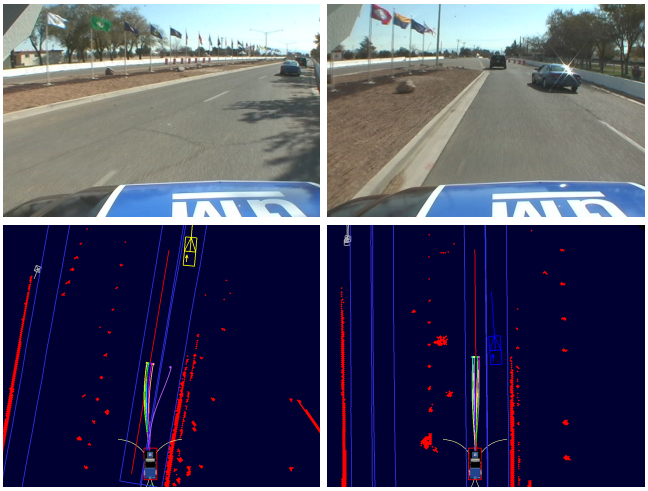


Fig. 8. Performing a lane change reliably and safely. Here, Boss changed lanes because another robot's chase vehicle was traveling too slowly in its original lane.

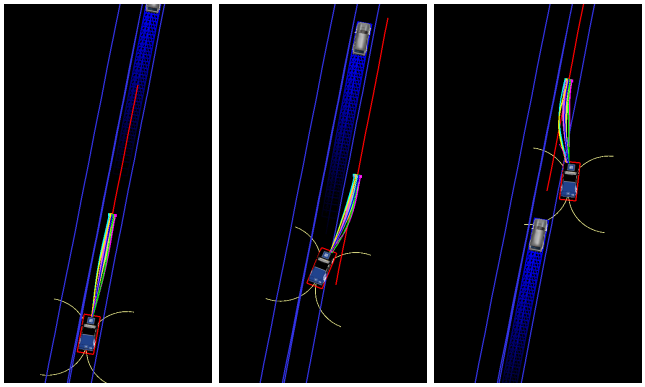


Fig. 9. Defensive Driving on roads. *Left*: Boss initially plans down its lane while the oncoming vehicle is far away. *Center*: When the oncoming vehicle is detected as dangerous, Boss generates a set of trajectories off the right side of the road. *Right*: After the oncoming vehicle has passed, Boss plans back onto the road and continues.

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 stopline).

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, 3 per initial trajectory).

D. 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. See [9] 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) 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.

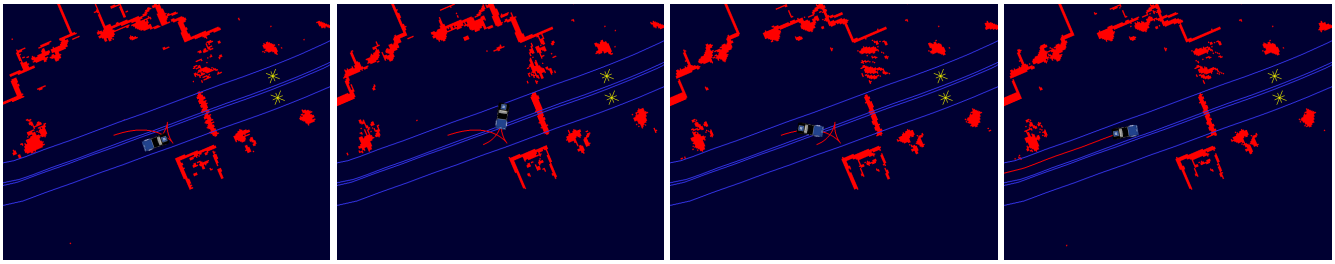
E. Lane Changing

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 similar way to normal lane driving: a set of trajectories are 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 are 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.

F. 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



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

Fig. 10. Performing a U-turn when encountering a road blockage. (a) Initial U-turn plan generated to reverse direction. (b, c) Tracking the resulting U-turn plan. (d) Reverting to lane planning when in desired lane.

path may consist of a single forwards segment or a three-point turn. This path is then tracked in a similar fashion to the lane centerline paths, using a series of trajectories with varying offsets. Figure 10 provides an example three-point turn performed during one of the qualification events at the Urban Challenge.

G. 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' 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 towards 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 similar fashion 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 9).

H. 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 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 Part II of this paper, and more details on the error detection and recovery process can be found in [10].

VI. Conclusions

We have presented the on-road motion planning framework for an autonomous vehicle navigating through urban environments. Our approach combines a model-predictive trajectory generation algorithm for computing dynamically-feasible actions with an efficient lane-based planner. It has been implemented on an autonomous vehicle that has traveled over 3000 autonomous kilometers and we have presented sample illustrations and results from the Urban Challenge, which it won in November 2007. Part II of this paper describes the unstructured component of this framework and discusses in more detail the existing body of related work.

VII. Acknowledgements

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

References

[1] A. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," *Autonomous Robots*, vol. 2, no. 2, pp. 127–145, 1995.

[2] A. Kelly, "An intelligent predictive control approach to the high speed cross country autonomous navigation problem," Ph.D. dissertation, Carnegie Mellon University, 1995.

[3] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr, "Recent progress in local and global traversability for planetary rovers," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[4] "Special Issue on the DARPA Grand Challenge, Part 1," *Journal of Field Robotics*, vol. 23, no. 8, 2006.

[5] "Special Issue on the DARPA Grand Challenge, Part 2," *Journal of Field Robotics*, vol. 23, no. 9, 2006.

[6] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz, "Global path planning on-board the Mars Exploration Rovers," in *Proceedings of the IEEE Aerospace Conference*, 2007.

[7] C. Thorpe, T. Jochem, and D. Pomerleau, "The 1997 automated highway demonstration," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 1997.

[8] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[9] D. Ferguson, M. Darms, C. Urmson, and S. Kolski, "Detection, Prediction, and Avoidance of Dynamic Obstacles in Urban Environments," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2008.

[10] C. Baker, D. Ferguson, and J. Dolan, "Robust Mission Execution for Autonomous Urban Driving," 2008, submitted to International Conference on Intelligent Autonomous Systems (IAS).