2008 IEEE/RSJ International Conference on Intelligent Robots and Systems
Acropolis Convention Center
Nice, France, Sept, 22-26, 2008

Motion Planning for Urban Driving using RRT

Yoshiaki Kuwata, Gaston A. Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P. How

Abstract— **This paper provides a detailed analysis of the motion planning subsystem for the MIT DARPA Urban Challenge vehicle. The approach is based on the Rapidly-exploring Random Trees (RRT) algorithm. The purpose of this paper is to present the numerous extensions made to the standard RRT algorithm that enable the on-line use of RRT on robotic vehicles with complex, unstable dynamics and significant drift, while preserving safety in the face of uncertainty and limited sensing. The paper includes numerous simulation and race results that clearly demonstrate the effectiveness of the planning system.**

I. Introduction

The DARPA Urban Challenge (DUC) was the third installment of a series of ground-breaking races for autonomous ground robots. The major new feature of the DUC compared to previous races was the introduction of traffic, with up to 70 robotic and human-driven vehicles on the course, resulting in hundreds of unscripted robot-on-robot (and robot on human driven vehicle) interactions. In order to ensure safe operations, all vehicles were required to abide by the traffic laws and rules of the road. For example, vehicles were expected to stay in the correct lane, maintain a safe speed, yield to other vehicles at intersections, pass other vehicles when safe to do so, recognize blockages and execute U-turns when needed, and park in an assigned space.

Developing a robotic vehicle that could complete the DUC was a major systems engineering effort, requiring the development and integration of state-of-the-art technologies in planning, control, and sensing [1], [2]. This paper provides additional details on the motion planning subsystem of MIT's vehicle, called *Talos*. This subsystem computes a path to reach a goal point specified by a higher-level subsystem (the Navigator), while avoiding collisions with other vehicles, static obstacles, and abiding by the rules of the road. The output of the motion planning subsystem is in turn fed to a lower-level system (the Controller), interfacing directly to the vehicle, and responsible for the execution of the motion plan.

There are many approaches to the motion planning problem available in the literature; and we will not discuss their relative merits but refer the reader to [3]–[7]. The primary challenges in designing the motion planning subsystem for DUC resulted from: (i) complex and unstable vehicle dynamics, with substantial drift, (ii) limited sensing capabilities in an uncertain, time-varying environment, (iii) temporal and logical constraints, arising from the rules of the road. Our planning system is based on the Rapidly-exploring Random Trees (RRT) algorithm [8], an incremental sampling-based method [6, Section 14.4]. The main reasons for this choice were: (i) Sampling-based algorithms are applicable to very general dynamical models; (ii) The incremental nature of the algorithms lends itself easily to real-time, on-line implementation, while retaining certain completeness guarantees; (iii) Sampling-based methods do not require the explicit enumeration of constraints, but allow trajectory-wise checking of possibly very complex constraints.

However, the application of incremental sampling-based motion planning methods to robotic vehicles with complex and unstable dynamics, such as the Landrover LR3 used for the race, is far from straightforward. For example, the unstable nature of the vehicle dynamics requires the addition of a path-tracking control loop whose performance is generally hard to characterize. Moreover, the momentum of the vehicle at speed must be taken into account, making it impossible to ensure collision avoidance by point-wise constraint checks. In fact, to the best of our knowledge, RRTs have never been used in on-line planning systems for robotic vehicles with the above characteristics, but have been restricted either to simulation, or to kinematic, essentially driftless, robots (i.e., can stop instantaneously by setting the control input to zero).

The rest of the paper is organized as follows. Following the overview of the algorithm in Section II, Section III presents several extensions made to RRT. Section IV discusses the effectiveness of our motion planning system, based on the analysis of actual data collected during the DUC race.

II. Overview of the Approach

A. Problem Statement

Given a near-term target, and a low-level controller that can track a path and a speed command, the problem statement for the motion planner is to provide a path and a speed command to the controller, such that the vehicle avoid obstacles and stay in lane boundaries. In order to account for the dynamic nature of the urban driving, the planner must be able to quickly react to the change in the perceived environment. Furthermore, the perceived information has inherent noise, and the RRT must be robust to the uncertainties.

B. Rapidly-exploring Random Tree (RRT)

RRT algorithms grow a tree of dynamically feasible trajectories by sampling numerous points randomly. In contrast to the standard RRT, which samples the input to the vehicle, our RRT algorithm samples the input to the controller [9].

Y. Kuwata was a Postdoctoral Associate in the Dept. of Aeronautics and Astronautics, Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, USA kuwata@alum.mit.edu

G. Fiore and J. Teo are with the Dept. of Aeronautics and Astronautics, MIT, gafiore@mit.edu, csteo@mit.edu

E. Frazzoli and J. How are with the Faculty of the Dept. of Aeronautics and Astronautics, MIT, frazzoli@mit.edu, jhow@mit.edu

978-1-4244-2058-2/08/$25.00 ©2008 IEEE.

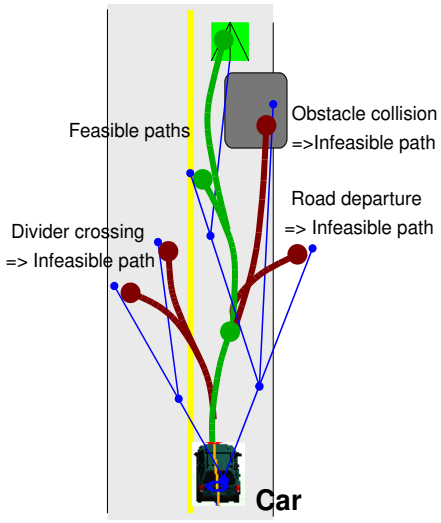


Fig. 1. Motion plans are propagated using the vehicle's dynamical model. Propagated paths are then evaluated for feasibility.

The dynamically feasible trajectory is obtained by running forward a simulation of the closed-loop system, consisting of the vehicle model and the controller. By using a stable closed-loop system, this approach has the advantage of enabling the efficient use of RRT algorithms on vehicles with unstable open-loop dynamics. For example, since the controller generates high-rate commands that stabilize and guide the vehicle along a given path, the motion planner output does not need to be very dense in time. As shown in Figure 1, the tree consists of the input to the controller (shown in blue) and the predicted trajectory (shown in green and red).

Algorithm 1 shows the overall flow. Similar to the standard RRT, the algorithm performs sampling (line 5), node selection (6), expansion (7), and constraint check (9), in this order. The planner provides the command to the controller at a fixed rate, and the tree expansion continues until this time limit is reached (23). The best trajectory is selected and then sent to the controller (28), and the tree expansion is resumed after updating the vehicle states and the situational awareness (2). The major extensions to RRT are presented in Section III.

The plan being executed is continuously monitored for collisions; should it become infeasible, and no other feasible trajectory is found while the car is moving, the planner commands an emergency braking maneuver. Such emergency conditions may arise only as a consequence of mismatches between the physical world and its representation used in the previous planning cycles (e.g., sudden appearance of an undetected obstacle in front of the vehicle).

For the selection of the best node sequence on line 28, each node stores two estimates of the cost-to-go: a lower bound and an upper bound. The Euclidean distance between the the node location and the target location is set as the lower bound. The upper bound is ∞ when no feasible trajectory to the target is found. Otherwise, the summation of the edge costs from the node to the target along the best node sequence gives an upper bound. Details on the edge cost are given in Subsection III-D.

Algorithm 1 RRT-based planning algorithm

```
1:  repeat
2:    Receive current vehicle states and environment
3:    Propagate states by computation time limit
4:    repeat
5:      Take a sample for input to controller
6:      Select a node in tree using heuristics
7:      Propagate from selected node to the sample until
        the vehicle stop
8:      Add branch nodes on the path
9:      if propagated path is feasible with the drivability
        map then
10:       Add sample and branch nodes to tree
11:     else
12:       if all the branch nodes are feasible then
13:         Add branch nodes to the tree and mark them
            as unsafe
14:       end if
15:     end if
16:     for each newly added node v do
17:       Propagate to the target
18:       if propagated path is feasible with drivability map
          then
19:         Add path to tree
20:         Set cost of propagated path as upper bound of
            cost-to-go at v
21:       end if
22:     end for
23:   until the time limit is reached
24:   Choose best safe trajectory in tree, and check feasi-
      bility with latest drivability map
25:   if best trajectory is infeasible then
26:     Remove infeasible portion from tree, goto line 24
27:   end if
28:   Send the best trajectory to controller
29: until Vehicle reaches the target
```

III. RRT Extensions

In order to efficiently generate the path in a dynamic and uncertain environment, several extensions have been made to the existing RRT approach [10]. The following subsections provide more details on the design choices embedded in the RRT algorithm. The line number in the subsection heading corresponds to that of Algorithm 1.

A. Biased sampling: line 5

The first extension to the RRT algorithm in [10] is that it uses the physical and logical structure of the environment to bias the sampling [11]. This biasing significantly increases the probability of generating feasible trajectories, enabling the online use of RRT. The samples are taken in 2D, and they are used to form the input to the steering controller.

The sample $(x_{\text{sample}}, y_{\text{sample}})$ is taken randomly but has some parameters to bias its location/shape, i.e.,

$$\begin{bmatrix} x_{\text{sample}} \\ y_{\text{sample}} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + r \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \text{ with } \begin{cases} r = \sigma_r |n_r| + r_0 \\ \theta = \sigma_\theta n_\theta + \theta_0 \end{cases}$$

where n_r and n_θ are random variables that have Gaussian distributions, σ_r and σ_θ give the 1-σ values of the radial and circumferential direction, r_0 and θ_0 are the offsets, and (x_0, y_0) is the center of the Gaussian cloud.

The uniqueness of this approach is that by varying the bias values based on the situational information, the planner can generate various maneuvers including lane following, passing, U-turn, and parking. Team MIT used a single planner for the entire race, which shows the flexibility and the extensibility of this planning algorithm.

B. Tree Expansion Heuristics: line 6

The algorithm has two connection heuristics, as introduced in [10]. Before a feasible trajectory to the target is found, the tree is grown mainly according to an *exploration* heuristic that attempts to connect the sample to the nearest node in the tree. The trajectory from the node to the sample tends to be short, and therefore is likely to be collision free. Then, most samples are added to the tree, resulting in rapid exploration of the environment with a smaller number of collision checks. This heuristic is widely used in the standard RRT, but when the vehicle has a minimum turn radius ρ, as in the car, the Euclidean distance between two points can be a poor estimate of the achievable path length.

A key difference between our approach and the previous work is the selection criteria of the nearest node, which uses the Dubins path length from the node to the sample. The node in the tree has a 2D position and a heading, and without loss of generality, it is assumed to be at the origin $p_0 = (0,0,0) \in SE(2)$. The sample is a 2D point represented by $q = (x, y) \in \mathbb{R}^2$. Since for the choice of p_0, the Dubins path lengths from p_0 to the points (x, y) and $(x, -y)$ are equal, it suffices to consider the case $\tilde{q} = (x, |y|) \in \mathbb{R} \times \mathbb{R}_+$. The minimum length $L_\rho(q)$ of a Dubins path from p_0 to q can be obtained analytically [12]. Let ρ denote the minimum turning radius of the vehicle. By defining $\mathscr{D}_\rho^+ = \{z \in \mathbb{R}^2 : \|z - (0,\rho)\| < \rho\}$, the minimum length is given by

$$L_\rho(q) = L_\rho(\tilde{q}) = \begin{cases} f(\tilde{q}) & \text{for } \tilde{q} \notin \mathscr{D}_\rho^+; \\ g(\tilde{q}) & \text{otherwise,} \end{cases}$$

where

$$f(\tilde{q}) = \sqrt{d_{\text{c}}^2(\tilde{q}) - \rho^2} + \rho \left(\theta_{\text{c}}(\tilde{q}) - \cos^{-1} \frac{\rho}{d_{\text{c}}(\tilde{q})} \right)$$

$$g(\tilde{q}) = \rho \left(2\pi - \alpha(\tilde{q}) + \sin^{-1} \frac{x}{d_{\text{f}}(\tilde{q})} + \sin^{-1} \frac{\rho \sin(\alpha(\tilde{q}))}{d_{\text{f}}(\tilde{q})} \right).$$

Here, $d_{\text{c}}(\tilde{q}) = \sqrt{x^2 + (|y| - \rho)^2}$ is the distance of $\tilde{q}$ from the point $(0,\rho)$, $\theta_{\text{c}}(\tilde{q}) = \text{atan2}(x, \rho - |y|)$ is the angle of $\tilde{q}$ from the point $(0,\rho)$, measured counter-clockwise from the negative y-axis, $d_{\text{f}}(\tilde{q}) = \sqrt{x^2 + (|y| + \rho)^2}$ is the distance of $\tilde{q}$ from the point $(0, -\rho)$, and $\alpha(\tilde{q}) = \cos^{-1}\left(\frac{5\rho^2 - d_{\text{f}}(\tilde{q})^2}{4\rho^2}\right)$. Note that the atan2 function is the 4 quadrant inverse tangent function with $\text{atan2}(a,b) = \tan^{-1}\left(\frac{a}{b}\right)$, and its range must be set to be $[0, 2\pi)$ to give a valid distance. This analytical calculation allows us to quickly evaluate all the nodes in the tree for a promising connection point.

Once a feasible trajectory to the target is found, the tree is grown primarily using an *optimization* heuristic that attempts to smooth out the trajectories. The nodes are now sorted by $h(v) + L_\rho(q)$, where $h(v)$ represents the cost from the root to the node v. This attempts to minimize the path cost from the root to the sample, resulting in a tree of trajectories that are close to optimal.

The implemented algorithm actually used both expansion modes at the same time but tended to place more emphasis on one or the other. The reason being that, even before having a trajectory that reaches the goal, there are benefits in performing some path optimization to reduce the waviness in the paths. Similarly, once a feasible trajectory to the goal has been found, there are still benefits in exploring the environment in case there is a better route, or another obstacle appears. In our implementation, the ratio of exploration vs. optimization heuristics was 70% vs. 30% before a trajectory to the target is found, and 30% vs. 70% once it is found.

C. Safety as an Invariant Property: line 7

Ensuring the safety of the vehicle is an important feature of any planning system, especially because the vehicle operates in a dynamic and uncertain environment. We define a state to be safe if the vehicle can remain in that state for an indefinite period of time, without violating the rules of the road and avoiding collisions with stationary and moving obstacles – where the latter are assumed to maintain their current driving path. A complete stop is used as safe states in this paper. More general notions of safe states are available [10], [13]. The large circles in Figure 1 show the safe stopping nodes in the tree. Each forward simulation terminates when the vehicle comes to a stop. By requiring that all leaves in the tree are safe states, this approach guarantees that there is always a feasible way to come to a stop while the car is moving. Unless there is a safe stopping node at the end of the path, the vehicle does not start executing it.

D. Risk Evaluation: line 9

The evaluation of whether a trajectory collides with obstacles or violates any rule of the road is done through the drivability map. This map is implemented using a grid-based lookup table with a resolution of 20 cm. All the perception data, including static and moving obstacles, lane boundaries, and road surface hazard, are rendered in the drivability map. Figure 2 shows a snapshot. The red region represents the non-drivable region, due to obstacle, lane, curb cuts, etc. The black/gray/white region represents it is drivable but with some penalty. The black means no penalty, and the white means high penalty, and the penalty value is scaled with the gray-scale color. The blue region is called "restricted", where the vehicle can drive through but is not allowed to stop. One typical use of the restricted region is with the obstacle on

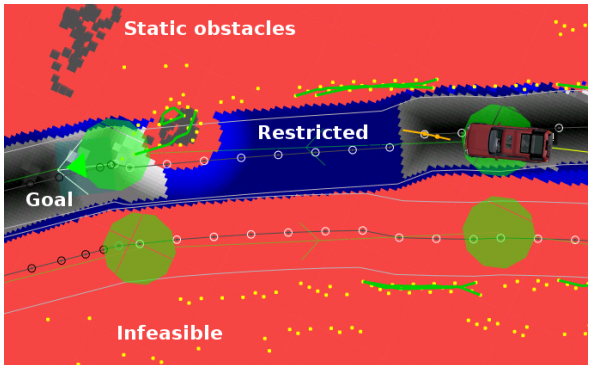


Fig. 2. Drivability map and its annotation.

the road. The vehicle stops with enough standoff distance to the obstacle, as shown in Figure 2, but once the oncoming lane becomes free to drive, the vehicle can go through the restricted area to pass the obstacle in the lane.

The penalty represents the risk in the environment, such as proximity to obstacles or lane boundaries, and curb cuts that are too faint to declare non-drivable. When evaluating the feasibility of the trajectory using the lookup table, the penalty stored in the drivability map is also obtained. The cost of the edge is then defined as the sum of travel time and the path integral of the penalty. Using this combined metric, the best trajectories tend to stay away from obstacles and lane boundaries, while allowing the vehicle to come close to constraints on a narrow road.

E. Unsafe Node: line 13

Another critical difference from the previous work [10] is the notion of "unsafe" node. In [10], when the propagated trajectory is not collision-free, the entire trajectory is discarded. In our approach, when only the final portion of the propagated trajectory is infeasible, the feasible portion of the trajectory is added to the tree. This avoids wasting the computational effort to find a good sample, propagate, and check for collision, while retaining the possibility to execute the portion that is found to be feasible. Because this trajectory does not end in a stopped state, the newly added nodes are marked as "unsafe". Then, if a safe trajectory, which ends in a stopped state, is added to the unsafe node, the node is marked as safe. When selecting the best trajectory (line 24) to send to the controller, the unsafe nodes are not considered. This approach uses unsafe nodes as potential connection points for samples, increasing the density of the tree, while ensuring the safety of the vehicle.

F. Lazy Check: line 24

In contrast to most motion planning algorithms, our RRT algorithm keeps growing the tree while the vehicle execute its portion. When the perceived environment is updated, the feasibility of each edge in the tree should be checked with the latest situational awareness. However, in the dynamically changing environment, a large tree would require constant feasibility re-checking for its thousands of edges, which

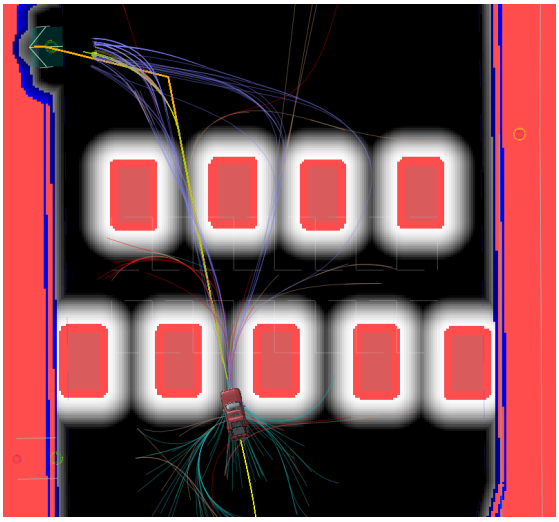


Fig. 3. Planning around obstacles. The vehicle started in the bottom, and the goal is in the upper left corner.

reduces the time the algorithm can spend on growing the tree.

The approach taken to overcome this issue was to re-evaluate the edge feasibility only when the edge is selected as the best trajectory sequence and is being sent to the controller. When the best trajectory is infeasible, the infeasible portion of the tree is deleted and the next best sequence is selected for re-evaluation. This so-called "lazy check" enables the algorithm to focus mainly on growing the tree, while ensuring that the executed trajectory is feasible in the latest drivability map. The difference from the previous work [14], [15] is that the lazy check in this paper is about re-checking of the constraints for previously feasible edges, whereas the previous work is about delaying the first collision detection in the static environment.

One limitation with the lazy check is that the penalty stored in the tree could be based on the obsolete situational awareness. Then, the best selected trajectory could be very close to the constraints. As long as it is feasible with the latest map, the planner sends it to the controller. In the constantly-changing dynamic environment, however, the computational efficiency obtained by the lazy check typically outweighs the potential risk to come close to the constraints.

IV. Application Results

This section presents four examples from simulation and data logged during the DUC. The planner runs at $\sim$10 Hz, and the average number of samples generated was about 700 samples per second on a dual-core 2.33 GHz Intel Xeon processor. Note that because we sample the controller input, a single sample could create a trajectory as long as a few seconds. The tree had about 1200 nodes on average.

Figure 3 shows navigation in the obstacle field. The orange line segments represents the input to the controller, generated by the sampling, and the green line shows the corresponding predicted trajectory. The following colors were used to

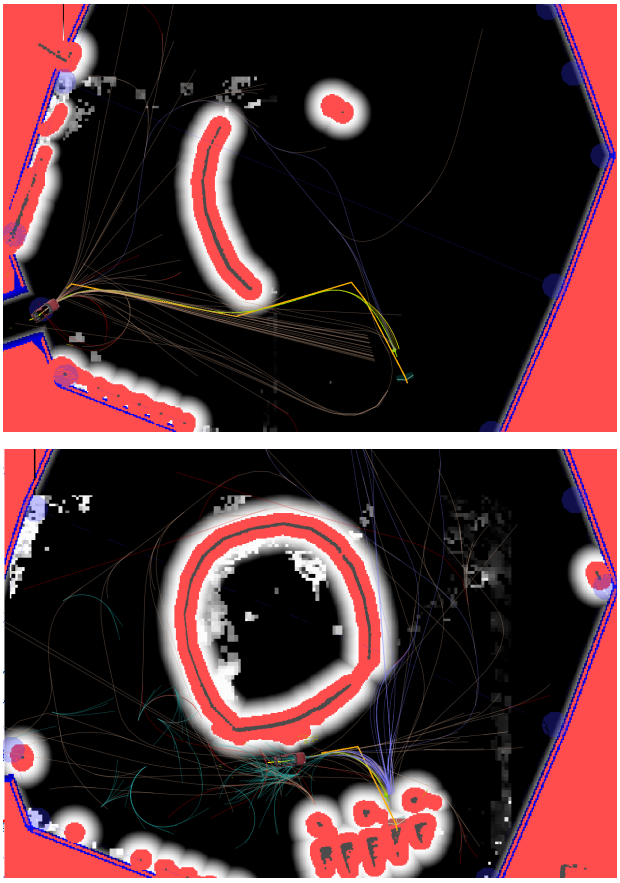


Fig. 4. Planning of a parking maneuver. Talos enters the zone from the left (top figure) aiming for a given parking point. Talos navigates around the central obstacle and perceives the parked cars around the target, while continuously planning a feasible path to the parking spot (bottom figure).

indicate different types of trajectories in the tree: purple (reaching the target), light brown (safe but not reaching the target), red (unsafe), and cyan (reverse). Several paths successfully reach the goal, and other paths explore the space around and between obstacles. For example, the tree contains paths, that reach the target by going to the right of obstacles; however, these paths are not selected because they are not of minimal cost. Note that unlike many implementations of the RRT, the generated paths are smooth. This behavior results from the propagation over the closed-loop system, and the optimization heuristics used when connecting samples to the tree.

Figure 4 shows the tree during the parking exercise in a run at the National Qualifying Event (NQE). Talos found a trajectory to the parking spot immediately after entering the parking zone, and the top figure demonstrates the exploration capabilities of the algorithm. In addition to the several paths that reach the parking spot, some paths reach the parking spot with different headings, and some others go around the partially detected large obstacle in the middle. As the vehicle proceeds, several obstacles were detected around the parking spot, as shown in the bottom figure. The tree contains various forward and reverse trajectories to complete the tight parking maneuver.

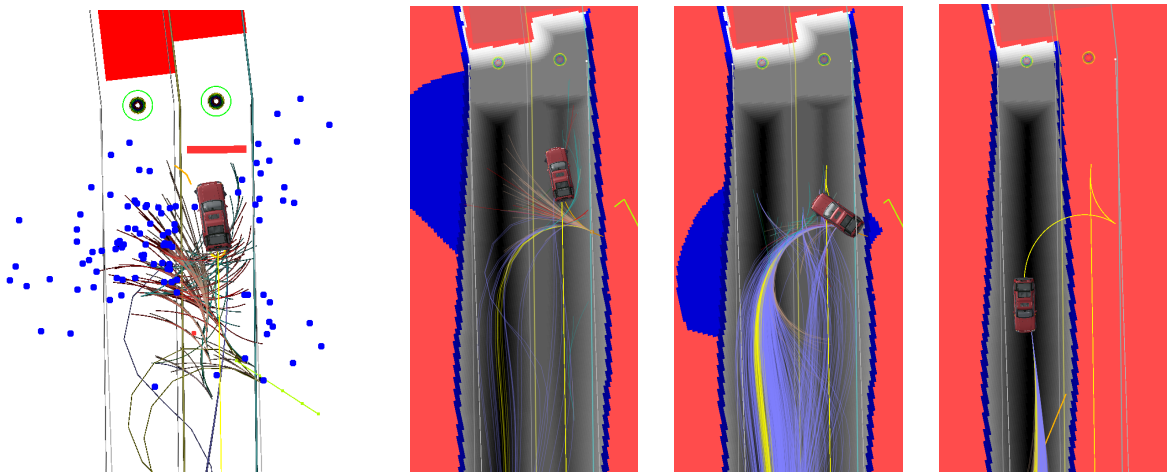
The third result is the U-turn, one of the more elaborate maneuvers required for the DUC, as shown in Figure 5. Biased sampling consisting of three Gaussian clouds with different traveling directions efficiently constructed a U-turn maneuver. The first cloud contains forward samples, generated to the left-front of the vehicle to initiate the turn. The second set consists of reverse samples, generated to the right-forward of the vehicle, which will be used during the reverse maneuver after executing the first forward leg of the turn. The third set is forward samples, generated to the left-rear of the current vehicle location for use when completing the turn. The parameter values used for each of the three sets are: $\sigma_r = 8$, $\sigma_\theta = \pi/10$, $r_0 = 3$, $\theta_0 = 4\pi/9$ (first cloud); $\sigma_r = 10$, $\sigma_\theta = \pi/10$, $r_0 = 5$, $\theta_0 = -\pi/4$ (second cloud); and $\sigma_r = 12$, $\sigma_\theta = \pi/10$, $r_0 = 7$, $\theta_0 = \pi$ (third cloud). The first Gaussian cloud is centered on the location of the vehicle before initiating the U-turn maneuver, whereas the other two clouds are centered on the location of the selected sample from the preceding cloud. Figure 5a is a snapshot of sample points generated in 0.1 second. A small cloud of reverse samples is also generated behind the vehicle in case it stopped very close to the road blockage, requiring a reverse maneuver to start the U-turn.

The last result in Figure 6 shows lane following along a curvy section of a road during the Urban Challenge Event (UCE). Talos is in the left of the figure heading towards the right. Note that a curb, shown by a red line near the bottom of the lane, projects into the offline estimate of the lane boundaries. This narrows the part of road that is passable, a challenging situation for the motion planner. In normal driving conditions, the sampling was biased along the lane, and the RRT successfully finds smooth trajectories along the narrow curvy lane.

V. Conclusion

This paper presented the design and implementation of an efficient and reliable motion planning system, based on RRTs, for Team MIT's entry to the DUC. The standard RRT is extended in several ways to be used for a large robotic vehicle driving in the dynamic and uncertain urban environment. To improve the computational efficiency, the input to the closed-loop system is sampled, which also enables RRT to handle complex/unstable dynamics of the vehicle. The lazy check enables RRT to focus on the tree expansion even with the constantly changing situational awareness. The uncertainty in the environment is captured in the form of a risk penalty in the tree. The sampling using the environmental structure also significantly reduced the time to find trajectories for various maneuvers. The safety of the vehicle is guaranteed by requiring that the trajectory sent to the controller end in a stopping state. The advantages of these features are demonstrated through several simulation and race results.

The algorithm was not tuned to any specific test cases posed by DARPA during the NQE or UCE. Furthermore,



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

Fig. 5. Planning of a U-turn maneuver. Figures 5b, 5c, and 5d show different evolutions of the tree as the vehicle executes a U-turn. Notice in Figure 5d the trace of the path followed by the vehicle (shown in yellow).

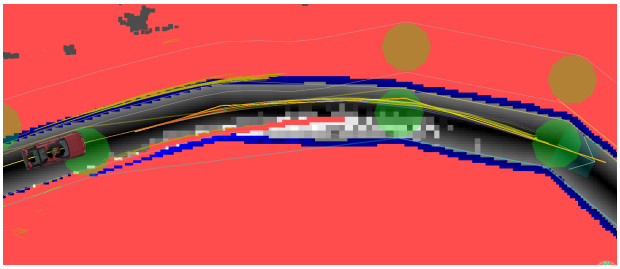


Fig. 6. Planning along a lane that becomes increasingly narrow due to detected curbs (red line emerging into the lane on the bottom). The motion planner is still able to find feasible trajectories that follow the road curvature.

there were numerous traffic and intersection scenarios that had never been tested before, and yet the motion planner demonstrated that it was capable of handling these situations successfully. The completion of the UCE using a single planner clearly demonstrated that this is a general-purpose motion planner capable of handling uncertain and very dynamic driving scenarios.

ACKNOWLEDGMENT

Research sponsored by DARPA, Program: Urban Challenge, DARPA Order No. W369/00, Program Code: DIRO. Issued by DARPA/CMO under Contract No. HR0011-06-C-0149, with J. Leonard, S. Teller, J. How at MIT and D. Barrett at Olin College as the PI's. The authors gratefully acknowledge the support of the MIT Urban Challenge Team, particularly Dr. Luke Fletcher and Edwin Olson for developing the drivability map, David Moore for the Navigator, and Sertac Karaman for his contributions to the low-level controller.

REFERENCES

[1] K. Iagnemma and M. Buehler, eds., "Special issue on the 2007 DARPA Urban Challenge," *Journal of Field Robotics*, 2008, to appear.

[2] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Mahelona, K. Moyer, T. Jones, R. Buckley, M. Attone, R. Galejs, S. Krishnamurthy, and J. Williams, "A perception driven autonomous urban robot," submitted to *International Journal of Field Robotics*, 2008.

[3] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[4] J.-P. Laumond, Ed., *Robot Motion Planning and Control*, ser. Lectures Notes in Control and Information Sciences. Springer Verlag, 1998, vol. 229.

[5] H. Choset, K. Lynch, S. Hutchinson, G. kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston, MA: MIT Press, 2005.

[6] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[7] K. Iagnemma and M. Buehler, "Special issue on the DARPA Grand Challenge: Editorial," *Journal of Field Robotics*, vol. 23, no. 9, pp. 655–656, 2006.

[8] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[9] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How, "Motion Planning in Complex Environments using Closed-loop Prediction," submitted to AIAA Conference on Guidance, Navigation, and Control.

[10] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA Journal of Guidance and Control*, vol. 25, no. 1, pp. 116–129, 2002.

[11] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *Proceedings IEEE International Conference on Robotics & Automation*, 2003.

[12] J. J. Enright, E. Frazzoli, K. Savla, and F. Bullo, "On multiple uav routing with stochastic targets: Performance bounds and algorithms," in *Proceedings AIAA Guidance, Navigation, and Control Conference and Exhibit*, Aug. 2005.

[13] T. Schouwenaars, J. How, and E. Feron, "Receding horizon path planning with implicit safety guarantees," in *Proceedings American Control Conference*, vol. 6, Jun. 2004.

[14] R. Bohlin and L. Kavraki, "Path planning using Lazy PRM," in *Proceedings IEEE International Conference on Robotics & Automation*, 2000.

[15] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Proceedings International Symposium on Robotics Research*, 2001.