Real-time Motion Planning with Applications to Autonomous Urban Driving

Yoshiaki Kuwata, *Member, IEEE,* Justin Teo, *Student Member, IEEE,*
Gaston Fiore, *Student Member, IEEE,* Sertac Karaman, *Student Member, IEEE,*
Emilio Frazzoli, *Senior Member, IEEE,* and Jonathan P. How, *Senior Member, IEEE*

***Abstract*—This paper describes a real-time motion planning algorithm, based on the Rapidly-exploring Random Tree (RRT) approach, applicable to autonomous vehicles operating in an urban environment. Extensions to the standard RRT are predominantly motivated by: (i) the need to generate dynamically feasible plans in real-time, (ii) safety requirements, (iii) the constraints dictated by the uncertain operating (urban) environment. The primary novelty is in the use of closed-loop prediction in the framework of RRT. The proposed algorithm was at the core of the planning and control software for Team MIT's entry for the 2007 DARPA Urban Challenge, where the vehicle demonstrated the ability to complete a 60 mile simulated military supply mission, while safely interacting with other autonomous and human driven vehicles.**

***Index Terms*—Real-time motion planning, dynamic and uncertain environment, RRT, urban driving, autonomous, DARPA Urban Challenge.**

I. Introduction

In November 2007, several autonomous automobiles from all over the world competed in the DARPA Urban Challenge (DUC), which was the third installment of a series of races for autonomous ground robots [1]. Unlike previous events that took place on desert dirt roads, the DUC course consisted mainly of paved roads, incorporating features like intersections, rotaries, parking lots, winding roads, and highways, typically found in urban/suburban environments. The major distinguishing characteristic of the DUC was the introduction of traffic, with up to 70 robotic and human-driven vehicles on the course simultaneously. This resulted in hundreds of unscripted robot-on-robot (and robot on human-driven vehicle) interactions. With a longer term goal of fielding such autonomous vehicles in environments with co-existing traffic and/or traffic infrastructure, all vehicles were required to abide by the traffic laws and rules of the road. Vehicles were expected to stay in the correct lane, maintain a safe speed at or under specified speed limits, correctly 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 [2], [3]. The reader is referred to [3] for a system-level report on the design and development of Team MIT's vehicle, *Talos*. The focus of this paper is on the motion planning subsystem (or "motion planner") and the associated controller (which is tightly coupled with the motion planner). The motion planner is an intermediate level planner, with inputs primarily from a high level planner called the Navigator, and outputs being executed by the low level controller. Given the latest environment information and vehicle states, the motion planner computes a *feasible* trajectory to reach a goal point specified by the Navigator. This trajectory is feasible in the sense that it avoids static obstacles and other vehicles, and abide by the rules of the road. The output of the motion planner is then sent to the controller, which interfaces directly to the vehicle, and is responsible for the execution of the motion plan.

The main challenges in designing the motion planning subsystem resulted from the following factors: (i) complex and unstable vehicle dynamics, with substantial drift, (ii) limited sensing capabilities, such as range and visibility, in an uncertain, time-varying environment, and (iii) temporal and logical constraints on the vehicle's behavior, arising from the rules of the road.

Numerous approaches to address the motion planning problem have been proposed in the literature, and the reader is referred to [1], [4]–[8], to name a few. For our motion planning system, we chose to build it on the Rapidly-exploring Random Trees (RRT) algorithm [9]–[11], which belongs to the class of incremental sampling-based methods [7, 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, and (iii) sampling-based methods do not require the explicit enumeration of constraints, but allow trajectory-wise checking of possibly very complex constraints.

In spite of their generality, the application of incremental sampling-based motion planning methods to robotic vehicles

Y. Kuwata was with Department of Aeronautics & Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA. He is now with Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109 USA. e-mail: Yoshiaki.Kuwata@jpl.nasa.gov.

J. Teo, E. Frazzoli and J. How are with Department of Aeronautics & Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA. e-mail: {csteo, frazzoli, jhow}@mit.edu.

G. Fiore was with Department of Aeronautics & Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA. He is now with School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, 02138 USA. e-mail: gafiore@alum.mit.edu.

S. Karaman is with Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA. e-mail: sertac@mit.edu.

Manuscript received September 12, 2008; revised December 23, 2008.

with complex and unstable dynamics, such as the full-size 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 been restricted either to simulation, or to kinematic (essentially driftless) robots (i.e., it can be stopped instantaneously by setting the control input to zero), and never been used in on-line planning systems for robotic vehicles with the above characteristics.

This paper reports on the design and implementation of an efficient and reliable general-purpose motion planning system, based on RRTs, for our team's entry to the DUC. In particular, we present an approach that enables the on-line use of RRTs on robotic vehicles with complex, unstable dynamics and significant drift, while preserving safety in the face of uncertainty and limited sensing. The effectiveness of our motion planning system is discussed, based on the analysis of actual data collected during the DUC race.

II. Problem Formulation

This section defines the motion planning problem. The vehicle has nonlinear dynamics

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ and $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ are the states and inputs of the system, and $\boldsymbol{x}_0 \in \mathbb{R}^{n_x}$ is the initial states at $t = 0$. The input $\boldsymbol{u}(t)$ is designed over some (unspecified) finite horizon $[0,\, t_f]$. Bounds on the control input, and requirements of various driving conditions, such as static and dynamic obstacles avoidance and the rules of the road, can be captured with a set of constraints imposed on the states and the inputs

$$\boldsymbol{x}(t) \in X_{\text{free}}(t), \qquad \boldsymbol{u}(t) \in U. \tag{2}$$

The time dependence of X_{free} expresses the avoidance constraints for moving obstacles. The goal region $X_{\text{goal}} \subset \mathbb{R}^{n_x}$ of the motion planning problem is assumed to be given by a higher level route planner. The primary objective is to reach this goal with the minimum time

$$t_{\text{goal}} = \inf \{t \in [0, t_f] \mid \boldsymbol{x}(t) \in X_{\text{goal}}\} \tag{3}$$

with the convention that the infimum of an empty set is $+\infty$. A vehicle driving too close to constraints such as lane boundaries incurs some penalty, which is modeled with a function $\psi(X_{\text{free}}(t), \boldsymbol{x}(t))$. The motion planning problem is now defined as:

Problem II.1 (Near Minimum-Time Motion Planning)**.** Given the initial states $\boldsymbol{x}_0$ and the constraint sets U and $X_{\text{free}}(t)$, compute the control input sequence $\boldsymbol{u}(t)$, $t \in [0, t_f]$, $t_f \in [0, \infty)$ that minimizes

$$t_{\text{goal}} + \int_0^{t_{\text{goal}}} \psi\left(X_{\text{free}}(t), \boldsymbol{x}(t)\right) dt,$$

while satisfying Eqs. (1)–(3).

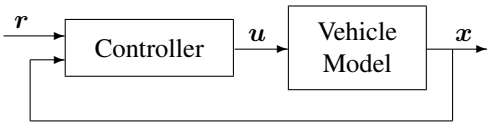


Fig. 1. Closed-loop prediction. Given a reference command $\boldsymbol{r}$, the controller generates high rate vehicle commands $\boldsymbol{u}$ to close the loop with the vehicle dynamics.

III. Planning over Closed-loop Dynamics

The existing randomized planning algorithms solve for the input to the vehicle $\boldsymbol{u}(t)$ either by sampling an input itself or by sampling a configuration and reverse calculating $\boldsymbol{u}(t)$, typically with a lookup table [9], [10], [12]–[14]. This paper presents the *Closed-loop RRT (CL-RRT)* algorithm, which extends the RRT by making use of a low-level controller and planning over the closed-loop dynamics. In contrast to the existing work, CL-RRT samples an input to the *stable closed-loop system* consisting of the vehicle and the controller [15].

Figure 1 shows the forward simulation of the closed-loop dynamics. The low-level controller takes a reference command $\boldsymbol{r}(t) \in \mathbb{R}^{n_r}$. For vehicles with complex dynamics, the dimension of the vehicle states n_x can be quite large, but the reference command $\boldsymbol{r}$ typically has a lower dimension (i.e., $n_r \ll n_x$). For example, in our application, the reference command is a 2D path for the steering controller and a speed command profile for the speed controller. The vehicle model in this case included 7 states, and the reference command consists of a series of triples (x, y, v_{cmd}), where x and y are the position of the reference path, and v_{cmd} is the associated desired speed. For urban driving, the direction of the vehicle motion (forward or reverse) is also part of the reference command, although this direction needs to be defined only once per reference path.

Given the reference command, CL-RRT runs a forward simulation using a vehicle model and the controller to compute the predicted state trajectory $\boldsymbol{x}(t)$. The feasibility of this output is checked against vehicle and environmental constraints, such as rollover and obstacle avoidance constraints.

This closed-loop approach has several advantages when compared to the standard approach that samples the input $\boldsymbol{u}$ to the vehicle [7], [13]. First, CL-RRT works for vehicles with unstable dynamics, such as cars and helicopters, by using a stabilizing controller. Second, the use of a stabilizing controller provides smaller prediction error because it reduces the effect of any modeling errors in the vehicle dynamics on the prediction accuracy, and also rejects disturbances/noises that act on the actual vehicle. Third, the forward simulation can handle any nonlinear vehicle model and/or controller, and the resulting trajectory $\boldsymbol{x}(t)$ satisfies Eq. (1) by construction. Finally, a single input to the closed-loop system can create a long trajectory (on the order of several seconds) while the controller provides a high-rate stabilizing feedback to the vehicle. This requires far fewer samples to build a tree, improving the efficiency (e.g., number of samples per trajectory length) of randomized planning approaches.

Algorithm 1 Expand_tree()

1: Take a sample $\boldsymbol{s}$ for input to controller.
2: Sort the nodes in the tree using heuristics.
3: **for** each node q in the tree, in the sorted order **do**
4: Form a reference command to the controller, by connecting the controller input at q and the sample $\boldsymbol{s}$.
5: Use the reference command and propagate from q until vehicle stops. Obtain a trajectory $\boldsymbol{x}(t)$, $t \in [t_1, t_2]$.
6: Add intermediate nodes q_i on the propagated trajectory.
7: **if** $\boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}(t)$, $\forall t \in [t_1, t_2]$ **then**
8: Add sample and intermediate nodes to tree. Break.
9: **else if** all intermediate nodes are feasible **then**
10: Add intermediate nodes to tree and mark them unsafe. Break.
11: **end if**
12: **end for**
13: **for** each newly added node q **do**
14: Form a reference command to the controller, by connecting the controller input at q and the goal location.
15: Use the reference command and propagate to obtain trajectory $\boldsymbol{x}(t)$, $t \in [t_3, t_4]$.
16: **if** $\boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}(t)$, $\forall t \in [t_3, t_4]$ **then**
17: Add the goal node to tree.
18: Set cost of the propagated trajectory as an upper bound C_{UB} of cost-to-go at q.
19: **end if**
20: **end for**

IV. Tree Expansion

Summarizing the previous section, the CL-RRT algorithm grows a tree of feasible trajectories originating from the current vehicle state that attempts to reach a specified goal set. At the end of the tree growing phase, the best trajectory is chosen for execution, and the cycle repeats. This section discusses how a tree of vehicle trajectories is grown, using the forward simulation presented in Section III, and identifies several extensions made to the existing work. Algorithm 1 shows the main steps in the tree expansion (called Expand_tree() function). Similar to the original RRT algorithm, CL-RRT takes a sample (line 1), selects the best node to connect from (line 2), connects the sample to the selected node (line 4), and evaluates its feasibility (line 7). Line 6 splits the trajectory between the newly added node and its parent into $n_b > 1$ (but typically $n_b \leq 4$) segments so that future samples can be connected to such nodes to create new branches in the tree.

The CL-RRT algorithm differs from the original RRT in several ways, including: samples are drawn in the controller's input space (line 1); nonlinear closed-loop simulation is performed to compute dynamically feasible trajectories (line 5); the propagation ensures that the vehicle is stopped and safe at the end of the trajectory (line 5); and the cost-to-go estimate is obtained by another forward simulation towards the goal (line 15).

Figure 2 shows an example of a tree generated by the Expand_tree() function of the CL-RRT algorithm. The tree consists of reference paths that constitute the input to the controller (orange) and the predicted vehicle trajectories (green and red). The trajectories found to be infeasible are marked in red. The large dots (•) at the leaves of the tree indicate that the vehicle is stopped and is in a safe state (discussed in detail in Section IV-E). Given this basic outline, the following subsections discuss the additional extensions to the RRT approach in Ref. [13] in more detail.

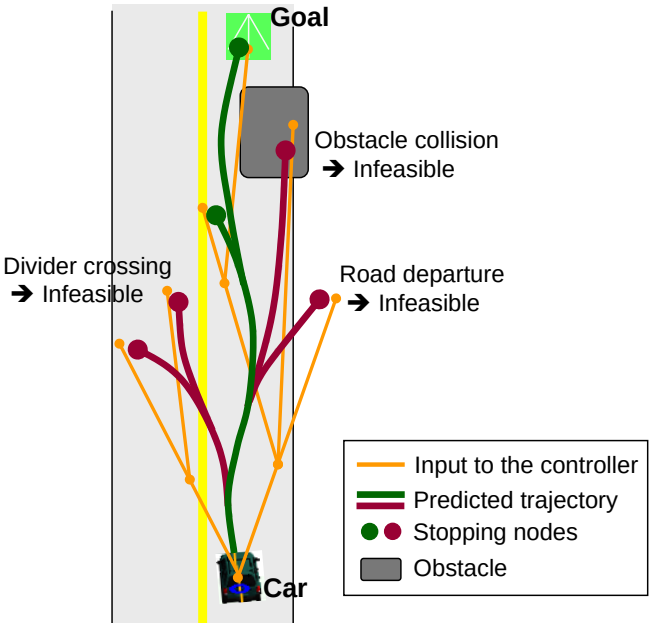


Fig. 2. Vehicle trajectories are generated using a model of the vehicle dynamics. The generated trajectories are then evaluated for feasibility and performance.

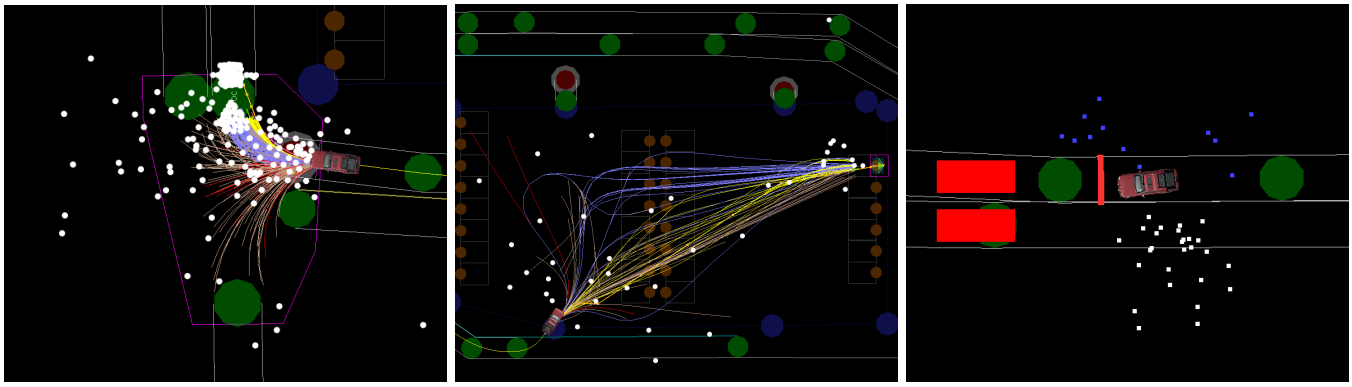
A. Sampling Strategies

In a structured environment such as urban driving, sampling the space in a purely random manner could result in large numbers of wasted samples due to the numerous constraints. Several methods have been proposed to improve the efficiency [16]–[20]. This subsection discusses a simple strategy that uses the physical and/or logical environment to bias the Gaussian sampling clouds to enable real-time generation of complex maneuvers.

The reference command for the controller is specified by an ordered list of triples $(s_x, s_y, v_{\text{cmd}})_i$, for $i \in \{1, 2, \ldots, n_{\text{ref}}\}$, together with a driving direction (forward/reverse). The 2D position points (s_x, s_y) are generated by random sampling, and the associated speed command for each point, v_{cmd}, is designed deterministically to result in a stopped state at the end of the reference command [15]. Each sample point $\boldsymbol{s} = [s_x, s_y]^{\text{T}}$ is generated with respect to some reference position and heading (x_0, y_0, θ_0) by

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + r \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \quad \text{with} \quad \begin{cases} r = \sigma_r |n_r| + r_0, \\ \theta = \sigma_\theta n_\theta + \theta_0. \end{cases}$$

where n_r and n_θ are random variables with standard Gaussian distributions, σ_r is the standard deviation in the radial direction, σ_θ is the standard deviation in the circumferential direction, and r_0 is an offset with respect to (x_0, y_0). Various



(a) Intersection. The vehicle is on the right, trying to make a right turn.

(b) Parking lot. The vehicle is in the lower left corner, and the goal is in the center right edge.

(c) U-turn maneuver. The vehicle is facing the road blockage, shown in red. White and blue samples corresponds to forward and reverse maneuvers respectively.

Fig. 3. Various sampling clouds (dots show samples generated in one planning cycle; lines show predicted trajectories).

maneuvers are generated simply by changing these parameters according to the vehicle location and the rules of the road.

1) Physical Environment as a Bias: On a lane, the sampling starts with randomly shifting the estimated lane center points in the lateral direction, to fully exploit the lane shape. After a few hundred samples, it switches to a long and narrow Gaussian cloud whose longitudinal axis follows the car heading.

At an intersection, a wide and relatively short Gaussian distribution is used that covers the open space inside the intersection boundary (see Figure 3(a)). The value of σ_r in intersections is set to the distance to the goal and σ_θ is set at 0.4π. Note that the samples do not necessarily lie inside the lane. This is because they are used only to construct the reference path, while the resultant vehicle trajectory may not track it exactly.

The sample $\boldsymbol{s}$ can consist of an ordered set of n_s points $\boldsymbol{s}_i$, $i \in \{1, \ldots, n_s\}$, called *batch sampling*. The input to the controller is generated by connecting these ordered points, and the first point in the sample is connected to a node in the tree (line 4). Batch sampling is useful when the intended trajectory has some specific shape, such as arriving at a goal with a certain heading. By placing the last two points of a batch sample along the direction of the goal heading, the planner can align the vehicle heading towards the goal heading.

In parking lots, sampling is taken both around the vehicle and around the parking spot. Around the vehicle, a wide and long Gaussian distribution in the direction of vehicle heading is used, while around the parking spot, the sampling is performed along the center line of the intended parking spot. Figure 3(b) shows an example of the parking samples. The values of $\sigma_r = 50$ m and $\sigma_\theta = \pi$ are used in the parking zones.

2) Logical Environment as a Bias: When passing a stopped vehicle or obstacles in the lane, a wider Gaussian distribution is used ($\sigma_\theta = 0.4\pi$) compared to the normal lane following ($\sigma_\theta = 0.056\pi$, or 0.25π if no trajectory to the goal is found).

For a U-turn, the sampling is biased for a three-point turn, as shown in Figure 3(c). The location of the different regions correspond to the different phases of a three-point turn. During a three-point turn, the vehicle first travels forward left; then reverses right, moving into the original lane of travel; and finally moves forward into the targeted lane with the correct heading. The parameter values used for each of the three sample sets are:

$$\begin{array}{lllll} 1^{\text{st}}: & \sigma_r = 5, & \sigma_\theta = 0.1\pi, & r_0 = 3, & \theta_0 = 0.44\pi \\ 2^{\text{nd}}: & \sigma_r = 5, & \sigma_\theta = 0.2\pi, & r_0 = 3, & \theta_0 = -0.17\pi \\ 3^{\text{rd}}: & \sigma_r = 10, & \sigma_\theta = 0.25\pi, & r_0 = 3, & \theta_0 = 0.83\pi \end{array}$$

The units are meters for r and radians for θ. Here, the origin (x_0, y_0) is the car location before initiating the U-turn, and σ_θ's are measured from the car heading there. Sometimes the car stops very close to the road blockage, requiring a short reverse maneuver before initiating a U-turn, so an additional cloud of reverse samples is also generated behind the vehicle.

B. Feasibility Check and Risk Evaluation

After the forward propagation, the resulting predicted trajectory is tested with obstacles and rules of the road, as shown in Figure 2. All the physical and logical constraints, such as static and moving obstacles, lane boundaries, and standoff distance are encoded in a *drivability map* [3], which is updated at 10 Hz. This drivability map is a 2D grid in (x, y) whose resolution is 20 cm, which gave enough accuracy with a manageable table size. Each grid cell stores a drivable/non-drivable flag and the associated penalty if drivable. This implementation has the advantage of performing the penalty evaluation at the same time as the binary collision check without additional computation. When evaluating the feasibility of a rectangular car with a specific heading, line searches are performed over the 2D grid along the longitudinal direction of the car.

Figure 4 shows the three different regions encoded in the drivability map. These are *infeasible* (red), *restricted* (blue), and *drivable with penalty* (white or gray or black). Infeasible regions represent obstacles and lane boundaries. Restricted regions may be driven through but only if the vehicle can then drive out of them. They are used to prevent the vehicle from stopping too close to obstacles, and also to maintain sufficient standoff distance before a passing maneuver. The risky regions such as those near obstacles or lane boundaries are marked

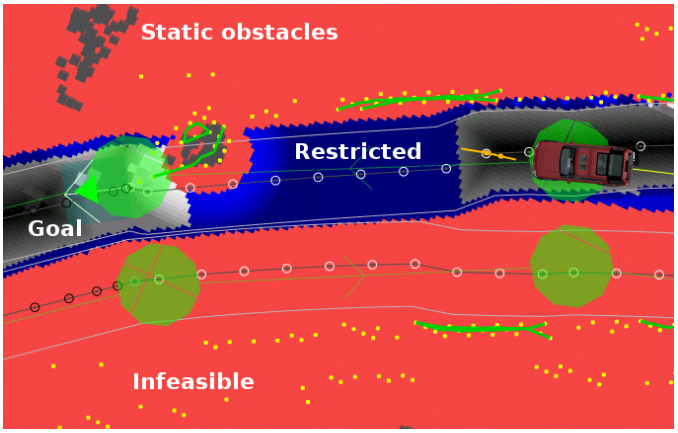


Fig. 4. Snapshot of the cost map.

as drivable with penalty. By adding the path integral of the penalty to the time to traverse, the CL-RRT does not choose paths that closely approach constraint boundaries, unless there is no other option (e.g., narrow passage).

C. Node Connection Strategy

This subsection describe how the nodes in the tree are sorted on line 2 of Algorithm 1, when connecting a sample to the tree.

1) Distance Metric: RRT attempts to connect the sample to the closest node in the tree. Because shorter paths have lower probability of collision, this approach requires minimum function calls for the computationally expensive collision detection [7]. RRT can thus quickly cover the free space without wasting many samples. Usually, the two-norm distance is used to evaluate the "closeness", but an extension is required when RRT is applied to a vehicle with limited turning capability. For example, a car would need to make a big loop to reach a point right next to the current location, due to the nonholonomic constraints. The most accurate representation for this distance would be the simulated trajectory of the vehicle as determined by the propagation of the vehicle model. However, performing a propagation step for each node in the tree would impose a heavy computational burden. Thus the CL-RRT algorithm uses the length of the Dubins path between a node and the sample. The use of Dubins path lengths has the advantage of capturing the vehicle's turning limitation with a simple evaluation of an analytical solution. A full characterization of optimal Dubins paths is given in [21] and a further classification in [22].

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 $p_0 = (0,\, 0,\, 0) \in SE(2)$, where $SE(n)$ denotes a special Euclidean group of dimension n. 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 [23]. Let ρ denote the minimum turning radius of the vehicle. By defining $\mathcal{D}_\rho = \{z \in \mathbb{R}^2 : \|z - (0,\, \rho)\| < \rho\}$, the minimum length is given by

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

where

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

$$g(\tilde{q}) = \rho \left(\alpha(\tilde{q}) + \sin^{-1} \frac{x}{d_{\mathrm{f}}(\tilde{q})} - \sin^{-1} \frac{\rho \sin(\alpha(\tilde{q}))}{d_{\mathrm{f}}(\tilde{q})} \right),$$

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

2) Selection Criteria: Several heuristics have been proposed in the past [13], [24] to select the node to connect the sample to. Because CL-RRT was designed to be used in a constrained environment, it attempts multiple nodes in some order (~ 10 in our implementation) before discarding the sample. The following presents the heuristics used to sort the nodes in the tree.

Before a trajectory to the goal is found, the tree is grown mainly according to an *exploration* heuristic, in which the emphasis of the algorithm is on adding new nodes to the tree to enlarge the space it covers. The nodes are sorted according to the Dubins distance $L_\rho(\boldsymbol{s})$, as presented above. Once a feasible trajectory to the goal has been found, the tree is grown primarily according to an *optimization* heuristic. To make the new trajectories progressively approach the shortest path, the nodes are now sorted in ascending order of total cost to reach the sample. This total cost C_{total} is given by

$$C_{\mathrm{total}} = C_{\mathrm{cum}}(q) + L_\rho(\boldsymbol{s})/v,$$

where $C_{\mathrm{cum}}(q)$ represents the cumulative cost from the root of the tree to a node q, v is the sampled speed, and the second term estimates the minimum time it takes to reach the sample from q.

Our experience has shown that, even before having a trajectory that reaches the goal and consequently when the focus should be on exploration, some use of the optimization heuristics is beneficial to reduce wavy trajectories. Similarly, once a feasible trajectory to the goal has been found and therefore the focus should be on refining the available solution, some exploration of the environment is beneficial for example in case an unexpected obstacle blocks the area around the feasible solution, or for having a richer tree when the goal location changes. Thus when implementing the algorithm, the ratio of exploration vs. optimization heuristics used was 70% vs. 30% before a trajectory to the goal was found, and 30% vs. 70% after.

Note that stopping nodes in the tree are not considered as potential connection points for the samples, unless it is the only node of the tree or the driving direction changes. This prevents the vehicle from stopping for no reason and helps achieve smooth driving behaviors.

D. Cost-To-Go Estimate

Every time a new node q is added to the tree, CL-RRT computes estimates of the cost-to-go from q to the goal (line 13–20). There are two estimates of the cost-to-go at each node in the tree, a *lower bound* and an *upper bound*. The lower bound C_{LB} is given by the Euclidean distance between the vehicle position at the node and the position of the goal. The upper bound C_{UB} is obtained differently from [13], which assumed the existence of the optimal control policy.

For each newly added node q, a trajectory from q to the goal is generated and evaluated for feasibility. If it is collision-free, the cost associated with that trajectory gives the upper bound C_{UB} at q. This upper bound is then propagated from q backward towards the root to see if it gives a better upper bound from ancestor nodes of q. Thus, C_{UB} is given by

$$C_{\mathrm{UB}} = \begin{cases} \infty & : \text{no trajectory to the goal available} \\ \min\limits_{c}(e_c + C_{\mathrm{UB}c}) & : \text{trajectory to the goal available} \\ C_{\mathrm{LB}} & : \text{node is inside the goal region} \end{cases}$$

where c represents the index of each child of this node, e_c is the cost from the node to the child node c, and $C_{\mathrm{UB}c}$ represents the upper bound cost at node c.

Note that this process adds a node at the goal to the tree if the goal is directly reachable from q, which helps CL-RRT quickly find a trajectory to reach the goal in most scenarios. As time progresses, more trajectories that reach the goal are constantly added to the tree.

These cost estimates are used when choosing the best trajectory to execute (as shown later on line 13 of Algorithm 2). Once a feasible trajectory to the goal is found, the best sequence of nodes is selected based on the upper bound C_{UB}. Before a goal-reaching trajectory is found, the node with the best lower bound is selected, in effect trying to move towards the goal as much as possible.

E. Safety as an Invariant Property

1) Safe States: Ensuring the safety of the vehicle is an important feature of any planning system, especially when the vehicle operates in a dynamic and uncertain environment. We define a state $\boldsymbol{x}(t_0)$ to be safe if the vehicle can remain in that state for an indefinite period of time

$$\dot{\boldsymbol{x}}(t_0) = 0, \quad \boldsymbol{x}(t_0) \in X_{\mathrm{free}}(t) \quad \forall t \geq t_0$$

without violating the rules of the road and at the same time is not in a collision state/path with stationary and/or moving obstacles – where the latter are assumed to maintain their current driving path. A complete stop is used as safe state in this paper. More general notions of safe states are available [13], [25]. The large circles in Figure 2 show the safe stopping nodes in the tree. Given a reference path (shown with orange), a speed profile is designed in such a way that the vehicle comes to a stop at the end. Then, 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. With stopping nodes at the leaves of the tree, some behaviors such as stopping at a stop line emerge naturally.

2) Unsafe Node: A critical difference from previous work [13] is the introduction of "unsafe" nodes. In [13], 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 required 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". When a safe trajectory ending in a stopped state is added to the unsafe node, the node is marked as safe. This approach uses unsafe nodes as potential connection points for samples, increasing the density of the tree, while ensuring the safety of the vehicle.

Algorithm 2 Execution loop of RRT.

1: **repeat**
2: Update the current vehicle states $\boldsymbol{x}_0$ and environmental constraints $\mathcal{X}_{\mathrm{free}}(t)$
3: Propagate the states by the computation time and obtain $\boldsymbol{x}(t_0 + \Delta t)$
4: **repeat**
5: Expand_tree()
6: **until** time limit Δt is reached
7: Choose the best safe node sequence in the tree
8: **if** No such sequence exists **then**
9: Send E-Stop to controller and goto line 2
10: **end if**
11: Repropagate from the latest states $\boldsymbol{x}(t_0 + \Delta t)$ using the $\boldsymbol{r}$ associated with the best node sequence, and obtain $\boldsymbol{x}(t),\ t \in [t_1, t_2]$
12: **if** $\boldsymbol{x}(t) \in X_{\mathrm{free}}(t)\ \forall t \in [t_1, t_2]$ **then**
13: Send the best reference path $\boldsymbol{r}$ to controller
14: **else**
15: Remove the infeasible portion and its children from the tree, and goto line 7
16: **end if**
17: **until** the vehicle reaches goal

V. Online Replanning

In a dynamic and uncertain environment, the tree needs to keep growing during the execution because of the constant update of the situational awareness. Furthermore, real-time execution requirements necessitate reuse of the information from the previous computation cycle [26]–[28].

Algorithm 2 shows how CL-RRT executes a part of the tree and continues growing the tree while the controller executes

the plan. The planner sends the input to the controller $\boldsymbol{r}$ at a fixed rate of every Δt seconds. The tree expansion continues until a time limit is reached (line 6). The best trajectory is selected and the reference path is sent to the controller for execution (line 13). The expansion of the tree is resumed after updating the vehicle states and the environment (line 2).

Note that when selecting the best trajectory on line 7, only the node sequences that end in a safe state are considered. If none is found, then the planner will command an emergency braking maneuver to the controller in order to bring the car to a stop as fast as possible.

A. Committed part of the tree

A naive way to implement an RRT-based planner is to build a new tree (discarding the old) at every planning cycle, and select the plan for execution independent of the plan currently being executed. If the tree from the previous planning cycle is discarded, almost identical computations would have to be repeated. In a real-time application with limited computing resources, such an inefficiency could result in a relatively sparse tree as compared to the case where available computation are used to add *new* feasible trajectories to the existing tree. Furthermore, if the tree is discarded every cycle and hence the plan for execution is selected independently of the plan being executed, the planner could switch between different trajectories of marginally close cost/utility at every planning cycle, potentially resulting in wavy trajectories.

To address these issues, the CL-RRT algorithm maintains a "committed" trajectory, the end of which coincides with the root node. After the first planning cycle, a feasible plan is sent to the controller for execution. The portion from the root to the next node is marked as committed. For the next planning cycle, this node is initialized as the new root node, and all other children branches from the old root are then deleted because these branches will never be executed. The tree growing phase will then proceed with all subsequent trajectories originating from this new root node. When the propagated vehicle states $\boldsymbol{x}(t_0 + \Delta t)$ (line 3) reach the end of the committed trajectory, the best child node of the root is initialized as the new root, and all other children are deleted. Therefore, the vehicle is always moving towards the root of the tree.

This approach ensures that the tree is maintained from the previous planning cycle, and that the plan that the controller is executing (corresponding to the committed part of the tree) is always continuous. Because the planner does not change its decision over the committed portion, it is important not to commit a long trajectory especially in a dynamic and uncertain environment. The CL-RRT ensures that the committed trajectory is not longer than 1 m, by adding branch points if the best child of the root is farther than that distance.

B. Lazy Reevaluation

In a dynamic and uncertain environment, the feasibility of each trajectory stored in the tree should be re-checked whenever the perceived environment is updated. A large tree, however, could require constant reevaluation of its thousands of edges, reducing the time available for growing the tree.

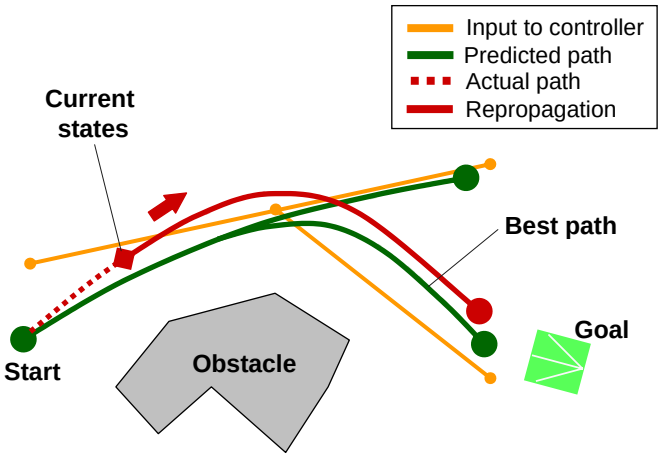


Fig. 5. Repropagation from the current states.

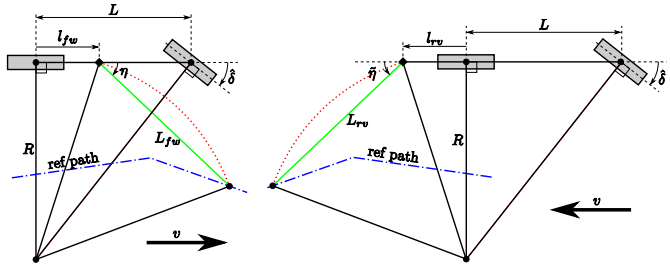
The approach introduced in this paper to overcome this issue was to reevaluate the feasibility of a certain edge only when it is selected as the best trajectory sequence to be executed. If the best trajectory is found infeasible, the infeasible portion of the tree is deleted and the next best sequence is selected for reevaluation. This "lazy reevaluation" enables the algorithm to focus mainly on growing the tree, while ensuring that the executed trajectory is always feasible with respect to the latest perceived environment. The primary difference from previous work [29], [30] 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.

C. Repropagation

Although the feedback loop embedded in the closed-loop prediction can reduce the prediction error, the state prediction could still have non-zero errors due to inherent modeling errors or disturbances. To address the prediction error, one can discard the entire tree and rebuild it from the latest states. However, this is undesirable as highlighted in Section V-A.

The CL-RRT algorithm reuses the controller inputs stored in the tree and performs a *repropagation* from the latest vehicle states, as shown in Figure 5. With a stabilizing controller, the difference between the original prediction and the repropagation would converge to zero if the reference path has a sufficiently long straight line segment.

Instead of repropagating over the entire tree, the CL-RRT algorithm repropagates from the latest states only along the best sequence of nodes (line 11 of Algorithm 1). If the repropagated trajectory is collision free, the corresponding controller input is sent to the controller. Otherwise, the infeasible part of the tree is deleted from the tree, and the next best node sequence is selected and repropagated. This approach requires only a few reevaluations of the edge feasibility, while ensuring the feasibility of the plan being executed regardless of the prediction errors.



(a) Forward Drive (b) Reverse Drive

Fig. 6. Definition of pure-pursuit variables. Two shaded rectangles represent the rear tire and the steerable front tire in the bicycle model. The steering angle in this figure, $\hat{\delta}$, corresponds to negative δ in Eq. (4).

VI. Implementation of the Controller

CL-RRT uses the controller in two different ways. One is in the closed-loop prediction together with the vehicle model, and the other is in the execution of the motion plan in real time. In our implementation, the controller ran at 25 Hz in the execution, and the same time step size was used in the closed-loop simulation. The same controller code was used for both execution and prediction.

A. Steering Controller

The steering controller is based on the pure-pursuit controller, which is a nonlinear path follower that has been widely used in ground robots [31] and more recently in unmanned air vehicles [32]. It is adopted due to its simplicity, as an intuitive control law that has a clear geometric meaning. Because the DUC rules restrict the operating speed to be under 30 mph and extreme maneuvers (e.g., high speed tight turns) are avoided, dynamic effects such as side slip are neglected for control design purposes.

The kinematic bicycle model is described by

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta, \qquad \dot{\theta} = \frac{v}{L}\tan\delta \tag{4}$$

where, x and y refer to the rear axle position, θ is the vehicle heading with respect to the x-axis (positive counter-clockwise), v is the forward speed, δ is the steer angle (positive counter-clockwise), and L is the vehicle wheelbase. With the slip-free assumption, Figure 6(a) shows the definition of the variables that define the pure-pursuit control law when driving forwards, and Figure 6(b) shows the variables when driving in reverse. Here, R is the radius of curvature, and "ref path" is the piecewise linear reference path given by the planner. In Figure 6(a), l_{fw} is the distance of the forward *anchor point* from the rear axle, L_{fw} is the forward drive look-ahead distance, and η is the heading of the look-ahead point (constrained to lie on the reference path) from the forward anchor point with respect to the vehicle heading. Similarly, in Figure 6(b), l_{rv} is the rearward distance of the reverse *anchor point* from the rear axle, L_{rv} is the reverse drive look-ahead distance, and $\tilde{\eta}$ is the heading of the look-ahead point from the reverse anchor point with respect to the vehicle heading offset by π rad. All angles and lengths shown in Figure 6 are positive by definition.

Using elementary planar geometry, the instantaneous steer angles required to put the anchor point on a collision course with the look-ahead point can be computed as

$$\delta = -\tan^{-1}\left(\frac{L\sin\eta}{\frac{L_{\text{fw}}}{2} + l_{\text{fw}}\cos\eta}\right) \quad : \text{forward drive}$$

$$\delta = -\tan^{-1}\left(\frac{L\sin\tilde{\eta}}{\frac{L_{\text{rv}}}{2} + l_{\text{rv}}\cos\tilde{\eta}}\right) \quad : \text{reverse drive}$$

which gives the modified pure-pursuit control law. Note that by setting $l_{\text{fw}} = 0$ and $l_{\text{rv}} = 0$, the anchor points coincide with the rear axle, recovering the conventional pure-pursuit controller [31].

A stability analysis of this controller has been carried out in Ref. [15]. It approximates the slew rate limitation of the steering actuation with a first order system of time constant τ, showing that $L_{\text{fw}} > v\tau - l_{\text{fw}}$ and $L_{\text{rv}} > -v\tau - l_{\text{rv}}$ for Hurwitz Stability. This means that in a high velocity regime, larger L_{fw} is required for stability. Some margin should be added to account for disturbances and modeling error. However, too large L_{fw} degrades the tracking performance. Through extensive simulations and field tests, the following numbers were selected

$$L_{\text{fw}}(v_{\text{cmd}}) = \begin{cases} 3 & \text{if } v_{\text{cmd}} < 1.34, \\ 2.24\, v_{\text{cmd}} & \text{if } 1.34 \le v_{\text{cmd}} < 5.36, \\ 12 & \text{otherwise.} \end{cases}$$

with $\tau = 0.717$. The same rule is also employed for backward driving, i.e., $L_{\text{fw}}(v_{\text{cmd}}) = L_{\text{rv}}(v_{\text{cmd}})$. From this point on, L_1 represents the look-ahead distance L_{fw} and L_{rv}. For the reason elaborated in Subsection VI-C1, L_1 is scaled with the commanded speed v_{cmd} and not the actual speed v.

B. Speed Controller

For speed tracking, a simple Proportional-Integral-Derivative (PID) type controller is considered. However, our assessment is that the PID controller offers no significant advantage over the PI controller because the vehicle has some inherent speed damping and the acceleration signal required for PID control is noisy. Hence, the PI controller of the following form is adopted

$$u = K_p\,(v_{\text{cmd}} - v) + K_i \int_0^t (v_{\text{cmd}} - v)\, d\tau$$

where u is the non-dimensional speed control signal, K_p and K_i are the proportional and integral gain respectively.

The controller coefficients K_p and K_i were determined by extensive testing guided by the parameter space approach of robust control [33]. Given certain design specifications in terms of relative stability, phase margin limits, and robust sensitivity; using the parameter space approach, it is possible to enclose a region in the $K_p - K_i$ plane for which the design specifications are satisfied. Then, via actual testing, controller parameters within this region that best fit the application was identified. As detailed more in [15], considering a low-bandwidth noise-averse control design specification, the following gains have been selected

$$K_p = 0.2, \qquad K_i = 0.04.$$

C. Accounting for the Prediction Error

When the controller does not track the reference path perfectly due to modeling errors or disturbances, the planner could adjust the reference command so that the vehicle achieves the original desired path. However, it introduces an additional feedback loop, potentially destabilizing the overall system. When both the planner and the controller try to correct for the same error, they could be overcompensating or negating the effects of the other.

In our approach, the planner generates a "large" signal in the form of the reference path, but it does not adjust the path due to the tracking error. It is controller's responsibility to track the path in a "small" signal sense. Thus, the propagation of trajectories (line 5 of Algorithm 2) starts from the predicted vehicle states at the node. One challenge here is that the collision is checked with the predicted trajectory, which can be as long as several seconds. Although Subsection V-C addressed the case when the prediction error becomes large, it is still critical to keep the prediction error small in order to have acceptable performance. This section presents several techniques to reduce prediction errors.

1) Use of v_{cmd} for L_1 scheduling: As discussed in Section VI-A, the look-ahead distance L_1 must increase with the vehicle speed to maintain stability. However, scheduling L_1 based on the speed v means that any prediction error in the speed directly translates into a discrepancy of the steer command between prediction and execution, introducing a lateral prediction error. This is problematic because achieving a small speed prediction error is very challenging especially during the transient in the low-speed regime, where the engine dynamics and gear shifting of the automatic transmission exhibits complicated nonlinearities. Another disadvantage of scaling L_1 with v is that the noise in the speed estimate introduces jitters in the steering command.

A solution to this problem is to use the speed command v_{cmd} to schedule the L_1 distance. The speed command is designed by the planner, uncorrupted by noise or disturbances, so the planner and controller have the same v_{cmd} with no ambiguity. Then, the same steering gain L_1 is used in the prediction and execution, rendering the steering prediction decoupled from the speed prediction.

2) Space-dependent speed command: In Section III, the reference command $\boldsymbol{r}$ was defined as a function of time. For the speed command v_{cmd}, however, it is advantageous to associate it to the position of the vehicle.

When the speed command is defined as a function of time, the prediction error in the speed could affect the steering performance, even if the look-ahead distance L_1 is scaled based on v_{cmd}. Suppose the vehicle accelerates from rest with a ramp command. If the actual vehicle accelerates more slowly than predicted, the time-based v_{cmd} would increase faster than the prediction with respect to the travel distance. This means that L_1 increases more in the execution before the vehicle moves much, and given a vehicle location, the controller places a look-ahead point farther down on the reference path compared to prediction.

To overcome this issue, the speed command is associated with the position of the vehicle with respect to the predicted path. Then, the steering command depends only on where the vehicle is, and is insensitive to speed prediction errors.

VII. Application Results

This section presents the application results of CL-RRT algorithm on MIT's DUC entry vehicle, Talos.

A. Vehicle Model

The following nonlinear model is used in the prediction.

$$\begin{aligned} \dot{x} &= v\cos\theta & \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{v}{L}\tan\delta\cdot G_{\text{ss}} & \dot{\delta} &= \frac{1}{T_d}(\delta_c-\delta) \\ \dot{v} &= a & \dot{a} &= \frac{1}{T_a}(a_c-a) \\ a_{\min} \le a \le a_{\max} & & & \\ \|\delta\| \le \delta_{\max} & & \|\dot{\delta}\| \le \dot{\delta}_{\max} & \end{aligned}$$

The inputs to this system are the steering angle command δ_c and the longitudinal acceleration command a_c. These inputs go through a first-order lag with time constants T_d and T_a for the steering and acceleration respectively. The maximum steering angle is given by $\delta_{\max}$, and the maximum slew rate for the steering is $\dot{\delta}_{\max}$. The forward acceleration is denoted by a, with the bounds of maximum deceleration $a_{\min}$ and maximum acceleration $a_{\max}$.

The term G_{ss}, which did not appear in Eq. (4), captures the effect of side slip

$$G_{\text{ss}} = \frac{1}{1+(v/v_{\text{CH}})^2}$$

and is a static gain of the yaw rate $\dot{\theta}$, i.e. the resulting yaw rate when the derivative of the side-slip angle and the derivative of yaw rate are both zero [33]. The parameter v_{CH} is called the characteristic velocity [33], [34] and can be determined experimentally. This side slip model has several advantages: it does not increase the model order, and hence the computation required for propagation; the model behaves the same as the kinematic model in the low speed regime; and it requires only one parameter to tune. Moreover, this model does not differ much from the nonlinear single track model [33] for the urban driving conditions where extreme maneuvers are avoided and up to 30 mph speeds are considered.

The following parameters were used for Talos, a 4.9-meter-long and 2.0-meter-wide Land Lover LR3.

$$\begin{aligned} \delta_{\max} &= 0.5435 \text{ rad} & \dot{\delta}_{\max} &= 0.3294 \text{ rad/s} \\ T_d &= 0.3 \text{ s} & T_a &= 0.3 \text{ s} \\ a_{\min} &= -6.0 \text{ m/s}^2 & a_{\max} &= 1.8 \text{ m/s}^2 \\ L &= 2.885 \text{ m} & v_{\text{CH}} &= 20.0 \text{ m/s} \end{aligned}$$

From the parameters above, the minimum turning radius is found to be 4.77 m.

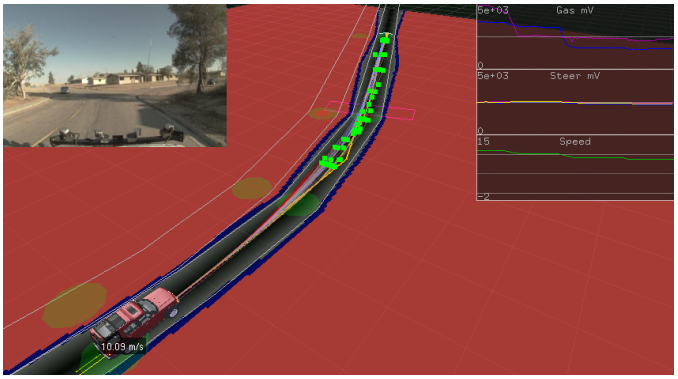


Fig. 7. Lane following on a high speed curvy section. The vehicle speed is 10 m/s. The green dots show the safe stopping nodes in the tree.

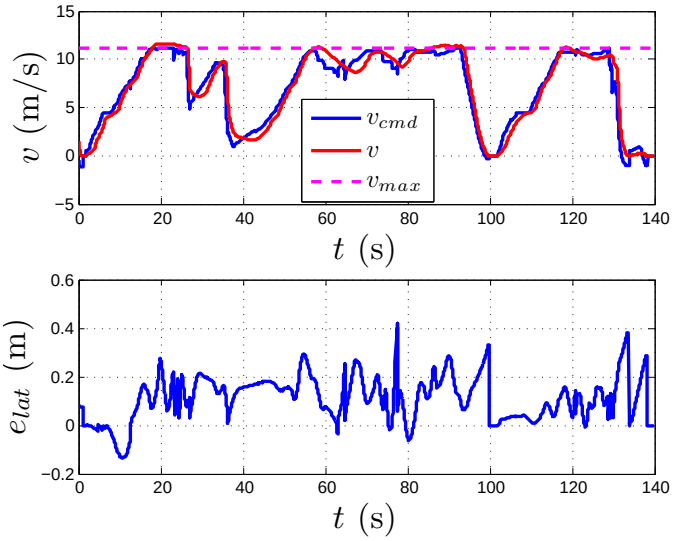


Fig. 8. Prediction error during this segment

B. Race Results

The following subsections present results from the National Qualification Event (NQE) and the final Urban Challenge Event (UCE). NQE consisted of three test areas A, B, and C, each focusing on testing different capabilities. During NQE, the CL-RRT algorithm was not tuned to any specific test area, showing the generality of the approach. UCE consisted of three missions, with a total length of about 60 miles. Talos was one of the six vehicles that completed all missions, finishing in 5 hours 35 minutes.

In Talos, the motion planner was executed on a dual-core 2.33 GHz Intel Xeon processor at approximately 10 Hz. The time limit Δt in Algorithm 2 is 0.1 second and the algorithm uses 100% CPU by design. The average number of samples generated was approximately 700 samples per second and the tree had about 1200 nodes on average. Notice that because the controller input is the parameter that is sampled, a single sample could create a trajectory as long as several seconds.

1) High speed behavior on a curvy road: Figure 7 shows a snapshot of the environment and the plan during UCE. The vehicle is in the lower left, going towards a goal in the upper middle of the figure. The small green squares

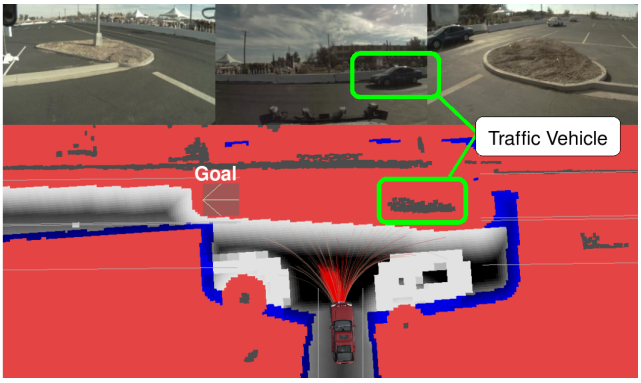


Fig. 9. Tree consisting of many unsafe nodes in the merge test.

represent the safe stopping nodes in the tree. The vehicle is moving at 10 m/s, so there are no stopping nodes in the close range. However, the planner ensures there are numerous stopping points on the way to the goal, should intermittently detected curbs or vehicles appear. Observe that even though the controller inputs are randomly generated to build the tree, the resulting trajectories naturally follow the curvy road. This road segment is about 0.5 mile long, and the speed limit specified by DARPA was 25 mph. Figure 8 shows the speed profile and the lateral prediction error for this segment. Talos reached the maximum speed several times on straight segments, while slowing down on curvy roads to observe the maximum lateral acceleration constraints. The prediction versus execution error has the mean, maximum, and standard deviation of 0.11 m, 0.42 m, and 0.093 m respectively. Note from the plot that the prediction error has a constant offset of about 11 cm, making the maximum error much larger than the standard deviation. This is due to the fact that the steering wheel was not perfectly centered and the pure-pursuit algorithm does not have any integral action to remove the steady state error.

Note that when the prediction error happens to become large, the planner does not explicitly minimize it. This occurs because the vehicle keeps executing the same plan as long as the repropagated trajectory is feasible. In such a scenario, the prediction error could grow momentarily. For example, during a turn with a maximum steering angle, a small difference between the predicted initial heading and the actual heading can lead to a relatively large error as the vehicle turns. Even with a large mismatch, however, the repropagation process in Section V-C ensures the safety of the future path from vehicle's current state.

2) Unsafe Nodes in the Dynamic Environment: Figure 9 is a snapshot from the merging test in NQE. Talos is in the bottom of the figure, trying to turn left into the lane and merge into the traffic. The red lines originating from Talos show the unsafe trajectories, which do not end in a stopped state.

Before the traffic vehicle comes close to the intersection, there were many trajectories that reach the goal. However, as the traffic vehicle (marked with a green rectangle) approached, its path propagated into the future blocks the goal of Talos, as shown in Figure 9, which rendered the end parts of these trajectories infeasible. However, the feasible portion of these trajectories remain in the tree as unsafe nodes (see

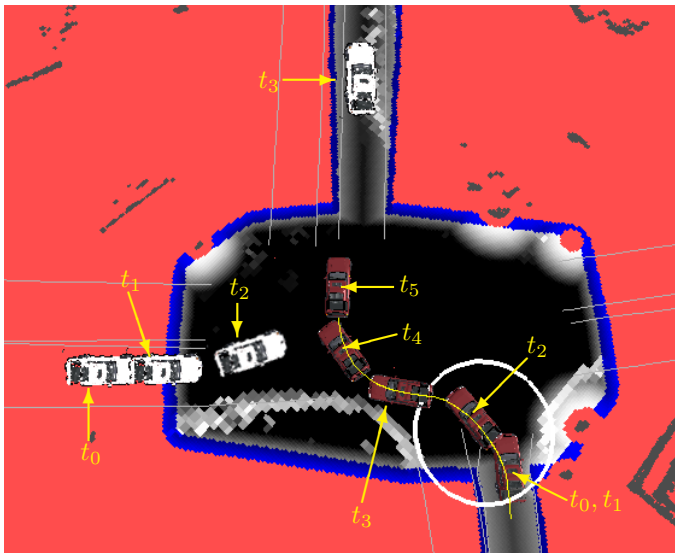


Fig. 12. Sequence of events. Talos (maroon) remains stationary between t_0 and t_1, and Odin (white) had progressed beyond the view window by t_4.

Section IV-E2), from which the tree is quickly grown to the goal once the traffic vehicle leaves the intersection.

3) Parking: Figure 10 shows Talos' parking maneuver inside the zone during NQE. Figure 10(a) shows a superimposed sequence of Talos' pose for the parking maneuver. Figure 10(b) shows a tree of trajectories before going into the parking spot. The purple lines show forward trajectories reaching the goal; the light brown lines show forward trajectories that do not reach the goal; the turquoise lines show reverse trajectories; and the red lines show unsafe trajectories. The motion plan is marked with two colors: the orange line is the input to the steering controller; and the green line is the predicted trajectory on which the speed command is associated.

In Figure 10(c), Talos is reversing to go out of the spot and intends to exit the parking zone. The goal for the planner is now at the exit of the zone located in the upper right corner of the figure, and there are many trajectories that go towards it. Following the plan selected from these trajectories, Talos successfully exited the parking zone.

4) Road Blockage: Figure 11 shows the behavior of Talos during a U-turn that was required in the Area C test. The motion planner always tries to make forward progress, but after 50 seconds of no progress, the Navigator declares a blockage and places the goal in the lane with the opposite travel direction. Figure 11(a) shows a sequence of Talos's pose during the maneuver. The road to the right is blocked by a number of traffic barrels. In Figure 11(b), it can be seen that there are several trajectories reaching the goal in the destination lane. Figure 11(c) shows the last segment of the U-turn. Because of the newly detected curbs in front of the vehicle, the maneuverable space has becomes smaller than in (b), resulting in a 5-pt turn.

5) Interaction with Virginia Tech's Odin: Figures 12 and 13 show an interaction that occurred between Talos and Virginia Tech's Odin in the UCE, with the main result being that Talos took an evasive swerving maneuver to avoid a collision.

Initially, Talos arrived at the intersection and stopped at the stop line (Figure 13(a)). Odin is approaching the intersection from the left, and its predicted trajectory goes into the intersection. Although Odin did not have a stop line on its lane, it stopped when it reached the intersection and remained motionless for about 3 seconds. With the intersection clear, and all vehicles stopped, Talos assumed precedence over Odin and started driving through the intersection (Figure 13(b)). Just as Talos started, however, Odin also started driving, heading towards the same lane that Talos was heading, and blocked Talos' path (Figure 13(c)). Talos took an evasive maneuver to avoid collision and stop safely by steering left and applying strong brake (white circle around Talos). Odin continued along its path, and once it (and the trailing human-occupied DARPA chase vehicle) cleared the intersection, Talos recomputes a new plan and proceeds through the intersection (Figure 13(d)).

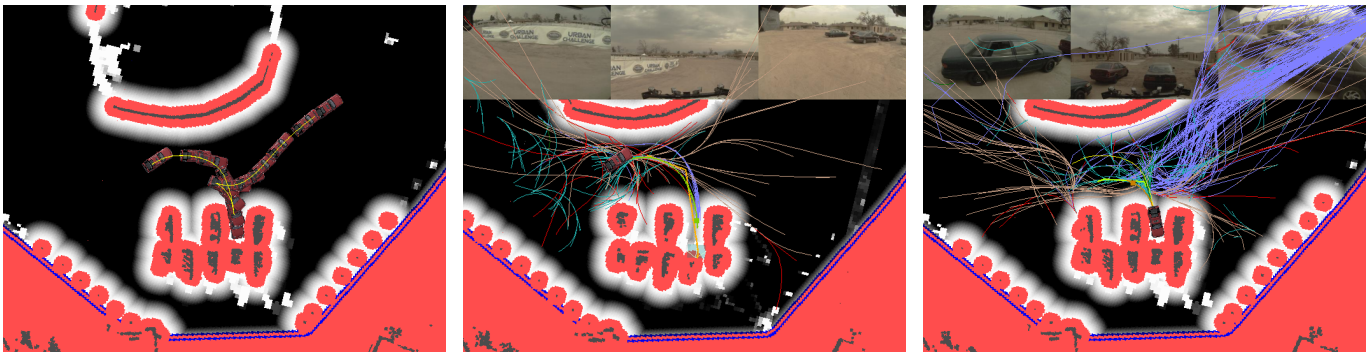
This is just one example of the numerous traffic and intersection scenarios that had never been tested before the race, and yet the motion planner demonstrated that it was capable of handling these situations safely. While Talos has demonstrated that it is generally operating safely even in untested scenarios, it was nonetheless involved in two crash incidents during the UCE [3]. However, the analysis in [35] indicates that the primary contributing factor in those two instances was an inability to accurately detect slow moving vehicles.

6) Repropagation: Most of the time during the race, the trajectory repropagated from the latest states and the trajectory stored in the tree are close. However, there are some instances when the repropagation resulted in a different trajectory. In Figure 13(d), Talos slowed down more than the prediction, so the trajectory repropagated from the latest states (the pink line shown with an arrow in the figure) is a little off from the trajectories stored in the tree. However, as long as the repropagation is feasible, Talos keeps executing the same controller input. This approach is much more efficient than discarding the tree when the prediction error exceeds an artificial limit and rebuilding the tree.

VIII. Conclusion

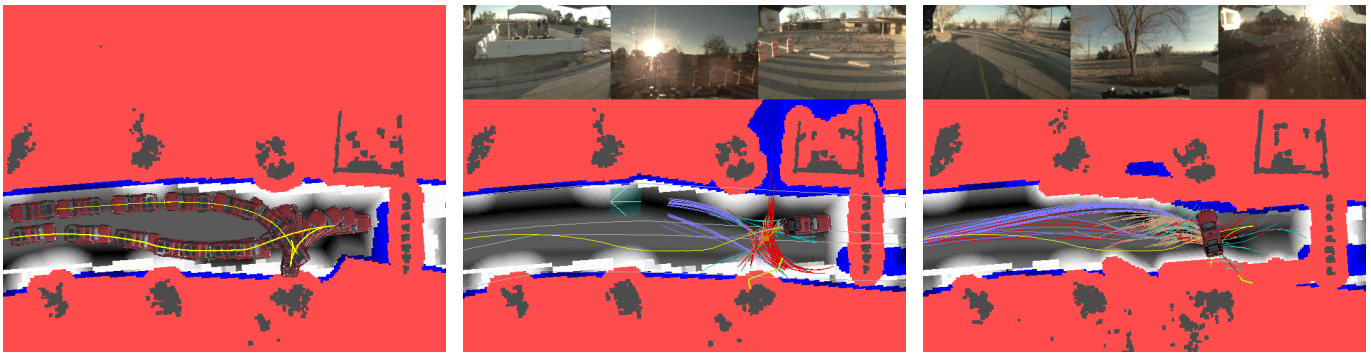
This paper presented the CL-RRT algorithm, a sampling-based motion planning algorithm specifically developed for large robotic vehicles with complex dynamics and operating in uncertain, dynamic environments such as urban areas. The algorithm was implemented as the motion planner for Talos, the autonomous Land Rover LR3 that was MIT's entry in the 2007 DARPA Urban Challenge.

The primary novelty is the use of closed-loop prediction in the framework of RRT. The combination of stabilizing controller and forward simulation has enabled application of CL-RRT to vehicles with complex, nonlinear, and unstable dynamics. Several extensions are presented regarding RRT tree expansion: a simple sampling bias strategy to generate various different maneuvers; penalty value in addition to the binary check of edge feasibility; use of Dubins path length when selecting the node to connect; and guaranteed safety even when



(a) Overall trajectory of Talos. (b) Going towards the parking spot. (c) After parked, Talos reversing to its left and going to the exit.

Fig. 10. Parking maneuver in the parking zone in an Area B test.



(a) Overall trajectory of Talos. (b) Talos initiated first segment of U-turn. (c) Talos starting last segment of U-turn.

Fig. 11. U-turn maneuver at a blocked road in an Area C test.

the vehicle is in motion. By retaining the tree during execution, the CL-RRT reuses the information from the previous computation cycle, which is important especially in real-time applications. The extensions made to online replanning include the lazy reevaluation to account for changing environments and the repropagation to account for prediction errors.

The advantages of the CL-RRT algorithm were demonstrated through the race results from DUC. The successful completion of the race with numerous differing scenarios using this single planning algorithm demonstrates the generality and flexibility of the CL-RRT planning algorithm.

ACKNOWLEDGMENT

Research was sponsored by Defense Advanced Research Projects Agency, Program: Urban Challenge, DARPA Order No. W369/00, Program Code: DIRO. Issued by DARPA/CMO under Contract No. HR0011-06-C-0149, with Prof. John Leonard, Prof. Seth Teller, Prof. Jonathan How at MIT and Prof. David Barrett at Olin College as the principal investigators. The authors gratefully acknowledge the support of the MIT DARPA Urban Challenge Team, with particular thanks to Dr. Luke Fletcher and Dr. Edwin Olson for their development of the drivability map, David Moore for developing the Navigator, Dr. Karl Iagnemma for his expert advice, Stefan Campbell for his initial design and implementation of the controller, and Steven Peters for his technical support during the development of Team MIT's vehicle.

REFERENCES

[1] K. Iagnemma and M. Buehler, eds., "Special issue on the DARPA grand challenge, part 1," *Journal of Field Robotics*, vol. 23, no. 8, pp. 461–652, Aug. 2006.

[2] ——, "Special issue on the 2007 DARPA urban challenge," *Journal of Field Robotics*, vol. 25, pp. 423–860, 2008.

[3] 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. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, "A perception driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

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

[5] J.-P. Laumond, Ed., *Robot Motion Planning and Control*. Berlin: Springer-Verlag, 1998, available online at http://www.laas.fr/~jpl/book.html.

[6] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

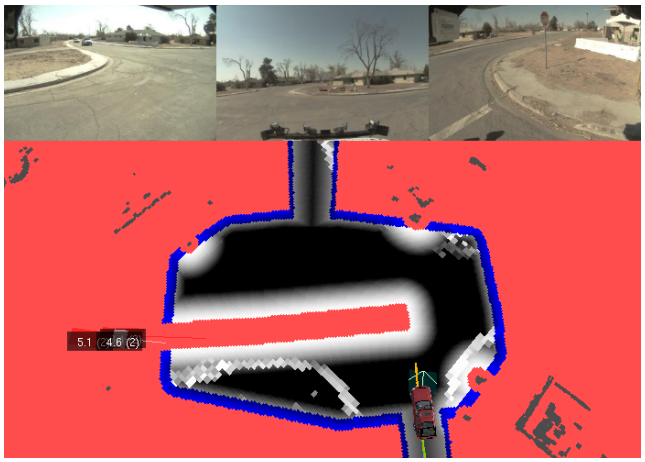
[7] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[8] K. Iagnemma and M. Buehler, eds., "Special issue: Special issue on the DARPA grand challenge, part 2," *Journal of Field Robotics*, vol. 23, no. 9, pp. 655–835, Sept. 2006.

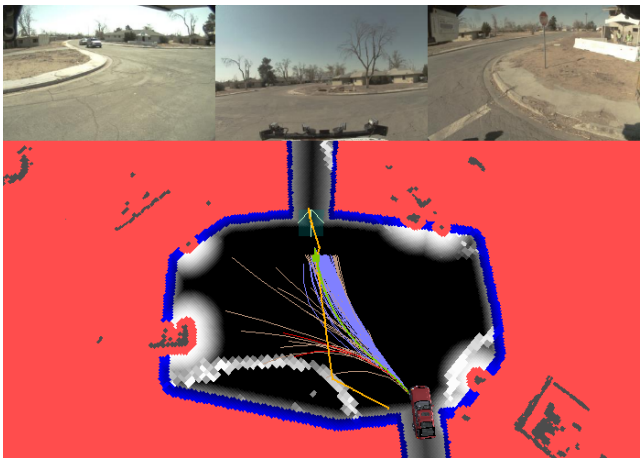
[9] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proceedings IEEE International Conference on Robotics and Automation*, 1999, pp. 473–479.

[10] ——, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

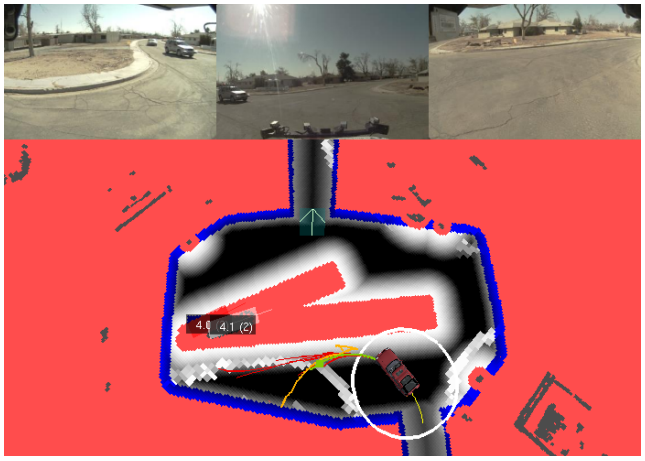
[11] ——, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. Wellesley, MA: A K Peters, 2001, pp. 293–308.



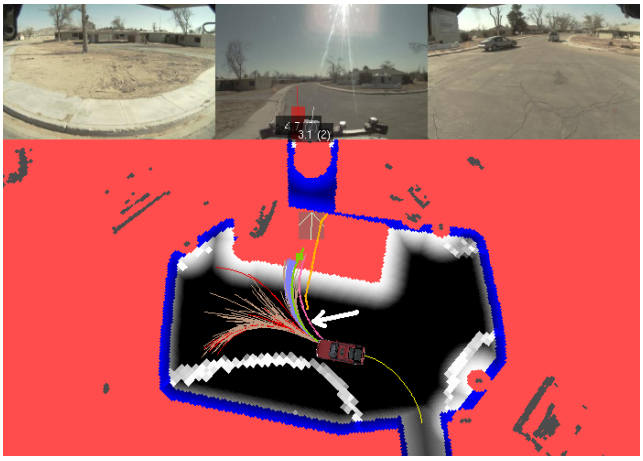
(a) Odin approaches the intersection, Talos stopped at t_0.



(b) Odin stops at intersection, Talos starts maneuver at t_1.



(c) Odin moves into intersection, Talos detected potential collision and executes avoidance maneuver at t_2.



(d) Odin clears intersection, Talos proceeds through intersection at t_3.

Fig. 13. Talos successfully avoids Virginia Tech's Odin at an intersection.

[12] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal Computational Geometry & Applications*, vol. 4, pp. 495–512, 1999.

[13] 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.

[14] 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.

[15] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. P. How, "Motion planning in complex environments using closed-loop prediction," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, HI, Aug. 2008, AIAA–2008–7166.

[16] S. R. Lindemann and S. M. LaValle, "Incrementally reducing dispersion by increasing Voronoi bias in RRTs," in *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[17] ——, "Steps toward derandomizing RRTs," in *IEEE Fourth International Workshop on Robot Motion and Control*, 2004.

[18] M. Strandberg, "Augmenting RRT-planners with local trees," in *Proceedings IEEE International Conference on Robotics & Automation*, 2004, pp. 3258–3262.

[19] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[20] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *Proceedings IEEE International Conference on Robotics and Automation*, 2005.

[21] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.

[22] A. M. Shkel and V. Lumelsky, "Classification of the dubins set," *Robotics and Autonomous Systems*, vol. 34, pp. 179–202, Mar. 2001.

[23] 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.

[24] P. Indyk, "Nearest neighbors in high-dimensional spaces," in *Handbook of Discrete and Computational Geometry, 2nd Ed.*, J. E. Goodman and J. O'Rourke, Eds. New York: Chapman and Hall/CRC Press, 2004, pp. 877–892.

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

[26] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings IEEE International Conference on Robotics & Automation*, 1994, pp. 3310–3317.

[27] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Proceedings IEEE International Conference on Robotics & Automation*, 2005.

[28] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2007.

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

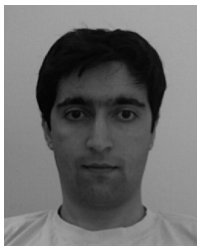
[30] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Proceedings International Symposium on Robotics Research*, 2001.

[31] O. Amidi and C. Thorpe, "Integrated Mobile Robot Control," in *Proceedings of SPIE*, W. H. Chun and W. J. Wolfe, Eds., vol. 1388. Boston, MA: SPIE, Mar. 1991, pp. 504–523.
[32] S. Park, J. Deyst, and J. P. How, "Performance and Lyapunov Stability of a Nonlinear Path-Following Guidance Method," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 6, pp. 1718–1728, Nov. 2007.
[33] J. Ackermann, *Robust Control: The Parameter Space Approach*. Springer, 2002.
[34] T. N. Gillespie, *Fundamentals of Vehicle Dynamics*. Warrendale, PA: Society of Automotive Engineers, 1992.
[35] L. Fletcher, S. Teller, E. Olson, D. Moore, Y. Kuwata, J. How, J. Leonard, I. Miller, M. Campbell, D. Huttenlocher, A. Nathan, and F.-R. Kline, "The MIT – Cornel collision and why it happened," *Journal of Field Robotics*, vol. 25, no. 10, pp. 775–807, 2008.

Yoshiaki Kuwata received the B.Eng degree from the University of Tokyo in 2001, and the S.M. and Ph.D. degrees in Aeronautics and Astronautics from Massachusetts Institute of Technology (MIT) in 2003 and 2007, respectively. In 2007, he worked on the DARPA Urban Challenge as a postdoctoral associate at MIT. He then joined the Robotics Section of Jet Propulsion Laboratory, California Institute of Technology in 2008. His current research interests include cooperative control, vision-based guidance, and path planning for unmanned aerial/ground/sea-surface vehicles.

Justin Teo received the B.Eng degree from Nanyang Technological University, Singapore, in 1999, and the M.Sc degree from National University of Singapore in 2003. He joined DSO National Laboratories, Singapore, in 1999. Since 2005, DSO National Laboratories has funded his Ph.D. studies in the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology. His research interests include nonlinear control, adaptive control, and trajectory planning.

Gaston Fiore received the B.S. degree in Aerospace Engineering with Information Technology and an M.S. degree in Aeronautics and Astronautics, both from the Massachusetts Institute of Technology, in 2006 and 2008 respectively. He is currently a Ph.D. candidate in Computer Science at the Harvard School of Engineering and Applied Sciences. His current research interests include amorphous computing, computational learning, and computational neuroscience.



Sertac Karaman received B.S. degrees from the Istanbul Technical University in Mechanical Engineering and Computer Engineering in 2006 and 2007, respectively. He is currently an S.M. candidate in the Department of Mechanical Engineering at Massachusetts Institute of Technology. His research interests include optimal scheduling, real-time motion planning, and formal languages.

Emilio Frazzoli (S'99–M'01–SM'08) is an Associate Professor in the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology. He received the Laurea degree in Aerospace Engineering from the University of Rome La Sapienza, Rome, Italy, in 1994, and the Ph.D. degree in Navigation and Control Systems from the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, in 2001. From 2001 to 2004, he was an Assistant Professor of Aerospace Engineering at the University of Illinois at Urbana-Champaign. From 2004 to 2006, he was an Assistant Professor of Mechanical and Aerospace Engineering at the University of California, Los Angeles. His current research interests include control of autonomous cyber-physical systems, guidance and control of agile vehicles, mobile robotic networks, and high-confidence embedded systems. Dr. Frazzoli received the National Science Foundation (NSF) CAREER Award in 2002.

Jonathan P. How is a Professor in the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology (MIT). He received a B.A.Sc. from the University of Toronto in 1987 and his S.M. and Ph.D. in Aeronautics and Astronautics from MIT in 1990 and 1993, respectively. He then studied for two years at MIT as a postdoctoral associate for the Middeck Active Control Experiment (MACE) that flew on-board the Space Shuttle Endeavour in March 1995. Prior to joining MIT in 2000, he was an Assistant Professor in the Department of Aeronautics and Astronautics at Stanford University. Current research interests include robust coordination and control of autonomous vehicles in dynamic uncertain environments. He was the recipient of the 2002 Institute of Navigation Burka Award, is an Associate Fellow of AIAA, and a senior member of IEEE.