Bounds on Tracking Error using Closed-Loop Rapidly-Exploring Random Trees

Brandon D. Luders, Sertac Karaman, Emilio Frazzoli, and Jonathan P. How

Abstract— **This paper considers the real-time motion planning problem for autonomous systems subject to complex dynamics, constraints, and uncertainty. Rapidly-exploring random trees (RRT) can be used to efficiently construct trees of dynamically feasible trajectories; however, to ensure feasibility, it is critical that the system actually track its predicted trajectory. This paper shows that under certain assumptions, the recently proposed closed-loop RRT (CL-RRT) algorithm can be used to accurately track a trajectory with known error bounds and robust feasibility guarantees, without the need for replanning. Unlike open-loop approaches, bounds can be designed on the maximum prediction error for a known uncertainty distribution. Using the property that a stabilized linear system subject to bounded process noise has BIBO-stable error dynamics, this paper shows how to modify the problem constraints to ensure long-term feasibility under uncertainty. Simulation results are provided to demonstrate the effectiveness of the closed-loop RRT approach compared to open-loop alternatives.**

I. Introduction

Rapidly-exploring random trees (RRT) have received increased attention over the past decade as a sampling-based approach to quickly identify feasible trajectories in complex motion planning problems [1,2]. Even though several approaches have been proposed to solve general motion planning problems [3]–[5], approaches that incorporate randomization or incremental sampling have demonstrated several advantages for real-time motion planning, including: trajectory-wise (*e.g.* non-enumerative) checking of possibly complex constraints; applicability to general dynamical models; and incremental construction, facilitating use in on-line implementations. Compared to other incremental sampling-based methods [6]–[8], the RRT algorithm is particularly useful for real-time motion planning of complex dynamical systems in cluttered, high-dimensional configuration spaces.

For practical real-time implementations of RRTs, it is critical that the predicted trajectories selected for execution be accurately tracked by the system. Since various feasibility conditions are checked based on the predicted trajectory, significant deviations may result in dynamical infeasibility and/or infeasibility due to collisions with obstacles. If deviations grow large enough, it may be necessary to re-initialize the tree to the system's present state, resulting in the loss of potentially useful planning computations. This problem is more complicated for systems that are unstable, inaccurately modeled, or subject to external disturbances.

The closed-loop RRT algorithm (CL-RRT) has been proposed to address these problems for autonomous vehicles [9]–[11]. The algorithm was successfully demonstrated as the motion planning subsystem for Team MIT's vehicle, *Talos*, at the 2007 DARPA Urban Challenge (DUC) [12]. The CL-RRT algorithm adds a path-tracking control loop in the system's RRT prediction model, such that RRT sampling takes place in the reference input space rather than in the vehicle input space. If the system executes a chosen path using the same prediction model, any deviations are propagated using the same closed-loop dynamics, resulting in more accurate tracking than with open-loop prediction [13].

A common "hybrid" approach is to design the trajectory in open-loop, and then close a loop on the actual system in executing this path. However, without also incorporating this loop closure in the prediction model, this paper shows that the mismatch may still be significant. Deviations can also be mitigated through periodic replanning, but each attempt to do so requires solving a new instance of the planning problem, undesirable in real-time applications with limited computational resources.

While Ref. [11] presents the overall CL-RRT framework and DUC results, this paper provides a more in-depth analysis of the properties of the algorithm. In particular it is shown that, under certain assumptions, CL-RRT can be used to accurately track a trajectory with known error bounds and robust feasibility guarantees, without the need for replanning. These properties are established by considering the evolution of the deviation between the predicted and actual trajectory. Through appropriate choice of reference model and input controller, bounds can be designed for the maximum prediction error given a known uncertainty distribution. Using the property that a stabilized linear system subject to bounded process noise has BIBO-stable error dynamics, this paper also shows that it is possible to modify the problem constraints to ensure long-term robust feasibility. Simulation results demonstrate the effectiveness of closed-loop RRT compared to open-loop alternatives.

II. Problem Statement

Consider the uncertain, nonlinear, discrete-time dynamical system

$$x_{t+1} = f(x_t, u_t, w_t), \tag{1}$$

where $x_t \in \mathbb{R}^{n_x}$ is the state vector, $u_t \in \mathbb{R}^{n_u}$ is the input vector, $w_t \in \mathbb{R}^{n_w}$ is a disturbance vector acting on the system,

Research Assistant, Dept. of Aeronautics and Astronautics, MIT, Member IEEE, `luders@mit.edu`

Research Assistant, Dept. of Electrical Engineering and Computer Science, MIT, Member IEEE, `sertac@mit.edu`

Professor, Dept. of Aeronautics and Astronautics, MIT, Senior Member IEEE, `frazzoli@mit.edu`

Professor, Dept. of Aeronautics and Astronautics, MIT, Senior Member IEEE, `jhow@mit.edu`

and $x_0 \in \mathbb{R}^{n_x}$ is the known initial state at time $t = 0$. It is assumed that the full state is available at all time steps. The disturbance w_t is unknown at current and future time steps, but has a known probability distribution $w_t \sim P(W)$. Each individual disturbance realization is assumed to be independent. This disturbance may represent several possible sources of uncertainty acting on the dynamics, such as modeling errors or external forces. The system is subject to state and input constraints, assumed decoupled,

$$x_t \in \mathscr{X}_t, \quad \text{and} \quad u_t \in \mathscr{U}. \tag{2}$$

The time dependence of $\mathscr{X}$ allows the inclusion of both static and dynamic state constraints.

The primary objective of the planning problem, and the focus of this paper, is to identify a path which reaches the goal region $\mathscr{X}_{\text{goal}} \subset \mathbb{R}^{n_x}$ while satisfying the constraints (2) for all time steps. The secondary objective, given that at least one feasible path has been found, is to identify a path which minimizes some cost function. This cost function may include minimum-time, minimum-fuel, obstacle avoidance, or other objectives [11].

III. Rapidly-exploring Random Trees

This section summarizes the RRT algorithm and its extension to the closed-loop RRT algorithm [11]. The fundamental operation in the standard RRT algorithm is the incremental growth of a tree of dynamically feasible trajectories, rooted at the system's current state x_t. A node's likelihood of being selected to grow the tree is proportional to its Voronoi region for a uniform sampling distribution. As a result, the RRT algorithm is naturally biased toward rapid exploration of the state space.

To grow a tree of dynamically feasible trajectories, it is necessary for the RRT to have an accurate model of the vehicle dynamics (1) for simulation. This model is assumed to be a disturbance-free (*e.g.* $w \equiv 0$) form of the true dynamics,

$$\begin{aligned} \hat{x}_{t+k+1|t} &= f(\hat{x}_{t+k|t}, \hat{u}_{t+k|t}, 0), \quad (3)\\ \hat{x}_{t|t} &= x_t, \quad (4) \end{aligned}$$

where t is the current system timestep, $\hat{x}_{t+k|t}$ is the predicted state at timestep $t+k$, and $\hat{u}_{t+k|t}$ is the input applied at timestep $t+k$. Because of w_t in the true dynamics (1), there may be a significant discrepancy between the actual state x_{t+k} and the prediction $\hat{x}_{t+k|t}$.

This paper considers the real-time RRT algorithm proposed in [9] and later extended in [11,13,14], using both open-loop and closed-loop control. The algorithm grows a tree of feasible trajectories originating from the current state that attempt to reach the goal region $\mathscr{X}_{\text{goal}}$. At the end of each phase of tree growth, the best feasible trajectory is selected for execution, and the process repeats. The two primary operations of the algorithm, tree expansion and the execution loop, are reviewed next.

Algorithm 1. Tree Expansion

```
    Inputs: Tree $\mathscr{T}$, goal region $\mathscr{X}_{\text{goal}}$, current timestep $t$
1:  Take a sample $x_{\text{samp}}$ from the environment
2:  Identify the $M$ nearest nodes using heuristics
3:  for $m \leq M$ nearest nodes, in the sorted order do
4:      $N_{near} \leftarrow$ current node, $\hat{x}_{t+k|t} \leftarrow$ final state of $N_{near}$
5:      while $\hat{x}_{t+k|t} \in \mathscr{X}_{t+k}$ and $\hat{x}_{t+k|t}$ has not reached $x_{\text{samp}}$ do
6:          Select input $\hat{u}_{t+k|t} \in \mathscr{U}$
7:          Simulate $\hat{x}_{t+k+1|t}$ using (3)
8:          Create intermediate nodes as appropriate
9:          $k \leftarrow k+1$
10:     end while
11:     for each feasible node $N$ do
12:         Update cost estimates for $N$
13:         Add $N$ to $\mathscr{T}$
14:         Try connecting $N$ to $\mathscr{X}_{\text{goal}}$ (lines 4-10)
15:         if connection to $\mathscr{X}_{\text{goal}}$ feasible do
16:             Update upper-bound cost-to-go of $N$ and ancestors
17:         end if
18:     end for
19: end for
```

A. Tree Expansion

The tree expansion algorithm, used to grow the tree, is given in Algorithm 1. Similar to the basic RRT algorithm [1], Algorithm 1 takes a sample (line 1), identifies nearest nodes to connect to it (line 2), and attempts to form the connection by generating a feasible trajectory (lines 5–10). The method of selecting inputs to simulate trajectories (line 7) depends on whether open-loop RRT or closed-loop RRT is being used, and is discussed in detail in Section III-C.

A number of heuristics have been utilized to facilitate tree growth, identify a feasible trajectory to the goal, and identify "better" feasible paths once at least one has been found (Section II). Samples are identified (line 1) by probabilistically choosing between a variety of global and local sampling strategies, some of which may be used to efficiently generate complex maneuvers [11,14]. The nearest node selection scheme (lines 2-3) strategically alternates between several distance metrics for sorting the nodes, including an exploration metric based on cost-to-go and a path optimization metric based on estimated total path length [9]. Each time a sample is generated, $m \geq 1$ attempts are made to connect a node to this sample before being discarded [11]. Intermediate nodes are occasionally inserted during the trajectory generation process (line 8) to increase the number of possible connection points and allow partial trajectories to be added to the tree [9]. Since the primary objective is to quickly find a feasible path to the goal, an attempt is made to connect newly-added nodes directly to $\mathscr{X}_{\text{goal}}$ (line 14). Finally, a branch-and-bound cost scheme is used (line 16) to prune portions of the tree whose lower-bound cost-to-go is larger than the upper-bound cost-to-go of an ancestor, since those portions have no chance of achieving a lower-cost path [9].

B. Execution Loop

For environments which are dynamic and uncertain, the RRT tree must keep growing during the execution cycle to account for changes in the situational awareness [9]. Furthermore, given the extensive computations involved to construct the tree, as much of the tree should be retained as

Algorithm 2. Real-time RRT

```
    Inputs: Initial state $x_I$, goal region $\mathscr{X}_{\text{goal}}$
1:  $t \leftarrow 0$, $x_t \leftarrow x_I$
2:  Initialize tree $\mathscr{T}$ with node at $x_I$
3:  while $x_t \notin \mathscr{X}_{\text{goal}}$ do
4:      Update current state $x_t$ and constraints $\mathscr{X}_t$
5:      Propagate the state $x_t$ by the computation time $\rightarrow \hat{x}_{t+\Delta t|t}$
        using (3)
6:      while time remaining for this timestep do
7:          Expand the tree by adding nodes (Algorithm 1)
8:      end while
9:      Use cost estimates to identify best path $\{N_{\text{root}}, \ldots, N_{\text{target}}\}$
10:     if no paths exist then
11:         Apply safety action and goto line 19
12:     end if
13:     Repropagate the best path from $\hat{x}_{t+\Delta t|t}$ using (3)
14:     if repropagated best path is feasible then
15:         Apply best path
16:     else
17:         Remove infeasible portion of best path and goto line 9
18:     end if
19:     $t \leftarrow t + \Delta t$
20: end while
```

possible, especially in real-time applications [15]. Algorithm 2 shows how the algorithm executes some portion of the tree while continuing to grow it.

The planner updates the current best path to be executed by the system every Δt seconds. During each cycle, the current state is updated (line 4) and propagated to the end of the planning cycle (line 5), yielding $\hat{x}_{t+\Delta t}$. The tree root is set to the node whose trajectory the propagated state is following; this node's trajectory is committed and must be followed. The remaining time in the cycle is used to expand the tree (lines 6–8). Following this tree growth, the cost estimates are used [9] to select the best path in the tree (line 9), and this path is repropgated using the model dynamics from the predicted state $\hat{x}_{t+\Delta t|t}$ (line 13). This "lazy check" approach eliminates the need to constantly re-check the entire tree for feasibility [13]. If this path is still feasible, it is chosen as the current path to execute (lines 14–15). Otherwise, the infeasible portion of the path is removed and the process is repeated (lines 16-17) until either a feasible path is found or the entire tree is pruned. If the latter case occurs, the system has no path to execute, and some "safety" motion primitive is applied to attempt to keep the vehicle in a safe state [13].

C. Open-Loop RRT vs. Closed-Loop RRT

The RRT algorithm uses the system model in two different ways: the prediction of the system motion, and the real-time execution of a motion plan. Thus for the purposes of trajectory following, two key considerations are (i) how the *model* is controlled and (ii) how the *system* executes a given path. The open-loop and closed-loop formulations are shown graphically in Figures 1 and 2, respectively.

Inputs are selected for application to the system model in line 6 of Algorithm 1, and may be selected either in open-loop or closed-loop. In open-loop, the input u may be selected through discretization of $\mathscr{U}$, one-step optimization of the trajectory over $\mathscr{U}$, or a sampling-based approximation of this optimization [1]. In closed-loop, some reference state $r \in \mathbb{R}^{n_r}$ is propagated based on the sample x_{samp}; the input

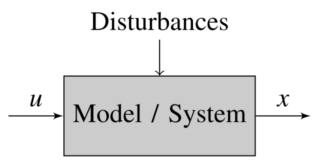


Fig. 1. Open-loop prediction using RRT; the predicted trajectory input sequence u is applied directly to the system.

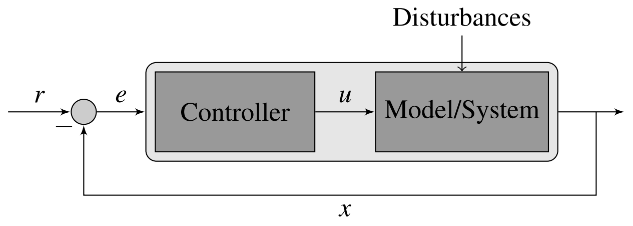


Fig. 2. Closed-loop prediction using RRT; reference commands r are used to generate inputs u for the system model.

is then selected through the controller

$$u = \kappa(r, x) \tag{5}$$

where u is the input to be applied to the node with state x and κ is the (nonlinear) control law. Through the reference r, the trajectories can be designed to meet certain qualitative considerations, such as a desired speed.

The CL-RRT algorithm is distinct from existing RRT approaches in that it samples in the space of the reference inputs r, with the reference control loop (5) around the system. Two separate trees are maintained: one for the reference inputs, and one for the simulated trajectories (Figure 3). The closed-loop simulation is *not* expected to accurately track the reference trajectory; it is instead important for the actual trajectory to match the predicted system trajectory.

Once a path has been selected via line 15 of Algorithm 2, the path may be executed either in open-loop or closed-loop. In open-loop, the inputs used to generate the path via line 6 of Algorithm 1 are applied directly to the system (1) in the same sequence. In closed-loop, some reference law (5) is applied to follow the path using closed-loop dynamics. Note that this does not require the model to be closed-loop, though the same reference law should be used in both cases.

IV. Linear System - Error Propagation

This section develops a model of the *error dynamics* for a system controlled using RRT to consider how the algorithm handles prediction mismatch whether in open-loop or closed-loop. The error dynamics capture the evolution of this prediction mismatch, or deviation between the predicted trajectory of (3) and the actual trajectory of (1). Under the assumption of linear time-invariance, we show that the model dynamics are also applicable to the error dynamics, whether open-loop or closed-loop; thus the controller can be designed to achieve known bounds on the prediction error over time.

Assume an LTI system with an additive disturbance; the system (1) and corresponding model (3) take the form

$$x_{t+1} = Ax_t + Bu_t + Gw_t, \tag{6}$$
$$\hat{x}_{t+1} = A\hat{x}_t + B\hat{u}_t. \tag{7}$$

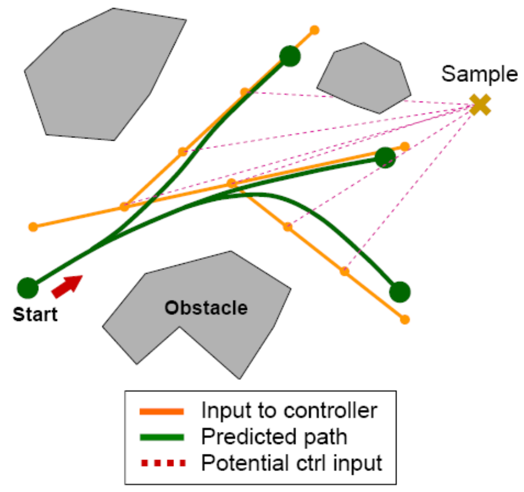


Fig. 3. Trajectories are propagated using the closed-loop vehicle model and checked for feasibility. The controller inputs are straight lines ending at the sample [11].

If a closed-loop control law (5) is applied to the system, it is assumed to take the linear form

$$u_t = K(x_t - r_t), \tag{8}$$

where r_t is the reference at timestep t; the model uses

$$\hat{u}_t = K(\hat{x}_t - \hat{r}_t), \tag{9}$$

where $\hat{r}_t$ is the reference at timestep t. Note that if both the system and model are closed-loop, then $\hat{r}_t = r_t \ \forall t$, since it is known *a priori*. Likewise, if both the system and model are open-loop, then $\hat{u}_t = u_t \ \forall t$.

We can use this simplified model to gain some intuition on how mismatch between the system and model propagates over time. Let $e_t = x_t - \hat{x}_t$ correspond to the error between the actual system state and predicted state at timestep t; we are interested in how how $e_t \in \mathbb{R}^{n_x}$ evolves over time. Assume that the error is initially zero, i.e. $e_0 = 0$.

A. Open-Loop Model, Open-Loop System

Suppose that both the system (6) and model (7) are applied in open-loop; the corresponding equations are then

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + w_t, & (10) \\ \hat{x}_{t+1} &= A\hat{x}_t + Bu_t. & (11) \end{aligned}$$

At timestep t, the actual state x_t and predicted state $\hat{x}_t$ can be shown to have the explicit representation

$$\begin{aligned} x_t &= A^t x_0 + \sum_{j=0}^{t-1} A^{t-j-1} Bu_j + \sum_{j=0}^{t-1} A^{t-j-1} w_j, & (12) \\ \hat{x}_t &= A^t x_0 + \sum_{j=0}^{t-1} A^{t-j-1} Bu_j. & (13) \end{aligned}$$

The resulting prediction error is

$$e_t = \sum_{j=0}^{t-1} A^{t-j-1} w_j \ \Rightarrow \ e_{t+1} = Ae_t + w_t. \tag{14}$$

Note that the error dynamics are stable if and only if the open-loop system is stable. There is thus no means to stabilize the error dynamics, meaning the error may grow without bound for an unstable system.

B. Closed-Loop Model, Closed-Loop System

The equations for closed-loop RRT, derived by inserting (8) into (6) and (9) into (7), are

$$\begin{aligned} x_{t+1} &= (A+BK)x_t - BKr_t + w_t, & (15) \\ \hat{x}_{t+1} &= (A+BK)\hat{x}_t - BKr_t. & (16) \end{aligned}$$

At timestep t, the actual state x_t and predicted state $\hat{x}_t$ can be shown to have the explicit representation

$$\begin{aligned} x_t &= (A+BK)^t x_0 - \sum_{j=0}^{t-1} (A+BK)^{t-j-1} BKr_j & (17) \\ &\quad + \sum_{j=0}^{t-1} (A+BK)^{t-j-1} w_j, \\ \hat{x}_t &= (A+BK)^t x_0 - \sum_{j=0}^{t-1} (A+BK)^{t-j-1} BKr_j. & (18) \end{aligned}$$

The resulting prediction error is

$$e_t = \sum_{j=0}^{t-1} (A+BK)^{t-j-1} w_j \ \Rightarrow \ e_{t+1} = (A+BK)e_t + w_t. \tag{19}$$

Thus, if K is chosen to stabilize the open-loop matrix A, the error dynamics will be stable, as well. The controller provides the user with parameters to explicitly model the error propagation bounds over time. Note that the error dynamics are independent of the reference, as long as the same reference is applied to both system and model.

C. Closed-Loop System, Open-Loop Model

Suppose the model is open-loop, but the system is actually executed in closed-loop (see Section III-C); then the equations for the system and model are (10) and (16), respectively. At timestep t, the actual state x_t and predicted state $\hat{x}_t$ can be shown to have the explicit representations (12) and (17). It is unreasonable to expect any kind of meaningful trajectory tracking in this case, since the system is using a controller in execution for which the RRT algorithm has zero knowledge. Section VII shows that by using a closed-loop model, not only is tracking performance improved, but the predicted trajectories tend to be more realistic since they are operating on the "stabilized" sample space.

V. Linear System - Robustness

This section establishes several theoretical properties for the CL-RRT algorithm on linear systems, with the additional assumption that the disturbance is bounded. In particular, using the property that a stable linear system subject to bounded process noise has error dynamics which are BIBO (bounded-input / bounded-output) stable, it is shown that appropriate tightening of the constraints $\mathcal{X}_t$ and $\mathcal{U}$ can be performed to guarantee robust feasibility. This is achieved by proving that conventional approaches in robust model predictive control (RMPC) [16] perform error propagation in the same manner as closed-loop RRT.

In Section II it was assumed that w_t has a known probability distribution $w_t \sim P(W)$. Here, it is further assumed that w_t is constrained within the bounded set $\mathscr{W}$, which contains the origin. Additionally, assume that (A,B) in (6) is stabilizable and $\mathscr{X}_t \equiv \mathscr{X}$ is time-invariant.

Suppose that CL-RRT (Algorithm 2) is applied to the system (6) using reference law (8) and corresponding models (7),(9), such that $A+BK$ is stable. It is straightforward to show that the error dynamics are BIBO stable, for example using the minimum robust positively invariant (mRPI) set [17]. This provides a finite bound on the maximum prediction mismatch, such that the constraints $\mathscr{X}$ and $\mathscr{U}$ can be modified within the RRT model to guarantee feasibility for the actual executed trajectory. As one example, we utilize the tube MPC technique for robustness [18]. Other RMPC approaches can also be used, e.g. [19].

The Tube MPC optimization seeks to identify a *tube* of feasible trajectories which nominally satisfies the original constraints [18]. The tube consists of a disturbance-free trajectory centerline and a tube "cross-section" at each timestep, comprised of the set Z centered on the centerline, both of which satisfy the nominal constraints. As the actual trajectory is executed, a state feedback policy using linear feedback K ensures that the system remains within this tube at all time steps. Since the tube cross-section Z is the same at every timestep, identifying a *tube* which satisfies the *original* constraints is equivalent to identifying a *trajectory* which satisfies the constraints *tightened by Z*,

$$\mathscr{X}^- = \mathscr{X} \ominus Z, \quad \text{and} \quad \mathscr{U}^- = \mathscr{U} \ominus KZ, \tag{20}$$

where $\ominus$ denotes Pontryagin set subtraction and Z satisfies the robust invariance constraint

$$(A+BK)x + w \in Z \quad \forall\, x \in Z,\ \forall\, w \in \mathscr{W}. \tag{21}$$

Theorem 1. *Given the above assumptions, suppose that the state constraints $\mathscr{X}$ and input constraints $\mathscr{U}$ are tightened in lines 5-6 of Algorithm 1 using (20). Then any path followed using CL-RRT (Algorithm 2) is guaranteed to satisfy these constraints.*

Proof: From Ref. [18], the control law takes the form

$$u_t = \hat{u}_t + K(x_t - \hat{x}_t); \tag{22}$$

thus the actual system and model are, respectively,

$$\begin{aligned} x_{t+1} &= Ax_t + B\hat{u}_t + BK(x_t - \hat{x}_t) + w_t. & (23)\\ \hat{x}_{t+1} &= A\hat{x}_t + B\hat{u}_t. & (24)\end{aligned}$$

At timestep t, the actual state x_t and predicted state $\hat{x}_t$ can be shown to have the explicit representation

$$\begin{aligned} x_t &= A^t x_0 + \sum_{j=0}^{t-1} A^{t-j-1} B\hat{u}_j + \sum_{j=0}^{t-1} (A+BK)^{t-j-1} w_j, & (25)\\ \hat{x}_t &= A^t x_0 + \sum_{j=0}^{t-1} A^{t-j-1} B\hat{u}_j. & (26)\end{aligned}$$

The error dynamics then take the form $e_{t+1} = (A+BK)e_t + w_t$, which is identical to (19) for closed-loop RRT. By applying (20), the closed-loop RRT path generation process is identical to the tube MPC path generation process, and thus yields paths which are robustly feasible [18]. ■

VI. Nonlinear Systems

For nonlinear systems in the presence of obstacles, most analytical techniques fail and combinatorial motion planning methods apply to only very simple cases, necessitating the use of sampling-based motion planning [5]. As opposed to other methods such as probabilistic roadmaps [7]), RRTs do not assume that an exact local planner exists. This allows RRTs to handle dynamical systems with nonlinearities such as nonholonomic constraints [5].

Even though the analysis in this paper focuses on linear systems, the CL-RRT algorithm is applicable to systems with various nonlinearities. Given a nonlinear system controlled with a robust nonlinear controller (see for instance Ref. [20]), it is expected that the robustness to disturbance in the CL-RRT planning framework will be better than planning using open-loop planners. In particular, if the system is linearized about a given trajectory and the controller prevents large deviations from this trajectory, similar results to the linear case might be expected [21]. We are currently exploring whether theoretical guarantees can be established for certain classes of nonlinear systems and/or controllers.

Section VII-B considers a nonlinear simulation scenario. In Refs. [11,22], the CL-RRT algorithm is used for mobile robots with differential constraints and nonlinear controllers. Experimental results have yielded prediction errors on the order of tens of centimeters, over complex trajectories up to 100 meters in length, for both a full-size robotic fork truck [22] and a full-size robotic Land Rover LR3 [11].

VII. Examples

The following simulation examples demonstrate how CL-RRT achieves superior trajectory tracking, critical for safe operations, compared to open-loop alternatives. In the linear example, both the model and system must be in closed-loop to achieve accurate tracking and likely feasibility. The nonlinear example provides empirical evidence closed-loop RRT can achieve accurate tracking for nonlinear systems, even in cluttered environments with significant uncertainty.

A. Linear Example

Consider the operation of a double integrator (quadrotor) in a relatively constrained, two-dimensional environment (Figure 4). The system dynamics are

$$\begin{aligned} \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v^x_{t+1} \\ v^y_{t+1} \end{bmatrix} &= \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v^x_t \\ v^y_t \end{bmatrix} \\ &\quad + \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} u^x_t \\ u^y_t \end{bmatrix} + \begin{bmatrix} w^x_t \\ w^y_t \end{bmatrix} \right), \end{aligned}$$

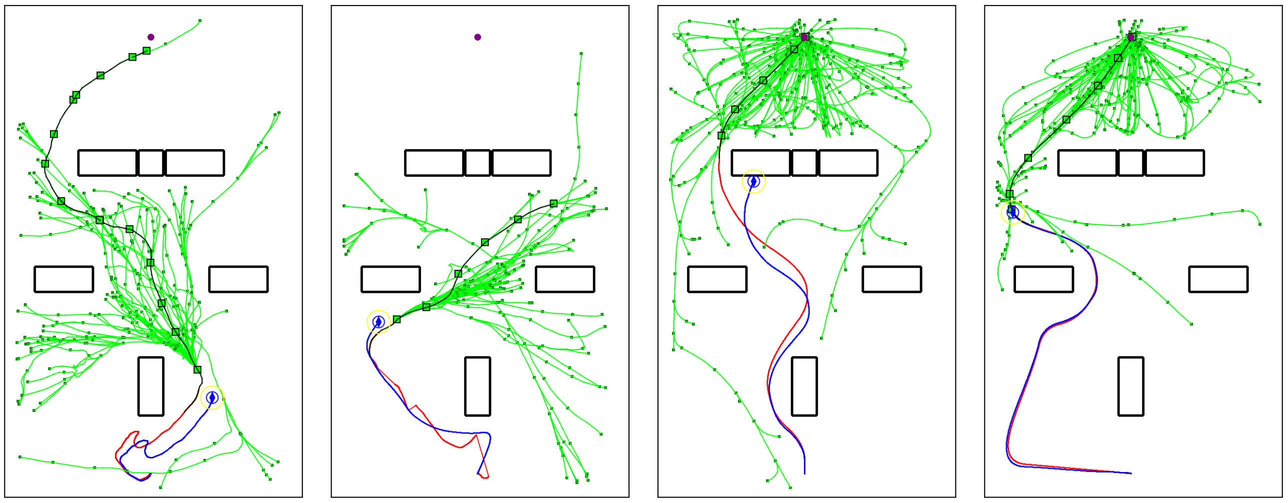

(a) Open-loop model, open-loop system (OL/OL)
(b) Open-loop model, closed-loop system (OL/CL)
(c) Closed-loop model, open-loop system (CL/OL)
(d) Closed-loop model, closed-loop system (CL/CL)

Fig. 4. Typical trajectory snapshot for each case in the linear example. The vehicle (circled blue object) starts at bottom-center and attempts to reach the goal waypoint (purple, top) while avoiding walls and obstacles (black). The blue path denotes the vehicle's actual trajectory, while the red path denotes the vehicle's predicted trajectory. The current RRT tree is marked in green; the current path chosen in the tree is emphasized with black edges.

where $dt = 0.02$s, subject to avoidance constraints $\mathscr{X}$, input constraints $\mathscr{U} = \{(u^x, u^y) \mid |u^x| \le 1, |u^y| \le 1\}$, and disturbance set $\mathscr{W} = \{(w^x, w^y) \mid |w^x| \le 0.3, |w^y| \le 0.3\}$. The reference r_t is moved from the parent node waypoint to the sample waypoint x_{samp} at 0.3 m/s, while the controller is

$$\begin{aligned} u_t^x &= -0.3(x_t - r_t^x) - 0.6(v_t^x - r_t^{v_x}), \\ u_t^y &= -0.3(y_t - r_t^y) - 0.6(v_t^y - r_t^{v_y}), \end{aligned}$$

where (r_t^x, r_t^y) is the reference position and $(r_t^{v_x}, r_t^{v_y})$ is the reference velocity (0.3 m/s in the direction of the waypoint). The vehicle's objective is to move across the room in minimum time while avoiding all obstacles. A 10cm buffer is also placed around the vehicle for safety reasons. Note that the closed-loop controller is relatively weak compared to the maximum disturbance magnitude; thus $\mathscr{E}_\infty$ is too large for Theorem 1 to be applied here.

Simulations were performed using an implementation of Algorithm 2 in Java, run on an Intel 2.53 GHz quad-core laptop with 3.48GB of RAM. In Algorithms 1–2, the tree capacity was limited to 1000 nodes, $M = 5$, and Δt = 0.5s. Twenty simulations of this planning problem were performed for each of four cases: whether the RRT model is open-loop or closed-loop, and whether the system's path is executed in open-loop or closed-loop (Section III-C). When an open-loop model was used, inputs were selected (line 6 of Algorithm 1) at each timestep by applying 20 random inputs and choosing the one which brings the system closest (in Euclidean norm) to the sample. Table I shows the feasibility, prediction error, and complexity results for these cases; all values are averaged over 20 trials. A representative trajectory and tree for each case is shown in Figure 4.

Table I and Figure 4 demonstrate that CL-RRT does a superior job of minimizing prediction error compared to open-loop alternatives. The CL/CL case achieves both average and maximum prediction errors an order of magnitude lower than those cases with open-loop execution (OL/OL and OL/CL).

TABLE I
SIMULATION RESULTS

Model?	**System?**	**% Feas.**	**Avg Error, m**	**Max Error, m**	**Time per Node, ms**
Open	Open	10	0.341	0.997	7.04
Open	Closed	45	N/A*	N/A*	9.66
Closed	Open	0	0.336	0.945	5.22
Closed	Closed	100	0.025	0.057	6.77

*Cannot be computed, as the (faster) predicted trajectory enters non-committed portions of the tree which are not executed.

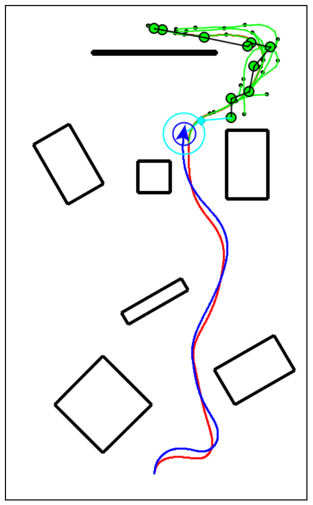
This is the expected result, given that A is unstable and $A + BK$ is stable, since the error dynamics have the same stability properties as the system model (Section IV). All 20 trials for CL/CL feasibly reach the goal, while all but 2 trials collide with an obstacle for the open-loop execution cases. Comparing CL/CL (Figure 4(d)) with OL/CL (Figure 4(b)), we see that both trajectories are smooth, but the latter path does not match well the jagged path generated by the open-loop RRT model, since the model is not aware of the controller used for execution. Because of this mismatch, only about half of the OL/CL paths feasibly reach the goal.

B. Nonlinear Example

Now consider the operation of a skid-steered vehicle in a highly cluttered two-dimensional environment (Figure 5). The nonlinear system dynamics are

$$\begin{aligned} x_{t+1} &= x_t + dt(1/2)(v_t^L + v_t^R)\cos\theta_t, \\ y_{t+1} &= y_t + dt(1/2)(v_t^L + v_t^R)\sin\theta_t, \\ \theta_{t+1} &= \theta_t + dt(v_t^R - v_t^L), \\ v_t^L &= \text{sat}\left(\bar{v}_t + 0.5\Delta v_t + w_t^L, 0.5\right), \\ v_t^R &= \text{sat}\left(\bar{v}_t - 0.5\Delta v_t + w_t^R, 0.5\right), \end{aligned}$$

where (x, y) is the vehicle position, θ is the heading, v^L and v^R are the left and right wheel speeds, respectively, sat is the saturation function and (w^L, w^R) is a bounded disturbance. A variation of the pure pursuit controller [13] is applied, assuming forward direction only. Note that the goal at top


(a) Uncertain wheel speeds

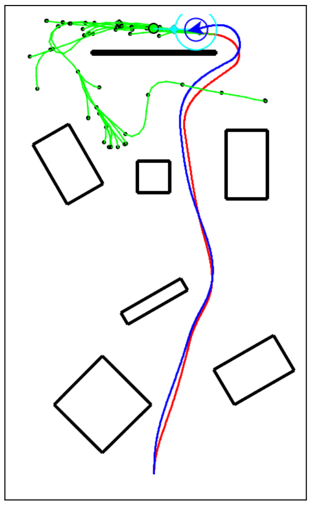

(b) Uncertain+biased wheel speeds

Fig. 5. Typical trajectory for each case in the nonlinear example; in both cases shown the vehicle successfully reaches the goal at top-center.

is now in a corridor of width only twice the diameter of the vehicle, requiring precise control to reach.

Figure 5 shows a representative trajectory and tree for each of two bounded disturbance sets, both using CL-RRT. In Figure 5(a), $\mathscr{W} = \{(w^L, w^R) \mid w^L \in [-0.1\bar{v}, +0.1\bar{v}],\ w^R \in [-0.1\bar{v}, +0.1\bar{v}]\}$; this might represent uneven terrain or uncertain wheel actuation. In open-loop, this uncertainty amounts to a random walk on the wheel speeds. However, over 20 simulations in closed-loop, a feasible path to the goal is still identified and executed in 14 trials (70%). In Figure 5(b), $\mathscr{W} = \{(w^L, w^R) \mid w^L \in [-0.2\bar{v}, 0],\ w^R \in [0, +0.2\bar{v}]\}$, representing a significant bias in the steering map in addition to previous uncertainty. In open-loop, the resulting heading drift would almost certainly result in infeasibility. By closing the loop, CL-RRT reduces this very poor mapping into a bounded offset between the predicted and actual paths, such that a feasible path is still found in 50% of trials.

VIII. Conclusions

This paper has shown that closed-loop RRT can be used to robustly track predicted trajectories in the presence of uncertainty. With accurate trajectory tracking, the likelihood that the system remains feasible improves significantly compared to open-loop alternatives, as demonstrated by the simulation results. Through appropriate choice of reference and controller, bounds can be designed on the maximum prediction error. For the case of a linear system subject to bounded disturbances, results show that by tightening the constraints for the error bounds, this *likelihood* of feasibility can be converted into a *guarantee* of robust feasibility. Future work will consider the effects of correcting for known prediction errors on system stability, as well as space-parametrized trajectory tracking.

Acknowledgements

The authors would like to recognize Daniel Levine for his contributions to this research effort. Research funded in part by AFOSR grant FA9550-08-1-0086.

References

[1] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999, pp. 473–479.

[2] ——, “Rapidly-exploring random trees: Progress and prospects,” in *Algorithmic and Computational Robotics: New Directions: the Fourth Workshop on the Algorithmic Foundations of Robotics*, 2001.

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

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

[5] S. M. LaValle, *Planning Algorithms*, S. M. LaValle, Ed. Cambridge, U.K.: Cambridge University Press, 2006.

[6] M. H. Overmars, “A random approach to motion planning,” Utretcht University, Utrecht, The Netherlands, Tech. Rep., October 1992, rUU-CS-92-32.

[7] L. Kavraki and J. C. Latombe, “Randomized preprocessing of configuration space for fast path planning,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1994.

[8] J. Barraquand, B. Langlois, and J. C. Latombe, “Numerical potential field techniques for robot path planning,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 224–241, March-April 1992.

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

[10] J. B. Saunders, B. Call, A. Curtis, R. W. Beard, and T. W. McLain, “Static and dynamic obstacle avoidance in miniature air vehicles,” in *Proceedings of the AIAA Infotech@Aerospace Conference*, Arlington, VA, September 2005.

[11] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, “Real-time motion planning with applications to autonomous urban driving,” *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, September 2009.

[12] J. Leonard and et al, “A perception-driven autonomous urban vehicle,” *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[13] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. P. How, “Motion planning in complex environments using closed-loop prediction,” 2008, submitted to the Proceedings of the IEEE Conference on Guidance, Navigation, and Control.

[14] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, “Motion planning for urban driving using RRT,” in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Nice, France, September 2008, pp. 1681–1686.

[15] A. Stentz, “Optimal and efficient path planning for partially-known environments,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1994.

[16] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, “Constrained model predictive control: Stability and optimality,” *Automatica*, vol. 36, pp. 789–814, 2000.

[17] I. Kolmanovsky and E. G. Gilbert, “Theory and computation of disturbance invariant sets for discrete-time linear systems,” *Mathematical Problems in Engineering*, vol. 4, pp. 317–367, 1998.

[18] D. Q. Mayne, M. M. Seron, and S. V. Raković, “Robust model predictive control of constrained linear systems with bounded disturbances,” *Automatica*, vol. 41, pp. 219–224, 2005.

[19] Y. Kuwata, A. Richards, and J. How, “Robust receding horizon control using generalized constraint tightening,” in *Proceedings of the IEEE American Control Conference*, New York City, NY, July 2007, pp. 4482–4487.

[20] R. A. Freeman and P. V. Kokotovic, *Robust Nonlinear Control Design: State-space and Lyapunov Techniques*. Boston, MA: Birkhäuser, 1996.

[21] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Prentice Hall, 1991.

[22] S. Teller, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J. P. How, J. Jeon, S. Karaman, B. Luders, N. Roy, T. Sainath, and M. R. Walter, “A voice-commanded robotic forklift working alongside humans in minimally-prepared outdoor environments,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2010 (to appear).