Resolution Complete Rapidly-Exploring Random Trees

Peng Cheng Steven M. LaValle

Dept. of Computer Science
University of Illinois
Urbana, IL 61801 USA
{pcheng1, lavalle}@cs.uiuc.edu

Abstract

Trajectory design for high-dimensional systems with nonconvex constraints has gained considerable attention recently. This paper addresses the issue of establishing resolution completeness for trajectory design algorithms by utilizing Lipschitz conditions and an accessibility graph. By combining systematic search ideas with randomized techniques, we introduce resolution complete rapidly-exploring random trees, which obtain both fast experimental performance and eventual deterministic resolution completeness. Some illustrative trajectory design examples from our implemented algorithms are shown.

1 Introduction

Trajectory design for high-dimensional nonlinear systems with nonconvex constraints attracts more and more attention in current research. A challenging problem is shown in Figure 1. In most previous work [1, 5, 6, 8, 15, 16, 18], dynamic programming variations are developed to produce optimal solutions for low-dimensional problems. Randomized techniques that find feasible non-optimal solutions to high-dimensional problems were introduced in [12, 13], based on the introduction of Rapidly-exploring Random Trees (RRTs). Their application to autonomous vehicles [7] and nonlinear underactuated vehicles [9, 19] has achieved good results. The application of randomized techniques in trajectory design is also considered in [10]. Many trajectory design methods can be considered as a form of single-direction or bidirectional search. To avoid searching the same state repeatedly and to terminate the searching in a finite number of iterations, many methods have been used to discretize the configuration or state space to yield a finite number of nodes in the search graph. In [6], the discretization is based on the grid built from acceleration bounds and a fixed time step. Nonuniform boxes are used to discretize the configuration space in [18]. In [1], the configuration space is decomposed into disjoint parallelpipeds of equal size and asymptotic completeness is derived. More generally in [2], a partition is used to discretize the state space, and there is no restriction on the form of sets in the partition. However, choosing an appropriate discretization resolution to guarantee that the existing solution will be found is challenging.

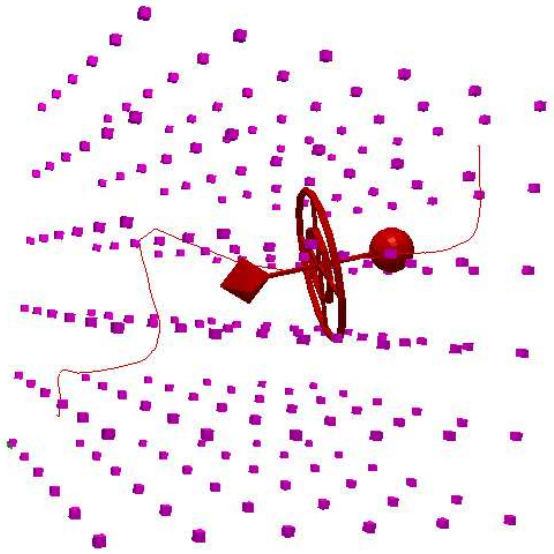

Figure 1: A trajectory design for an underactuated 12-dimensional spacecraft with three thrusters moving in a 3D grid.

We present extensions of the RRT that yield solutions quickly in practice, and have theoretical guarantees of resolution completeness, as obtained in many approaches designed for lower-dimensional state spaces. Thus, we can obtain both the advantages of fast randomized solutions combined with systemic search in the worst case. By considering an accessibility graph and Lipschitz conditions, conditions are presented, which guarantee that the planner will find an existing solution, if the discretization resolution is in a specified range. It is different from the asymptotic completeness in [1], which requires that for a small-time locally controllable system, the control period is small enough, the search depth is big enough and the resolution is high enough.

2 Problem Formulation

The trajectory design problem, Δ, considered in this paper is defined as a nine-tuple $(X, X_{free}, x_{init}, X_{goal}, U, \delta t, \rho, \tau, f)$. The *state space*, $X \subset \mathbb{R}^n$, is an n-dimensional compact differentiable manifold; $X_{free} \subseteq X$ is the set of all states that satisfy global constraints, such as collision avoidance and velocity bounds; $x_{init} \in X_{free}$ and $X_{goal} \subset X_{free}$ are boundary conditions; U is the set of inputs (we could also allow state-dependent input sets); δt denotes the control period during which a constant $u \in U$ is applied. A metric on the state space is defined, $\rho : X \times X \rightarrow [0, \infty)$ defined on X. A prescribed solution tolerance $\tau \geq 0$ is given for achieving a goal state. Finally, f gives the *equation of motion*, $\dot{x} = f(x, u)$, which encodes kinematic and dynamic constraints.

The objective is to find a *solution control function*, $u : [t_0, t_f] \rightarrow U$, in which t_0 is the starting time and $t_f = t_0 + K\delta t$. Furthermore, $u(t)$ is constant over each control period, resulting in $u(t) = u_i \in U$ during each control period i. The *solution path*, $\pi : [t_0, t_f] \rightarrow X_{free}$ results from integration of f over the control periods, and must satisfy the boundary conditions as follows: $\pi(t_0) = x_{init}$ and $\rho(\pi(t_f), X_{goal}) \leq \tau$ (note that ρ is measuring the distance from a point to a set).

A solution π will be represented for convenience by either an input sequence $\pi_u = \{u_1, u_2, \cdots, u_K\}$ in which $u_i = u(t_0 + (i-1)\delta t)$ or a state sequence $\pi_x = \{ x_0 = x_{init}, x_1, \cdots, x_K\}$ in which $x_i = \pi(t_0 + i\delta t)$.

3 The Accessibility Graph

The problem, Δ, represents a multi-stage process. Each stage corresponds to a time and has a set of accessible states. In related work, stability and accessibility have been studied in the context of quantized control systems [4, 17]. In this section, we present an accessibility graph, which characterizes the set of all possible discrete-time trajectories for a given Δ (starting in x_{init}, and reaching anywhere in X_{free}).

We first consider the transition that occurs from the application of a constant input $u \in U$, at a state $x(t) \in X$, for the control period δt. Let the *system transition equation*, $s(u, x, t, \delta t)$ represent the resulting state by integration:

$$s(u, x, t, \delta t) = x + \int_t^{t+\delta t} f(x, u) dt.$$

Note that this formulation does not take into account X_{free}. Therefore, we also define a *violation-free system transition equation*, $\tilde{s}(u, x, t, \delta t)$, which simply restricts the domain of s to include only inputs the yield a violation-free trajectory from time t to time $t + \delta t$. Let $\tilde{U}(x)$ denote this state-dependent set of inputs.

Let N_k be the set of accessible states at time $t_0 + (k-1)\delta t$. At first, $N_1 = \{x \mid x = x_{init}\}$. Next,

$$N_2 = \{x \mid x = \tilde{s}(u, x', t_0, \delta t), u \in \tilde{U}(x'), x' \in N_1\}.$$

For the k^{th} stage,

$$N_k = \{x \mid x = \tilde{s}(u, x', t_k, \delta t), u \in \tilde{U}(x'), x' \in N_{k-1}\},$$

in which $t_k = t_0 + (k-1)\delta t$.

Based on the above sequence, the (directed) *accessibility graph*, $G_\infty(N_\infty, E_\infty)$, for a given Δ, is defined by the set, $N_\infty = \cup_{k=1}^{\infty} N_k$ of nodes, and the set,

$$E_\infty = \{e(x, x') \mid x \in N_\infty, \exists u \in \tilde{U}(x), x' = \tilde{s}(u, x, t, \delta t)\},$$

of edges.

If $\exists x_i \in N_i$ and $\exists x_j \in N_j$ such that $i \neq j$ and $x_i = x_j$, then we say G_∞ has *cyclic paths* and the corresponding Δ is *cyclic*; otherwise, Δ is *acyclic*. If $\exists m > 1$ such that $N_m - \bigcup_{k=1}^{m-1} N_k = \varnothing$, then N_∞ is finite; otherwise, N_∞ is infinite.

We view incremental planning algorithms as techniques that explore G_∞. During each iteration of a search algorithm, more of G_∞ is revealed. If there exists a solution path from x_{init} to x_{goal} under the control period δt and tolerance τ, it must exist in G_∞. Let $G_{sub}(N_{sub}, E_{sub})$ denote the graph of trajectories that are explored by a search algorithm. If exact computations are assumed then G_{sub} is a subgraph of G_∞ that is explored by a search algorithm. In Section 4, we present RRT-based algorithms that incrementally reveal G_∞ (also, numerical errors are taken into account). The analysis in Section 5 applies to these algorithms; moreover, the general theory can be adapted to other incremental search algorithms by considering the appropriate G_{sub} obtained by the algorithm.

4 RRT-Based Planners

Resolution complete RRT (RC-RRT) planning algorithms are given here. The basic RRT algorithm is presented in [12]. Initially, x_{init} is the only node of G_{sub}. For each iteration, a random state $x_{rand} \in X$ is chosen, and $x_{near} \in N_{sub}$ is selected as the nearest state to x_{rand} according to a metric function ρ. For x_{near}, an input u_{best} is chosen to generate a new state x_{new} which is closest to x_{rand} among all states generated by applying a constant control from x_{near} over the control period. If x_{new} satisfies the global constraints, then x_{new} will be added to G_{sub}. The performance of RRTs degrades when ρ poorly approximates the real path cost [3, 13]. Time is often wasted choosing states destined to eventually violate the global constraints, and the probability of expanding further along the solution path might decrease with more and more iterations.

A basic RC-RRT In [3], we presented a basic RC-RRT algorithm, named here as RC-RRT$_1$, in which *exploration information* and the *constraint violation frequency (CVF)* are collected and used during the exploration to reduce metric sensitivity. Exploration information records whether an input has already tried from a

state, so that it is not repeated in the future. The CVF provides an underapproximation to the percentage of trajectories from that state which ultimately leave X_{free}. To choose x_{near}, exploration information, the CVF, and ρ are used. If all of the inputs have been expanded from a state x, it will be precluded in future iterations. In each iteration, while considering nearest states, a state will be skipped with a probability that is the CVF of x. If no node is chosen, the nearest node with unexpanded inputs will be chosen. To choose u_{best}, exploration information and ρ are used, and u_{best} is applied to x_{best} to generate x'_{new}. From x_{best}, only inputs that were not expanded in previous iterations will be considered. The violation-free x'_{new} with the shortest distance to x_{rand} will be added to G_{sub}. If $x'_{new} \notin X_{free}$, the CVF information is propagated up the tree (see [3] for details). RC-RRT$_1$ exhibits good experimental performance in [3]; however, completeness only occurs if G_∞ is acyclic with finite N_∞. Two extensions are presented next, which lead to resolution complete planners for general G_∞.

An improvement via neighborhood analysis The RC-RRT$_1$ employs only exploration information to exclude repeated states; however, if G_∞ is *cyclic* or has infinite N_∞, the planner might continue running forever. To ensure that the planner terminates in finite time for any Δ, a covering of X, $\Pi(X)$, is used to yield a finite approximation to X. We define a collection of balls, each of which is centered on an element of N_{sub}. Let $r_b > 0$ be the radius of each of these balls. Thus, for any $x \in N_{sub}$, let $B(x, r_b) = \{x' \in X \mid \rho(x, x') < r_b\}$ denote its corresponding ball. Let $\mathcal{B}(N_{sub})$ denote the union of $B(x, r_b)$ for all $x \in N_{sub}$. RC-RRT$_2$ uses these balls, and a new node is added to G_{sub} only if $x_{new} \notin \mathcal{B}(N_{sub})$. Thus, there will be at most one node per neighborhood, which is similar to the effect obtained in [1] but using a grid. Here, however, we do not explicitly construct a grid; grid-like coverage of accessible states will be obtained in the limit if a solution is not found early. Of course, r_b, needs to be small enough to avoid missing a solution for a given tolerance, but it should be large enough with respect to numerical precision errors. If exact arithmetic is used, then r_b can be made arbitrarily small to yield true asymptotic convergence.

Converging to optimal solutions Once neighborhood is introduced, once can also allow iterative refinement of trajectories into optimal solutions. Dynamic programming (e.g., [1]) explores all of the state space (up to some resolution) and returns an optimal solution; however, randomized methods sacrifice optimality by only exploring part of the state space. In RC-RRT$_3$, we combine optimizations with the RRT to yield optimal solutions on the part of the space so far explored. In the limit, this converges to finding globally optimal paths. Assume that a standard stage-additive loss functional is defined on the space of trajectories. Assume the

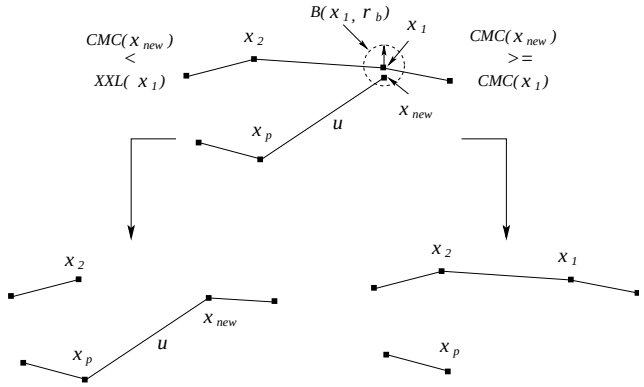

Figure 2: The current minimum path cost, $L(x)$, is maintained for the repeated state.

current minimum path cost, $L(x)$, from x_{init} to x, is calculated for all $x \in N_{sub}$. As shown in Figure 2, consider an RRT iteration in which x_{new} is successfully generated from x_p. If $x_{new} \notin \mathcal{B}(N_{sub})$, then x_{new} will be added to N_{sub}, and its path cost $L(x_{new}) = L(x_p) + l(x_p, u, \delta t)$ is stored, in which $u \in U$ is the input that leads x_p to x_{new}, and $l(x_p, u, \delta t)$ is the per-stage loss. Otherwise, if $x_{new} \in B(x', r_b)$ for some $x' \in N_{sub}$ there are two possibilities. If $L(x_{new}) \geq L(x')$, then x_{new} will be discarded. If $L(x_{new}) < L(x_1)$, then x_{new} replaces x' in N_{sub}. In addition to the edge from x_p to x_{new}, the new state inherits all in and out edges from x'. This generally leads to small gaps in the solution trajectories, but convergence is still obtained due to Lipschitz conditions. Using RC-RRT$_3$, the minimum-loss path among all paths so far considered will be obtained. In the limiting case, all actions and states will be considered, which results in convergence to a global optimum.

Building planners from RC-RRTs Practical planners can formed from RRTs in many ways; see [14] for many examples. Single tree planners can be constructed by biasing the sampling toward the goal. For example, with a small probability, instead of picking a random sample in X, a sample in X_{goal} can be chosen. Using bidirectional search ideas, dual-tree planners can be constructed that divide their time between exploring and attempting to connect to each other. One tree is rooted at the initial state and the other is rooted at a goal state. A trajectory is found when the two trees meet. The RC-RRT can be used as a replacement to the RRT in any existing RRT-based planners.

5 Resolution Completeness Analysis

The analysis is divided into two parts. First, in Theorem 5.1 it is shown that in spite of discretization of the

system and numerical rounding errors, the search graph to be explored by the RC-RRT algorithms contains a solution trajectory that falls within the specified solution tolerance, if a solution exists for the exact, continuous-time problem. If G_∞ is finite, then Theorem 5.1 implies that a solution will be found after a finite number of iterations. If G_∞ is infinite, then Theorem 5.2 is needed to show that RC-RRT$_2$ and RC-RRT$_3$ find a solution after a finite number of iterations. We give an explicit bound based on the resolution parameters and number of inputs. We note that the analysis here provides worst-case assurances that our planners are resolution complete; however, it does not characterize the good computational performance observed in practice. RRTs are designed to find a solution to most problems long before systematic coverage of the state space occurs.

For a given Δ, recall that G_∞ is fixed. If every parameter is represented algebraically, an exact geometric computation method [20] can be used, and no numerical computation error will exist. In this case, RC-RRTs generate $G_{sub} \subset G_\infty$. If floating point numbers are used to represent the parameters, then the search graph will approximate G_∞ with the numerical computation error. The algorithm might report a wrong solution when the computation error causes a non-solution path to enter the neighborhood of X_{goal}. The numerical computation error is related to the computer architecture, numerical algorithms, and state transition equation integration. It is difficult to analyze all of these factors. To characterize the effect of the computation error on the resolution completeness, only round-off error is considered here, and all other computations are assumed to be accurate.

Theorem 5.1 *Suppose $\check{x}$ is the round-off value of x, and $\rho(x, \check{x})$ is bounded by some fixed $\eta > 0$ for all $x \in X$. If there exists a solution π of length K with tolerance $\frac{\tau}{2}$ for a given Δ, then choosing τ to satisfy the following conditions ensures that the graphs being explored by RC-RRT$_2$ and RC-RRT$_3$ using $B(x, \epsilon)$ contain a solution with tolerance τ. If G_∞ is acyclic and N_∞ is finite, then RC-RRT$_1$ will find a solution with tolerance τ.*

1. The system transition equation $s(u, x, t, \delta t)$ meets the Lipschitz condition: $\forall u \in U$, $\forall t \in [t_0, t_f]$ and $\forall x_1, x_2 \in X$, $\rho(s(u, x_1, t, \delta t), s(u, x_2, t, \delta t)) < L_s \rho(x_1, x_2)$.

2. The round-off error bound $\eta < \frac{\tau(L_s - 1)}{2({L_s}^{K+1} - 1)}$.

3. The ball region radius, ϵ satisfies $\frac{{L_s}^{K_c+1} - 1}{L_s - 1}\eta < \epsilon < \min\{\frac{\tau(L_s - 1)}{2(L_s^K - L_s)} - \frac{{L_s}^{K+1} - 1}{L_s^K - L_s}\eta, \min_{k=0,\cdots,K-1} \rho(\pi(t_0 + k\delta t), \pi(t_0 + (k+1)\delta t))\}$, in which K_c is the path length of the largest cyclic path in the search graph G_∞.

4. There exists a violation-free "tunnel" around π such that $x \in X_{free}$ for all $x \in \{x \mid \rho(x, \pi(t)) \leq \tau, t \in [t_0, t_f]\}$.[1]

[1]The "tunnel" here is different from the "tube" in [6]. The "tunnel" is used to guarantee the completeness and only related to π and τ; however, the "tube" is used for the safety consideration and is related to robot sensors and velocity error correction rate.

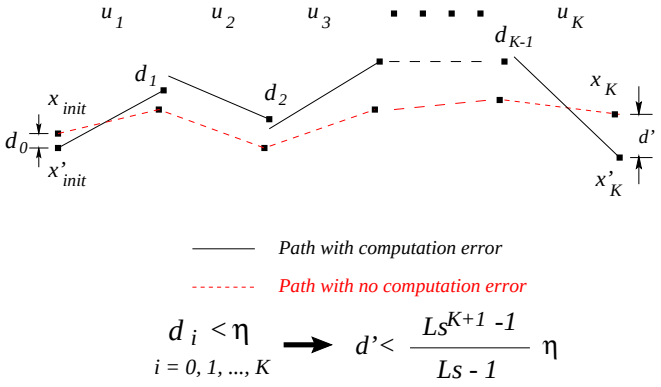

Figure 3: Accumulated computation error results from the round-off error.

Proof: Completeness arguments are given in three parts based on properties of G_∞:

Part I (finite N_∞ and acyclic G_∞) In this case, all RC-RRTs have resolution completeness without considering neighborhoods. Only completeness conditions of RC-RRT$_1$ are presented here because conditions for RC-RRT$_2$ and RC-RRT$_3$ for this G_∞ are the same as those for more general G_∞, which will be discussed in Part II.

If exact computation is used, the completeness is guaranteed by exploring until $G_{sub} = G_\infty$; therefore, assume numerical computation is used. Let $U_s = (u_1, u_2, \cdots, u_p)$ be a control sequence, and $\Phi(U_s, x, t_0) = s(u_p, (s(u_{p-1}, \cdots (s(u_1, x, t_0, \delta t)) \cdots, t_0 + (p-2)\delta t, \delta t)), t_0 + (p-1)\delta t, \delta t)$ be the final state by applying U_s on state x at time t_0. If numerical computation is used, for the solution path π_u, the initial state round-off error leads to the error bound $\rho(\Phi(\pi_u, x_{init}, t_0), \Phi(\pi_u, \check{x}_{init}, t_0)) < \frac{{L_s}^{K+1} - 1}{L_s - 1}\eta$ at the final state (Figure 3). If $\eta < \frac{\tau(L_s - 1)}{2({L_s}^{K+1} - 1)}$, then $\Phi(\pi_u, \check{x}_{init}, t_0)$ will have tolerance τ, and solution π will be returned.

Part II (finite N_∞ and cyclic G_∞) Because only exploration information is used in RC-RRT$_1$, it might run forever by exploring a cyclic path; however RC-RRT$_2$ and RC-RRT$_3$ are resolution complete in this case.

Using exact computation, G_{sub} generated by RC-RRT$_2$ and RC-RRT$_3$ is a subset of G_∞. If $B(x, 0)$ is used to exclude repeated states, the completeness is achieved by searching until $G_{sub} = G_\infty$. If $B(x, \epsilon)$ is used for some $\epsilon > 0$, then some $x_r \in N_{sub}$ is a representative of the node set $S_x = \{x \in N_\infty \mid \rho(x, x_r) < \epsilon\}$. Considering the solution π_x, if x_p is the state in π_x that is replaced by state x_r, then $\rho(\Phi(U_p, x_i, t_0 + p\delta t), \Phi(U_p, x_p, t_0 + p\delta t)) < L_s^{(K-p)}\epsilon$ in which $U_p = \{u_{p+1}, \cdots, u_K\} \subset \pi_u$. In the worst-case, if $x_1, x_2, \cdots, x_{K-1} \in \pi_x$ are replaced, then $\epsilon < \frac{\tau(L_s - 1)}{2(L_s^K - L_s)}$ will guarantee a solution with tolerance τ

will exist in the search graph.

Considering computation error, the limiting case of $B(x,0)$ cannot be reached because two identical states resulting from a cyclic path might appear different because of the computation error. $B(x,\epsilon)$, for some $\epsilon > 0$ must be used to account for the computation error. If ϵ is too small, a single state may result in two different state nodes in N_{sub}. Assume that K_c is the maximum length of all of cyclic paths. If ball radius ϵ is chosen so that $\epsilon > \frac{{L_s}^{K_c+1}-1}{L_s-1}\eta$, then two identical states will stay in the same ball region for any cyclic path in G_∞. Also, $\epsilon < \frac{\tau(L_s-1)}{2(L_s^K-L_s)} - \frac{{L_s}^{K+1}-1}{L_s^K-L_s}\eta$ is needed to guarantee a solution with tolerance τ will be found.

Part III (infinite N_∞ and G_∞) Only RC-RRT$_2$ and RC-RRT$_3$ using $B(x,\epsilon)$ for $\epsilon > 0$ are suitable for this problem. Clearly, N_{sub} must be finite because X is compact and no two states in N_{sub} may be within ϵ of each other. Similar to the above argument for exact computation, $\epsilon < \frac{\tau(L_s-1)}{2(L_s^K-L_s)}$ will guarantee resolution completeness; for numerical computation, the condition is $\frac{{L_s}^{K_c+1}-1}{L_s-1}\eta < \epsilon < \frac{\tau(L_s-1)}{2(L_s^K-L_s)} - \frac{{L_s}^{K+1}-1}{L_s^K-L_s}\eta$.

Finally, we address several minor concerns. Because of the computation error or the discretization, the path generated by the planner will be close to π. It might seem possible to miss a solution because the computed path could be violate the global constraints, even though π traverses X_{free}. However, this cannot happen because we required a violation-free "tunnel" around π such that $x \in X_{free}$ for $\forall x, \rho(x,\pi(t)) \leq \tau$ and $t \in [t_0, t_f]$.

Another concern is that to find the solution π, the planner has to add $x \in \pi_x$ to G_{sub} one by one. If ϵ is chosen such that $\pi_k \in B(\pi_{k-1},\epsilon)$ for some k, then π_k will not be added to G_{sub}, and the solution will not be found. If $\epsilon < \min_{k=0,\cdots,K-1} \rho(\pi(t_0+k\delta t), \pi(t_0+(k+1)\delta t))$, the above situation will not happen, and the solution will be eventually found. ■

Theorem 5.1 shows that resolution problems will not prevent RC-RRTs from exploring until a solution is found. Theorem 5.2 establishes that RC-RRTs will find the solution after a finite number of iterations, which corresponds to resolution completeness in the standard, deterministic sense. By using this result, an RC-RRT-based planner could iteratively improve the resolution parameters each time failure occurs, terminating only if a solution is found. This would yield the resolution completeness behavior obtained for methods such as approximate cell decomposition (see [11] for a discussion of resolution completeness in this context).

Theorem 5.2 *Let m be the number of inputs in U. If the resolution completeness conditions of Theorem 5.1 are satisfied, then RC-RRT$_2$ or RC-RRT$_3$ needs at most $n_p m$ iterations to determine whether there exists a solution for Δ.*

Proof: If the resolution completeness conditions of Theorem 5.1 are satisfied, a covering set $\Pi(X) = \{S_1, S_2, \cdots, S_{n_p}\}$ can be constructed that satisfies $X = \bigcup_{i=1}^{n_p} S_i$ and there exists some $\kappa > 0$ for all $S_i \in \Pi(X)$ with

$$\mu(S_i - \bigcup_{j \in \langle 1,2,\cdots,i-1,i+1,\cdots,n_p \rangle} S_j) > \kappa,$$

in which μ is the Lebesgue measure. Let r be the size of the neighborhood satisfying the conditions from Theorem 5.1, a ball neighborhood $B(x,r)$ can be inserted one by one to cover X and generate a $\Pi(X)$. Using $\Pi(X)$ in the planner will also achieve the same resolution completeness. It is similar to the condition in Theorem 5.1, except here the covering sets are built from the beginning; in Theorem 5.1 they are incrementally built.

For a compact set X, $\mu(X)$ is finite. Since each element of uniquely covers at least κ of $\mu(X)$, so there are only a finite number of sets in $\Pi(X)$ and n_p is finite.

For a given Δ, whether G_∞ has finite N_∞ or infinite N_∞, a finite $\Pi(X)$ can always be constructed. According to the RC-RRT algorithms, there is at most one node in any set in $\Pi(X)$, so there are at most n_p in the explored graph. RC-RRT$_2$-based planner and RC-RRT$_3$-based planner will stop searching when all of the inputs for all state nodes existing in N_{sub} are expanded. Marking one input for a state node as expanded needs at most one iteration. Thus, at most $n_p m$ iterations are required to determine whether a solution exists for a given Δ. ■

The following corollary follows from the previous theorems, and indicates that RC-RRT$_3$ will find a solution that is close to the optimal solution. By adding Lipschitz conditions to the loss functional, we could additionally ensure that the cost of the computed solution comes arbitrarily close to the true optimum.

Corollary 5.3 *If there exists a solution π of length K with tolerance $\frac{\tau}{2}$ for a given Δ, and π is optimal in $G_{sub} \subseteq G_\infty$, RC-RRT$_3$ will find π with tolerance τ if the conditions of Theorem 5.1 are satisfied and G_{sub} is generated by RC-RRT$_3$ when π is found.*

6 Experimental Evaluations

We briefly indicate some of the experimental advantages of planners based on RC-RRTs. In addition to resolution completeness implied by the theory from Section 5, we have observed significant performance improvement on challenging examples. For each experiment, we first set up the problem to make sure a solution exists, then ran the planners for 50 trials. The experiment ran until all trials were successfully completed, or early termination occurred if one trial failed after 48 hours.

For the virtual driving experiment (Figure 4), the goal biased RC-RRT$_1$-based planner found the solution in each trial with an average of 1629.62s, in which "goal biased" means choosing x_{goal} in the place of x_{random}

Figure 4: The task is to drive a car (a 9D nonlinear system with one steering input) at 72 k.p.h. recklessly through a town while avoiding collisions with buildings.

with a small probability (0.05 in this case). For the corresponding planner using a regular goal-biased RRT, we attempted fifty trials twice. In the first experiment, the first trial failed after 48 hours; in the second experiment, the eighth trial failed after 48 hours (although the first seven trials averaged 406.36s).

Figure 1 shows a trajectory design problem for a floating spacecraft (the equations of motion are in [3]). We used the bidirectional planning algorithm called RRTExtExt in [14], but replaced the RRT with RC-RRT$_1$. The new planner solved the problem fifty times with an average of 8059.31s. We attempted two trials using the original RRTExtExt planner; they each failed after 48 hours. Thus, the RC-RRT offers considerable reliability in solving very challenging trajectory design problems.

7 Conclusion

In this paper, based on the accessibility graph and Lipschitz conditions, resolution complete RRTs were presented, analyzed, and shown to give good experimental performance on challenging trajectory design problems. The new planners can solve challenging problems quickly, while offering the reliable advantages of systematic search techniques in the worst case. Additional effort is required in future work to analyze the effects of other errors, such as those due to numerical integration. Furthermore, we hope to implement and evaluate RC-RRT$_2$ and RC-RRT$_3$ in practice.

Acknowledgments We thank Jim Bernard his help with vehicle dynamics and Andrew Olson for constructing the spacecraft example. This work was funded in part by NSF CAREER IRI-9875304 and NSF IIS-0118146.

References

[1] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[2] D. P. Bertsekas. Convergence in discretization procedures in dynamic programming. *IEEE Trans. Autom. Control*, 20(3):415–419, June 1975.

[3] P. Cheng and S. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2001.

[4] D. F. Delchamps. Stabilizing a linear system with quantized output record. *IEEE Trans. Autom. Control*, 35(8):916–926, 1990.

[5] B. R. Donald and P. G. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14(6):443–479, 1995.

[6] B. R. Donald, P. G. Xavier, J. Canny, and J. Reif. Kinodynamic planning. *Journal of the ACM*, 40:1048–66, November 1993.

[7] E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicles motion planning. Technical Report LIDS-P-2468, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1999.

[8] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robotic manipulator: A provably good approximation algorithm. In *IEEE Int. Conf. Robot. & Autom.*, pages 150–155, Cincinnati, OH, 1990.

[9] T. Karatas and F. Bullo. Randomized searches and nonlinear programming in trajectory planning. In *IEEE Conference on Decision and Control*, 2001.

[10] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *IEEE Int. Conf. Robot. & Autom.*, 2000.

[11] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[12] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University., Oct. 1998.

[13] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[14] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[15] S. M. LaValle and P. Konkimalla. Algorithms for computing numerical optimal feedback motion strategies. *Int. J. Robot. Res.* 20(9):729-752, 2001.

[16] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. Robot. Res.*, 15(6):533–556, 1996.

[17] A. Marigo and B. Piccoli and A. Bicchi Reachability Analysis for a Class of Quantized Control Systems. In *Proc. IEEE Conf. on Decision and Control*, 2000.

[18] J. Reif and H. Wang. Non-uniform discretization approximations for kinodynamic motion planning. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 97–112. A K Peters, Wellesley, MA, 1997.

[19] G. J. Toussaint, T. Başar, and F. Bullo. Motion planning for nonlinear underactuated vehicles using hinfinity techniques. Coordinated Science Lab, University of Illinois, September 2000.

[20] C. Yap and T. Dubé. *The exact computation paradigm.* World Scientific Press, 1995.