Archives of Control Sciences
Volume 11(XLVII), 2001
No. 3–4, *pages 51–78*

RRT-Based Trajectory Design for Autonomous Automobiles and Spacecraft

PENG CHENG, ZUOJUN SHEN and STEVEN M. LAVALLE

This paper considers the design of open loop control laws for nonlinear systems that are subject to nonconvex state space constraints. The focus is on finding feasible trajectories for two challenging sets of nonlinear systems: 1) the determination of automobile trajectories through obstacle courses; 2) the design of re-entry trajectories for a reusable launch vehicle model based on the NASA X33 prototype. Our algorithms are based on Rapidly-exploring Random Trees (RRTs) that incrementally explore the state space while satisfying both the global constraints imposed by obstacles and velocity bounds and differential constraints imposed by the equations of motion. The method is particularly suited for high-dimensional problems in which classical numerical approaches such as dynamic programming cannot be applied in practice. Our algorithms for trajectory design have been implemented and evaluated on several challenging examples, which are presented in the paper.

Key words: trajectory planning, motion planning, nonlinear systems, robotics, algorithms

1. Introduction

The design of feasible open-loop trajectories for nonlinear systems that have complicated state space constraints is a challenging task. Even without state space constraints, it is already very challenging to design feasible trajectories for nonlinear systems. The consideration of state space constraints combines the complexity of both classical motion planning from robotics with the challenges of nonlinear system control. If algorithms can be designed that are able to efficiently find and optimize such trajectories for broad classes of problems, many application areas would be greatly impacted, including robotics, aeronautics, and automotive design. The successful development of such

P. Cheng is with the Dept. of Computer Science, 1304 W. Springfield Ave., University of Illinois, Urbana, IL, 61801, USA. e-mail: pcheng1@cs.uiuc.edu. Z. Shen is with the Dept. of Aerospace Engineering and Engineering Mechanics, 2271 Howe Hall, Iowa State University, Ames, IA 50011, USA. e-mail: zjshen@iastate.edu. S. M. LaValle is with the Dept. of Computer Science, 1304 W. Springfield Ave., University of Illinois, Urbana, IL, 61801, USA. e-mail: lavalle@cs.uiuc.edu

This work was funded in part by NSF CAREER Award IRI-9875304 (LaValle). We thank Jim Bernard for many helpful discussions on vehicle dynamics, and his suggestions of models. We thank Ping Lu for his suggestions regarding the modeling of the X-33. We thank Francesco Bullo for helpful discussions regarding trajectory optimization.

algorithms will most likely depend on a culmination of ideas from both the algorithmic motion planning and nonlinear systems communities.

The primary interest in the motion planning community has been to compute collision free paths in the presence of complicated constraints on the configuration space of one or more movable bodies (while ignoring differential motion constraints). It is widely known that the class of problems is NP-hard [47], which has caused the focus of research in this area to move from exact, complete algorithms to randomized (or Monte-Carlo) algorithms that can solve many challenging high-dimensional problems efficiently at the expense of completeness. Within the past decade, randomized versions of earlier ideas were developed. The classic notion of a *roadmap* [11, 30, 43], is a network of collision-free paths that captures the configuration-space topology, and is generated by preprocessing the configuration space independently of any initial-goal query. This evolved into a Monte-Carlo variation termed a probabilistic roadmap (PRM) [25], which is formed by selecting numerous configurations at random, and generating a network of paths by attempting to connect nearby points. In contrast to roadmaps, classical *incremental search* ideas are based heavily on a particular initial-goal query, and include methods such as dynamic programming, A^* search, or bidirectional search. These evolved into randomized approaches such as the randomized potential field approach [4], Ariadne's clew algorithm [39], the planner in [23][1], and Rapidly-exploring Random Trees (RRTs) [28].

Once differential constraints are introduced, a challenging problem emerges that involves both nonlinear control and traditional path planning issues. This problem is often referred to as *nonholonomic planning* [5, 31, 32, 42] or kinodynamic planning [10, 13–16, 18]. Many of these methods, such as those in [5, 15], follow closely the incremental search paradigm. It is generally more challenging to design a roadmap-based algorithm due to the increased difficulty of connecting numerous pairs of states in the presence of differential constraints (often referred to as the steering problem [32]). The first randomized approach to kinodynamic planning appeared in [34], and was based on Rapidly-exploring Random Trees [33]. In that paper, RRTs were applied to trajectory design problems for hovercrafts and rigid spacecrafts that move in a cluttered 2D or 3D environment. Theoretical performance bounds are presented in [35]. In [19], RRTs were applied to the design of collision-free trajectories for a helicopter in a cluttered 3D environment. In [49], RRTs were applied to the design of trajectories for underactuated vehicles. Recently, the incremental search method in [26] was extended to the case of kinodynamic planning in time-varying environments.

Section 2 presents a brief formulation of the general problem. Section 3 introduces RRTs and RRT-based trajectory design methods. Section 4 considers designing trajectories for autonomous automobiles. Section 5 considers designing re-entry trajectories for a reusable launch vehicle. Section 6 briefly discusses trajectory optimization issues. Finally, some conclusions are presented in Section 7.

[1]We note that the method introduced here is termed a PRM by the authors. Since the method is based on incremental search, we include it here to help categorize the methods based on their conceptual similarities.

2. Problem Description

The class of problems considered in this paper can be formulated in terms of six components:

1. **State Space:** A bounded manifold, $X \subset \mathbb{R}^n$
2. **Boundary Values:** $x_{init} \in X$ and $X_{goal} \subset X$
3. **Collision Detector:** A function, $D : X \to \{true, false\}$, that determines whether global constraints are satisfied from state x. This could alternatively be a real-valued function that indicates distance from the constraint boundary.
4. **Inputs:** A set, U, which specifies the complete set of controls or actions that can affect the state.
5. **Incremental Simulator:** Given the current state, $x(t)$, and inputs applied over a time interval, $\{u(t')|t \leqslant t' \leqslant t+\Delta t\}$, the incremental simulator yields $x(t+\Delta t)$. This usually occurs through numerical integration of a state transition equation, $\dot{x} = f(x,u)$.
6. **Metric:** A real-valued function, $\rho : X \times X \to [0,\infty)$, which specifies the distance between pairs of points in X (however, ρ is not necessarily symmetric).

Trajectory planning will generally be viewed as a search in a state space, X, for a control u that brings the system from an initial state, x_{init} to a goal region $X_{goal} \subset X$ or goal state $x_{goal} \in X$. It is assumed that a complicated set of global constraints is imposed on X, and any solution path must keep the state within this set. A collision detector reports whether a given state, x, satisfies the global constraints. We generally use the notation X_{free} to refer to the set of all states that satisfy the global constraints. Local, differential constraints are imposed through the definition of a set of inputs (or controls) and an incremental simulator. Taken together, these two components specify possible changes in state. The incremental simulator can be defined by numerical integration of a *state transition equation* of the form $\dot{x} = f(x,u)$, or can simply be achieved by a simulation software package. Finally, a metric is defined to indicate the closeness of pairs of points in the state space. This metric will be used in Section 3.1, when the RRT is introduced.

3. Trajectory Planning Method

Section 3.1 introduces the RRT. Section 3.2 describes how RRTs are used to construct trajectory design algorithms. Section 3.3 summarizes results regarding the analysis of RRTs.

```
BUILD_RRT($x_{init}$)
1  $\mathcal{T}$.init($x_{init}$);
2  for $k = 1$ to $K$ do
3      $x_{rand} \leftarrow$ RANDOM_STATE();
4      EXTEND($\mathcal{T}, x_{rand}$);
5  Return $\mathcal{T}$
```

```
EXTEND($\mathcal{T}, x$)
1  $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x, \mathcal{T}$);
2  if NEW_STATE($x, x_{near}, x_{new}, u_{new}$) then
3      $\mathcal{T}$.add_vertex($x_{new}$);
4      $\mathcal{T}$.add_edge($x_{near}, x_{new}, u_{new}$);
```

Figure 1. The basic RRT construction algorithm.

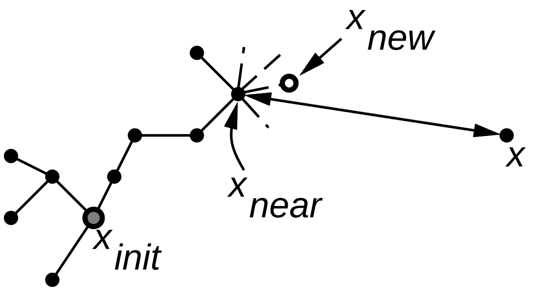

Figure 2. The EXTEND operation.

3.1. Rapidly-exploring Random Trees (RRTs)

The Rapidly-exploring Random Tree (RRT) was introduced in [33] as an exploration algorithm for quickly searching high-dimensional spaces that have both global constraints (arising from workspace obstacles and velocity bounds) and differential constraints (arising from kinematics and dynamics). The key idea is to bias the exploration toward unexplored portions of the space by randomly sampling points in the state space, and incrementally “pulling” the search tree toward them.

The basic RRT construction algorithm is given in Figure 1. A simple iteration is performed in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly-selected state, $x \in X$. The EXTEND function, illustrated in Figure 2, selects the nearest vertex already in the RRT to x. The “nearest” vertex is chosen according to the metric, ρ. The function NEW_STATE makes a motion toward x by applying an input $u \in U$ for some time increment Δt. This input can be chosen at random, or selected by trying all possible inputs and choosing the one that yields a new

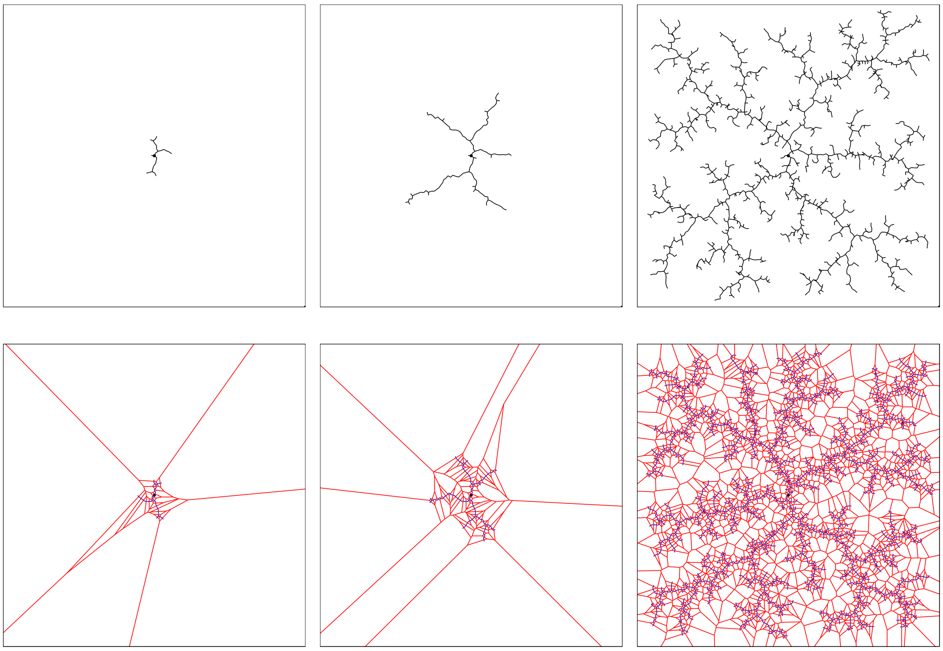

Figure 3. The RRT rapidly explores in the beginning, before converging to the sampling distribution. Below each frame, the corresponding Voronoi regions are shown to indicate the exploration bias.

state as close as possible to the sample, x (if U is infinite, then a finite approximation or analytical technique can be used). NEW_STATE implicitly uses the collision detection function to determine whether the new state (and all intermediate states) satisfy the global constraints. For many problems, this can be performed quickly ("almost constant time") using incremental distance computation algorithms [21, 37, 41] by storing the relevant invariants with each of the RRT vertices. If NEW_STATE is successful, the new state and input are represented in x_{new} and u_{new}, respectively. The left column of Figure 3 shows an RRT grown from the center of a square region in the plane. In this example, there are no differential constraints (motion in any direction is possible from any point). The incremental construction method biases the RRT to rapidly explore in the beginning, and then converge to a uniform coverage of the space [35]. Note that the exploration is naturally biased towards vertices that have larger Voronoi regions. This causes the exploration to occur mostly on the unexplored portion of the state space.

3.2. Incremental Search Algorithms

Section 3.1 introduced the basic RRT and its exploration properties. Now the focus is on developing path planners using RRTs. We generally consider the RRT as a building block that can be used to construct an efficient planner, as opposed to a path planning algorithm by itself. For example, one might use an RRT to escape local minima in a randomized potential field path planner. In [50], an RRT was used as the local planner for the probabilistic roadmap planner. We present several alternative RRT-based planners in this section. The recommended choice depends on several factors, such as whether differential constraints exist, the type of collision detection algorithm, or the efficiency of nearest neighbor computations.

3.2.1. Single-RRT Planners

In principle, the basic RRT can be used in isolation as a path planner because its vertices will eventually cover a connected component of X_{free}, coming arbitrarily close to any specified x_{goal}. The problem is that without any bias toward the goal, convergence might be slow. An improved planner, called RRT-GoalBias, can be obtained by replacing RANDOM_STATE in Figure 2 with a function that tosses a biased coin to determine what should be returned. If the coin toss yields "heads", then x_{goal} is returned; otherwise, a random state is returned. Even with a small probability of returning heads (such as 0.05), RRT-GoalBias usually converges to the goal much faster than the basic RRT. If too much bias is introduced; however, the planner begins to behave like a randomized potential field planner that is trapped in a local minimum. An improvement called RRT-GoalZoom replaces RANDOM_STATE with a decision, based on a biased coin toss, that chooses a random sample from either a region around the goal or the whole state space. The size of the region around the goal is controlled by the closest RRT vertex to the goal at any iteration. The effect is that the focus of samples gradually increases around the goal as the RRT draws nearer. This planner has performed quite well in practice; however, it is still possible that performance is degraded due to local minima. In general, it seems best to replace RANDOM_STATE with a sampling scheme that draws states from a nonuniform probability density function that has a "gradual" bias toward the goal. There are still many interesting research issues regarding the problem of sampling. It might be possible to use some of the sampling methods that were proposed to improve the performance of probabilistic roadmaps [1, 8].

One more issue to consider is the size of the step that is used for RRT construction. This could be chosen dynamically during execution on the basis of a distance computation function that is used for collision detection. If the bodies are far from colliding, then larger steps can be taken. Aside from following this idea to obtain an incremental step, how far should the new state, x_{new} appear from x_{near}? Should we try to connect x_{near} to x_{rand}? Instead of attempting to extend an RRT by an incremental step, EXTEND can be iterated until the random state or an obstacle is reached, as shown in the CONNECT algorithm description in Figure 4. CONNECT can replace EXTEND, yielding an RRT that grows very quickly, if permitted by collision detection constraints and the differential constraints. One of the key advantages of the CONNECT function is that a long path can be constructed with only a single call to the NEAREST_NEIGHBOR algorithm. This advantage motivates the choice of a greedier algorithm; however, if an efficient nearest-neighbor algorithm [2, 3, 20, 24, 27, 40, 44, 48, 51] is used, as opposed to the obvious linear-time method, then it might make sense to be less greedy. After performing dozens of experiments on a variety of problems, we have found CONNECT to yield the best performance for holonomic planning problems, and EXTEND seems

```
CONNECT($\mathcal{T}$, $x$)
1  repeat
2      $S \leftarrow$ EXTEND($\mathcal{T}$, $x$);
3  until not ($S = Advanced$)
4  Return $S$;
```

Figure 4. The CONNECT function.

```
RRT_BIDIRECTIONAL($x_{init}$, $x_{goal}$)
1  $\mathcal{T}_a$.init($x_{init}$); $\mathcal{T}_b$.init($x_{goal}$);
2  for $k = 1$ to $K$ do
3      $x_{rand} \leftarrow$ RANDOM_STATE();
4      if not (EXTEND($\mathcal{T}_a$, $x_{rand}$) $= Trapped$) then
5          if (EXTEND($\mathcal{T}_b$, $x_{new}$) $= Reached$) then
6              Return PATH($\mathcal{T}_a$, $\mathcal{T}_b$);
7      SWAP($\mathcal{T}_a$, $\mathcal{T}_b$);
8  Return Failure
```

Figure 5. A bidirectional RRT-based planner.

to be the best for nonholonomic problems. One reason for this difference is that CONNECT places more faith in the metric, and for nonholonomic problems it becomes more challenging to design good metrics.

3.2.2. Bidirectional Planners

Inspired by classical bidirectional search techniques [45], it seems reasonable to expect that improved performance can be obtained by growing two RRTs, one from x_{init} and the other from x_{goal}; a solution is found if the two RRTs meet. For a simple grid search, it is straightforward to implement a bidirectional search; however, RRT construction must be biased to ensure that the trees meet well before covering the entire space, and to allow efficient detection of meeting.

Figure 5 shows the RRT_BIDIRECTIONAL algorithm, which may be compared to the BUILD_RRT algorithm of Figure 1. RRT_BIDIRECTIONAL divides the computation time between two processes: 1) exploring the state space; 2) trying to grow the trees into each other. Two trees, $\mathcal{T}_a$ and $\mathcal{T}_b$ are maintained at all times until they become connected and a solution is found. In each iteration, one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex. Then, the roles are reversed by swapping the two trees. Growth of two RRTs was also proposed in [34] for kinodynamic planning; however, in each iteration both trees were incrementally extended toward a random state. The current algorithm attempts to grow the trees into each other half of the time, which has been found to yield much better performance.

Several variations of the above planner can also be considered. Either occurrence of EXTEND may be replaced by CONNECT in RRT_BIDIRECTIONAL. Each replacement makes the operation more aggressive. If the EXTEND in Line 4 is replaced with CONNECT, then the planner aggressively explores the state space, with the same tradeoffs that existed for the single-RRT planner. If the EXTEND in Line 5 is replaced with CONNECT, the planner aggressively attempts to connect the two trees in each iteration. This particular variant was very successful at solving holonomic planning problems. For convenience, we refer to this variant as RRT-ExtCon, and the original bidirectional algorithm as RRT-ExtExt. Among the variants discussed thus far, we have found RRT-ExtCon to be most successful for holonomic planning [28], and RRT-ExtExt to be best for nonholonomic problems. The most aggressive planner can be constructed by replacing EXTEND with CONNECT in both Lines 4 and 5, to yield RRT-ConCon. We are currently evaluating the performance of this variant.

Through extensive experimentation over a wide variety of examples, we have concluded that, when applicable, the bidirectional approach is much more efficient than a single RRT approach. One shortcoming of using the bidirectional approach for nonholonomic and kinodynamic planning problems is the need to make a connection between a pair of vertices, one from each RRT. For a planning problem that involves reaching a goal region from an initial state, no connections are necessary using a single-RRT approach. The gaps between the two trajectories can be closed in practice by applying steering methods [32], if possible, or classical shooting methods [7], which are often used for two-point boundary value problems.

3.2.3. Other Approaches

If a dual-tree approach offers advantages over a single tree, then it is natural to ask whether growing three or more RRTs might be even better. These additional RRTs could be started at random states. Of course, the connection problem will become more difficult for nonholonomic problems. Also, as more trees are considered, a complicated decision problem arises. The computation time must be divided between attempting to explore the space and attempting to connect RRTs to each other. It is also not clear which connections should be attempted. Many research issues remain in the development of this and other RRT-based planners.

It is interesting to consider the limiting case in which a new RRT is started for every random sample, x_{rand}. Once the single-vertex RRT is generated, the CONNECT function from Figure 4 can be applied to every other RRT. To improve performance, one might only consider connections to vertices that are within a fixed distance of x_{rand}, according to the metric. If a connection succeeds, then the two RRTs are merged into a single graph. The resulting algorithm simulates the behavior of the probabilistic roadmap approach to path planning [25]. Thus, the probabilistic roadmap can be considered as an extreme version of an RRT-based algorithm in which a maximum number of separate RRTs are constructed and merged.

3.3. Analysis Results

Several analytical properties of RRTs have been established in [35,36]. A summary of these results is given here. First, consider the case of an RRT in which there are no differential constraints (i.e., $\dot{x} = u$). For simplicity, assume that the planner consists of a single RRT. The bidirectional planner is only slightly better in terms of the analysis, and a single RRT is easier to analyze. Furthermore, assume that the step size is large enough so that the planner always attempts to connect x_{near} to x_{rand}.

3.3.1. The limiting distribution of vertices

Let $D_k(x)$ denote a random variable whose value is the distance of x to the closest vertex in G, in which k is the number of vertices in an RRT. Let d_k denote the value of D_k. Let ε denote the incremental distance traveled in the EXTEND procedure (the RRT step size).

Theorem 1 *Suppose x_{init} and x_{goal} lie in the same connected component of a nonconvex, bounded, open, n-dimensional connected component of an n-dimensional state space. The probability that an RRT constructed from x_{init} will find a path to x_{goal} approaches one as the number of RRT vertices approaches infinity.*

This establishes probabilistic completeness, as considered in [25], of the basic RRT.

The next step is to characterize the limiting distribution of the RRT vertices. Let $\mathbf{X}$ denote a vector-valued random variable that represents the sampling process used to construct an RRT. This reflects the distribution of samples that are returned by the RANDOM_STATE function in the EXTEND algorithm. Usually, $\mathbf{X}$ is characterized by a uniform probability density function over X_{free}; however, we will allow $\mathbf{X}$ to be characterized by any smooth probability density function. Let $\mathbf{X}_k$ denote a vector-valued random variable that represents the distribution of the RRT vertices.

Theorem 2 $\mathbf{X}_k$ *converges to* $\mathbf{X}$ *in probability.*

3.3.2. Convergence for trajectory design

We now consider the more general case. Suppose that motions obtained from the incremental simulator are locally constrained. For example, they might arise by integrating $\dot{x} = f(x,u)$ over some time Δt. Suppose that the number of inputs to the incremental simulator is finite, Δt is constant, no two RRT vertices lie within a specified $\varepsilon > 0$ of each other according to ρ, and that EXTEND chooses the input at random. It may be possible eventually to remove some of these restrictions; however, we have not yet pursued this route. Suppose x_{init} and x_{goal} lie in the same connected component of a nonconvex, bounded, open, n-dimensional connected component of an n-dimensional state space. In addition, there exists a sequence of inputs, $u_1, u_2, \ldots, u_k$, that when applied to x_{init} yield a sequence of states, $x_{init} = x_0, x_1, x_2, \ldots, x_{k+1} = x_{goal}$. All of these states lie in the same open connected component of X_{free}.

Let $\mathcal{A} = \{A_0, A_1, \ldots, A_k\}$ be a sequence of subsets of $\mathbf{X}$, referred to as an *attraction sequence*. Let $A_0 = \{x_{init}\}$. The remaining sets must be chosen with the following rules. For each A_i in $\mathcal{A}$, there exists a *basin*, $B_i \subseteq \mathbf{X}$ such that the following hold:

1. For all $x \in A_{i-1}$, $y \in A_i$, and $z \in \mathbf{X} \setminus B_i$, the metric, ρ, yields $\rho(x,y) < \rho(x,z)$.
2. For all $x \in B_i$, there exists an m such that the sequence of inputs $\{u_1, u_2, \ldots, u_m\}$ selected by the EXTEND algorithm will bring the state into $A_i \subseteq B_i$.

Finally, it is assumed that $A_k = \mathbf{X}_{goal}$.

Each basin B_i can intuitively be considered as both a safety zone that ensures an element of B_i will be selected by the nearest neighbor query, and a potential well that attracts the state into A_i. An attraction sequence should be chosen with each A_i as large as possible and with k as small as possible. If the space contains narrow corridors, then the attraction sequence will be longer and each A_i will be smaller. The analysis indicates that the planning performance will significantly degrade in this case, which is consistent with analysis results obtained for randomized holonomic planners [22]. Note that for kinodynamic planning, the choice of metric, ρ, can also greatly affect the attraction sequence, and ultimately the performance of the algorithm.

Let p be defined as

$$p = \min_i \{\mu(A_i)/\mu(\mathbf{X}_{free})\},$$

which corresponds to a lower bound on the probability that a random state will lie in a particular A_i.

The following theorem characterizes the expected number of iterations.

Theorem 3 *If a connection sequence of length k exists, then the expected number of iterations required to connect x_{init} to X_{goal} is no more than k/p.*

The following theorem establishes that the probability of failure decreases exponentially with the number of iterations.

Theorem 4 *If an attraction sequence of length k exists, for a constant $\delta \in (0,1]$, the probability that the RRT fails to finds a path after n iterations is at most $e^{\frac{-1}{2}(np-2k)}$.*

The following establishes the probabilistic completeness of the planner.

Theorem 5 *The probability that the RRT initialized at x_{init} will contain x_{goal} as a vertex approaches one as the number of vertices approaches infinity.*

Although this convergence is probabilistic, we have recently introduced a variant of the RRT (not considered here) that yields deterministic convergence [12].

4. Trajectory Design for Automobiles

Trajectory design for automobiles is an important problem both in robotics and virtual prototyping. In robotics, algorithms are needed that can compute trajectories for autonomous vehicles in complicated environments. In the automotive industry, simulators are used extensively to evaluate vehicle performance for virtual prototyping. Most of these simulations involve close interaction between a human driver and a simulation system. The experiments presented in this section can be considered as a component that might be used in virtual prototyping of automobile designs. As opposed to requiring human interaction, the RRT-based planner serves as a kind of "virtual stunt driver" that attempts to achieve specified conditions, such as attempting to race through an obstacle course.

4.1. Kinematic nonlinear car models

The differential constraints considered in this section arise purely from kinematic considerations. Thus, the state space reduces to the configuration space (the set of all transformations that can be applied to the bodies). Section 4.2 considers a complicated car with nonlinear dynamics.

We first consider a three-dimensional car model in which the state is (x, y, θ) and the state transition equation is

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} s\cos\theta \\ s\sin\theta \\ \frac{s}{L}\tan\theta \end{pmatrix}, \tag{1}$$

in which L denotes the distance between the front and rear axles, s denotes the speed, and ϕ denotes the steering angle. It is assumed that the steering angle is bounded, $|\phi| \leqslant \phi_{max} < \frac{\pi}{2}$. The input vector is (s, ϕ). If the car travels forward only, we set $s = 1$, and obtain the Dubins car [17]. If the car can travel in forward or reverse we allow inputs $s = 1$ or $s = -1$, and obtain the Reeds-Shepp car [46]. Figure 6 shows RRTs and computed paths using a bidirectional RRT planner in a cluttered environment for both the Dubins and Reeds-Shepp cars. Our algorithms were implemented using GNU C++ and Linux on a 500Mhz PC. The images in Figures 6.a and 6.c show a two-dimensional projection of the RRT into the XY plane. Note that the RRT in Figure 6.a contains cusps which corresponds to reversals, and the RRT in 6.c contains no cusps. Figure 7 shows two examples that are more challenging. The model in (1) assumes that the steering angle may change instantaneously, which leads to discontinuous curvature in the path. The model can be extended to yield a four-dimensional state space, in which each state is represented as (x, y, ϕ, θ) and the following state transition equation appears:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} s\cos\theta \\ s\sin\theta \\ \omega \\ \frac{s}{L}\tan\phi \end{pmatrix}.$$

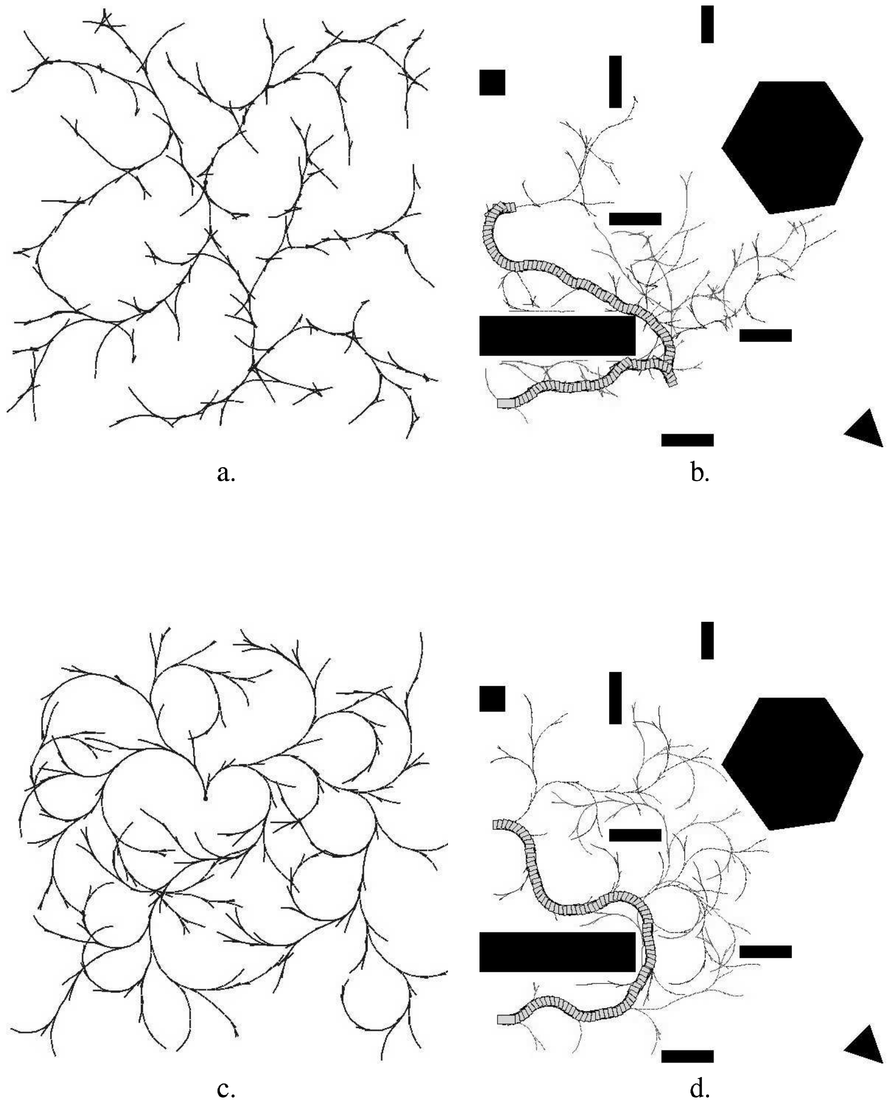

Figure 6. a) An RRT for the Reeds-Shepp car; b) a computed path using the Reeds-Shepp car; c) an RRT for the Dubins car; d) a computed path for Dubins car.

The new input ω represents a change in steering angle. From the results in Figure 7.a-b it can be observed that the paths are smooth. Figures 7.c-d shows results for a Dubins car that has an additional restriction: it must always turn left! As shown in Figure 7.d, RRTs can be used to construct solutions in very complicated environments.

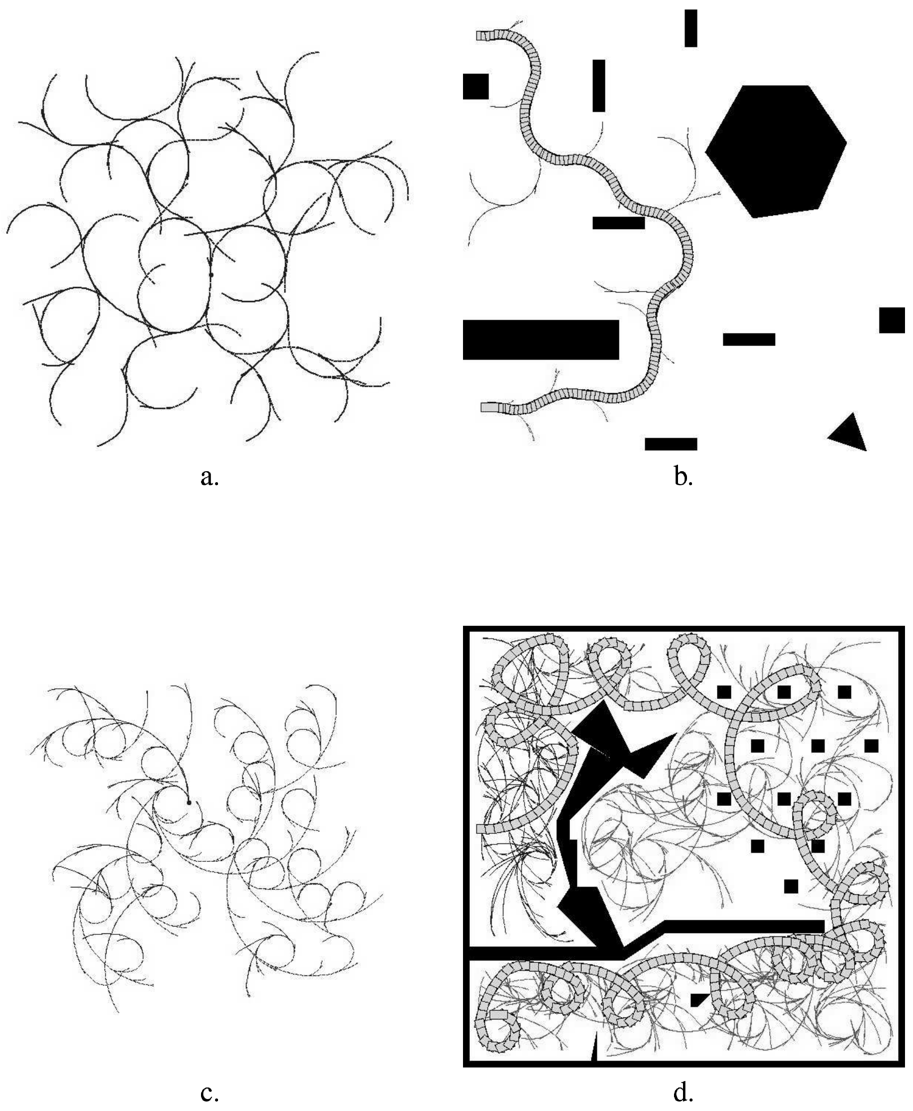

Figure 7. a) An RRT for the smooth car; b) a computed path using the smooth car; c) an RRT for the turn-left-only car; d) a computed path for the turn-left-only car.

Consider constructing a feasible solution to the problem of parking a car that pulls trailers as considered in [42]:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{\theta}_0 \\ \vdots \\ \dot{\theta}_i \\ \vdots \end{pmatrix} = \begin{pmatrix} s\cos\theta \\ s\sin\theta \\ \omega \\ \frac{s}{L}\tan\phi \\ \vdots \\ \frac{s}{d_i}\left(\prod_{j=1}^{i-1}\cos(\theta_{j-1}-\theta_j)\right)\sin(\theta_{i-1}-\theta_i) \\ \vdots \end{pmatrix},$$

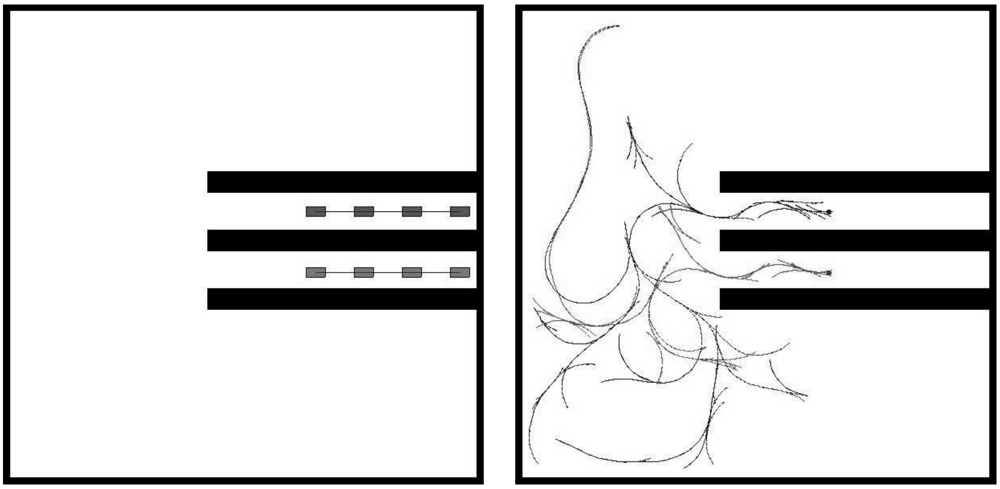

Figure 8. The smooth car pulling three trailers: a) the goal is to move from the upper stall to the lower one; b) the computed RRTs.

in which θ_0 is the orientation of the car, θ_i is the orientation of the i^{th} trailer, and d_i is the distance from the i^{th} trailer wheel axle to the hitch point. Figure 8.a shows an example that involves three trailers, resulting in a seven-dimensional state space. The computed RRTs using a bidirectional approach are shown in Figure 8.b. The solution path is illustrated in Figure 9.

4.2. A complicated nonlinear model

We next consider the more challenging case of trajectory design problems that involve nonlinear vehicle dynamics. The nine-dimensional model used here (and others we have used in our experiments) are minor variations of the models considered in [6].

4.2.1. Variables and constants

The nine-dimensional state vector is $(x, y, r, \psi, \phi, q, \nu, s, \beta)$. The 3D coordinate frame is designed with the x coordinate increasing from left to right, the y coordinate increasing from top to bottom, and the z coordinate inward (to form a right-handed coordinate system). Let β be the steering angle. Let a and b be the distance from the front and rear axles to the car center, respectively. Let ψ be the yaw angle of the car. Let s be the forward speed of the car, and let ν be the sideways speed (arising from slipping). Let r be the angular velocity. Let ϕ be the roll (which describes the sideways tilting of the car). Let q be the roll angle rate. Let α_f and α_r be the slipping angle of the front and rear wheels, respectively. These are expressed as

$$\alpha_f = \frac{\nu + ar}{s} - \beta$$

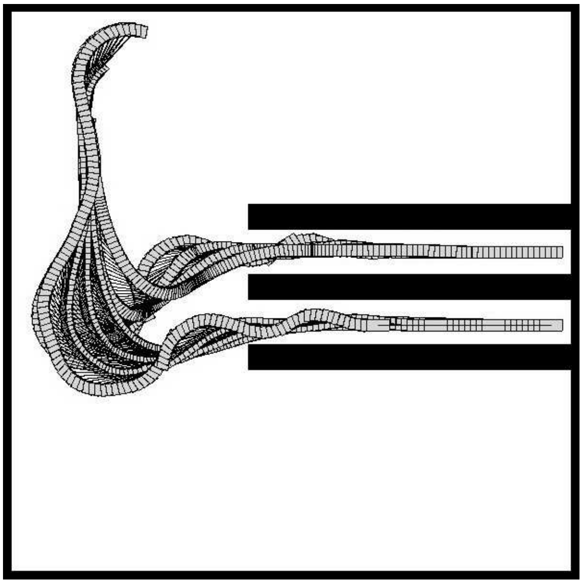

Figure 9. The resulting solution for the seven-dimensional nonholonomic system.

and

$$\alpha_r = \frac{\nu - br}{s}.$$

Let $C_{\alpha f}$ and $C_{\alpha r}$ be the cornering stiffness between the forces along the y axis, F_{yf} and F_{yr}, respectively, on the front and rear wheels. Under some conditions, it is possible for the car to slip sideways. If $N_f\mu/2 > C_{\alpha f}\tan(|\alpha_f|)$, the calculated friction force is less than the maximum possible friction, then $F_{yf} = -C_{\alpha f}\alpha_f$; otherwise, $F_{yf} = \mu N_f Sgn(\alpha_f)(1 - x_f/2)$, in which Sgn denotes the sign function, μ is a constant, and $x_f = N_f\mu/2C_{\alpha f}\tan(|\alpha_f|)$. Similarly, if $N_r\mu/2 > C_{\alpha r}\tan(|\alpha_r|)$, then $F_{yr} = -C_{\alpha r}\alpha_r$; otherwise, $F_{yr} = \mu N_r Sgn(\alpha_r)(1 - x_r/2)$, in which $x_r = N_f\mu/2C_{\alpha r}\tan(|\alpha_r|)$. Let M be the car mass, and let I be the yaw moment of inertia. Let H_2 be the distance from the joint connecting the chassis with the car frame (the chassis and frame are flexibly attached to model a simple suspension system). The constants K, c, and other details are described an in [6].

4.2.2. Equations of motion

In the equations, let $h = (-(K - MgH_2)\phi - cq - (F_{yf} + F_{yr})H_2)/I$. The following represent the nine equations of motion: $\dot{x} = s\cos\psi - \nu\sin\psi$, $\dot{y} = s\sin\psi + \nu\cos\psi$, $\dot{r} = (F_{yf}a - F_{yr}b)/I$, $\dot{\psi} = r$, $\dot{\phi} = q$, $\dot{q} = h$, $\dot{\nu} = (F_{yf} + F_{yr})/M - sr - H_2h$, $\dot{s} = u_1$, $\dot{\beta} = u_2$.

The inputs are u_1, which is linear acceleration, and u_2, which is the rate of change in the steering angle.

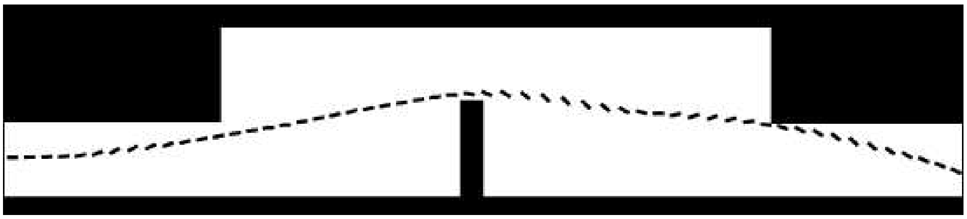

Figure 10. A double lane change at high speed.

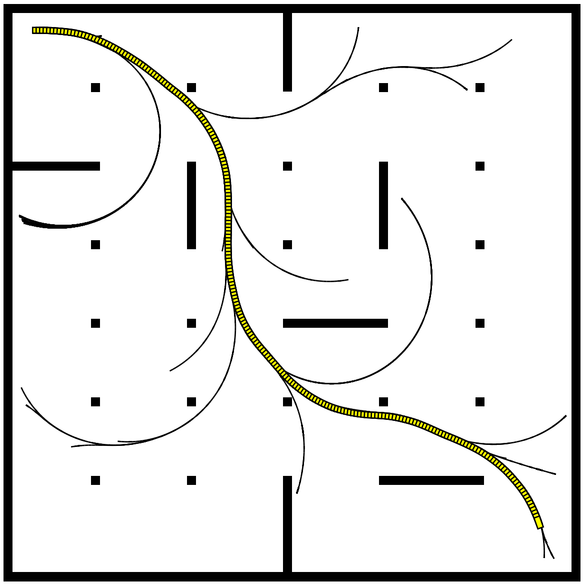

Figure 11. Fast driving through an obstacle course.

4.2.3. Experiments

Figures 10-11 show three separate experiments that were performed for the car model using a simple RRT with a goal bias (the goal was chosen with probability 1/20, instead of a random sample). Figure 10 involves a lane-changing problem, which is referred to in the automotive industry as the Consumer Union Short Course. The car is moving at 60 m.p.h. from left to right, and planner generates a steering input that avoids collision. Figure 11 shows an example in which a car must travel at 108 k.p.h. through an obstacle course. To further complicate the problem, the load on all four tires was computed using the models in [6], and a state constraint was defined that prevents any tire from lifting off of the ground. In Figure 12, the car starts at 18 k.p.h., and is requested to reach a state that causes a tire to lift from the ground. The car executes a spiraling path to generate sufficient speed, and then turns while going 190 k.p.h. to achieve the goal. Such a test could be used, for example, in a vehicle safety study.

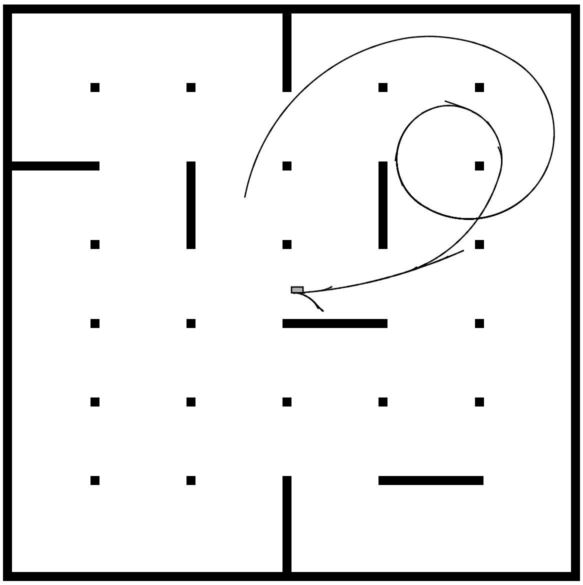

Figure 12. Forcing the tires to lift by accumulating speed.

5. Trajectory Design for a Reusable Launch Vehicle

Autonomous control of spacecraft represents a fundamental challenge in the design of systems for space flight. One of the current goals in space transportation is to design a new generation of launch vehicles to dramatically lower the costs of putting payloads in space. Instead of the complex systems in use today, the emphasis is being placed on a simple, fully reusable vehicle. Since July 1996, NASA has commissioned Lockheed Martin Skunk Works to design, build and test the X-33 experimental vehicle, which is depicted in Figure 13. Recently, the X-33 missions have been canceled; however, other similar prototypes are being evaluated. The purpose of the vehicle is to serve as a prototype for demonstrating the feasibility of the new concept. The craft is capable of reaching speeds beyond Mach 13. A formidable challenge is to design a trajectory that will guide the X-33 safely to earth. Skilled engineers sometimes spend weeks designing good trajectories for this system. In this section, we describe our early, but promising results in the design of trajectories for the X-33.

5.0.4. Variables and constants

Our model is a minor variation of the one presented in [38]. The equations of motion will be expressed in terms of dimensionless variables. In the basic model there are six state variables, $r, \theta, \phi, V, \gamma$, and ψ. Let r be the radial distance from the center of the earth to the flying spacecraft, normalized by the radius of the earth, $R_0 = 6378km$. The longitude and latitude are θ and ϕ, respectively. Derivatives of these variables will be taken with respect to dimensionless time τ, in which $\tau = \frac{t}{\sqrt{R_0 g_0}}$ and $g_0 = 9.81m/sec^2$. Let V be

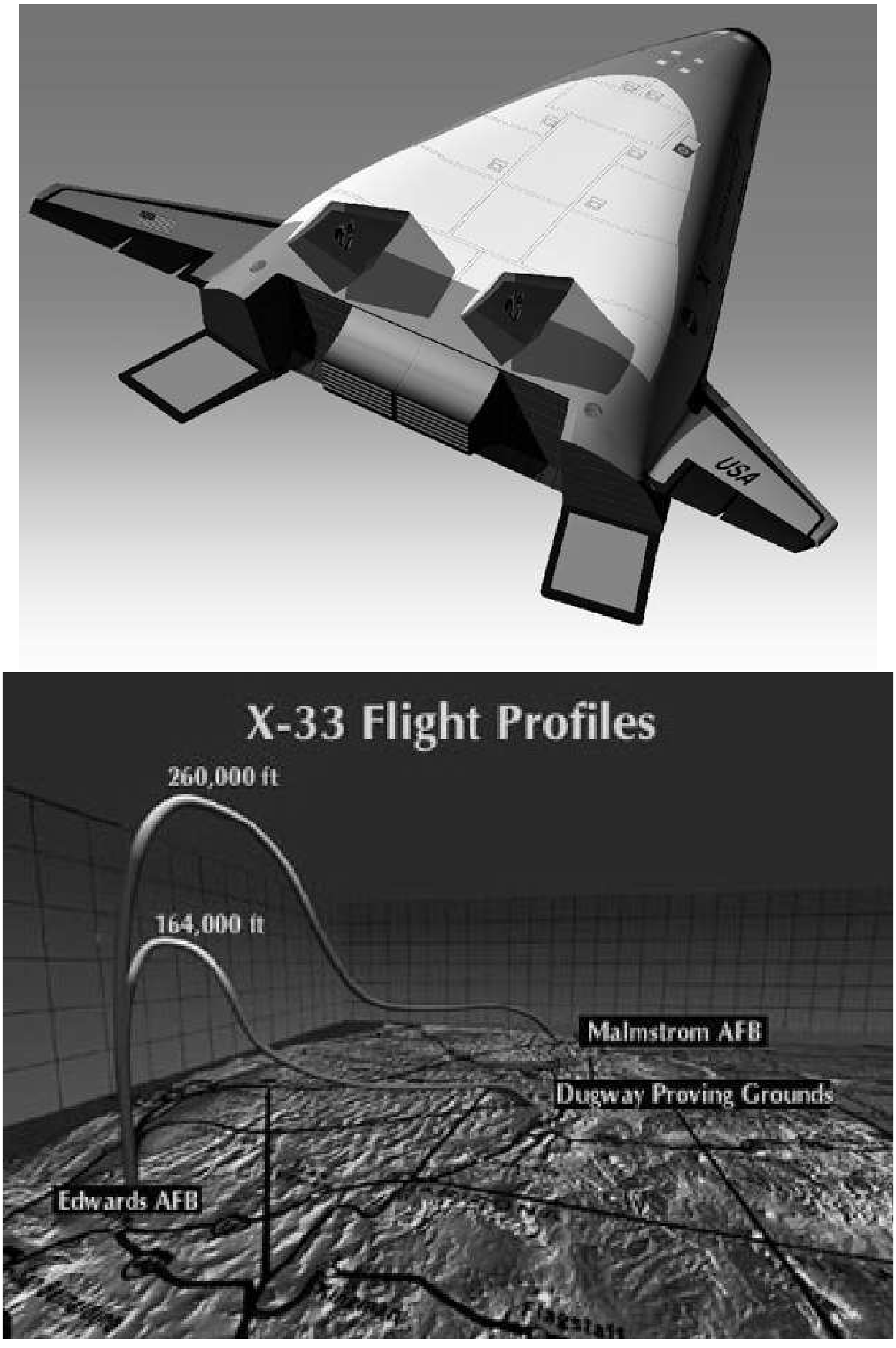

Figure 13. The NASA X-33 prototype reusable launch vehicle and possible re-entry profiles.

the earth-relative velocity, normalized by $\sqrt{R_0 g_0}$. Let γ be the flight path angle measured from horizon surface downward. Let ψ be the velocity azimuth angle, measured from the north in a clockwise direction.

The equations of motions will include the variables D, L, and Ω, which are all functions of state. The quantities D and L are aerodynamic accelerations in g's, which are expressed as

$$L = \frac{1}{2}\rho V^2 S C_L \tag{2}$$

$$D = \frac{1}{2}\rho V^2 S C_D, \tag{3}$$

in which S is a constant based on surface area, and $C_L = h_1(M, \alpha)$ and $C_D = h_2(M, \alpha)$ are the lift and drag coefficients, respectively. These are each functions of Mach number, M, and the angle of attack, α. The quantity ρ is the air density, which is a function of r. The quantity α could be pre-scheduled to be a function of M, or α could alternatively be used as a control variable. For simplicity, we assume it is a function of M. The functions, h_1 and h_2 are defined using table lookup and interpolation.

Let Ω be the rotation rate of the earth normalized by $\sqrt{R_0 g_0}$. Let σ be the bank angle, which can be considered as the principle control input of the re-entry flight (the actual control surface inputs on the craft are computed in terms of bank angle).

5.0.5. Equations of motion

The dimensionless equations of three dimensional motion over a spherical rotating earth are:

$$\dot{r} = V \sin\gamma \tag{4}$$

$$\dot{\theta} = \frac{V \cos\gamma \sin\psi}{r \cos\phi} \tag{5}$$

$$\dot{\phi} = \frac{V \cos\gamma \cos\psi}{r} \tag{6}$$

$$\dot{V} = -D - \left(\frac{\sin\gamma}{r^2}\right) + \Omega^2 r \cos\phi(\sin\gamma\cos\phi - \cos\gamma\sin\phi\cos\psi) \tag{7}$$

$$\begin{aligned}\dot{\gamma} = \frac{1}{V}\Big\{ & L\cos\sigma + \left(V^2 - \frac{1}{r}\right)\left(\frac{\cos\gamma}{r}\right) + \\ & 2\Omega V \cos\phi \sin\psi + \Omega^2 r \cos\phi(\cos\gamma\cos\phi - \sin\gamma\cos\psi\sin\phi)\Big\}\end{aligned} \tag{8}$$

$$\dot{\psi} = \frac{1}{V}\left[\frac{L\sin\sigma}{\cos\gamma} + \frac{V^2}{r}\cos\gamma\sin\psi\tan\phi \right.$$
$$\left. -2\Omega V(\tan\gamma\cos\psi\cos\phi - \sin\phi) + \frac{\Omega^2}{\cos\gamma}\sin\psi\sin\phi\cos\phi\right]. \quad (9)$$

5.0.6. Boundary conditions

Initial state is given by orbit flight conditions. The goal state is termed the TAEM (Terminal Area Energy Management) point, which is:

$$r(\tau_f) = r_f, \quad \theta(\tau_f) = \theta_f, \quad \phi(\tau_f) = \phi_f \quad (10)$$

$$V(\tau_f) = V_f, \quad \gamma_{min} \leqslant \gamma(\tau_f) \leqslant \gamma_{max}, \quad \psi(\tau_f) = \psi_f \quad (11)$$

5.0.7. State constraints

The following constraints define X_{free}. Although the X-33 is not in "collision" in the physical sense, failure to satisfy these state-space constraints will result in a collision that must be reported by the collision detector, as described in Section 2.

Normal load constraint:

$$|L\cos\alpha + D\sin\alpha| \leqslant n_{z_{max}} \quad (12)$$

Dynamic pressure constraint:

$$q \leqslant q_{max} \quad (13)$$

Heat rate constraint:

$$\dot{Q}_s \leqslant \dot{Q}_{max} \quad (14)$$

Equilibrium glide constraint:

$$\left[\frac{1}{r} - V^2\right]\left(\frac{1}{r}\right) - L \leqslant 0 \quad (15)$$

Both $\dot{Q}_s$ and q are complicated functions of state. A slice of the resulting constraints is shown in Figure 14.

5.0.8. Varying models

We have performed experiments with several variations of the model. Each is progressively more complicated than the previous one, due to a difference in the input.

A The state is $(r, \theta, \phi, V, \gamma, \psi)$, and the input is the bank angle, σ.

B The state is $(r, \theta, \phi, V, \gamma, \psi, \sigma)$, and the input is bank angle velocity, $\dot{\sigma}$.

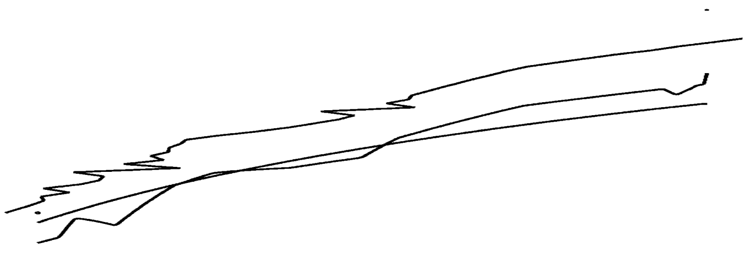

Figure 14. The flight corridor for the X-33 reusable launch vehicle (altitude vs. speed).

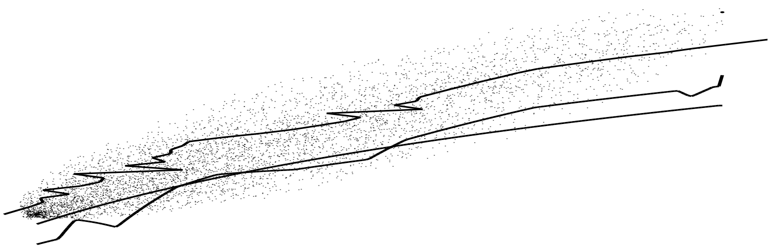

Figure 15. Biased sampling along the flight corridor.

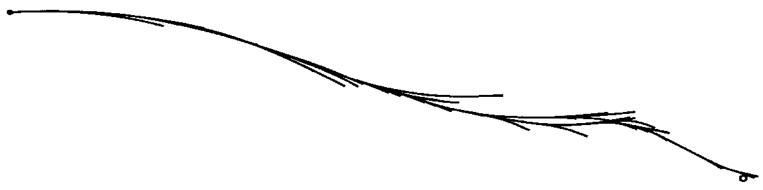

Figure 16. Altitude vs. range of the designed trajectory.

C The state is $(r, \theta, \phi, V, \gamma, \psi, \sigma, \nu)$, and the input is bank angle acceleration, $\ddot{\sigma}$. The new state variable is defined as $\nu = \dot{\sigma}$.

Model A allows instantaneous setting of the bank angle, which is unrealistic in practice. Model B allows the bank angle velocity to be changed. Model C is the most realistic (and most difficult), because only the acceleration of the bank angle is provided by the input. The limits on the bank angle derivatives are $|\dot{\sigma}| \leqslant 10^o/sec$ and $|\ddot{\sigma}| \leqslant 5^o/sec^2$.

5.0.9. Experiments

In our experiments, we have experienced considerable success design trajectories using RRT-based planners on Models A and B. In both cases we are able to reach the TAEM. Currently, we are experimenting with Model C. Figures 14-19 show a com-

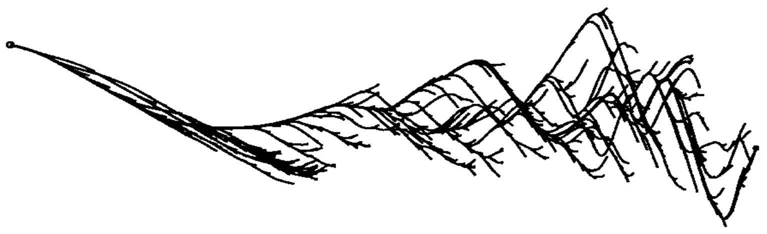
Figure 17. Heading angle vs. range of the designed trajectory.

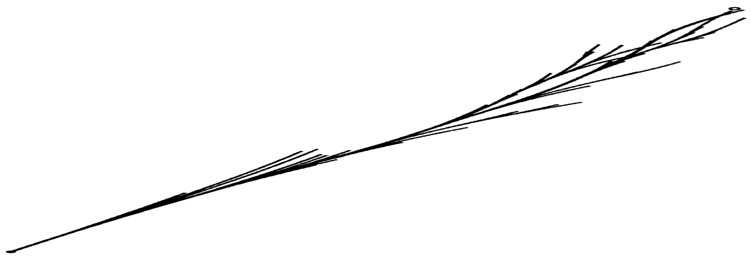
Figure 18. Latitude vs. longitude of the designed trajectory.

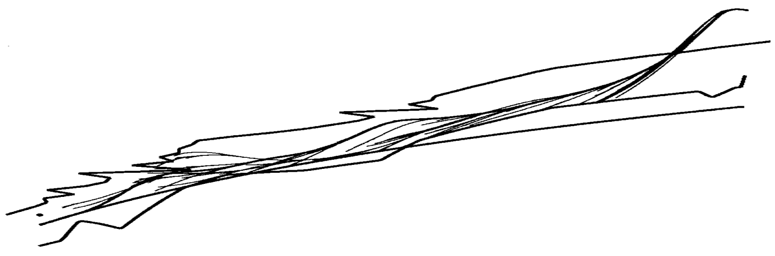
Figure 19. Altitude vs. speed of the designed trajectory.

puted trajectory for Model B. The initial state is $r = 1.0086$, $\theta = 4.257$, $\phi = 0.6400$, $V = 0.3630$, $\gamma = 0.01098$, $\psi = 0.6138$, $\sigma = 0.0$. The goal state is $r = 1.0045$, $\theta = 4.305$, $\phi = 0.6967$, $V = 0.1156$, $\gamma = -0.1062$, $\psi = 0.7522$, $\sigma = -0.2250$. A weighted Euclidean metric was used in the RRT algorithm. Figure 14 shows the projection of the state space constraints, plotted as speed vs. range. The state must remain below the jagged upper curve, and above both of the two lower curves (although the initial state may not initially be below the upper curve). The initial and goal states are indicated with black dots on the right and left of the figure, respectively. Figure 15 shows a projection of the random samples that were used in the RRT. Instead of using uniform random samples, the samples were concentrated in an elliptical region based on the flight corridor, with some bias towards the goal state. Figures 16 - 19 show the paths generated by the RRT. In each case, the initial and goal states are shown as dots. The resulting trajectory comes sufficiently close to the goal state in each of the perspectives shown.

6. Path Smoothing and Optimization

Although randomization is helpful in the design of feasible trajectories, it often leads to jagged paths or useless fluctuations in the inputs. This motivates our investigation into the problem of refining trajectories that are computed by our planning algorithms. Existing optimization methods, such as first-order gradient descent [9], are usually not designed to handle complicated, implicit state-space constraints that usually emerge in motion planning, and also make differentiability assumptions on the model (which would might prohibit their use in the models used in Sections 4 and 5. A recent survey of trajectory optimization techniques appears in [7].

We are currently exploring the use of randomization to refine trajectories in the presence of state space constraints. Some experiments are shown here for the Dubins car. In this model, there are three state variables (position and orientation). The car moves forward-only at constant speed; the only input is the steering angle. The maximum steering angle induces a bound on the curvature of the trajectory.

Two methods are currently being evaluated. For the first method, consider the classical first-order gradient method presented in [9]. For a given control history, $u(t)$ for $t \in [0,t_f)$, this iterative method yields a perturbation, $u(t)+\delta u(t)$, that locally (in the trajectory space) reduces a performance criterion. Eventually, convergence to a locally-optimal solution is obtained; however, the basic method becomes quickly trapped if obstacles exist. One way to avoid this problem is to iteratively select a segment of the path at random, perform the classical optimization on it, and then insert the new segment into the original path. Figure 20.a shows a path computed by an RRT-based planner, and Figure 20.b shows the resulting path after performing a few hundred iterations of the classical gradient descent method on randomly-selected path segments. We note that Dubins' shortest path curves [17, 29] could be used instead of the method in [9]; however, this approach cannot be generalized to many systems. An alternative method is to iteratively compute $\delta u(t)$, and use any perturbation $u(t)+\delta u(t)$ that satisfy collision constraints and reduce the performance criterion. This method can be particularly useful if the equations of motion are not differentiable or not even explicitly known. Figure 20.c shows an initial path, two intermediate paths, and a final optimized path that was obtained using random perturbations.

7. Discussion

The experiments illustrate the power of RRTs in the design of trajectories for challenging nonlinear systems with complicated state-space constraints. However, many

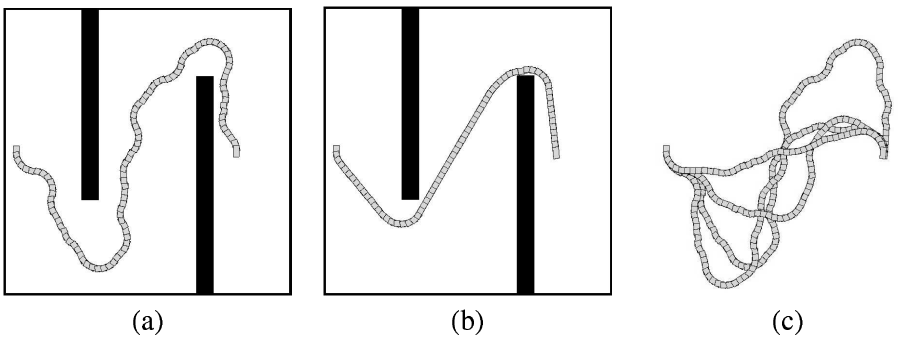

Figure 20. a) A path computed using an RRT-based planner for a Dubins car; b) a refined path obtained by iteratively optimizing random intervals using the first-order gradient method; c) path refinement by random perturbations.

challenges remain. One of the greatest difficulties in motion planning of trajectories is designing a metric that yields good performance. Some recent work on reducing metric sensitivity in RRTs appears in [12]. For some systems is may be possible to utilize cost-to-go or Lyapunov functions; however, it also appears valuable to develop randomized planning methods that have less sensitivity when presented with a poor metric. A substantial amount of work remains on the problem of trajectory optimization in the presence of obstacles. The work presented here represents a step in that direction be considering ways to combine classical optimization methods with modern algorithmic and planning ideas.

References

[1] N. M. AMATO and Y. WU: A randomized roadmap method for path and manipulation planning. *IEEE Int. Conf. Robot. & Autom.*, (1996), 113–120.

[2] S. ARYA and D. M. MOUNT: Approximate nearest neihgbor queries in fixed dimensions. *ACM-SIAM Sympos. Discrete Algorithms*, (1993), 271–280.

[3] S. ARYA, D. M. MOUNT, N. S. NETANYAHU, R. SILVERMAN, and A. Y. WU: An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, **45** (1998), 891–923.

[4] J. BARRAQUAND, B. LANGLOIS, and J. C. LATOMBE: Numerical potential field techniques for robot path planning. *IEEE Trans. Syst., Man, Cybern.*, **22**(2), (1992), 224–241.

[5] J. BARRAQUAND and J.-C. LATOMBE: Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, **10** (1993), 121–155.

[6] J. BERNARD, J. SHANNAN, and M. VANDERPLOEG: Vehicle rollover on smooth surfaces. *Proc. SAE Passenger Car Meeting and Exposition*, Dearborn, Michigan, (1989).

[7] J. T. BETTS: Survey of numerical methods for trajectory optimization. *J. of Guidance, Control, and Dynamics*, **21**(2), (1998), 193–207.

[8] V. BOOR, N. H. OVERMARS, and A. F. VAN DER STAPPEN: The gaussian sampling strategy for probabilistic roadmap planners. *IEEE Int. Conf. Robot. & Autom.*, (1999), 1018–1023.

[9] A. E. BRYSON and Y.-C. HO: Applied Optimal Control. Hemisphere Publishing Corp., New York, NY, 1975.

[10] J. CANNY, A. REGE, and J. REIF: An exact algorithm for kinodynamic planning in the plane. *Discrete and Computational Geometry*, **6** (1991), 461–484.

[11] J. F. CANNY: The Complexity of Robot Motion Planning. MIT Press, Cambridge, MA, 1988.

[12] P. CHENG and S. M. LAVALLE: Resolution complete rapidly-exploring random trees. *Submitted to IEEE Int'l Conf. on Robotics and Automation*, (2002).

[13] M. CHERIF: Kinodynamic motion planning for all-terrain wheeled vehicles. *IEEE Int. Conf. Robot. & Autom.*, (1999).

[14] C. CONNOLLY, R. GRUPEN, and K. SOUCCAR: A Hamiltonian framework for kinodynamic planning. *Proc. of the IEEE International Conf. on Robotics and Automation (ICRA'95)*, Nagoya, Japan, (1995).

[15] B. DONALD and P. XAVIER: Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, **14**(6), (1995), 443–479.

[16] B. R. DONALD, P. G. XAVIER, J. CANNY, and J. REIF: Kinodynamic planning. *Journal of the ACM*, **40** (1993), 1048–66.

[17] 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*, **79** (1957), 497–516.

[18] TH. FRAICHARD and C. LAUGIER: Kinodynamic planning in a structured and time-varying 2d workspace. *IEEE Int. Conf. Robot. & Autom.*, (1992), 1500–1505.

[19] 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.

[20] J. H. FRIEDMAN, J. L. BENTLEY, and R.A. FINKEL: An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, **3**(3), (1977), 209–226.

[21] L. J. GUIBAS, D. HSU, and L. ZHANG: H-Walk: Hierarchical distance computation for moving convex bodies. *Proc. ACM Symposium on Computational Geometry*, (1999), 265–273.

[22] D. HSU, L. E. KAVRAKI, J.-C. LATOMBE, R. MOTWANI, and S. SORKIN: On finding narrow passages with probabilistic roadmap planners. P. Agarwal, editor, *Robotics: The Algorithmic Perspective*, A.K. Peters, Wellesley, MA, 1998, 141–154.

[23] D. HSU, J.-C. LATOMBE, and R. MOTWANI: Path planning in expansive configuration spaces. *Int. J. Comput. Geom. & Appl.*, **4** (1999), 495–512.

[24] P. INDYK and R. MOTWANI: Approximate nearest neighbors: Towards removing the curse of dimensionality. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, (1998), 604–613.

[25] L. E. KAVRAKI, P. SVESTKA, J.-C. LATOMBE, and M. H. OVERMARS: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. & Autom.*, **12**(4), (1996), 566–580.

[26] R. KINDEL, D. HSU, J.-C. LATOMBE, and S. ROCK: Kinodynamic motion planning amidst moving obstacles. *IEEE Int. Conf. Robot. & Autom.*, (2000).

[27] J. M. KLEINBERG: Two algorithms for nearest-neighbor search in high dimensions. *ACM Symposium on Theory of Computing*, (1997), 599–608.

[28] J. J. KUFFNER and S. M. LAVALLE: RRT-connect: An efficient approach to single-query path planning. *Proc. IEEE Int'l Conf. on Robotics and Automation*, (2000), 95–1001.

[29] J.-C. LATOMBE: A fast path planner for a car-like indoor mobile robot. *Proc. Am. Assoc. Artif. Intell.*, (1991), 659–665.

[30] J.-C. LATOMBE.: Robot Motion Planning. Kluwer Academic Publishers, Boston, MA, 1991.

[31] J.-P. LAUMOND: Finding collision-free smooth trajectories for a non-holonomic mobile robot. *Proc. Int. Joint Conf. on Artif. Intell.*, (1987), 1120–1123.

[32] J. P. LAUMOND, S. SEKHAVAT, and F. LAMIRAUX: Guidelines in nonholonomic motion planning for mobile robots. J.-P. Laumond, editor, *Robot Motion Plannning and Control*, Springer-Verlag, Berlin, (1998), 1–53.

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

[34] S. M. LAVALLE and J. J. KUFFNER: Randomized kinodynamic planning. *Proc. IEEE Int'l Conf. on Robotics and Automation*, (1999), 473–479.

[35] S. M. LAVALLE and J. J. KUFFNER: Rapidly-exploring random trees: Progress and prospects. *Workshop on the Algorithmic Foundations of Robotics*, (2000).

[36] S. M. LAVALLE and J. J. KUFFNER: Randomized kinodynamic planning. *International Journal of Robotics Research*, **20**(5), (2001), 378–400

[37] M. C. LIN and J. F. CANNY: Efficient algorithms for incremental distance computation. *IEEE Int. Conf. Robot. & Autom.*, (1991).

[38] P. LU and J. M. HANSON: Entry guidance for the X-33 vehicle. *J. Spacecraft and Rockets*, **35**(3), (1998), 342–349.

[39] E. MAZER, G. TALBI, J. M. AHUACTZIN, and P. BESSIÈRE: The Ariadne's clew algorithm. *Proc. Int. Conf. of Society of Adaptive Behavior*, Honolulu, (1992).

[40] G. L. MILLER, S.-H. TENG, W. THURSTON, and S. A. VAVASIS: Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM*, **44**(1), (1997), 1–29.

[41] B. MIRTICH: V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electronics Research Laboratory, 1997.

[42] R. M. MURRAY and S. SASTRY: Nonholonomic motion planning: Steering using sinusoids. *Trans. Automatic Control*, **38**(5), (1993), 700–716.

[43] C. O'DUNLAING and C. K. YAP: A retraction method for planning the motion of a disc. *Journal of Algorithms*, **6** (1982), 104–111.

[44] M. H. OVERMARS and J. VAN LEEUWEN: Dynamic multidimensional data structures based on Quad- and K-D trees. *Acta Informatica*, **17** (1982), 267–285.

[45] I. POHL: Bi-directional and heuristic search in path problems. Technical report, Stanford Linear Accelerator Center, 1969.

[46] J. A. REEDS and L. A. SHEPP: Optimal paths for a car that goes both forwards and backwards. *Pacific J. Math.*, **145**(2), (1990), 367–393.

[47] J. H. REIF: Complexity of the mover's problem and generalizations. *Proc. of IEEE Symp. on Foundat. of Comp. Sci.*, (1979), 421–427.

[48] R. L SPROULL: Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, **6** (1991), 579–589.

[49] G. J. TOUSSAINT, T. BA■AR, and F. BULLO; Motion planning for nonlinear underactuated vehicles using hinfinity techniques. Coordinated Science Lab, University of Illinois, 2000.

[50] D. VALLEJO, C. JONES, and N. AMATO: An adaptive framework for "single shot" motion planning. Texas A&M, 1999.

[51] P. N. YIANILOS: Data structures and algorithms for nearest neighbor search in general metric spaces. *ACM-SIAM Symposium on Discrete Algorithms*, (1993), 311–321.