Randomized Kinodynamic Motion Planning with Moving Obstacles

David Hsu* Robert Kindel† Jean-Claude Latombe† Stephen Rock*

†*Department of Computer Science* **Department of Aeronautics & Astronautics*

Stanford University

Stanford, CA 94305, U.S.A.

Abstract

We present a randomized motion planner for robots that must avoid moving obstacles and achieve a specified goal under kinematic and dynamic constraints. The planner samples the robot's state×time space by picking control inputs at random and integrating the equations of motion. The result is a roadmap of sampled state×time points, called milestones, connected by short admissible trajectories. The planner does not precompute the roadmap as traditional probabilistic roadmap planners do; instead, for each planning query, it generates a new roadmap to find a trajectory between an initial and a goal state×time point. We prove in this paper that the probability that the planner fails to find such a trajectory when one exists quickly goes to 0, as the number of milestones grows. The planner has been implemented and tested successfully in both simulated and real environments. In the latter case, a vision module estimates obstacle motions just before planning starts; the planner is then allocated a small, fixed amount of time to compute a trajectory. If a change in the obstacle motion is detected while the robot executes the planned trajectory, the planner recomputes a trajectory on the fly.

1 Introduction

In its simplest form, motion planning is a purely geometric problem: given the geometry of a robot and static obstacles, compute a collision-free path of the robot between two specified configurations. This formulation ignores several key aspects of the physical world. Robot motions are subject to kinematic and dynamic constraints (kinodynamic constraints [DXCR93], for short) that, unlike obstacles, cannot be represented by forbidden regions in the configuration space. The environment may also contain moving obstacles requiring that computed paths be parametrized by time. In this paper, we consider motion planning problems with both kinodynamic constraints and moving obstacles. We propose an efficient randomized algorithm for this class of problems.

Our work extends the probabilistic roadmap (PRM) framework originally developed for planning collision-free geometric paths [KŠLO96]. A PRM planner samples the robot's configuration space at random and connects the generated free samples, called *milestones*, by simple local paths, typically straight-line segments in the configuration space. The result is an undirected graph called a probabilistic roadmap. Multi-query PRM planners precompute the roadmap [KŠLO96], while single-query planners compute a new roadmap for each query [HLS99]. It has been proven that, under reasonable assumptions, a small number of milestones picked uniformly at random are sufficient to capture the free space's connectivity with high probability [HLS99, KLMR98]. However, when kinodynamic constraints are present, straight local paths may violate the constraints. Moreover, allowing obstacles to move requires indexing milestones by the times when they are attained.

To address these issues, our algorithm represents both kinematic and dynamic constraints by a *control system*, which is a system of differential equations that describes all the possible movements of a robot locally. It builds a roadmap of milestones in the collision-free subset of the robot's state$\times$time space, where a state typically encodes both the configuration and the velocity of the robot (Section 3). To sample a new milestone, the algorithm first selects at random a control input from the set of admissible controls and then integrates the motion induced by this input over a short duration of time, from a previously-generated milestone. The local trajectory thus obtained automatically satisfies the motion constraints. If it does not collide with the obstacles, its endpoint is added to the roadmap as a new milestone. This iterative process yields a tree-shaped roadmap that is rooted at the initial state$\times$time point and is oriented along the time axis. The algorithm terminates when a sampled milestone falls in an "endgame" region from which it is known how to reach the goal. Unlike most traditional PRM planners, the new planner does not precompute the roadmap, but generates a small new roadmap on the fly for each planning query. It is thus more suitable for environment in which unexpected changes may occur frequently, as we will see in Section 7.4.

The rest of the paper is organized as follows. We start with a brief review of related work (Section 2). Next we formulate our problem in the state$\times$time space and present an efficient algorithm based on random sampling (Section 3). The performance of the planner has been evaluated through both analysis and experiments. We have proven that the probability that our algorithm fails to find a trajectory when one exists converges to 0 at a rate exponential in the number of sampled milestones (Section 4). We have also tested the algorithm in both simulated and real environments. In simulation, we verified that it can solve difficult problems with different kinematic, dynamic constraints and many moving obstacles (Sections 5 and 6). In the hardware robot testbed (Figure 1), we checked that the planner behaves properly despite various uncertainties and delays inherent in an integrated system (Section 7). In this testbed, the planner operates in a real-time

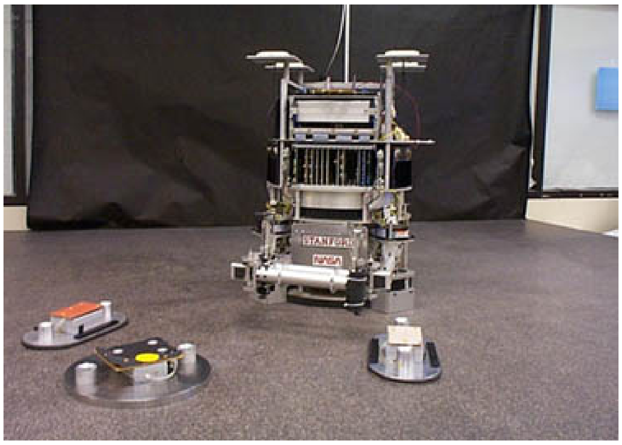


Figure 1: Robot testbed consisting of an air-cushioned robot among moving obstacles.

mode: a vision module estimates obstacle motions, and the planner must compute a trajectory in a short, predefined amount of time. To cope with the uncertainty in obstacle motion, the vision module monitors the obstacles continuously while the robot executes the planned trajectory. If an obstacle deviates from its predicted trajectory, the planner recomputes the robot's trajectory on the fly.

2 Previous work

Motion planning by random sampling This approach was originally developed to solve geometric path planning problems for robots with many degrees of freedom (dofs) [ABD$^+$98, BL91, HLS99, KŠLO96, NSL99, BOvdS99]. Random sampling uses collision checking to replace the prohibitive computation of an explicit representation of the free space. Proposed techniques differ mainly in their sampling strategies. An important distinction exists between *multi-query* planners that precompute a roadmap (*e.g.*, [KŠLO96]) and *single-query* planners that don't (*e.g.*, [HLS99]). Single-query planners build a new roadmap for each query by constructing trees of sampled milestones from the initial and goal configurations [HLS99, LK99]. Our planner falls in this second category.

It has been shown in [HLS99, KKL98, KLMR98, vO98] that, under reasonable assumptions, a multi-query PRM path planner that samples milestones uniformly from the configuration space is probabilistically complete and converges quickly. More precisely, the probability that it fails to find a path when one exists converges to 0 exponentially with the number of milestones. In [HLS99], this result is established under the assumption that the free space satisfies a property of the configuration space, called *expansiveness*. In this paper, we generalize this property to state$\times$time space and prove that our new randomized planner for kinodynamic planning is also probabilistically complete with a convergence rate exponential in the number of sampled milestones. No formal guarantee of performance had previously been established for single-query planners.

Kinematic and dynamic constraints Kinodynamic planning refers to problems in which the robot's motion must satisfy nonholonomic and dynamic constraints. With few exceptions (*e.g.*, [Fra99]), previous work has considered these two types of constraints separately.

Planning for nonholonomic robots has attracted considerable interest [BL93, Lau86, LJTM94, LCH89, LM96, SŠLO97, ŠO94]. One approach [Lau86, LJTM94] is to first generate a collision-free path, ignoring the nonholonomic constraints, and then to break this path into small pieces and replace them by admissible canonical paths (*e.g.*, Reeds and Shepp curves [RS90]). The two-stage algorithm can be extended in various ways, which are all based on the idea of successive path transformations, but differ in what transformations to use and how to perform the them [SŠLO97, SL98, Fer98]. A related approach [SŠLO97, ŠO94] uses a multi-query PRM algorithm to connects milestones directly by canonical path segments such as the Reeds and Shepp curves. All these methods require the robots to be locally controllable [BL93, LM96]. An alternative approach, presented in [BL93], generates a tree of sampled configurations rooted at the initial configuration. At each iteration, it expands a chosen node in the tree into a few new nodes by integrating the robot's equation of motion over a short duration of time with controls picked deterministically. A space partitioning scheme regulates the density of samples in the configuration space. This approach has been shown to be also applicable to robots that are not locally controllable, but is restricted to simple robots with two to four dofs.

Algorithms for dealing with dynamic constraints are comparable to those for nonholonomic constraints. In [BDG85, SD91], a collision-free path is first computed, ignoring the dynamic constraints; a variational technique then deforms this path into a trajectory that both conforms the dynamic constraints and optimizes a criterion such as minimal execution time. No formal guarantee of performance has been established for these planners. The algorithm in [DXCR93] places a regular grid over the robot's state space and searches the grid for a trajectory using dynamic programming. It is related to the work by Barraquand and Latombe [BL93] for nonholonomic motion planning. This algorithm offers provable performance guarantees, but is only applicable to robots with few dofs (typically, 2 or 3), as the size of the grid grows exponentially with the number of dofs. Our planner is related to this approach, but discretizes the state$\times$time space randomly, instead of placing a regular grid over it. It is probabilistically complete whether the robot is locally controllable or not. The planner in [LK99] resembles ours, but no guarantee of performance has been established for it.

Although most of the previous work considers nonholonomic and dynamic constraints separately, we will see in the next section that the mathematical nature of these two types of constraints is the same, and they can be treated in a unified fashion.

Moving obstacles When obstacles are moving, the planner must compute a trajectory parametrized by time. This problem has been proven to be computationally hard, even for robots with few dofs [RS85]. Heuristic algorithms [FZ96, Fuj95, KZ86] have been developed, but they usually do not consider constraints on the robot's motion other than an upper bound on its velocity. The technique proposed in [KZ86] first ignores the moving obstacles and computes a collision-free path of the robot among the static obstacles; it then tunes the robot's velocity along this path to avoid colliding with moving obstacles. The resulting planner is clearly incomplete. The planner in [Fuj95] tries to reduce the incompleteness by generating a network of paths instead of a single one in the first stage. The planner in [Fra99] is a rare example of planner dealing with both kinodynamic constraints and moving obstacles. It extends the approach of [DXCR93] to state$\times$time space and thus is also limited to robots with few dofs.

3 Planning framework

Our algorithm builds a probabilistic roadmap in the collision-free subset $\mathcal{F}$ of the state$\times$time space of the robot. The roadmap is computed in the connected component of $\mathcal{F}$ that contains the robot's initial state$\times$time point.

3.1 State-space formulation

Motion constraints We consider a robot whose motion is governed by an equation of the form

$$\dot{s} = f(s, u), \tag{1}$$

where $s \in \mathcal{S}$ is the robot's state, $\dot{s}$ is its derivative relative to time, and $u \in \Omega$ is the control input. The sets $\mathcal{S}$ and Ω are the robot's *state space* and *control space*, respectively. We assume that $\mathcal{S}$ and Ω are bounded manifolds of dimensions n and m with $m \leq n$. By defining appropriate charts, we can treat $\mathcal{S}$ and Ω as subsets of R^n and R^m. Given a state at time t and a control function $u\colon [t, t'] \rightarrow \Omega$, where $[t, t']$ is a time interval, the solution of Eq. (1) is a function $s\colon [t, t'] \rightarrow \mathcal{S}$ describing the robot's state over $[t, t']$.

Eq. (1) can represent both nonholonomic and dynamic constraints. The motion of a nonholonomic robot is constrained by k independent, non-integrable scalar equations of the form $F_i(q, \dot{q}) = 0$, $i = 1, 2, \ldots, k$, where q and $\dot{q}$ denote the robot's configuration and velocity, respectively. Define the robot's state to be $s = q$. It is shown in [BL93] that, under appropriate conditions, the constraints $F_i(s, \dot{s}) = 0$ are equivalent to Eq. (1) in which u is a vector in $\mathrm{R}^m = \mathrm{R}^{n-k}$. In particular, Eq. (1) can be rewritten as $k = n - m$ independent equations of the form $F_i(s, \dot{s}) = 0$. Dynamic constraints are closely related to nonholonomic constraints. In Lagrangian mechanics,

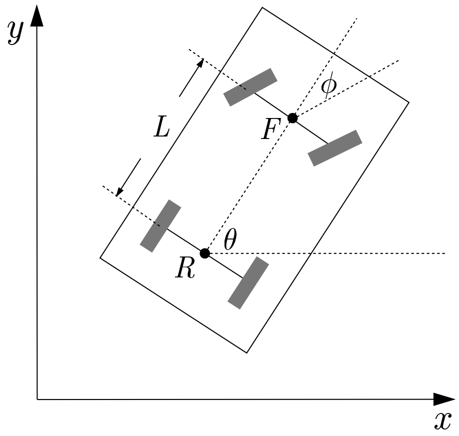


Figure 2: A simple model of a car-like robot.

dynamics equations are of the form $G_i(q, \dot{q}, \ddot{q}) = 0$, where q, $\dot{q}$, and $\ddot{q}$ are the robot's configuration, velocity, and acceleration, respectively. Defining the robot's state as $s = (q, \dot{q})$, we can rewrite the dynamics equations in the form $F_i(s, \dot{s}) = 0$, which, as in the nonholonomic case, is equivalent to Eq. (1).

Robot motions may also be constrained by inequalities of the forms $F_i(q, \dot{q}) \leq 0$ and $G_i(q, \dot{q}, \ddot{q}) \leq 0$. These-constraints restrict the sets of admissible states and controls to subsets of R^n and R^m.

Examples These notions are illustrated below with two examples that will also be useful later in the paper:

Nonholonomic car navigation. Consider the car example in Figure 2. Let (x, y) be the position of the midpoint R between the rear wheels of the robot and θ be the orientation of the rear wheels with respect to the x-axis. Define the car's state to be $(x, y, \theta) \in \mathrm{R}^3$. The non-holonomic constraint $\tan\theta = \dot{y}/\dot{x}$ is equivalent to the system

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= (v/L)\tan\phi. \end{aligned}$$

This reformulation corresponds to defining the car's state to be its configuration (x, y, θ) and choosing the control input to be the vector (v, ϕ), where v and ϕ are the car's speed and steering angle. Bounds on (x, y, θ) and (v, ϕ) can be used to restrict $\mathcal{S}$ and Ω to subsets of R^3 and R^2, respectively. For instance, if the maximum speed of the car is 1, then we have $|v| \leq 1$.

Point-mass robot with dynamics. For a point-mass robot P moving on a horizontal plane, we typically want to control the forces applied to P. This leads us to define the state of P as $s =$

(x, y, v_x, v_y), where (x, y) and (v_x, v_y) are the position and the velocity of P. The control inputs are chosen to be the forces applied to P in the x- and y-direction. Hence the equations of motion are

$$\begin{array}{rclcrcl} \dot{x} &=& v_x & \qquad & \dot{v_x} &=& u_x/m \\ \dot{y} &=& v_y & \qquad & \dot{v_y} &=& u_y/m, \end{array} \tag{2}$$

where m is the mass of P and (u_x, u_y) is the applied force. The velocity (v_x, v_y) and force (u_x, u_y) are restricted to subsets of R^2 due to limits on the maximum velocity and force.

Planning query Let $\mathcal{ST}$ denote the state×time space $\mathcal{S} \times [0, +\infty)$. Obstacles in the robot's workspace are mapped into this space as forbidden regions. The *free space* $\mathcal{F} \subset \mathcal{ST}$ is the set of all collision-free points (s, t). A trajectory $s\colon [a, b] \to \mathcal{S}$ is *admissible* if, for all $t \in [a, b]$, $s(t)$ is an admissible state and $(s(t), t)$ is collision-free.

A planning query is specified by an initial state×time point (s_b, t_b) and a goal state×time point (s_g, t_g). A solution to this query is a control function $u\colon [t_b, t_g] \to \Omega$ that produces an admissible trajectory from state s_b at time t_b to state s_g at time t_g. In the following, we will consider piecewise-constant functions $u(t)$ only.

3.2 The planning algorithm

Like the planner in [HLS99], our algorithm iteratively builds a tree-shaped roadmap T rooted at $m_b = (s_b, t_b)$. At each iteration, it first picks at random a milestone (s, t) from T, a time t' with $t' \leq t_g$, and a control function $u\colon [t, t'] \to \Omega$. It then computes the trajectory induced by u by integrating Eq. (1) from (s, t). If this trajectory is admissible, its endpoint (s', t') is added to T as a new milestone; a directed edge is created from (s, t) to (s', t'), and u is stored with this edge. The kinodynamic constraints are thus naturally enforced in all trajectories represented in T. The planner exits with success when the newly generated milestone falls in an "endgame" region that contains (s_g, t_g).

Milestone selection The planner assigns a weight $w(m)$ to each milestone m in T. The weight of m is the number of other milestones lying in the neighborhood of m. So $w(m)$ indicates how densely the neighborhood of m has already been sampled. At each iteration, the planner picks an existing milestone m in T at random with probability $\pi_T(m)$ inversely proportional to $w(m)$. Hence, a milestone lying in a sparsely-sampled region has a greater chance of being selected than a milestone lying in an already densely-sampled region. This technique avoids oversampling any particular region of $\mathcal{F}$.

Control selection Let $\mathcal{U}_\ell$ be the set of all piecewise-constant control functions with at most ℓ constant pieces. So every $u \in \mathcal{U}_\ell$ admits a finite partition $t_0 < t_1 < \ldots < t_\ell$ such that $u(t)$ is a constant $c_i \in \Omega$ over the time interval (t_{i-1}, t_i), for $i = 1, 2, \ldots, \ell$. We also require $t_i -$

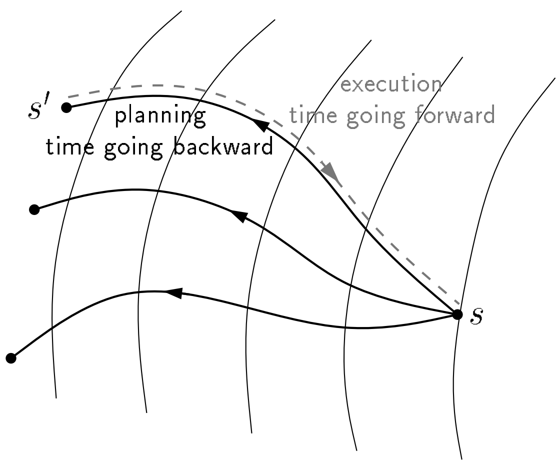


Figure 3: Building a secondary tree of milestones by integrating backwards in time.

$t_{i-1} \leq \delta_{\max}$, where $\delta_{\max}$ is a constant. Our algorithm picks a control $u \in \mathcal{U}_\ell$, for some pre-specified ℓ and $\delta_{\max}$, by sampling each constant piece of u independently. For each piece, c_i and $\delta_i = t_i - t_{i-1}$ are selected uniformly at random from Ω and $[0, \delta_{\max}]$, respectively. The specific choices of the parameters ℓ and $\delta_{\max}$ will be discussed in Section 4.5. In the actual implementation of the algorithm, however, one may chose $\ell = 1$, because any trajectory passing through several consecutive milestones in the tree T is obtained by applying a sequence of constant controls.

Endgame connection The above "control-driven" sampling technique does not allow us to reach the goal (s_g, t_g) exactly. We need to "expand" the goal into an *endgame* region that the sampling algorithm will eventually attain with high probability.

To do so, we build a secondary tree T' of milestones from the goal in the same way as that for the primary tree T, except that Eq. (1) is integrated backwards in time. Let (s', t') be a new milestone obtained by integrating backwards from an existing milestone (s, t) in T'. By construction, if the time goes forward, the control function drives the robot from state s' at time t' to state s at time t (Figure 3). Thus there is a known trajectory from every milestone in T' to the goal. The sampling process terminates with success when a milestone $m \in T$ is in the neighborhood of a milestone $m' \in T'$. In this case, the endgame region is the union of the neighborhoods of milestones in T'. To generate the final trajectory, we simply follow the appropriate edges of T and T'; however, there is a small gap between m and m'. The gap can often be dealt with in practice. For example, beyond m, one can use a PD controller to track the trajectory extracted from T'. Constructing endgame regions by backward integration is a very general technique and can be applied to any system described by (1).

Alternatively, for some systems, it is possible to compute analytically one or several canonical control functions that connect two given states exactly while obeying the kinodynamic constraints.

An example is the Reeds and Shepp curves established for nonholonomic car-like robots [RS90]. If such control functions are available and can be computed efficiently, we can test whether a milestone m belongs to the endgame region by checking that a canonical control function induces an admissible trajectory from m to (s_g, t_g).

Algorithm in pseudo-code The planning algorithm is summarized in the following pseudo-code. We will refer to it as CRE, for Control-driven Randomized Expansion.

Algorithm 1 Control-driven randomized expansion.

1. Insert m_b into T; $i \leftarrow 1$.
2. **repeat**
3.   Pick a milestone m from T with probability $\pi_T(m)$.
4.   Pick a control function u from $\mathcal{U}_\ell$ uniformly at random.
5. $m' \leftarrow$ PROPAGATE(m, u) .
6.   **if** $m' \neq nil$ **then**
7.   Add m' to T; $i \leftarrow i + 1$.
8.   Create an edge e from m to m'; store u with e.
9.   **if** $m' \in$ ENDGAME **then** exit with SUCCESS.
10.   **if** $i = N$ **then** exit with FAILURE.

In line 5, PROPAGATE(m, u) integrates the equations of motion from m with control u. It returns a new milestone m' if the computed trajectory is admissible; otherwise it returns *nil*. If there exists no admissible trajectory from $m_b = (s_b, t_b)$ to (s_g, t_g), the algorithm cannot detect it. Therefore, in line 10, we bound the maximum number of milestones to be sampled by a constant N.

4 Analysis of the planner

The experiments to be described in Sections 5–7 demonstrate that CRE provides an efficient solution for difficult kinodynamic motion planning problems. Nevertheless some important questions cannot be answered by experiments alone. What is the probability γ that the planner fails to find a trajectory when one exists? Does γ converge to 0 as the number of milestones increases? If so, how fast? In this section, we generalize the notion of expansiveness, originally proposed in [HLS99] for (geometric) path planning. We show that in an expansive space, the failure probability γ decreases exponentially with the number of sampled milestones. Hence, with high probability, a relatively small number of milestones are sufficient to capture the connectivity of the free space and answer the query correctly.

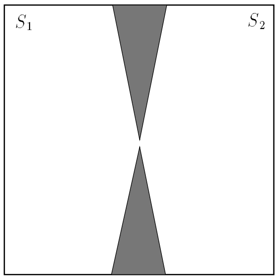


Figure 4: A free space with a narrow passage.

4.1 Expansive state×time spaces

Expansiveness tries to characterize how difficult it is to capture the connectivity of the free space by random sampling. To be concrete, consider the simple example shown in Figure 4. Assume that there are no kinodynamic constraints and a point robot can move freely in the space shown. Let us say that two points in the free space $\mathcal{F}$ *see* each other, or are mutually *visible*, if the straight line segment between them lies entirely in $\mathcal{F}$. The free space $\mathcal{F}$ in Figure 4 consists of two subsets S_1 and S_2 connected by a narrow passage. Few points in S_1 see a large fraction of S_2.

Recall that a classic PRM planner samples $\mathcal{F}$ uniformly at random and tries to connect pairs of milestones that see each other. Let the *lookout* of S_1 be the subset of all points in S_1 that sees a large fraction of S_2. If the lookout of S_1 were large, it would be easy for the planner to sample a milestone in S_1 and another in S_2 that see each other. However, in our example, S_1 has a small lookout due to the narrow passage between S_1 and S_2: few points in S_1 see a large fraction of S_2. Thus it is difficult for the planner to generate a connection between S_1 and S_2. This example suggests that we can characterize the complexity of a free space for random sampling by the size of lookout sets. In [HLS99], a free space $\mathcal{F}$ is said to be expansive if every subset $S \subset \mathcal{F}$ has a large lookout. It has been shown that in an expansive space, a classic PRM planner with uniform random sampling converges at an exponential rate as the number of sampled milestones increases.

When kinodynamic constraints are present, the basic issues remain the same, but the notion of visibility (connecting milestones with straight-line paths) is inadequate. CRE generates a different kind of roadmaps, in which trajectories between milestones may be neither straight, nor reversible. This leads us to generalize the notion of visibility to that of *reachability*.

Given two points (s,t) and (s',t') in $\mathcal{F} \subset \mathcal{ST}$, (s',t') is *reachable* from (s,t) if there exists a control function $u\colon [t,t'] \to \Omega$ that induces an admissible trajectory from (s,t) to (s',t'). If (s',t') remains reachable from (s,t) by using $u \in \mathcal{U}_\ell$, a piecewise-constant control with at most ℓ constant pieces as defined in Section 3.2, then we say that (s',t') is *locally reachable*, or *ℓ-reachable*, from (s,t). Let $\mathcal{R}(p)$ and $\mathcal{R}_\ell(p)$ denote the set of points reachable and ℓ-reachable from some point p, respectively; we call them the *reachability* and the *ℓ-reachability set* of p. For any subset $S \subset \mathcal{F}$,

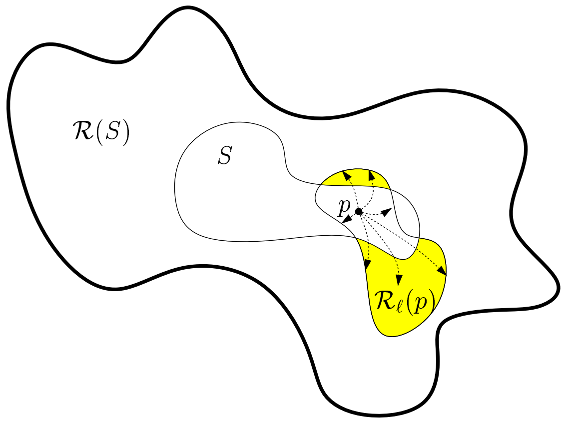


Figure 5: The lookout of a set S.

the reachability (ℓ-reachability) set of S is the union of the reachability (ℓ-reachability) sets of all the points in S:

$$\mathcal{R}(S) = \bigcup_{p \in S} \mathcal{R}(p) \quad \text{and} \quad \mathcal{R}_\ell(S) = \bigcup_{p \in S} \mathcal{R}_\ell(p).$$

We define the lookout of a set $S \subset \mathcal{F}$ as the subset of all points in S whose ℓ-reachability sets overlap significantly with the reachability set of S that is outside S (Figure 5):

Definition 1 *Let β be a constant in $(0, 1]$. The* β-lookout *of a set $S \subset \mathcal{F}$ is*

$$\beta\text{-LOOKOUT}(S) = \{p \in S \mid \mu(\mathcal{R}_\ell(p)) \setminus S) \geq \beta\, \mu(\mathcal{R}(S) \setminus S)\},$$

where $\mu(X)$ denote the volume of a set $X \subset \mathcal{F}$.

The free space $\mathcal{F}$ is expansive if for every point $p \in \mathcal{F}$, every subset $S \in \mathcal{R}(p)$ has a large lookout:

Definition 2 *Let α and β be two constants in $(0, 1]$. For any $p \in \mathcal{F}$, the set $\mathcal{R}(p)$ is (α, β)-*expansive *if for every connected subset $S \subset \mathcal{R}(p)$,*

$$\mu(\beta\text{-LOOKOUT}(S)) \geq \alpha\, \mu(S).$$

*The free space $\mathcal{F}$ is (α, β)-*expansive *if for every $p \in \mathcal{F}$, $\mathcal{R}(p)$ is (α, β)-*expansive.

To better grasp these definitions, think of p in Definition 2 as the initial milestone $m = (s_b, t_b)$ and S as the ℓ-reachability set of a set of milestones sampled by CRE. If α and β are both reasonably large, then CRE has a good chance to sample a new milestone whose ℓ-reachability set adds significantly to the size of S. In fact, we show below that with high probability, the ℓ-reachability set of the sampled milestones expands quickly to cover most of $\mathcal{R}(m_b)$; hence, if the goal (s_g, t_g) lies in $\mathcal{R}(m_b)$, then the planner will quickly find an admissible trajectory with high probability.

4.2 Ideal sampling

To simplify our presentation and focus on the most important aspects of our planner, let us assume for now that we have an ideal sampler IDEAL-SAMPLE that picks a point uniformly at random from the ℓ-reachability set of existing milestones. If it is successful, IDEAL-SAMPLE returns a new milestone p' and a trajectory from an existing milestone p to p'. With ideal sampling, the planning algorithm can be restated as follows:

Algorithm 2 Randomized expansion with IDEAL-SAMPLE.

Insert $p_0 = m_b$ into a tree T; $R_0 \leftarrow \mathcal{R}_\ell(p_0)$.
repeat
  Invoke IDEAL-SAMPLE(R_i), which samples a new milestone p' and returns a trajectory from an existing milestone p to p' if the trajectory is admissible.
  if $p' \neq$ *nil* **then**
    Insert p' into T.
    Create a directed edge e from p to p', and store the trajectory with e.
    $R_{i+1} \leftarrow R_i \cup \mathcal{R}_\ell(p')$; $i \leftarrow i + 1$.
    if $p' \in$ ENDGAME **then** exit with SUCCESS.

This algorithm is similar to Algorithm 1, except that the use of IDEAL-SAMPLE replaces lines 3–5 in Algorithm 1. We will discuss how to approximate IDEAL-SAMPLE in Section 4.4.

4.3 Bounding the number of milestones

Let $\mathcal{X} = \mathcal{R}(m_b)$ be the set of all points reachable from m_b under piecewise-constant controls. CRE determines whether the goal lies in $\mathcal{X}$ by sampling a set of milestones; it terminates as soon as a milestone falls in the endgame region. The running time of the planner is thus proportional to the number of sampled milestones. In this subsection, we give a bound on the number of milestones needed in order to guarantee a milestone in the endgame region with high probability, assuming the intersection of the endgame region and $\mathcal{X}$ is non-empty.

Let $M = (m_0, m_1, m_2, \ldots)$ be a sequence of milestones generated by Algorithm 2, and let M_i denote the first i milestones in M. A milestone m_i is called a *lookout point* if it lies in the β-lookout of $\mathcal{R}_\ell(M_{i-1})$. Lemma 1 below states that the ℓ-reachability set of M spans a large volume if it contains enough lookout points, and Lemma 2 gives an estimate of the probability of this happening. Together they imply that with high probability, the ℓ-reachability set of a relatively small number of milestones spans a large volume in $\mathcal{X}$.

The following results assume that $\mathcal{X}$ is (α, β)-expansive. For convenience, let us scale up all the volumes so that $\mu(\mathcal{X}) = 1$.

Lemma 1 *If a sequence of milestones M contains k lookout points, then $\mu(\mathcal{R}_\ell(M)) \geq 1 - e^{-\beta k}$.*

Proof. Let $(m_{i_0}, m_{i_1}, \ldots, m_{i_k})$ be the subsequence of lookout points in M. For any $i = 1, 2, \ldots$, we have

$$\mu(\mathcal{R}_\ell(M_i)) = \mu(\mathcal{R}_\ell(M_{i-1})) + \mu(\mathcal{R}_\ell(m_i) \setminus \mathcal{R}_\ell(M_{i-1})). \tag{3}$$

Thus $\mu(\mathcal{R}_\ell(M_i)) \geq \mu(\mathcal{R}_\ell(M_j))$ for all $i \geq j$. In particular,

$$\mu(\mathcal{R}_\ell(M)) \geq \mu(\mathcal{R}_\ell(M_{i_k})). \tag{4}$$

Using (3) with $i = i_k$ in combination with the fact that m_{i_k} is a lookout point, we get

$$\mu(\mathcal{R}_\ell(M_{i_k})) \geq \mu(\mathcal{R}_\ell(M_{i_k-1})) + \beta\, \mu(\mathcal{X} \setminus \mathcal{R}_\ell(M_{i_k-1})).$$

Let $v_i = \mu(\mathcal{R}_\ell(M_i))$. Since $\mu(\mathcal{X} \setminus \mathcal{R}_\ell(M_{i_k-1})) = \mu(\mathcal{X}) - \mu(\mathcal{R}_\ell(M_{i_k-1})) = 1 - v_{i_k-1}$, we have $v_{i_k} \geq v_{i_k-1} + \beta\,(1 - v_{i_k-1})$, which can be rewritten as

$$v_{i_k} \geq v_{i_{k-1}} + \beta\,(1 - v_{i_{k-1}}) + (1-\beta)(v_{i_k-1} - v_{i_{k-1}}). \tag{5}$$

Note $i_k - 1 \geq i_{k-1}$ and thus $v_{i_k-1} - v_{i_{k-1}} \geq 0$. It follows from (5) that

$$v_{i_k} \geq v_{i_{k-1}} + \beta\,(1 - v_{i_{k-1}}).$$

Setting $w_k = v_{i_k}$ leads to the recurrence $w_k \geq w_{k-1} + \beta\,(1 - w_{k-1})$, with the solution

$$w_k \geq (1-\beta)^k w_0 + \beta \sum_{j=0}^{k-1} (1-\beta)^j = 1 - (1-\beta)^k (1 - w_0).$$

As $w_0 \geq 0$ and $1 - \beta \leq e^{-\beta}$, we get $w_k \geq 1 - e^{-\beta k}$. Combined with (4), it yields

$$\mu(\mathcal{R}_\ell(M)) \geq 1 - e^{-\beta k}.$$

□

Lemma 2 *A sequence of r milestones contains k lookout points with probability at least $1 - ke^{-\alpha r/k}$.*

Proof. Let M be the sequence of r milestones, and L be the event that M contains k lookout points. We divide M into k subsequences of r/k consecutive milestones each. Denote by L_i the event that the *i*th subsequence contains at least one lookout point. Since the probability of M having k lookout points is greater than the probability of every subsequence having at least one lookout point, we have

$$\Pr(L) \geq \Pr(L_1 \cap L_2 \ldots \cap L_k),$$

which implies

$$\Pr(\overline{L}) \;\leq\; \Pr(\overline{L}_1 \cup \overline{L}_2 \ldots \cup \overline{L}_k) \;\leq\; \sum_{i=0}^{k} \Pr(\overline{L}_i).$$

Since each milestone picked by IDEAL-SAMPLE has probability α of being a lookout point, the probability $\Pr(\overline{L}_i)$ of having no lookout point in the ith subsequence is at most $(1-\alpha)^{r/k}$. Hence

$$\Pr(L) = 1 - \Pr(\overline{L}) \geq 1 - k(1-\alpha)^{r/k}.$$

Note that $(1-\alpha)^{r/k} \leq e^{-\alpha r/k}$. So we have $\Pr(L) \geq 1 - ke^{-\alpha r/k}$.

□

The main result, stated in the theorem below, establishes a bound on the number of milestones needed in order to guarantee a milestone in the endgame region with high probability.

Theorem 1 *Let $g > 0$ be the volume of the endgame region E in $\mathcal{X}$ and γ be a constant in $(0,1]$. A sequence M of r milestones contains a milestone in E with probability at least $1-\gamma$, if $r \geq (k/\alpha)\ln(2k/\gamma) + (2/g)\ln(2/\gamma)$, where $k = (1/\beta)\ln(2/g)$.*

Proof. Let us divide $M = (m_0, m_1, m_2, \ldots, m_r)$ into two subsequences M' and M'' such that M' contains the first r' milestones and M'' contains the next $r'' = r - r'$ milestones.

By Lemma 2, M' contains k lookout points with probability at least $1 - k(1-\alpha)^{r'/k}$. If there are k lookout points in M', then by Lemma 1, $\mathcal{R}_\ell(M')$ has volume at least $1 - g/2$, provided that

$$k \geq 1/\beta \ln(2/g).$$

As a result, $\mathcal{R}_\ell(M')$ has a non-empty intersection I with the endgame region of volume at least $g/2$, and so does $\mathcal{R}_\ell(M_i)$, for $i \geq r'$.

The procedure IDEAL-SAMPLE picks a milestone uniformly at random from the ℓ-reachability set of the existing milestones, and therefore every milestone m_i in M'' falls in I with probability $(g/2)/\mu(\mathcal{R}_\ell(M_{i-1}))$. Since $\mu(\mathcal{R}_\ell(M_{i-1})) \leq 1$ for all i, and the milestones are sampled independently, M'' contains a milestone in I with probability at least $1 - (1-g/2)^{r''} \geq 1 - e^{-r''g/2}$.

If M fails to contain a milestone in the endgame region E, then either the ℓ-reachability set of M' does not have a large enough intersection with E (event A), or no milestone of M'' lands in the intersection (event B). From the preceding discussion, We know that $\Pr(A) \leq \gamma/2$ if $r' \geq (k/\alpha)\ln(2k/\gamma)$ and $\Pr(B) \leq \gamma/2$ if $r'' \geq (2/g)\ln(2/\gamma)$. Choosing $r \geq (k/\alpha)\ln(2k/\gamma) + (2/g)\ln(2/\gamma)$ guarantees that $\Pr(A \cup B) \leq \Pr(A) + \Pr(B) \leq \gamma$. Substituting $k = (1/\beta)\ln(2/g)$ into the inequality bounding r, we get the final result

$$r \geq \frac{\ln(2/g)}{\alpha\beta} \ln \frac{2\ln(2/g)}{\beta\gamma} + \frac{2}{g} \ln \frac{2}{\gamma}.$$

□

If the planner returns FAILURE, either the query admits no solution, *i.e.*, $(s_g, t_g) \notin \mathcal{X}$, or the algorithm has failed to find one. The latter event, which corresponds to returning an incorrect answer to the query, has probability less than γ. Since the bound in Theorem 1 contains only logarithmic terms of γ, the probability of an incorrect answer converges to 0 exponentially in the number of milestones.

The bound given by Theorem 1 also depends on the expansiveness parameters α, β and the volume g of the endgame region. The greater α, β, and g, the smaller the bound. In practice, it is often possible to establish a lower bound for g. However, α and β are difficult to estimate, except for every simple cases. This prevents us from determining the parameter N, the maximal number of milestones needed for CRE *a priori*. Nevertheless these results are important. First, they tell us that the failure probability of our planner decreases exponentially with the number of milestones sampled. Second, the number of milestones needed increases only moderately when ϵ, α, and β decrease, *i.e.*, when the space becomes less expansive.

4.4 Approximating IDEAL-SAMPLE

The above analysis assumes the use of IDEAL-SAMPLE, which picks a new milestone uniformly at random from the the ℓ-reachability set of the existing milestones. One way to implement IDEAL-SAMPLE would be rejection sampling [KW86], which throws away a fraction of samples in regions that are more densely sampled than others. However, rejection sampling is not efficient: many potential candidates are thrown away in order to achieve the uniform distribution.

So instead, our planners try to approximate the ideal sampler. The approximation is much faster to compute, but generates a slightly less uniform distribution. Recall that to sample a new milestone p', we first choose a milestone p from the existing milestones and then sample in the neighborhood of p. Every new milestone p' thus created tends to be relatively close to p. If we selected uniformly among the existing milestones, the resulting distribution would be very uneven; with high probability, we would pick a milestone in an already densely-sampled region and obtain a new milestone in that same region. Therefore the distribution of milestones tends to cluster around the initial state$\times$time point. To avoid this problem, we associate with every milestone p a weight $w(p)$, which is the number of milestones in a small neighborhood of p, and pick an existing milestone to expand with probability inversely proportional to $w(p)$. So it is more likely to sample a region containing a smaller number of milestones. The distribution $\pi_T(p) \propto 1/w(p)$ contributes to the diffusion of milestones over the free space and avoids oversampling any particular region. In general, maintaining the weights $w(m)$ as the roadmap is being built incurs a much smaller computational cost than performing rejection sampling.

There is also a slightly greater chance of generating a new milestone in an area where the

ℓ-reachability sets of several existing milestones overlap. However, milestones with overlapping ℓ-reachability sets are more likely to be close to one another than milestones with no such overlapping. Thus it is reasonable to expect that using $\pi_T \propto 1/w(t)$ keeps the problem from worsening as the number of milestones grows.

If there are no kinodynamic constraints on the robot's motion, then other than the two issues mentioned above, Theorem 1 gives an asymptotic bound that quite closely characterizes the amount of work that the planner must do in order to guarantee finding a trajectory with high probability whenever one exists. In particular, the result applies to (geometric) path planning problems.

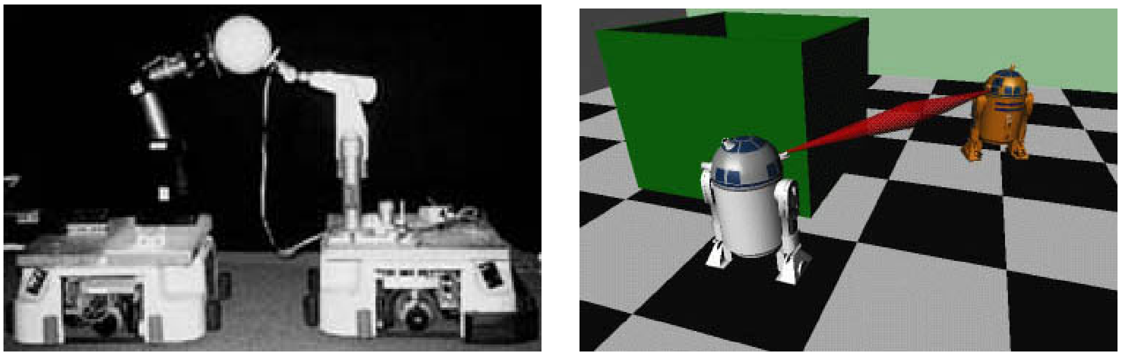
There is, however, one more issue to consider when kinodynamic constraints are present. Though line 4 of Algorithm 1 selects u uniformly at random from $\mathcal{U}_\ell$, the distribution of m' in $\mathcal{R}_\ell(m)$ is not uniform in general, because the mapping from $\mathcal{U}_\ell$ to $\mathcal{R}_\ell(m)$ may not be linear. In some cases, one may precompute a distribution π_U such that picking u from $\mathcal{U}_\ell$ with probability $\pi_U(u)$ yields a uniform distribution of m' in $\mathcal{R}_\ell(m)$. In other cases, rejection sampling can be used locally. First pick several control functions $u_i, i = 1, 2, \ldots$ and compute the corresponding m'_i, the endpoint of the trajectory induced by u_i. Then throw away some of them to achieve a uniform distribution among the remaining m'_i's, and pick a remaining m'_i at random.

4.5 Choosing suitable control functions

To sample new milestones, CRE picks at random a piecewise-constant control function u from $\mathcal{U}_\ell$. Every $u \in \mathcal{U}_\ell$ has at most ℓ constant pieces, each of which lasts for a time duration less than $\delta_{\max}$. The parameters ℓ and $\delta_{\max}$ are chosen according to the properties of each specific robot.

In theory, ℓ must be large enough so that for any $p \in \mathcal{R}(m_b)$, $\mathcal{R}_\ell(p)$ has the same dimension as $\mathcal{R}(m_b)$. Otherwise, $\mathcal{R}_\ell(p)$ has zero volume relative to $\mathcal{R}(m_b)$, and $\mathcal{R}(m_b)$ cannot be expansive even for arbitrarily small values of α and β. This can only happen when some dimensions of $\mathcal{R}(m_b)$ are not spanned directly by basis vectors in the control space Ω, but these dimensions can then be generated by combining several controls in Ω using Lie-brackets [BL93]. The mathematical definition of a Lie bracket can be interpreted as an infinitesimal "maneuver" involving two controls. Spanning all the dimensions of $\mathcal{R}(m_b)$ may require combining more than two controls of $\mathcal{U}$ by imbricating multiple Lie brackets. At most $n-2$ Lie brackets are needed, where n is the dimension of the state space. Hence it is sufficient to choose $\ell = n - 2$.

In general, the larger ℓ is, the greater α and β tend to be. So according to our analysis, fewer milestones are needed; on the other hand, the cost of integration and collision checking during the generation of a new milestone becomes more expensive. The choice of $\delta_{\max}$ is somewhat related. A larger δ_{max} may result in greater α and β, but also lead the planner to integrate longer trajectories that are more likely to be inadmissible. Experiments show that ℓ and $\delta_{\max}$ can be selected in relatively wide intervals without significant impact on the performance of the planner. However, if



(a) (b)

Figure 6: Two-cart nonholonomic robots. (a) Cooperative mobile manipulators. (b) Two wheeled nonholonomic robots that maintain a direct line of sight and a distance range.

the values for ℓ and $\delta_{\max}$ are too large, the approximation to IDEAL-SAMPLE becomes very poor.

5 Nonholonomic robots

We implemented CRE for two different robot systems. One consists of two nonholonomic carts connected by a telescopic link and moving among static obstacles. The other is an air-cushioned robot that is actuated by air thrusters and operates among moving obstacles on a flat table. The air-cushioned robot is subject to strict dynamic constraints. In this section, we discuss the implementation of CRE for the nonholonomic carts. In the next two sections, we will do the same for the air-cushioned robot.

5.1 Robot description

Wheeled mobile robots are a classical example for nonholonomic motion planning. The robot considered here is a new variation on this theme. It consists of two independently-actuated carts moving on a flat surface (Figure 6). Each cart obeys a nonholonomic constraint and has non-zero minimum turning radius. In addition, the two carts are connected by a telescopic link whose length is lower and upper bounded. This system is inspired by two scenarios. One is the mobile manipulation project at the University of Pennsylvania's GRASP Laboratory [DK99]; the two carts are each mounted with a manipulator arm and must remain within a certain distance range so that the two arms can cooperatively manipulate an object (Figure 6a). The manipulation area between the two carts must be free of obstacles. In the other scenario, two carts patrolling an indoor environment must maintain a direct line of sight and stay within a certain distance range, in order to allow visual contact or simple directional wireless communication (Figure 6b).

We project the geometry of the carts and the obstacles onto the horizontal plane. For $i = 1, 2$, let R_i be the midpoint between the rear wheels of the ith cart, F_i be the midpoint between the front wheels, and L_i be the distance between R_i and F_i. We define the state of the system by $s = (x_1, y_1, \theta_1, x_2, y_2, \theta_2)$, where (x_i, y_i) are the coordinates of R_i, and θ_i is the orientation of the rear wheels of ith cart relative to the x-axis (Figure 2). To maintain a distance range between the two cart, we require $d_{\min} \leq \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \leq d_{\max}$ for some constants $d_{\min}$ and $d_{\max}$.

Each cart has two scalar controls, u_i and ϕ_i, where u_i is speed of R_i, and ϕ_i is the steering angle. The equations of motion for the system are

$$\begin{array}{rclcrcl} \dot{x_1} & = & u_1 \cos\theta_1 & \qquad & \dot{x_2} & = & u_2 \cos\theta_2 \\ \dot{y_1} & = & u_1 \sin\theta_1 & & \dot{y_2} & = & u_2 \sin\theta_2 \\ \dot{\theta_1} & = & (u_1/L_1)\tan\phi_1 & & \dot{\theta_2} & = & (u_2/L_2)\tan\phi_2. \end{array} \tag{6}$$

The control space is restricted by $|u_i| \leq u_{\max}$ and $|\phi| \leq \phi_{\max}$, which bound the carts' velocities and steering angles.

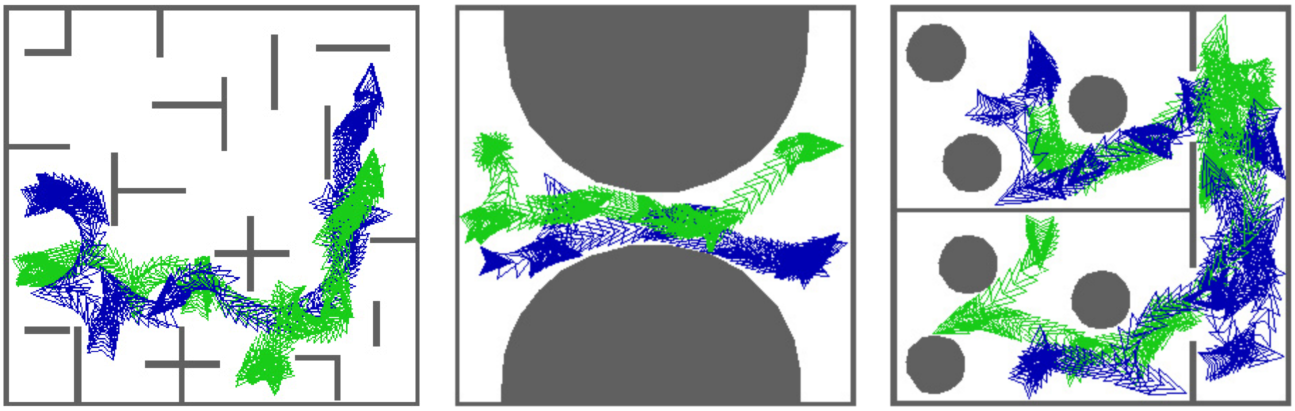
We assume that all obstacles are stationary. So the planner builds a roadmap T in the robot's 6-D state space.

Computing the weights To compute the weight $w(m)$ of a milestone m, we define the neighborhood of m to be a small ball of fixed radius centered at m. The current implementation uses a naive method that checks every new milestone m' against all the milestones currently in T. Thus, for every new milestone, updating w takes linear time in the number of milestones in T. More efficient range search techniques [Aga97] would certainly improve the planner's running time for problems requiring very large roadmaps.

Implementing PROPAGATE Given a milestone m and a control function u, PROPAGATE uses the Euler method with a fixed step size to integrate (6) from m and computes a trajectory σ of the system under the control u. More sophisticated integration methods, *e.g.*, fourth-order Runge-Kutta or extrapolation method [PTVP92], can improve the accuracy of integration, but at a higher computational cost.

We then discretize σ into a sequence of states and returns *nil* if any of these states is in collision. For each cart, we precompute a 3-D bitmap that represents the collision-free configurations of the cart prior to planning. It then takes constant time to check whether a given configuration is in collision. A well-known disadvantage of this method is that if the resolution of the bitmap is not fine enough, we may get wrong answers. In the experiments reported below, we used an $128 \times 128 \times 64$ bitmap, which was adequate for our test cases.

Endgame region We obtain the endgame region by generating a secondary tree T' of milestones from the goal s_g.



(a) (b) (c)

Figure 7: Computed examples for nonholonomic carl-like robots.

5.2 Experimental results

We experimented with the planner in many workspaces. Each one is a 10 m $\times$ 10 m square region with static obstacles. The two carts are identical, each represented by a polygon contained in a circle of radius 0.4 m, and $L_1 = L_2 = 0.5$ m. The speed of the carts ranges from -3 m/s to 3 m/s, and its steering angle ϕ varies between $-30°$ and $30°$. The allowable distance between R_1 and R_2 ranges from 1.4 m to 3.3 m.

5.3 Implementation details

Figure 7 shows three computed examples. Environment (a) is a maze; the robot carts must navigate from one side of it to the other. Environment (b) contains two large obstacles separated by a narrow passage. The two carts, which are initially parallel to one another, change formation and proceed in a single file through the passage, before becoming parallel again. Environment (c) consists of two rooms cluttered with obstacles and connected by a hallway. The carts need to move from the room at the bottom to the one at the top. The maximum steering angles and the size of the circular obstacles conspire to increase the number of required maneuvers.

We ran the planner for several different queries in each workspace shown in Figure 7. For every query, we ran the planner 30 times independently with different random seeds. The results are shown in Table 1. Our planner was written in C++, and the running times reported were obtained on an SGI Indigo2 workstation with a 195 Mhz R10000 processor.

Every row of the table corresponds to a particular query. Columns 2–5 list the average running time, the average number of collision checks, and their standard deviations. Columns 6–7 give the total number of milestones sampled and the number of calls to PROPAGATE. The running times range from less than a second to a few seconds. The first query in environment (c) takes longer

Scene	Time (sec.)		$N_{\rm clear}$		$N_{\rm mil}$	$N_{\rm pro}$
	mean	std	mean	std		
(a)	1.39	0.91	62402	27001	2473	21316
	0.74	0.65	43564	23640	1630	15315
	0.54	0.41	35960	18410	1318	12815
	0.55	0.44	38384	20772	1310	14066
(b)	4.45	3.92	126126	61836	4473	45690
(c)	14.09	7.42	287828	86987	9123	107393
	0.92	0.51	56367	20825	1894	20250

Table 1: Performance statistics of the planner on the nonholonomic robot.

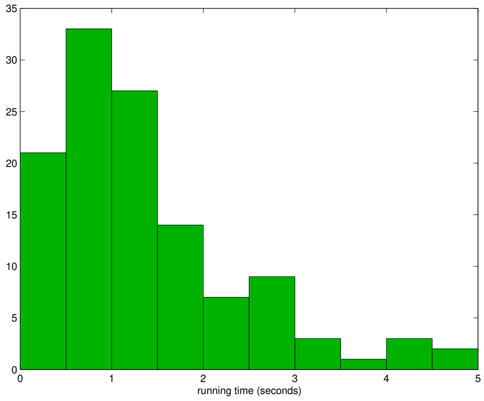


Figure 8: Histogram of planning times for more than 100 runs on a particular query. The average time is 1.4 sec, and the four quartiles are 0.6, 1.1, 1.9, and 4.9 seconds.

because the carts must perform several maneuvers in the hallway before reaching the goal (see the example in Figure 7c).

The standard deviations in Table 1 are larger than what we would like. In Figure 8, we show a histogram of more than 100 independent runs for a particular query. In most runs, the running time is well under the mean or slightly above. This indicates that our planner performs well most of the time. The large deviation is caused by a few runs that take as long as four times the mean. The long and thin tail of the distribution is typical of the tests that we have performed.

6 Air-cushioned robots

6.1 Robot description

Our algorithm has also been implemented and evaluated on a second system, which was developed at the Stanford Aerospace Robotics Laboratory for testing space robotics technology. This air-cushioned robot (Figure 1) moves frictionlessly on a flat granite table among moving obstacles.

Eight air thrusters provides omni-directional motion capability, but the thrust available is small compared to the robot's mass, resulting in tight acceleration constraints.

We define the state of the robot to be $(x, y, \dot{x}, \dot{y})$, where (x, y) are the coordinates of the robot's center, and $(\dot{x}, \dot{y})$ is the velocity. The equations of motion are

$$\ddot{x} = \frac{1}{m} u \cos\theta \quad \text{and} \quad \ddot{y} = \frac{1}{m} u \sin\theta,$$

where m is the robot's mass, and u and θ are the magnitude and direction of the force generated by the thrusters. We have $0 \leq u/m \leq 0.025$ m/s^2 and $0° \leq \theta \leq 360°$. The maximum speed of the robot is 0.18 m/s.

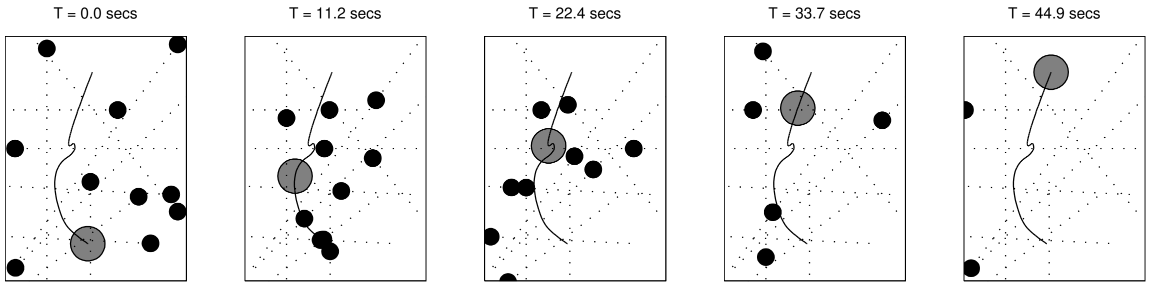
For planning purposes, the workspace is represented by a 3 m $\times$ 4 m rectangle, the robot by a disc of radius 0.25 m, and the obstacles by discs of radii between 0.1 and 0.15 m. The planner assumes that the obstacle moves along a straight-line path at constant speed between 0 and 0.2 m/s (more complex trajectories will be considered in Section 7.4). When an obstacle reaches the workspace's boundary, it leaves the workspace and is no longer considered a threat to the robot.

6.2 Implementation details

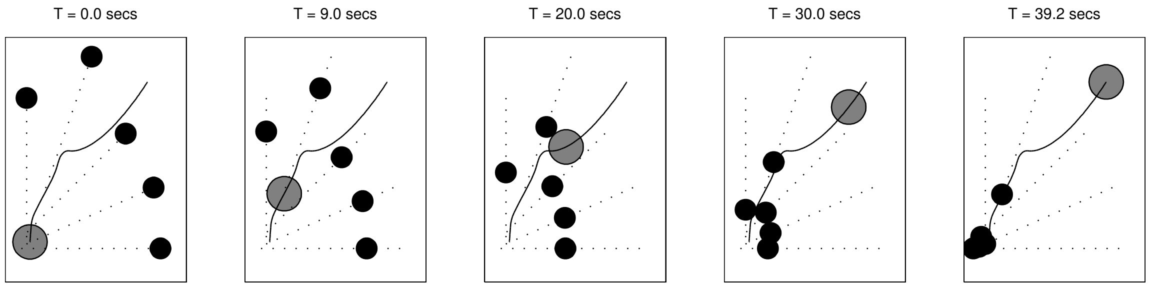
The planner builds a roadmap T in the robot's 5-D state$\times$time space, It is given an initial state$\times$time $(s_b, 0)$ and a goal state$\times$time (s_g, t_g), where t_g is any time less than a given $t_{\max}$. In addition, the planner is given the obstacle trajectories. Unlike the experiments with the real robot in the next section, planning time is not limited here. This is equivalent to assuming that the world is frozen until the planner returns a trajectory.

Computing the weights The 3-D configuration$\times$time space of the robot is partitioned into an 8$\times$11$\times$10 array of identically sized rectangular boxes called bins. When a milestone is inserted in T, the planner adds it to the list of milestones associated with the bin in which it falls. To implement line 3 of Algorithm 1, the planner first picks at random a bin containing at least one milestone and then a milestone from within this bin. Both choices are made uniformly at random. This corresponds to picking a milestone with probability approximately proportional to the inverse of the density of samples in the robot's configuration$\times$time space (rather than its 5-D state$\times$time space). We did some experiments with bins in state$\times$time space, but the results did not differ significantly.

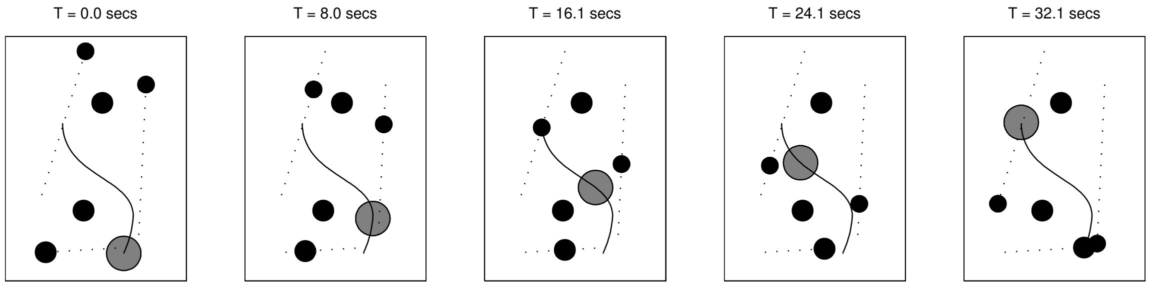
Implementing PROPAGATE The simplicity of the equations of motion makes is possible to compute trajectories analytically. The trajectories are then discretized, and at each discretized state$\times$time point, the robot is checked for collision against every obstacle. This naive technique works reasonably well when the number of obstacles is small, but can be easily improved to handle a large number of obstacles.



(a)



(b)



(c)

Figure 9: Computed examples for the air-cushioned robot.

Endgame region The endgame region is generated with specialized curves, specifically, third-order splines. Whenever a new milestone m is added to T, it is checked for connection with k goal points (s_g, t_g), for some pre-defined constant k. Each of the k values of t_g is chosen uniformly at random from the interval $[t_{\min}, t_{\max}]$, where $t_{\min}$ is an estimate of the earliest time when the robot may reach s_g, given its maximum velocity. For each value of t_g, the planner computes the third-order spline between m and (s_g, t_g). It then verifies that the spline is collision free and satisfies the velocity and acceleration bounds. If all the tests succeed, then m lies in the endgame region. In all the experiments reported below, k is set to 10.

Scene	Time (sec)		N_{mil}	
	mean	std	mean	std
(a)	0.249	0.264	2008	2229
(b)	0.270	0.285	1946	2134
(c)	0.002	0.005	22	25

Table 2: Performance statistics of the planner on the air-cushioned robot.

6.3 Experimental results

We performed experiments in more than one hundred simulated environments. To simplify the simulation, collisions among obstacles are ignored. So two obstacles may overlap temporarily without changing courses. In a small number of queries, the planner failed to return a trajectory, but in none of these cases were we able to determine whether an admissible trajectory actually existed. On the other hand, the planner successfully solved several queries for which we initially thought there were no solution.

Three examples computed by the planner are shown in Figure 9. For each example, we display five snapshots labeled by time. The large gray disc indicates the robot; the smaller black discs indicate the obstacles. The solid and dotted lines mark the trajectories of the robot and the obstacles, respectively. For each of the three queries, we ran the planner 100 times independently with different random seeds. The planner successfully returned a trajectory in all runs. Table 2 lists the means and standard deviations of the planning times and the number of sampled milestones for each query. The reported times were obtained from a planner written in C and running on a Pentium-III PC with a 550 Mhz processor and 128 MB of memory.

In the first two examples, the moving obstacles create narrow passages through which the robot must pass in order to reach the goal. Yet planning time remains much under one second. The fact that the planner never failed in 100 runs testifies to its reliability. To point out the difficulty of these queries, we show in Figure 10 the configuration$\times$time space for example (b). In the configuration$\times$time space, the robot maps to a point (x, y, t). Since the obstacles are assumed to move with constant linear velocity, they map into cylinders. The velocity and acceleration constraints force any solution trajectory to pass through a small gap between the cylinders.

Example (c) is much simpler. There are two stationary obstacles obstructing the middle of the workspace and three moving obstacles. Planning time is well below 0.01 second, with an average of 0.002 second. The number of milestones is also small, confirming the result of Theorem 1 that in the absence of narrow passages, CRE is very efficient. As in the experiments on nonholonomic robot carts, the running time distribution of the planner tends to have a long and thin tail due to long execution time in a small number of runs, but overall the planner is very fast.

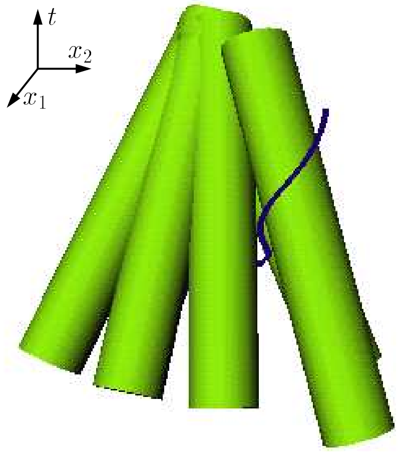


Figure 10: Configuration×space for the example in Figure 9b.

7 Experiments with the real robot

To further test the performance of the planner, we connected the planner described in the previous section to the air-cushioned robot in Figure 1. In these tests, we examined the behavior of CRE running in real-time mode on a system integrating control and sensing modules over a distributed architecture and operating in a physical environment with uncertainties and time delays.

7.1 Testbed description

The robot shown in Figure 1 is untethered and moves frictionlessly on an air bearing on a 3 m×4 m table. Gas tanks provide compressed air for both the air-bearing and thrusters. An onboard Motorola ppc2604 computer performs motion control at 60 Hz. Obstacles are also on air-bearings, but have no thrusters. They are initially propelled by hand from various locations and then move frictionlessly on the table at roughly constant speed until they reach the boundary of the table, where they stop due to the lack of air bearing.

An overhead vision system estimates the positions of the robot and the obstacles at 60 Hz by detecting LEDs placed on the moving objects. The measurement is accurate to 5 mm. Velocity estimates are derived from position data.

Our planner runs offboard on a 333 Mhz Sun Sparc 10. The planner, the robot, and the vision module communicate over the radio Ethernet.

7.2 System integration

Implementing the planner on the hardware testbed raises several new challenges.

Time delays Various computations and data exchanges occurring at different parts of the system lead to delays between the instant when the vision module measures the motion of the robot and the obstacles and the instant when the robot starts executing the planned trajectory. These delays, if ignored, would cause the robot to begin executing the planned trajectory behind the start time assumed by the planner. The robot may not then be able to catch up with the planned trajectory before a collision occurs. To deal with this issue, the planner computes a trajectory assuming that the robot will start executing it 0.4 second into the future. It also assumes that the obstacles move at constant velocities, as measured by the vision module, and extrapolates their positions accordingly. The 0.4 second includes all the delays in the system, in particular, the time needed for planning. This time could be further reduced by implementing the planner more carefully and running it on a machine faster than the relatively slow Sun Sparc 10 currently being used.

Sensing errors Although the planner assumes that the obstacles move along straight lines at constant velocities measured by the vision module, the actual trajectories are slightly different due to asymmetry in air-bearings and inaccuracy in the measurements. The planner deals with these errors by growing the obstacles. As time elapses, the radius of each moving obstacle is increased by $\xi V t$, where ξ is a fixed constant, V is the measured velocity of the obstacle, and t is the time. So the planner can avoid erroneously asserting that a state$\times$time point is collision-free when it is actually not.

Trajectory tracking The robot receives from the planner a trajectory that specifies the position, velocity, and acceleration of the robot at all times. A PD-controller with feedforward is used to track this trajectory. The maximum tracking errors for the position and velocity are 0.05 m and 0.02 m/s, respectively. As a result, we increase the size of the disc modeling the robot by 0.05 m during the planning to guarantee that the computed trajectory is collision-free.

Trajectory optimization Since the planner is very efficient in general, the 0.4 second allocated is often more than what is needed for finding a first solution. So the planner exploits the extra time to generate additional milestones and keeps track of the best trajectory found so far. The cost function for comparing trajectories is $\sum_{i=1}^{k}(u_i + b)\delta_i$, where k is the number of segments in the trajectory, u_i is the magnitude of the force exerted by the thrusters along the ith segment, b is a fixed constant, and δ_i is the duration of the ith segment. The cost function takes into account both fuel consumption and execution time. A larger b yields faster motion, while a smaller b yields less fuel consumption. In our experiments, the cost of trajectories was reduced, on the average, by 14% with this simple improvement.

Safe-mode planning If the planner fails to find a trajectory to the goal within the allocated time, we found it useful to compute an *escape* trajectory. The endgame region E_{esc} for the escape trajectory consists of all the reachable, collision-free states (s_e, t_e) with $t_e \geq T_{\text{esc}}$ for some time T_{esc}. An escape trajectory corresponds to any acceleration-bounded, collision-free motion in the workspace

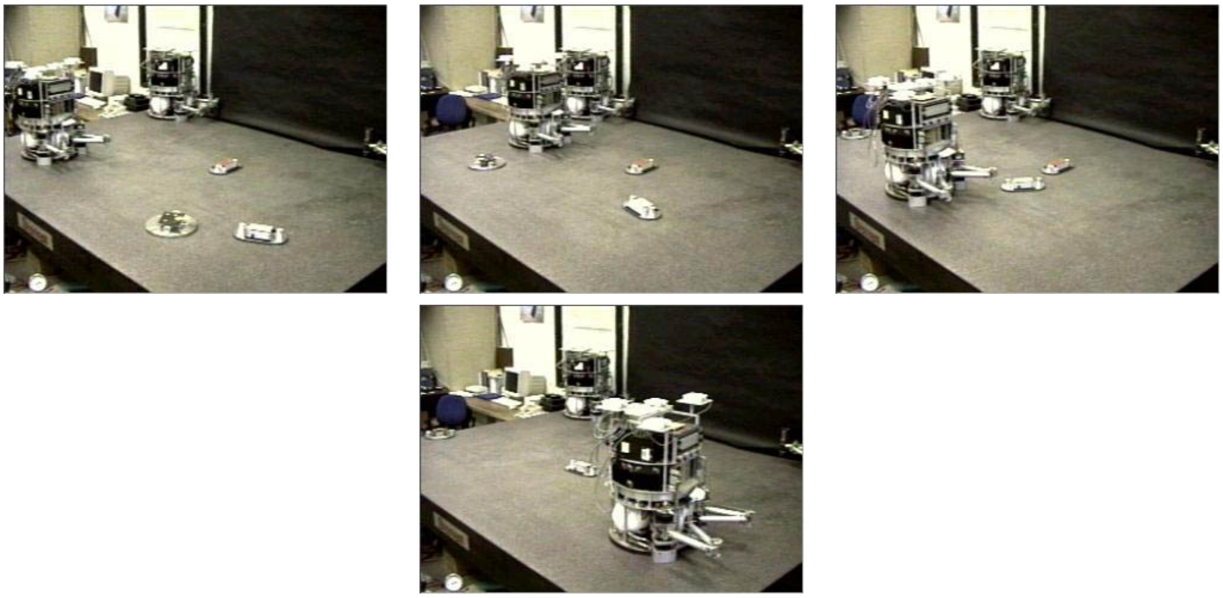


Figure 11: Snapshots of the robot executing a trajectory.

for a small duration of time. In general, E_{esc} is very large, and so generating an escape trajectory often takes little time. To ensure collision-free motion beyond T_{esc}, a new escape trajectory must be computed long before the end of the current escape trajectory so that the robot can escape collision despite the acceleration constraints. We modified the planner to compute simultaneously a normal and an escape trajectory. The modification increased the running time of the planner by about 2% in our experiments, but it leads to a system that is much more useful practically.

7.3 Experimental results

The planner successfully produced complex maneuvers of the robot among static and moving obstacles in various situations, including obstacles moving directly toward the robot or perpendicular to the line connecting its initial and goal positions. The tests also demonstrated the ability of the system to wait for an opening to occur when confronted with moving obstacles in the robot's desired direction of movement and to pass through openings that are less than 10 cm larger than the robot. In almost all the trials, a trajectory was computed within the allocated time. Figure 11 shows snapshots of the robot during one of the trials, in which the robot maneuvers among three incoming obstacles to reach the goal at the front corner of the table.

Several problems limited the complexity of the planning problems which we could try in this testbed. Two are related to the testbed itself. First the accelerations provided by the robot's air thrusters are quite limited. Second the size of the table is small relative that of the robot and the obstacles, which limits the available space for the robot to maneuver. The third problem results

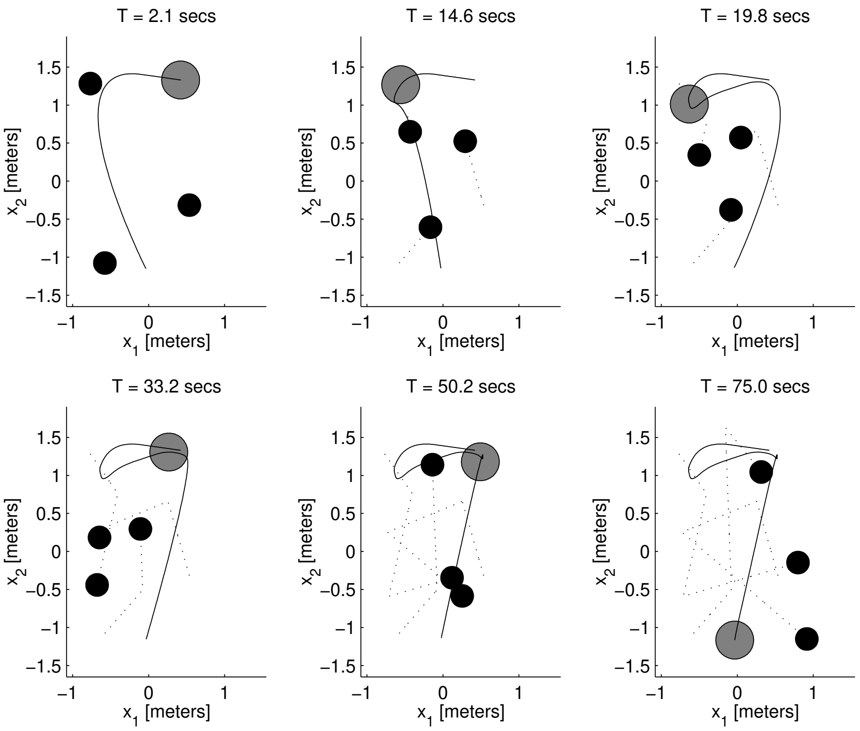


Figure 12: A computed example with replanning in a simpulated environment.

from the design of our system. The planner assumes that obstacles move at constant linear velocities and do not collide with one other, an assumption which is likely to fail in practice. To address this last and important issue, we introduce on-the-fly replanning.

7.4 On-the-fly replanning

An obstacle may deviate from its predicted trajectory, because either the error in the measurements is larger than expected, or the obstacle's direction of motion has suddenly changed due to a collision with other obstacles. Whenever the vision module detects this, it alerts the planner. The planner then recomputes a trajectory on the fly within the same allocated time limit, by projecting the state of the world 0.4 second into the future. On-the-fly replanning allows much more complex experiments to be performed. We show two examples below, one in simulation and one on the real robot.

In the example shows in Figure 12, eight replanning operations occurred over the entire course (75 seconds) of the experiment. Initially the robot moves to the left to reach the goal at the bottom middle (snapshot 1). Then the upper-left obstacle changes its motion and blocks the robot's way, resulting in a replan (snapshot 2). Soon after, the motion of the upper-right obstacle also changes, forcing the robot to reverse the direction and approach the goal from the other side of the workspace

Figure 13: An example with the real robot using on-the-fly replanning.

(snapshot 3). In the remaining time, new changes in obstacle motion cause the robot to pause (see the sharp turn in snapshot 5) until a direct approach to the goal is possible (snapshot 6).

The efficacy of the replanning procedure on the real robot is demonstrated by the example in Figure 13. The robot's goal is to move from the back left of the table to the front middle. Initially the obstacle in the middle is stationary, and the other two obstacles are moving towards the robot (snapshot 1). The robot dodges the faster-moving obstacle from the left and proceeds toward the goal (snapshot 2). The obstacle is then redirected twice (in snapshots 3 and 5) to block the trajectory of the robot, causing it to slow down and stay behind the obstacle to avoid collision (snapshots 3–6). Right before snapshot 7, the rightmost obstacle is directed back towards the robot. The robot waits for the obstacle to pass (snapshot 8) and finally attains the goal (snapshot 9). The entire motion lasts about 40 seconds. Throughout this experiment, other replanning operations (not shown) occurred as a result of errors in the measurement of the obstacle motions. However, none resulted in a major redirection of the robot.

8 Conclusion

We have presented a simple, efficient randomized planner for kinodynamic motion planning in the presence of moving obstacles. Our algorithm represents the motion constraints by an equation of the form $\dot{s} = f(s, u)$ and constructs a roadmap of sampled milestones in the state space of a robot. It samples new milestones by first picking at random a point in the space of admissible control functions and then mapping the point into the state space by integrating the equations of motion. Thus the motion constraints are enforced naturally during the construction of roadmaps. The algorithm is general and can be applied to a wide class of systems, including ones that are not locally controllable.

The performance of the algorithm has been evaluated through both analysis and experiments. We have proven that in an expansive space, the failure probability of the planner converge to 0 at an exponential rate, when a solution exists. This result also holds for robots that are not locally controllable. Experimentally the planner has demonstrated its effectiveness both in simulation and on a real robot. The experiments on the real robot indicates that the planner remains effective despite many adversarial conditions, including (i) severe dynamic constraints on the motion of the robot, (ii) moving obstacles, and (iii) various time delays and uncertainties inherent to an integrated system operating in a physical environment. In particular, they show that the planner can be used in real time when obstacles trajectories are not known in advance.

In the future, we plan to apply the planner to environments with more complex geometry and robots with higher dofs. Geometrical complexity increases the cost of collision-checking, but hierarchical techniques deal with this issue well. In [HLS99], a similar, but simpler planner was successfully applied to compute geometric disassembly paths with CAD models having up to 200,000 triangles.

We are also interested in reducing the standard deviation of the running times of our randomized planner. Quite possibly, the long thin tail of the running time distribution shown in Figure 8 is typical of all PRM planners developed so far. However, it seems more important to reduce it for single-query planners, since they are intended to be used interactively or in real time. Large standard deviations in these settings are highly undesirable.

Acknowledgments: This work was supported by ARO MURI grant DAAH04-96-1-007, NASA TRIWG Coop-Agreement NCC2-333, Real-Time Innovations, and the NIST ATP program. David Hsu has also been the recipient of a Microsoft Graduate Fellowship, and Robert Kindel, the recipient of an NSF Graduate Fellowship.

References

[ABD⁺98] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal et al., editors, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 155–168, Wellesley, MA, 1998. A. K. Peters.

[Aga97] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, pages 575–598. CRC Press, 1997.

[BDG85] J. E. Bobrow, S. Dubowsky, and J.S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(3):3–17, 1985.

[BL91] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, 1991.

[BL93] J. Barraquand and J. C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4):121–155, 1993.

[BOvdS99] V. Boor, M. H. Overmars, and F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1018–1023, 1999.

[DK99] J. P. Desai and V. Kumar. Motion planning for cooperating mobile manipulators. *Journal of Robotic Systems*, 16(10):557–579, 1999.

[DXCR93] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Jounal of the ACM*, 40(5):1048–1066, 1993.

[Fer98] P. Ferbach. A method of progressive constraints for nonholonomic motion planning. *IEEE Transactions on Robotics and Automation*, 14(1):172–179, 1998.

[Fra99] T. Fraichard. Trajectory planning in a dynamic workspace: A 'state-time space' approach. *Advanced Robotics*, 13(1):75–94, 1999.

[Fuj95] K. Fujimura. Time-minimum routes in time-dependent networks. *IEEE Transactions on Robotics and Automation*, 11(3):343–351, 1995.

[FZ96] P. Fiorini and Z.Shiller. Time optimal trajectory planning in dynamic environments. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1553–1558, 1996.

[HLS99] D. Hsu, J. C. Latombe, and S. Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *Proc. IEEE International Symposium on Assembly and Task Planning*, pages 280–285, 1999.

[KKL98] L. E. Kavraki, M. Kolountzakis, and J. C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, 1998.

[KLMR98] L. E. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot motion planning. *Jounal of Computer and System Sciences*, 57(1):50–60, 1998.

[KŠLO96] L. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[KW86] M. H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*, volume 1. John Wiley & Son, New York, 1986.

[KZ86] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *International Journal of Robotics Research*, 5(3):72–89, 1986.

[Lau86] J.-P. Laumond. Feasible trajectories for mobile robots with kinematic and environmental constraints. In *Proc. Int. Conf. on Intelligent Autonomous Systems*, pages 346–354, 1986.

[LCH89] Z. Li, J. F. Canny, and G. Heinzinger. Robot motion planning with nonholonomic constraints. In H. Miura et al., editors, *Robotics Research: The Fifth International Symposium*, pages 309–316. MIT Press, Cambridge, MA, 1989.

[LJTM94] J-P. Laumond, P. E. Jacobs., M. Taïx., and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, 1994.

[LK99] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 473–479, 1999.

[LM96] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research*, 15(6):533–556, 1996.

[NSL99] C. Nissoux, T. Siméon, and J.-P. Laumond. Visibility based probabilistic roadmaps. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1316–1321, 1999.

[PTVP92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Plannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[RS85] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 144–154, 1985.

[RS90] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[SD91] Z. Shiller and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, 7(6):785–797, 1991.

[SL98] S. Sekhavat and J.-P. Laumond. Topological property for collision-free nonholonomic motion planning: The case of sinusoidal inputs for chained form systems. *IEEE Transactions on Robotics and Automation*, 14(5):671–680, 1998.

[ŠO94] P. Švestka and M. H. Overmars. Motion planning for car-like robots using a probabilistic learning approach. Technical Report RUU-CS-94-33, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands, 1994.

[SŠLO97] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystms. In J.-P. Laumond et al., editors, *Algorithms for Robotic Motion and Manipulation: 1996 Workshop on the Algorithmic Foundations of Robotics*, pages 79–96. A. K. Peters, Wellesley, MA, 1997.

[vO98] P. Švestka and M. Overmars. Probabilistic path planning: robot motion planning and control. In *Lecture Notes in Control and Information Sciences*, volume 229, pages 255–304. Springer, 1998.