APPEARED IN
The Fourth International Workshop on Algorithmic Foundations of Robotics, 2000

Randomized Kinodynamic Motion Planning with Moving Obstacles

David Hsu, *Stanford University, Stanford, CA, USA*
Robert Kindel, *Stanford University, Stanford, CA, USA*
Jean-Claude Latombe, *Stanford University, Stanford, CA, USA*
Stephen Rock, *Stanford University, Stanford, CA, USA*

A randomized motion planner is presented for robots that must avoid collision with moving obstacles under kinematic and dynamic constraints. This planner samples the robot's state×time space by picking control inputs at random and integrating the equations of motion. The result is a probabilistic roadmap, i.e., a collection of sampled state×time points, called milestones, connected by short admissible trajectories. The planner does not precompute the roadmap; instead, for each planning query, it generates a new roadmap to connect the input initial and goal state×time points. This paper shows that the probability that the planner fails to find a trajectory when one exists quickly goes to 0 as the number of milestones grows. The planner has been tested successfully in both simulated and real environments. In the latter case, a vision module estimates the obstacle motions just before planning, and the planner is then allocated a small amount of time to compute a trajectory. If a change in obstacle motion is detected while the robot executes the planned trajectory, the planner re-computes a trajectory on the fly.

1 Introduction

In its most basic 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 configurations. This formulation ignores several key aspects of the real world. Robot motions are subject to kinematic and dynamic constraints 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 algorithm for this class of problems.

Our work extends the probabilistic roadmap (PRM) framework originally developed for planning collision-free geometric paths [18]. 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 configuration space. The result is a undirected graph called a probabilistic roadmap. Multi-query PRM planners precompute the roadmap [18], while single-query planners compute a new roadmap for each query [11]. 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 [11, 17]. However, with nonholonomic and/or dynamic constraints, straight local paths are not feasible. Moreover, allowing obstacles to move requires indexing milestones by the times when they are attained.

For each planning query, our algorithm builds a new roadmap in the collision-free subset of the robot's state×time space, where a state typically encodes both the configuration and velocity of the robot (Section 3). Each milestone is obtained by selecting a control input at random in the set of admissible controls and integrating 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, and 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 rooted at the initial state×time and oriented along the time axis. It ends when a milestone falls in an "endgame" region from which it is known how to reach the goal.

In Section 4, we show that the probability that our algorithm fails to find a trajectory when one exists converges toward 0 (probabilistic completeness), with a convergence rate that is exponential in the number of generated milestones. We have also tested the algorithm in both simulated and real environments. In simulation (Sections 5 and 6), we have verified that it can solve tricky problems. In the hardware robot testbed (Section 7), we have checked that the planner operates properly despite various uncertainties and delays associated with an integrated system. In this testbed, a vision module measures obstacle motions, and the planner has a short, predefined amount of time to compute a trajectory (real-time planning). The vision module monitors the obstacles while the robot executes the computed trajectory. If an obstacle deviates from its predicted trajectory, the planner re-computes the robot's trajectory on the fly.

2 Previous Work

Motion planning by random sampling This approach was originally proposed to solve geometric path-planning problems for robots with many degrees of freedom (dofs) [2, 3, 11, 16, 18]. Sampling replaces the prohibitive computation of an explicit representation of the free space by collision checking operations. Proposed techniques differ mainly in their sampling strategies. An important distinction is between *multi-query* planners that precompute a roadmap (e.g., [18]) and *single-query* planners that don't (e.g., [11]). Single-query planners build a new roadmap for each query by constructing trees of sampled milestones from the initial and goal configurations [11, 21]. Our planner falls in this second category.

It has been shown in [11, 15, 17, 29] that, under some assumptions, a multi-query PRM path planner that samples milestones uniformly from the configuration space is probabilistically complete and converges quickly. More formally, the probability that it fails to find a path when one exists converges toward 0 exponentially with the number of milestones. In [11], this result is established under the assumption that the free space verifies a geometric property 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/or dynamic constraints. With few exceptions (e.g., [9]), previous work has considered these two types of constraints separately.

Planning for nonholonomic robots has attracted considerable interest [4, 19, 20, 22, 23, 26, 28]. One approach [19, 20] 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 feasible canonical paths (e.g., Reeds and Shepp curves [24]). This approach works well for simple robots (e.g., car-like robots), but requires the robots to be locally controllable [4, 23]. A related approach [26, 28] uses a multi-query PRM algorithm that connects milestones by canonical path segments such as the Reeds and Shepp curves. Another approach, presented in [4], generates a tree of sampled configurations rooted at the initial configuration. At each iteration, a chosen sample in the tree is expanded into a few new samples, by integrating the robot's equation of motion over a short duration of time with deterministically picked controls. A space partitioning scheme limits the density of samples in any region of the configuration space. This approach has been shown to be also applicable to robots that are not locally controllable. Our new planner has many similarities with this approach, but picks controls at random. It is probabilistically complete whether the robot is locally controllable or not.

Algorithms for dealing with dynamic constraints are comparable to those developed for nonholonomic constraints. In [5, 27] 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 approach in [7] places a regular grid over the robot's state space and searches the grid for a trajectory using dynamic programming. It 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 second approach, but randomly discretizes the state$\times$time space, instead of placing a regular grid over it. The planner in [21] resembles ours, but no guarantee of performance has been established for it.

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 [25]. Heuristic algorithms [8, 10, 14] have been proposed, but they usually do not consider constraints on the robot's motion other than an upper bound on its velocity. The technique in [14] 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 [10] tries to reduce incompleteness by generating a network of paths. The planner in [9] is a rare example of planner dealing concurrently with kinodynamic constraints and moving obstacles. It extends the approach of [7] to state×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×time space of the robot [9]. The roadmap is computed in the connected component of $\mathcal{F}$ that contains the robot's initial state×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 a control input. The set $\mathcal{S}$ and Ω are the robot's *state space* and *control space*, respectively. Given a state at time t and a control function $u\colon [t, t'] \to \Omega$, where $[t, t']$ is a time interval, the solution of Eq. (1) is a function $s\colon [t, t'] \to \mathcal{S}$ describing the robot's state over $[t, t']$. We assume that $\mathcal{S}$ and Ω are bounded manifolds of dimensions n and m ($m \leq n$), respectively. By defining appropriate charts, we can treat $\mathcal{S}$ and Ω as subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$.

Eq. (1) can represent both nonholonomic and dynamic constraints. 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. Let the robot's state be $s = q$. It is shown in [4] that, under appropriate mathematical conditions, the constraints $F_i(s, \dot{s}) = 0$ are equivalent to Eq. (1) in which u is a vector in $\mathbb{R}^m = \mathbb{R}^{n-k}$. Reciprocally, Eq. (1) can be rewritten into $k = n - m$ independent equations of the form $F_i(s, \dot{s}) = 0$. Furtheremore, in Lagrangian mechanics, 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).

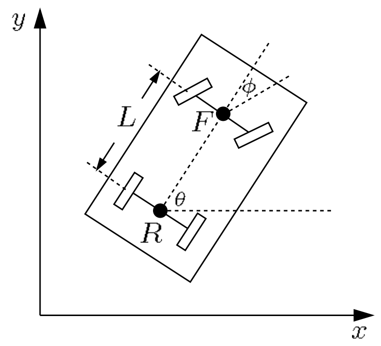


Figure 1: *Model of a car-like robot*

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

Examples We illustrate these notions with two examples that will be useful later in the paper:

Nonholonomic car-like robot. Consider a car A modeled as shown in Figure 1. Let (x, y) be the position of the midpoint R between A's rear wheels and θ be the orientation of the rear wheels with respect to the x-coordinate axis. We encode A's configuration by $(x, y, \theta) \in \mathbb{R}^3$. The nonholonomic constraint $\tan\theta = \dot{y}/\dot{x}$ is equivalent to the system:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = (v/L)\tan\phi,$$

which has the same form as Eq. (1). This reformulation corresponds to defining the state of A to be its configuration and choosing the vector (v, ϕ), where v and ϕ denote the car's linear velocity and steering angle, as the input control. Bounds on v and ϕ define Ω

as a subset of $\mathbb{R}^2$.

Point-mass robot. For a point-mass robot A moving in a plane, we typically want to control the force applied to A. So, we define A's state to be $s = (x, y, \dot{x}, \dot{y})$, where (x, y) is A's position. The equations of motion are:

$$\ddot{x} = u_x/m, \qquad \ddot{y} = u_y/m,$$

where m and (u_x, u_y) denote A's mass and the force applied to it. Bounds on the magnitudes of $(\dot{x}, \dot{y})$ and (u_x, u_y) define $\mathcal{S}$ and Ω as subsets of $\mathbb{R}^4$ and $\mathbb{R}^2$, respectively.

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* $\mathcal{F} \subseteq \mathcal{ST}$ 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 (s_b, t_b) and a goal state×time (s_g, t_g). A solution to this query is a 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 consider piecewise-constant functions $u(t)$ only.

3.2 The planning algorithm

Like the planner in [11], our algorithm iteratively builds a tree-shaped roadmap T rooted at $m_b = (s_b, t_b)$. At each iteration, it picks at random a milestone (s, t) from T, a time $t' \leq t_g$, and a control function $u\colon [t, t'] \to \Omega$. The trajectory from (s, t) induced by u is computed by integrating Eq. (1). If this trajectory is admissible, its endpoint (s', t') is added to T as a new milestone; an arc is created from (s, t) to (s', t'), and u is stored with this arc. The kinodynamic constraints are thus naturally enforced in all trajectories represented in T. The planner exits with success when the newly generated milestone lies 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 contained in a neighborhood of m. So $w(m)$ represents 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. Hence, for any $u \in \mathcal{U}_\ell$, there exist $t_0 < t_1 < \ldots < t_{\ell-1} < t_\ell$ such that $u(t)$ is a constant $c_i \in \Omega$ over the time interval (t_{i-1}, t_i), for $i = 1, \ldots, \ell$. We also require $t_i - t_{i-1} \leq \delta_{\max}$, where $\delta_{\max}$ is a constant. Our algorithm picks a control $u \in \mathcal{U}_\ell$, for some prespecified 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 choices of the parameters ℓ and $\delta_{\max}$ will be discussed in Subsection 4.4.

Endgame connection The above "control-based" 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. There are several ways of creating such an endgame region.

For some robots, it is possible to analytically compute one or several canonical control functions that exactly connect two given points while obeying the kinodynamic constraints. The Reeds and Shepp curves developed for nonholonomic car-like robots are an example of such functions [24]. If such control functions are available, one can test if a milestone m belongs to the engame region by checking that a canonical function generates an admissible trajectory from m to (s_g, t_g).

A more general technique is to build a secondary tree T' of milestones rooted at the goal, in the same way as that for the primary tree T, except that Eq. (1) is integrated backwards in time. The endgame region is then the union of small neighborhoods of the milestones in T': the planner exits with success when a milestone $m \in T$ falls in the neighborhood of a milestone $m' \in T'$. The trajectory following the appropriate arcs of T and T' contains a small gap between m and m', but this 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'.

Algorithm in pseudo-code The planning algorithm is formalized in the following pseudo-code. We will refer to it as KDP.

Algorithm 1

1. Initialize T with m_b; $i \leftarrow 1$
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 arc 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 total number of milestones by a constant N.

4 Analysis of the Planner

The experiments that will be described in Sections 5–7 demonstrate that KDP provides an effective approach for solving 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 toward 0 as the number of milestones increases? In this section we show that 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. Our analysis is based on a generalization of the notion of expansiveness proposed in [11].

4.1 Expansive state×time space

Expansiveness characterizes the difficulty of finding a path between two points in a given space by random sampling. To be concrete, let us first consider the simple example in Figure 2. The free space $\mathcal{F}$ consists of two subsets S_1 and S_2 connected by a narrow passage.

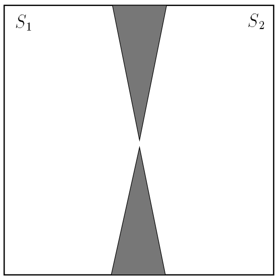


Figure 2: *A free space with a narrow passage*

Let us say that two points in $\mathcal{F}$ *see* each other (or are mutually *visible*) if the straight line segment between them lies entirely in $\mathcal{F}$ (no kinodynamic constraint is considered in this example). A classical PRM planner samples $\mathcal{F}$ uniformly at random and connects any two 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, the planner would easily pick a milestone in S_1 and another in S_2 that see each other. However, due to the narrow passage between S_1 and S_2, S_1 has a small lookout. Consequently, it is difficult for the planner to generate a connection between S_1 and S_2. In [11], $\mathcal{F}$ is said to be expansive if every subset $S \subset \mathcal{F}$ has a large lookout. It is shown that in an expansive space, the convergence rate of a classical PRM planner is exponential in the number of milestones.

KDP generates a different kind of roadmap, in which trajectories between milestones may neither be 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 said to be *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 segments, 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 p, respectively; we call them the *reachability* and the *ℓ-reachability set* of p. For any subset $S \subset \mathcal{F}$, we define:

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

We define the lookout of a subset S of $\mathcal{F}$ as the subset of all points in S whose ℓ-reachability sets overlap significantly with their reachability sets outside S:

Definition 1 Let β be a constant in $(0,1]$. The β-*lookout* of $S \subset \mathcal{F}$ 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 $\mu(Y)$ denote the volume of any subset $Y \subset \mathcal{R}(S)$ relative to $\mathcal{R}(S)$.

The free space $\mathcal{F}$ is expansive if every subset $S \subset \mathcal{F}$ has a large lookout:

Definition 2 Let α and β be two constants in $(0,1]$. For any $p \in \mathcal{F}$, $\mathcal{R}(p)$ is (α,β)-*expansive* if for every connected subset $S \subseteq \mathcal{R}(p)$, $\mu(\beta\text{-LOOKOUT}(S)) \geq \alpha\, \mu(S)$. The free space $\mathcal{F}$ is (α,β)-*expansive* if for every $p \in \mathcal{F}$, $\mathcal{R}(p)$ is (α,β)-*expansive.*

Think of p in Definition 2 as the initial milestone m_b and S as the ℓ-reachability set of a set of milestones produced by KDP after some iterations. If α and β are both reasonably large, then KDP 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 milestones sampled by KDP 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 with high probability, the planner will quickly find an admissible trajectory.

4.2 Probabilistic convergence of the planner

Let $\mathcal{X} = \mathcal{R}(m_b)$ be the reachability set of m_b. Suppose that $\mathcal{X}$ is (α,β)-expansive. We establish below an upper bound on the number of milestones needed to guarantee that a milestone lies in the endgame region E with high probability, if $E \cap \mathcal{X}$ has non-zero volume relative to $\mathcal{X}$. For convenience, we scale all volumes so that $\mu(\mathcal{X}) = 1$.

Let us assume for now that there is an ideal sampler IDEAL-SAMPLE that picks a point uniformly at random from the ℓ-reachability set $\mathcal{R}_\ell(M)$ of any set of milestones M. The procedure IDEAL-SAMPLE replaces lines 3-5 in KDP. We will discuss how to approximate IDEAL-SAMPLE in Subsection 4.3.

Let $M = (m_0, m_1, m_2, \ldots)$ be a sequence of milestones generated by KDP with IDEAL-SAMPLE ($m_0 = m_b$), and let M_i denote the first i milestones in M. The milestone m_i is called a *lookout point* if it lies in the β-lookout of $\mathcal{R}_\ell(M_{i-1})$. Lemma 3 states that the ℓ-reachability set of M spans a large volume if it contains enough lookout points, and Lemma 4 estimates the probability that this happens. Together, they imply that with high probability, the ℓ-reachability set of a relatively small number of milestones spans a large volume in $\mathcal{X}$.

Lemma 3 *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:

$$\mu(\mathcal{R}_\ell(M_i)) = \mu(\mathcal{R}_\ell(M_{i-1})) + \mu(\mathcal{R}_\ell(m_i) \setminus \mathcal{R}_\ell(M_{i-1})). \tag{2}$$

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

$$\mu(\mathcal{R}_\ell(M)) \geq \mu(\mathcal{R}_\ell(M_{i_k})). \tag{3}$$

Using (2) 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 $v_i = \mu(\mathcal{R}_\ell(M_i))$. We observe:

$$\mu(\mathcal{X} \setminus \mathcal{R}_\ell(M_{i_k-1})) = \mu(\mathcal{X}) - \mu(\mathcal{R}_\ell(M_{i_k-1})) = 1 - v_{i_k-1}.$$

Hence, $v_{i_k} \geq v_{i_k-1} + \beta\,(1 - v_{i_k-1})$, which can be rewritten as:

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

Since $i_k - 1 \geq i_{k-1}$, it follows that $v_{i_k-1} - v_{i_{k-1}} \geq 0$. Therefore, the previous inequality yields:

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

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 (3), it yields:

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

□

Lemma 4 *A sequence of r milestones contains k lookout points with probability at least* $1 - ke^{-\alpha \lfloor r/k \rfloor}$.

Proof: Let M be the sequence of r milestones and L be the event that M contains k lookout points. Assume that r is an integer multiple of k. We divide M into k subsequences of r/k consecutive milestones. Denote by L_i the event that the ith 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:

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

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:

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

If r is not an integer multiple of k, we divide M into $k-1$ subsequences of length $\lfloor r/k \rfloor$ and a kth subsequence of length $r - k\lfloor r/k \rfloor$. We get:

$$\Pr(L) \geq 1 - ke^{-\alpha \lfloor r/k \rfloor}.$$

□

We are now ready to state our main result which establishes an upper bound on the number of milestones needed to guarantee that the planner finds a trajectory with high probability, if one exists.

Theorem 5 *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: 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'' milestones with $r' + r'' = r$. By Lemma 4, 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 3, $\mathcal{R}_\ell(M')$ 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 E of volume at least $g/2$, and so does the ℓ-reachability set of every subsequence $M_i \supset M'$.

Since IDEAL-SAMPLE picks a milestone uniformly at random from the ℓ-reachability set of the existing milestones, 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 have a milestone in E, then either the ℓ-reachability set of M' does not have a large enough intersection I with E (event A), or no milestone of M'' lands in I (event B). 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:

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

□

If KDP 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 5 contains only logarithmic terms of γ, the probability of an incorrect answer converges toward 0 exponentially in the number of milestones.

The bound given by Theorem 5 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 or impossible to estimate, except for trivial cases. This prevents us from determining the parameter N (maximal number of milestones) for KDP *a priori*. This is not different from previous analyses of PRM path planners [11, 15, 17, 29].

4.3 Approximating IDEAL-SAMPLE

One way of implementing IDEAL-SAMPLE would be to use rejection sampling [13]: generate many samples and throw away a fraction of them in the more densely sampled regions. However, this would lead KDP to generate and then discard many potential milestones.

KDP seeks to approximate IDEAL-SAMPLE in a more efficient way. Note that every new milestone m' created in line 5 of Algorithm 1 tends to be relatively close to m, because long trajectories induced by controls picked at random are often in collision. Therefore, if we selected milestones uniformly in line 3, the resulting distribution would be very uneven; indeed, with high probability, at line 3, the planner would pick a milestone in an already densely sampled region, which would yield a new milestone in that same region in line 5. The distribution $\pi_T(m) \sim 1/w(m)$ used at line 3 contributes to the diffusion of milestones over $\mathcal{R}(m_b)$ and avoids oversampling. In general, maintaining the weights $w(m)$ as the roadmap is being built has a much smaller computational cost than performing rejection sampling.

There is a slightly greater chance of generating a new milestone in an area where the ℓ-reachability sets of several milestones already in T overlap. However, milestones in T with overlapping ℓ-reachability sets are more likely to be close to one another than milestones with no such overlapping. Therefore, the use of π_T at line 3 keeps the problem under control by preventing it from worsening as the number of milestones grows.

There is yet another issue to consider. Though line 4 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 many 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 as follows: in line 4, pick several control functions u_i; in line 5, compute the corresponding m'_i, throw away some of them to achieve a uniform distribution among the remaining m'_i, and pick a remaining m'_i at random.

4.4 Choice of ℓ and $\delta_{\max}$

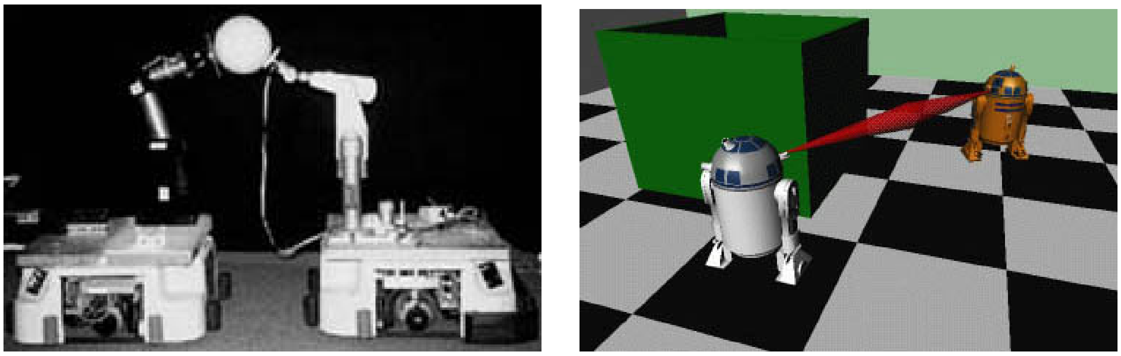
In theory, the parameter ℓ must be chosen such that for any $p \in \mathcal{R}(m_b)$, $\mathcal{R}_\ell(p)$ has the same dimension as $\mathcal{R}(m_b)$. Otherwise, $\mathcal{R}_\ell(p)$ has zero volume relative to $\mathcal{R}(m_b)$, and $\mathcal{R}(m_b)$ cannot be expansive even for arbitrarily small values of α and β. This can only happen when some dimensions of $\mathcal{R}(m_b)$ are not directly spanned by constant controls in Ω. But these dimensions can then be generated by combining several controls in Ω using Lie-brackets [4]. 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 Ω, by imbricating multiple Lie brackets. At most $n-2$ Lie brackets are needed, where n is the dimension of $\mathcal{S}$. Hence, it is sufficient in all cases to choose $\ell = n - 2$.

To simplify the implementation, however, one may choose $\ell = 1$, since a path passing through several consecutive milestones in T corresponds to applying a sequence of constant controls. In general, the larger ℓ, the greater α and β, hence the smaller the number of milestones needed according to our analysis, but also the more costly the generation of each milestone. The choice of $\delta_{\max}$ is somewhat related. A larger $\delta_{\max}$ results in greater α and β, but also leads the planner to integrate longer trajectories that are more likely to be non-admissible. Experiments show that ℓ and $\delta_{\max}$ can be selected in rather wide intervals without significant impact on the performance of the planner. However, if the values for ℓ $\delta_{\max}$ are too large, it may be difficult to approximate IDEAL-SAMPLE well.

5 Nonholonomic Robots

5.1 Robot description

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



(a) (b)

Figure 3: *Two-cart nonholonomic robots*

Wheeled vehicles 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 3). 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 has been inspired by two scenarios. One is the mobile manipulation project in the GRASP Laboratory at the University of Pennsylvania [6]; 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 3(a)). The manipulation area between the two carts must be clear of obstacles. In the other scenario, two carts patrolling an indoor environment must remain in a direct line of sight of each other (Figure 3(b)), within some distance range, in order to allow visual contact or simple directional wireless communication.

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 and F_i be the midpoint between the front wheels. Let 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 cart i relative to the x-axis (Figure 1). The distance constraint between the two carts is expressed as $d_{\min} \leq \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \leq d_{\max}$.

Each cart has two scalar controls: the magnitude u_i of the velocity of R_i and the steering angle ϕ_i (the orientation of F_i's velocity relative to the rear wheels). The equations of motion for the system are:

$$\begin{array}{rclrcl} \dot{x}_1 & = & u_1 \cos\theta_1 & \dot{x}_2 & = & u_2 \cos\theta_2 \\ \dot{y}_1 & = & u_1 \sin\theta_1 & \dot{y}_2 & = & u_2 \sin\theta_2 \\ \dot{\theta}_1 & = & (u_1/L_1)\tan\phi_1 & \dot{\theta}_2 & = & (u_2/L_2)\tan\phi_2 \end{array}$$

The control space is restricted by $|u_i| \leq U_i$ and $|\phi| \leq \Phi_i$, which bound the carts' velocities and steering angles.

5.2 Implementation details

Since all obstacles are stationary, 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 ball of radius ρ centered at m. Our 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 size of T. More efficient range search techniques [1] would improve the planner's running time for problems requiring very large roadmaps.

Implementing PROPAGATE Given a milestone m and a control function u, PROPAGATE (m, u) uses the Euler method with a fixed step size to integrate the trajectory τ of the robot. It then discretizes τ into a sequence of states and returns *nil* if any of these states is in collision. For each cart, a 3-D bitmap that represents the collision-free configurations of the cart is precomputed prior to planning. PROPAGATE then takes constant time to check whether a configuration

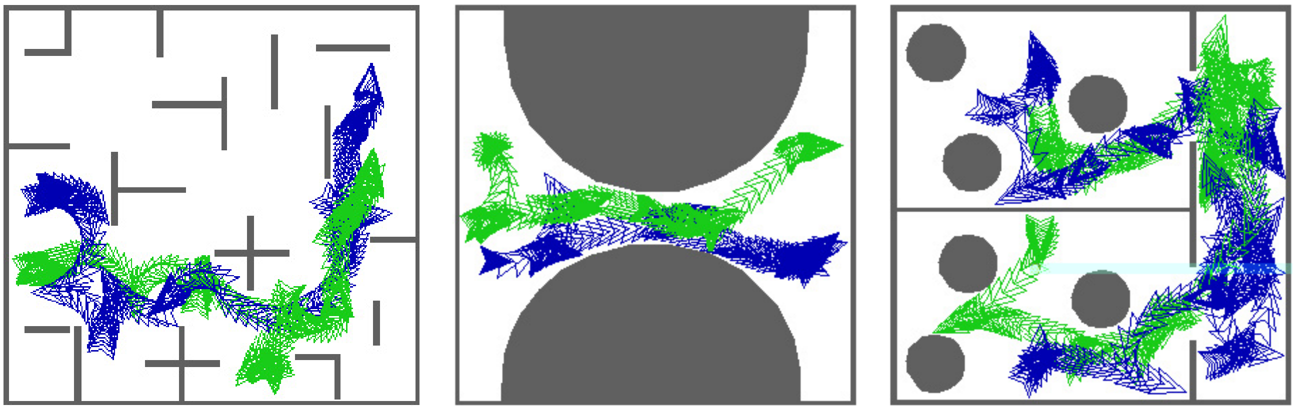


Figure 4: *Examples of nonholonomic paths computed by the planner*

is in collision, or not. In our experiments, we used a $128 \times 128 \times 64$ bitmap.

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

5.3 Experimental results

We experimented with the planner in many workspaces. Each 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. $L_1 = L_2 = 0.5$ m. Each cart's speed ranges from -3 m/s to 3 m/s, and its steering angle ϕ varies between $-30°$ and $30°$ The distance between R_1 and R_2 ranges between 1.4 m and 3.3 m.

Figure 4 shows three computed examples. Workspace (a) is a maze; the robot must navigate from one side of it to the other. Workspace (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. Workspace (c) consists of two rooms cluttered with obstacles and connected by a hallway. The carts need to move from the bottom one to the the top one. The maximum steering angles and the size of the circular obstacles conspire to increase the number of required maneuvers.

We have run the planner for several different queries in each workspace shown in Figure 4. For every query,

E	Q	time (sec)		coll.	mil.	prop.
		mean	std			
(a)	1	1.39	0.91	62,400	2473	21316
	2	0.74	0.65	43,600	1630	15315
	3	0.54	0.41	36,000	1318	12815
	4	0.55	0.44	38,400	1310	14066
(b)	1	4.45	3.92	126,100	4473	45690
(c)	1	14.09	7.42	287,800	9123	107393
	2	0.92	0.51	56,400	1894	20250

Table 1: *Planning statistics for the nonholonomic robot*

we ran the planner 30 times independently with different random seeds. The results are collected in Table 1. Every row of the table corresponds to a particular query. Columns 3-7 list the average running time, its standard deviation, the average number of collision checks, the average number of milestones generated, and the average 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 because the carts must perform several maneuvers in the hallway before reaching the goal (see Figure 4(c)). The planner was written in C++, and the running times were collected on a 195 Mhz SGI Indigo2 with an R10000 processor.

The standard deviations in Table 1 are larger than we would like. Figure 5 plots a histogram of more than 100 independent runs for a given query. In most runs, planning 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

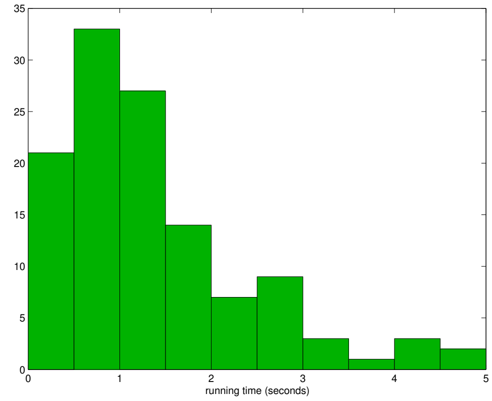


Figure 5: *Histogram of planning times for more than 100 runs on a particular query. The average time is 1.4 sec, and the four quartiles are 0.6, 1.1, 1.9, and 4.9 sec.*

thin tail of the distribution is typical of the tests that we have performed.

6 Air-Cushioned Robots

6.1 Robot description

The second robot system used to evaluate our algorithm was developed in the Stanford Aerospace Robotics Laboratory for testing space robotic technologies. This robot (Figure 6) moves frictionlessly on an air bearing on a flat granite table. Air thrusters provides omni-directional motion capability, but the thrust available is small compared to the robot's mass, resulting in tight acceleration limits. We represent the workspace 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 cm. Each obstacle moves along a straight path at constant velocity ranging between 0 and 0.2 m/s (more complex trajectories will be considered in Subsection 7.4). In the simulation environment, collisions among obstacles are ignored. So two obstacles may temporarily overlap without changing courses. When an obstacle reaches the workspace's boundary, it leaves the workspace and is no longer considered a threat to the robot.

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

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

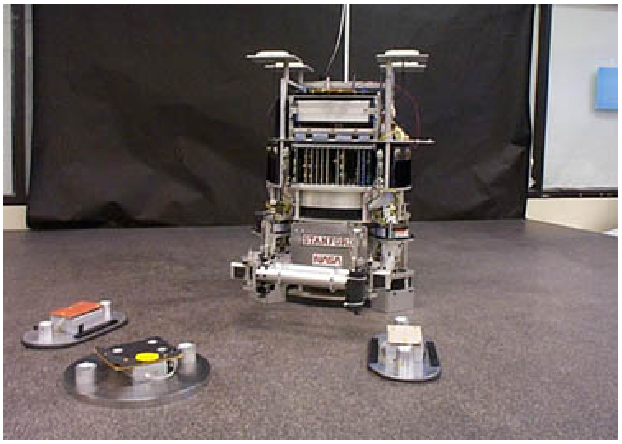


Figure 6: *The air-cushioned robot*

where m is for the robot's mass, and u and θ denote the magnitude and direction of the force generated by the thrusters. We have $u/m < 0.025\ \mathrm{m/s^2}$, and θ varies freely between $0°$ and $360°$. We also bound the robot's velocity by 0.18 m/s.

6.2 Implementation details

The planner builds a roadmap T in the robot's 5-D state$\times$time space, The initial state$\times$time is of the form $(s_b, 0)$ and the goal is of the form (s_g, t_g) where t_g can be any time less than a given $t_{\max}$. The planner is given the obstacle trajectories, and unlike in the experiments with the real robot in the next section, planning time is not limited. 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 8x11x10 array of identically sized rectangles called bins. When a milestone is inserted in T, the planner adds it to a list of milestones associated with the bin in which it falls. In line 3 of KDP, the planner 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 a probability approximately proportional to the inverse of the density of samples in the robot's configuration$\times$time space (rather than its state$\times$time space). We did some experiments with higher-dimensional bin arrays, but the results were not improved significantly.

Implementing PROPAGATE The simple equations

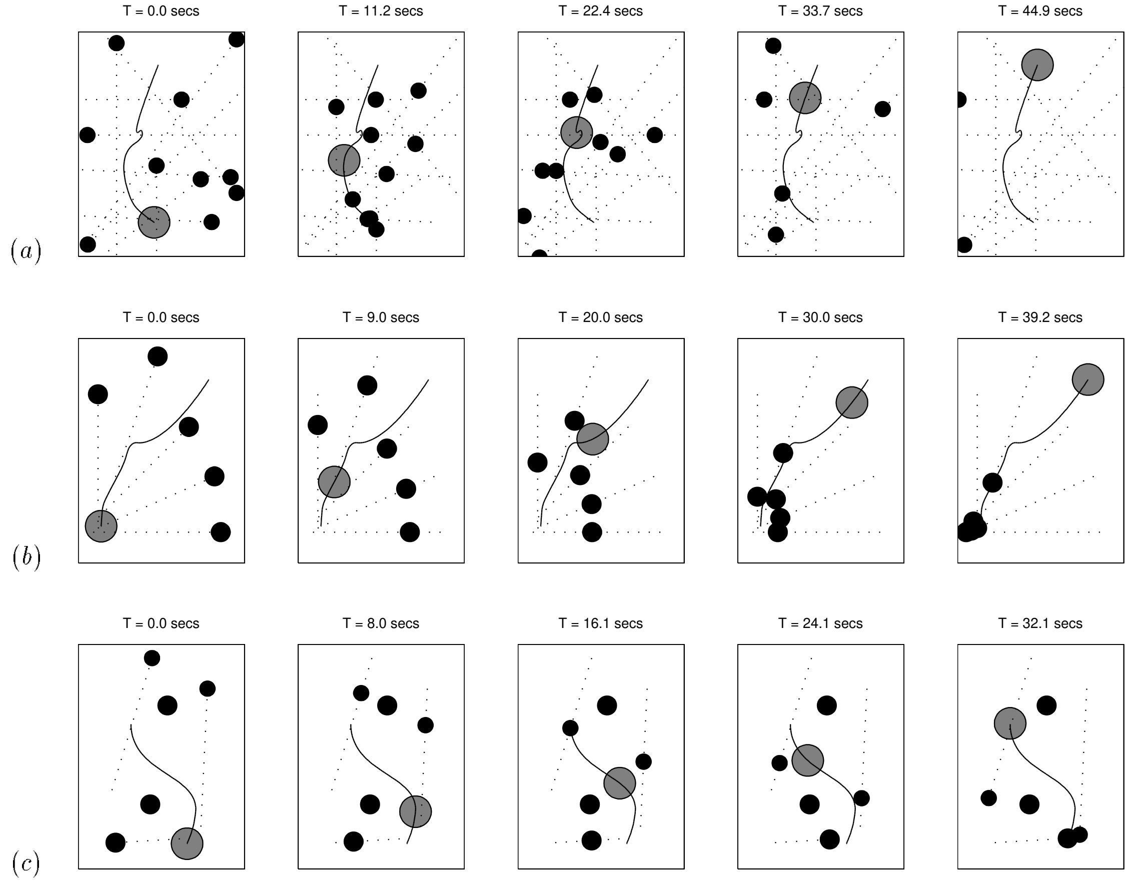


Figure 7: *Trajectories produced by the planner for the air-cushioned robot*

of motion make is possible to compute trajectories analytically. Trajectories are discretized and at each discretized state×time the robot disc is checked for collision with each obstacle disc. This naive technique works reasonably well when the number of obstacles is small. It could easily be improved.

Endgame region When a milestone m is added to T, it is checked for a connection to k goal points (s_g, t_g). Each of the k values of t_g is picked uniformly at random in 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 maximal velocity and assuming no obstacles. For each value of t_g, the planner computes the third-order spline connecting m to (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 below, k is set to 10.

6.3 Experimental results

We performed numerous experiments in more than one hundred simulated environments. In each case, planning time was limited to five minutes. For a small number of queries, the planner failed to return a trajectory, but for none of them were we able to show that an admissible trajectory exists. On the other hand, the planner successfully solved several queries for which we initially thought there were no solution.

Three trajectories computed by the planner are shown in Figure 7. For every example, we display five snapshots labeled by time. The large disc is the robot; the smaller discs are the obstacles. The solid and dotted lines mark the trajectories of the robot and the obstacles, respectively. For each query, we ran the planner 100 times independently with different random seeds. The planner successfully returned a trajectory

E	time (sec)		milestones	
	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: *Planning statistics for the air-cushioned robot*

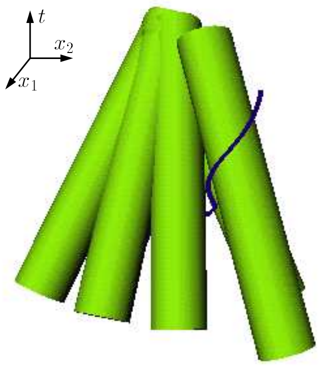


Figure 8: *Configuration×space for Example* (b)

in all runs. In Example (a), the duration of the trajectory varied from 40 to 70 sec. Table 2 lists the means and standard deviations of the planning time and number of milestones for each example. The reported times are based on a planner written in C and running on a Pentium-III PC with a 550 Mhz processor and 128M of memory.

In the first two examples, the moving obstacles create narrow passages through which the robot must pass to reach the goal. Yet planning time remains much under 1 second. The fact that the planner never failed in 100 runs indicates its reliability. The configuration×time space for Example (b) is shown in Figure 8. The robot maps to a point (x, y, t), and the obstacles to cylinders. The velocity and acceleration constraints force any solution trajectory to pass through a small gap between 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 5 that in the absence of narrow passages, KDP 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.

7 Experiments with the Real Robot

We connected the planner of the previous section to the air-cushioned robot of Figure 6 in order to verify that KDP remains useful in a system integrating control and sensing modules over a distributed architecture and operating in a physical environment with uncertainties, time delays, and real-time constraints.

7.1 Testbed description

The robot in Figure 6 moves frictionlessly on an air bearing within the limits of a 3 m×4 m table. Obstacles, also on air-bearings, translate without friction on the table. The positions of the robot and the obstacles are measured at 60 Hz by an overhead vision system thanks to LEDs placed on each moving object. The measurement is accurate to 5 mm. Velocity estimates are derived from position data.

The robot is untethered. Gas tanks provide compressed air for both the air-bearing and thrusters. An onboard Motorola ppc2604 computer performs motion control at 60 Hz. The planner runs on on a 333 Mhz Sun Sparc 10. The robot communicates with the planner and the vision module over the radio Ethernet.

The obstacles have no thrusters. They are initially propelled by hand from various locations, and then move at constant speed until they reach the boundary of the table, where they stop due to the lack of air bearing.

7.2 System integration

Implementing the planner on the hardware testbed raises a number of new challenges:

Delays Various computations and data exchanges produce delays between the instant when the vision module measures the trajectories of the robot and the obstacles and the instant when the robot starts executing the planned trajectory. Ignoring these delays would lead the robot to begin executing the planned trajectory behind the start time assumed by the planner. It then may not be able to catch up with the planned trajectory before 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 extrapolates the positions of the obstacles accordingly, as well as that of the robot if its initial velocity is non-zero. The 0.4 second contains all the delays in the system (not just the time needed for planning). It could be further reduced by running the planner on a machine faster than the Sun Sparc 10 that we are using currently.

Path optimization Because robot starts executing the trajectory 0.4 second after planning begins, the planner exploits any extra time after obtaining a first trajectory to generate additional milestones and keeps track of the best trajectory generated. The cost function used to compare trajectories is $\sum_{i=1}^{i=k}(u_i + b)\delta_i$, 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, δ_i is the duration of the ith segment, and b is a constant. This cost combines fuel consumption with execution time. A larger b yields a faster motion, while a smaller b yields less fuel consumption. In our experiments, the cost of trajectories was reduced, on average, by 14% with this simple improvement.

Sensing errors The obstacle trajectories are assumed to be straight lines at constant velocities. However, inaccuracy in the measurements by the vision module and assymmetry in air-bearings cause actual trajectories to be slightly different. The planner deals with these errors by increasing the radius of each moving obstacle by an amount $\xi V_{\max} t$, where t denotes time, $V_{\max}$ is the measured velocity of the obstacle, and ξ is a constant.

Safe-mode planning If the planner does not find a trajectory to the goal within the allocated time, we find 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 $t_e \geq T_{\mathrm{esc}}$ for some T_{esc}. An escape trajectory corresponds to any acceleration-bounded, collision-free motion in the workspace 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 also 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 concurrently both a normal and an escape trajectory. In our experiments, the modification slowed down the planner by about 2%.

Trajectory tracking The trajectory received by the robot specifies the position, velocity, and acceleration of the robot at all times. A PD-controller with feedforward is used to track this trajectory. Maximum tracking errors are 0.05 m and 0.02 m/s. The size of the disc modeling the robot is increased by 0.05 m to ensure safe collision checking by the planner.

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, as well as 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 every trial, a trajectory was computed within the allocated time. Figure 9 shows snapshots of the robot during one test.

Several constraints limited the complexity of the planning problems which we could test. Two are related to the testbed itself: the size of the table relative to the robot and the obstacles, and the robot's small acceleration. The other two constraints result from the design of our system: the requirement that obstacles move along straight lines and hence do not collide with each other, and the relatively high uncertainty on their movements, which forces the planner to grow the obstacles and thus reduce free space. To eliminate the last two constraints, we introduced on-the-fly replanning.

7.4 On-the-fly replanning

Whenever an obstacle leaves the disc in which the planner believes it lies (because either the error on the predicted motion is larger than expected, or the obstacle's direction of motion has changed), the vision module alerts the planner. The planner recomputes a trajectory on the fly as if it were a normal planning operation within the same time limit, by projecting the state of the world 0.4 second into the future. On-the-fly replanning makes it possible to perform much more complex experiments in the testbed. We show two examples below.

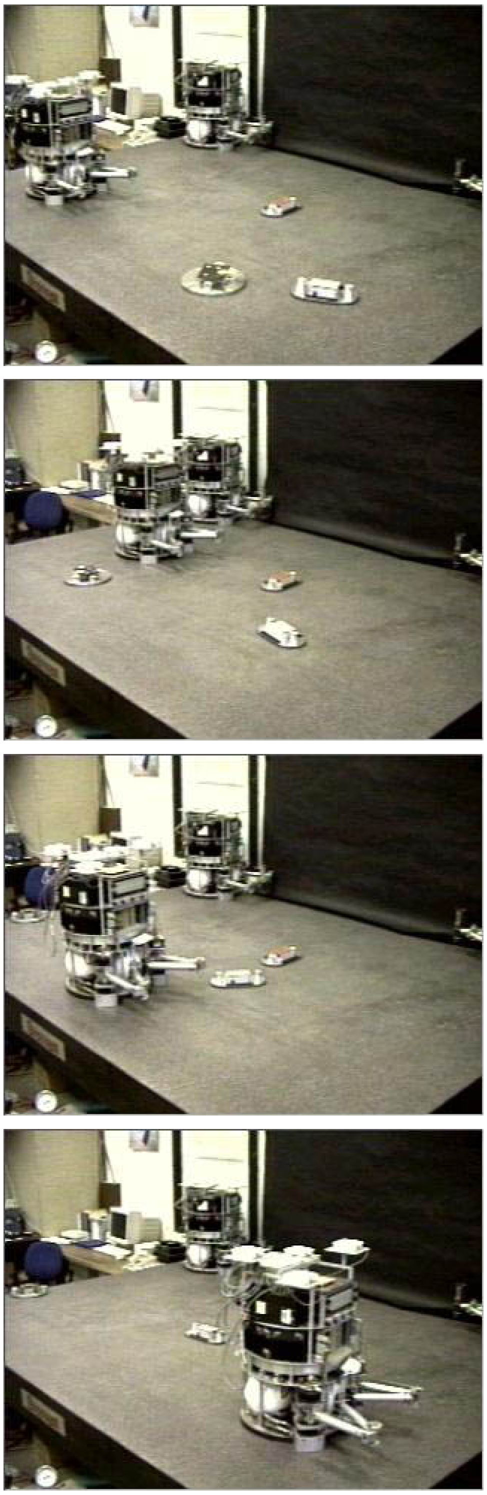


Figure 9: *Snapshots of the robot executing a trajectory*

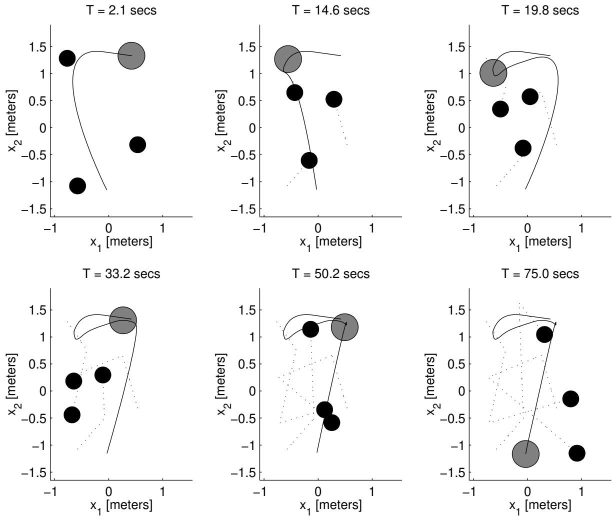


Figure 10: *Simulation example with replanning*

In the example in Figure 10, eight replanning operations are performed over the 75 seconds taken by the robot's motion. Initially the robot moves to the left to reach the goal at the bottom middle. At 14 second, the upper-left obstacle changes course, forcing a replan (snapshot 2). Soon after, the motion of the upper-right obstacle changes, forcing the robot to reverse direction and approach the goal from the other side of the workspace (snapshot 3). In the remaining time, new changes in obstacle motion cause the robot to pause (tight angle in snapshot 5) before a straight trajectory to the goal is possible (snapshot 6).

The efficacy of the replanning procedure in the testbed is demonstrated by the example in Figure 11. In snapshot 1, the middle obstacle is stationary, while the two outer obstacles are moving towards the robot. The robot attempts to move from the back middle to the front middle of the workspace. In snapshot 2, the robot dodges the faster-moving obstacle from the left, in order to let it pass by and then proceeds toward the goal. In snapshot 3, that obstacle is redirected to block the trajectory of the robot, causing it to slow down and stay behind the obstacle to avoid collision. Snapshot 5 shows the leftmost obstacle being redirected again, this time towards the robot's goal. The robot follows this obstacle and move slowly towards the goal. Before snapshot 7, however, the rightmost obstacle is directed back towards the robot, forcing the robot to wait and let it pass (snapshot 8). Finally, in the last snapshot,

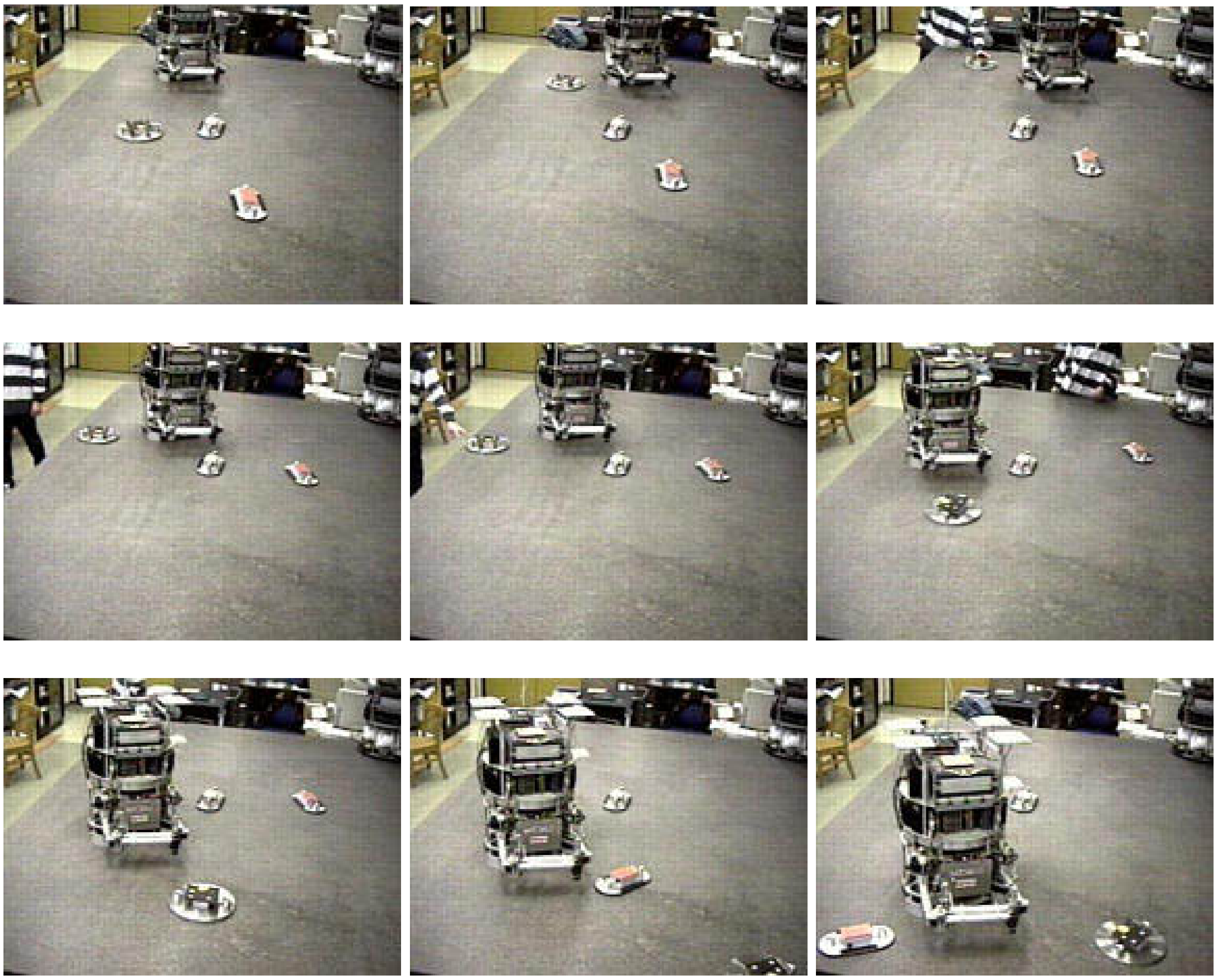


Figure 11: *An example with the real robot using replanning*

the robot attains the goal. The entire motion lasted 40 seconds. Throughout this example, other replanning operations were performed due to errors in the measurement of the obstacle trajectories (We set ξ such that sensing errors caused replanning to occur, on average, every 5 seconds). However, none resulted in a major redirection of the robot.

8 Conclusion

We have presented a simple, efficient randomized planner for kinodynamic motion planning problems in the presence of moving obstacles. Under the expansiveness assumption, we have formally proven that the planner is probabilistically complete and converges quickly when a solution exists. This proof also applies to robots that are not locally controllable. The planner was tested successfully with simulated and real robots. The experiments in the hardware robot testbed demonstrate that the planner remains effective despite various delays and uncertainties inherent to an integrated system interacting with the physical world. They also show that the planner can be used in real-time when obstacle trajectories are not known in advance.

In the future, we plan to apply the planner to environments with more complex geometry. Geometrical complexity essentially increases the cost of collision-checking, but hierarchical techniques deal with this issue well. In [11], a similar, but simpler planner was successfully applied to compute geometric disassembly paths with CAD models having up to 200,000 triangles. Another issue that we would like to investigate is the reduction of the standard deviation of the planning time. We suspect that the long thin tail shown in Figure 5 is typical of all PRM planners developed so far. But it seems more critical to reduce it for single-query planners, since such planners are more likely to be used interactively or in real-time than multi-query ones.

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

[1] P. K. Agarwal. Range searching. In *Handbook of discrete and computational geometry*, J.E. Goodman and J. O'Rourke (eds.), CRC Press, Chap. 31, p. 575–598, 1997.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*, P.K. Agarwal, L.E. Kavraki, and M.T Mason (eds.), A K Peters, p. 155–168, 1998.

[3] J. Barraquand and J. C. Latombe. Robot motion planning: a distributed representation approach. *Int. J. of Robotics Research*, MIT Press, 10(6):628-649, 1991.

[4] 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.

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

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

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

[8] P. Fiorini and Z. Shiller. Time optimal trajectory planning in dynamic environments. *Proc. IEEE Int. Conf. on Robotics and Autom.*, p. 1553–1558, 1996.

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

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

[11] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Comp. Geometry and Applications*, 9(4-5):495–512, 1999.

[12] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *Proc. IEEE Int. Conf. on Robotics and Autom.*, p. 2–7, 1989.

[13] M.H. Kalos and P.A. Whitlock. *Monte Carlo Methods*, Vol. 1, John Wiley & Son, 1986.

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

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

[16] L.E. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. *Proc. IEEE Int. Conf. on Robotics and Autom.*, p. 2138–2145, 1994.

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

[18] L.E. Kavraki, P. Švestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Tr. Robotics and Autom.*, 12(4):566–580, 1996.

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

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

[21] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *Proc. IEEE Int. Conf. on Robotics and Autom.*, p. 473–479, 1999.

[22] Z. Li, J. F. Canny, and G. Heinzinger. Robot motion planning with nonholonomic constraints. In *Robotics Research: The 5th Int. Symp.*, H. Miura et al. (eds.), MIT Press, p. 309–316, 1989.

[23] K. M. Lynch and M. T. Mason. Stable pushing: mechanics, controllability, and planning. *Int. J. of Robotics Research*, 15(6):533–556, 1996.

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

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

[26] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. *Algorithms for Robotic Motion and Manipulation*, J.P. Laumond and M. Overmars (eds.), A.K. Peters, p. 79–96, 1997.

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

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

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