TO APPEAR IN
IEEE International Conference on Robotics and Automation, 2000

Kinodynamic Motion Planning Amidst Moving Obstacles

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

*Department of Aeronautics & Astronautics †Department of Computer Science
Stanford University
Stanford, CA 94305, U.S.A.

Abstract

This paper presents a randomized motion planner for *kinodynamic asteroid avoidance* problems, in which a robot must avoid collision with moving obstacles under kinematic, dynamic constraints and reach a specified goal state. Inspired by probabilistic-roadmap (PRM) techniques, the planner samples the state$\times$time space of a robot by picking control inputs at random in order to compute a roadmap that captures the connectivity of the space. However, the planner does not precompute a roadmap as most PRM planners do. Instead, for each planning query, it generates, on the fly, a small roadmap that connects the given initial and goal state. In contrast to PRM planners, the roadmap computed by our algorithm is a directed graph oriented along the time axis of the space. To verify the planner's effectiveness in practice, we tested it both in simulated environments containing many moving obstacles and on a real robot under strict dynamic constraints. The efficiency of the planner makes it possible for a robot to respond to a changing environment without knowing the motion of moving obstacles well in advance.

1 Introduction

In this paper, we consider *kinodynamic asteroid avoidance problems*, a class of motion planning problems that take into account kinematic and dynamic constraints on robots, as well as moving obstacles in the environment. We present a simple, efficient randomized algorithm for kinodynamic asteroid avoidance problems and demonstrate its efficiency in both simulation and hardware implementation (Figure 1).

The primary motivation of our work is to control rigid-body space robots at the task level. Space robots are often under severe dynamic constraints due to limited actuator forces and torques. They perform various tasks, including inspection and assembly, amid moving obstacles (e.g., other robots and astronauts). Automated means to control their motion are needed in order to free astronauts from the tedious task of teleoperation.

The algorithm that we propose is general and not limited to space robots. In addition to the experiments that will be described in Sections 3-4, it has also been applied to nonholonomic vehicle navigation [10]. Another important potential application of our planner is the design and control of complex, multi-robot manufacturing cells.

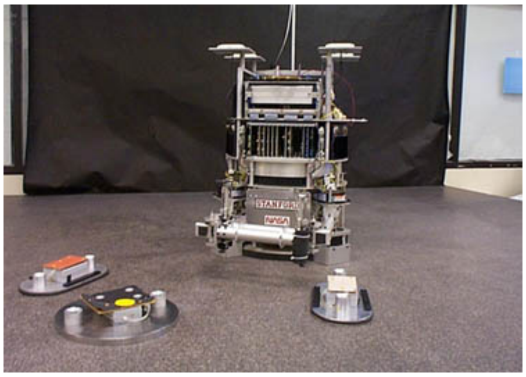

Figure 1. The testbed for the planner. The robot and obstacles float frictionlessly on a granite table via air-bearings.

kinodynamic motion planning and asteroid avoidance have both been separately investigated in the literature.

Kinodynamic planning [6] refers to planning problems in which a robot's dynamics must be taken into account. One approach divides the solution into two main steps [4, 19]. First, ignore the dynamic constraints and compute a collision-free path. Second, deform this path into a trajectory that conforms to the dynamic constraints with an optimization procedure. However, the final trajectory thus obtained may be far from optimal, a drawback that would seriously complicate the extension of the approach to environments with moving obstacles. Another approach is to discretize the robot's state space and search it for a trajectory directly, using dynamic programming [6]. This approach offers provable performance guarantees, but is only applicable to robots with few degrees of freedom (dofs) – typically 2 or 3 – since the size of the discretization grid grows exponentially with the number of dofs.

Asteroid avoidance problems require planning trajectories among moving obstacles with known trajectories [17]. The robot's velocity and acceleration may or may not be upper-bounded. Asteroid avoidance problems are provably hard, even with a small number of dofs [17, 5]. Effective algorithms [7] exist in some specific cases, but they usually do not consider constraints on the robot's motion other than an upper bound on its velocity.

The algorithm proposed in this paper is inspired by the success of probabilistic roadmap (PRM) techniques for planning geometric paths of robots with many dofs [1, 11, 13]. A PRM planner computes a path by sampling a robot's configuration space at random and connecting the sampled configurations, called *milestones*, by simple canonical paths (typically, straight-line segments in the configuration space). The result is a undirected graph called a probabilistic roadmap. Most PRM planners precompute a roadmap [13] in order to process queries as fast as possible; others compute a new roadmap for each query [11] in order to deal with changing environments more efficiently. It can be shown that under reasonable geometric assumptions on the configuration space, a small number of milestones are sufficient to capture the connectivity of the space with high probability [12, 11]. Despite the success of PRM techniques, their use in nonholonomic and kinodynamic motion planning [20] has been limited, because PRM planners require appropriate canonical paths to connect two given milestones. Constructing such canonical paths in the presence of nonholonomic and dynamic constraints is only possible for relatively simple robots.

To overcome this difficulty, our planner incrementally builds a new roadmap in the state$\times$time space of a robot for each planning query. A state encodes both the configuration and the velocity of the robot. To sample a new milestone, the planner selects a control input at random from the set of admissible controls and integrates the equations of motion under this control input from an existing milestone for a short duration of time. By construction, the trajectory thus obtained automatically satisfies the motion constraints. If it does not collide with the obstacles, the state at the end of the trajectory is stored in the roadmap as a milestone. This iterative process yields a directed tree-shaped roadmap rooted at the initial state and oriented along the time axis. The planner terminates when a milestone is close enough to the goal state (we will be more precise about this in Section 2).

The idea of generating a new milestone by selecting a control input and integrating the equations of motion, rather than directly sampling the configuration (or the state) space was originally proposed and applied in [3] to solve nonholonomic planning problems with a deterministic sampling strategy, and it was recently used in [14] to solve kinodynamic problems in static environments with random-sampling techniques. However, the fact that our planner produces roadmaps in the form of directed graphs (trees, to be more precise) oriented along the time axis makes our planner quite different from previous PRM planners.

We experimented with our planner both in simulated environments and on a real robot. In simulated environments, we tested it on difficult motion-planning problems involving many moving obstacles. With a real robot, we verified that the planner can be integrated into a larger system. In our hardware experiments, a vision system estimates the motion of obstacles, which are assumed to move with constant linear velocities, just before planning; the planner must then compute a collision-free trajectory in a fixed amount of time (0.25 s). The planner can also recompute the trajectory on the fly in response to any change in the motion of obstacles. For the experiments reported here, we modeled the robot as a disc moving in two dimensions, but we also successfully tested the planner on a six-dof articulated nonholonomic robot system in static environments [10]. These additional experiments, as well as previous results on the performance of PRM planners on many-dof robots, indicate that our planner will scale up well with the number of dofs of the robot.

The rest of the paper is organized as follows. Section 2 describes the planner. Section 3 presents experimental results in simulation. Section 4 describes the implementation and experimental results of our planner on a real robot. Finally, Section 5 discusses current and future research.

2 Description of the Planner

State space formulation We consider a robot whose motion is described by an equation of the form

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

where $q \in \mathcal{C}$ is the state of a robot, $u \in \Omega$ is the control input, f is a smooth function, and $\dot{q}$ is the derivative of q with respect to time. The set $\mathcal{C}$ is the state space of the robot, which contains all the distinguishable states that the robot may be in at any given time. The set Ω represents the control space, which contains all admissible values for the control input. Eq. (1) specifies $\dot{q}$, the rate of change of the robot's state over time, as a function of the current state q and the control input u. With no loss of generality, we assume that $\mathcal{C}$ and Ω are subsets of R^n and R^m, respectively. Eq. (1) is quite general and covers many robots with complex nonholonomic and dynamic constraints.

In the version of the planner presented here, the robot is modeled as a disc with point-mass, non-dissipative dynamics. It translates in a plane among static and moving obstacles. Let (x_1, x_2) and $(\dot{x_1}, \dot{x_2})$ denote the position and velocity of the robot. We define the state of the robot as $q = (x_1, x_2, \dot{x_1}, \dot{x_2}) \in \mathrm{R}^4$. The control input u is the force exerted by the actuators; the magnitude of the force is bounded, but the orientation of the force is unconstrained. Let (u_1, u_2) denote the components of this force along the x_1- and x_2-axis, respectively. For a robot with unit mass, Newton's law yields the following control equations:

$$\ddot{x}_1 = u_1 \qquad \ddot{x}_2 = u_2,$$

where $\ddot{x}_1$ and $\ddot{x}_2$ are the components of the robot's acceleration. The bound on the control input leads to the additional constraint $u_1^2 + u_2^2 \leq M$, which defines Ω as a subset of R^2. We could also bound the robot's velocity as well.

A planning query is specified by an initial and a goal state$\times$time, denoted by $(q_{\rm init}, t_{\rm init})$ and $(q_{\rm goal}, t_{\rm goal})$, respectively. A solution to this query is a finite sequence of fixed control inputs, each applied over some time interval, such that these inputs induce a collision-free trajectory from state $q_{\rm init}$ at time $t_{\rm init}$ to state $q_{\rm goal}$ at time $t_{\rm goal}$. In our planner, we set $t_{\rm init}$ to be 0 and we constrain $t_{\rm goal}$ to be in some given interval $I_{\rm goal}$, meaning that any arrival time $t_{\rm goal}$ in this interval is acceptable as long as no collision occur in the interval $[0, t_{\rm goal}]$.

In general, the robot's control equations may contain dynamic couplings among dofs and dissipative terms. The algorithmic principles of the planner described below, and most of the implementation, would remain unchanged.

Roadmap construction Our planner processes a query by iteratively expanding a tree T of milestones (the roadmap) generated at random, in a way similar to the geometric path planner presented in [11]. Here, however, T is built in the state$\times$time space of the robot, instead of its configuration space. The sampling strategy is also different in order to deal with the constraints on the robot's motion.

At each iteration, we obtain a new candidate milestone (q', t') by first picking a milestone (q, t) already in T and an admissible value u of the control input at random. Then the robot's equations of motion are integrated from (q, t) with the input u over a duration δ, also selected at random from a given interval $[0, \delta_{\rm max}]$; hence, $t' = t + \delta$. The trajectory between q and q' is checked for collision using the discretization technique given in [2] and adapted to deal with moving obstacles. If no collision is detected and t' is smaller than the latest arrival time in $I_{\rm goal}$, (q', t') is accepted as a new milestone in T, with a directed edge from (q, t) to (q', t'). The selected control value u is stored with the edge. This way, the kinodynamic constraints of the robot are naturally enforced. If a collision is detected, (q', t') is simply discarded. If there is no valid trajectory from $q_{\rm init}$ to $q_{\rm goal}$, then the planner would not terminate. Therefore, we place a limit on the number of iterations that it performs.

The above sampling technique does not allow the planner to achieve the goal state $q_{\rm goal}$ precisely. To deal with this issue, whenever a new milestone (q', t') is added to T, the planner checks whether the third-order spline connecting (q', t') to $(q_{\rm goal}, t_{\rm goal})$, for some $t_{\rm goal}$ in $I_{\rm goal}$, is collision-free. If so, $(q_{\rm goal}, t_{\rm goal})$ is inserted into T, with an edge pointing from (q', t') to $(q_{\rm goal}, t_{\rm goal})$, and the planner exits with success. The net effect of using spline connection is to enlarge the goal into a relatively large *endgame region* that the sampling technique can eventually attain with high probability. Other endgame connections could be considered, but for our simple acceleration-bounded robot, the third-order spline between two given states is unique and easily computed.

Another important issue for our planner is to avoid an oversampling of any region of the state$\times$time space, in particular, around $(q_{\rm init}, 0)$. Ideally we would like the milestone distribution to converge progressively toward a uniform distribution. Our planner handles this issue by selecting at each iteration the milestone (q, t) to expand with a probability inversely proportional to the number of other other milestones within some predefined distance of (q, t). Another technique proposed in [14] consists of picking a state uniformly at random in the state space and choosing the milestone in T that is the closest to this state.

Several heuristics could be used to bias the randomized construction of the tree T. For example, at each iteration, to choose a milestone (q, t) to be expanded, one may use a probability distribution that favors the states that are close to $q_{\rm goal}$ and select control input to generate a state that is even closer to the goal. However, the effectiveness of any such heuristics depends on the kind of planning problems submitted to the planner. The suggested heuristics might not work well if obstacles are elongated barriers requiring long detours to reach the goal. Our planner uses no biasing heuristics.

To perform collision checking appropriately, our planner needs to know the motion of obstacles. So in our experiments with a real robot, we use a vision system to measure the linear velocities of moving obstacles at 60 *Hz* just before the planning starts. The planner must then complete the computation of a trajectory within 0.25 s (which has been the case in almost all our experiments). Thus $t_{\rm init}$, the initial time of the query is set to the current time plus 0.25 s.

Implementation details Some additional details need to be specified to complete the description of the planner.

Milestone selection. A simple method is used to avoid oversampling any region of the state$\times$time space. The two-dimensional configuration space of the robot is partitioned into an array of bins of equal sizes. Whenever the planner chooses a new milestone to insert into T, it also adds the milestone to the corresponding bin. At each iteration, the planner selects the milestone (q, t) to expand by picking a bin at random and a milestone from this bin. This corresponds approximatively to selecting a milestone with probability inversely proportional to the local density of milestones. We could estimate the local density of milestones in the state$\times$space more accurately by range search techniques [8], but range search is much more complex to implement. In addition, for a robot moving in a plane, our current implementation is also more efficient than range search. We regard it as a good trade-off for the resulting slightly non-uniform distribution.

Control selection. The control input u is a constant acceleration of magnitude between 0 and 0.036 m/s^2 and direction between 0 and 2π. At each iteration the magnitude and the direction of u are selected from their respective intervals, independently and uniformly at random. The maximal integration time δ_{max} is set to be 6.0 s.

Endgame connection. The connection of each new milestone inserted into T to the goal state is tested for a maximum of K different arrival times randomly selected in I_g. In all the experiments reported below, K was set to 10.

Path optimization. The trajectory computed by a randomized planner may contain unnecessary zig-zags, because of the random steps taken by the algorithm. For this reason, the planner may choose not to exit after finding the first valid trajectory between $(q_{\text{init}}, t_{\text{init}})$ and $(q_{\text{goal}}, t_{\text{goal}})$. Instead it continues sampling more milestones until a number of valid trajectories are found and returns the one with the minimum cost. In our case, the cost function takes into account both the time and the amount of thrust required to execute the path.

The robot and the workspace. The workspace is a rectangle of 3 m by 4 m. The robot is modeled as a circular disc with a radius of roughly 25 *cm*. Obstacles are also circular discs with varied radii, mostly between 10 and 15 *cm*. The obstacles move at different, but fixed linear velocities. The velocities of obstacles range from 0 to 0.2 m/s.

Software. The planner is written in C. It runs both on a PC and on a Sun Sparc workstation.

Guarantees of performance In [12, 11] it is shown that under reasonable geometric assumptions on the free space $\mathcal{F}$ (collision-free subset of the robot's configuration space), a PRM path planner generating milestones distributed uniformly at random over $\mathcal{F}$ can find a collision-free path with high probability, whenever one exists. More precisely, the assumptions state that each configuration in $\mathcal{F}$ "sees" a significant portion of $\mathcal{F}$ (a property called ϵ-goodness) and that no two regions of $\mathcal{F}$ are connected only by a very narrow passage (a property called expansiveness). Under these two assumptions, the probability that the PRM planner fails to find a path between two free configurations lying in the same connected component of $\mathcal{F}$ decreases exponentially with the number of sampled milestones.

We have extended this result to our current planner that deals with kinematic, dynamic constraints as well as moving obstacles. This extension generalizes the definition of ϵ-goodness and expansiveness to the robot's state$\times$time space, by taking into account that the reachability relationship between points in this new space is no longer symmetrical. However, the intuition behind the new definitions and results remains similar to that of geometric path planning. For lack of space, we refer the reader to [10] for a complete discussion of these results.

The exponential convergence of the planner requires a uniform distribution of the milestones. This is why the planner must avoid oversampling any region in the state$\times$time space.

Table 1. Running time and the number of milestones (including endgame connections) for computed examples.

Example	time		milestones	
	mean	std	mean	std
1	0.249	0.264	2008	2229
2	0.270	0.285	1946	2134
3	0.002	0.005	22	25

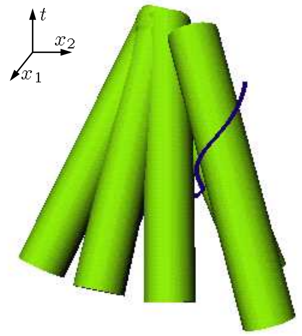

Figure 2. The configuration$\times$time space for Example 2. The cylinders indicate the moving obstacles in the configuration$\times$time space. The thick line marks the trajectory of the robot.

3 Experiments in Simulation

We now present experimental results obtained with our planner in simulated environments. As previously indicated, our main goal is to verify that the planner can solve efficiently tricky problems in an environment with a substantial number of obstacles. Such problems are very difficult to set up within the physical limitation of our real robot testbed. The simulation problems were crafted by hand to require delicate maneuvers by the robot.

In our experiments, each obstacle moves at constant linear velocity. To simplify the implementation, collision among obstacles are ignored. As a result, two obstacles may temporarily overlap without changing their respective courses. When an obstacle reaches the boundary of the robot's workspace, it just stays there and is no longer deemed a threat to the robot. The planner is given the obstacle trajectories. Unlike the experiments with the real robot, planning time is not limited. Figures 3-5 show three examples computed by the planner. In each case, we ran the planner 100 times independently for the same query; the mean and standard deviation of the planning time, as well as those of the number of sampled milestones in the roadmap, are shown in Table 1. The planning times were measured on a PC with a 550 *MHz* Pentium-III processor. The planner successfully returned a trajectory in every run.

Example 1. The robot (grey disc) must move from the lower edge of the workspace to the upper edge in the presence of 10 moving obstacles (black discs). The path computed for the robot is shown in solid line, and the paths of the obstacles, in dashed lines (Figure 3).

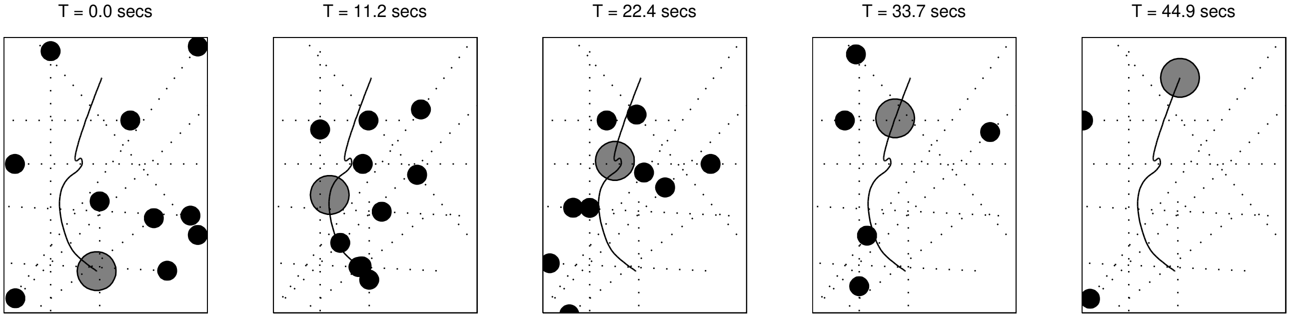

Figure 3. A robot moves among many moving obstacles. The grey disc indicates the robot. Black discs indicate obstacles. The solid and dotted lines indicate the trajectories of the robot and obstacles respectively.

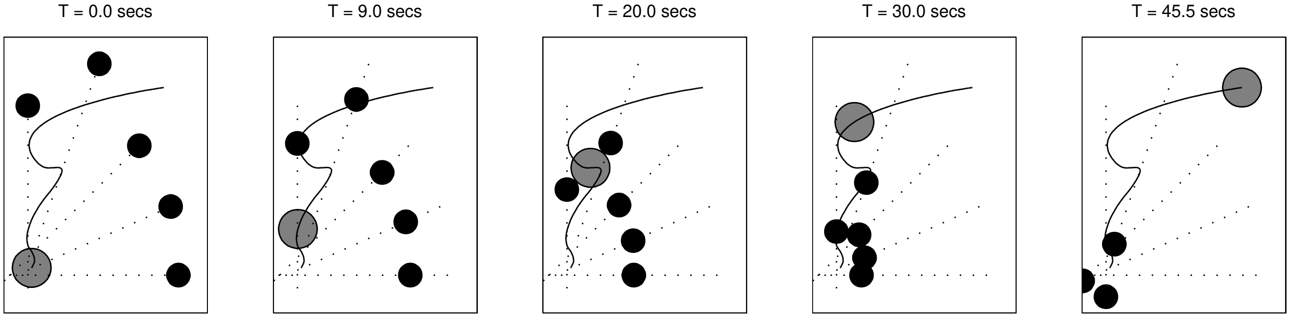

Figure 4. A robot moves among "hostile" obstacles.

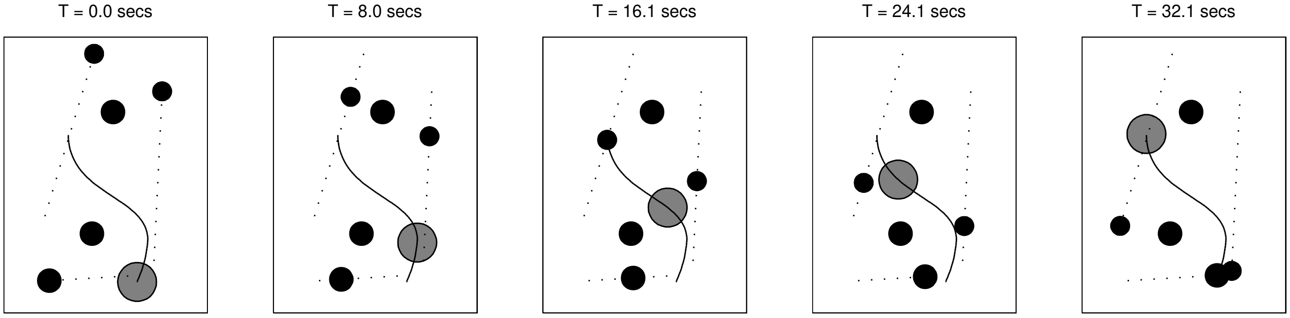

Figure 5. An representative environment for space robotics missions.

Example 2. The five moving obstacles in this example allow a single small opening for the robot to escape collision with the obstacles that all converge toward the initial position of the robot (Figure 4). Figure 2 shows the corresponding configuration×time space. The robot maps into this space as a point (x_1, x_2, t); the obstacles are grown by the robot's radius and are extended into cylinders along their linear trajectories. The acceleration constraint makes it impossible for the robot to move through most of the free space and forces the feasible trajectory (as the one is shown in Figure 2) to pass through the small gap between cylinders to attain the goal. The environment is considerably more "hostile" than that expected in most space robotics applications.

Example 3. This example (Figure 5), is more representative of the environments that are expected to occur during typical space robotic tasks. There are two stationary obstacles near the middle of the workspace and three moving obstacles that are aimed not to collide with any other obstacle. The very small average planning time (0.002 s), confirms that in the absence of narrow passages randomized motion planners are extremely efficient.

4 Experiments on a Real Robot

Testbed description We integrated our planner with the controller of the "free-flying" robot testbed in the Stanford Aerospace Robotics Laboratory. This testbed provides a frictionless 2-D environment for testing robotics technologies for space applications. It consists of a 3 m×4 m granite table providing a flat workspace upon which a robot and a number of obstacles moving frictionlessly on air-bearings (see Figure 1). Previous work with this testbed includes multiple-robot assembly [18] and kinodynamic motion planning in static environments [15].

The robot has a roughly cylindrical shape. It is untethered and carries all of its vital systems on-board. Compressed air is used both to maintain the air-bearing and to provide propulsion. Eight horizontal thrusters are located in pairs around the circumference of the robot, providing omnidirectional thrusting. The gas tanks provide enough air for half an hour of station-keeping maneuvers or about 5 minutes of path following. On-board batteries provide power for about 30 minutes of full actuation without recharging. Robot control is performed at 60 *Hz* on a Motorola ppc2604 real-time computer.

Position sensing is performed by an overhead vision system. The measured position are accurate to $0.005\ m$. The update rate is 60 *Hz*. Velocity estimates are derived from the position data with an error of 0.005 m/s.

The planner runs off-board on a Sun Sparc Ultra10 workstation with a 333 *MHz* processor and 128 *MB* of memory.

Communications among the robot, the vision system, and the planner are implemented with 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.

Integration of the planner Running the planner on the testbed raises a number of new challenges:

Delays. The sensor measurement for the states of the robot and obstacle arrive asynchronously and incur a delay of up to 1/30 s each. The execution of the planner then takes some amount of time, and the transmission of the path to the robot takes up to another 1/60 s. If these delays are not taken into account the robot would thus start out about 0.25 s behind the plan it is attempting to execute. With acceleration limits on the vehicle it might not be possible to catch up with the planned path before collision occurs. To eliminate this problem, the planner starts planning assuming the robot will start executing the yet-to-be-computed trajectory 0.25 s into the future and extrapolates the robot's initial position if its current velocity is non-zero. If the total delay turns out to be less than that, the robot controller will wait until the delay period is over before moving along the planned trajectory. The delay of 0.25 s is quite conservative for the experiments carried out in the testbed; we expect that it can be reduced well below 0.1 s in the future.

Sensor errors. Our planner assumes that the moving obstacles move along straight-line trajectories with constant velocities as measured by the vision system. However, inaccuracy in the measurement of the vision sensor, asymmetry in the air-bearing supporting the moving obstacles, and tiny collisions with dust particles on the table all cause the actual obstacle trajectories to be slightly different from the predicted ones. To correct for these errors, the planner increases the radius of each moving obstacle as a function of time and velocity, assuming a constant velocity error term. As a result, the size of the obstacle region grows to take into account the uncertainty in determining its position.

Trajectory following. The trajectory received by the robot contains the desired position, velocity and accelerations for the motion. A PD-controller with feedforward is used to track the trajectory. A simple thrust-mapper is used to activate and deactivate the bang-bang thrusters to approximate a linear plant. Tracking errors average approximately 0.02 m with a maximum of $0.05\ m$. The size of the disc modeling the robot is increased by the maximum tracking error to ensure safe collision-checking operations.

Results Our experiments show that the planner is able to maneuver the free-flying robot successfully among static and moving obstacles on the granite table under strict dynamic constraints. In almost all trials, the trajectories were computed within the prescribed 0.25 s. Tests were performed for a number of different environments. Figure 6 shows snapshots of the robot motion in one of these tests.

The planner was tested in canonical situations to observe the robot's behavior. The robot avoided obstacles moving directly toward it, as well as obstacles moving perpendicular to the line connecting the initial and the goal position. It also showed the ability to wait for an opening when confronted with moving obstacles in the desired direction of movement and to move through openings that were less than 10 cm larger than the robot.

The major limitations of the testbed are the size of the granite table relative to that of the robot and the obstacles, the bound on the robot's acceleration, and the relatively high uncertainty in the sensor data. These constraints limit the complexity of the planning queries and the robot motions that can be tested.

On-the-fly replanning Because of the very short running times of the planner, we can use the overhead vision system to detect unexpected changes in the motion of obstacles and invoke the planner to replan a trajectory for the robot. This makes it possible to remove the assumption that the motion of obstacles is known *a priori*. It also allows us to use less conservative error bounds on the obstacle trajectories, resulting in smaller obstacle regions and increased free space for the robot to maneuver and thus further improving the running time of the planner. Experiments with replanning are reported in [10]. Currently the planner is given 0.25 s for each replanning query, but we believe that it is quite feasible to reduce the time to 0.1 s or faster and integrate the planner into the control loop of the robot.

5 Conclusion

We have presented a simple, efficient randomized planner for kinodynamic motion planning in the presence of moving

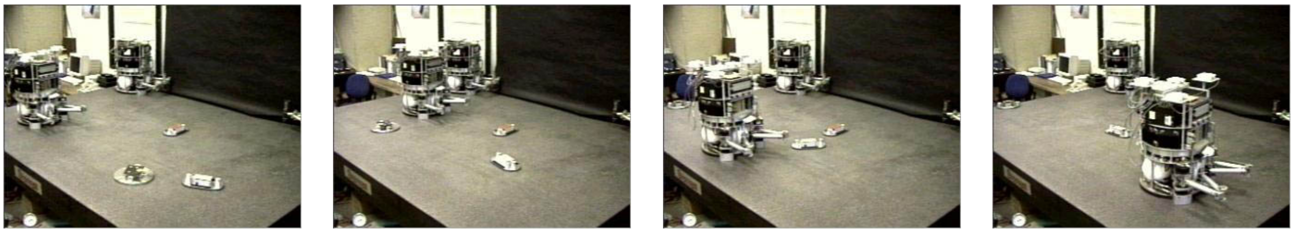

Figure 6. Snapshots of a robot executing a trajectory on the hardware testbed.

obstacles. This planner was successfully tested both in simulated environments and on a hardware testbed developed to study robotics technology for space applications. The planner was also tested on articulated nonholonomic vehicles with six dofs. For lack of space, the experiments with this system are not reported here, but the results can be found in [10]. This planner demonstrates that random-sampling techniques extend well to motion planning problems beyond pure geometric path planning.

In the future, we plan to apply our planner to objects with complex geometry in three-dimensional environments and test the planner with many-dof robots amid moving obstacles. Our previous experience with PRM planners indicates that the random-sampling planning approach scales up well with both complex geometry and many dofs. Geometric complexity essentially increases the cost of collision checking, but hierarchical techniques (e.g., [9, 16]) deal with this issue well. We have successfully applied a randomized path planner to environments with up to 200,000 triangles [11].

Another important future direction of research is to integrate the planner with the controller and the sensing modules that detect moving obstacles. The efficiency of the planner should make it possible to directly include it in the control loop.

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

References

[1] 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: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 155–168, 1998.

[2] J. Barraquand, L. Kavraki, J. C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Int. J. of Robotics Research*, 16(6):759–774, 1997.

[3] 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.

[4] 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.

[5] J. F. Canny. Some algebraic and geometric computations in pspace. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 460–467, 1988.

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

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

[8] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, New York, 1997.

[9] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 96 Conference Proceedings*, pages 171–180, 1996.

[10] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. To appear in *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[11] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2719–2726, 1997.

[12] L. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. In *ACM Symp. on Theory of Computing*, pages 353–362, 1995.

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

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

[15] D. W. Miles. *Real-Time Dynamic Trajectory Optimization with Application to Free-Flying Space Robots*. PhD thesis, Stanford University, Stanford, CA, 1997.

[16] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3324–3329, 1994.

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

[18] J. Russakow, S. Rock, and O. Khatib. An operational space formulation for a free-flying, multi-arm space robot. In *Proc. Int. Symp. on Experimental Robotics*, pages 448–457, 1995.

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

[20] 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.