SUBMITTED TO
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 and dynamic constraints. 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. The planner does not precompute a roadmap as most PRM planners do, since this requires knowledge of obstacle trajectories well in advance. Instead, for each planning query, it tries to generate a small roadmap that connects the given initial and goal states. A major difference with PRM planners is that the computed roadmap is a directed graph oriented along the time axis of the space. To verify the planner's effectiveness in practice, it was tested it both in simulated environments containing many moving obstacles and on a real robot under strict dynamic constraints.

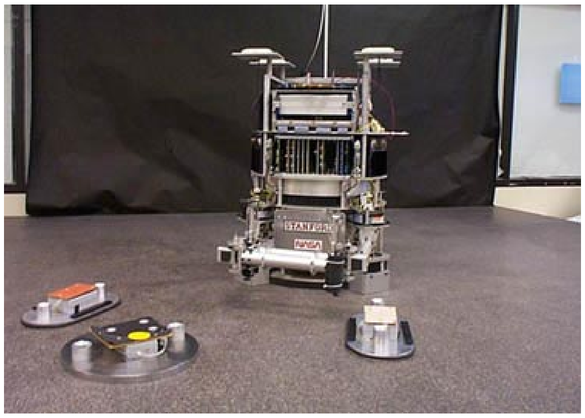
Figure 1. The testbed for our planner. The robot and obstacles float frictionlessly on a granite table.

1 Introduction

In this paper, we consider *kinodynamic asteroid avoidance problems*, a class of motion planning problems that take into account both 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 will 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 teleoperating them.

The planner that we propose is nevertheless general and not limited to space robots. In addition to the experiments that will be described in Section 3, it has also been applied to nonholonomic vehicle navigation [9]. Another important potential application of our planner is the design and control of complex, multi-robot manufacturing cells.

Kinodynamic motion planning and asteroid avoidance have both been separately investigated in the literature:

- Kinodynamic planning [6] refers to planning problems in which the robot's dynamics must be taken into account. One approach divides the solution into two main steps [4, 18]. 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 [16]. 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 [16, 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, 10, 12]. A PRM planner computes a path by sampling the robot's configuration space at random and connecting the generated samples, called *milestones*, by simple canonical paths (typically, straight-line segments in the configuration space). The result is a undirected graph called a probabilistic roadmap. Some PRM planners precompute a roadmap [12] in order to process planning queries as fast as possible; others compute a new roadmap for each new query [10] 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 [11, 10]. Despite the success of PRM techniques, the use of random sampling techniques for nonholonomic and kinodynamic motion planning has been limited [19] 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 a previously generated 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 that it reaches 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. It ends 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 algorithm, and it was recently used in [13] to solve kinodynamic problems in static environments with random-sampling techniques. On the other hand, 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 tested the planner's effectiveness both in simulated environments and on a real robot. In simulated environments, we verified that the planner can solve difficult motion-planning problems involving many moving obstacles. With a real robot, we checked that our planner can be integrated into a larger system. In our hardware experiments, the obstacles move at constant linear velocities and are detected by a vision system just before planning; the planner must then return a trajectory in a fixed amount of time (0.25 s). In all the experiments presented in this paper, the robot is modeled as a disc moving in two dimensions. We also tested the planner on a 6-dof articulated nonholonomic robot in static environments [9]. These additional experiments, as well as previous PRM results with many-dof robots, lead us to expect 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 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 $q \in \mathcal{C}$, $u \in \Omega$, f is a smooth function, and $\dot{q}$ denotes the derivative of q with respect to time. The set $\mathcal{C}$ is the state space of the robot, that is, the set of distinguishable states that the robot may be in at any given time. The set Ω represents the control space, i.e., the set of admissible values for the control input. Eq. (1) specifies the rate of change of the robot's state over time ($\dot{q}$) as a function of the 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 dynamics and/or nonholonomic constraints.

In the version of the planner presented here, the robot is modeled as a disc with point-mass, non-dissipative dynamics. This robot translates in the plane among moving and static obstacles. We let (x_1, x_2) and $(\dot{x_1}, \dot{x_2})$ denote the position and velocity of this 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 this force is bounded, but its orientation is unconstrained. Let (u_1, u_2) denote the components of this force along the x_1 and x_2 axes, respectively. Assuming a unit-mass robot, 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.

Consider the state$\times$time space $\mathcal{C} \times [0, +\infty)$. Given any point (q, t_0) in this space and a fixed control input u, we can compute the trajectory followed by the robot over the time interval $t \geq t_0$. A planning query is specified by an initial and a goal state$\times$time, denoted by (q_i, t_i) and

(q_g, t_g), respectively. A solution to this query is a finite sequence of fixed control inputs, each applied over a defined time interval, such that these inputs induce a collision-free trajectory from state q_i at time t_i to state q_g at time t_g. In our planner, we set t_i to be 0 and we constrain t_g to be in some given interval I_g, meaning that any arrival time t_g in this interval is acceptable as long as no collision occur in the interval $[0, t_g]$.

In a more general version of the planner, the robot's control equations would typically contain dynamic couplings among dofs and dissipative terms. The algorithmic principles of the planner described below, and most of the implementation, would remain unchanged. If the motion equation is not analytically integrable, we rely on numeric techniques, instead.

Roadmap construction Our planner processes a planning 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 [10]. Here, however, T is built in the state$\times$time space of the robot, instead of its configuration space (thus, it is rooted at $(q_i, 0)$). The sampling strategy is also different to deal with the kinodynamic constraints.

At each iteration, a new candidate milestone (q', t') is obtained as follows. Both a milestone (q, t) already in T and an admissible value u of the control input are picked at random. The robot's equation of motion is integrated from (q, t) with the input u, over a duration δ also selected at random from a given interval $[0, \delta_{max}]$; hence, $t' = t + \delta$. The trajectory between q and q' is checked for collision using a version of 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_g, (q', t') is included as a new milestone in T, with a directed edge from (q, t) to (q', t'). The selected control value u is associated with this edge. In 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_i to q_g, then the planner would not exit. Therefore, we place a limit on the number of iterations it performs, or on its running time.

The above sampling technique does not allow the planner to precisely achieve the goal state q_g. We solve this issue as follows. Whenever a new milestone (q', t') is added to T, the planner checks that the third-order spline connecting (q', t') to (q_g, t_g), for some t_g in I_g, is collision-free. If this is the case, (q_g, t_g) is inserted into T, with an edge pointing from (q', t') to (q_g, t_g), and the planner exits with success. The effect of considering this 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 is unique (for a given t_g) and easily computed.

Another issue is to avoid an oversampling of any region of the state$\times$time space, especially around $(q_i, 0)$. Ideally we would like the milestone distribution to progressively converge 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 milestones already in T within some predefined distance of (q, t). Another technique proposed in [13] 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.

In practice, our planner turns out to be fast and able to find solutions for difficult problems. To perform collision checking appropriately, the obstacle trajectories are given to the planner, but in our experiments with a real robot the obstacles are detected and their linear velocities measured by a vision system at 60 *Hz* frame rate just before planning. The planner must then complete the computation of a trajectory within 0.25 s (which has been the case in almost all our trials) and thus selects t_i to be the current time augmented by 0.25 s.

Several heuristics could be used to bias the randomized construction of the tree T. For example, the choice of the milestone (q, t) to be expanded at each iteration could be done using a probabilistic distribution that favors the states that are closer to q_g and the control input could be chosen to generate a milestone that is even closer to the goal. However, the effectiveness of any such heuristics is likely to depend on the kind of planning problems submitted to the planner (for example, the suggested heuristics might not work well if obstacles are long barriers requiring long detours to reach the goal). Our planner uses no biasing heuristics.

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

Robot and workspace. The workspace is a rectangle of 3 m by 4 m. The robot is modeled as a circular disc with a radius of 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 velocity of an obstacle ranges between 0 and 0.2 m/s.

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

Milestone selection. A simple weighting function is used to avoid oversampling any region of the state$\times$time space, while avoiding the cost of computing the density of milestones around each milestone. The two-dimensional configuration space of the robot is partitioned into an array of square bins of equal size. Whenever a new milestone is inserted in T, the planner determines the bin to which it belongs. At each iteration, the planner selects the milestone (q, t) to expand by successively picking a bin at random and a milestone from within this bin. This corresponds approximatively to selecting a milestone with a probability inversely proportional to the local density of milestones, but it is much faster to compute. We could have set the bins in the higher dimensional state$\times$time space, but this was not determined to be useful for our implementation.

Endgame connection. The connection of each milestone inserted in 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.

Software. The planner is written in C. It runs both on a Pentium-III computer (simulation) and on a Sun Sparc Ultra10 (connected to real robot).

Guarantees of performance In [11, 10] 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 F can find a collision-free path with high probability, whenever one exists. More precisely, the geometric assumptions are that each configuration in F "sees" a significant portion of $\mathcal{F}$ (a property called ϵ-goodness) and that no two regions of $\mathcal{F}$ communicate by a narrow passage (a property called expansiveness). Under those 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 toward 0 with the number of milestones in the roadmap.

We have extended this result to our planner in [9]. This extension requires re-defining ϵ-goodness and expansiveness in the robot's state$\times$time space, by taking into account that visibility between points in that space is no longer symmetrical. But the intuition behind the new definitions and results remains the same as for geometric path planning. For lack of space, we refer the reader to [9] for a complete presentation.

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.

3 Experiments in Simulation

We now present experimental results obtained with our planner in simulated environments. As previously indicated, our main goal was to verify the planner's ability to efficiently solve tricky problems in the presence of a substantial number of obstacles. Such problems would have been quasi-impossible to set up within the physical limitations of our real robot testbed. The simulation problems were crafted by hand to require delicate maneuvers by the robot. Each obstacle moves at constant linear velocity. Collisions between obstacles are ignored, meaning that two obstacles may temporarily overlap without changing their respective courses. When an obstacle reaches the boundary of the robot's workspace, it just leaves the workspace and is no longer a threat to the robot. The planner is given the obstacle trajectories, and unlike in the experiments with the real robot, planning time is not limited. (This is equivalent to assuming that the obstacles wait for the planner to return a trajectory before moving; when the planner returns a trajectory, the time is set to 0, and both the obstacles and the robot start moving.)

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

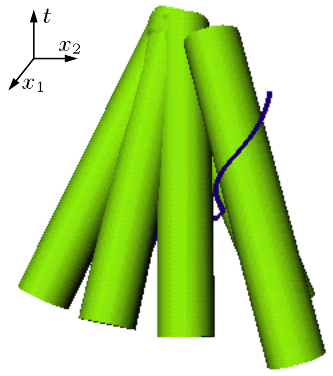

Figure 2. The configuration$\times$time space for Example 2.

We present three examples. In each example, we ran the planner 100 times for the same query with different seeds of the random number generator, and we obtained 100 different trajectories. Table 1 lists the mean and standard deviation of the planning time, as well as the mean and standard deviation of the number of milestones needed for each example. The planning times are for the planner running on a Pentium-III computer at 550 *MHz*. In every run, the planner successfully returned a trajectory.

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 obstacle, in dashed lines Figure 3.

Example 2. The five moving obstacles in this example offer 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$\times$time space. The robot maps into this space

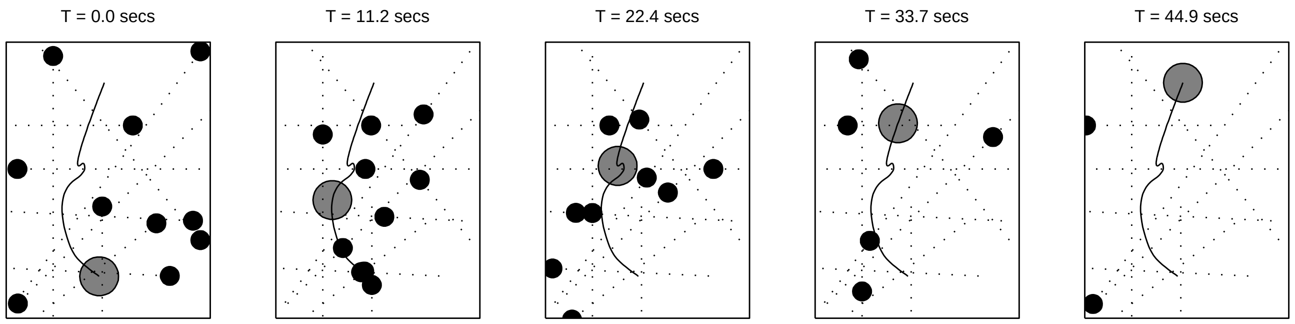

Figure 3. A robot moves among many moving obstacles. The grey disc indicates the robot. Black discs indicate obstacles. The solid and dotted lines indicate the trajectories of the robot and obstacles respectively.

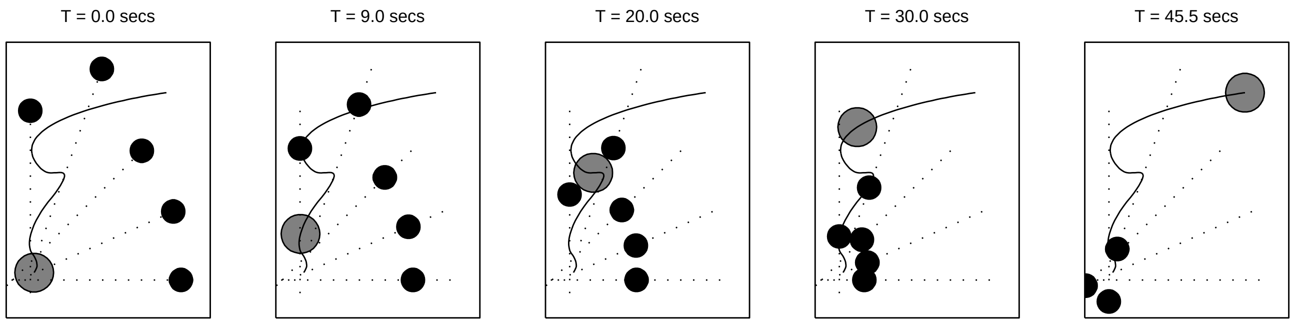

Figure 4. A robot moves among "hostile" obstacles.

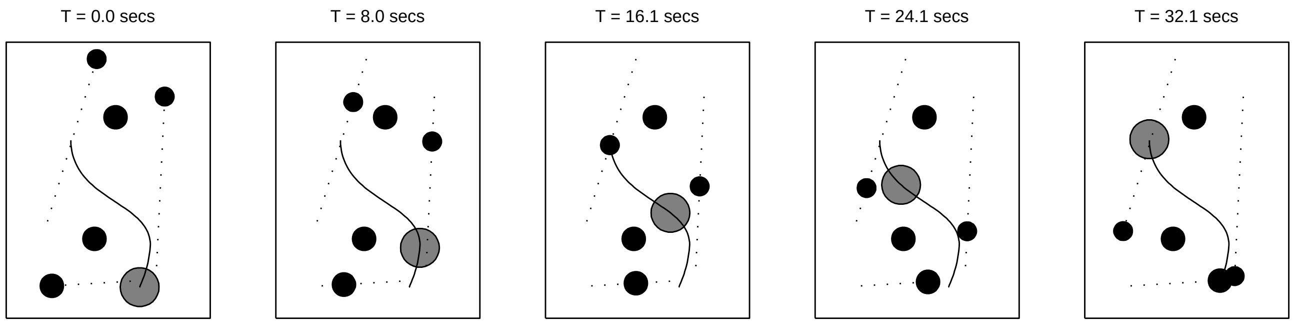

Figure 5. An representative environment for space robotic missions.

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 any feasible trajectory (one is shown in the figure) to pass through the small gap between cylinders to attain the other side of the free space. The environment is considerably more "hostile" than that expected in most applications.

Example 3. This example (Figure 5), is more representative of the environments that are expected to occur during typical space robotic missions. 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 (.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 connected our planner to the controller of the "free-flying" robot testbed in the Stanford Aerospace Robotics Laboratory. This testbed provides a frictionless 2D environment for testing robotics technologies for space applications. It consists of a 3 $m \times 4$ m granite table providing a flat workspace upon which a robot and a number of obstacles moving frictionlessly on an air-bearing (see Figure 1). Previous work with this testbed includes multiple-robot assembly [17] and kinodynamic motion planning in stationary environments [14].

The robot is roughly cylindrical in shape. It is untethered, carrying 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 capability. Robot control is performed at 60 *Hz* by a Motorola ppc2604 real-time computer. Communications with the vision system, the planner and user interface are done over radio ethernet. On-board batteries provide power for about 30 minutes of full actuation without recharging. The gas tanks provide enough air for half an hour of station-keeping maneuvers or about 5 minutes of path following.

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.

Position sensing is performed by an overhead vision system. The position data output by this system are accurate to $0.005\ m$. The update rate is 60 *Hz*. Velocity estimates are derived from this position data to provide velocity estimates with a standard deviation of 0.005 m/s.

Planning is performed off-board on a Sun Sparc 10 running at 333 *MHz* with 128 *MB* of memory.

Integration of the planner Running the planner on the hardware testbed raised additional challenges:

Delays. A number of delays exist in any robot system. The robot and obstacle state data arrive asynchronously and incur a delay of up to 1/30 s each. The execution of the planner then takes an uncertain 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 ends up being less than that, then 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 we can carry in the testbed; we expect that it can be reduced well below 0.1 s in the future.

Sensor errors. The planner assumes perfect prior knowledge of the obstacle trajectories in order to compute the robot trajectory. The trajectories of our uncontrolled moving obstacles are assumed to be straight lines at constant velocities. But inaccuracy in the measurement of the velocities by the vision sensor, asymmetry in the air-bearing, and tiny collisions with dust particles on the table all cause the actual obstacles 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. In this way the apparent footprint of the obstacle grows approximately as its uncertainty in position grows.

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 was increased by the maximum tracking error to ensure safe collision-checking operations.

Results The planner was able to maneuver the free-flying robot successfully among static and moving obstacles on the granite table. In almost all trials, the trajectories were computed within the 0.25 s planning constraint. Tests were performed for a number of different environments. Figure 6 shows snapshots of the robot motion during one of the 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 to 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 constraints on testing were the size of the granite table relative to that of the robot and the obstacles, the rigorous limit on the robot's acceleration, the requirements that obstacles not collide, and the relatively high uncertainty on their movements. These constraints limited the complexity of the planning queries and the robot motions which could be tested.

We expect to bring several improvements in the future. In particular, because of the extremely short planning times provided by the planner, we could use the overhead vision system to detect unexpected variations in the environment and replan to accommodate them. We believe that it will be quite feasible to run the planner at 10 *Hz*, or faster, and to integrate it into the control loop of the robot. This would make it possible to replan after a collision between two obstacles. It would also allow us to use less conservative error bounds on the obstacle trajectories, which would result into smaller obstacle discs and increased free space for the robot to maneuver.

5 Conclusion

We have presented a simple, efficient randomized planner for kinodynamic motion planning in the presence of moving 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 an articulated nonholonomic vehicle with six dofs. For lack of space, the experiments with this robot have not been reported here, but the results can be found in [9]. This planner demonstrates that

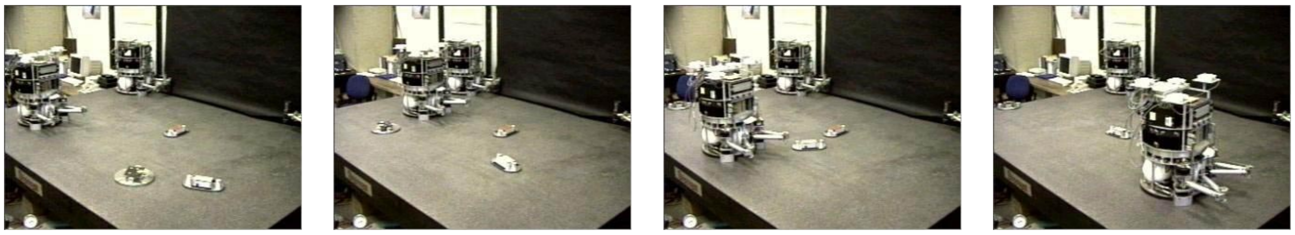

Figure 6. Snapshots of a robot executing a trajectory on the hardware testbed.

random-sampling techniques extend well to motion planning problems beyond pure geometric path planning.

In the future, we plan to extend 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., [8, 15]) deal with this issue well. We have successfully applied a randomized path planner to environments with up to 200,000 triangles [10].

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. *International Journal 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. *International Journal 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. *Journal 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] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 171–180, 1996.

[9] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Control-based randomized motion planning for dynamic environments. To appear in *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.