Proceedings of the American Control Conference
Arlington, VA June 25-27, 2001

REAL-TIME MOTION PLANNING FOR AGILE AUTONOMOUS VEHICLES

Emilio Frazzoli [1] Munther A. Dahleh [2]
Eric Feron [3]
Massachusetts Institute of Technology, Cambridge, MA 02139

Abstract

Planning the path of an autonomous, agile vehicle in a dynamic environment is a very complex problem, especially when the vehicle is required to use its full maneuvering capabilities. Recent efforts aimed at using randomized algorithms for planning the path of kinematic and dynamic vehicles have demonstrated considerable potential for implementation on future autonomous platforms. This paper builds upon these efforts by proposing a randomized motion planning planning architecture for dynamical systems in the presence of fixed and moving obstacles. This architecture addresses the dynamic constraints on the vehicle's motion, and it provides at the same time a consistent decoupling between low-level control and motion planning. Simulation examples involving a ground robot and a small autonomous helicopter, are presented and discussed.

1 Introduction

This paper is concerned with the problem of generating and executing a motion plan for an autonomous vehicle. In other terms, this paper considers developing an algorithm that enables the robot to move from its original location to a new location (presumably to perform a given task such as performing an observation or delivering a payload), while avoiding collisions with fixed or moving obstacles.

The full exploitation of the vehicle's dynamics and maneuvering capabilities often plays a crucial role in achieving the mission goals. This is true in cases in which the characteristic dimensions of the environment (e.g. the size of the obstacles, and of the gaps between them) and of the vehicle's dynamics (e.g. the turning radius) are comparable. In such cases motion planning strategies should account for the dynamics of the vehicle to the best extent possible. In other words, it is desirable that the output from the motion planning be executable by the system dynamics.

[1]Research Assistant, Department of Aeronautics and Astronautics. AIAA Member.

[2]Professor, Department of Electrical Engineering and Computer Science.

[3]Associate Professor, Department of Aeronautics and Astronautics. AIAA Senior Member

Motion planning research is a well-established discipline in Robotics.[1–4] The computational complexity of a deterministic, complete algorithm, even in the simplest cases of motion planning, grows exponentially with the number of the degrees of freedom of the vehicle.[5,6] In order to circumvent the computational complexity of deterministic, complete algorithms, a new class of motion planning algorithms, known as Probabilistic RoadMap (PRM) planners, was recently introduced.[7–12] Probabilistic Roadmap Planners are based on the construction of a graph of feasible paths (roadmap), connecting different locations (milestones) in the workspace. An incremental version of PRMs, the Rapidly-exploring Random Tree (RRT), was introduced by LaValle and Kuffner[13–15] as a means to provide rapid, single-query exploration of the environment, as well as to take into account the dynamics of the system. In Hsu, Kindel, *et al.*[16] a new incremental roadmap building algorithm is introduced that provides not only probabilistic completeness, but also performance guarantees on the algorithm. That is, the probability of the algorithm finding a solution if one exists converges exponentially fast to one with the number of planning iterations. One difficulty pointed out by the authors, however, lies with the fact that algorithm performance relies upon uniform sampling of milestones in the reachable space of the workspace. Practical implementations (which have reportedly demonstrated excellent performance[16]) rely upon uniform sampling of the system's control inputs instead, which in general does not guarantee uniform sampling of the workspace.

Motivated by these considerations, this paper proposes a new randomized, incremental path planning algorithm. This incremental roadmap building algorithm is able to effectively deal with the system's dynamics, in an environment characterized by moving obstacles. Central to this algorithm is the assumption that an *obstacle-free* guidance loop is available, which is able to steer the system from any state (including location and velocities) to any desired location at rest, assuming that there are no obstacles in the environment. This guidance loop enables uniform sampling of the workspace while generating trajectories that are executable by the dynamical system. As a consequence, this path planning algorithm satisfies the technical con-

0-7803-6495-3/01/$10.00 © 2001 AACC

ditions elicited by Hsu, Kindel *et al*[16] and therefore offers guaranteed performance, in the sense of bounds on the convergence rate to unity of the probability of correct termination. Considering real-time computation issues, the path planning algorithm provides *safety* guarantees in the sense that it provides intermediate milestones with guaranteed buffer time before a collision occurs. If possible and practical, the buffer time can be extended to infinity, thus resulting, in principle, in hard safety guarantees on the generated plan.

2 Problem formulation

This section introduces the elements used to formulate the path planning algorithm. The algorithm for path planning presupposes the existence of a closed-loop architecture that enables the guidance of the vehicle from any state to any location at rest. Thus, rather than working with an "open-loop" system, as presented in earlier publications,[13,15,16] our basic dynamical system is a closed-loop one.

2.1 System dynamics

The usual representation of the dynamics of an autonomous vehicle or robot is a set of Ordinary Differential Equations (ODEs) of the form:

$$\frac{dx}{dt} = f(x,u) \tag{1}$$

where $x \in \mathcal{X}$ is the state, belonging to a n-dimensional manifold $\mathcal{X}$ (the *state space*), and u is the control input, taking values in the set $\mathcal{U} \subseteq \mathbb{R}^m$. In some cases, additional inequality constraints of the form $F_i(x) < 0$ must be added on the state variables, to ensure safe operation of the system (e.g. flight envelope protection).

Assume that the state space $\mathcal{X}$ can be decomposed, at least locally, in the product $\mathcal{C}\times\mathcal{Y}$. Furthermore, assume that the system dynamics are *invariant* with respect to group operations (e.g. translations or rotations) on $\mathcal{C}$. The space $\mathcal{C}$ can be regarded as a *reduced configuration space*, in the sense that it is defined by a subset of the configuration variables of the system, with respect to which the system has certain symmetry properties. The space $\mathcal{Y}$ encodes the remaining configuration variables, as well as the vehicle's velocity and higher order derivatives of the configuration variables.

In this paper we consider problems in which the desired destination is an *equilibrium point*. We can define equilibrium points for the system (1) as the points $(\bar{x},\bar{u})$ for which $f(\bar{x},\bar{u}) = 0$. Since the system dynamics are invariant with respect to translations (and rotations) on $\mathcal{C}$, a family of equilibrium states can be expressed by a point in $\mathcal{C}\times\{\bar{y}\}$, where $\bar{y} \in \mathcal{Y}$ is a constant.

A major consequence of this restriction is the fact that the motion planning algorithm is directly applicable only to a class of autonomous vehicles which are indeed able to "stop". This includes autonomous helicopters, VTOL aircraft, spacecraft, ground vehicles, and surface/underwater vessels; in its basic configuration the algorithm may not include conventional fixed-wing aircraft.

2.2 Obstacle-Free Guidance system

The second element that is assumed to be available is a guidance algorithm in environments with no obstacles. More specifically, we assume knowledge of a guidance law that can steer the system, in the absence of obstacles, from any state to a particular target set $\mathcal{T}(x_{eq})$, centered at an equilibrium point x_{eq}.

While, admittedly, finding such a guidance law is *per se* a very difficult problem, it also has been the object of an enormous amount of work over the past century: In many cases efficient, obstacle-free guidance laws may be computed analytically.[17,18] This is the case of systems with linear dynamics and a quadratic cost. It also includes numerous cases of aerospace interest such as double or triple integrators with control amplitude and rate limits. In addition, many of these problems, although they may not admit closed-form solutions, may be solved numerically via the approximate or exact solution to an appropriate optimal control problem by minimizing the cost functional

$$J(x_0,x_{eq}) = \int_{t_0}^{t_f} g(x,x_{eq},u)dt, x(t_f) \in \mathcal{T}(x_{eq}) \tag{2}$$

for some initial conditions x_0, under the dynamics constraints (and possibly state and control constraints).

Such a formulation includes for example minimum time and minimum energy control problems. Advances in computer power, combined with appropriate plant simplifications (such as the introduction of the maneuver model outlined in[19]) make it possible in many cases of practical interest to compute and store an approximate expression for the optimal value function $J^*(x,x_{eq})$, for all $x \in \mathcal{X}$, and all equilibrium points $x_{eq} \in \mathcal{C}\times\{\bar{y}\}$, using for example iterative methods.[19,20]

If the optimal cost function $J^*(x,x_{eq})$ is known for all $x \in \mathcal{X}$, and all equilibrium points $x_{eq} \in \mathcal{C}\times\{\bar{y}\}$, then it is relatively easy to recover the optimal control policy $\pi : \mathcal{X}\times\mathcal{C} \to \mathcal{U}$, as a (feedback) policy that returns at each time instant the control input that minimizes the total (future) cost-to-go to the target.[21] The feedback policy π can be thought of as a function of the state x, parameterized by the destination equilibrium point x_{eq}. The solution to an optimal control problem in the free space thus provides us with a control policy π that ensures that the system is driven towards a target set in finite time, effectively parameterized by configurations in $\mathcal{C}$.

2.3 Environment characterization

We consider an environment in which both fixed and moving obstacles are present, and we assume that the motion of the obstacles (or conservative estimates thereof) is known in advance. Since the environment is time-varying, collisions must be checked on (space $\times$ time) pairs $(x,t) \in \mathcal{X}\times\mathbb{R}$. For this purpose, a colli-

sion checking algorithm is assumed to be available (via trajectory sampling or other appropriate method). A pair (x,t) is said to be feasible if it does not result in a collision, and x satisfies the flight envelope constraints.

2.4 Problem formulation

The motion planning problem can now be stated as follows: Given an initial state $x_0 \in \mathcal{X}$, at time t_0, and a goal *equilibrium* configuration $x_f \in \mathcal{C} \times \{\bar{y}\}$, find a control input $v : [t_0, t_f] \rightarrow \mathcal{U}$, that can steer the system the system from x_0 to $\mathcal{T}(x_f)$. A motion planning algorithm is said *complete* if it returns a feasible solution whenever there exists one and returns failure otherwise. By feasible here we mean a solution for which no collisions occur, and the flight envelope and control saturation constraints are satisfied. While the usual formulation of the motion planning problem is concerned only with finding a feasible trajectory, in many engineering applications we are also interested in finding a trajectory minimizing some cost. In this paper, we will assume the cost is defined by a functional of the form (2), that is $J(x_0, x_{eq}) = \int_{t_0}^{t_f} g(x, x_{eq}, u)dt$.

The motion planning problem, even in its simplest formulation, has been proven computationally hard. It is possible to circumvent this difficulty through the definition of a randomized motion planning algorithm which, by replacing completeness with probabilistic completeness (in the sense that the probability of the algorithm terminating correctly approaches one as the number of iterations grows) achieves computational tractability, while retaining formal guarantees on the behavior of the algorithm.

3 Path planning in the presence of obstacles

The motion planning algorithm in the presence of obstacles is based on the determination of a time-parameterized sequence of "attraction points" $x_{eq}(t)$ that effectively steers the system to the desired configuration while avoiding obstacles. In this way, the obstacle-free solution to an optimal control problem forms the basis for the problem of motion planning in the presence of obstacles. Such an approach casts the location of the equilibrium configuration as a function of time as a "pseudo-control" input for the system. Since the actual control inputs can be computed from the knowledge of the optimal control policy $\pi(\cdot, x_{eq})$, this means that the low-level control layer (the layer actually interacting with the vehicle) and the high-level, guidance layer are effectively decoupled, while at the same time ensuring full consistency between the two levels. As a consequence, unlike earlier randomized motion planning approaches, the motion planning algorithm can be run at a rate that is much slower than the rate required for the low-level control layer.

Note also that the ideas outlined above in a probabilistic roadmap setting can be seen as a motion planning technique through scheduling of Lyapunov functions, where the Lyapunov functions are defined to be the value function J^* associated with the optimal guidance law introduced earlier. While the concept is not entirely new in control theory,[22–24] to the authors' knowledge, this is the first application to motion planning in a workspace with moving obstacles. A fundamental difference can also be seen in the fact that in our algorithm the ordering of Lyapunov functions is performed on-line, whereas in the references the ordering was determined *a priori*.

The algorithm can be outlined as follows. Starting with a node representing the initial condition, a tree is built by iteratively adding new "milestones", which are connected to the tree by a feasible trajectory segment. Note that in the dynamic environment case the tree must be rooted at the initial condition, at θ seconds in the future, because of the finite computation time; θ is the amount of time allotted for each motion planning cycle . Each new milestone is computed through the generation of a random equilibrium point $x_{rand} \in \mathcal{C} \times \{\bar{y}\}$ and the associated target set $\mathcal{T}(x_{rand})$; the control policy $\pi(\cdot, x_{rand})$ is applied in turn to all the nodes of the current tree, until the ensuing propagated trajectory is feasible, and the end point satisfies certain safety conditions (discussed in the following). In this case the new milestone is added to the tree. If a feasible trajectory can not be found for any of the nodes in the current tree, the candidate milestone is discarded. The milestones generated according to the above procedure can be regarded as *primary* milestones. We assume that x_{rand} is generated from a uniform distribution on the subset of $\mathcal{C}$ defining the workspace.

The order in which the nodes in the tree are tested in conjunction with the new candidate milestone is left as yet unspecified. However, it can be convenient from a computational point of view to test first the nodes in the trees from which the candidate milestone is more likely to be reachable. This can be accomplished by sorting the nodes in the tree according to some measure of "distance" to the candidate milestone. An appropriate measure of distance in the case we are examining is given by the obstacle-free optimal cost function J^*.

The control policy to the destination $\pi(\cdot, x_f)$ is applied to the newly generated milestone: if the resulting trajectory is feasible, a complete trajectory from the initial condition (tree root) to the target set $\mathcal{T}(x_f)$ has been found. Hence the feasibility problem is solved. However, it is possible to use the remaining part of the allotted time to try to improve the current solution. For this purpose, a record of estimates of the lower and upper bound on the cost to go at each node can be maintained. Each time a new feasible trajectory is found, it is possible to climb the tree back towards the root, along the feasible trajectory, updating the estimates on the upper bound on the cost-to-go (see below).

Notice that, even though the system is moving among equilibrium points, the resulting trajectories will not, in general, be composed of straight lines; moreover, the

trajectories cannot, in general, be reversed in time. In this sense the resulting roadmap is different from the roadmap constructed in the original PRM algorithm.

To our knowledge, the algorithm outlined above is the first one to which the convergence result in[16] fully applies. In this sense the most significant contribution of this paper is to propose to shift the search for reachable milestones from an *open-loop* process, where by exploration is done by randomly sampling the controls available to the system to a *closed-loop* process, whereby the exploration is done by randomly (and uniformly) sampling the milestones, and the obstacle-free guidance system then chooses the controls leading the vehicle from its current state to that milestone. In this sense the guidance law can in fact be interpreted as a practical implementation of an ideal inversion mechanism proposed in.[16]

3.1 Improving performance

As it can be easily recognized, the algorithm outlined above consists of jumps from equilibrium point to equilibrium point, and as such is unlikely to provide satisfactory performance, in terms of the cost (2).

However, performance may be restored by realizing that the available guidance policy may not only steer the vehicle from equilibrium state to equilibrium state, but from *any* state to an equilibrium state. This suggests introducing the following step: consider the tree at some point in time and a newly added milestone to the tree. A *secondary milestone* is defined to be any state of the system (continuous or hybrid) along the path leading from the parent node in the tree to the newly added milestone. Pick $n \geq 1$ such secondary milestones at random along that path. Because the vehicle is in motion along the path, these secondary milestones are likely to be at points in the state space that are "far" from the equilibrium manifold.

These secondary milestones are added to the tree, and, as for all newly generated milestones, feasibility is checked for the resulting trajectory to the destination x_f. Moreover secondary milestones can be selected as the tree node to be expanded in later iterations. Note that all secondary milestones, by construction, have a primary milestone in a child subtree.

3.2 Data structure

The roadmap is constructed as a tree, consisting of nodes and edges. At the tree nodes all the information concerning each milestone is stored, including: 1) The propagated state of the vehicle (i.e. state $x \in \mathcal{X}$, time $t \in \mathbb{R}$). 2) The cumulative cost and upper and lower bounds on the cost-to-go. The lower bound on the cost-to-go coincides with the value of the cost function $J^*(x, x_f)$, that is, it corresponds to the cost to go to the target state assuming the presence of no obstacle. The upper bound on the cost-to-go is initialized to $+\infty$, meaning that a feasible path from the particular node has not been found yet. 3) A counter of the total number of milestones in the children trees. The (state×time) couple is initialized through propagation of the system dynamics, and the cumulative cost is updated, according to eq.(2).

At the tree edges, the following data are stored: 1) Information regarding the transitions between single milestones (namely, the parameters identifying the control law implemented in the transition, or the target equilibrium point); 2) The incremental cost incurred during the transition, mainly for bookkeeping purposes; again, this is done using the cost functional expressed in eq.(2).

3.3 Real-time considerations

A significant issue arising from the usage of randomized algorithms for path planning is the distinct possibility of driving the system towards a dead-end due to finite computation times. The notion of τ-safety is introduced to prevent such situations to develop:

Definition 1 (τ-safety) *A milestone* $(x, t) \in \mathcal{C} \times \{\bar{y}\} \times \mathbb{R}$ *is said to be* τ-safe *if* $(x, \tilde{t})$ *is feasible for all* $\tilde{t} \in [t, t + \tau]$.

Primary milestones are added to the tree *only if* τ-safe. If possible, and practical, primary milestones can be checked for the absence of collisions over an infinite horizon ($\tau \to \infty$). This is possible in many cases of interest (including all static environments), and results in hard safety guarantees on the resulting motion plan, if the initial condition is itself safe. In cases in which safety cannot be ensured over an infinite time horizon, τ-safety only ensures that the algorithm will always have at least τ seconds to compute a new solution.

The τ-safety of the generated plan derives from the fact that all the primary milestones are by construction τ-safe, and all secondary milestones have at least one primary milestones in their sub-tree. Maintaining safety guarantees in the face of finite computation times is particularly important, since the algorithm itself has no deterministic guarantees of success. In the sense outlined above the algorithm will always produce safe motion plans, even in the case in which a feasible trajectory to the target set has not been found.

The time available for computation is bounded by either θ, or by the duration of the current trajectory segment. When the time is up, a new tree must be selected from the children of the current root. If there are none, since every primary milestone is τ-safe, the system has at least τ seconds of guaranteed safety, available for computing a new tree (secondary milestones *always* have at least one child). If the current root has children, then two cases arise: At least one of the children leads to the destination through an already computed feasible solution. If there is more than one such feasible solution, the solution with the least upper bound on the cost-to-go is chosen; No feasible solutions have been computed yet. In this case there is no clear

indication of the best child to explore. Maintaining the same approach at the basis of the algorithm, the child to descend can be selected randomly, according either to a uniform distribution, or to a distribution weighted on the total number of primary milestones in the sub-children of each tree. In the latter case the selected tree is likely to cover a bigger portion of the reachable set.

3.4 Tree pruning

The upper and lower bounds on the cost to go stored for each tree milestone may be profitably used for pruning the tree and speeding up computations. Recall the lower bound coincides with the optimal cost-to-go in the obstacle-free case, and the upper bound is equal to the cost of the best trajectory from the milestone to the destination x_f if this trajectory has been found or $+\infty$ otherwise.

Every time a new feasible solution is found, the upper bounds on the cost-to-go may be updated by climbing the tree backwards along that feasible solution towards the tree root. While performing this operation, it is also possible to look at all the children of the node being updated. If the lower bound on the total cost-to-go for such a children (plus the cost of the corresponding edge) is higher than the upper bound on the cost-to-go for the current node, the corresponding subtree can be safely removed, as it cannot possibly provide a better solution than the one which has just been found.

The end result of such a process is the removal from the trajectory tree of all the provably bad candidates for the "optimal" solution. The trajectory tree, following this pruning process, contain a smaller number of nodes, thus improving the overall computational efficiency. However, it must be kept in mind that tree pruning can only be carried out once a feasible solution has been found, and is of no help before that happens.

4 Application Examples

In the next sections we present three examples which show the power and the flexibility of the proposed algorithm. We consider first a linear system, then a system endowed with a control architecture patterned after the hybrid control structure introduced in.[19] All algorithms have been implemented in C++ on a Pentium II 300 MHz machine, using the LEDA[25] library. The statistical data are referred to a data set of one thousand simulation runs for each example. Computer animations of the simulation examples are available from the authors.

4.1 Ground robot

In this section, we are interested in planning the path of a planar system with equations of motion

$$\begin{aligned} \ddot{x}_1 + \dot{x}_1 &= u_1 \\ \ddot{x}_2 + \dot{x}_2 &= u_2 \end{aligned} \tag{3}$$

The magnitude of each control u_1 and u_2 is assumed to be bounded by u_{max}. Although this system model is quite simple, it is a good representation of the ground robots used by the Cornell University team to win the RoboCup-2000 contest.[26]

For any one axis, let the initial position and velocity be x_0 and v_0; the final (equilibrium) conditions are characterized by a desired final position x_f and zero velocity. In order to provide The minimum time maneuver from origin to destination for any of the degrees of freedom (assuming a general maximum control intensity $u_{\max}$) is a bang-bang control law[18] given by

$$\begin{aligned} u(t) = U \quad &for \quad 0 < t < t_1 \\ u(t) = -U \quad &for \quad t_1 < t < t_1 + t_2 \end{aligned} \tag{4}$$

The sign of the initial control value U can be determined through the switching function:

$$\Delta_0 := \begin{cases} x_0 - x_f + v_0 - u_{max} \log(1 + v_0/u_{max}) & \text{for } v_0 \geq 0 \\ x_0 - x_f + v_0 + u_{max} \log(1 - v_0/u_{max}) & \text{for } v_0 < 0 \end{cases} \tag{5}$$

If the initial conditions are such that $\Delta_0 \geq 0$ then $U = -u_{max}$, and $U = u_{max}$ otherwise.

The time length of the two bang-bang segments can be determined as follows:

$$\begin{aligned} t_1 &= t_2 - C/U \\ t_2 &= \log(1 + \sqrt{1 - \exp(C/U)(1 - v_0/U)}) \end{aligned} \tag{6}$$

with $C = x_0 + v_0 - x_f$.

The policy π used to control the vehicle described by (3) is then defined as follows: Considering the two degrees of freedom x_1 and x_2, the "slowest" axis is determined first, and the corresponding time optimal control is applied. Let t^*_{min} be the minimum time corresponding to that axis. The other, fastest axis is then controlled using a minimum "effort" solution, by solving the minimum time problem using the equations (4), with $U = \pm\gamma u_{\max}$, and by iterating over the parameter $\gamma \in (0, 1)$ until $t_{min} = t^*_{min}$.

The randomized path planning has been tested in several examples, including cases with both fixed and moving obstacles, and in general proved to be very fast and reliable. The first example involves navigating the ground robot through a set of obstacles represented as spheres in the configuration space, as shown in Fig. 1 . This example was very easily handled by the proposed planner: a feasible solution was found on average in 14 ms (standard deviation 14 ms).

In the second example the robot must go through moving openings in two walls: This example can be particularly difficult for randomized planners, since the environment is characterized by the presence of "narrow passages" (see Fig. 2). However, the algorithm proposed in the paper was able to deal with this scenario quite effectively. A feasible solution was found on average in 61.7 ms, (standard deviation 96.5 ms). Moreover, notice that in about 75% of the simulation runs, the first feasible solution was found in under 100 ms.

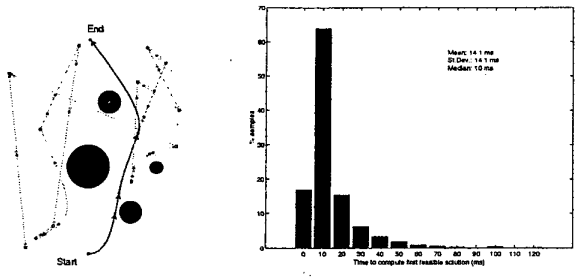

Figure 1: Ground robot moving among fixed spheres

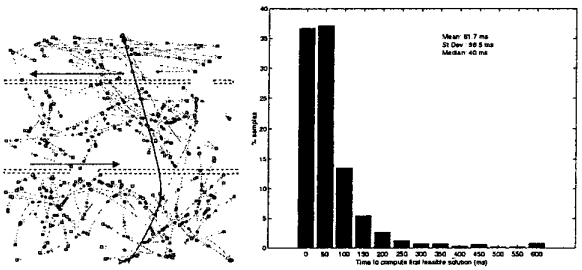

Figure 2: Ground robot through moving doors

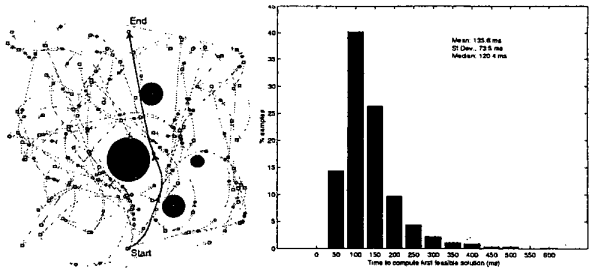

Figure 3: Helicopter flying among fixed spheres

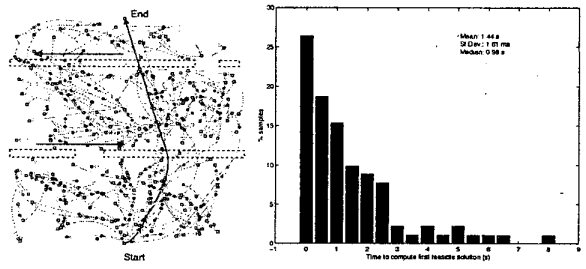

Figure 4: Helicopter flying through moving doors

4.2 Small autonomous helicopter

One of the prime motivations for the development of the algorithm presented in this paper is the path planning task for a small autonomous helicopter.

This section presents simulation results for a test case involving a small autonomous helicopter. The simulations rely upon a fully non-linear helicopter simulation, based on a commonly used minimum-complexity model.[27] The motion planning algorithms operating on the hybrid automaton structure[19,28] are complemented a nonlinear control law[29] to ensure tracking of the reference trajectory.

The planner was tested using the same scenarios as for the ground robot examples. The output of the simulations was scaled, in such a way as to provide a meaningful comparison of the two cases.

The cost function used in all the examples is the total time needed to go to the destination. The first example involves navigating the helicopter through a set of obstacles represented as spheres in the configuration space (Fig. 3) . This example was very easily handled by the proposed planner: a feasible solution was found on average in 136 ms (standard deviation 73.5 ms).

In the second example the helicopter must go through moving openings in two walls (see Fig. 4) . The basic algorithm, was not able to handle this case very effectively.[30] In most runs, it took upwards of 20 seconds to compute a feasible trajectory. However, the extensions presented in this paper resulted in a significant reduction in the required computation time: tests performed indicate the average time required to compute a feasible solution to be only 1.44 seconds (standard deviation 1.61 seconds).

5 Conclusions

In this paper a randomized motion planning algorithm was presented, based on using an obstacle-free guidance systems as local planners in a probabilistic roadmap framework. The main advantage of the algorithm is the capability to address in an efficient and natural fashion the dynamics of the system, while at the same time providing a consistent decoupling between the motion planning and the low-level control tasks.

From a theoretical point of view, it was shown how to perform uniform sampling in the reachable space of the vehicle, as opposed to sampling in the input space. Real-time issues were directly addressed: in the case in which finite computation time and available resources do not allow the computation of a feasible solution before a decision has to be made, it was shown how to ensure safety and how to choose likely candidates for further exploration. Future work address motion planning in uncertain environment, with limited sensor range, and multi-vehicle operations.

References

[1] J.-C. Latombe. Motion planning: a journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics. Research*, 18(11), November 1999.

[2] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[3] Z. Li and J.F. Canny, editors. *Nonholonomic Mo-*

tion Planning. Kluwer Academic Publishers, Boston, MA, 1993.

[4] J.-P. Laumond, editor. *Robot Motion Planning and Control*, volume 229 of *Lectures Notes in Control and Information Sciences.* Springer, 1998. Available at `http://www.laas.fr/ jpl/book.html` at the time of writing.

[5] J.H. Reif. Complexity of the mover's problem and generalizations. In *FOCS*, pages 421–427, 1979.

[6] J. Canny. *The Complexity of Robot Motion Planning: ACM Doctoral Dissertation Award.* MIT Press, 1988.

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

[8] M. H Overmars and P. Svestka. A paradigm for probabilistic path planning. Technical report, Department of Computer Science, Utrecht University, March 1996.

[9] L. E. Kavraki, M. N Kolountzakis, and J.C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 3020–3025, 1996.

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

[11] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the 1998 Workshop on Algorithmic Foundations of Robotics*, Houston, TX, March 1998.

[12] 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.

[13] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Iowa State University, Ames, IA, Oct. 1998.

[14] S.M. LaValle and J.J Kuffner. Randomized kinodynamic planning. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, 1999.

[15] J.J. Kuffner and S.M. LaValle. RRT-Connect: an efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, 2000.

[16] D. Hsu, J.C. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proc. Workshop on Algorithmic Foundations of Robotics (WAFR'00)*, Hanover, NH, March 2000.

[17] M. Athans and P. Falb. *Optimal Control.* McGraw-Hill, 1966.

[18] A. E. Bryson and Y. C. Ho. *Applied Optimal Control.* Hemisphere Publishing, New York, 1975.

[19] E. Frazzoli, M.A. Dahleh, and E. Feron. A hybrid control architecture for aggressive maneuvering of autonomous helicopters. In *IEEE Conf. on Decision and Control*, December 1999.

[20] D. P. Bertsekas and J.T. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[21] D.P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA, 1995.

[22] A. Leonessa, V.S. Chellaboina, and W.M. Haddad. Globally stabilizing controllers for multi-mode axial flow compressors via equilibria-dependent lyapunov functions. In *Proc. IEEE Conf. Dec. Contr.*, San Diego, CA, December 1997.

[23] R. R. Burridge, A. A. Rizzi, and D.E. Koditscheck. Sequential decomposition of dynamically dexterous robot behaviors. *International Journal of Robotics Research,*, 18(6):534–555, June 1999.

[24] M.W. McConley, B.D. Appleby, M.A. Dahleh, and E.Feron. A computationally efficient Lyapunov-based scheduling procedure for control of nonlinear systems with stability guarantees. *IEEE Transactions on Automatic Control*, January 2000.

[25] K. Melhorn and St. Näher. *The LEDA platform of Combinatorial and Geometric Computing.* Cambridge University Press, 1999.

[26] R. D'Andrea, T. Kalmár-Nagy, P. Ganguly, and M. Babish. The Cornell robot soccer team. In P. Stone, editor, *RoboCup-00: Robot Soccer World Cup IV*, Lecture Notes in Computer Science. Springer, 2001.

[27] T.J. Koo and S. Sastry. Output tracking control design of a helicopter model based on approximate linearization. In *Proc. IEEE Conf. on Decision and Control*, December 1998.

[28] E. Frazzoli, M.A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *IEEE Conf. on Decision and Control*, Sydney, Australia, 2000.

[29] E. Frazzoli, M.A. Dahleh, and E. Feron. Trajectory tracking control design for autonomous helicopters using a backstepping algorithm. In *American Control Conference*, Chicago, IL, 2000.

[30] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. In *AIAA Conf. on Guidance, Navigation and Control*, Denver, CO, August 2000.