AIAA 2000-4056

REAL-TIME MOTION PLANNING FOR AGILE AUTONOMOUS VEHICLES

Emilio Frazzoli Munther A. Dahleh Eric Feron

AIAA Guidance, Navigation, and Control Conference and Exhibit

August 14–17, 2000/Denver, CO

AIAA 2000–4056

REAL-TIME MOTION PLANNING FOR AGILE AUTONOMOUS VEHICLES

Emilio Frazzoli *
Munther A. Dahleh †
Eric Feron ‡

The operation of an autonomous vehicle in an unknown, dynamic environment is a very complex problem, especially when the vehicle is required to use its full maneuvering capabilities, and to react in real time to changes in the operational environment.

A new class of algorithms, based on the construction of probabilistic roadmaps, has been recently introduced, and proven to provide a very fast and efficient scheme for motion planning for robots with many degrees of freedom, while maintaining completeness guarantees (in a probabilistic sense). In this paper we will present an extension of the probabilistic roadmap approach, which is able to deal effectively with the system dynamics, in an environment characterized by moving obstacles. This is accomplished through a Lyapunov function based approach to the construction of the roadmap.

The proposed algorithm can be directly applied to a very general class of dynamical systems, including traditional state space systems, as well as hybrid systems (systems including both discrete and continuous dynamics). Simulation examples, involving a small autonomous helicopter, will be presented and discussed.

Introduction

RECENT advances in computational capabilities, both in terms of hardware and algorithms, communication architectures, and sensing and navigation devices make it possible to develop research efforts aimed at designing autonomous, single or multi-agent systems that exhibit a high degree of reliability in their operation, in the face of a dynamic and uncertain environment, operating conditions, and goals. These systems must be able to construct a proper representation of the environment and of their own conditions from the available sensory data and/or knowledge base, and have to be able to make timely decisions aiming at interacting with the environment in an "optimal" way.

One of the basic problems which have to be faced by autonomous vehicles or moving robots is the generation and the execution of a motion plan, that enables the robot to move to some place to perform a given task, while avoiding collisions with obstacles, or other undesired behaviors. When dealing with realistic situations, the robot will have to generate and execute this plan in an environment with a complex topology, and with dynamically changing and uncertain components. In extreme cases, the environment could include other agents actively seeking out the robot with hostile intentions (e.g. autonomous air vehicles in military operations).

Moreover, often we are interested in devising motion planning strategies for agile autonomous vehicles, operating in an environment such that the exploitation of their dynamics and maneuvering capabilities plays a crucial role in achieving the mission goals.

The problem of navigating a simple kinematic robot in a known environment with polyhedral obstacles has been proven to be computationally hard.[1,2] Even though complete algorithms are available, these cannot be used for real-time path planning in many real-world applications.[1] Note also that these algorithms require an explicit characterization of the environment (i.e. of the obstacles), which is practically unfeasible for large configuration spaces.

When the dynamics of the vehicle are also considered, there is strong evidence that the computational complexity of a complete algorithm will grow exponentially fast in the number of dimensions of the *state space*. Motion planning in the presence of dynamics is usually referred to as kinodynamic planning, and has been the object of considerable interest in the recent past. The system is subject to kinematic or dynamic constraints that, unlike obstacles, cannot be encoded as "forbidden" zones in the state space. Moreover, if the environment is changing over time, the output

*Research Assistant, Laboratory for Information and Decision Systems, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, email:frazzoli@mit.edu. AIAA Member

†Professor, Laboratory for Information and Decision Systems, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, email: dahleh@lids.mit.edu

‡Associate Professor, Laboratory for Information and Decision Systems, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, email: feron@mit.edu. AIAA Senior Member

of the motion planning algorithm must be a time-parametrized feasible trajectory.

A new class of motion planning algorithms, known as probabilistic roadmap planners, has been recently introduced in ([3]), and further explored in ([4–12]). This class of algorithms is particularly interesting since, by relaxing the completeness requirements to probabilistic completeness (i.e. completeness in expectation), we can achieve computational tractability, while maintaining formal guarantees on the behavior of the algorithm.

Note that another significant advantage of a randomized sampling scheme is that no explicit characterization of the environment is needed. Sampling replaces the prohibitive computation of an explicit representation of the free space by collision-checking operations.

In this paper, an extension of the available algorithms is presented, which is able to deal effectively with the system's dynamics, in an environment characterized by moving obstacles. Moreover, it is shown how this class of algorithms is particularly well suited to application to a new class of integrated guidance and control architectures, based on a hybrid control formulation.

The paper is organized as follows: first, the planning framework is presented, including specifications on the system dynamics, on the environment, and on the type of problem we want to address. A short review of previous work in probabilistic roadmap algorithms will follow, after which we will present, discuss, and analyze our extensions for real-time motion planning of agile vehicles. Finally, we will present and discuss some simulation results.

Planning framework

System dynamics

In the following we will present different modeling approaches for describing the system dynamics. While the continuous state-space formulation is the most traditional and widely used representation of a vehicle's dynamics, recent advances in hybrid systems suggest an alternative formulation, which presents significant computational advantages, especially for aerospace vehicles with complex, high-dimensional and fast dynamics.

State space formulation

The usual representation of the dynamics of an autonomous vehicle or robot is based on a state-space formulation. The dynamics of the system are described by an ordinary differential equation of the form:

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

where $x \in \mathcal{X}$ is the state, belonging to an n-dimensional manifold $\mathcal{X}$ (the *state space*), and u represents the control input signals, taking values in the set $\mathcal{U} \subseteq \mathbb{R}^m$. The function $f : \mathcal{X} \times \mathcal{U} \to T\mathcal{X}$ is assumed to be locally Lipschitz in its arguments. Note that the above formulation can represent both nonholonomic and dynamic constraints.[13] In some cases, we also have to consider additional inequality constraints on the state variables $F_i(x) < 0$, to ensure safe operation of the vehicle (e.g. flight envelope protection).

Assume that the state space $\mathcal{X}$ can be decomposed, at least locally, in the product $\mathcal{C} \times \mathcal{V}$. We can regard $\mathcal{C}$ as the *configuration space* of the vehicle, while $\mathcal{V}$ encodes the vehicle's velocity and higher order derivatives of the configuration variables.

A significant reduction in the complexity of the motion planning problem can be achieved under the assumption that the system dynamics are *invariant* with respect to translations (or, in general, group operations) on $\mathcal{C}$. One of the main consequence of the invariance (or symmetry) is the existence of relative equilibria, or trim trajectories.[14–17] While the motion planning problem can be posed in terms of relative equilibria, for the sake of simplicity, in this paper we will limit the analysis to 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 on $\mathcal{C}$, a family of equilibrium states can be expressed by a point in $\mathcal{C} \times \{\bar{v}\}$, where $\bar{v} \in \mathcal{V}$ is a constant (e.g. the zero vector).

Hybrid formulation

While the state space formulation is probably the most commonly used representation of a vehicle's dynamics, it could not be the most appropriate choice in all cases. In particular, the state space of non-trivial systems is typically very large, and the "curse of dimensionality" makes the solution of motion planning problems in such large-dimension spaces computationally intractable.

An alternative approach is represented by the formulation of the system dynamics, and hence of the motion planning problem, in what can be regarded as the *maneuver space* of the vehicle.[18,19] Such an approach entails the definition of a hybrid control architecture (a *hybrid automaton*), based on quantization of the system dynamics into a library of feasible "trajectory primitives", or behaviors. We will not discuss in detail the architecture proposed in the references, it will suffice to mention its main features.

First of all, we have to identify the trajectory primitives that will be included in the library. As mentioned in the previous section, from invariance to translation in $\mathcal{C}$, it follows that there exists a particular class of trajectories, known as relative equilibria. Most vehicles exhibit invariance to translation in the horizontal plane, and to rotation about a vertical axis. If altitude changes are limited (and hence air density variations

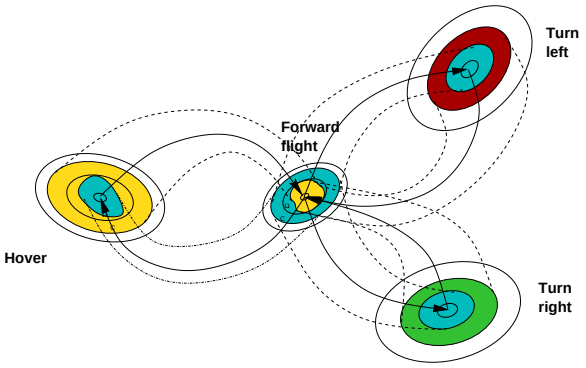


Fig. 1 Simplified Robust Hybrid Automaton

are small), we can also assume invariance to vertical translation; this is the case when we consider, for example, Nap-Of-the-Earth (NOE) missions. Under the above assumptions, the relative equilibria, or trim trajectories, will be represented by helices, with a vertical axis, in the physical space. An appropriately chosen set of relative equilibria constitutes the first class of primitives. A possible such choice can be obtained through a gridding of the compact set defined by the flight envelope. A second class of primitives, that we can call "maneuvers" are defined as feasible, finite time transitions between trim trajectories; in particular, maneuvers can be designed in order to solve some local optimal control problem, or other criteria (designer's insight into what are the desirable maneuvers that the vehicle should be able to perform).

We can graphically depict the hybrid automaton as a directed graph, where the nodes represent the trim trajectories, and the edges represent the maneuvers. If we restrict the evolution of the vehicle to follow trajectories generated by the automaton, the complete state will be described by the index of the current trajectory primitive, the current time, and the time and location on $\mathcal{C}$ at the primitive inception time. Explicit conditions can be given to ensure reachability of the resulting dynamical system, as well as consistency or robustness to uncertainties in the environment and in the plant.[19]

Note that, while maneuvers have a definite time duration, trim trajectories can be followed indefinitely. As a consequence, the decision variables, or control inputs, acting on the hybrid automaton, consist of the amount of time the vehicle must remain in the trim trajectory (*coasting time*), and in which maneuver it has to execute next (*jump destination*).

The hybrid automaton can be seen as a new *modeling tool*, in which the continuous dynamics ODE (1) is replaced by the transition rules on the directed graph representing the automaton, and by the associated hybrid system evolution.[18,19] Using this representation, the assumption of invariance to translations in $\mathcal{C}$ is enough to ensure that the hybrid automaton encodes all the relevant information about dynamics and the flight envelope constraints of *any* vehicle in a space of small dimension, that is $\mathcal{M} = \mathcal{C} \times Q_T$. For most vehicles, the dimension of $\mathcal{C}$ is at most 4, and Q_T is a finite set of trim trajectory indices. The maneuver space $\mathcal{M}$ can be equivalently seen as the union of $|Q_T|$ copies of the configuration space $\mathcal{C}$.

The price that we have to pay for using the hybrid automaton is the sub-optimality of the computed solutions, owing to the fact that the stored trajectory primitives do not represent the whole dynamics of the system. However, the number of trajectory primitives stored in the automaton can be increased, depending on the available memory and computational resources, so the sub-optimality gap can be reduced to a point were it is not noticeable for practical purposes. Moreover, for most practical purposes a sub-optimal solution which is computable on-line can be worth more that an optimal solution that requires computational resources only available for off-line planning.

In the rest of the paper, to simplify the notation, we will just use the letter $\mathcal{X}$ to indicate the "state space", with the understanding that this can be regarded as either the proper, continuous state space or the hybrid maneuver space $\mathcal{M}$.

Environment characterization

We will consider an environment which presents both fixed and moving obstacles, and we will assume that the motion of the obstacles (or conservative estimates thereof) is known in advance. Since the environment is time-varying, collisions have to be checked on (space $\times$ time) pairs $(x,t) \in \mathcal{X} \times \mathbb{R}$. We will define the *feasible set* $\mathcal{F} \subset \mathcal{X} \times \mathbb{R}$ as the set of all (x,t) pairs for which there are no collisions, and the flight envelope constraints are satisfied.

Given an initial condition (x_0, t_0), a pair (x_f, t_f) is said *reachable* if it is possible to find a control function $\hat{u} : [t_0, t_f] \to \mathcal{U}$, such that the ensuing trajectory of the system, from the above initial conditions is feasible, and terminates at the required final condition. In other words, we say that (x_f, t_f) is reachable from (x_0, t_0) if the time-parametrized curve $\chi : [t_0, t_f] \to \mathcal{X}$ is an integral curve of the ODE (1) (or is compatible with the automaton dynamics), given the control input $\hat{u}(t)$, and is such that $\chi(t_0) = x_0$, $\chi(t_f) = x_f$, and $(\chi(t), t) \in \mathcal{F}$, for all $t \in [t_0, t_f]$. We can define the *reachable set* $\mathcal{R}(x_0, t_0) \subset \mathcal{F}$ as the set of all points that are reachable from (x_0, t_0). Accordingly, given a set $S \subset \mathcal{F}$, we define:

$$\mathcal{R}(S) = \bigcup_{(x,t) \in S} \mathcal{R}(x,t) \tag{2}$$

Problem formulation

The motion planning problem we are interested in can now be stated as: given an initial state $x_0 \in \mathcal{X}$, at time t_0, and a goal *equilibrium* configuration $x_f \in \mathcal{C} \times \{\bar{v}\}$, find a control input $v : [t_0, t_f] \to \mathcal{U}$, such

that the resulting trajectory χ is feasible, and satisfies the boundary conditions $\chi(t_0) = x_0$, $\chi(t_f) = x_f$. A motion planning algorithm is said *complete* if it returns a feasible solution if there exists a time t_f such that $(x_f, t_f) \in \mathcal{R}(x_0, t_0)$, and returns failure otherwise. 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, often expressed as a functional of the form $J := \int_{t_0}^{t_f} g(\chi(t), \upsilon(t))dt + , (x(t_f), t_f)$. The motion planning problem, even in its simplest formulation has been proven computationally hard. However, very recently a new class of algorithms for motion planning has been introduced, which, by trading off completeness with probabilistic completeness (in the sense that the algorithm terminates correctly with *high probability*) achieves computational tractability, while retaining formal guarantees on the behavior of the algorithm.

Review of previous work

Since the algorithm we will present builds directly on previous algorithms available in the literature, a brief discussion of previous work will be helpful in understanding its characteristics and novel components.

Probabilistic Roadmap

The Probabilistic RoadMap (PRM) planner was first introduced as a fast and efficient algorithm for geometric, multiple-query path planning.[3] The algorithm can be divided into an off-line processing phase and an on-line querying phase. The off-line phase consists of the construction of the roadmap, which is accomplished by (1) picking N_{PRM} configurations (*milestones*) uniformly at random in the free space $\mathcal{F}$; (2) building the visibility graph R of milestones that "see" each other. We say that a milestone m_1 is visible by another milestone m_2 if a straight line can be drawn from m_1 to m_2 inside $\mathcal{F}$, that is without colliding with obstacles. The visibility graph R defines our *roadmap*. Once the roadmap has been built, we store it in the robot's memory. The on-line planning will then consist of a query phase, in which we try to find a path connecting the assigned start and end configurations, through the stored roadmap, plus short linking segment from the starting and ending configuration to milestones in the roadmap, achieved through some form of local planner.

The PRM algorithm has been proven to be complete in a probabilistic sense, i.e. the probability of a correct termination approaches unity as the number of milestones increases. Moreover, performance bounds have been derived as a function of certain characteristics of the environment (*expansiveness*) which prove that the probability of non-correct termination decreases exponentially fast in the number of milestones.[20]

In its original form the algorithm is only applicable to path planning problems for holonomic robots (no dynamics, no kinematic constraints). Also, the PRM algorithm was originally designed for multiple-query motion planning: this means that the foreseen applications would involve the generation of a fixed knowledge base, from which feasible paths could be found easily for several queries. In this case, we process all the available information during the roadmap construction, and do very little on-line computation. Finally, the basic PRM algorithm is only concerned with checking feasibility of a given start-end configuration couple; no performance issues are addressed.

Lazy PRM

In the lazy PRM algorithm, both the roadmap construction and the query phase are carried out on-line.[21] A number $N_{lazy} << N_{PRM}$ of milestones is generated, but their visibility is not checked: on the contrary, a full connectivity graph is generated (possibly limiting the connectivity to nodes less than some distance d apart). At each time step the algorithm finds the shortest path from the current roadmap, and check all the nodes and edges belonging to this path for obstacles. If collisions are detected, the unfeasible nodes/edges are removed from the roadmap, and the search is performed again on the modified roadmap. This process is repeated until a feasible path is found, or it has been established that no paths exist in the current roadmap. If this is the case, other additional N_{lazy} points are added to the roadmap, and the process is repeated again. In order to improve efficiency, nodes/edges that have already been checked and found feasible can be marked to avoid further checking.

The properties of the Lazy PRM, with respect to completeness and performance bounds are the same as in the original PRM case: the main concerns of the authors was to limit the number of collision checks. However, note that at the basis of the Lazy PRM is also a notion of *performance* (in the search for a shortest path), that was lacking in the basic PRM.

Rapidly-exploring Random Trees

Rapidly exploring Random Trees constitute an example of single-query probabilistic roadmap algorithm: in this class of algorithms, the roadmap is not constructed off-line, but during the on-line exploration phase.[10,11] The RRT algorithm starts with the initial configuration as the root of a configuration tree. At each iteration, we generate a random sample configuration q_i in the free space $\mathcal{F}$, and select the node m_i that is "closest" to q_i. From m_i, use a greedy control algorithm in order to go "towards" q_i for a small time δt; call the endpoint of this exploration step m_i'. If we do not have collisions, we add m_i' to the tree, and iterate until one node of the tree is sufficiently close to the target. Another option would be to also grow a tree backwards in time from the target, and stop when

some nodes in the two trees are sufficiently close.

The RRT algorithm has been proven probabilistically complete.[12] A similar algorithm was introduced very recently:[9] in this alternative formulation, the milestone from which a new leaf is generated is chosen randomly (from a weighted distribution, aiming at avoiding too many leaves from the same milestone). The new milestone is generated by generating a random piecewise-constant in time control, and propagating the parent milestone according to the system dynamics. The authors were able to prove, in addition to probabilistic completeness, performance bounds for the algorithm.

Notice that RRTs and similar algorithms represent the only class of probabilistic roadmap algorithm that can be immediately extended to allow for moving obstacles, since the trees (directed graphs by definition) are rooted at the initial time and configuration, and hence time can be simply propagated to each node in the tree. The RRT algorithm does not involve implicitly performance criteria, but these are easily integrated in the framework.[9]

Sampling point generation

In general, the proofs on the properties of probabilistic planners rely on uniform distributions of sampling points. The asymptotic results (i.e. completeness) can be extended to all distributions for which the probability density function is strictly positive (almost) everywhere on the configuration space $\mathcal{C}$.[12]

However, many sampling selection schemes have been published, in the hope to enhance the algorithm's performance with some heuristics, especially in the presence of narrow passages (i.e. environments with small "expansiveness").

Some sampling techniques available in the literature include: (1) allow for some degree of penetration of the sampling point into obstacle, up to some depth;[7] (2) if a milestone is inside an obstacle, shoot a number of rays in random directions to the boundary if the obstacle, and use these configurations as milestones[22] (3) generate couples of "close" milestones, and keep one of them if the other is inside an obstacle, otherwise discard both;[23] (4) generate low-discrepancy milestones.[24]

Notice that the intuition behind the first three techniques above relies on the minimum principle of Pontryagin: a "good" solution will be very close to the obstacles (i.e. the constraints will be active). Low-discrepancy roadmaps try to avoid clustering of random points, but instead ensure uniform exploration of the configuration space through the use of pseudo-random sequences with certain desirable characteristics.

Planning in an obstacle-free environment

Before addressing the problem of motion planning in the presence of obstacles, we will discuss the solution to the problem of optimal motion planning in an obstacle-free environment. In the following, we will try to make it clear how the solution to this problem will be instrumental in devising (sub)optimal trajectories when obstacles are included.

We will consider the problem of finding the control input that, given a target *equilibrium point* x_{eq}, minimizes the total cost:

$$J(x_0, x_{eq}) = \int_{t_0}^{t_f} g(x, x_{eq}, u)dt + , \ (x(t_f), x_{eq}, t_f) \quad (3)$$

for some initial conditions x_0, under the dynamics constraints (and possibly state and control constraints). This is a special case of problems that constitute the subject of optimal control theory.[25,26]

If we know the optimal cost function $J^*(x, x_{eq})$, for all $x \in \mathcal{X}$, and all equilibrium points $x_{eq} \in \mathcal{C} \times \{\bar{v}\}$, then it is relatively easy to formulate an 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. Alternatively, given a stabilizing control law, in some cases it is possible to find out a corresponding optimal cost function (inverse optimal control problem).

In general, the computation of the optimal cost function, and of the optimal control is a very difficult problem. However, in some cases an explicit form of the solution can be easily computed: most notably, in the case of linear systems subject to a quadratic cost.

When this is not possible, other techniques can be used to compute an estimate of the optimal controller. In particular, the hybrid formulation outlined in the previous sections can be profitably used for autonomous vehicles: using a dynamic programming formulation, an approximation of the optimal cost function (an upper bound) can be computed "easily" using a numerical procedure, even for vehicles with complex dynamics, including all state and control constraints.[18,27] Other approaches for kinematic motion planning of nonholonomic vehicles involve the construction of optimal solutions via the interconnection of canonical paths.[28–30]

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 an equilibrium point, effectively parameterized by configurations in $\mathcal{C}$. Additionally, we notice that, once a cost function has been computed for all states, it can be used as a meaningful measure of the "distance" from points in $\mathcal{X}$ and equilibrium points at arbitrary locations on $\mathcal{C}$ (under the invariance hypothesis). Moreover, the same cost function induces a metric on the configuration space $\mathcal{C}$. Notice that the time to reach the desired

equilibrium point given a stabilizing policy π is not always finite (unlike the cost); in this case, we will consider that the system has reached the equilibrium point when the state enters a ball of radius ϵ centered at the equilibrium.

Motion planning algorithm

The motion planning algorithm in the presence of obstacles that we will present in the following will be based on the determination of a time-parametrized 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 will form 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.

This has an advantage over other approaches, where a random piecewise-constant in time control input is generated.[9] Random control inputs could result in violation of the vehicle's operating envelope, or other undesirable behaviors for non-trivial dynamic systems; on the other hand, using a Lyapunov function approach, we ensure that the closed loop system remains "stable" at all times. Also, greedy approaches on Euclidean distance in the configuration space[10] can lead to instability for even very simple dynamic systems.

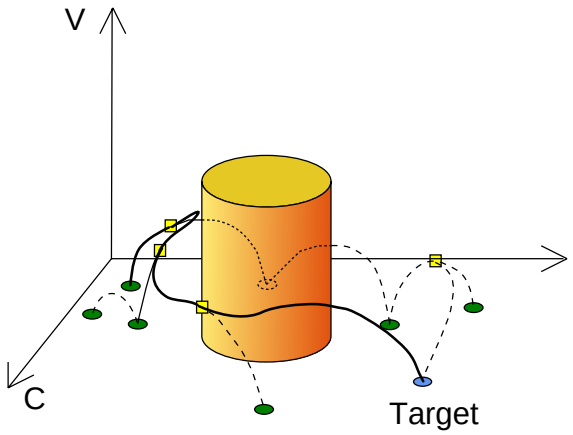
Note also that the ideas outlined above in a probabilistic roadmap setting can be seen as a motion planning technique through scheduling of Lyapunov functions. While the concept is not entirely new in control theory (see for example[31–33]), to the authors' knowledge, ours is the first application to motion planning in a workspace with moving obstacles. A fundamental difference can also be seen in the fact that in our implementation the ordering of Lyapunov functions is performed on-line, whereas in the references the ordering was pre-determined.

The algorithm can be outlined as follows (a pseudo-code version is also given below). Starting with a node representing the initial condition, we build a tree by iteratively adding new "milestones", which are connected to the tree by a feasible trajectory segment. Each new milestone is generated through the generation of a random equilibrium state $x_{rand} \in \mathcal{C} \times \{\bar{v}\}$; the control policy $\pi(\cdot, x_{rand})$ is applied to a randomly chosen node of the current tree, and if the ensuing propagated trajectory is feasible, x_{rand} is added to the tree. The milestones generated according to the above procedure can be regarded as *primary* milestones. In the following, we will assume that x_{rand} is generated from a uniform distribution on the subset of $\mathcal{C}$ defining our workspace; other sampling techniques can be used, as discussed in the earlier sections.

Algorithm 1 Motion planning algorithm

```
1:  Initialize R with (x_0, t_0 + τ);
2:  for i = 1 to N_max do
3:    repeat
4:      generate a new random target x_rand ∈ C × {v̄}
5:      randomly select a node (x, t) in R
6:      if J*(x, x_rand) > J_min then
7:        propagate state using the control policy
          π(·, x_rand), until the state is "sufficiently"
          close to x_rand, and call the resulting state
          (x', t')
8:        if no collisions detected, and (x', t') is τ-
          safe then
9:          add new primary milestone (x', t') to R
10:         select a point at random on the trajectory
            from (x, t) to (x', t'), call it (x'', t'')
11:         add new secondary milestone (x'', t'') to
            R
12:         propagate from (x'', t'') using the control
            policy π(·, x_f)
13:         if no collisions detected then
14:           feasible solution found
15:           add new primary milestone (at the tar-
              get) to R
16:           update upper bounds on cost-to-go.
17:   until time is up
18:   if feasible path found then
19:     descend child tree with minimum upper
        bound on cost
20:   else if no children then
21:     add milestone (x_root, t_root + τ) to the tree
22:   else
23:     descend child tree selected from a random dis-
        tribution, weighted according to the number
        of children in each subtree
24:   destroy old root, and all the children that were
      not chosen
25:   if at destination then
26:     exit with SUCCESS
27: exit with FAILURE
```

As an addition to the above stated rule for generating primary milestones, we will discard the random point x_{rand} before propagating the state if it is "too close" to the randomly selected node in the tree, in the sense defined by the function J^*. In the probabilistic roadmap literature, it is common to try to distribute milestones uniformly over the configuration space $\mathcal{C}$. This seems to produce good results in practice for motion planning in a static workspace. However, there is no theoretical justification for such a criterion in the case of moving obstacles. What is desirable is that the roadmap covers uniformly the *reachable space*.



tb

Fig. 2 Example of generated roadmap (projected on $\mathcal{X}$). Primary milestones are marked as circles, and secondary milestones as squares

As it can be easily recognized, the algorithm outlined above consists of jumps from equilibrium point to equilibrium point, and as such cannot be expected to provide impressive performance, especially in terms of time. In order to increase the performance, we will introduce the following step: at random points along the trajectory to the new randomly generated point, we add $n_s \geq 1$ additional, *secondary*, milestones. Secondary milestones are likely to be at points in the state space that are "far" from equilibria. From secondary milestones we can apply the control policy to the destination $\pi(\cdot, x_f)$: if the resulting trajectory is feasible, we have solved the feasibility problem. In this case, we climb the tree back towards root, updating the estimates on the upper bound on the cost-to-go. Both in the case in which a feasible trajectory is found, and in the case in which a collision is detected, the secondary milestones are added to the tree, and can be selected as the starting point for later iterations. Note that each secondary milestone, by construction, will have a primary milestone in a child subtree (see Fig. 2).

In order to guard ourselves from dead-ends due to finite computation times, we also require that primary milestones be checked for τ-safety. We say that a milestone $(x,t) \in \mathcal{C} \times \{\bar{v}\} \times \mathbb{R}$ is τ-*safe* if $(x,\tilde{t}) \in \mathcal{F}$ for all $\tilde{t} \in [t, t+\tau]$. The time horizon τ should be chosen as a time interval in which at least one new τ-safe milestone can be computed with high probability.

The time available for computation is bounded by either τ, or by the duration of the current trajectory segment. When the time is up, we have to select a new tree from the children of the current root. If there are none, we know that every primary milestone will be τ-safe, and hence we will have τ seconds for computing a new tree (secondary milestones will *always* have at least one child). If we have children, there are two possibilities. In the most favorable case, at least one of the children will lead to the destination through an already computed feasible solution. If there is more than one such feasible solution, we select the one with the least upper bound on the cost-to-go. On the other hand, it can be the case (especially during the first iterations of the search), that no feasible solutions have been computed yet. In this case we randomly select the child to descend according to a distribution weighted on the total number of milestones in each tree. The selected tree will be likely to cover a bigger portion of the reachable set.

Name	Domain
State	$\mathcal{X}$
Time	$\mathbb{R}_+$
CumulativeCost	$\mathbb{R}_+$
LBCost	$\mathbb{R}_+$
UBCost	$\mathbb{R}_+$
NChildren	$\mathbf{N}$

Table 1 Information stored at the tree nodes

Name	Domain
Target	$\mathcal{C}$
IncrementalCost	$\mathbb{R}_+$

Table 2 Information stored at the tree edges

Data structure

The roadmap is constructed as a tree; at the nodes of the tree we store all the information concerning each milestone (see Table (1)). More specifically, the data stored at the tree nodes include the propagated state of the vehicle (i.e. state $x \in \mathcal{X}$, time $t \in \mathbb{R}$), plus information on the "quality" of the motion plan, such as the cumulative cost and estimates on the cost-to-go. Finally, we include 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.(3). The lower bound on the cost-to-go coincides with the value of the cost function $J^*(x, x_f)$, while 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.

On the tree edges, we store information regarding the transitions between single milestones (see Table (2)). This means that we have to store parameters identifying the control law implemented in the transition, namely the target equilibrium point. Finally, we can store the incremental cost incurred during the transition, mainly for bookkeeping purposes.

Analysis

In order to analyze the behavior of the algorithm, with the purpose of ensuring probabilistic completeness and possibly obtaining performance bounds, we have to introduce some additional definitions, and assumptions on the environment characteristics. The remainder of the section will present concepts that are at the basis of most of the available results in analysis

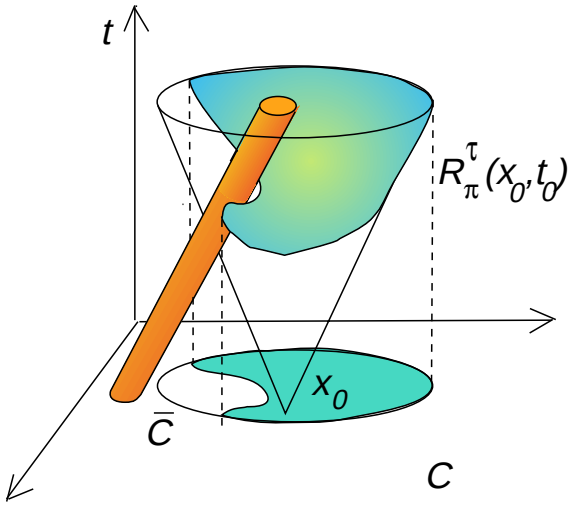


Fig. 3 Visualization of $\mathcal{R}_\pi^\tau(x_0,t_0)$ and its projection on $\mathcal{C}$. The cone represents $\mathcal{R}(x_0,t_0)$

of probabilistic roadmaps. However we will give our own definitions, as needed by adaptation and extension of these concept to our algorithm.

First of all we have to characterize the effect of restricting the trajectories of the system to those that can be obtained through the application of control policies π derived by the optimal cost function J^*. Let us consider a point $(x,t) \in \mathcal{F}$, and an equilibrium point at (x'_{eq},t'). We say that (x'_{eq},t') is (π,τ)-*reachable* from (x,t) if it is the terminal point of the trajectory obtained by applying the control policy $\pi(\cdot,x'_{eq})$ starting from (x,t), the trajectory is feasible, and (x'_{eq},t') is τ-safe. We can define the (π,τ)-reachable set $\mathcal{R}_\pi^\tau(x_0,t_0) \subset (\mathcal{C}\times\{\bar{v}\}\times\mathbb{R}) \cap \mathcal{F}$ as the set of all points that are (π,τ)-reachable from (x_0,t_0); this set belongs to a manifold with the same dimension as $\mathcal{C}$, embedded in the larger space $\mathcal{X}\times\mathbb{R}$ (see Fig. 3). Accordingly, given a set $S \subset \mathcal{F}$, define:

$$\mathcal{R}_\pi^\tau(S) := \bigcup_{p\in S} \mathcal{R}_\pi^\tau(p) \tag{4}$$

Given a set $S \subset F$, denote by $\mu_{\mathcal{C}}(S)$ the volume of its projection on $\mathcal{C}$. Moreover, assume that we are interested in a workspace $\bar{\mathcal{C}}$ such that $\mu_{\mathcal{C}}(\bar{\mathcal{C}}) = 1$.

We have to make sure that for all equilibrium points the (π,τ)-reachable set is not "too small". This property corresponds to the ϵ-goodness of a workspace.[34] In our case, we will say that the planning environment (as defined by both the workspace and the control policy) is (ϵ,τ)-good if, for all sets $S_\tau \subset \mathcal{F}$ of τ-safe equilibrium points we have that:

$$\mu_{\mathcal{C}}(\mathcal{R}_\pi^\tau(S_\tau)) \geq \epsilon \tag{5}$$

Now we have to characterize sets whose (π,τ)-reachable set is "large". Let β be a constant in $(0,1]$; define the β-*lookout* of $S \subset \mathcal{F}$ as:

$$\beta\text{-lookout}(S) := \{p \in S \,| \tag{6}$$
$$|\, \mu_{\mathcal{C}}(\mathcal{R}_\pi^\tau(p)) - \mu_{\mathcal{C}}(S) \geq \beta\,(\mu_{\mathcal{C}}((S)) - \mu_{\mathcal{C}}(S))\}$$

Lastly, we have to make sure that the dimensions of the β-lookout of a set are not too small; we call this property (α,β)-expansiveness.[20] Given two constants α, β in $(0,1]$, the environment is said (α,β)-expansive if for all sets $S \in \mathcal{F}$ we have that:

$$\mu_{\mathcal{C}}(\beta\text{-lookout}(S)) \geq \alpha\mu_{\mathcal{C}}(S) \tag{7}$$

Consider the initial condition (x_0,t_0), and assume it is an equilibrium point (if not, generate a primary milestone using the algorithm presented in the previous section). Let us define the *endgame region* $E \subset \mathcal{C}$ as a region such that all equilibrium points contained in it can be connected without collisions to the desired destination x_f using the policy $\pi(\cdot,x_f)$, for all times t. Then, if the environment is (α,β)-expansive, and the desired destination x_f is contained in the reachable set $\mathcal{R}(x_0,t_0)$, it can be shown that the probability of the algorithm returning a feasible trajectory connecting x_0 to x_f approaches unity exponentially fast:[9]

Theorem 1 (Hsu,Kindel,Latombe and Rock) *Let $g > 0$ be the volume of the endgame region $E \subset \mathcal{C}$, and γ be a constant in $(0,1]$. A sequence M of r primary milestones contains a milestone in E with probability at least $1-\gamma$, if:*

$$r \geq \frac{k}{\alpha}\ln\frac{2k}{\gamma} + \frac{2}{g}\ln\frac{2}{\gamma} \tag{8}$$

where $k := (1/\beta)\ln(2/g)$.

Given the definitions and assumptions stated earlier in this section, the proof in the reference applies to our algorithm. We will not present the proof in detail here, and we refer the reader to the original paper. Roughly, the argument is based on the following facts. Consider the set M of primary milestones in the tree; from the (ϵ,τ)-goodness and expansivity assumptions we have that: (1) uniform sampling on $\mathcal{C}$ will produce target points in $\mathcal{R}_\pi^\tau(M)$ with probability at least ϵ; (2) the newly added milestones will be in the β-lookout of M with probability at least α; (3) each new milestone in β-lookout(M) reduces $\mu_{\mathcal{C}}(\mathcal{R}(M)) - \mu_{\mathcal{C}}(\mathcal{R}_\pi^\tau(M))$ by a factor of at least $1-\beta$.

Note that the proofs in the reference are not completely consistent with the algorithm presented therein: the proofs rely on uniform exploration of reachable sets, while the search algorithm actually explores uniformly on *control inputs*. This inconsistency was noted by the authors, but not resolved rigorously. On the other hand, the same arguments do apply consistently in the case of our algorithm, with

the definitions and under the assumptions given earlier in this section, since we are indeed sampling the (π, τ)-reachable set of milestones uniformly.

Notice that the performance bounds that can be obtained for this algorithm establish only its theoretical soundness, but cannot be used for obtaining an explicit estimate of the probability of successful termination, since α,β, and ϵ cannot be determined easily, for non-trivial environments.

Note also that the use of secondary milestones does not impact adversely the main results on probabilistic completeness (when evaluated on the number of *primary* milestones) , since they only add to the (π, τ)-reachable sets. On the other hand, secondary milestones help the convergence of the algorithm when close to the endgame region, and enhance the overall quality of the resulting trajectory (i.e. the trajectory is "faster", or in general less expensive with respect to the assigned cost).

Application example: Small autonomous helicopter

A small helicopter is a very good example of the systems to which the algorithm presented in this paper can be profitably applied, especially when used in conjunction within a hybrid control framework.[18]

Radio-controlled helicopters are capable of remarkably agile and aggressive maneuvers, which are impossible to perform using traditional control techniques, especially when the on-line solution of the motion planning problem is required. Among the reasons for that we can state that helicopters are essentially unstable systems, with a very high bandwidth, and their dynamics change considerably throughout the flight envelope.

In this section, we will present simulation results for a test case involving a small autonomous helicopter. The simulations are carried out on a fully non-linear helicopter simulation, based on a widely used minimum-complexity model.[35] The motion planning algorithms operating on the hybrid automaton structure[18,19] are complemented by the control law presented in[36] to ensure tracking of the reference trajectory.

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 examples that we will discuss in the following are based on the examples presented in a previous work of the authors.[19] The cost function used in all the examples is the total time needed to go to the destination (we are solving a minimum-time problem). The algorithm was implemented in C on a 300 MHz Pentium II computer.

The first examples involves navigating the helicopter through a set of obstacles represented as spheres in the configuration space. Note that in the following examples we constrain the helicopter to fly at constant altitude (even though the model and the control laws are fully three-dimensional). We tested the algorithm in several runs, involving both fixed and randomly moving spheres. This particular scenario was easily tackled by our algorithm: the mean computing time on one thousand runs was $0.45s$ (standard deviation $0.24s$) in the fixed obstacles case, and $0.46s$ (standard deviation $0.28s$) in the randomly moving obstacle case. Note that the fact that the obstacles are moving does not result in a significant increase in the computation time. An example of a computed trajectory is presented in Fig. 4, and a histogram of the computation times is reported in Fig. 5.

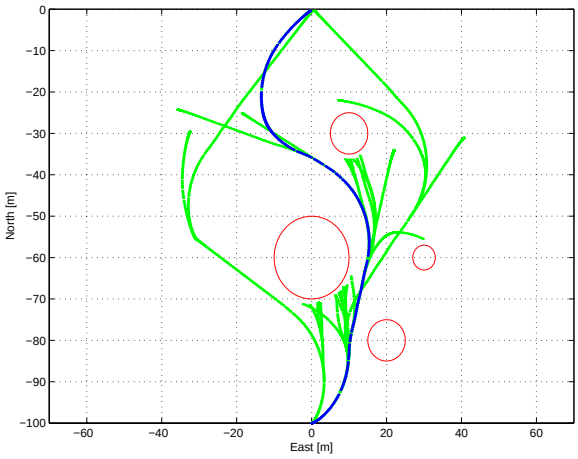


Fig. 4 Example 1: trace of the trajectory tree, and best trajectory found

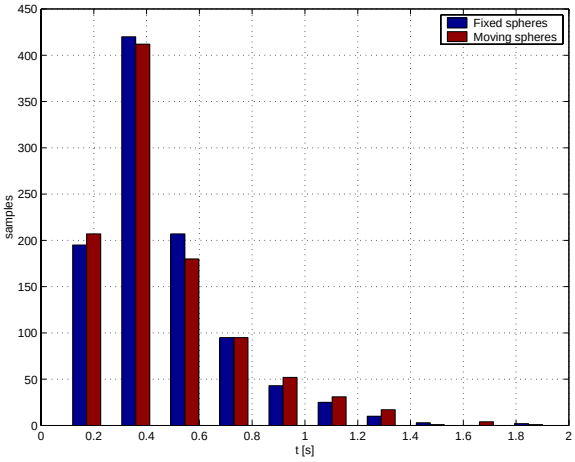


Fig. 5 Example 1: computation times

Note that the computation times are very small when compared to the time scales for the required maneuver switches in the hybrid formulation; in this particular case the planner has about 2 seconds of time at its disposal, before deciding what maneuver to execute after the initial acceleration from hover to forward flight at maximum speed. This scenario was particularly easy, since the environment does not present "narrow passages" (i.e. it is highly expansive).

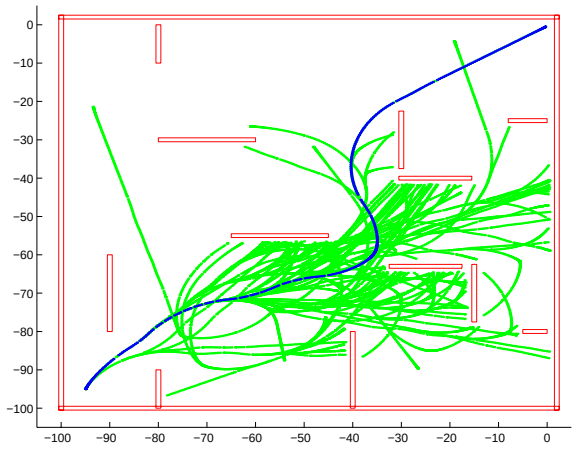


Fig. 6 Example 2: trace of the trajectory tree, and best trajectory found

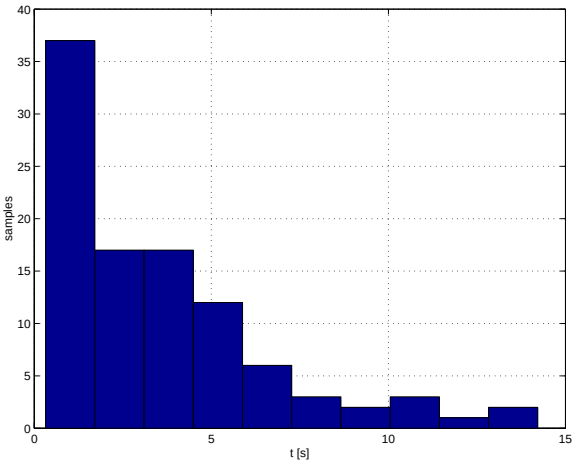


Fig. 7 Example 2: computation times

A second example involves navigating through a maze of fixed walls. This example proved to be slightly more difficult for our algorithm, which sometimes was forced to move down a subtree not yet proven feasible. The average computation time was $3.5s$, and the standard deviation was $3.1s$. A computed trajectory is presented in Fig. 6, and a histogram of computation times is presented in Fig.7.

Lastly, we examined a scenario in which the helicopter had to fly through moving slots in two walls dividing the workspace into three components. This problem proved to be quite difficult for our algorithm, which often resulted in the helicopter to "hop" between different hovering points in front of one of the walls, looking for a feasible solution. In this case, the algorithm in this paper actually behaved more poorly than a similar algorithm proposed by the authors,[19] which was endowed with a more complex heuristic (but for which no formal performance bounds were available). Animations showing the behavior of the helicopter in all the above scenarios are available from the authors.

Conclusions

In this paper a new algorithm for autonomous vehicle motion planning, based on a probabilistic roadmap approach, has been presented. The algorithm is based on the use of Lyapunov functions for motion planning, and hence provides a very efficient and natural way of dealing with the system dynamics, with the associated stability guarantees. Moreover the algorithm is very general, and can be used with both continuous state-space and hybrid models of a vehicle's dynamics. Also, the configuration space needs not be $\mathbb{R}^n$, but we can apply our algorithm in the case in which $\mathcal{C}$ is a more general manifold. This make it possible to directly address also attitude motion planning, i.e. motion planning on $SO(3)$, which is a problem of interest in many aerospace application (e.g. spacecraft attitude motion with pointing constraints).

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 will address motion planning in uncertain environment, with limited sensor range, and multi-vehicle operations.

Acknowledgments

This research was supported by the C. S. Draper Laboratory through the IR&D grant DL-H-505334, by the AFOSR, under grant F49620-99-1-0320, and by the ONR, under Young Investigator Award N-00014-99-1-0668.

References

[1] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8]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.

[9]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.

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

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

[12]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.

[13]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.

[14]F. Bullo. Stabilization of relative equilibria for systems on riemannian manifolds. In *American Control Conference*, pages 1618–1622, San Diego, CA, June 1999.

[15]J.E. Marsden. *Lectures on Mechanics*. Cambridge University Press, New York, NY, 1992.

[16]J. E. Marsden and T.S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer Verlag, New York, NY, 1999. Second edition.

[17]V.I. Arnold. *Mathematical Methods of Classical Mechanics*, volume 60 of *GTM*. Springer Verlag, New York, NY, second edition edition, 1989.

[18]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.

[19]E. Frazzoli, M.A. Dahleh, and E. Feron. Robust hybrid control for autonomnous vehicle motion planning. Technical Report LIDS-P-2468, Massachusetts Institute of Technology, 1999. Revised version submitted to IEEE Trans. on Automatic Control.

[20]D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proceedings of the 1997 IEEE International COnference on Robotics and Automation*, 1997.

[21]R. Bohlin and L. Kavraki. Path planning using lazy prm. In *International Conference on Robotics and Automation*, San Francisco, CA, 2000.

[22]N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation (ICRA'96)*, pages 113–120, 1996.

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

[24]M. S. Branicky and K. Olson. Quasi-random roadmaps: Using deterministic low discrepancy point sets to break the curse of dimensionality in motion planning. Submitted to the 4th Int. Workshop on Algorithmic Foundations of Robotics, 1999.

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

[26]A.E. Bryson and Y.C. Ho. *Applied optimal control : optimization, estimation, and control*. Hemisphere Pub. Corp., New York, 1975.

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

[28]L.E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[29]J.A. Reeds and R.A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2), 1990.

[30]P. Soueres and J.D. Boissonnat. Optimal trajectories for nonholonomic mobile robots. In J.P. Laumond, editor, *Robot Motion Planning and Control*, volume 229 of *Lecture Notes in Control and Information Sciences*. Springer, 1998.

[31]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.

[32]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.

[33]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*, December 1999.

[34]L.E. Kavraki and J.C. Latombe. Probabilistic roadmaps for robot path planning. In K Gupta and A del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 33–53. John Wiley, 1998.

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

[36]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.