JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 25, No. 1, January–February 2002

Real-Time Motion Planning for Agile Autonomous Vehicles

Emilio Frazzoli*
University of Illinois at Urbana–Champaign, Urbana, Illinois 61801
and
Munther A. Dahleh† and Eric Feron‡
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

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 path 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. The path planning algorithm retains the convergence properties of its kinematic counterparts. System safety is also addressed in the face of finite computation times by analyzing the behavior of the algorithm when the available onboard computation resources are limited, and the planning must be performed in real time. The proposed algorithm can be applied to vehicles whose dynamics are described either by ordinary differential equations or by higher-level, hybrid representations. Simulation examples involving a ground robot and a small autonomous helicopter are presented and discussed.

I. Introduction

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

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 accomplish an assigned task such as performing an observation or delivering a payload), while avoiding collisions with fixed or moving obstacles.

The problem of planning a trajectory in an environment cluttered by obstacles has been the object of considerable interest in the robotics and artificial intelligence communities[1–4]; most of the activity has focused on holonomic or nonholonomic kinematic motion problems. Roughly speaking, it is possible to identify three general approaches to the motion-planning problem, namely, cell decomposition methods, roadmap methods, and artificial potential field methods.[1]

Many motion-planning algorithms rely on the notions of configuration and configuration space. A configuration of a robot identifies the position of all of its points with respect to an inertial reference frame. We will assume that such a configuration can be described by a finite number of parameters. The configuration space is the set of all possible configurations of the robot.

Cell decomposition methods rely on the partition of the configuration space into a finite number of regions, in each of which collision-free paths can be found easily. The motion-planning problem then is translated into the problem of finding a sequence of neighboring cells, including the initial and final conditions.[5]

In roadmap methods a network of collision-free connecting paths is constructed, which spans the free configuration space (i.e., the subset of the configuration space that does not result in collisions with obstacles). The path-planning problem then reduces to finding paths connecting the initial and final configuration to the roadmap and then selecting a sequence of paths on the roadmap. There are several methods for building such a roadmap, among which we can mention visibility graphs[6,7] and Voronoi diagrams.[8]

Finally, in artificial potential field methods a collision-free trajectory is generated by the robot moving locally according to "forces" defined as the negative gradient of a potential function.[9–11] This function is designed to provide attractive forces toward the goal and repulsive forces, which push the robot away from obstacles (the potential function is bowl shaped with the goal at the bottom, and obstacles are represented by peaks). This class of methods is based on the definition of a feedback control policy (i.e., the control is computed at each instant in time as a function of the current state), as opposed to the open-loop approach of the preceding two classes. A shortcoming of this formulation is the possible existence of local minima in which the robot might become trapped. An artificial potential function, which does not have local minima, is a said navigation function, but computing such a function is in the general case as difficult as solving the motion planning problem for all initial conditions.[12]

The algorithms for motion planning must be evaluated in terms of completeness and computational complexity. An algorithm is said to be complete if it returns a valid solution to the motion-planning problem if one exists and returns failure if and only if the problem is not feasible: This is what we will call a correct termination for a motion-planning algorithm. The computational complexity of some basic formulations of the motion planning problem has been studied in detail. The so-called generalized mover's problem, involving motion planning for holonomic kinematic robots made of several polyhedral parts among polyhedral obstacles, has been proven by Reif[13] to be PSPACE hard. (The complexity class PSPACE includes decision problems for which answers can be found with resources,

Received 20 November 2000; revision received 20 August 2001; accepted for publication 23 August 2001.

*Assistant Professor, Department of Aeronautical and Astronautical Engineering, 321b Talbot Laboratory, 104 S. Wright Street; frazzoli@uiuc.edu. Member AIAA.

†Professor, Department of Electrical Engineering and Computer Science, 77 Massachusetts Avenue, Room 35-401; dahleh@lids.mit.edu.

‡Associate Professor, Department of Aeronautics and Astronautics, 77 Massachusetts Avenue, Room 35-417; feron@mit.edu. Senior Member AIAA.

such as memory, which are polynomial in the size of the input. The run time is not constrained. The complexity class NP is known to be a subset of PSPACE; moreover, it is believed to be a proper subset.[14]) An algorithm for path planning in a configuration space of dimension n, with obstacles defined by m polynomial constraints of degree d, has been established by Schwartz and Sharir[15]; the time complexity of the algorithm is (twice) exponential in n and polynomial in m (geometrical complexity) and in d (algebraic complexity). The most efficient algorithm currently available for solving the same problem, from Canny,[16] is singly exponential in n. The preceding results strongly suggest that the complexity of the path-planning problem grows exponentially in the dimension of the configuration space.

Moreover, kinematic, holonomic path planning is not enough for many problems of interest, particularly problems involving "agile" autonomous vehicles, for which we have to take into account the additional constraints on the vehicle's motion arising from its dynamics or from nonholonomic constraints (that is, nonintegrable constraints on the state and its derivatives).

Even though in some cases it is possible to force a dynamical system to follow trajectories generated by a kinematic model, this is not true in general, especially when constraints on the available control inputs are taken into account.

For example, differentially flat systems[17] can follow any arbitrary trajectory for the so-called flat outputs.[18] However, no general method is available yet to ascertain differential flatness of a given system. Moreover, even for differentially flat systems currently there is no straightforward way of taking into account control saturation or flight envelope constraints (some advances in this direction appeared recently[19,20]).

As another example, some nonholonomic systems are able to approximate arbitrary paths, but usually this requires the execution of complex sequences of small-amplitude movements about the reference path, which could possibly result in a large loss in terms of performance of the final trajectory.[3,21–23]

If the output of a kinematic planner is used as a reference trajectory for an inner loop consisting of a tracking control law, the discrepancies between the planned trajectory and the trajectory actually executed by the system can be relevant and lead to collision with obstacle even in the case in which the planned trajectory was collision free. This is true, for example, 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 at some nominal velocity) are comparable.

As a consequence, it is desirable that motion-planning strategies take fully into account the dynamics of the vehicle. In other words, it is desirable that the output from the motion planning be executable by the system dynamics: this is the object of a recent direction in motion planning research, usually referred to as *kinodynamic motion planning*.

Clearly, such a problem is at least as difficult as the path-planning problem. Moreover, constraints deriving from the system's dynamics or from nonholonomic constraints cannot be represented as "forbidden zones" in the state space (i.e., the space encoding the configuration of the vehicle, as well as its velocities). As a consequence, the direct application of kinematic motion-planning techniques to the dynamic case is not possible in the general case.

Perhaps the best formulated general method for addressing motion-planning problems is the use of optimal control.[24,25] However, the solution of such a problem using the traditional optimal control tools (such as variational calculus, the minimum principle of Pontryagin and other necessary conditions[26] and dynamic programming[27]) is computationally intractable for all but trivial cases. In addition, some techniques from kinematic motion planning can be also extended to the dynamic case by planning motions in the state space instead of the configuration space.

However, there is strong evidence that any deterministic and complete algorithm for the solution of kinodynamic motion-planning problems will require at least exponential time in the dimension of the state space of the dynamical system, which is usually at least twice the dimension of the underlying configuration space, and polynomial in the number of obstacles. As a consequence, available algorithms are implementable in practice, at least at the current technology levels, only for systems of very small dimension (e.g., less than five). Because the state space of aerospace vehicles is at least 12 (when vehicles are modeled as rigid bodies), one has to resort to heuristic techniques or seek alternative formulations of the problem.

To circumvent the computational complexity of deterministic, complete algorithms, a new class of motion-planning algorithms, known as probabilistic roadmap (PRM) planners, was introduced by Kavraki et al.[28] and Overmars and Svestka[29] and subsequently refined.[4,30–33] The PRM path-planning architecture was first introduced as a fast and efficient algorithm for geometric, multiple-query path planning. The original PRM planner is based on an off-line preprocessing phase and an on-line query phase. The preprocessing phase is aimed at constructing a graph of feasible paths in the entire configuration space (the roadmap), which would make future queries easy to solve. The on-line query phase selects an appropriate path from the already computed roadmap, together with the computation of two "short" paths to connect starting and ending points to the closest nodes (or milestones) of the roadmap. The PRM algorithm has been proven to be complete in a probabilistic sense, that is, the probability of 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 (i.e., its expansiveness, tied to the rate at which the set of points that can be connected to the roadmap grows with the number of milestones), which prove that the probability of correct termination approaches one exponentially fast in the number of milestones.[33]

However, for many path-planning applications building a roadmap a priori may not be required, or even feasible (e.g., for planning in a dynamic, rapidly changing environment). In addition, the basic roadmap algorithms neglect the vehicle dynamics: the vehicle usually needs to navigate along a piecewise linear trajectory with very sharp turning points. To address both points, a new randomized motion-planning algorithm was developed by introducing the concept of rapidly exploring random trees (RRTs).[34–36] The RRT algorithm consists of building on-line a tree of feasible trajectories by extending branches toward randomly generated target points. Although in the PRM approach the idea was to explore the configuration space exhaustively in the preprocessing phase, RRTs tend to achieve fast and efficient single-query planning by exploring the environment as little as possible. A significant feature of the RRTs is that the resulting trajectories are by definition executable by the underlying dynamical system. In addition, under appropriate technical conditions, the RRT algorithm has been proven probabilistically complete,[36] that is, the probability of finding a path from origin to destination converges to one if such a feasible path exists.

In Hsu et al.[37] a new incremental roadmap building algorithm is introduced that provides not only probabilistic completeness, but also recovers performance guarantees on the algorithm. That is, the probability of the algorithm finding a solution if one exists converges to one exponentially fast with the number of random samples used to build the tree. Interestingly, the rate of convergence does not depend on the number of obstacles, but rather on geometric properties of the environment (e.g., its expansiveness). 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 tree. Practical implementations (which have reportedly demonstrated excellent performance[37]) 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 recent developments, a new randomized, incremental motion-planning algorithm is proposed in this paper. 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 configuration and velocity) to any desired configuration 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, it is shown that this

path-planning algorithm satisfies the technical conditions elicited by Hsu et al.[37] and therefore offers guaranteed performance in the form 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 motion plan. Moreover, in the case in which the obstacle-free planner is optimal with respect to some meaningful cost it is possible to implement branch-and-bound algorithms to bias the search for a solution to maximize the performance of the computed solution.

The paper is organized as follows: first, the planning framework is introduced, including specifications on the system closed-loop dynamics, on the environment, and on the path-planning problem in the presence of obstacles. The randomized, real-time motion-planning algorithm for agile vehicles is then introduced and discussed, and its performance is analyzed. Finally, the randomized algorithm is demonstrated on a ground robot example first, and then on the model of a small autonomous helicopter. Simulation results are presented and discussed.

II. Motion-Planning Framework

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 configuration at rest. Thus, rather than working with an open-loop system, as presented in earlier publications,[34,36,37] our basic dynamical system is a closed-loop one.

A. System Dynamics

In this section we introduce the dynamics of the systems for which the motion-planning algorithm is applicable. Because this paper concentrates mostly on guidance and planning tasks, we introduce two guidance models, which we believe are relevant to the problem under consideration.

1. System Representation via Ordinary Differential Equations

The usual representation of the dynamics of an autonomous vehicle or robot is a set of ordinary differential equations (ODEs) of the form

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, u) \tag{1}$$

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 $\mathcal{U} \subseteq \mathbb{R}^m$. The preceding formulation can include both nonholonomic and dynamic constraints.[38] In some cases additional inequality constraints of the form

$$F(x) \leq 0 \tag{2}$$

must be added on the state variables to ensure safe operation of the system (e.g., flight envelope protection). In Eq. (2) $F(x)$ can represent a vector of constraints, and the inequality must be understood component-wise.

Finally, assume that the state space $\mathcal{X}$ can be decomposed, at least locally, into the product $\mathcal{C} \times \mathcal{Y}$ and that the system dynamics, as well as the flight envelope constraints, are invariant with respect to group actions (e.g., translations or rotations) on $\mathcal{C}$.

The space $\mathcal{C}$ is 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. In a Lagrangian mechanics setting these variables correspond to the so-called cyclic variables, which do not appear explicitly in the Lagrangian of the system.[39] In many cases of interest, one is effectively interested in planning the motion on such a reduced configuration space. For example, when planning the motion of a helicopter one is interested in specifying a final hovering position and possibly heading. Pitch and roll angle at this final condition are not specified, as long as they are such to allow hovering.

The space $\mathcal{Y}$ encodes the remaining configuration variables, as well as the vehicle's velocity and higher-order derivatives of the configuration variables. These somewhat abstract notions are more clearly illustrated on simple examples.

Example 1 (system with integrators): Consider the system described by the following set of ODEs:

$$\dot{z} = y, \qquad \dot{y} = f(y, u)$$

In this case the space $\mathcal{C}$ is spanned by the vector z, whereas the space $\mathcal{Y}$ is spanned by y.

Example 2 (attitude dynamics): Consider the attitude dynamics of a spacecraft in the absence of a gravitational field

$$\dot{R} = R\hat{\omega}, \qquad J\dot{\omega} = -\omega \times J\omega + M(u)$$

In the preceding equations J is the inertia tensor of the spacecraft, M denotes the moments generated by the controls u, and the skew matrix $\hat{\omega}$ is defined as the unique matrix for which $\hat{\omega}v = \omega \times v$ for all vectors $v \in \mathbb{R}^3$. In this case the space $\mathcal{C}$ corresponds to the group of rotations R in the three-dimensional space $SO(3)$, and the space $\mathcal{Y}$ corresponds to the linear space of velocities in body axes $\omega \in \mathbb{R}^3$.

Example 3 (helicopter dynamics): Finally, consider the following simplified model of the motion of a helicopter[40,41]

$$\dot{x} = v, \qquad m\dot{v} = mg + RF_b(v, \omega, u)$$

$$\dot{R} = R\hat{\omega}, \qquad J\dot{\omega} = -\omega \times J\omega + M_b(v, \omega, u)$$

where x is the position of the center of mass of the vehicle, v is its velocity in an inertial reference frame, $R \in SO(3)$ represents the attitude of the helicopter, and ω is the vector of angular velocities in body axes. The mass and rotational inertia of the vehicle are indicated by m and J, and g represents the gravity acceleration. Finally F_b and M_b are respectively the forces and moments (expressed in body axes) generated by the controls u. In this case, because the dynamics of the vehicle do not depend on its position and on its heading angle, the space $\mathcal{C}$ corresponds to the space of translations in the position space and rotations about a vertical axis. The variables in $\mathcal{Y}$ encode the remaining degrees of freedom (namely, roll and pitch rotations), as well as the translational and rotational rates. This decomposition is only local.

2. Hybrid System Representation

Although the formulation in terms of ODEs is probably the most commonly used representation of a vehicle's dynamics, it might not be the most appropriate choice for the purpose of vehicle guidance. In particular, the state space of nontrivial 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.[42–44] This alternative formulation of the vehicle dynamics builds on a formalization of the concept of maneuver, which in a more unstructured form, is commonly referred to in the aerospace literature.[45–49]

The main feature of a maneuver model is the selection of appropriate motion primitives. Closely related to the invariance of the system dynamics with respect to group actions on $\mathcal{C}$ is the existence of trim trajectories (or relative equilibria). Loosely speaking, a trim trajectory can be defined as a trajectory where the vehicle is in a state of equilibrium relative to a body-fixed reference frame. During a trim trajectory, the state variables in $\mathcal{C}$ evolve according to the system dynamics, whereas the state variables in $\mathcal{Y}$ are fixed, as are the controls u. The existence of trim trajectories is a fundamental property of the system dynamics, and it is natural to consider trim trajectories as a first class of motion primitives. A finite number of such trajectories can be selected, for example, by gridding the compact subset of $\mathcal{Y} \times \mathcal{U}$ defined by the flight envelope and control saturation constraints.

Example 4 (autonomous vehicle dynamics): Most ground and air 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 are negligible), we can also assume invariance to vertical translation. In this case trim trajectories are helicoidal curves, with a vertical axis.

Such a selection of trajectory primitives is appropriate for systems in which the dynamic behavior can be neglected (e.g., kinematic systems, aircraft models for air traffic control). If the dynamics of the system cannot be neglected, the transients related to switches between different trim trajectories must be accounted for. A second class of primitives, called *maneuvers*, are defined as feasible, finite-time transitions between trim trajectories. Maneuvers can be designed by solving a local optimal control problem or can be derived from flight data.

Although maneuvers have a definite time duration, trim trajectories can in principle be followed indefinitely. As a consequence, the decision variables, or control inputs, 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 result of the process of extracting trajectory primitives is the creation of a guidance-level system representation as a maneuver automaton, whose states are the trim states of the vehicle and whose transitions correspond to maneuvers. The maneuver automaton can be seen as a new model of the vehicle, in which the continuous dynamics ODEs (1) are replaced by the transition rules on the directed graph representing the automaton and by the associated hybrid system evolution. The full state of the system will then be described by the maneuver primitive being executed and the time and position on $\mathcal{C}$ of its inception. Using this representation, the assumption of invariance to group actions in $\mathcal{C}$ is enough to ensure that the maneuver automaton encodes all of the relevant information about dynamics and the flight envelope constraints of any vehicle in a space of smaller dimension, that is, $\mathcal{M} = \mathcal{C} \times Q_T$, where Q_T is the set of indices of the selected trim trajectories.

Example 5: Many planar nonholonomic robot models assume that the robot, cruising at constant speed, can only move straight forward, or turn left with a given radius, or turn right with the same turning radius. In this case, assuming the robot to be a kinematic, rigid body, $\mathcal{C}$ would be the set of all positions and orientations of the robot, and $Q_T = \{\text{move straight, turn left, turn right}\}$.

A full discussion of the maneuver automaton framework is outside of the scope of this paper, but is available on other publications by the authors,[44] to which we refer the interested reader.

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

B. Obstacle-Free Guidance System

The second major element that is assumed to be available is a guidance algorithm in environments with no obstacles. In this paper we consider problems in which the desired destination is a relative equilibrium or trim trajectory. 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 $\mathcal{T}(x_{\text{eq}})$, centered at a relative equilibrium x_{eq}. Because the system dynamics are invariant with respect to group actions on $\mathcal{C}$, a family of relative equilibria can be expressed by a point in $\mathcal{C} \times \{\bar{y}\}$, where $\bar{y} \in \mathcal{Y}$ is a constant.

For simplicity, in the examples we will only consider equilibrium points (i.e., terminal conditions with zero velocity) and will consider only autonomous vehicles, which are indeed able to "stop." These include autonomous helicopters, vertical takeoff and landing aircraft, spacecraft, ground vehicles, and surface/underwater vessels. However, the algorithm that will be presented is applicable to other vehicles such as fixed-wing aircraft and paragliders, as long as the terminal equilibrium condition for the guidance law, such as "hover at point p," is replaced by a relative equilibrium condition, such as "enter a steady turn starting at point p."

Although, 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 can be computed analytically.[24,25] This is the case of system with linear dynamics with 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 might not admit closed-form solutions, can be solved numerically via the approximate or exact solution to an appropriate optimal control problem by minimizing a cost functional of the form

$$J[x(\cdot), u(\cdot)] = \int_{t_0}^{t_f} \gamma[x(t), u(t)]\, \mathrm{d}t \tag{3}$$

for some initial conditions x_0, under the boundary condition $x(t_f) \in \mathcal{T}(x_{\text{eq}})$, and the dynamics and flight envelope constraints (1) and (2). We make the additional assumption that the incremental cost $\gamma(x, u)$ is invariant with respect to group actions on $\mathcal{C}$. Such a formulation includes for example minimum time, minimum path length, and minimum energy control problems.

Advances in computer power, combined with appropriate plant simplifications (such as the introduction of the maneuver model outlined earlier), make it possible in many cases of practical interest to compute and store an approximate expression for the optimal cost-to-go function (or return function) $J^*(x, x_{\text{eq}})$, for all $x \in \mathcal{X}$, and all equilibrium points $x_{\text{eq}} \in \mathcal{C} \times \{\bar{y}\}$, using for example iterative methods.[42,50] Considering the case of a small autonomous helicopter represented by a maneuver model such as introduced earlier in this paper, storage of such a cost function required about 1 MB of memory, easily implementable on a computer (the cost function was approximated using a simple look-up table, with semilogarithmic spacing).[42] Other approaches for kinematic motion planning of nonholonomic vehicles involve the construction of optimal solutions via the interconnection of canonical paths.[51–53]

If the optimal cost function $J^*(x, x_{\text{eq}})$ is known for all $x \in \mathcal{X}$ and all equilibrium points $x_{\text{eq}} \in \mathcal{C} \times \{\bar{y}\}$, 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.[54] 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 toward a target set with a finite cost. Moreover, the optimal cost-to-go function provides a meaningful measure of the distance between the initial conditions and the target.

Along with the optimal control law in an obstacle-free environment, we assume the availability of a simulator of the system, that is a function which is able to predict the evolution of the closed-loop system, and generate the corresponding (state, time) trajectory. Such a simulator can be easily developed from the knowledge of the system's dynamics and of the optimal control law.

C. 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. In this case obstacle avoidance constraints can be written a priori as

$$G(x, t) \leq 0 \tag{4}$$

where $G(x, t)$ can be a vector of constraints and the inequality must be understood component-wise. Because the environment is time-varying, collisions must be checked on (state, time) pairs $(x, t) \in \mathcal{X} \times \mathbb{R}$. For this purpose a collision-checking algorithm is assumed to be available, via trajectory sampling or other appropriate method.

The assumption that information on the obstacle motion is available a priori is admittedly very limiting, because in most practical applications this kind of information is not available, especially if the obstacles are actually other vehicles, moving according to their own control laws and goals. Nevertheless for the time being we will concentrate on this problem, which is already extremely challenging.

It is important to explicitly remark that the simple fact that the obstacle avoidance constraints (4) are time varying (i.e., the obstacles are moving) translates into hard real-time requirements for the motion-planning algorithm, even under a perfect information assumption on the motion of the obstacles. This is a consequence of the fact that the feasibility of the computed solution with respect to the obstacle avoidance constraints has to be checked on (state, time) couples: the solution must be a time-parameterized trajectory, where the time "tags" are of fundamental importance. Because of the finite computation times in physical systems, the motion planning has to be carried out starting from initial conditions at some time in the future (i.e., a finite lead time is needed). If the motion-planning algorithm is too late in delivering the solution to the vehicle, that is, if the computation time exceeds the allowed lead time, the late solution is in general not guaranteed to be feasible.

Before proceeding to the problem formulation, we need to introduce some notation. The feasible set $\mathcal{F} \subset \mathcal{X} \times \mathbb{R}$ is defined to be the set of all pairs (x, t) for which no collisions occur, 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 preceding initial conditions is feasible, and terminates inside the set $\mathcal{T}(x_f)$. In other words, we say that (x_f, t_f) is reachable from (x_0, t_0) if the time-parameterized curve $\chi : [t_0, t_f] \to \mathcal{X}$ is an integral curve of the ODE (1) (or can be executed as a sequence of maneuvers encoded in the maneuver model), given the control input $\hat{u}(t)$, and is such that $\chi(t_0) = x_0$, $\chi(t_f) \in \mathcal{T}(x_f)$, 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

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

D. 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 $x_f \in \mathcal{C} \times \{\bar{\nu}\}$, find a control input $v : [t_0, t_f] \to \mathcal{U}$ 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 a time t_f such that $(x_f, t_f) \in \mathcal{R}(x_0, t_0)$ and returns failure otherwise. Although 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 a cost of the form (3).

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.

To implement our randomized motion-planning algorithm, we need to limit the feasible trajectories of the system to a compact subset of the state space $\mathcal{X}$ and hence a compact subset of the reduced configuration space $\mathcal{C}$. This can be done easily through the introduction of a confinement constraint of the form (4). The motivation for such a requirement derives from the necessity of generating uniform distributions of target equilibria.

III. Motion Planning in the Presence of Obstacles

The motion-planning algorithm in the presence of obstacles is based on the determination of a sequence of random attraction points x_r and the corresponding control laws $\pi(\cdot, x_r)$, which effectively steer 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. Because the actual control inputs can be computed from the knowledge of the optimal control policy $\pi(\cdot, x_r)$, 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. In other words, the output of the guidance layer is control policies, not reference states or inputs. 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.

An additional advantage of the proposed approach is that the scheduling of feedback control policies provides robustness to external disturbances, uncertainties and modeling errors, which is missing in other, open-loop, motion-planning approaches.

The ideas just outlined in a probabilistic roadmap setting can be seen as a motion-planning technique through scheduling of control policies and corresponding Lyapunov functions [i.e., the optimal cost-to-go functions $J^*(\cdot, x_r)$]. Although the concept is not entirely new in control theory,[55–57] to the author's 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. In other words, in the references just mentioned there exists a preestablished partial ordering of the Lyapunov functions: the system is driven through the attraction domains of a predefined sequence of Lyapunov functions (and corresponding control policies) that draws the state closer and closer to the origin. In our algorithm such a sequence is computed on-line.

Imposing that the motion plan be constructed as a sequence of local optimal control policies steering the system toward a sequence of attraction points limits the class of possible motion plans, which are output by the algorithm. Hence, the motion-planning algorithm we are going to present will not be complete in the sense that it is not capable of generating all possible feasible trajectories. On the other hand, we can analyze the completeness of the motion-planning algorithm with respect to the class of trajectories it is able to generate.

Given the control policy π, we say that a point (x_f, t_f) is π reachable from (x_i, t_i) if the closed-loop evolution of the system under the control policy $\pi(\cdot, x_f)$ with initial conditions (x_i, t_i) is such that a feasible, collision-free trajectory is generated through (x_f, t_f) (or a neighborhood thereof). The π reachable set $\mathcal{R}_\pi(x_i, t_i)$ is thus defined as the set of all (state, time) couples, which are π reachable from (x_i, t_i); this set is bounded from below (in the sense of time) by a manifold with the same dimension as the symmetry group H, embedded in the larger space $\mathcal{X} \times \mathbb{R}$. Accordingly, we can define the π reachable set of a set S as:

$$\mathcal{R}_\pi(S) := \bigcup_{(x,t) \in S} \mathcal{R}_\pi(x, t)$$

In the following, we will consider a weaker notion of completeness, limited to trajectories that can be generated by the application of a sequence of given control policies from the initial conditions.

A. Overview of the Algorithm

The basic idea of the algorithm is the following: starting from the initial conditions (x_i, t_i), we incrementally build a tree of feasible trajectories, trying to explore efficiently the reachable set $\mathcal{R}(x_i, t_i)$. (A tree is a directed graph, with no cycles, in which all nodes have several outgoing edges, to each of which corresponds another node, called a *child*. A child node can in turn have several other children. All nodes, excluding the root, have one and only one incoming edge, which comes from the *parent*. The root has no parent. A common example of a tree is the directory structure in a computer file system).

At each step we will add a new branch (edge) and a new milestone (node) to the tree. For this purpose at each tree expansion step we have to address two points: 1) which node do we want to expand? and 2) in which direction shall we explore?

The first incremental randomized motion-planning algorithm was recently introduced by LaValle and Kuffner,[35] based on previous work by Kavraki et al.,[30] with the name of RRT. In the original RRT

the answers to the preceding questions were provided (roughly) in the following way: 1) pick a configuration x_r at random, and choose the node to expand as the closest (in the sense of a Euclidean distance) node currently in the tree; and 2) apply a constant input for a time δt in such a way that the final point is moved as close as possible to x_r. If the resulting trajectory is feasible, then it is added to the tree. The preceding procedure is iterated until one of the nodes of the tree is close enough to the target point x_f.

Although the RRT algorithm proved to be extremely efficient in many difficult problems, as reported in the literature,[34–36] it could not be appropriate in the general case of a dynamical system. Selecting the control inputs according to a greedy policy on the Euclidean distance could result into instability for the dynamical system (it essentially amounts to pure proportional feedback). Also, only a probabilistic completeness proof is available. Even though it is guaranteed that the probability of correct termination of the algorithm converges to one as the number of iterations increases there are no indications on how fast this convergence will occur. Moreover, in a dynamic environment case this approach is not probabilistically complete and might fail altogether, as will be shown in the examples.

In Hsu et al.[37] a different approach was introduced, based on the following main steps: 1) choose the node to be expanded at random, and 2) apply a random control input for an interval δt.

The algorithm has been proven to be probabilistically complete, even in the presence of moving obstacles. Moreover, its proponents were able to prove performance bounds on the algorithm, provided that at each iteration the reachable set of the set of current milestones (nodes in the tree) is explored uniformly. This is not accomplished in the general case by the two steps just outlined: a random selection of the inputs from a uniform distribution does not necessarily result into a uniform distribution on the outputs (the location of the new candidate milestone). Moreover, a uniform random choice of the node to be expanded does not result in a uniform exploration of the reachable set of the tree because the reachable sets from each of the nodes overlap and can have different volumes. As a consequence, the exploration mechanism considered in the proof was not the one outlined in the preceding two steps, but was instead assumed to be the output of an idealized procedure, denoted as IDEAL-SAMPLE. The random selection of nodes and control inputs was used as an approximation of IDEAL-SAMPLE in the practical implementation. Moreover, the complete randomness of both the main steps in the tree construction results in the generations of trees, which appear to lack in sense of purpose and do not appear to explore very efficiently the free configuration space (efficient exploration was, on the other hand, the main advantage of the RRT algorithm).

To address these problems, we advocate the fact that the optimal cost function in the obstacle-free environment provides the most appropriate information to address both the issues of node selection and trajectory generation. Using the optimal control function and the corresponding optimal control policy, we will indeed be able to implement an IDEAL-SAMPLE procedure and achieve the corresponding performance bounds.[37] The key ideas are the following: the correct measure of distance is the optimal cost-to-go, and the optimal input can be derived from the associated optimal control policy. Moreover, the optimal (feedback) control policy can be seen as the inversion mechanism, which translates the uniform distribution in the output space (i.e., the location of the newly generated candidate milestone x_r) into the corresponding distribution in the input.

In our algorithm we proceed as follows: 1) pick a configuration x_r at random, and try to expand all of the nodes in the tree in sequence in order of increasing cost $J^*(x_i, x_r)$ [i.e., starting from the closest node, using the measure of distance provided by $J^*(\cdot, x_r)$]; and 2) apply the optimal control policy $\pi(\cdot, x_r)$ until the system gets to x_r (or a neighborhood of x_r).

As it can be easily recognized, the preceding steps correspond to the steps in the RRT algorithm with two fundamental differences: at each iteration we try to connect the new candidate milestone in turn to each one of the nodes currently in the tree before discarding it as unreachable from the current tree (in the original RRT only the closest node was tested for reachability). The RRT criterion of testing the closest node translates into the heuristics of testing the nodes in ascending distance order. The second main difference is that the optimal cost function in the obstacle-free case is used as a measure of distance, both in the selection of nodes to expand and in the computation of the optimal control.

In the next subsections we will discuss more in detail the components of the motion-planning algorithm and how they fit together.

B. Data Structure

First of all, we will discuss briefly the elements that characterize each node (also referred to as milestone) and each edge (or branch) in the tree.

1. *Data Stored at Nodes*

The main piece of information stored for each milestone consists of the predicted (state, time) couple that will be attained if the chosen motion plan includes the node at hand; this can be computed by integrating over time the equations describing the dynamics of the system.

Moreover, at each node in the tree we can store data used to compute estimates (lower and upper bounds) on the total cost of motion plans including the node. The total cost of a trajectory [assuming the cost functional is additive, of the form (3)] can be split into an accumulated cost and a cost-to-go. The accumulated cost will in general be available through bookkeeping: it is a consequence of decisions made in the past. The cost-to-go is, in general, much more difficult to compute.

On the other hand, we can easily compute a lower bound on the cost-to-go: this is given by the value of the optimal cost in the absence of obstacles. The cost in the presence of obstacles cannot be lower than the obstacle-free cost because the presence of obstacles translates into additional constraints on the problem.

The upper bound on the cost-to-go is a priori unknown; as a matter of fact, we do not even know before starting the computations if the problem is feasible or not. As a consequence, we must initialize the upper bound on the cost-to-go to the symbolic value of infinity. However, if the optimal trajectory from a node to the target is collision free, then the corresponding cost clearly gives an upper bound on the cost-to-go. For a node that can be connected to the target by an optimal trajectory, the lower bound and upper bound coincide. Methods for updating the upper bound on the cost will be discussed in Sec. III.G.

As a final piece of information, for each node we store the total number of other nodes that are part of its offspring.

2. *Data Stored at Edges*

Whereas nodes in the tree represent states of the system along trajectories, edges can be thought of as representing decisions taken in the construction of the motion plan.

As a consequence, the main pieces of information to be stored at edges are the parameters identifying the control law implemented in the transition, namely, the next (randomly generated) equilibrium point x_r.

As a convenience for bookkeeping purposes, we can store the incremental cost incurred along the edge. In other words, this corresponds to the difference in the accumulated costs at the edge destination node and at the edge source node.

C. Initialization

The first step in the algorithm is the initialization of the tree. The first node, which will become the root of the tree, contains the initial conditions, projected at some point in the future, which depends on the amount of time that will be allocated to the computation of the motion plan. As an example, if the vehicle at the initial time t_0 is at position x_0, moving at constant velocity v_0, and we devote θ s to the computation of a motion plan, the root node must be set at $(x_0 + v_0\theta, t_0 + \theta)$.

The accumulated cost at the root node can be set to zero (all decision variables will have effect later in time, so there is no point with starting with a nonzero value). The lower bound on the cost-to-go can be computed as the value of the optimal cost function from the root state. To set the upper bound, we attempt to reach

the target state x_f using the optimal control law $\pi(\cdot, x_f)$. If the resulting trajectory is collision free, the optimal motion planning problem is evidently solved, and the algorithm can return with the optimal solution. Otherwise, the upper bound is set to the symbol ∞ (i.e., it is not yet known whether the motion planning problem admits a feasible solution or not), and the tree-building iteration is entered.

D. Tree Expansion Step

The main function to be performed at each step in the iteration is the expansion of the tree. The expansion is aimed at adding new nodes, or milestones, to the tree in such a way that the volume of the π-reachable space of the tree is increased rapidly.

At each iteration a new random target configuration x_r is generated from a uniform distribution. This requires that the motion-planning algorithm concentrates on a compact subset of the configuration space. This limitation can be the result of an obstacle avoidance constraint (such as in the cases in which the vehicle moves within an enclosed environment) or can be artificially imposed by limiting the distance the vehicle can stray from the initial and final points.

Given the new random candidate milestone x_r, we must check whether or not it is in the π-reachable set of the current tree. To do this, we apply the control policy $\pi(\cdot, x_r)$ starting from the (state, time) initial condition of each node in the tree, until a feasible, collision-free trajectory, which reaches x_r (or a neighborhood) at time t_r, is generated. If no such a trajectory is found, then x_r is not in the π-reachable set of the current tree and is discarded. Otherwise (and pending the safety check discussed in the next section), the new trajectory segment is added to the tree, as well as the new milestone (x_r, t_r).

It is easy to see that the function just outlined does indeed result in uniform sampling of the π-reachable set of the current tree (i.e., it is an implementation of an IDEAL-SAMPLE procedure on the symmetry group H). The new candidate milestone x_r is generated from a uniform distribution on H, and a thorough test is carried out to check whether or not it is in the π-reachable set of the current tree. The feedback control law $\pi(\cdot, x_r)$ is used as the inversion mechanism that provides the appropriate inputs at each instant in time.

The order in which the milestones currently in the tree are checked is as yet left unspecified. A random ordering is acceptable. However, some performance improvements are obtained in simulations (see Sec. V) if π reachability of the random candidate milestone x_r is tested from tree nodes in a more efficient way. As a matter of fact, in randomized motion-planning algorithms most of the time is typically spent in checking trajectories for collisions with obstacles, and methods are sought to reduce the number of collision checks.[58]

1. *Exploration Heuristics*

Before a feasible trajectory is found, the emphasis of the algorithm is on exploration, that is, on the addition on new milestones to the tree that enlarge its reachable set. To enhance exploration while keeping the number of collision checks to a minimum, it can be convenient to sort the nodes in ascending order of distance from x_r, where as a measure of distance we use the value of the optimal cost function in the obstacle-free case $J^*(\cdot, x_r)$. At the cost of the additional computation time required by the sorting of the nodes (and by the evaluation of distances), reachability is tested first from the nodes that are closer to the candidate milestone and hopefully more likely to provide a collision-free trajectory. This will result in a lower number of calls to the collision-checking routine (e.g., a routine that evaluates $G(x, t)$ along newly generated trajectories). This exploration heuristics is very closely related to the RRT algorithm, with the only difference that potentially every node in the tree is tested during the expansion step, whereas in the RRT algorithm only the closest node is tested.

2. *Optimization Heuristics*

Once a feasible trajectory has been found, the focus of the search shifts from exploration to the optimization of the computed trajectory. To enhance the quality of the new additions to the tree, we can sort the nodes in ascending order of total cost to reach x_r. This total cost consists of the accumulated cost up to the generic node (x_i, t_i) plus the cost to go from x_i to x_r, that is, $J^*(x_i, x_r)$. At the cost of the computation time required to sort the nodes, reachability is tested first from the nodes, which will provide the best trajectory to the new milestone. This will likely result in a better quality of the computed solution. This optimization heuristic is most appropriate when a feasible solution is already available.

As a last detail, every time a new node is added to the tree, the children counter of all the nodes on the path from the new node to the root of the tree must be increased by one. A pseudocode version of the tree expansion function is given as Algorithm 1.

Algorithm 1: Pseudocode for the function EXPAND-TREE (Tree)

```
1) Generate a random configuration x_r
2) sort the nodes in the tree according to the desired heuristics
   [random | exploration | optimization].
3) for all nodes in the tree (in the order established at the previous
   step) do
4)    Generate a trajectory to x_r, using the control policy π(·, x_r)
5)    Check the generated trajectory for obstacle avoidance
6)    if the trajectory is collision-free then
7)       return generated trajectory
8) return failure
```

E. Safety Check

Because obstacles are moving, checking for the absence of collision point-wise in time could not be enough. If we assume bounded accelerations, the reachable set of any collision-free point can be made arbitrarily small (e.g., a point a distance d in front of an obstacle moving with velocity v is collision free, but will not be such d/v seconds in the future). To ensure that the tree being built does not include such dead ends (i.e., milestones with a very small reachable set), before adding a new milestone to the tree we check for its safety over a time buffer τ. We will call τ safety the property of a milestone to be collision free over a time τ. Accordingly, we will say that a point (x_f, t_f) is (π, τ) reachable from (x_i, t_i) and belongs to the set $\mathcal{R}^\tau_\pi(x_i, t_i)$ if it is π reachable and τ safe.

The τ-safety check can be carried out on the predicted evolution of the system at the relative equilibrium corresponding to the new milestone. If possible, and practical, the time buffer τ can be extended to infinity. This is the case, for example, for static environments. Otherwise, τ should be chosen long enough that the successful computation of another motion plan is very likely. Assuming that the initial conditions are τ safe, the τ-safety check ensures that the same safety properties are maintained along the computed motion plan. In the case in which $\tau = \infty$, the τ safety check translates into hard safety guarantees for the whole motion plan. In cases for 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.

F. Improving Performance

The tree expansion step outlined in Sec. III.D. generates trajectories consisting of jumps from equilibrium point to equilibrium point—or, in the general case, from a relative equilibrium to the same relative equilibrium in a different location on the reduced configuration space $\mathcal{C}$—and as such is unlikely to provide satisfactory performance, in terms of the cost (3).

However, and building upon previous ideas,[55–57] 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 (which we will denote as a *primary* milestone). A *secondary* milestone is defined to be any state of the system along the path leading from the parent node in the tree to the newly added milestone. We can split any newly generated edge into $n > 1$ segments at random; the breakpoints will be the secondary milestones, and the endpoint is the primary milestone. Because the vehicle is in motion along the path, the secondary milestones are likely to be at points in the state space that are far from the equilibrium manifold. Secondary milestones are made available for

future tree expansion steps. All secondary milestones, by construction, have a τ-safe primary milestone in a child subtree and hence are themselves τ safe.

G. Update on the Cost-to-Go Estimates

Each time a new milestone is added to the tree, a check is made to see if the final target point is in its π-reachable set. In other words, we try applying the control policy $\pi(\cdot, x_f)$ to each newly generated milestone. In the case in which this results in a feasible, collision-free trajectory, we have found a sequence of milestones and control policies, which steers the system from the initial conditions to the final target point along a feasible, collision-free trajectory, and hence the feasibility problem is solved. However, we might be interested in looking for the best possible trajectory satisfying the feasibility problem and eventually try to approximate the solution of the optimal motion planning problem involving the minimization of the cost functional (3).

With this purpose in mind, every time a new feasible solution is found the estimates of the upper bound on the cost-to-go of the affected node in the tree are updated. The affected nodes are those in the path from the newly added node to the root. As a consequence, in order to update the upper bound on the costs we climb the tree backward toward the root: at each step we compare the old upper bound at the parent node with the upper bound on the cost following the newly found trajectory. If the latter is smaller that the old upper bound, we update the bound and reiterate from the parent. Otherwise the procedure stops (there is another subtree of the parent, which presents a lower upper bound on the cost). A pseudocode version of the cost estimate update function is given in Algorithm 2.

Algorithm 2: Pseudocode for the function UPDATE-COST-ESTIMATES (Tree, node, target)

```
1) node.LowerBound ← J*(node.x, target.x).
2) Generate the obstacle-free optimal trajectory to x_f, using the
   control policy π(·, target.x).
3) Check the generated trajectory for satisfaction of the obstacle
   avoidance constraints
4) if the trajectory is collision-free then
5)    node.UpperBound ← node.LowerBound.
6)    while node ≠ Tree.root and node.Parent.UpperBound >
      node.UpperBound + node.incomingEdge.cost do
7)       node.Parent.UpperBound ← node.UpperBound + node.
         incomingEdge.cost
8)       node ← node.Parent
9) return success
10) else
11)   node.UpperBound ← +∞.
12)   return failure
```

H. Tree Pruning

The upper and lower bounds on the cost-to-go stored for each tree milestone can be profitably used for pruning the tree and speeding up computations. Recall that 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 can be updated by climbing the tree backward along that feasible solution toward the tree root. While performing this operation, it is also possible to look at all of the children of the node being updated. If the lower bound on the total cost-to-go for such 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 that has just been found.

The end result of such a process is the removal from the trajectory tree of all of the provably bad candidates for the optimal solution. The trajectory tree, following this pruning process, contains 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.

I. Real-Time Considerations

A significant issue arising from the usage of randomized algorithms for path planning is the distinct possibility of driving the system toward a dead end as a result of finite computation times. The notion of τ safety was introduced in Sec. III.E. to prevent such situations to develop.

The τ safety of the generated plan derives from the fact that all of the primary milestones are by construction τ safe and all secondary milestones have at least one primary milestones in their subtree. Maintaining safety guarantees in the face of finite computation times is particularly important because the algorithm itself has no deterministic guarantees of success. In the sense just outlined 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, because 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:

1) At least one of the children leads to the destination through an already computed feasible solution. If there are more than one such feasible solutions, the solution with the least upper bound on the cost-to-go is chosen.

2) 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 subchildren of each tree. In the latter case the selected tree is likely to cover a bigger portion of the reachable set.

J. Complete Algorithm

The flow of the algorithm then proceeds as follows. After the initialization of the tree, a loop is entered until the target has been reached. As the very first step in each iteration of the loop, an attempt is made to join the current root (the initial conditions) to the target configuration using the optimal control law computed for the obstacle-free case. If the attempt is successful, an optimal solution has been found, and the algorithm terminates successfully.

Otherwise, an inner loop is entered, which tries to expand the trajectory tree. At each iteration of the inner loop, a random configuration is sampled, and an attempt is made through the EXPAND-TREE function to join a node in the current tree to the new candidate milestone. If this is possible and results in a τ-safe trajectory (i.e., the candidate milestone is in the (π, τ)-reachable set of the current tree), then the generated trajectory is split up into a number of segments, which are added to the tree as new edges, and the corresponding endpoints, which are added to the tree as new (primary or secondary) milestones.

From the newly generated milestones an attempt is made to reach the final target. If successful, this attempt enables the update of the upper bound estimates on the cost-to-go on the tree nodes. In this case the tree can be pruned to eliminate nodes in the tree, which are guaranteed to be bad candidates for further exploration (because they cannot improve on the quality of the solution already computed).

This procedure is perhaps better understood through an example, given in Fig. 1. The current tree is depicted as a set of thick lines, the squares represent the primary milestones, and the circles represent secondary milestones. The target point x_f is not reachable from any of the milestones currently in the tree by using the policy $\pi(\cdot, x_f)$ (i.e., the optimal policy to x_f assuming there are no obstacles). Thus, a new candidate milestone x_r is randomly generated, and then attempts are made to join the nodes of the tree to x_r. The closest milestone, in the sense induced by the cost-to-go function $J^*(\cdot, x_r)$, is indicated by the number one. Application of the policy $\pi(\cdot, x_r)$ with initial condition corresponding to this milestone results in a collision with one of the obstacles. The same can be said from the second closest milestone, indicated by the number two. With the third-closest milestone we have better luck, and we can generate a new collision-free trajectory. A new primary milestone

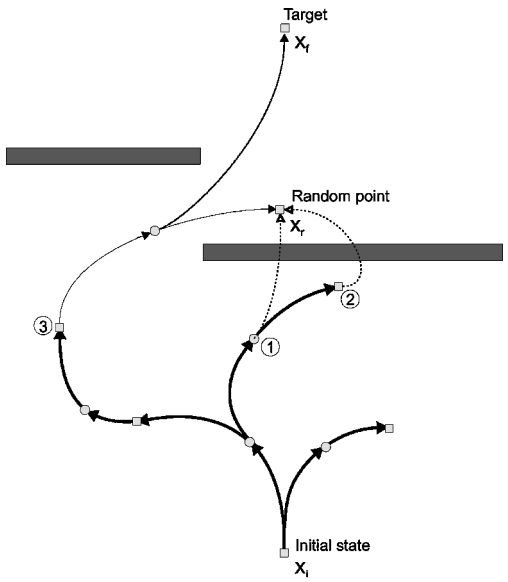

Fig. 1 Example of tree expansion procedure.

is created at x_r, a new secondary milestone is created at a random point on the newly generated trajectory, and the two new milestones are connected to the tree through edges encoding the new trajectory. Finally, we check whether or not the target point is reachable from the new milestones. In this case the answer is indeed positive, and the feasibility problem is solved. If this is not the case (or a better solution is sought), the iteration would be started again with a new random point x_r.

This iteration is carried out until the time allowed for computation has elapsed. This time is given by the time duration of the edge of the tree that is currently being executed, and is therefore not a priori fixed. The iteration can be preempted with no adverse consequence on the algorithm.

At this point if at least one feasible solution had been found, the subtree of the root corresponding to the least upper bound on the cost is chosen. If no feasible solution has been found yet, a subtree to descend is chosen randomly from a distribution weighted with the total number of milestones in each subtree. If the current root has no children, a new root is created, which corresponds to the predicted root state at some time in the future (this time interval being the computation time allotted for the next main loop iteration).

The rest of the tree is deleted and removed from memory because it contains only trajectories that can no longer be reached (it is not possible to go back in time). The main loop iteration is then repeated. A pseudocode version of the main algorithm is given in Algorithm 3.

Algorithm 3: Pseudocode for the motion-planning algorithm

1) Initialize Tree with the initial conditions at time $t_0 + \theta$.
2) **loop**
3) **if** UPDATE-COST-ESTIMATES (Tree, root, target) = success **then**
4) **Terminate** with success
5) **repeat**
6) newTrajectory = EXPAND-TREE (Tree)
7) **if** newTrajectory $\neq$ failure and newTrajectory is τ safe **then**
8) Split newTrajectory and generate primary and secondary milestones
9) **for all** new milestones **do**
10) UPDATE-COST-ESTIMATES (Tree, node, target)
11) Prune tree
12) **until** Time is up
13) **if** Tree.root.UpperBound $< \infty$ {Feasible solution found} **then**
14) find the child of root which gives the least upper bound on the cost-to-go
15) Tree.root $\leftarrow$ best child
16) **else if** Root has children **then**
17) Choose a child according to a random distribution, weighted with the number of children in each subtree
18) **else**
19) propagate root for a time θ in the future

IV. Analysis

This section aims at analyzing the behavior of the algorithm, proving probabilistic completeness and obtaining performance bounds. This requires additional definitions and assumptions about the environment characteristics. The remainder of the section presents the concepts supporting most of the available results about algorithms based on probabilistic roadmaps. New definitions are needed to adapt earlier results to the motion algorithm presented in this chapter, in particular to account for the closed-loop dynamics and to better characterize the assumptions about the dynamic environment.

A. Assumptions on the Environment

First of all, we require that for all τ-safe equilibrium points the (π, τ)-reachable set be not too small, taking into account also computational delays. This property corresponds to the ϵ goodness of a static workspace.[59] The planning environment (as defined by both the workspace and the control policy) is said to be (ϵ, τ) good if, for all sets $S_{\tau-\theta} \subset \mathcal{F}$ of $(\tau - \theta)$ safe equilibrium points, the following holds:

$$\mu\left[\mathcal{R}^{\tau}_{\pi}(S_{\tau-\theta})\right] \geq \epsilon$$

In the preceding $\mu(S)$ indicates the volume of the projection of the set $S \subset \mathcal{X} = \mathcal{C} \times \mathcal{Y}$ on $\mathcal{C}$. The volume measure is normalized in such a way that the volume of the compact subset of $\mathcal{X}$ defining the workspace is equal to one. The proof of algorithm performance relies upon the following properties, which will be assumed to be true for the system under consideration.

Let β be a constant in (0, 1]; define the β-lookout of $S \subset \mathcal{F}$ as

$$\beta\text{-lookout}(S) := \left\{ p \in S \middle| \mu\left(\mathcal{R}^{\tau}_{\pi}(p) \middle\backslash S\right) \geq \beta\mu[\mathcal{R}(S)\backslash S] \right\}$$

We require that the dimensions of the β lookout of any set should not be too small; this property is called (α, β) expansiveness[33]: Given two constants α, β in (0, 1], the environment is said (α, β) expansive if for all sets $S \in \mathcal{F}$ the following holds:

$$\mu[\beta\text{-lookout}(S)] \geq \alpha\mu(S)$$

B. Algorithm Performance

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 preceding section). Define the end-game region $E \subset H$ 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. The results that will be presented in the following are based on Hsu et al.,[37] with a few minor modifications to address the usage of the policy π in the exploration process, as opposed to random, piecewise constant inputs. The proofs of the lemmas are available in the reference and are omitted here for lack of space.

First of all, we need to characterize the expected number of lookout points in a tree. We have the following:

Lemma 1 (number of look-out points[37]): In a tree with r milestones, the probability of having k look-out points is at least $1 - k\exp(-\alpha\lfloor r/k \rfloor)$.

The following step is to quantify the size of the (π, τ) reachable set given the presence of at least k lookout points in the tree.

Lemma 2 (size of reachable set[37]): If a sequence of milestones contains k lookout points, the volume of its (π, τ) reachable set is at least $1 - \exp(-\beta k)$.

Finally, we can state the main theorem.

Theorem 1 (performance of the randomized algorithm[37]): A sequence of r (primary) milestones contains a milestone in the end-game region E with probability at least $1-\gamma$, if

$$r \geq (k/\alpha)\log(2k/\gamma) + [2/\mu(E)]\log(2/\gamma)$$

where $k := (1/\beta)\log[2/\mu(E)]$.

Proof: Split the sequence of milestones into two subsequences of r_1 and r_2 milestones, respectively. If the number of milestones in the first sequence is greater than $k = 1/\beta \log[2/\mu(E)]$, then the (π, τ) reachable set of the first sequence has an intersection of volume at least $\mu(E)/2$ with the end-game region E. Let us call event A the event that the number of lookout points is greater than k. This event occurs with a probability

$$\Pr(A) \geq 1 - \exp(-\alpha \lfloor r_1/k \rfloor)$$

If we want to make this probability at least $1-\gamma/2$, we need

$$r_1 \geq (k/\alpha)\log(2/\gamma)$$

Assume that the intersection of the (π, τ)-reachable set of the first sequence of milestones with the end-game region is $\mu(E)/2$. Then the probability that at least one of the milestones in the second sequence will be in the end-game region (event B) is

$$\Pr(B) \geq 1 - \exp[-r_2\mu(E)/2]$$

To make this probability at least $1-\gamma/2$, we need

$$r_2 \geq [2/\mu(E)]\log(2/\gamma)$$

If events A and B occur, then one of the milestones in the complete sequence is in the end-game region. We have that if $r \geq r_1 + r_2$

$$\Pr(A \wedge B) = \Pr(A)\Pr(B) \geq (1-\gamma/2)^2 \geq 1-\gamma$$

which proves the result. □

To our knowledge, the algorithm presented in this paper is the first one to which Theorem 1 fully applies, with the definitions and under the assumptions given earlier in this section, because the (π, τ)-reachable set is indeed uniformly sampled. In some sense the most significant contribution of this paper is to propose to shift the search for reachable milestones from an open-loop process, whereby 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 the ideal inversion mechanism considered in the literature.[37]

Moreover, we are able to recover, by sorting nodes according to distance, the levels of performance shown in practice by RRT algorithms. As a consequence, we are able to achieve probabilistic completeness and formal performance bounds in a dynamic environment, and at the same time we can exploit the rapid exploration capabilites of RRTs: in this sense the algorithm presented in Sec. III recovers all of the best properties of its predecessors.

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 because α, β, and ϵ cannot be computed easily for nontrivial environments. An interesting fact is that the computational complexity (in terms of convergence rate of the probability of correct termination) does not depend on the dimension of the state space or on the number of obstacle directly. Instead, it depends on parameters that in some sense quantify the geometric complexity of the environment.

Using secondary milestones does not adversely impact the main results on probabilistic completeness (when evaluated on the number of primary milestones) because they can only increase the (π, τ)-reachable set of the whole tree. In addition, secondary milestones have been found to help the convergence of the algorithm, when close to the end-game 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).

V. Application Examples

In the next sections we present three examples that show the power and the flexibility of the proposed algorithm. We consider first a linear system subject to saturation constraints, and then a system endowed with a control architecture patterned after the hybrid control structure described in Sec. II.A.2. All algorithms have been implemented in (soft) real time on a C++ on a Pentium II 300 MHz machine running Linux, using the LEDA library.[60] Comparisons will be presented on the computation times required by the following variations of incremental randomized planners:

Algorithm A: Only one node, chosen randomly from the current tree, is tested for expansion at each EXPAND-TREE step. This corresponds roughly to the algorithm proposed by Hsu et al.[37]

Algorithm B: Only one node, chosen as the closest one to the candidate milestone, is tested for expansion at each EXPAND-TREE step. This corresponds to the RRT algorithm,[35] with the difference that the obstacle-free optimal cost-to-go is used to compute distances.

Algorithm C: All nodes are tested in random order.

Algorithm D: All nodes are tested, in increasing order of distance. This corresponds to the full-fledged implementation of the algorithm in Sec. III.

The inputs are always chosen according to the optimal control policy in the obstacle-free case. Random control inputs are not appropriate for most of the dynamical systems we are considering because they could lead easily to instability (for example, a helicopter is an open-loop unstable system). If distances are measured according to the optimal cost function in the obstacle-free case, a greedy search for the control generation (as in the RRT algorithm) does provide a stabilizing control law. Moreover, in all cases any additional computation time available after the first feasible solution is found is used to optimize the solution.

The statistical data are referred to a data set of 1000 simulation runs for each example.

A. Ground Robot

In this section we are interested in minimum time motion planning for a planar system with (scaled) equations of motion

$$\ddot{x}_1 + \dot{x}_1 = u_1, \qquad \ddot{x}_2 + \dot{x}_2 = u_2 \tag{5}$$

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.[61,62] The following control law is adapted from the same references.

1. Minimum-Time, Minimum-Energy Control Law

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. The minimum time maneuver from origin to destination for each of the degrees of freedom (assuming a general maximum control intensity $u_{\max}$) is a bang-bang control law[24] given by

$$\begin{aligned} u(t) &= U \qquad \text{for} \qquad 0 < t < t_1 \\ u(t) &= -U \qquad \text{for} \qquad t_1 < t < t_1 + t_2 \end{aligned} \tag{6}$$

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

$$\Delta_0 := \begin{cases} x_0 - x_f + v_0 - u_{\max}\log(1 + v_0/u_{\max}) & \text{for} \quad v_0 \geq 0 \\ x_0 - x_f + v_0 + u_{\max}\log(1 - v_0/u_{\max}) & \text{for} \quad v_0 < 0 \end{cases} \tag{7}$$

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:

$$\begin{aligned} t_1 &= t_2 - C/U \\ t_2 &= \log\left[1 + \sqrt{1 - \exp(C/U)(1 - v_0/U)}\right] \end{aligned} \tag{8}$$

with $C = x_0 + v_0 - x_f$.

The policy π used to control the vehicle described by Eqs. (5) 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 Eqs. (6), with $U = \pm\gamma u_{\max}$, and by iterating over the parameter $\gamma \in (0, 1)$ until $t_{\min} = t^*_{\min}$.

2. Fixed and Moving Spheres

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. In the tests both fixed spheres and spheres moving at a constant random speed were considered, with no significant statistical difference in the data sets. This example was very easily handled by the randomized algorithms. A feasible solution was found by all algorithms in less than 20 ms in 50% of the cases, with no significant differences in performance. A feasible solution was found by all algorithms in all test cases. A summary of the average time required to find the first feasible solution t_{feas} and its standard deviation is reported in Table 1.

The cost of the trajectory that is eventually executed has been analyzed. In the examples, the lower bound on the cost is 11.39 s (this is the cost of an optimal trajectory in the obstacle-free case; there does not exist a unique optimal trajectory, but many trajectories minimize the cost). In all cases but for Algorithm 2, an optimal trajectory was found in at least 50% of the test cases. On average, the cost of the solution was less than 5% higher than the lower bound, showing that the algorithms do indeed provide a solution that is almost optimal. In this scenario, Algorithms A and C, relying completely on randomization (as opposed to RRT-like heuristics), seemed to fare slightly better than the others. A summary of the average cost of the solution t_f provided by the algorithms and its standard deviation is also reported in Table 1.

In this case the environment is open, and all of the randomized algorithms can achieve extremely high levels of performance, even in the presence of moving obstacles.

3. Sliding Doors

In the second example the robot must go through moving openings in two walls. The walls are 40 m apart and have "doors" 10 m wide (Fig. 2). Both the walls slide along their longitudinal axis according to a harmonic law. The frequencies of the two oscillations are 0.5 rad/s for the wall at the bottom and 0.25 rad/s for the wall at the top of the picture. The amplitude of the harmonic motion in both the cases is 40 m, yielding a maximum door velocity of 20 m/s, twice as much as the maximum velocity of the robot.

This scenario was expected to be a difficult challenge for the randomized planners; however, most of the algorithms were able to deal with it quite effectively (see Table 2). Algorithm A was slow in finding a feasible solution (it took an average time of over 40 s), but the quality of the resulting plans was consistently good. Algorithms C and D were again the best performers, finding a feasible trajectory within a few seconds. Moreover, the average cost of the plans generated by Algorithm C is within 22% of the lower bound on the optimal cost.

The reason for the poor performance of Algorithm A in this case is easily seen from Fig. 2: this algorithm suffers from the lack of efficient exploration and finding the narrow passage between the moving doors requires on average much more samples than using the other algorithms. In the figures the heavy line represents the trajectory that is eventually executed (i.e., the best computed solution), whereas the lighter lines represent the other trajectories (edges) in the trajectory tree. The squares represent primary milestones, and the circles represent secondary milestones. Milestones that can be connected to the target by the obstacle-free optimal control law are filled in a darker shade.

B. Small Autonomous Helicopter

One of the prime motivations for the development of the algorithm presented in this chapter 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 nonlinear helicopter simulation, based on the model presented in Refs. 40 and 41 and in the references therein. The motion-planning algorithms operating on the nominal maneuver automaton structure discussed in Sec. II.A.2 are complemented by a nonlinear tracking control law[41] to ensure tracking of the computed trajectory.

Table 1 Ground robot moving amidst fixed spheres: simulation results

Algorithm	Average, (t_{feas}), ms	St. dev. (t_{feas}), ms	Average (t_f), s	St. dev. (t_f), s
A	17.68	10.10	11.52	0.29
B	16.34	7.93	12.03	0.68
C	18.12	10.01	11.55	0.31
D	17.3	8.57	11.86	0.55

Table 2 Ground robot moving through sliding doors: simulation results

Algorithm	Average (t_{feas}), s	St. dev. (t_{feas}), s	Average (t_f), s	St. dev. (t_f), s
A	42.67	27.25	14.95	6.47
B	14.53	25.68	18.78	5.96
C	3.12	13.16	13.87	5.54
D	1.29	8.43	17.35	5.88

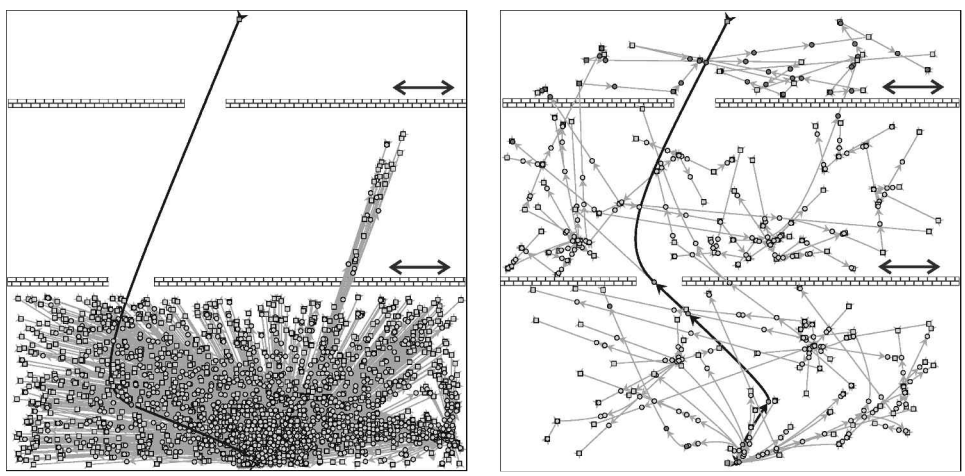

Fig. 2 Ground robot moving through sliding doors: trajectory tree traces. Algorithm A (left) and algorithm D (right).

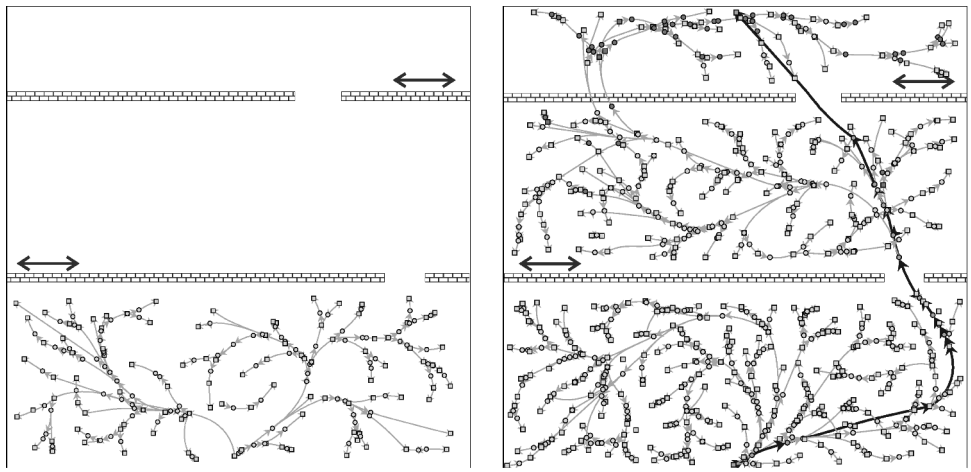

Fig. 3 Helicopter flying through sliding doors: trajectory tree traces. Algorithm B (left) and algorithm D (right). The RRT-like algorithm on the left can get "locked" by the first computed trajectories.

Table 3 Helicopter moving amidst fixed spheres: simulation results

Algorithm	Average (t_{feas}), s	St. dev. (t_{feas}), s	Average (t_f), s	St. dev. (t_f), s
A	0.365	0.390	16.44	2.30
B	0.117	0.060	18.36	3.59
C	0.181	0.124	17.05	2.58
D	0.183	0.132	17.26	2.42

Table 4 Helicopter flying through sliding doors: simulation results (If the computation times exceeded the allotted time of 120 s, the value of 120 s was taken into account in the statistics.)

Algorithm	Average (t_{feas}), s	St. dev. (t_{feas}), s	Average (t_f), s	St. dev. (t_f), s
A	——	——	——	——
B	108.26	35.58	118.04	28.78
C	48.89	42.33	64.83	25.79
D	52.44	38.51	68.18	32.59

The trajectory primitive library was built by choosing trim trajectories in level flight, with no sideslip, and flying at the following velocities: {0, 1.25, 2.5, 5, 10} m/s with the following heading turning rates: $\{-1, -0.5, 0, 0.5, 1\}$ rad/s. This gives a total of 25 trim trajectories. Time-optimal maneuvers were generated, connecting each of the trim trajectories to all of the others, giving a total of 600 maneuvers.

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 of the examples is the total time needed to go to the destination. The initial conditions are at hover heading due East (i.e., to the right in the pictures).

1. Fixed and Moving Spheres

The first example involves navigating the helicopter through a set of fixed spheres. As in the case of the ground robot, this example was very easily handled by the proposed planners: the average time required for computation of a feasible solution is of the order of a few tenths of a second (see Table 3).

2. Sliding Doors

In the last example the helicopter must go through moving openings in two walls (see Fig. 3). This scenario proved to be very challenging for the randomized planners. Algorithm A failed to find a feasible solution within two minutes in the totality of the simulations runs (1000). Algorithm B did better, finding a feasible solution within two minutes in 10% of the cases (in which the feasible trajectory was found extremely quickly, in about 2.5 s). Algorithms C and D on the other hand always succeeded in finding a solution, even though it took about 50 s on average. The total cost of the solutions found was slightly above one minute on average (see Table 4).

C. Discussion of the Results

From the analysis of the simulation results, both in the ground robot example and in the helicopter example, we can conclude that all of the tested algorithms perform equally well in simple cases with very open environments. As the difficulty of the test scenarios increases, it is apparent that Algorithms C and D (two versions of the algorithm proposed in this chapter) performed better than Algorithms A and B, both in terms of computation time required to solve the feasibility problem and in terms of the quality of the computed solution.

Although Algorithm A does have probabilistic completeness guarantees in the case of dynamic environments (as opposed to Algorithm B, which does not), it suffers from inefficient exploration, as it can easily be recognized from Fig. 2.

Algorithm B, although performing well in static environments, shows its limits in the case of dynamic environments (in fact, it is not probabilistically complete in this case). This algorithm is characterized by a very efficient exploration of the workspace: however, it tends to get locked in the first computed trajectories, which, even though close to the objective, could not result in feasible trajectories to the target (Fig. 3). It is believed that this is the reason of the behavior noted in the case of the helicopter flying through sliding doors.

Algorithms C and D provided the best results among the algorithms tested. Even though the performance characteristics are very similar, Algorithm D seems to find the first feasible solution in a shorter time than C. The reason for this can be found in the fact that by sorting the nodes in order of increasing distance from the candidate milestone the efficient exploration characteristics of RRTs are recovered, while ensuring probabilistic completeness and exponential convergence to one of the probability of correct execution even in a dynamic environment. On the other hand, Algorithm C generally resulted in solutions with a lower cost: this can be seen as another effect of the fact that RRT-like algorithms tend to get locked in the first solutions they compute and do not explore other options.

VI. Conclusions

In this paper a randomized motion-planning algorithm was presented, based on using 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. The resulting algorithm has been shown to be flexible enough to handle a broad variety of dynamical systems,

including systems described by ordinary differential equations, as well as hybrid systems. In a related work by the authors, the same algorithm has been applied to dynamical systems evolving on manifolds, as in the case of attitude motion planning for spacecraft.[63]

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 environments, with limited sensor range, and multivehicle operations.

Acknowledgments

This research was supported by the C. S. Draper Laboratory through the IR&D Grant DL-H-505334, by the U.S. Air Force Office of Scientific Research under Grant F49620-99-1-0320, and by the Office of Naval Research under Young Investigator Award N-00014-99-1-0668. The authors would like to thank the anonymous reviewers for many valuable comments and suggestions.

References

[1]Latombe, J.-C., "Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts," *International Journal of Robotics Research*, Vol. 18, No. 11, 1999, pp. 1119–1128.

[2]Latombe, J.-C., *Robot Motion Planning*, Kluwer, Dordrecht, The Netherlands, 1991, Chaps. 1–4.

[3]Li, Z., and Canny, J. (eds.), *Nonholonomic Motion Planning*, Kluwer Academic, Norwell, MA, 1993, Chaps. 1–11.

[4]Laumond, J.-P. (ed.), *Robot Motion Planning and Control*, Lecture Notes in Control and Information Sciences, Vol. 229, Springer-Verlag, London, 1998.

[5]Lozano-Pérez, T., "Automaton Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 11, No. 10, 1981, pp. 681–698.

[6]Lozano-Pérez, T., and Wesley, M., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, Vol. 22, No. 10, 1979, pp. 560–570.

[7]Edelsbrunner, H., *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987, pp. 275–278.

[8]O'Dunlaing, P., and Yap, C., "A Retraction Method for Planning the Motion of a Disc," *Journal of Algorithms*, Vol. 6, No. 1, 1982, pp. 104–111.

[9]Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, Vol. 5, No. 1, 1986, pp. 90–98.

[10]Hwang, Y., and Ahuja, N., "A Potential Field Approach to Path Planning," *IEEE Transactions of Robotics and Automation*, Vol. 8, No. 1, 1992, pp. 23–32.

[11]Barraquand, J., Langlois, B., and Latombe, J., "Numerical Potential Field Techniques for Robot Path Planning," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 22, No. 2, 1992, pp. 224–241.

[12]Rimon, E., and Koditscheck, D., "Exact Robot Navigation Using Artificial Potential Fields," *IEEE Trasactions on Robotics and Automation*, Vol. 8, No. 5, 1992, pp. 501–518.

[13]Reif, J., "Complexity of the Mover's Problem and Generalizations," *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, Inst. of Electrical and Electronics Engineers, New York, 1979, pp. 421–427.

[14]Sedgewick, P., *Algorithms*, 2nd ed., Addison Wesley Longman, Reading, MA, 1988, Chap. 45.

[15]Schwarz, J., and Sharir, M., "On the Piano Mover's Problem: I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," *Communications on Pure and Applied Mathematics*, Vol. 36, No. 1, 1983, pp. 345–398.

[16]Canny, J., *The Complexity of Robot Motion Planning: ACM Doctoral Dissertation Award*, Massachusetts Inst. of Technology Press, Cambridge, MA, 1988.

[17]Fliess, M., Lévine, J., Martin, P., and Rouchon, P., "Flatness and Defect of Nonlinear Systems: Introductory Theory and Examples," *International Journal of Control*, Vol. 61, No. 6, 1995, pp. 1327–1361.

[18]van Nieuwstadt, M. J., and Murray, R. M., "Real-Time Trajectory Generation for Differentially Flat Systems," *International Journal of Robust and Nonlinear Control*, Vol. 8, No. 11, 1998, pp. 995–1020.

[19]Faiz, N., Agrawal, S., and Murray, R., "Trajectory Planning of Differentially Flat Systems with Dynamics and Inequalities," *Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 2, 2001, pp. 219–227.

[20]Milam, M. B., Mushambi, K., and Murray, R., "A Computational Approach to Real-Time Trajectory Generation for Constrained Mechanical Systems," *Proceedings of the 34th IEEE Conference on Decision and Control*, Inst. of Electrical and Electronics Engineers, New York, 2000, pp. 845–851.

[21]Laumond, J.-P., "Singularities and Topological Aspects in Nonholonomic Motion Planning," *Nonholonomic Motion Planning*, edited by Z. Li and J. Canny, Kluwer Academic, Norwell, MA, 1993, pp. 149–200.

[22]Laferriere, G., and Sussmann, H. J., "A Differential Geometric Approach to Motion Planning," *Nonholonomic Motion Planning*, edited by Z. Li and J. Canny, Kluwer Academic, Norwell, MA, 1993, pp. 235–270.

[23]Bullo, F., and Lynch, K. M., "Kinematic Controllability for Decoupled Trajectory Planning in Underactuated Mechanical Systems," *IEEE Transactions on Robotics and Automation*, Vol. 17, No. 4, 2001, pp. 402–412.

[24]Bryson, A. E., and Ho, Y. C., *Applied Optimal Control*, Hemisphere, New York, 1975, Chaps. 2, 3.

[25]Athans, M., and Falb, P., *Optimal Control*, McGraw–Hill, New York, 1966, Chaps. 1–10.

[26]Pontryagin, L., Boltanskii, V., Gramkrelidze, R., and Mischenko, E., *The Mathematical Theory of Optimal Processes*, Interscience, New York, 1962, Chaps. 1, 2.

[27]Bellman, R., *Dynamic Programming*, Princeton Univ. Press, Princeton, NJ, 1957, Chap. 3.

[28]Kavraki, L. E., Kolountzakis, M. N., and Latombe, J., "Analysis of Probabilistic Roadmaps for Path Planning," *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Vol. 4, Inst. of Electrical and Electronics Engineers, New York, 1996, pp. 3020–3025.

[29]Overmars, M. H., and Svestka, P., "A Probabilistic Learning Approach to Motion Planning," TR UU-CS-1994-03, Dept. of Computer Science, Utrecht Univ., Utrecht, The Netherlands, Jan. 1994.

[30]Kavraki, L., Svestka, P., Latombe, J., and Overmars, M., "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, Vol. 12, No. 4, 1996, pp. 566–580.

[31]Kavraki, L. E., Latombe, J., Motwani, R., and Raghavan, P., "Randomized Query Processing in Robot Path Planning," *Journal of Computer and System Sciences*, Vol. 57, No. 1, 1998, pp. 50–60.

[32]Hsu, D., Kavraki, L., Latombe, J., Motwani, R., and Sorkin, S., "On Finding Narrow Passages with Probabilistic Roadmap Planners," *Robotics: The Algorithmic Perspective*, edited by P. K. Agarwal, L. Kavraki, M. Mason, and A. K. Peters, Natick, MA, 1998, pp. 141–153.

[33]Hsu, D., Latombe, J.-C., and Motwani, R., "Path Planning in Expansive Configuration Spaces," *International Journal of Computational Geometry and Applications*, Vol. 9, No. 4–5, 1999, pp. 495–512.

[34]LaValle, S. M., "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Iowa State Univ., TR 98-11, Ames, IA, Oct. 1998.

[35]LaValle, S., and Kuffner, J., "Randomized Kinodynamic Planning," *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, Vol. 1, Inst. of Electrical and Electronics Engineers, New York, 1999, pp. 473–479.

[36]Kuffner, J., and LaValle, S., "RRT-Connect: An Efficient Approach to Single-Query Path Planning," *IEEE International Conference on Robotics and Automation*, Vol. 2, Inst. of Electrical and Electronics Engineers, New York, 2000, pp. 995–1001.

[37]Hsu, D., Kindel, R., Latombe, J.-C., and Rock, S., "Randomized Kinodynamic Motion Planning with Moving Obstacles," *Algorithmic and Computational Robotics: New Directions*, edited by B. Donald, K. Lynch, and D. Rus, A. K. Peters, Boston, 2001, pp. 247–264.

[38]Barraquand, J., and Latombe, J., "Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles," *Algorithmica*, Vol. 10, No. 2–4, 1993, pp. 121–155.

[39]Arnold, V., *Mathematical Methods of Classical Mechanics*, 2nd ed., GTM, Vol. 60, Springer-Verlag, New York, 1989, Chap. 3.

[40]Koo, T., and Sastry, S., "Output Tracking Control Design of a Helicopter Model Based on Approximate Linearization," *Proceedings of the IEEE Conference on Decision and Control*, Vol. 4, Inst. of Electrical and Electronics Engineers, New York, 1998, pp. 3635–3640.

[41]Frazzoli, E., Dahleh, M., and Feron, E., "Trajectory Tracking Control Design for Autonomous Helicopters Using a Backstepping Algorithm," *American Control Conference*, Vol. 6, Inst. of Electrical and Electronics Engineers, New York, 2000, pp. 4102–4107.

[42]Frazzoli, E., Dahleh, M., and Feron, E., "A Hybrid Control Architecture for Aggressive Maneuvering of Autonomous Helicopters," *IEEE Conference on Decision and Control*, Vol. 3, Inst. of Electrical and Electronics Engineers, New York, 1999, pp. 2471–2476.

[43]Frazzoli, E., Dahleh, M., and Feron, E., "Robust Hybrid Control for Autonomous Vehicle Motion Planning," *IEEE Conference on Decision and Control*, Vol. 1, Inst. of Electrical and Electronics Engineers, New York, 2000, pp. 821–826.

[44]Frazzoli, E., "Robust Hybrid Control for Autonomous Vehicle Motion Planning," Ph.D. Dissertation, Dept. of Aeronautics and Astronautics, Massachusetts Inst. of Technology, Cambridge, MA, June 2001.

[45]Shaw, R. L., *Fighter Combat: Tactics and Maneuvering*, Naval Inst. Press, Annapolis, MD, 1985, Chaps. 2–4.

[46]Austin, F., Carbone, G., Falco, M., Hinz, H., and Lewis, M., "Game Theory for Automated Maneuvering During Air-to-Air Combat," *Journal of Guidance, Control, and Dynamics*, Vol. 13, No. 6, 1990, pp. 1143–1149.

[47]Smith, R., and Dike, B., "Learning Novel Fighter Combat Maneuver Rules via Genetic Algorithms," *International Journal of Expert Systems*, Vol. 8, No. 3, 1995, pp. 247–276.

[48]Thomson, D., and Bradley, R., "Mathematical Definition of Helicopter Maneuvers," *Journal of the American Helicopter Society*, Vol. 42, No. 4, 1997, pp. 307–309.

[49]Boyle, D. P., and Chamitoff, G., "Autonomous Maneuver Tracking for Self-Piloted Vehicles," *Journal of Guidance, Control, and Dynamics*, Vol. 22, No. 1, 1999, pp. 58–67.

[50]Bertsekas, D. P., and Tsitsiklis, J., *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996, Chap. 6.

[51]Dubins, L., "On Curves of Minimal Length with a Constraint on Average Curvature and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, Vol. 79, No. 1, 1957, pp. 497–516.

[52]Reeds, J., and Shepp, R., "Optimal Paths for a Car That Goes Both Forwards and Backwards," *Pacific Journal of Mathematics*, Vol. 145, No. 2, 1990, pp. 367–393.

[53]Soueres, P., and Boissonnat, J., "Optimal Trajectories for Nonholonomic Mobile Robots," *Robot Motion Planning and Control*, edited by J. Laumond, Lecture Notes in Control and Information Sciences, Vol. 229, Springer-Verlag, London, 1998, pp. 93–170.

[54]Bertsekas, D., *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA, 1995, Chap. 1.

[55]Leonessa, A., Chellaboina, V., and Haddad, W. M., "Globally Stabilizing Controllers for Multi-Mode Axial Flow Compressors via Equilibria-Dependent Lyapunov Functions," *Proceedings of the 1997 American Control Conference*, Vol. 2, Inst. of Electrical and Electronics Engineers, New York, 1997, pp. 993–997.

[56]Burridge, R. R., Rizzi, A. A., and Koditscheck, D. E., "Sequential Decomposition of Dynamically Dexterous Robot Behaviors," *International Journal of Robotics Research*, Vol. 18, No. 6, 1999, pp. 534–555.

[57]McConley, M., Appleby, B., Dahleh, M., and Feron, E., "A Computationally Efficient Lyapunov-Based Scheduling Procedure for Control of Nonlinear Systems with Stability Guarantees," *IEEE Transactions on Automatic Control*, Vol. 45, No. 1, 2000, pp. 33–49.

[58]Bohlin, R., and Kavraki, L. E., "A Randomized Algorithm for Robot Path Planning Based on Lazy Evaluation," *Handbook on Randomized Computing*, edited by P. Pardalos, S. Rajasekaran, and J. Rolim, Kluwer Academic, Norwell, MA, 2000.

[59]Kavraki, L., and Latombe, J., "Probabilistic Roadmaps for Robot Path Planning," *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, edited by K. Gupta and A. del Pobil, Wiley, New York, 1998, pp. 33–53.

[60]Mehlhorn, K., and Naher, S., *The LEDA Platform of Combinatorial and Geometric Computing*, Cambridge Univ. Press, Cambridge, England, U.K., 1999, Chaps. 1–3, 5, 6, 11, 12.

[61]D'Andrea, R., Kalmár-Nagy, T., Ganguly, P., and Babish, M., "The Cornell Robot Soccer Team," *RoboCup-00: Robot Soccer World Cup IV*, edited by P. Stone, Lecture Notes in Computer Science, Springer-Verlag (to be published).

[62]Kalmár-Nagy, T., Ganguly, P., and D'Andrea, R., "Real-Time, Near-Optimal Trajectory Control of an Omni-Directional Vehicle," *Proceedings of the ASME IMECE'01 Symposium on Modeling, Control and Diagnostics of Large-Scale Systems* (to be published).

[63]Frazzoli, E., Dahleh, M., Feron, E., and Kornfeld, R., "A Randomized Attitude Slew Planning Algorithm for Autonomous Spacecraft," AIAA Paper 2001-4155, Aug. 2001.