R

ROBOT MOTION PLANNING

INTRODUCTION

The aim in robot motion planning is to be able to specify a task in a high-level, expressive language and have the robot(s) automatically convert the specification into a set of low-level primitives, such as feedback controllers and communication protocols, to accomplish the task (1,2). The robots can vary from manipulator arms used in manufacturing or surgery, to autonomous vehicles used in search and rescue or in planetary exploration, and to smart wheelchairs for disabled people. They are subject to mechanical constraints (e.g., a car-like robot cannot move sideways and an airplane cannot stop in place) and have limited computation, sensing, and communication capabilities. The environments can be cluttered with possibly moving and shape-changing obstacles and can contain dynamic (moving, appearing, or disappearing) targets. The challenge in this area is the development of a computationally efficient framework accommodating both the robot constraints and the complexity of the environment, while allowing for a large spectrum of task specifications.

A robot combines moving mechanical pieces such as wheels, gears, and breaks with digital devices such as processors and sensing and communication devices, in continuous interaction with a possibly changing environment. Therefore, motion planning is a highly interdisciplinary area, combining tools from computer science, mechanics, control theory, and differential geometry. Given the variety of applications, many motion planning approaches have been developed over the years. Depending on the task they address, motion planning problems can roughly be divided into four main groups: *navigation, coverage, mapping, and localization*(3).

In *navigation,* the problem is to find a collision-free motion between two configurations of the robot. *Coverage* is the problem of moving a robot sensor or end effector in such as way that it reaches all points in a target space (e.g., painting a surface). *Mapping* involves exploring an environment with the goal of producing a representation that can be used, for example, in navigation and coverage. Finally, in *localization,* the problem is to use a map and sensor data to determine the configuration (state) of the robot. Localization and mapping are sometimes performed simultaneously, such as in simultaneous localization and mapping (SLAM)(4).

Motion planners also differ depending on the robot model they consider. For example, it is much easier to plan the motion of a robot that is free to move instantaneously in all directions of its configuration space (omnidirectional robot), rather than generating motion for a car-like or an airplane-like vehicle that cannot move sideways (i.e., nonholonomic robot; see Ref. 5 for a collection of motion planning approaches for such robots). Motion planning can be performed for kinematic robot models, which capture only the configuration and velocity of the robot, or for dynamic robot models, which capture forces and accelerations.

Motion planning approaches can also be classified depending on the properties of the underlying algorithms. A motion plan is optimal if the produced motion minimizes energy consumption, execution time, trajectory length, and so on. Computational complexity is also a determining factor. For example, in most cases, it is desired that the amount of necessary memory and running time scale polynomially with the size of the input of the planner, which can be the number of obstacles, the number of degrees of freedom of the robot, and so on. Finally, a planner is complete if it always finds a path if one exists. Others are resolution complete, if a solution is found whenever one exists at a given discretization resolution, or probabilistic complete, if the probability of finding a solution during an iterative discretization process converges to 1 when the solution exists.

WORKSPACE AND CONFIGURATION SPACE

Given a robotic system, a *configuration* is a complete description that determines the position of every point of the system uniquely. Its *configuration space,* called for simplicity C-space, is the set of all possible configurations of the system. The number of degrees of freedom of a robotic system is the dimension of its minimal configuration space or, in other words, the minimum number of parameters needed to describe the system completely. The space in which the robotic system does work is called the *workspace*, which can be seen as the Euclidean space $\mathbb{R}^2$ or $\mathbb{R}^3$, depending on whether the motion is in plane or space. Most often, however, the workspace is defined more precisely as the subset of $\mathbb{R}^2$ or $\mathbb{R}^3$ that can be reached by a point of interest on the robot, such as the end effector of a manipulator arm.

Consider, for example, a two-joint planar robot arm, where a point on the first link is pinned to the ground, and the base of the second link is pinned to the end of the first, such that the only possible motion of the second link is a rotation about the (revolute) joint [Fig. 1(a)]. If we denote by θ_1 and θ_2 the angles formed by the two links with the horizontal, then (θ_1, θ_2) can be seen as coordinates of the configuration space, which is $S^1 \times S^1 = T^2$, where S^1 and T^2 denote the unit circle and the torus, respectively [Fig. 1(c)]. The workspace of this robot, however, is an annulus, with the outer radius determined by the sum of the lengths of the two links, and the inner radius is given by the difference between their lengths [Fig. 1(b)].

The configuration space of a planar square robot (or any other rigid body) that can only translate without rotation (see Fig. 2) is $\mathbb{R}^2$. For a planar robot that can only rotate about a fixed point, the configuration space is $SO(2)$, called the Special Orthogonal group in the plane, which is

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

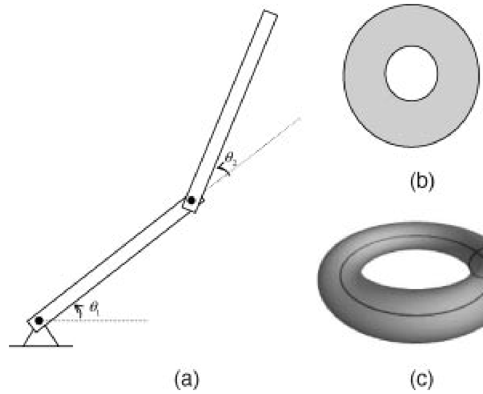


Figure 1. A two-link manipulator (a), its workspace (b), and its configuration space (c).

isomorphic to S^1. The configuration space of a robot allowed to translate and rotate in the plane is $SE(2)$, called the Special Euclidean group in the plane, and defined as $SE(2) = \mathbb{R}^2 \times SO(2)$. In space, a rotating and translating robot modeled as a rigid body evolves in $SE(3) = \mathbb{R}^3 \times SO(3)$, where $SO(3)$ is the group of rotations in space (3).

If obstacles are present in the workspace, it is useful to define explicitly the set of robot configurations for which collisions occur. For each obstacle in the workspace, the *configuration space obstacle* is defined as the set of all configurations at which the robot intersects the obstacle in the workspace. The free configuration space, also called free C-space, is the set of configurations at which the robot does not intersect any obstacle. Figure 2(b) shows the free C-space for the square robot moving in the environment shown in Fig. 2(a), if only translational motion is allowed (no rotation). The reader is referred to Ref. 3, p. 509 for an example of the free C-space construction for a polytopal robot translating and rotating in a polytopal environment with polytopal obstacles. In this setup, a navigation problem, as defined above, translates to finding a continuous curve between the initial and the final configuration in the free C-space.

RIGID BODY MOTION INTERPOLATION

A navigation problem for a robot, represented simply as a rigid body moving in space or plane with no obstacles, is also called rigid body motion interpolation. In the configuration space of the robot, which is $SE(3)$ or $SE(2)$ as defined above, this problem translates to generating a (possibly smooth) curve interpolating between two points. The problem of

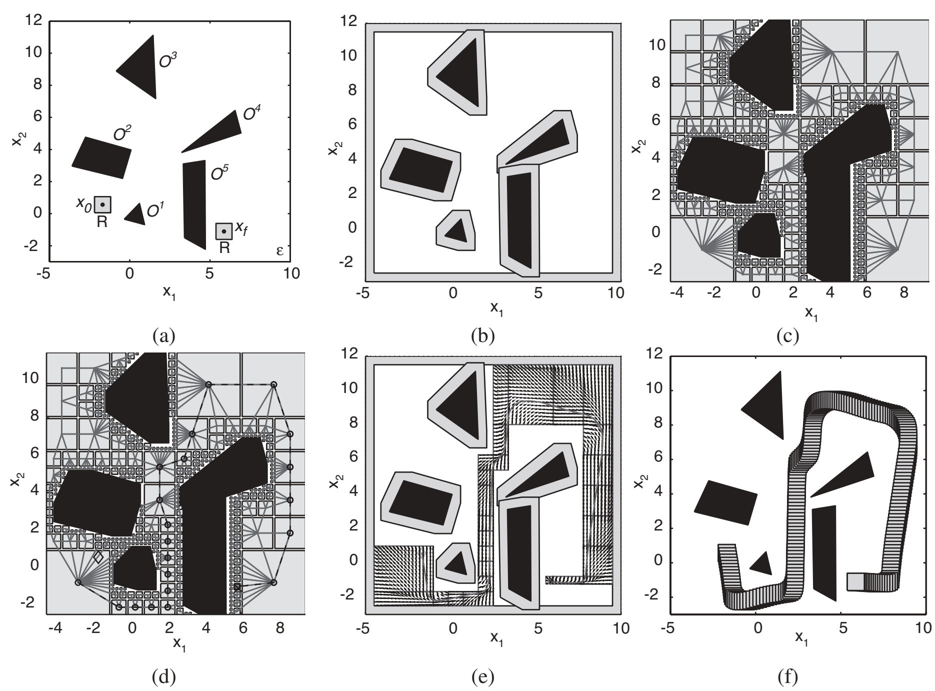


Figure 2. Cell decomposition and simultaneous planning and control for a square robot translating (no rotation) in a 2-D rectangular environment with polyhedral obstacles. The observable is the centroid of the robot: (a) initial (left) and final (right) positions of the robot. (b) The free C-space is obtained by enlarging the obstacles, shrinking the environment boundaries, and reducing the robot to its observable point. (c) Rectangular (quadtree) partition of the free C-space and the quotient graph of the partition. (d) Optimal path from initial to final node (rectangle) in the quotient graph; $\diamond$ and $\times$ denote the initial and final position of the robot observable, respectively. (e) Vector field assignment (used in simultaneous planning and control) and the resulting trajectory of the observable. (f) Robot motion in the initial environment.

finding a smooth interpolating curve is well understood in Euclidean spaces (e.g., a line segment is a smooth interpolating curve between two points), but it is not easy to generalize such techniques to curved spaces, such as $SE(3)$ and $SE(2)$. Most work in this area proposes to generalize the notion of interpolation from the Euclidean space to a curved space. For example, in Ref. 6, Bezier curves are used for interpolating rotations based on a spherical analog of the well-known de Casteljau algorithm. Other examples include spatial rational B-splines and Hermite interpolation (see Ref. 7 for an overview).

The above methods find immediate applications in computer graphics (e.g., generate a "natural" motion for an object thrown from one place to another). However, to generate a robot motion plan, two more issues have to be taken into consideration: optimality of the trajectory and invariance with respect to the choice of a world frame. The optimality requirement is particularly relevant in applications such as deep space formations. For example, to achieve interferometry, a group of satellites is required to maintain a rigid body formation. A reconfiguration demands a fuel-optimal trajectory to preserve mission life and is constrained by the limited thrust available.

Coordinate-free approaches leading to trajectories that are invariant to reference frames exist for the generation of shortest paths and minimum acceleration trajectories on $SO(3)$ (the set of all rotations in $\mathbb{R}^3$) and $SE(3)$ (the set of all poses in $\mathbb{R}^3$) (see Ref. 8 for an overview) (see Fig. 3 for examples). However, analytical solutions are available only in the simplest cases, and the procedure for solving optimal motions, in general, is computationally intensive. A relaxation based on the generation of optimal curves in an embedding Euclidean space and subsequent projection, which leads to suboptimal solutions, is proposed in Ref. 7.

POTENTIAL-BASED PLANNERS

Potential-based motion planners are based on the idea that a robot configuration can be driven to a desired value in the same way in which a particle moves in a force field. More precisely, a potential ϕ is a differentiable, real-valued function defined on the configuration space. Its gradient $\nabla\phi$ is a vector that points in the direction of maximum (local) increase of ϕ. This gradient can be used to define a vector field, (i.e., an assignment of a vector to each point of the configuration space). Guiding a robot through an environment with obstacles toward a goal can, therefore, be achieved by constructing a potential function in the configuration space, with a minimum value at the goal and high values at the obstacles, and by setting the velocity of the robot configuration equal to the negative of the gradient of the potential. Such an approach can be used directly to accommodate kinematic robot models. However, this approach can be extended for dynamic models, by designing control laws guaranteeing the convergence of the velocities to a desired vector field (9).

The robot motion terminates at a point of zero velocity or, equivalently, at a point where the gradient of the potential function vanishes, which is a critical point for the potential function. This point can be, in general, a minimum, a maximum, or a saddle point, and it can be degenerate or nondegenerate (isolated), depending on the Hessian matrix of the potential function. Explicitly, a critical point is degenerate if and only if the Hessian i.e., the matrix of second derivatives; see Ref. 3 is singular at that point. A positive-definite Hessian indicates a local minimum, a negative-definite Hessian indicates a local maximum, and a Hessian of indefinite sign indicates a saddle point. The goal in potential-based motion planning is to have the robot stop at the minimum corresponding to the goal. Although the robot can, in theory, stop at a local maximum or at a saddle point, this is impractical, because such points are unstable, and the probability that this happens is basically zero. Other possible local minima, on the other hand, are attractive, and the robot can get "caught" into such undesirable points in its way to the goal. Most of the existing potential-based planners, where the potential function is constructed through the superposition of attractive (to the goal) functions and repulsive (from the obstacles) functions, suffer from this problem.

To address the local minima problem, two types of approaches have been developed (see Ref. 3 for a detailed overview). In the first approach, the potential field is augmented with a search-based planner. For example, the randomized path planner (RPP)(3) uses a variety of potential

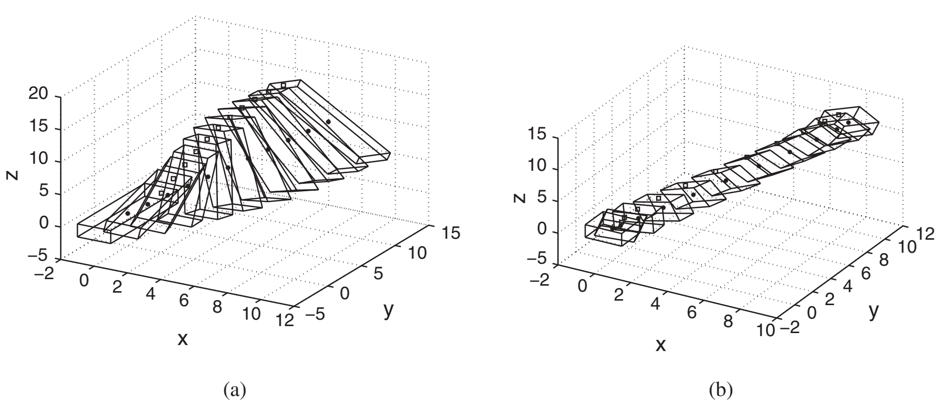


Figure 3. A geodesic (minimum length, or energy) curve for a cuboid and a minimum acceleration curve for a cube.

functions, and when stuck at a local minimum, it performs a random walk, with the goal of escaping the local minimum. In the second approach, a special type of potential function, called a navigation function, is constructed. Although guaranteed to have exactly one minimum, a navigation function can only be applied to a limited class of configuration spaces, which are diffeomorphic to sphere spaces.

ROADMAPS

If several navigation tasks are expected to occur in an environment, then building a map of the environment and then performing navigation using the map can prove to be more efficient than performing navigation from scratch every time such a request occurs. The most used of such maps are topological, or graph-like, structures, where nodes correspond to "interesting features" and the edges show adjacency between nodes. For example, the nodes can be points of interest for a specific task such as targets or intersections, whereas the edges can label actions required from the robot to move from a location to another.

Roadmaps (3) are topological maps embedded in the free space. In other words, in a roadmap, the nodes correspond to physical locations in the environment, and the edges correspond to paths between different locations. A roadmap is, therefore, both a graph and a collection of one-dimensional manifolds (curves). Robots use roadmaps in the same way drivers use the interstates. Instead of planning a trip from point A to point B on small streets, a driver would plan her trip from A to a close interstate, then on the interstate for as long as possible, and then from the interstate to the destination B. Similarly, if a roadmap is available, a robot planner would find a collision-free path to the roadmap, then travel on the roadmap until close to the destination, and then find another collision-free path from the exit point on the roadmap to the destination. Most motion occurs on the roadmap, which is low dimensional, as opposed to the motion to and from the roadmap, which occurs in a possibly high-dimensional configuration space.

Several types of roadmaps have been developed over the years, which include visibility maps, deformation retracts, and silhouettes. In visibility maps, which work for polygonal environments with polygonal obstacles, the nodes are the vertices of the polygons, and an edge between two nodes means that a line of sight exists between the nodes. Deformation retracts capture the "essential" topology of an environment, and they include generalized Voronoi diagrams(3). Finally, silhouette methods are based on repeated projection of the robot-free configuration space onto lower dimensional spaces until a one dimensional representation is reached.

SAMPLING-BASED ALGORITHMS

The construction of roadmaps, as presented above, is based on an explicit representation of the free C-space. As a result, as the dimension of the configuration space increases (e.g., a manipulator arm with several joints and a gripper with fingers can have tens of degrees of freedom), motion planners based on roadmaps become computationally infeasible. In such cases, sampling-based approaches are more appropriate. In short, a sampling-based planner generates samples (i.e., collision-free configurations of the robot) and then interpolating paths for the samples. The latter process is also often achieved through sampling but at a finer rate.

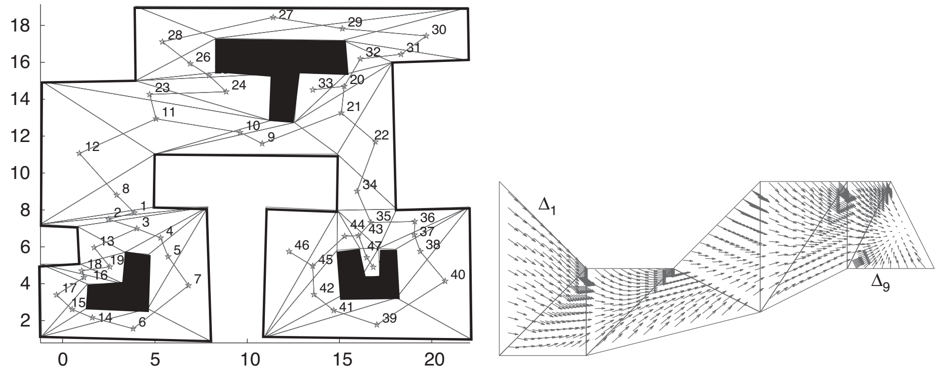
The most representative sampling-based algorithm is the Probabilistic Road Map Planner (PRM) (3). The main idea behind PRM is that it is easy (and cheap) to check for collisions with obstacles. In other words, it is easy to see whether a sample is in the free C-space. PRM uses coarse sampling to obtain the nodes of the roadmap and fine sampling to construct its edges. Once the roadmap is constructed, it is used in exactly the same way as the "classic" roadmap presented in the previous section. The existing PRMs differ by the way samples and interpolating paths are generated. The basic PRM uses uniform distribution for node sampling. Other, more sophisticated, PRMs use sampling schemes such as importance sampling in areas that are difficult to explore and deterministic sampling such as sampling on grids and quasirandom sampling.

As PRM is a roadmap planner, it is optimized for several queries. For single queries, other sampling-based algorithms are effective, such as the rapidly exploring random tree planner (RRT)(3). A combination of multiple-query and single-query methods, such as the sampling-based roadmap of trees (SRT)(3), tries to find a compromise between using a roadmap versus a large sampling tree in very difficult planning problems. Other developments in this area led to sampling-based planners that take into account kinematic and dynamic constraints, stability requirements, energy constraints, visibility constraints, and contact constraints. Sampling-based algorithms opened a new direction in robot motion planning, by making it possible to approach very high-dimensional robotic systems.

CELL DECOMPOSITIONS

Cell decompositions are among the most used techniques for robot motion planning. To illustrate the main ideas, we assume for simplicity that the motion task is a navigation task. The computation behind most of the existing approaches consists of three main steps(3). In the first step, the free configuration space is partitioned, and the quotient graph of the partition is constructed [see Fig. 2(c)]. In this graph, a node labels a free cell, and an edge between two nodes shows an adjacency relation between the corresponding cells. In the second step, the cells corresponding to the initial and the final configurations are determined, and a path is determined between the corresponding nodes in the quotient graph [see Fig. 2(d)]. This path can be optimal with respect to some cost, which in the simplest cases penalizes the distance traveled by the robot. Alternatively, the cost can prevent from generating paths "too close" to the obstacles. Finally, in the third step, a desired robot trajectory is constructed inside the configuration-space tube determined by the path in the quotient graph, and a trajectory-following controller is synthesized for the robot.

The several cell-decomposition methods can be classified according to the underlying partition scheme. The most popular cell decompositions are trapezoidal decomposi-



(a) (b)

Figure 4. (a) A triangulation of the free space in a polygonal environment and the corresponding quotient graph. (b) A sequence of triangles (such as resulting from a path on the quotient graph of a triangulation) is executed by constructing affine vector fields in each triangle.

tions, triangulations, and rectangular grids. For example, Fig. 2(c) shows a rectangular partition, whereas Fig. 4(a) shows a triangulation of a polygonal environment cluttered with polygonal obstacles. Note that although efficient algorithms exist for planar trapezoidal partitions and triangulations, these procedures become cumbersome in higher dimensional spaces. Rectangular partitions, although not particularly efficient in plane, have the advantage of working in higher dimensions, especially when a 2^n-trees (i.e., quad-trees in plane and oct-trees in space) are used during the partition process.

The three-step, top-down process presented above has two main disadvantages. First, because no robot control and/or actuation constraints are taken into account during the decomposition, it is possible that the reference trajectory generated in the last step cannot be followed by the robot. To deal with this issue, in recent years, researchers proposed approaches for the so-called *simultaneous planning and control*. In these studies, the environment partitioning is performed at the same time with the assignment of robot-compatible controllers for each region. For example, polygonal partitions of planar environments followed by assignment of vector fields obtained as solutions of Laplaces equation in each of the regions were considered in Ref. 10. Triangular partitions and rectangular partitions can be also accompanied by the assignment of vector fields with arbitrary polyhedral bounds, if the robot dynamics are restricted to affine and multi-affine (see, for example, Ref. 11). In Figs. 2(e) and 4(b), we show how vector fields are assigned in each rectangle and triangle from a path in the quotient graph. In this setup, the "execution" of a "discrete" path is provably correct (i.e., regardless of the actual position of the robot inside each region), therefore avoiding the trajectory generation and following process.

Trapezoidal, triangular, and rectangular decompositions, as presented above, are mostly used for navigation tasks, and they are not appropriate for coverage(3). Indeed, even if coverage of cells can be efficiently achieved through coverage algorithms on graphs, covering the space inside each cell might be difficult because of the size and shape of the resulting cells. If coverage is the task at hand, then Boustrophedon decompositions and Morse Cell decompositions are more appropriate(3). Roughly put, Boustrophedon decompositions start from a trapezoidal decomposition and reorganize cells such that shorter and more efficient paths can cover the same area. Morse decompositions are based on the same idea, but they allow us to achieve coverage in non polygonal environments. Finally, for a special class of motion planning problems, called pursuit/evasion problems (games), a visibility-based cell decomposition is more appropriate. Roughly, moving from one cell to an adjacent one in this decomposition corresponds to a change in visibility (i.e., target or obstacles appear or disappear).

SYMBOLIC APPROACHES TO MOTION PLANNING AND CONTROL

The current approaches to robot motion planning and control presented above have two main limitations. First, the specification language is not rich enough for a large spectrum or robotic applications. For example, a navigation task is always formulated as "go from A to B and avoid obstacles." Nevertheless, the accomplishment of a mission might require the attainment of either A or B, convergence to a region ("reach A eventually and stay there for all future times"), visiting targets sequentially ("reach A, and then B, and then C"), surveillance ("reach A and then B infinitely often"), and so on. Second, as mentioned, some of the approaches above, such as cell decomposition, do not explicitly take into account the control, sensing, and communication constraints of the robot.

Symbolic approaches to robot motion planning and control have been developed recently to address these limitations. They draw on well-established concepts in related areas, such as behavior-based robotics and hybrid control systems. As the specification language is enriched and real-world robot control, sensing, and communication constraints are taken into account, concepts and tools from the theory of computation such as automata and languages develop naturally, hence the name "symbolic" (see Ref. 12

for a more detailed overview of these methods and the main challenges in the area).

To introduce the main ideas, note that the typical cell-decomposition approach to the navigation problem is a hierarchical, three-level process. At the first (top-most) level, the specification "go from A to B and avoid obstacles" is given, the obstacle-free configuration space of the robot is partitioned into cells, and the quotient graph is constructed (see Figs. 2(c) and 4(a) for examples). As any path connecting the cell containing A to the cell containing B in this graph satisfies the specification (i.e., it avoids obstacles), this is called the specification level. In the second step, a path on this graph is chosen, which can be seen as a "discrete" execution of the robot, hence, the name *execution level* for this step. Finally, in the third step, called the *implementation level*, a reference trajectory traversing the sequence of cells given by the path is generated, and robot controllers are constructed so that the reference trajectory is followed.

Symbolic approaches to motion planning fit into the three-level hierarchy described above, and they can be divided into two main groups: top-down approaches and bottom-up approaches. In top-down approaches (also referred to as middle-out approaches(13)), the focus is on the expressivity of the specification language, and the hope is that, while going down the three-level hierarchy presented above, implementations are possible for real-world robots. In bottom-up approaches, the starting point is a careful analysis of the control and sensing communication of the robot, possible executions are generated at the execution level, and the hope is that the set of such robot-compatible executions give rise to an expressive specification language. However, a significant gap exists between these two approaches. Bridging in this gap is one of the main challenges in the area(12).

Top-Down Symbolic Approaches

It was recently suggested that, to define a rich specification language for robot motion, inspiration can be taken from temporal logics, which are commonly used for specifying and verifying the correctness of digital circuits and computer programs. Roughly, any rich, human-like, temporal, and logic statement about the reachability of regions of interest by the robot (including the ones given as examples above) translate naturally to a formula in such a logic. Consider, for example, that a robot moves in an environment with three obstacles o_1, o_2, and o_3 and three targets r_1, r_2, and r_3 that need to be surveyed (visited infinitely many times). In other words, the task can be formulated as "*Always avoid obstacles* o_1, o_2, o_3 *and visit regions* r_1, r_2, r_3, *in this order, infinitely often.*" This specification immediately translates to the following formula of linear temporal logic (LTL) over the set of symbols o_1, o_2, o_3, r_1, r_2, r_3: $\mathbf{G}(\mathbf{F}(r_1 \wedge \mathbf{F}(r_2 \wedge \mathbf{F}r_3)) \wedge \neg(o_1 \vee o_2 \vee o_3))$, where $\neg$ and $\wedge$ stand for Boolean negation and disjunction and $\mathbf{G}$ and $\mathbf{F}$ are temporal operators that mean "always" and "eventually," respectively.

The semantics of LTL formulas are given over labeled transition graphs (also called Kripke structures or transition systems). Such a transition system can be obtained from the dual graph of the partition induced by the regions of interest, if the nodes are labeled according to their being part of obstacles or of targets, and if the edges are viewed as transitions that a robot can take. To compute a transition between two nodes (or a self-transition), one could proceed by checking for the existence of robot feedback controllers taking the robot from one region to another in finite time (or keeping the robot inside the region forever), regardless of the initial position of the robot. If this is achieved, then a certain type of equivalence relation exists between the initial control system describing the motion of the robot in the environment and the finite transition system, called bisimulation, which guarantees that the two systems satisfy the same LTL formula. Therefore, provided that the two types of controllers can be constructed, the motion planning problem is reduced to a classic model checking procedure, for which exist several off-the-shelf tools developed by the formal verification community(14).

Currently, two classes of systems are available for which such quotients can be efficiently constructed: affine systems with polyhedral partitions, and multi-affine systems (i.e., polynomial systems where the maximum power at which a variable can occur is one) with rectangular partitions. Although these two classes of systems seem restrictive for robot dynamics, it is important to note that multi-affine dynamics capture vector cross products, and they can therefore accommodate dynamics of aircraft with gas-jet actuators and underwater vehicles. In addition, differential- drive and car-like vehicles can be easily accommodated by solving an additional input–output feedback linearization. Fully automatic computational frameworks for control of affine and multi-affine dynamics from rich specifications given as arbitrary LTL formulas over linear and rectangular predicates were developed in Ref. 15 and 16. A related approach was used in Ref. 17 to control a nonholonomic robot model. In Ref. 18, it is shown that a significant decrease in computation can be achieved if the specifications are restricted to a fragment of LTL.

Bottom-Up Symbolic Approaches

The top-down symbolic approaches presented above use environment discretization to capture the complexity of the environment. While allowing for a rich specification language over the partition regions, they are (in current form) restricted to static, *a priori* known environments and simple robot dynamics, such as fully actuated or affine dynamics with polyhedral speed constraints. As suggested, robots with more complex dynamics such as helicopter-like vehicles might not be able to implement executions strings over partition regions. In this situation, the discretization may be more appropriate at the level of controllers rather than environments. The argument behind such a control-driven discretization is that the global control task can be broken down into more easily defined behavioral building blocks, each defined with respect to a particular subtask, sensing modality, or operating point. Strings over such behaviors make up words in so-called motion description languages (MDLs)(19). An example of such a string is $(k_{i_1}, \xi_{i_1}), \ldots, (k_{i_q}, \xi_{i_q})$, where $k_{i_j} : \mathcal{R}^+ \times X \to U$ are feedback control laws and $\xi_{i_j} : \mathcal{R}^+ \times X \to \{0, 1\}$ are temporal or environmentally driven interrupt conditions, $j = 1, \ldots, q$. The robot "parses" such words as $\dot{x} = f(x, k_{i_1}(t, x))$ until $\xi_{i_1}(t, x) = 1$, at which point the timer t is reset to 0, and $\dot{x} = f(x, k_{i_2}(t, x))$ until $\xi_{i_2}(t, x) = 1$, and so on.

An attractive alternative to MDL is to use motion primitives. The idea is that, instead of using controllers chosen from a collection of controls, one could think of simplifying a robot control problem by piecing together, in an appropriate way, a set of elementary trajectories chosen from a small "library"—that are themselves guaranteed to satisfy the constraints. Such feasible trajectories that can be combined sequentially to produce more complicated trajectories are called "motion primitives"(20). The compatibility rules between such primitives can be, as above, modeled as finite-state machines, called Maneuver Automata. Motion primitives can be generated in several ways, for example, by recording the actions of a human pilot; if an accurate model of the robot' s dynamics is available, model-based approaches are also possible (e.g., to design optimal maneuvers).

Although the symbolic approaches to motion planning described in this section have been applied successfully to challenging problems in autonomous mobile robotics, including acrobatic aircraft, and off-road races, several challenges still need to be addressed. For example, the problem of choosing the control modes (quanta) or motion primitives for achieving a given task is not at all obvious. One way of addressing it is by letting the control mode selection be driven by experimental data. For instance, one can envision a scenario in which a human operator is controlling a mobile platform and then, through an analysis of the input–output sample paths, construct motion description languages that reproduce the human-driven robot behavior.

BIBLIOGRAPHY

1. J. C. Latombe, *Robot Motion Planning*, Boston, MA: Kluger Academic Publishers., 1991.
2. S. M. LaValle, *Planning Algorithms*, Cambridge, UK: Cambridge University Press, 2006.
3. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, Boston, MA: MIT Press, 2005.
4. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2005.
5. Z. Li and J. F. Canny, (eds.), *Nonholonomic Motion Planning*, Norwell, MA: Kluwer Academic Publishers, 1992.
6. K. Shoemake, Animating rotation with quaternion curves, *ACM Siggraph*, **19** (3): 245–254, 1985.
7. C. Belta, Geometric methods for multi-robot motion planning and control, PhD thesis, Philadelphia, PA, University of Pennsylvania, 2003.
8. M. Žefran, V. Kumar, and C. Croke, On the generation of smooth three-dimensional rigid body motions, *IEEE Trans. Robotics Auto.*, **14** (4): 579–589, 1995.
9. E. Rimon and D. E. Koditschek, Exact robot navigation using artificial potential functions, *IEEE Trans. Robotics Auto.*, **8** (5): 501–518.
10. D. C. Conner, A. A. Rizzi, and H. Choset, Composition of local potential functions for global robot control and navigation, *Proc. of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, 2003.
11. C. Belta, V. Isler, and G. J. Pappas, Discrete abstractions for robot planning and control in polygonal environments, *IEEE Trans. Robotics*, **21** (5): 864–874, 2005.
12. C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, Symbolic planning and control of robot motion, *IEEE Robotics Auto. Mag.*, **14** (1): 61–71, 2007.
13. M. S. Branicky, T. A. Johansen, I. Petersen, and E. Frazzoli, On-line techniques for behavioral programming, *Proc. of the IEEE Conference on Decision and Control*, Sydney, Australia, 2000.
14. E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA: The MIT Press, 2000.
15. L. C. G. J. M. Habets, P. J. Collins, and J. H. van Schuppen, Reachability and control synthesis for piecewise-affine hybrid systems on simplices, *IEEE Trans. Aut. Control*, **51**: 938–948, 2006.
16. M. Kloetzer and C. Belta, A fully automated framework for control of linear systems from temporal logic specifications, *IEEE Trans. Auto. Cont.*, **53** (1): 287–297, 2008.
17. D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas, Valet parking without a valet, *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, 2007.
18. G. Fainekos, S. Loizou, and G. J. Pappas, Translating temporal logic to controller specifications, *Proc. 45th IEEE Conference on Decision and Control*, San Diego, CA, 2006.
19. M. Egerstedt and R. W. Brockett, Feedback can reduce the specification complexity of motor programs, *IEEE Trans. Auto. Cont.*, **48** (2): 213–223, 2003.
20. E. Frazzoli, M. A. Dahleh, and E. Feron, Maneuver-based motion planning for nonlinear systems with symmetries, *IEEE Trans. Robotics*, **21** (6): 1077–1091, 2005.

CALIN BELTA
Boston University
Brookline, Massachusetts