# 1 *Introduction*

SOME OF the most significant challenges confronting autonomous robotics lie in the area of automatic motion planning. The goal is to be able to specify a task in a high-level language and have the robot automatically compile this specification into a set of low-level motion primitives, or feedback controllers, to accomplish the task. The prototypical task is to find a path for a robot, whether it is a robot arm, a mobile robot, or a magically free-flying piano, from one configuration to another while avoiding obstacles. From this early *piano mover's problem,* motion planning has evolved to address a huge number of variations on the problem, allowing applications in areas such as animation of digital characters, surgical planning, automatic verification of factory layouts, mapping of unexplored environments, navigation of changing environments, assembly sequencing, and drug design. New applications bring new considerations that must be addressed in the design of motion planning algorithms.

Since actions in the physical world are subject to physical laws, uncertainty, and geometric constraints, the design and analysis of motion planning algorithms raises a unique combination of questions in mechanics, control theory, computational and differential geometry, and computer science. The impact of automatic motion planning, therefore, goes beyond its obvious utility in applications. The possibility of building computer-controlled mechanical systems that can sense, plan their own motions, and execute them has contributed to the development of our math and science base by asking fundamental theoretical questions that otherwise might never have been posed.

This book addresses the theory and practice of robot motion planning, with an eye toward applications. To focus the discussion, and to point out some of the important concepts in motion planning, let's first look at a few motivating examples.

### Piano Mover's Problem

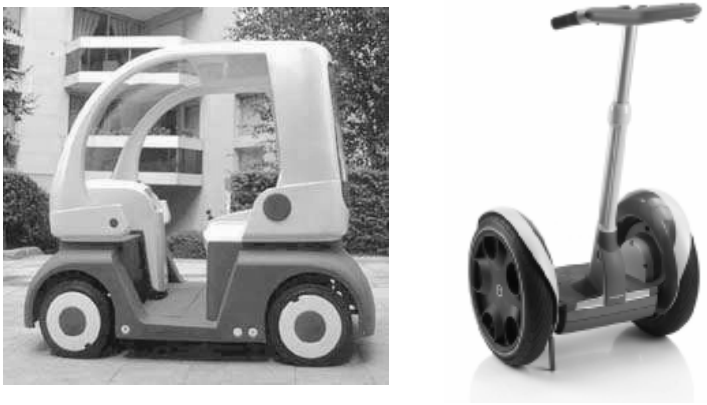
The classic path planning problem is the piano mover's problem [373]. Given a three-dimensional rigid body, for example a polyhedron, and a known set of obstacles, the problem is to find a collision-free *path* for the omnidirectional free-flying body from a start configuration to a goal configuration. The obstacles are assumed to be stationary and perfectly known, and execution of the planned path is exact. This is called *offline* planning, because planning is finished in advance of execution. Variations on this problem are the *sofa mover's problem,* where the body moves in a plane among planar obstacles, and the *generalized mover's problem,* where the robot may consist of a set of rigid bodies linked at joints, e.g., a robot arm.

The key problem is to make sure no point on the robot hits an obstacle, so we need a way to represent the location of all the points on the robot. This representation is the *configuration* of the robot, and the *configuration space* is the space of all configurations the robot can achieve. An example of a configuration is the set of joint angles for a robot arm or the one orientation and two position variables for a sofa in the plane. The configuration space is generally *non-Euclidean,* meaning that it does not look like an $n$-dimensional Euclidean space $\mathbb{R}^n$. The dimension of the configuration space is equal to the number of independent variables in the representation of the configuration, also known as the *degrees of freedom* (DOF). The piano has six degrees of freedom: three to represent the position ($x$-$y$-$z$) and three to represent the orientation (roll-pitch-yaw). The problem is to find a curve in the configuration space that connects the start and goal points and avoids all *configuration space obstacles* that arise due to obstacles in the space.

### The Mini AERCam

NASA's Johnson Space Center is developing the Mini AERCam, or Autonomous Extravehicular Robotic Camera, for visual inspection tasks in space (figure 1.1). It is a free-flying robot equipped with twelve cold gas thrusters, allowing it to generate a force and torque in any direction. When operating in autonomous mode, it must be able to navigate in a potentially complex three-dimensional environment. In this respect the problem is similar to the piano mover's problem. Since we have to apply thrusts to cause motion, however, we need to plan not only the path the robot is to follow, but also the speed along the path. This is called a *trajectory,* and the thruster inputs are determined by the *dynamics* of the robot. In the piano mover's problem, we only worried about geometric or *kinematic* issues.

**Figure 1.1** NASA's Mini AERCam free-flying video inspection robot.



(a) (b)

**Figure 1.2** (a) The CyCab. (b) The Segway Human Transporter.

### Personal Transport Vehicles

Small personal transport vehicles may become a primary means of transportation in pedestrian-filled urban environments where the size, speed, noise, and pollution of automobiles is undesirable. One concept is the CyCab [355], a small vehicle designed by a consortium of institutions in France to transport up to two people at speeds up to 30 km/h (figure 1.2a). Another concept is the Segway HT, designed to carry a single rider at speeds up to 20 km/h (figure 1.2b).

To simplify control of vehicles in crowded environments, one capability under study is automatic parallel parking. The rider would initiate the parallel-parking procedure, and the onboard computer would take over from there. Such systems will soon be commercially available in automobiles. On the surface, this problem sounds like the sofa mover's problem, since both involve a body moving in the plane among obstacles. The difference is that cars and the vehicles in figure 1.2 cannot instantaneously slide sideways like the sofa. The velocity constraint preventing instantaneous sideways motion is called a *nonholonomic* constraint, and the motion planner must take this constraint into account. Systems without velocity constraints, such as the sofa, are omnidirectional in the configuration space.

### Museum Tour Guides

In 1997, a mobile robot named RHINO served as a fully autonomous tour-guide at the Deutsches Museum Bonn (figure 1.3). RHINO was able to lead museum visitors from one exhibit to the next by calculating a path using a stored map of the museum. Because the perfect execution model of the piano mover's problem is unrealistic in this setting, RHINO had to be able to *localize* itself by comparing its sensor readings to its stored


**Figure 1.3** RHINO, the interactive mobile tour-guide robot.

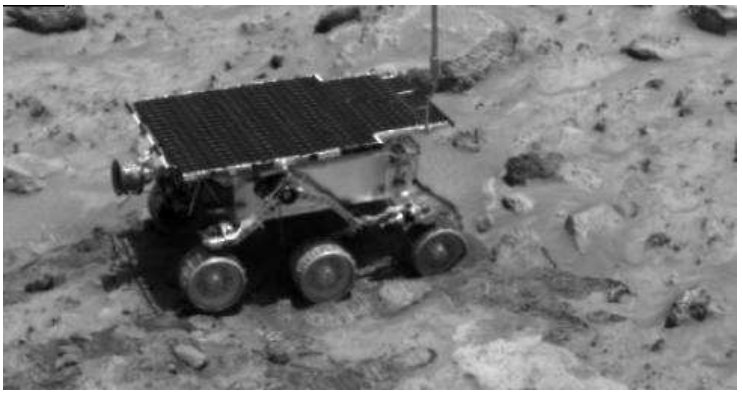


**Figure 1.4** The Mars rover Sojourner. http://mars.jpl.nasa.gov/MPF/rover/sojourner.html.

map. To deal with uncertainty and changes in the environment, RHINO employed a *sensor-based* planning approach, interleaving sensing, planning, and action.

### Planetary Exploration

One of the most exciting successes in robot deployment was a mobile robot, called Sojourner (figure 1.4), which landed on Mars on July 4, 1997. Sojourner provided up-close images of Martian terrain surrounding the lander. Sojourner did not move very far from the lander and was able to rely on motion plans generated offline on Earth and uploaded. Sojourner was followed by its fantastically successful cousins, Spirit and Opportunity, rovers that landed on Mars in January 2004 and have provided a treasure trove of scientific data. In the future, robots will explore larger areas and thus will require significantly more autonomy. Beyond navigation capability, such robots will have to be able to generate a map of the environment using sensor information. *Mapping* an unknown space with a robot that experiences positioning error is an especially challenging "chicken and egg" problem—without a map the robot cannot determine its own position, and without knowledge about its own position the robot cannot compute the map. This problem is often called *simultaneous localization and mapping* or simply *SLAM*.

### Demining

Mine fields stifle economic development and result in tragic injuries and deaths each year. As recently as 1994, 2.5 million mines were placed worldwide while only 100,000 were removed.

Robots can play a key role in quickly and safely demining an area. The crucial first step is finding the mines. In demining, a robot must pass a mine-detecting sensor over all points in the region that might conceal a mine. To do this, the robot must traverse a carefully planned path through the target region. The robot requires a *coverage* path planner to find a motion that passes the sensor over every point in the field. If the planner is guaranteed to find a path that covers every point in the field when such a path exists, then we call the planner *complete*. Completeness is obviously a crucial requirement for this task.

Coverage has other applications including floor cleaning [116], lawn mowing [198], unexploded ordnance hunting [260], and harvesting [341]. In all of these applications, the robot must simultaneously localize itself to ensure complete coverage.

### Fixed-base Robot Arms in Industry

In highly structured spaces, fixed-base robot arms perform a variety of tasks, including assembly, welding, and painting. In painting, for example, the robot must deposit a uniform coating over all points on a target surface (figure 1.5). This coverage problem presents new challenges because (1) ensuring equal paint deposition is a more severe requirement than mere coverage, (2) the surface is not usually flat, and (3) the robot must properly coordinate its internal degrees of freedom to drive the paint atomizer over the surface.

Industrial robot installations are clearly driven by economic factors, so there is a high priority on minimizing task execution time. This motivates motion planners that return *time-optimal* motion plans. Other kinds of tasks may benefit from other kinds of optimality, such as energy- or fuel-optimality for mobile robots.

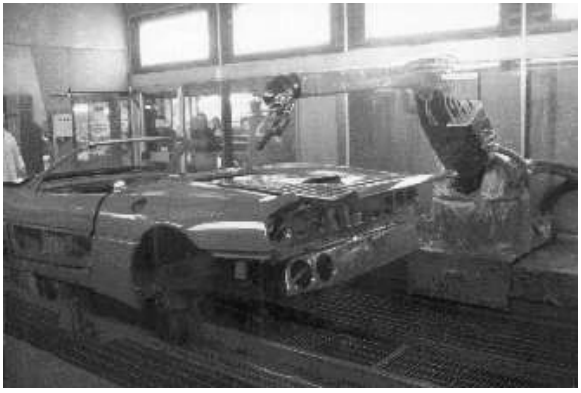


**Figure 1.5** ABB painting robot named Tobe.

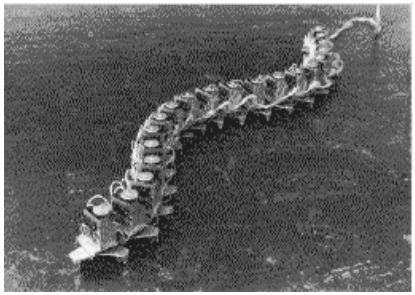


**Figure 1.6** The Hirose active cord.

**Figure 1.7** The Carnegie Mellon snake robot Holt mounted on a mobile base.

### Snake Robots for Urban Search and Rescue

When a robot has more degrees of freedom than required to complete its task, the robot is called *redundant*. When a robot has many extra degrees of freedom, then it is called *hyper-redundant*. These robots have multidimensional non-Euclidean configuration spaces. Hyper-redundant serial mechanisms look like elephant trunks or snakes (figures 1.6 and 1.7), and they can use their extra degrees of freedom to thread through tightly packed volumes to reach locations inaccessible to humans and conventional machines. These robots may be particularly well-suited to urban search and rescue, where it is of paramount importance to locate survivors in densely packed rubble as quickly and safely as possible.

### Robots in Surgery

Robots are increasingly used in surgery applications. In noninvasive stereotactic radiosurgery, high-energy radiation beams are cross-fired at brain tumors. In certain cases,

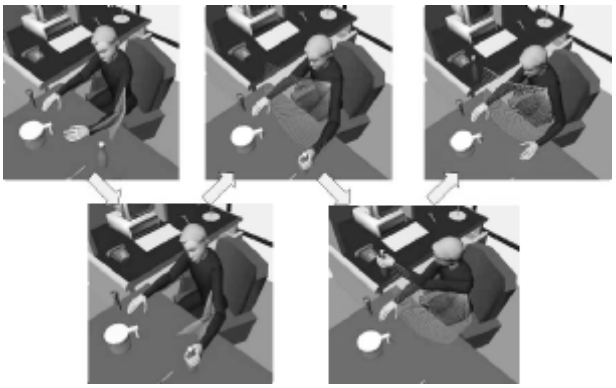


**Figure 1.8** The motions of a digital actor are computed automatically. (Courtesy of J.C. Latombe)

these beams are delivered with high accuracy, using six degrees of freedom robotic arms (e.g., the CyberKnife system [1] ). Robots are also used in invasive procedures. They often enhance the surgeon's ability to perform technically precise maneuvers. For example, the da Vinci Surgical System [2] can assist in advanced surgical techniques such as cutting and suturing. The ZEUS System [3] can assist in the control of blunt retractors, graspers, and stabilizers. Clearly, as robotics advances, more and more of these systems will be developed to improve our healthcare.

### Digital Actors

Algorithms developed for motion planning or sensor interpretation are not just for robots anymore. In the entertainment industry, motion planning has found a wide variety of applications in the generation of motion for digital actors, opening the way to exciting scenarios in video games, animation, and virtual environments (figure 1.8).

### Drug Design

An important problem in drug design and the study of disease is understanding how a protein folds to its native or most stable configuration. By considering the protein as an articulated linkage (figure 1.9), researchers are using motion planning to identify likely folding pathways from a long straight chain to a tightly folded configuration. In pharmaceutical drug design, proteins are combined with smaller molecules to form complexes that are vital for the prevention and cure of disease. Motion planning

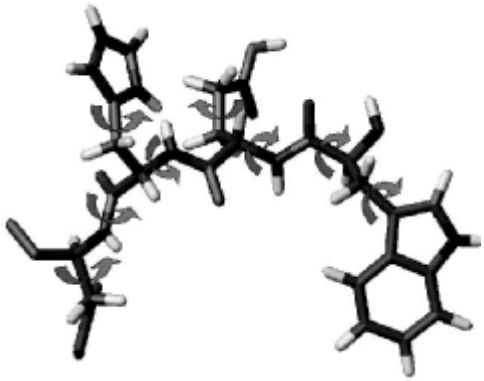


**Figure 1.9** A molecule represented as an articulated linkage.

methods are used to analyze molecular binding motions, allowing the automated testing of drugs before they are synthesized in the laboratory.

## 1.1 Overview of Concepts in Motion Planning

The previous examples touched on a number of ways to characterize the motion planning problem and the algorithm used to address it. Here we summarize some of the important concepts. Our characterization of a motion planner is according to the task it addresses, properties of the robot solving the task, and properties of the algorithm.[1] We focus on topics that are covered in this book (table 1.1).

### Task

The most important characterization of a motion planner is according to the problem it solves. This book considers four tasks: *navigation, coverage, localization,* and *mapping*. *Navigation* is the problem of finding a collision-free motion for the robot system from one configuration (or state) to another. The robot could be a robot arm, a mobile robot, or something else. *Coverage* is the problem of passing a sensor or tool over all points in a space, such as in demining or painting. *Localization* is the problem of using a map to interpret sensor data to determine the configuration of the robot. *Mapping* is the problem of exploring and sensing an unknown environment

1. This classification into three categories is somewhat arbitrary but will be convenient for introduction.

| Task | Robot | Algorithm |
|---|---|---|
| Navigate | Configuration space, degree of freedom | Optimal/nonoptimal motions |
| Map | Kinematic/dynamic | Computational complexity |
| Cover | Omnidirectional or motion constraints | Completeness (resolution, probabilistic) |
| Localize | | Online/offline |
| | | Sensor-based/world model |

**Table 1.1** Some of the concepts covered in this book.

to construct a representation that is useful for navigation, coverage, or localization. Localization and mapping can be combined, as in SLAM.

There are a number of interesting motion planning tasks not covered in detail in this book, such as navigation among moving obstacles, manipulation and grasp planning, assembly planning, and coordination of multiple robots. Nonetheless, algorithms in this book can be adapted to those problems.

### Properties of the Robot

The form of an effective motion planner depends heavily on properties of the robot solving the task. For example, the robot and the environment determine the number of *degrees of freedom* of the system and the shape of the *configuration space*. Once we understand the robot's configuration space, we can ask if the robot is free to move instantaneously in any direction in its configuration space (in the absence of obstacles). If so, we call the robot omnidirectional. If the robot is subject to velocity constraints, such as a car that cannot translate sideways, both the constraint and the robot are called *nonholonomic*. Finally, the robot could be modeled using *kinematic* equations, with velocities as controls, or using *dynamic* equations of motion, with forces as controls.

### Properties of the Algorithm

Once the task and the robot system is defined, we can choose between algorithms based on how they solve the problem. For example, does the planner find motions that are *optimal* in some way, such as in length, execution time, or energy consumption? Or does it simply find a solution satisfying the constraints? In addition to the quality of the output of the planner, we can ask questions about the *computational complexity* of

the planner. Are the memory requirements and running time of the algorithm constant, polynomial, or exponential in the "size" of the problem description? The size of the input could be the number of degrees of freedom of the robot system, the amount of memory needed to describe the robot and the obstacles in the environment, etc., and the complexity can be defined in terms of the worst case or the average case. If we expect to scale up the size of the inputs, a planner is often only considered practical if it runs in time polynomial or better in the inputs. When a polynomial time algorithm has been found for a problem that previously could only be solved in exponential time, some key insight into the problem has typically been gained.

Some planners are *complete,* meaning that they will always find a solution to the motion planning problem when one exists or indicate failure in finite time. This is a very powerful and desirable property. For the motion planning problem, as the number of degrees of freedom increases, complete solutions may become computationally intractable. Therefore, we can seek weaker forms of completeness. One such form is *resolution completeness.* It means that if a solution exists at a given resolution of discretization, the planner will find it. Another weaker form of completeness is *probabilistic completeness.* It means that the probability of finding a solution (if one exists) converges to 1 as time goes to infinity.

Optimality, completeness, and computational complexity naturally trade off with each other. We must be willing to accept increased computational complexity if we demand optimal motion plans or completeness from our planner.

We say a planner is *offline* if it constructs the plan in advance, based on a known model of the environment, and then hands the plan off to an executor. The planner is *online* if it incrementally constructs the plan while the robot is executing. In this case, the planner can be *sensor-based,* meaning that it interleaves sensing, computation, and action. The distinction between offline algorithms and online sensor-based algorithms can be somewhat murky; if an offline planner runs quickly enough, for example, then it can be used in a feedback loop to continually replan when new sensor data updates the environment model. The primary distinction is computation time, and practically speaking, algorithms are often designed and discussed with this distinction in mind. A similar issue arises in control theory when attempting to distinguish between feedforward control (commands based on a reference trajectory and dynamic model) and feedback control (commands based on error from the desired trajectory), as techniques like *model predictive control* essentially use fast feedforward control generation in a closed loop. In this book we will not discuss the low-level feedback controllers needed to actually implement robot motions, but we will assume they are available.

## 1.2 Overview of the Book

Chapter 2 dives right into a class of simple and intuitive "Bug" algorithms requiring a minimum of mathematical background to implement and analyze. The task is to navigate a point mobile robot to a known goal location in a plane filled with unknown static obstacles. The Bug algorithms are sensor-based—the robot uses a contact sensor or a range sensor to determine when it is touching or approaching an obstacle, as well as odometry or other sensing to know its current position in the plane. It has two basic motion primitives, moving in a straight line and following a boundary, and it switches between these based on sensor data. These simple algorithms guarantee that the robot will arrive at the goal if it is reachable.

To move beyond simple point robots, in chapter 3 we study the configuration space of more general robot systems, including rigid bodies and robot arms. The mathematical foundations in this chapter allow us to view general path planning problems as finding paths through configuration space. We study the dimension (degrees of freedom), topology, and parameterizations of non-Euclidean configuration spaces, as well as representations of these configuration spaces as surfaces embedded in higher-dimensional Euclidean spaces. The forward kinematic map is introduced to relate one choice of configuration variables to another. The differential of this map, often called the Jacobian, is used to relate the velocites in the two coordinate systems. Material in this chapter is referenced throughout the remainder of the book.

Chapter 4 describes a class of navigation algorithms based on *artificial potential functions*. In this approach we set up a virtual potential field in the configuration space to make obstacles repulsive and the goal configuration attractive to the robot. The robot then simply follows the downhill gradient of the artificial potential. For some navigation problems, it is possible to design the potential field to ensure that following the gradient will always take the robot to the goal. If calculating such a potential field is difficult or impossible, we can instead use one that is easy to caclulate but may have the undesirable property of local minima, locations where the robot gets "stuck." In this case, we can simply use the potential field to guide a search-based planner. Potential fields can be generated offline, using a model of the environment, or calculated in real-time using current sensor readings. Purely reactive gradient-following potential field approaches always run the risk of getting stuck in local minima, however.

In chapter 5, we introduce more concise representations of the robot's free space that a planner can use to plan paths between two configurations. These structures are called *roadmaps*. A planner can also use a roadmap to explore an unknown space. By using sensors to incrementally construct the roadmap, the robot can then use the roadmap for future navigation problems. This chapter describes several roadmaps

including the visibility graph, the generalized Voronoi diagram, and Canny's original roadmap. Chapter 6 describes an alternative representation of the free space called a *cell decomposition* which consists of a set of cells of the free space and a graph of cells with connections between adjacent cells. A cell decomposition is useful for coverage tasks, and it can be computed offline or incrementally using sensor data.

Constructing complete and exact roadmaps of an environment is generally quite computationally complex. Therefore, chapter 7 develops sampling-based algorithms that trade completeness guarantees for a reduction of the running time of the planner. This chapter highlights recent work in probabilistic roadmaps, expansive-spaces trees, and rapidly-exploring random trees and the broad range of motion planning problems to which they are applicable.

Probabilistic reasoning can also address the problems of sensor uncertainty and positioning error that plague mobile robot deployment. We can model these uncertainties and errors as probability distributions and use *Kalman filtering* (chapter 8) and *Bayesian estimation* (chapter 9) to address localization, mapping, and SLAM problems.

Just as the description of configuration space in chapter 3 provides many of the kinematic and geometric tools used in path planning, the description of second-order *robot dynamics* in chapter 10 is necessary for feasible trajectory planning, i.e., finding motions parameterized by time. We can then pose time- and energy-optimal *trajectory planning* problems for dynamic systems subject to actuator limits, as described in chapter 11.

Chapter 11 assumes that the robot has an actuator for each degree of freedom. In chapter 12 we remove that assumption and consider robot systems subject to *nonholonomic* (velocity) constraints and/or acceleration constraints due to missing actuators, or *underactuation*. We study the reachable states for such systems, i.e., controllability, using tools from differential geometry. The chapter ends by describing planners that find motions for systems such as cars, cars pulling trailers, and spacecraft or robot arms with missing actuators.

## 1.3 Mathematical Style

Our goal is to present topics in an intuitive manner while helping the reader appreciate the deeper mathematical concepts. Often we suppress mathematical rigor, however, when intuition is sufficient. In many places proofs of theorems are omitted, and the reader is referred to the original papers. For the most part, mathematical concepts are introduced as they are needed. Supplementary mathematical material is deferred to the appendices to allow the reader to focus on the main concepts of the chapter.

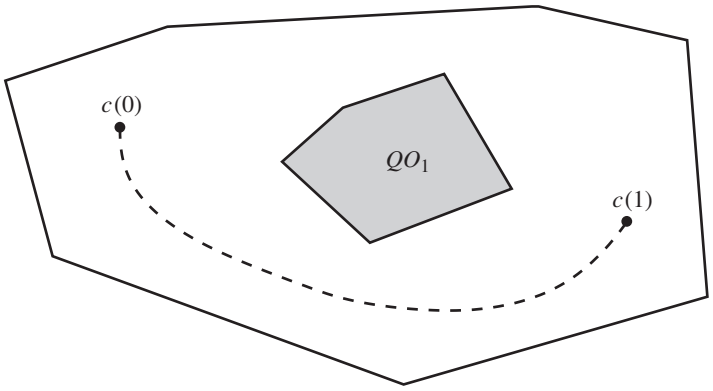



**Figure 1.10** A path is a curve in the free configuration space $\mathcal{Q}_{\text{free}}$ connecting $c(0)$ to $c(1)$.

Throughout this book, robots are assumed to operate in a planar ($\mathbb{R}^2$) or three-dimensional ($\mathbb{R}^3$) ambient space, sometimes called the *workspace* $\mathcal{W}$. This workspace will often contain obstacles; let $\mathcal{WO}_i$ be the $i$th obstacle. The *free workspace* is the set of points $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \bigcup_i \mathcal{WO}_i$ where the $\setminus$ is a subtraction operator.

Motion planning, however, does not usually occur in the workspace. Instead, it occurs in the configuration space $\mathcal{Q}$ (also called C-space), the set of all robot configurations. We will use the notation $R(q)$ to denote the set of points of the ambient space occupied by the robot at configuration $q$. An obstacle in the configuration space corresponds to configurations of the robot that intersect an obstacle in the workspace, i.e., $\mathcal{QO}_i = \{q \mid R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}$. Now we can define the *free configuration space* as $\mathcal{Q}_{\text{free}} = \mathcal{Q} \setminus \bigcup_i \mathcal{QO}_i$. We sometimes simply refer to "free space" when the meaning is unambiguous.

In this book we make a distinction between *path planning* and *motion planning*. A *path* is a continuous curve on the configuration space. It is represented by a continuous function that maps some path parameter, usually taken to be in the unit interval [0, 1], to a curve in $\mathcal{Q}_{\text{free}}$ (figure 1.10). The choice of unit interval is arbitrary; any parameterization would suffice. The solution to the *path planning* problem is a continuous function $c \in C^0$ (see appendix C for a definition of continuous functions) such that

$$c : [0, 1] \rightarrow \mathcal{Q} \text{ where } c(0) = q_{\text{start}},\ c(1) = q_{\text{goal}} \text{ and } c(s) \in \mathcal{Q}_{\text{free}}\ \ \forall s \in [0, 1]. \tag{1.1}$$

When the path is parameterized by time $t$, then $c(t)$ is a trajectory, and velocities and accelerations can be computed by taking the first and second derivatives with

respect to time. This means that $c$ should be at least twice-differentiable, i.e., in the class $C^2$. Finding a feasible trajectory is called trajectory planning or *motion planning*.

In this book, configuration, velocity, and force vectors will be written as column vectors when they are involved in any matrix algebra. For example, a configuration $q \in \mathbb{R}^n$ will be written in coordinates as $q = [q_1, q_2, \ldots, q_n]^T$. When the vector will not be used in any computation, we may simply refer to it as a tuple of coordinates, e.g., $q = (q_1, q_2, \ldots, q_n)$, without bothering to make it a column vector.