Chapter 10

Motion Planning

Motion planning is the problem of finding a robot motion from a start state to a goal state that avoids obstacles in the environment and satisfies other constraints, such as joint limits or torque limits. Motion planning is one of the most active subfields of robotics, and it is the subject of entire books. The purpose of this chapter is to provide a practical overview of a few common techniques, using robot arms and mobile robots as the primary example systems (Figure 10.1).

The chapter begins with a brief overview of motion planning. This is followed by foundational material including configuration space obstacles and graph search. We conclude with summaries of several different planning methods.

10.1 Overview of Motion Planning

A key concept in motion planning is configuration space, or **C-space** for short. Every point in the C-space $\mathcal{C}$ corresponds to a unique configuration q of the robot, and every configuration of the robot can be represented as a point in C-space. For example, the configuration of a robot arm with n joints can be represented as a list of n joint positions, $q = (\theta_1, \ldots, \theta_n)$. The **free C-space** $\mathcal{C}_{\text{free}}$ consists of the configurations where the robot neither penetrates an obstacle nor violates a joint limit.

In this chapter, unless otherwise stated, we assume that q is an n-vector and that $\mathcal{C} \subset \mathbb{R}^n$. With some generalization, the concepts of this chapter apply to non-Euclidean C-spaces such as $\mathcal{C} = SE(3)$.

The control inputs available to drive the robot are written as an m-vector $u \in \mathcal{U} \subset \mathbb{R}^m$, where $m = n$ for a typical robot arm. If the robot has second-

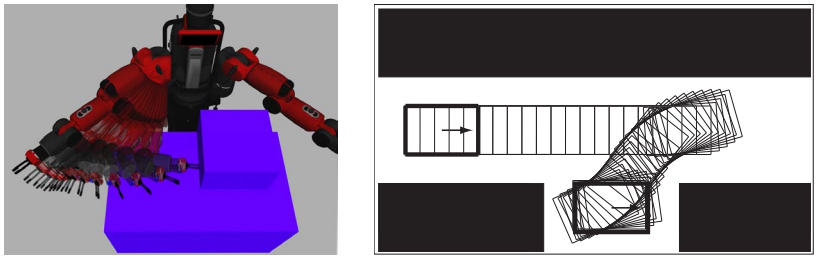

Figure 10.1: (Left) A robot arm executing an obstacle-avoiding motion plan. The motion plan was generated using MoveIt! [180] and visualized using rviz in ROS (the Robot Operating System). (Right) A car-like mobile robot executing parallel parking.

order dynamics, such as that for a robot arm, and the control inputs are forces (equivalently, accelerations), the *state* of the robot is defined by its configuration and velocity, $x = (q, v) \in \mathcal{X}$. For $q \in \mathbb{R}^n$, typically we write $v = \dot{q}$. If we can treat the control inputs as velocities, the state x is simply the configuration q. The notation $q(x)$ indicates the configuration q corresponding to the state x, and $\mathcal{X}_{\text{free}} = \{x \mid q(x) \in \mathcal{C}_{\text{free}}\}$.

The equations of motion of the robot are written

$$\dot{x} = f(x, u) \tag{10.1}$$

or, in integral form,

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt. \tag{10.2}$$

10.1.1 Types of Motion Planning Problems

With the definitions above, a fairly broad specification of the motion planning problem is the following:

Given an initial state $x(0) = x_{\text{start}}$ and a desired final state x_{goal}, find a time T and a set of controls $u : [0, T] \to \mathcal{U}$ such that the motion (10.2) satisfies $x(T) = x_{\text{goal}}$ and $q(x(t)) \in \mathcal{C}_{\text{free}}$ for all $t \in [0, T]$.

It is assumed that a feedback controller (Chapter 11) is available to ensure that the planned motion $x(t)$, $t \in [0, T]$, is followed closely. It is also assumed that an accurate geometric model of the robot and environment is available to evaluate $\mathcal{C}_{\text{free}}$ during motion planning.

There are many variations of the basic problem; some are discussed below.

Path planning versus motion planning. The path planning problem is a subproblem of the general motion planning problem. Path planning is the purely geometric problem of finding a collision-free path $q(s), s \in [0, 1]$, from a start configuration $q(0) = q_{\text{start}}$ to a goal configuration $q(1) = q_{\text{goal}}$, without concern for the dynamics, the duration of motion, or constraints on the motion or on the control inputs. It is assumed that the path returned by the path planner can be time scaled to create a feasible trajectory (Chapter 9). This problem is sometimes called the **piano mover's problem**, emphasizing the focus on the geometry of cluttered spaces.

Control inputs: $m = n$ versus $m < n$. If there are fewer control inputs m than degrees of freedom n, then the robot is incapable of following many paths, even if they are collision-free. For example, a car has $n = 3$ (the position and orientation of the chassis in the plane) but $m = 2$ (forward–backward motion and steering); it cannot slide directly sideways into a parking space.

Online versus offline. A motion planning problem requiring an immediate result, perhaps because obstacles appear, disappear, or move unpredictably, calls for a fast, online, planner. If the environment is static then a slower offline planner may suffice.

Optimal versus satisficing. In addition to reaching the goal state, we might want the motion plan to minimize (or approximately minimize) a cost J, e.g.,

$$J = \int_0^T L(x(t), u(t))dt.$$

For example, minimizing with $L = 1$ yields a time-optimal motion while minimizing with $L = u^{\mathrm{T}}(t)u(t)$ yields a "minimum-effort" motion.

Exact versus approximate. We may be satisfied with a final state $x(T)$ that is sufficiently close to x_{goal}, e.g., $\|x(T) - x_{\text{goal}}\| < \epsilon$.

With or without obstacles. The motion planning problem can be challenging even in the absence of obstacles, particularly if $m < n$ or optimality is desired.

10.1.2 Properties of Motion Planners

Planners must conform to the properties of the motion planning problem as outlined above. In addition, planners can be distinguished by the following properties.

Multiple-query versus single-query planning. If the robot is being asked to solve a number of motion planning problems in an unchanging environment, it may be worth spending the time building a data structure that accurately represents $\mathcal{C}_{\text{free}}$. This data structure can then be searched to solve multiple planning queries efficiently. Single-query planners solve each new problem from scratch.

"Anytime" planning. An anytime planner is one that continues to look for better solutions after a first solution is found. The planner can be stopped at any time, for example when a specified time limit has passed, and the best solution returned.

Completeness. A motion planner is said to be **complete** if it is guaranteed to find a solution in finite time if one exists, and to report failure if there is no feasible motion plan. A weaker concept is **resolution completeness**. A planner is resolution complete if it is guaranteed to find a solution if one exists at the resolution of a discretized representation of the problem, such as the resolution of a grid representation of $\mathcal{C}_{\text{free}}$. Finally, a planner is **probabilistically complete** if the probability of finding a solution, if one exists, tends to 1 as the planning time goes to infinity.

Computational complexity. The computational complexity refers to characterizations of the amount of time the planner takes to run or the amount of memory it requires. These are measured in terms of the description of the planning problem, such as the dimension of the C-space or the number of vertices in the representation of the robot and obstacles. For example, the time for a planner to run may be exponential in n, the dimension of the C-space. The computational complexity may be expressed in terms of the average case or the worst case. Some planning algorithms lend themselves easily to computational complexity analysis, while others do not.

10.1.3 Motion Planning Methods

There is no single planner applicable to all motion planning problems. Below is a broad overview of some of the many motion planners available. Details are left to the sections indicated.

Complete methods (Section 10.3). These methods focus on exact representations of the geometry or topology of $\mathcal{C}_{\text{free}}$, ensuring completeness. For all but simple or low-degree-of-freedom problems, these representations are mathematically or computationally prohibitive to derive.

Grid methods (Section 10.4). These methods discretize $\mathcal{C}_{\text{free}}$ into a grid and search the grid for a motion from q_{start} to a grid point in the goal region. Modifications of the approach may discretize the state space or control space or they may use multi-scale grids to refine the representation of $\mathcal{C}_{\text{free}}$ near obstacles. These methods are relatively easy to implement and can return optimal solutions but, for a fixed resolution, the memory required to store the grid, and the time to search it, grow exponentially with the number of dimensions of the space. This limits the approach to low-dimensional problems.

Sampling methods (Section 10.5). A generic sampling method relies on a random or deterministic function to choose a sample from the C-space or state space; a function to evaluate whether the sample is in $\mathcal{X}_{\text{free}}$; a function to determine the "closest" previous free-space sample; and a local planner to try to connect to, or move toward, the new sample from the previous sample. This process builds up a graph or tree representing feasible motions of the robot. Sampling methods are easy to implement, tend to be probabilistically complete, and can even solve high-degree-of-freedom motion planning problems. The solutions tend to be satisficing, not optimal, and it can be difficult to characterize the computational complexity.

Virtual potential fields (Section 10.6). Virtual potential fields create forces on the robot that pull it toward the goal and push it away from obstacles. The approach is relatively easy to implement, even for high-degree-of-freedom systems, and fast to evaluate, often allowing online implementation. The drawback is local minima in the potential function: the robot may get stuck in configurations where the attractive and repulsive forces cancel but the robot is not at the goal state.

Nonlinear optimization (Section 10.7). The motion planning problem can be converted to a nonlinear optimization problem by representing the path or controls by a finite number of design parameters, such as the coefficients of a polynomial or a Fourier series. The problem is to solve for the design parameters that minimize a cost function while satisfying constraints on the controls, obstacles, and goal. While these methods can produce near-optimal solutions, they require an initial guess at the solution. Because the objective function and feasible solution space are generally not convex, the optimization process can get stuck far away from a feasible solution, let alone an optimal solution.

Smoothing (Section 10.8). Often the motions found by a planner are jerky.

A smoothing algorithm can be run on the result of the motion planner to improve the smoothness.

A major trend in recent years has been toward sampling methods, which are easy to implement and can handle high-dimensional problems.

10.2 Foundations

Before discussing motion planning algorithms, we establish concepts used in many of them: configuration space obstacles, collision detection, graphs, and graph search.

10.2.1 Configuration Space Obstacles

Determining whether a robot at a configuration q is in collision with a known environment generally requires a complex operation involving a CAD model of the environment and robot. There are a number of free and commercial software packages that can perform this operation, and we will not delve into them here. For our purposes, it is enough to know that the workspace obstacles partition the configuration space $\mathcal{C}$ into two sets, the **free space** $\mathcal{C}_{\text{free}}$ and the **obstacle space** $\mathcal{C}_{\text{obs}}$, where $\mathcal{C} = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}}$. Joint limits are treated as obstacles in the configuration space.

With the concepts of $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$, the path planning problem reduces to the problem of finding a path for a point robot among the obstacles $\mathcal{C}_{\text{obs}}$. If the obstacles break $\mathcal{C}_{\text{free}}$ into separate **connected components**, and q_{start} and q_{goal} do not lie in the same connected component, then there is no collision-free path.

The explicit mathematical representation of a C-obstacle can be exceedingly complex, and for that reason C-obstacles are rarely represented exactly. Despite this, the concept of C-obstacles is very important for understanding motion planning algorithms. The ideas are best illustrated by examples.

10.2.1.1 A 2R Planar Arm

Figure 10.2 shows a 2R planar robot arm, with configuration $q = (\theta_1, \theta_2)$, among obstacles A, B, and C in the workspace. The C-space of the robot is represented by a portion of the plane with $0 \leq \theta_1 < 2\pi$, $0 \leq \theta_2 < 2\pi$. Remember from Chapter 2, however, that the topology of the C-space is a torus (or doughnut) since the edge of the square at $\theta_1 = 2\pi$ is connected to the edge $\theta_1 = 0$; similarly, $\theta_2 = 2\pi$ is connected to $\theta_2 = 0$. The square region of $\mathbb{R}^2$ is obtained by slicing