11 *Trajectory Planning*

IN CHAPTER 10 we described dynamic models for robot systems. Equipped with such a dynamic model, the trajectory planning problem is to find control (force) inputs $u(t)$ yielding a trajectory $q(t)$ that avoids obstacles, takes the system to the desired goal state, and perhaps optimizes some objective function while doing so. This can be considered a complete "motion-planning" problem, as opposed to a "path-planning" problem that only asks for a feasible curve $q(s)$ in the configuration space, without reference to the speed of execution.

In this chapter we study two approaches to trajectory planning for a dynamic system: the decoupled approach, which involves first searching for a path in the configuration space and then finding a time-optimal time scaling for the path subject to the actuator limits; and the direct approach, where the search takes place in the system's state space. Examples of the latter approach include optimal control and numerical optimization, grid-based searches, and randomized probabilistic methods. In this chapter we focus on fully actuated systems—systems where there is an actuator for each degree of freedom.

In section 11.1 we provide some definitions used throughout the chapter. In section 11.2 we describe an algorithm for finding the time-optimal execution of a path subject to actuator limits, and describe how this can be used in a decoupled trajectory planner. In section 11.3 we outline several approaches to trajectory planning directly in the system state space, including optimal control, gradient-based numerical methods, and dynamic programming. Other methods, such as rapidly-exploring random trees (RRTs), are described in chapter 7.

11.1 Preliminaries

In this chapter a path $q(s)$ is assumed to be a twice-differentiable curve on the configuration space $\mathcal{Q}$, $q : [0, 1] \to \mathcal{Q}$. A *time scaling* $s(t)$ is a function $s : [0, t_f] \to [0, 1]$ assigning an s value to each time $t \in [0, t_f]$. Together, a path and a time scaling specify a trajectory $q(s(t))$, or $q(t)$ for short. The time scaling $s(t)$ should be twice-differentiable and monotonic ($\dot{s}(t) > 0$ for all $t \in (0, t_f)$). The twice-differentiability of $s(t)$ ensures that the acceleration $\ddot{q}(t)$ is well defined and bounded. Note that *uniform* time scalings $s(t) = kt$ are a subset of the more general time-scaling functions considered here.

Configurations q and forces u are both $n_{\mathcal{Q}}$-dimensional vectors.

11.2 Decoupled Trajectory Planning

Given a collision-free path $q(s)$ for a robot system, what is the fastest feasible trajectory that follows this path? In other words, what is the time-optimal time scaling $s(t)$ subject to actuator constraints? This question is of considerable importance for maximizing the productivity of robot systems when a path has been given by task specifications or found by a path planner. This problem has been solved elegantly by Shin and McKay [385] and Bobrow, Dubowsky, and Gibson [51], with subsequent enhancements by Pfeiffer and Johanni [348], Slotine and Yang [388], and Shiller and Lu [384].

Let us assume that the equations of motion of our system are in the standard form of equation (10.7) or equation (10.10) from chapter 10. The robot is subject to the actuator limits

$$u_i^{\min}(q, \dot{q}) \leq u_i \leq u_i^{\max}(q, \dot{q}). \tag{11.1}$$

In general, the actuator limits may be functions of the system configuration and velocity. For example, the torque available to accelerate a DC motor decreases as its angular velocity increases. The simplest example of actuator limits are the symmetric, state-independent bounds

$$|u_i| \leq u_i^{\max}.$$

For a given path $q(s)$, we can substitute

$$\frac{dq}{ds}\dot{s} = \dot{q} \tag{11.2}$$

$$\frac{d^2q}{ds^2}\dot{s}^2 + \frac{dq}{ds}\ddot{s} = \ddot{q} \tag{11.3}$$

into equation (10.10) to get

$$M(q(s))\left(\frac{d^2q}{ds^2}\dot{s}^2+\frac{dq}{ds}\ddot{s}\right)+\left(\frac{dq}{ds}\dot{s}\right)^T\Gamma(q(s))\left(\frac{dq}{ds}\dot{s}\right)+g(q(s))=u \tag{11.4}$$

or

$$\left(M(q(s))\frac{dq}{ds}\right)\ddot{s}+\left(M(q(s))\frac{d^2q}{ds^2}+\left(\frac{dq}{ds}\right)^T\Gamma(q(s))\frac{dq}{ds}\right)\dot{s}^2+g(q(s))=u. \tag{11.5}$$

These equations can be expressed compactly as the vector equation

$$a(s)\ddot{s}+b(s)\dot{s}^2+c(s)=u \tag{11.6}$$

defining the dynamics constrained to the path $q(s)$. The vector functions $a(s)$, $b(s)$, and $c(s)$ are inertial, velocity product, and gravitational terms in terms of s, respectively.

As the robot travels along the path $q(s)$, its state at any time is identified by $(s, \dot{s})$. Actuator limits can be expressed as a function of the path state by substituting equation (11.2) into equation (11.1), yielding $u^{\min}(s, \dot{s})$ and $u^{\max}(s, \dot{s})$. Therefore, at all times the system must satisfy the constraints

$$u^{\min}(s,\dot{s})\le a(s)\ddot{s}+b(s)\dot{s}^2+c(s)\le u^{\max}(s,\dot{s}). \tag{11.7}$$

Let $L_i(s, \dot{s})$ and $U_i(s, \dot{s})$ be the minimum and maximum accelerations $\ddot{s}$ satisfying the ith component of equation (11.7), and define

$$\alpha_i(s,\dot{s})=\frac{u_i^{\max}(s,\dot{s})-b_i(s)\dot{s}^2-c_i(s)}{a_i(s)},\quad \beta_i(s,\dot{s})=\frac{u_i^{\min}(s,\dot{s})-b_i(s)\dot{s}^2-c_i(s)}{a_i(s)}. \tag{11.8}$$

Then $U_i(s, \dot{s}) = \alpha_i(s, \dot{s})$, $L_i(s, \dot{s}) = \beta_i(s, \dot{s})$ if $a_i(s) > 0$, and $U_i(s, \dot{s}) = \beta_i(s, \dot{s})$, $L_i(s, \dot{s}) = \alpha_i(s, \dot{s})$ if $a_i(s) < 0$. (If $a_i(s) = 0$ for any i, the system is at a *zero inertia point,* and we will set aside this possibility until subsection 11.2.1.) We define

$$L(s,\dot{s})=\max_{i\in 1\ldots n_{\mathcal{Q}}}L_i(s,\dot{s}),\quad U(s,\dot{s})=\min_{i\in 1\ldots n_{\mathcal{Q}}}U_i(s,\dot{s}).$$

The actuator limits (11.7) can then be expressed as

$$L(s,\dot{s})\le\ddot{s}\le U(s,\dot{s}). \tag{11.9}$$

The problem can now be stated:

> Given a path $q : [0, 1] \to \mathcal{Q}$, an initial state $(0, \dot{s}_0)$, and a final state $(1, \dot{s}_f)$, $\dot{s}_0, \dot{s}_f \ge 0$, find a monotonically increasing twice-differentiable time scaling $s : [0, t_f] \to [0, 1]$ that (1) satisfies $s(0) = 0$, $\dot{s}(0) = \dot{s}_0$, $s(t_f) = 1$, $\dot{s}(t_f) = \dot{s}_f$, and (2) minimizes the total travel time t_f along the path while respecting the actuator constraints (11.9) for all time $t \in [0, t_f]$.

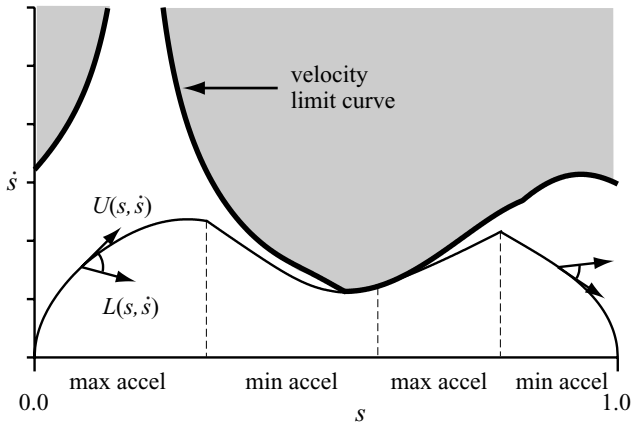

Figure 11.1 At each point $(s, \dot{s})$ in the phase plane we can draw a motion cone defined by the maximum and minimum accelerations $\ddot{s}$ satisfying the actuator limits. The time-optimal trajectory from $(0, \dot{s}_0)$ to $(1, \dot{s}_f)$ is the curve that maximizes the area underneath it while remaining on the boundary of the motion cones. In this example, the trajectory switches between maximum and minimum acceleration three times. The velocity limit curve indicates the states where the cone collapses to a single tangent vector, and the gray region represents inadmissible states where the cone disappears—no feasible actuation will keep the system on the path.

We can conveniently visualize this problem in the $(s, \dot{s})$ state space. At any state $(s, \dot{s})$, the constraints (11.9) specify a range of feasible accelerations along the path, $L(s, \dot{s}) \leq \ddot{s} \leq U(s, \dot{s})$. This range can be interpreted as a cone of tangent vectors in the state space, as illustrated in figure 11.1. The problem is to find a curve from $(0, \dot{s}_0)$ to $(1, \dot{s}_f)$ such that $\dot{s} \geq 0$ everywhere and the tangent at each state is inside the cone at that state. Further, the curve should maximize the speed $\dot{s}$ at each s to minimize the time of motion. A consequence of this is that the curve always follows the upper or lower bound of the cone (maximum or minimum acceleration) at each state.[1] This kind of trajectory is called a "bang-bang" trajectory, and at least one of the actuators is always saturated. The heart of the time-scaling problem is to find the switching points between maximum and minimum acceleration.

At some states $(s, \dot{s})$, the actuation constraints (11.9) indicate that there is no feasible acceleration that will allow the system to continue to follow the path. Such regions of the state space are shown in gray in figure 11.1. We will call these regions *inadmissible* regions. At any inadmissible state, the robot is doomed to leave the path immediately. At admissible states, the robot may still be doomed to eventually leave

1. Except perhaps at zero inertia points, as described in subsection 11.2.1.

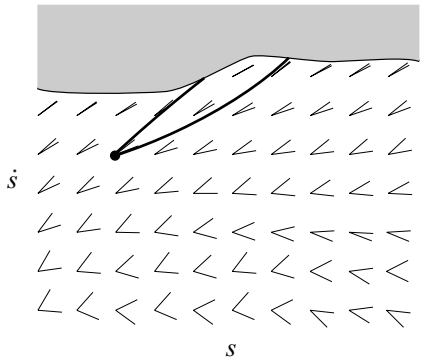

Figure 11.2 Beginning from the $(s, \dot{s})$ state represented by the dot, the system is doomed to leave the path. The set of all trajectories tangent to the motion cones is bounded by the two integral curves shown, showing that all feasible trajectories penetrate the velocity limit curve.

the path. This happens if any integral curve originating from the state, with tangents remaining inside the tangent cones, eventually must reach the inadmissible region (figure 11.2).

We will assume that, for any s, the robot is strong enough to maintain its configuration statically, so all $\dot{s} = 0$ states are admissible and the path can be executed arbitrarily slowly. We will also assume that as $\dot{s}$ increases from zero for a given s, there will be at most one switch from admissible to inadmissible. This occurs at the *velocity limit curve* $v(s)$, consisting of states $(s, \dot{s})$ satisfying

$$L(s, \dot{s}) = U(s, \dot{s}). \tag{11.10}$$

The velocity limit $v(s)$ is obtained by equating $L_i(s, \dot{s}) = U_j(s, \dot{s})$ for all $i, j = 1 \ldots n_{\mathcal{Q}}$ and solving each equation for $\dot{s}$ (if a solution exists). Call the solution $\dot{s}_{ij}(s)$. For all i, j, keep the minimum value: $v(s) = \min_{i,j} \dot{s}_{ij}(s)$.[2]

Note that because of the max$(\cdot)$ and min$(\cdot)$ functions used in calculating $L(s, \dot{s})$, $U(s, \dot{s})$, and $v(s)$, these functions are generally not smooth.

As mentioned earlier, the problem is to find the switches between maximum and minimum acceleration. The following algorithm uses numerical integration to find the set of switches, expressed as the s values at which the switches occur.

2. In general, equation (11.10) may be satisfied for multiple values of $\dot{s}$ for a single value of s. This may occur due to friction in the system, weak actuators that cannot hold each configuration statically, or the form of the actuation limit functions. In this case, there may be inadmissible "islands" in the phase plane. This significantly complicates the problem of finding an optimal time scaling, and we ignore this possibility. See [385] for a time-scaling algorithm for this case.

Time-Scaling Algorithm

1. Initialize an empty list of switches $\mathcal{S} = \{\}$ and a switch counter $i = 0$. Set $(s_i, \dot{s}_i) = (0, \dot{s}_0)$.

2. Integrate the equation $\ddot{s} = L(s, \dot{s})$ *backward* in time from $(1, \dot{s}_f)$ until the velocity limit curve is penetrated (reached transversally, not tangentially), $s = 0$, or $\dot{s} = 0$ at $s < 1$. There is no solution to the problem if $\dot{s} = 0$ is reached. Otherwise, call this phase plane curve F and proceed to the next step.

3. Integrate the equation $\ddot{s} = U(s, \dot{s})$ forward in time from $(s_i, \dot{s}_i)$. Call this curve A_i. Continue integrating until either A_i crosses F or A_i penetrates the velocity limit curve. (If A_i crosses $s = 1$ or $\dot{s} = 0$ before either of these two cases occurs, there is no solution to the problem.) If A_i crosses F, then increment i, let s_i be the s value at which the crossing occurs, and append s_i to the list of switches $\mathcal{S}$. The problem is solved and $\mathcal{S}$ is the solution. If instead the velocity limit curve is penetrated, let $(s_{\text{lim}}, \dot{s}_{\text{lim}})$ be the point of penetration and proceed to the next step.

4. Search the velocity limit curve $v(s)$ forward in s from $(s_{\text{lim}}, \dot{s}_{\text{lim}})$ until finding the first point where the feasible acceleration ($L = U$ on the velocity limit curve) is tangent to the velocity limit curve. If the velocity limit is $v(s)$, then a point $(s_0, v(s_0))$ satisfies the tangency condition if $\frac{dv}{ds}|_{s=s_0} = U(s_0, v(s_0))/v(s_0)$. Call the first tangent point reached $(s_{\text{tan}}, \dot{s}_{\text{tan}})$.[3] From $(s_{\text{tan}}, \dot{s}_{\text{tan}})$, integrate the curve $\ddot{s} = L(s, \dot{s})$ backward in time until it intersects A_i. Increment i and call this new curve A_i. Let s_i be the s value of the intersection point. This is a switch point from maximum to minimum acceleration. Append s_i to the list $\mathcal{S}$.

5. Increment i and set $(s_i, \dot{s}_i) = (s_{\text{tan}}, \dot{s}_{\text{tan}})$. This is a switch point from minimum to maximum acceleration. Append s_i to the list $\mathcal{S}$. Go to step 3.

An illustration of the time-scaling algorithm is shown in figure 11.3.

11.2.1 Zero Inertia Points

Until now, we have been making the assumption that each $a_i(s)$ in (11.8) is always nonzero. In this usual case, the velocity limit occurs when $L(s, \dot{s}) = U(s, \dot{s})$. If

3. An alternative approach to finding $(s_{\text{tan}}, \dot{s}_{\text{tan}})$ is to choose a point $(s_{\text{lim}}, \dot{s}')$, where $\dot{s}' < \dot{s}_{\text{lim}}$, integrate L forward from there, and check if the solution penetrates the velocity limit curve. If so, choose $\dot{s}'' < \dot{s}'$; if not, choose $\dot{s}'' > \dot{s}'$. Perform the integration of L from $(s_{\text{lim}}, \dot{s}'')$. Repeat the binary search until the forward integration just touches the velocity limit curve tangentially. The tangent point is $(s_{\text{tan}}, \dot{s}_{\text{tan}})$.

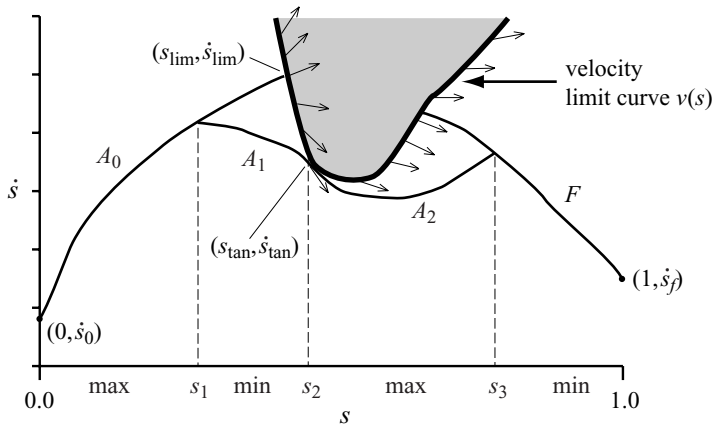

Figure 11.3 An illustration of the time-scaling algorithm. (Step 2) Beginning from $(1, \dot{s}_f)$, the minimum acceleration is integrated backward until the velocity limit curve is reached. The resulting phase plane curve is denoted F. (Step 3) Beginning from $(0, \dot{s}_0)$, the maximum acceleration is integrated forward until the velocity limit curve is reached at $(s_{\text{lim}}, \dot{s}_{\text{lim}})$. This phase plane curve is denoted A_0. (Step 4) The velocity limit curve $v(s)$ is searched forward from s_{lim} until a point is found where the feasible acceleration is tangent to the limit curve. The figure shows the feasible accelerations at points on the velocity limit curve as arrows. An arrow becomes tangent to the velocity limit curve at $(s_{\text{tan}}, \dot{s}_{\text{tan}})$. From $(s_{\text{tan}}, \dot{s}_{\text{tan}})$, the minimum acceleration is integrated backward until it reaches A_0. This curve is called A_1. The s value of the intersection is s_1 and is added to the switch list, $\mathcal{S} = \{s_1\}$. (Step 5) The point $(s_2, \dot{s}_2)$ is set equal to $(s_{\text{tan}}, \dot{s}_{\text{tan}})$, and s_2 is added to the switch list, $\mathcal{S} = \{s_1, s_2\}$. (Step 2) Maximum acceleration is integrated forward from $(s_2, \dot{s}_2)$ until it hits F. This curve is denoted A_2, and the s value of the intersection, s_3, is added to the switch list, yielding $\mathcal{S} = \{s_1, s_2, s_3\}$. The algorithm terminates.

$a_i(s) = 0$, however, the force at the ith actuator is independent of $\ddot{s}$, and therefore the ith actuator defines no acceleration constraints $L_i(s, \dot{s})$, $U_i(s, \dot{s})$. Instead, it defines directly a *velocity* constraint using (11.7):

$$u_i^{\min}(s, \dot{s}) \leq b_i(s)\dot{s}^2 + c_i(s) \leq u_i^{\max}(s, \dot{s}). \tag{11.11}$$

In the case of a zero inertia point where k of the $a_i(s)$ are zero, let $\dot{s}_{\text{zip}}^{\max}(s)$ be the maximum velocity satisfying all k constraints (11.11), and let $\dot{s}^{\max}(s) = \min(v(s), \dot{s}_{\text{zip}}^{\max}(s))$. Then $\dot{s}^{\max}(s)$ is the *true* velocity limit curve, generalizing the curve $v(s)$ by allowing for the possibility of zero inertia points.

If a point on the velocity limit curve $\dot{s}^{\max}(s)$ is determined by a zero inertia point velocity constraint, then $L(s, \dot{s}) < U(s, \dot{s})$ at this point. This point is called a *critical* point. If, in addition, either $U(s, \dot{s})$ integrated forward from this point, or $L(s, \dot{s})$

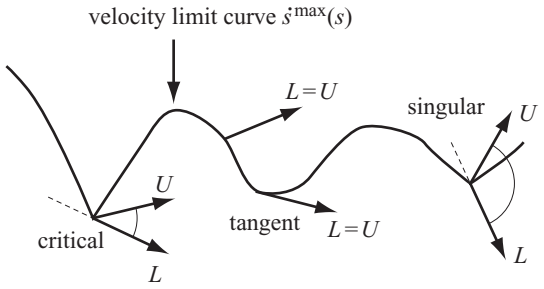

Figure 11.4 At critical and singular points, $U(s, \dot{s}) \neq L(s, \dot{s})$. At a singular point, following U forward or L backward results in penetration of the velocity limit curve.

integrated backward from this point, would result in immediate penetration of the velocity limit curve, then the point is called *singular* (figure 11.4).

At a singular point $(s_s, \dot{s}^{\max}(s_s))$, let

$$\ddot{s}_{\text{tangent}} = \dot{s}^{\max}(s_s)\frac{d\dot{s}^{\max}}{ds}|_{s=s_s}$$

be the acceleration defined by the tangent to the velocity limit curve. If the velocity limit curve is not differentiable at the singular point (as is often the case), then define

$$\ddot{s}^{+}_{\text{tangent}} = \dot{s}^{\max}(s_s)\frac{d\dot{s}^{\max}}{ds}|_{s=s_s^{+}}$$

$$\ddot{s}^{-}_{\text{tangent}} = \dot{s}^{\max}(s_s)\frac{d\dot{s}^{\max}}{ds}|_{s=s_s^{-}}$$

to be the right and left limits, respectively. Then, to prevent penetration of the velocity limit curve, the maximum feasible acceleration at $(s_s, \dot{s}^{\max}(s_s))$ is

$$\ddot{s}^{\max} = \min(\ddot{s}^{+}_{\text{tangent}}, U).$$

Similarly, the minimum feasible acceleration is

$$\ddot{s}^{\min} = \max(\ddot{s}^{-}_{\text{tangent}}, L).$$

This is shown graphically in figure 11.5.

Critical points occur on a lower-dimensional subset of the robot's configuration space where $M(q)$ is not full rank. If the path passes through this subset transversally, the path will have isolated critical points. If the path travels along this lower-dimensional subset, however, we may have a continuous *critical arc* of critical points. Similarly, we may have *singular arcs*.

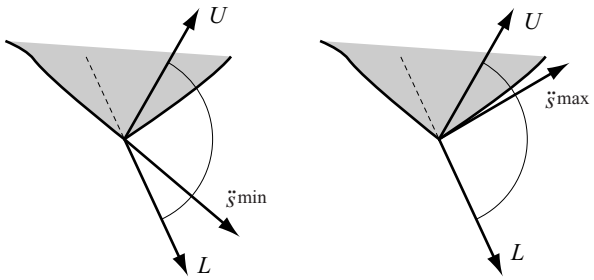

Figure 11.5 At this singular point on the velocity limit curve, integrating $L(s, \dot{s})$ backward in time would result in penetration of the velocity limit curve. The true minimum feasible acceleration at this point is $\ddot{s}^{\min}$, defined by the left tangent of the velocity limit curve at the singularity. Similarly, integrating $U(s, \dot{s})$ forward results in penetration. The true maximum feasible acceleration is $\ddot{s}^{\max}$, defined by the right tangent at the singularity.

We can now modify the time-scaling algorithm to properly account for zero inertia points. At singular points, we integrate $\ddot{s}^{\min}(s, \dot{s})$ and $\ddot{s}^{\max}(s, \dot{s})$ instead of $L(s, \dot{s})$ and $U(s, \dot{s})$, respectively. This will also allow the algorithm to "slide" along singular arcs using an acceleration between $L(s, \dot{s})$ and $U(s, \dot{s})$, instead of switching rapidly back and forth between them. In step 4, we search the velocity limit curve $\dot{s}^{\max}(s, \dot{s})$ for any critical or tangent point, not just tangent points.

EXAMPLE 11.2.1 *Consider the two-joint RP robot arm with the dynamics derived in chapter 10. We have planned a path to follow the straight line shown in figure 11.6, and we wish to find the time-optimal time scaling of the path. Let* $x = [x_1, x_2]^T$ *be the Cartesian coordinates of the center of mass of the second link, as shown in the figure. The path we wish to follow, parameterized by* s*, is expressed as*

$$x(s) = [x_1(s), x_2(s)]^T = [2s - 1, 1]^T, \quad s \in [0, 1]. \tag{11.12}$$

The first thing we will do is express this path in the generalized coordinates $q = [q_1, q_2]^T$ *of figure 10.2 in chapter 10. To do this, we can define the forward kinematics of the robot arm to be the mapping* ϕ *from joint coordinates* q *to Cartesian coordinates* x*:*

$$\begin{aligned} x &= \phi(q) \\ [x_1, x_2]^T &= [q_2 \cos q_1, q_2 \sin q_1]^T \end{aligned} \tag{11.13}$$

The inverse kinematics ϕ^{-1} *gives the joint coordinates* q *as a function of the Cartesian coordinates* x*. For the RP arm, the inverse kinematics are unique within the reachable*

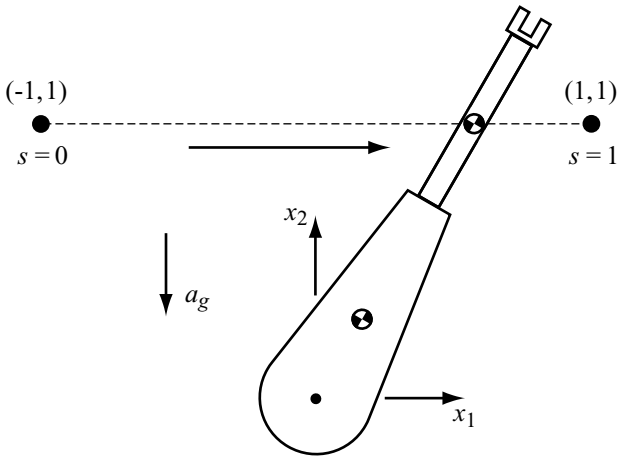

Figure 11.6 The path followed by the RP manipulator.

workspace of the robot (*where* $q_2 \geq 0$)*:*

$$q = \phi^{-1}(x)$$

$$[q_1, q_2]^T = \left[\text{atan2}(x_2, x_1), \sqrt{x_1^2 + x_2^2}\right]^T \tag{11.14}$$

where $\text{atan2}(x_2, x_1)$ *is the two argument arctangent returning the unique angle in* $[-\pi, \pi)$ *of the Cartesian point* (x_1, x_2) (*where* $x_1^2 + x_2^2 \neq 0$)*. Plugging equation (11.12) into equation (11.14), we find the parameterized path in joint coordinates*

$$q(s) = \left[\text{atan2}(1, 2s - 1), \sqrt{4s^2 - 4s + 2}\right]^T. \tag{11.15}$$

Differentiating, we get the velocity and acceleration

$$\dot{q} = \begin{bmatrix} \frac{-\dot{s}}{2s^2-2s+1} \\ \frac{(4s-2)\dot{s}}{\sqrt{4s^2-4s+2}} \end{bmatrix} \tag{11.16}$$

$$\ddot{q} = \begin{bmatrix} \frac{(4s-2)\dot{s}^2+(-2s^2+2s-1)\ddot{s}}{(2s^2-2s+1)^2} \\ \frac{\sqrt{2}(\dot{s}^2+(4s^3-6s^2+4s-1)\ddot{s})}{(2s^2-2s+1)^{3/2}} \end{bmatrix}. \tag{11.17}$$

In chapter 10 we derived the equations of motion:

$$u_1 = \left(I_1 + I_2 + m_1 r_1^2 + m_2 q_2^2\right)\ddot{q}_1 + 2m_2 q_2 \dot{q}_1 \dot{q}_2$$
$$+ a_g(m_1 r_1 + m_2 q_2)\cos q_1 \tag{11.18}$$

$$u_2 = m_2 \ddot{q}_2 - m_2 q_2 \dot{q}_1^2 + a_g m_2 \sin q_1 \tag{11.19}$$

Substituting in equations (11.15), (11.16), and (11.17), we get

$$a(s)\ddot{s} + b(s)\dot{s}^2 + c(s) = u,$$

where

$$a(s) = \begin{bmatrix} \frac{I_1+I_2+2m_2+m_1r_1^2-4m_2s+4m_2s^2}{-2s^2+2s-1} \\ \frac{\sqrt{2}m_2(2s-1)}{\sqrt{2s^2-2s+1}} \end{bmatrix} \tag{11.20}$$

$$b(s) = \begin{bmatrix} \frac{2(I_1+I_2+m_1r_1^2)(2s-1)}{(2s^2-2s+1)^2} \\ 0 \end{bmatrix} \tag{11.21}$$

$$c(s) = \begin{bmatrix} (m_1r_1 + m_2\sqrt{4s^2-4s+2})a_g \cos(\text{atan2}(1, 2s-1)) \\ m_2a_g \sin(\text{atan2}(1, 2s-1)) \end{bmatrix}. \tag{11.22}$$

Note that $s = \frac{1}{2}$ defines a zero inertia point, as $a_2(\frac{1}{2}) = 0$. (Understand this intuitively by considering the $s = \frac{1}{2}$ point in figure 11.6.) In this case there is no velocity constraint due to the second actuator, since $b_2 = 0$ (see problem 5).

We now choose the following parameters for the robot arm: $m_1 = 5$ kg, $I_1 = 0.1$ kg-m^2, $r_1 = 0.2$ m, $m_2 = 3$ kg, and $I_2 = 0.05$ kg-m^2. The actuator limits are taken to be ± 20 N-m for joint 1 and ± 40 N for joint 2, and gravity is $a_g = 9.8$ m/s^2. Figure 11.7 shows the time-optimal trajectory along the path for these choices. The minimum-time execution of the path is approximately 0.888 s. Note that one of the actuators is saturated at all times. In this example, the velocity limit curve is never reached, so there is just one switch between maximum and minimum acceleration.

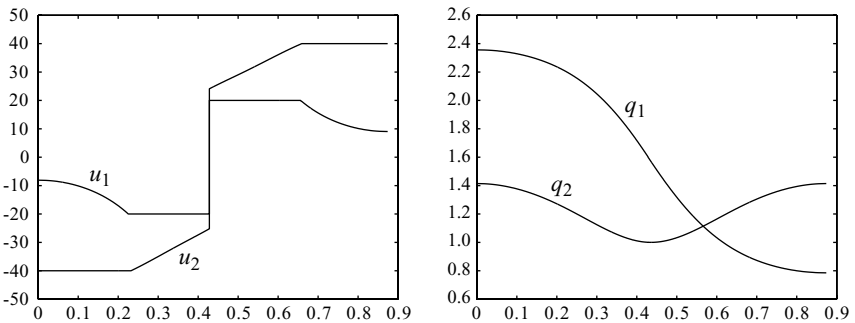

Figure 11.7 The time-optimal actuator histories and trajectory of the RP manipulator along the straight-line path of example 11.2.1.

11.2.2 Global Time-Optimal Trajectory Planning

The time-scaling algorithm finds the time-optimal trajectory along a given path. What if our real goal is to find the time-optimal trajectory between two states when we are free to choose any collision-free path? Can we use the time-scaling algorithm in conjunction with a collision-free path planner? Conceptually, imagine running the time-scaling algorithm on all possible paths between the start and goal states. Then the fastest of these is the global time-optimal trajectory.

Naturally, the problem is how to efficiently test a large number of possible paths. One approach to this problem for robot manipulators was proposed by Shiller and Dubowsky [383]. The approach is quite involved, and we only sketch it here. The first step is to define a grid on the workspace and construct all collision-free paths (without sharp turns) between the start and goal states moving along edges or diagonals of the grid. The next step is to quickly compute rough lower-bound estimates of the traveling times of these paths using a maximum velocity limit during the motion. The fastest paths are selected and smoothed by using the grid points as control points for cubic splines. The best of these paths is then submitted to the full time-scaling algorithm, generating an upper bound on the optimal travel time. All paths with lower bounds above this upper bound can be pruned. Of the remaining paths, the lower bounds are more carefully calculated, and the process continues, using increasingly accurate estimates of the lower bounds as the number of candidate paths is reduced. When the pruning process has ended, only the best path in each path-neighborhood is considered further. These best paths are submitted to a local optimization that may locally alter the paths to allow them to be executed more quickly. This process uses the travel times returned by the time-scaling algorithm as the objective function.

This approach combines collision-free path planning and time scaling in an iterative fashion to arrive at a global near-time-optimal trajectory. In the next section, we discuss methods that do not separate the path-planning and time-scaling problems, but solve directly in the state space.

11.3 Direct Trajectory Planning

In section 11.2, trajectory planning is decoupled into collision-free path planning followed by time scaling. In this section, we study methods for planning the trajectory directly in the state space. If we are interested in finding trajectories that optimize some cost function, such as motion time or expended energy, optimal control theory provides necessary conditions on the trajectories. Unfortunately, these conditions are complex for almost any robot system and cannot be solved analytically. Because of

this, we consider two numerical approaches: nonlinear optimization and grid-based search.

First we describe how to transform the optimal control problem to a finite-dimensional parameter optimization problem, allowing nonlinear optimization to be used to numerically solve the optimality conditions. If the problem is well formulated (e.g., the objective and constraint functions are sufficiently smooth), nonlinear optimization may result in rapid convergence to a locally optimal trajectory. The drawbacks of this approach are that the method requires an initial guess (possibly provided by a grid-based search method), and the locally optimal solution reached generally depends heavily on this guess. Also, evaluation of constraint and objective functions, and their gradients, may be computationally costly.

We then introduce a grid-based search method that allows the user to specify how near to time-optimal the motion plan should be, while meeting "safety" requirements on obstacle avoidance. The planned motions are approximate in that the goal state may not be exactly reached, but the user has control over how large the final error can be. An advantage is that this is a global approach, unlike gradient-based nonlinear optimization. A drawback of grid-based search is that the size of the grid grows exponentially in the dimension of the state space, making the approach impractical for high-dimensional systems.

A third approach is to use RRT's, as described in chapter 7. This approach trades off optimality for planner run-time—it may be able to quickly find a feasible trajectory that is in no sense optimal.

Finally, a fourth approach is based on artificial potential fields (see chapter 4). An artificial potential field is constructed to make obstacles repulsive and the goal configuration attractive. The robot senses its current configuration, calculates the gradient of the artificial potential at this configuration, and applies the gradient forces at the actuators. (Damping forces may be included to stabilize the goal configuration.) A potential field, therefore, implicitly defines a trajectory to the goal from all initial states. This approach is fundamentally different from the previous "open-loop" approaches in that a feedback law is specified for all robot states.

11.3.1 Optimal Control

Given a dynamic system

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = u, \tag{11.23}$$

we would like to find a solution $(q(t), u(t))$, $t \in [0, t_f]$ to equation (11.23) that avoids obstacles and joint limits, respects actuator limits, and takes the system from the initial state $(q_{\text{start}}, \dot{q}_{\text{start}})$ at time $t = 0$ to the final state $(q_{\text{goal}}, \dot{q}_{\text{goal}})$ at time $t = t_f$. Of all

the trajectories that accomplish this, we might want to find one that minimizes some objective function J. In general, J might be a function of the controls, the trajectory, and the total motion time t_f, i.e., $J = J(u(t), q(t), t_f)$.

Before proceeding further, let's express the state of the system as $x = [q^T, \dot{q}^T]^T$ and rewrite the equations of motion in the general form

$$\dot{x}(t) = f(x(t), u(t), t), \tag{11.24}$$

where f is the vector differential equation describing the kinematics and dynamics of the system. In our case, the state equations do not change with time, so $f(x, u, t) = f(x, u)$. The state equations (11.24) can be viewed as constraints defining the relationship between $x(t)$ and $u(t)$. Let the objective function J be written

$$J = \int_0^{t_f} \mathcal{L}(x(t), u(t), t)dt, \tag{11.25}$$

where the integrand $\mathcal{L}$ is called the *Lagrangian*. (The Lagrangian $\mathcal{L}$ for optimal control is actually a generalization of the Lagrangian L for dynamics.) A typical choice of $\mathcal{L}$ is "effort," modeled as a quadratic function of the control, e.g., $\mathcal{L} = u^T W u$, where W is a positive definite weighting matrix, e.g., the identity matrix. Another common choice is to leave t_f free and choose $\mathcal{L} = 1$, implying $J = t_f$, a minimum-time problem.

For now, let us ignore the issues of actuator limits and obstacles, and assume that the final time t_f is fixed. The problem then is to find a state and control history $(x(t), u(t))$, $t \in [0, t_f]$ that satisfies the constraints of equation (11.24), satisfies the terminal conditions $x(t_f) = x_f$, and minimizes the cost in equation (11.25). To write a necessary condition for optimality, we define the *Hamiltonian* $\mathcal{H}$

$$\mathcal{H}(x(t), u(t), \lambda(t), t) = \mathcal{L}(x(t), u(t), t) + \lambda^T f(x(t), u(t)), \tag{11.26}$$

where $\lambda(t)$ is a vector of *Lagrange multipliers*.[4] Then the *Pontryagin minimum principle*[5] says that at an optimal solution $(x^*(t), u^*(t), \lambda^*(t))$,

$$\mathcal{H}^*(t) = \mathcal{H}(x^*(t), u^*(t), \lambda^*(t), t) \leq \mathcal{H}(x^*(t), u(t), \lambda^*(t), t). \tag{11.27}$$

In other words, if the control history $u^*(t)$ is optimal, then at any time t, any other feasible control $u(t)$ will give an $\mathcal{H}(t)$ greater than or equal to that of the optimal $\mathcal{H}^*(t)$. In the absence of any other constraints on the state and control, then, a necessary

4. These Lagrange multipliers play a similar role to those in dynamics with constraints—they are used to enforce constraints, here the state equation constraints.

5. Often known as the Pontryagin maximum principle. In our case, we are minimizing a cost function; we could equivalently maximize a utility function, or the negative of the cost function.

condition for optimality can be written

$$\frac{\partial \mathcal{H}}{\partial u} = 0. \tag{11.28}$$

This says that the linear sensitivity of the Hamiltonian to changes in u is zero, meaning that the control is *extremal,* but not necessarily optimal. A sufficient condition for local optimality of a solution is that equation (11.28) is satisfied and the Hessian of the Hamiltonian is positive definite along the trajectory of the solution:

$$\frac{\partial^2 \mathcal{H}}{\partial u^2} > 0 \tag{11.29}$$

This is known as the convexity or Legendre-Clebsch condition. The Lagrange multipliers evolve according to the adjoint equation

$$\dot{\lambda} = -\frac{\partial \mathcal{H}}{\partial x}. \tag{11.30}$$

Equation (11.28) can sometimes be used to write u as a function of x and λ. In this case, optimization boils down to choosing initial conditions for equation (11.30) to ensure that the goal is reached.

EXAMPLE 11.3.1 *Consider a simple double-integrator system with one degree of freedom, $\ddot{q} = u$, such as a point mass on a line actuated by a force. Let $x = [x_1, x_2]^T = [q, \dot{q}]^T$, so the equations of motion can be written in the form*

$$\dot{x} = f(x, u), \quad \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ u \end{bmatrix}.$$

Choose the objective function

$$J = \int_0^{t_f} u^2 dt.$$

Then the Hamiltonian is

$$\mathcal{H} = u^2 + \lambda_1 x_2 + \lambda_2 u$$

and the necessary condition (11.28) is written

$$\frac{\partial \mathcal{H}}{\partial u} = 2u + \lambda_2 = 0. \tag{11.31}$$

The adjoint equation (11.30) is written

$$\dot{\lambda} = -\frac{\partial \mathcal{H}}{\partial x}, \quad \begin{bmatrix} \dot{\lambda}_1 \\ \dot{\lambda}_2 \end{bmatrix} = -\begin{bmatrix} 0 \\ \lambda_1 \end{bmatrix}. \tag{11.32}$$

Equation (11.32) shows that λ_1 is constant and λ_2 is a linear function of time, so by equation (11.31), u is also a linear function of time, e.g., $u(t) = c_0 + c_1 t$.

Now we can specify the initial and final state for the system to solve for the control and state history. Let $x(0) = [0, 0]^T$ and $x(t_f) = [d, 0]^T$, i.e., the system starts at rest and ends at rest having moved a distance d in time t_f. Then we have the stopping conditions that the first and second integral of $u(t)$ evaluated over $[0, t_f]$ be zero and d, respectively:

$$x_2(t_f) = \int_0^{t_f} u(t)dt = 0$$

$$x_1(t_f) = \int_0^{t_f} \int_0^{t} u(\eta)d\eta \, dt = d.$$

Solving, we get $c_0 = 6d/t_f^2$, $c_1 = -12d/t_f^3$ defining the extremal control history. The extremal state history is obtained by integration of the control.

To check if this extremal solution is a minimizer, we can use the convexity condition. In this case, the Hessian is the scalar $\partial^2\mathcal{H}/\partial u^2 = 2 > 0$, indicating that the solution is indeed (locally) optimal.

In the previous example, the convexity condition is satisfied. However, if an extremum is achieved ($\partial\mathcal{H}/\partial u = 0$) but convexity is not ($\partial^2\mathcal{H}/\partial u^2$ is only positive *semidefinite*), it does not mean that the control is not optimal. In this case, auxiliary conditions have to be satisfied to ensure optimality. An optimal control of this type is an example of a *singular* optimal control, as in the previous section with time-optimal control at zero inertia points.

What if there are actuator limits or obstacles? At an optimal solution, the minimum principle (11.27) will always be satisfied, but the control or state history may bump up against limits preventing equation (11.28) from being satisfied. The optimal solution may be constrained by these limits rather than the extremality condition (11.28). Consider the one-degree-of-freedom double integrator above, with the actuator limits $|u| < u^{\max}$, and choose the minimum-time objective function

$$J = \int_0^{t_f} 1dt,$$

where the time t_f is left free. The Hamiltonian is $\mathcal{H} = 1 + \lambda_1 x_2 + \lambda_2 u$, and $\partial\mathcal{H}/\partial u = \lambda_2$ does not contain the control variable. Therefore, it provides no information on the choice of $u(t)$. Further, $\partial^2\mathcal{H}/\partial u^2$ is zero, so $\mathcal{H}$ is not convex. We know, however, that the optimal solution for this problem is a bang-bang trajectory, just like the bang-bang trajectories found by the time-scaling algorithm.

We can recover the bang-bang solution using the minimum principle (11.27). We write

$$\begin{aligned}\mathcal{H}^*(t) = \mathcal{H}(x^*(t), u^*(t), \lambda^*(t)) &\leq \mathcal{H}(x^*(t), u(t), \lambda^*(t)) \\ 1 + \lambda_1^*(t)x_2^*(t) + \lambda_2^*(t)u^*(t) &\leq 1 + \lambda_1^*(t)x_2^*(t) + \lambda_2^*(t)u(t) \\ \lambda_2^*(t)u^*(t) &\leq \lambda_2^*(t)u(t).\end{aligned}$$

Therefore, $u^*(t)$ is the maximum feasible value $u^{\max}$ when $\lambda_2^*(t) < 0$ and the minimum feasible value $-u^{\max}$ when $\lambda_2^*(t) > 0$. As before, λ_2 is a linear function of time, and the terminal state conditions allow us to find the complete solution.

Only for very simple systems is it possible to solve the extremality conditions analytically. In most cases it is necessary to resort to numerical methods to solve the conditions approximately. One such method is called *shooting*. The user guesses initial values for the Lagrange multipliers $\lambda(0)$, which then are numerically integrated according to the adjoint equation $\dot{\lambda} = -\partial\mathcal{H}/\partial x$, while the control vector u evolves according to $\partial\mathcal{H}/\partial u = 0$. After integrating for time t_f (for fixed-time problems), if the final state is not equal to the desired final state, the initial guess of the Lagrange multipliers is modified in some reasonable way and the process repeats. In other words, by modifying the initial conditions, we "shoot" at the goal until we hit it. Typically the initial conditions are modified using an approximation of the gradient of the map taking the initial conditions to the final state.

Other numerical methods for approximately solving for optimal controls include dynamic programming and gradient-based nonlinear optimization. In the next subsection we discuss a nonlinear optimization approach to finding the optimal parameters of a finite parameterization of the system's trajectory or controls. In subsection 11.3.3 we introduce a grid-based search method for finding near-time-optimal trajectories.

For more on optimal control, see the books by Bryson and Ho [72], Bryson [71], Kirk [236], Lewis and Syrmos [287], Stengel [396], and Pontryagin, Boltyanskii, Gamkrelidze, and Mishchenko [354].

11.3.2 Nonlinear Optimization

The general problem can be stated

$$\text{find} \quad t_f, q(t), u(t) \tag{11.33}$$

$$\text{minimizing} \quad J(u(t), q(t), t_f) \tag{11.34}$$

$$\text{subject to} \quad M(q(t))\ddot{q}(t) + C(q(t), \dot{q}(t))\dot{q} + g(q(t)) = u(t), \quad 0 \leq t \leq t_f \tag{11.35}$$

$$u^{\min}(q(t), \dot{q}(t)) \leq u(t) \leq u^{\max}(q(t), \dot{q}(t)), \quad 0 \leq t \leq t_f \tag{11.36}$$

$$h(q(t)) \leq 0, \quad 0 \leq t \leq t_f \tag{11.37}$$

$$q(0) = q_{\text{start}}, \quad \dot{q}(0) = \dot{q}_{\text{start}} \tag{11.38}$$

$$q(t_f) = q_{\text{goal}}, \quad \dot{q}(t_f) = \dot{q}_{\text{goal}}, \tag{11.39}$$

where $h(q) \leq 0$ are configuration inequality constraints representing obstacles and joint limits.

To approximately solve this problem by nonlinear optimization, we approximate the continuous constraints (11.36) and (11.37) by a finite number of constraints. This is typically done by ensuring that the constraints are satisfied at a fixed number of points distributed evenly over the interval $[0, t_f]$. We also choose a finite-parameter representation of the state and control histories. We have three choices of how to do this:

1. *Parameterize the trajectory* $q(t)$. In this case, we solve for the parameterized trajectory directly. The control forces u at any time are calculated using equation (11.35).

2. *Parameterize the control* $u(t)$. We solve for $u(t)$ directly, and calculating the state $(q(t), \dot{q}(t))$ requires integrating the equations of motion (11.35).

3. *Parameterize both* $q(t)$ *and* $u(t)$. We have a larger number of variables, since we are parameterizing both $q(t)$ and $u(t)$. Also, we have a larger number of constraints, as $q(t)$ and $u(t)$ must satisfy the dynamic equations (11.35) explicitly, typically at a fixed number of points distributed evenly over the interval $[0, t_f]$. We must be careful to choose the parameterizations of $q(t)$ and $u(t)$ to be consistent with each other, so that the dynamic equations can be satisfied at these points.

A trajectory or control history can be parameterized in any number of ways. The parameters can be the coefficients of a polynomial in time, the coefficients of a truncated Fourier series, spline coefficients, wavelet coefficients, piecewise constant acceleration or force segments, etc. For example, the control $u_i(t)$ could be represented by $p + 1$ coefficients a_j of a polynomial in time:

$$u_i(t) = \sum_{j=0}^{p} a_j t^j$$

In addition to the parameters for the state or control history, the total time t_f may be another control parameter. The choice of parameterization has implications for the efficiency of the calculation of $q(t)$ and $u(t)$ at a given time t. The choice of parameterization also determines the sensitivity of the state and control to the parameters, and whether each parameter affects the profiles at all times $[0, t_f]$ or just on a finite-time support base. These are important factors in the stability and efficiency of the numerical optimization.

Let X be the vector of the control parameters to be solved. Assuming that either $q(t)$ or $u(t)$ has been parameterized (but not both), and that the $k+1$ constraint checks are spaced at $\Delta t = t_f/k$ intervals, the constrained nonlinear optimization can be written

$$\text{find} \quad X \tag{11.40}$$

$$\text{minimizing} \quad J(X) \tag{11.41}$$

$$\text{subject to} \quad u^{\min}(X, j\Delta t) \leq u(X, j\Delta t) \leq u^{\max}(X, j\Delta t), \quad j = 0 \ldots k \tag{11.42}$$

$$h(X, j\Delta t) \leq 0, \quad j = 0 \ldots k \tag{11.43}$$

$$q(X, 0) = q_{\text{start}}, \quad \dot{q}(X, 0) = \dot{q}_{\text{start}} \tag{11.44}$$

$$q(X, t_f) = q_{\text{goal}}, \quad \dot{q}(X, t_f) = \dot{q}_{\text{goal}}. \tag{11.45}$$

A variant of this formulation approximately represents the constraints (11.42)–(11.45) by penalty functions in the objective function, allowing the use of unconstrained optimization.

A nonlinear program of this type can be solved by a number of methods, including sequential quadratic programming (SQP). Any solver will require the user to provide functions to take a guess X and calculate the objective function $J(X)$ and the constraints (11.42)–(11.45). Often the objective function will have to be calculated by numerical integration. All solvers also need the gradients of the objective function and the constraints with respect to X. These can be calculated numerically by finite differences, or, if possible, analytically. Finally, most solvers make use of Hessians of the objective function and constraint functions with respect to X. Most solvers update a numerical approximation to these Hessians rather than requesting the user to provide these. Details on different methods for nonlinear optimization can be found in [164, 326, 338]. Code for nonlinear optimization includes FSQP and CFSQP [4], NPSOL [5], and routines in the IMSL [6], NAG [7], and MATLAB Optimization Toolbox libraries.

The most important point is that for any of these solvers to work, the objective and constraints must be sufficiently smooth with respect to the control parameters X. They must be at least C^1, but usually C^2 so that Hessian information can be used to speed convergence. A key part of the problem formulation is ensuring this smoothness. When possible, the gradients, and even the Hessians, should be calculated analytically. This will minimize the possibility of the solver failing to converge due to numerical problems, a very real practical concern! Even if the objective function is calculated approximately by numerical integration, it may be possible to calculate the exact gradient of this approximation analytically.

Since nonlinear optimization uses local gradient information, it will converge to a locally optimal solution. For some problems, the control parameter space will be

littered with many local optima. Therefore, the solution achieved will depend heavily on the initial guess. To ensure a good solution, the process can be started from several initial guesses, keeping the best local optimum. Nonlinear optimization can also be used as a final step to locally improve a trajectory found by some other global search method. A survey of nonlinear optimization methods for trajectory generation can be found in [49].

11.3.3 Grid-Based Search

An alternative approach, specifically for time-optimal trajectory planning, uses grid search. As motivation, consider the simple double-integrator system

$$\ddot{q} = a$$
$$|a| \leq a_{\max},$$

where a is the acceleration control. Let q be one-dimensional, so the system can be viewed as a point mass moving on a line with a control force. The time-optimal control from an initial state $(q_{\text{start}}, \dot{q}_{\text{start}})$ to a goal state $(q_{\text{goal}}, \dot{q}_{\text{goal}})$ is bang-bang—the actuator is saturated at all times.

Things will not be so simple when we deal with multidimensional problems with obstacles and velocity limits, so let's consider a grid-based approach that we will be able to generalize to more dimensions. First, we discretize the control set to $\{-a_{\max}, 0, a_{\max}\}$. Next, we choose a timestep h. Now, beginning from $(q_{\text{start}}, \dot{q}_{\text{start}})$, we integrate the three controls forward in time by h to obtain three new states. Think of the initial state $(q_{\text{start}}, \dot{q}_{\text{start}})$ as the root of a tree, and the three new states as children of the root (figure 11.8). From each of these three, we integrate the controls forward to obtain a new level of the tree. We continue in a breadth-first fashion. If the trajectory to a new node in the tree passes through an obstacle or exceeds a velocity limit, this node

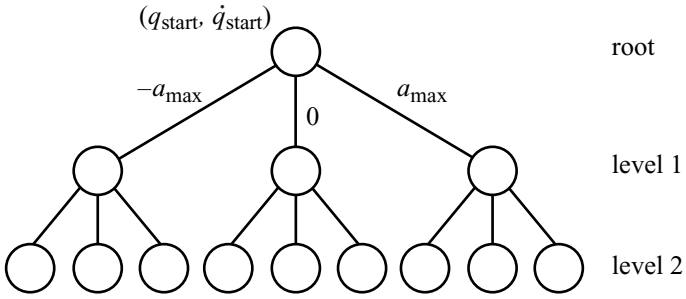

Figure 11.8 The search tree for three controls.

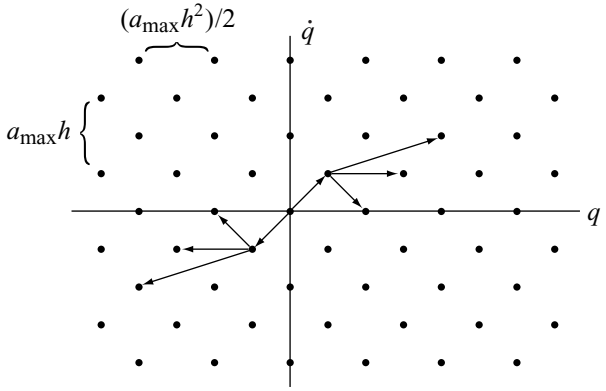

Figure 11.9 The search tree of figure 11.8 shown on the state space grid. The search begins at (0, 0). The actual trajectories between vertices are not straight lines, but quadratics.

is pruned from the tree. The search continues until a trajectory reaches a state in a specified goal region. The trajectory is specified by the piecewise-constant controls to traverse the tree from the root node to this final node. Since the search is breadth-first, exploring all reachable states at time kh before moving on to time $(k+1)h$, the trajectory is time-optimal for the chosen discretization of time and controls.

We call this a grid-based search because each of the nodes reached during the growth of the tree lies on a regular grid on the $(q, \dot{q})$ state space. From any state, the new state obtained by integrating one of the discretized accelerations for time h will involve a change in $\dot{q}$ equal to an integral multiple of $a_{\max}h$ and a change in q equal to an integral multiple of $\frac{1}{2}a_{\max}h^2$. An example of such a grid is shown in figure 11.9. The search tree shown on this grid is two levels deep, beginning at (0, 0). The key point here is that, given some bounds on q and $\dot{q}$, the size of the grid is easily computed, so an upper bound on the computational complexity of the search of this grid is also easily computed.

Let us now consider a more general problem statement. The system is described by $q \in D \subset \mathbb{R}^{n_\mathcal{Q}}$, where D is a bounded subset of $\mathbb{R}^{n_\mathcal{Q}}$, and velocity and acceleration bounds of the form

$$|\dot{q}_i| \leq v_{\max}, \quad i = 1 \ldots n_\mathcal{Q}$$
$$|\ddot{q}_i| \leq a_{\max}, \quad i = 1 \ldots n_\mathcal{Q}.$$

Note that this is a very limited class of systems, as the maximum feasible $\ddot{q}_i$'s are constant and independent of the state $(q, \dot{q})$. An example of such a system might be a point in $n_\mathcal{Q}$-dimensional Euclidean space with a thruster for each degree of

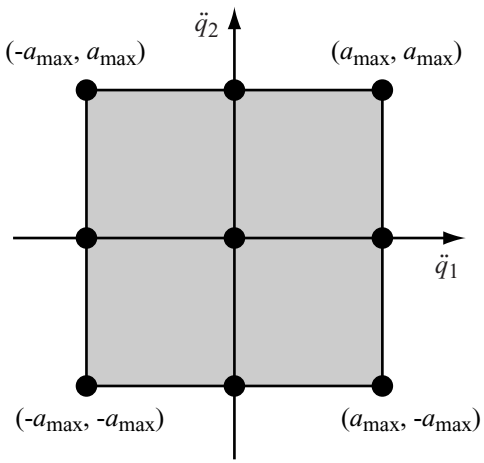

Figure 11.10 The control discretization for $n_{\mathcal{Q}} = 2$.

freedom, or a Cartesian robot arm consisting of all prismatic joints and actuators with state-independent bounds. The problem is to find a collision-free, approximately time-optimal trajectory from $(q^*_{\text{start}}, \dot{q}^*_{\text{start}})$ [at or near the desired start state $(q_{\text{start}}, \dot{q}_{\text{start}})$] to a goal state near the desired goal state $(q_{\text{goal}}, \dot{q}_{\text{goal}})$.

The algorithm uses a discretized control set A consisting of the cross products of $\{-a_{\max}, 0, a_{\max}\}$ for each degree of freedom, yielding $3^{n_{\mathcal{Q}}}$ distinct controls (figure 11.10). These controls result in a regular grid on $2n_{\mathcal{Q}}$-dimensional state space, similar to figure 11.9. Algorithm 21, GRID_SEARCH, is described in pseudocode below.

This algorithm is straightforward, except for one twist: the algorithm prunes a node if the trajectory passes *close* to an obstacle, not just if it passes through an obstacle. Thus the algorithm will only return trajectories that are *safe*. We define a trajectory to be $\delta_v(c_0, c_1)$-*safe* if there exists a speed-dependent ball of free configurations about each q in the trajectory, where the radius of the ball is $c_0 + c_1\|\dot{q}\|$. The parameters c_0 and c_1 are safety parameters.

Since the algorithm uses a finite timestep h, any trajectory it finds will only be an *approximately* time-optimal safe trajectory. Instead of directly choosing h, the user could have control over a parameter ϵ, $0 < \epsilon < 1$, which defines the crudeness of the approximation. Larger values of ϵ correspond to cruder approximations, and the timestep h goes to zero as ϵ goes to zero. As we will see, ϵ may be viewed as a measure of how much we will allow $\delta_v(c_0, c_1)$-safety to be violated, giving $(1 - \epsilon)\delta_v(c_0, c_1)$-safety.

Algorithm 21 GRID_SEARCH

Input: Start node $(q^*_{\text{start}}, \dot{q}^*_{\text{start}})$, goal region $\mathcal{G}$
Output: A trajectory to $\mathcal{G}$ or FAILURE

```
1:  Place (q*_start, q̇*_start) at root of tree T (level 0)
2:  level ← 0, solved ← FALSE, ANS ← ∅
3:  while not solved do
4:     if no nodes in level level of T then
5:        return FAILURE
6:     end if
7:     for each node in level level of T do
8:        for each control in A do
9:           Integrate control for time h from node, getting newnode
10:          if newnode has not been previously reached, and trajectory does not pass close
             to an obstacle nor exceed v_max then
11:             add newnode to T as child of node
12:          end if
13:          if trajectory enters G then
14:             solved ← TRUE, store newnode in list ANS
15:          end if
16:       end for
17:    end for
18:    level ← level + 1
19: end while
20: For each node in ANS, find the trajectory that reaches G first
```

Although the algorithm itself is straightforward, analysis of the algorithm is quite involved. Given a desired safety margin and ϵ, we would like to know how to choose h to guarantee completeness of the algorithm, and how ϵ and the safety margin relate to the algorithm's running time. This is the problem that was studied in detail by Donald and Xavier [135], building on work by Canny, Donald, Reif, and Xavier [94, 133]. Their analysis holds for a point robot with $n_{\mathcal{Q}} = 2$ or 3 moving among polygonal or polyhedral obstacles. They give us the following result.

THEOREM 11.3.2 *Let ℓ be the diameter of the robot configuration space, c_0 and c_1 be safety parameters, and $v_{\max}$ and $a_{\max}$ be the velocity and acceleration limits. Let $0 < \epsilon < 1$. Assume there exists a $\delta_v(c_0, c_1)$-safe trajectory from $(q_{\text{start}}, \dot{q}_{\text{start}})$ to $(q_{\text{goal}}, \dot{q}_{\text{goal}})$ taking time T_{opt} by some control function with $|\ddot{q}_i| \leq a_{\max}$ for all i and*

$t \in [0, T_{\text{opt}}]$. Choose the largest h so that

$$h \leq \frac{v_{\max}}{a_{\max}}, \quad h \leq \frac{c_0 \epsilon}{2a_{\max}c_1(1-\epsilon) + 5v_{\max}},$$

and $v_{\max}$ is an integral multiple of $a_{\max}h$. Choose an approximate starting state $(q^*_{\text{start}}, \dot{q}^*_{\text{start}})$ where, for each coordinate i,

$$\dot{q}^*_{\text{start},i} = \text{the multiple of } a_{\max}h \text{ closest to } \dot{q}_{\text{start},i}$$

$$q^*_{\text{start},i} = q_{\text{start},i} - \frac{h^2}{2}(\dot{q}_{\text{start},i} - \dot{q}^*_{\text{start},i}).$$

Define the goal neighborhood to be all points within $(\frac{5a_{\max}h^2}{2}, 2a_{\max}h)$ of $(q_{\text{goal}}, \dot{q}_{\text{goal}})$. [Note that the distance from $(q^*_{\text{start}}, \dot{q}^*_{\text{start}})$ to $(q_{\text{start}}, \dot{q}_{\text{start}})$, and the distance from any point in the goal neighborhood to $(q_{\text{goal}}, \dot{q}_{\text{goal}})$, is $O(\epsilon)$.] Then the algorithm outlined above is guaranteed to find a $(1-\epsilon)\delta_v(c_0, c_1)$-safe trajectory taking at most time T_{opt} from $(q^*_{\text{start}}, \dot{q}^*_{\text{start}})$ to the goal neighborhood. The running time of the algorithm is

$$O\left(c^{n_{\mathcal{Q}}} N \left(\frac{v_{\max}(a_{\max}c_1 + v_{\max})^3 \ell}{a^2_{\max}c^3_0\epsilon^3}\right)^{n_{\mathcal{Q}}}\right),$$

where $n_{\mathcal{Q}} = 2$ or 3, c is a constant, and N is the number of faces in the obstacles. In terms of the dimension $n_{\mathcal{Q}}$ and the approximation variable ϵ, the running time goes as $O((\frac{1}{\epsilon})^{3n_{\mathcal{Q}}})$, i.e., polynomial in ϵ and exponential in $n_{\mathcal{Q}}$.

The proof of this theorem is beyond the scope of this chapter, and it depends on efficient goal and safety checking between grid vertices. One important property of the algorithm is that the $(1-\epsilon)\delta_v(c_0, c_1)$-safe trajectory found by the algorithm may be quite different from the time-optimal $\delta_v(c_0, c_1)$-safe trajectory. The only guarantee is that the running time of the approximate trajectory will be no greater than T_{opt}, the time for a time-optimal $\delta_v(c_0, c_1)$-safe trajectory.

Figure 11.11 shows a cartoon example of different kinds of optimal paths for a point in the plane. The true time-optimal trajectory, with no consideration for safety, is a straight-line motion between the start and the goal. In this example, the time-optimal $\delta_v(c_0, c_1)$-safe trajectory avoids the narrow passage, as it would require unacceptably slow speeds to be safe. Finally, the algorithm outlined above finds the approximately time-optimal $(1-\epsilon)\delta_v(c_0, c_1)$-safe trajectory from an approximate start state to an approximate goal state.

We would like to generalize the grid-search algorithm to handle more general dynamic systems of the form of equation (10.7), such as open-chain robot manipulators. Unlike the previous case, the dynamics are not decoupled generally, and the

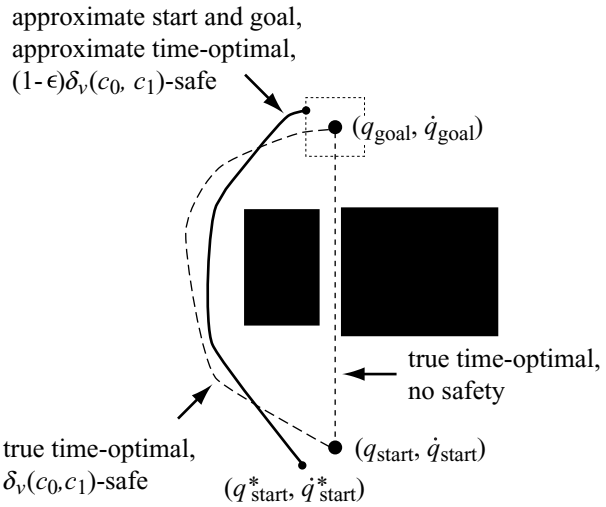

Figure 11.11 A time-optimal motion, a $\delta_v(c_0, c_1)$-safe time-optimal motion, and a solution found by the algorithm. The dashed box at the goal is the the goal neighborhood projected to the configuration space.

current state of the system affects the feasible $\ddot{q}$:

$$\ddot{q} = M^{-1}(q)(u - C(q, \dot{q})\dot{q} - g(q)) \tag{11.46}$$

To simplify matters somewhat, we will assume constant bounds on the available controls, possibly different for each i:

$$|u_i| \leq u_i^{\max} \tag{11.47}$$

At a given state $(q, \dot{q})$, equation (11.46) transforms the rectangular parallelepiped of feasible controls u implied by the constraints (11.47) to a parallelepiped in the $\ddot{q}$ space, as shown in figure 11.12. Let $\mathbf{A}(q, \dot{q})$ denote the state-dependent parallelepiped of feasible $\ddot{q}$. We assume that $\mathbf{A}(q, 0)$ contains the origin of the $\ddot{q}$ space in its interior for all $q \in \mathcal{Q}$, i.e., the actuators are strong enough to hold the robot stationary at any configuration.

To apply the algorithm from before, imagine placing a constant grid on the $\ddot{q}$ space, as shown in figure 11.12, discretizing the feasible accelerations. For a fixed time interval h, these controls will again create a regular grid of reachable points on the state space. For the current state, we use the $\ddot{q}$ grid points inside $\mathbf{A}(q, \dot{q})$ as our set of actions A. One problem is that $\mathbf{A}(q, \dot{q})$ changes during the time interval h, so that a $\ddot{q}$ that is feasible at the beginning of the timestep may no longer be feasible at the

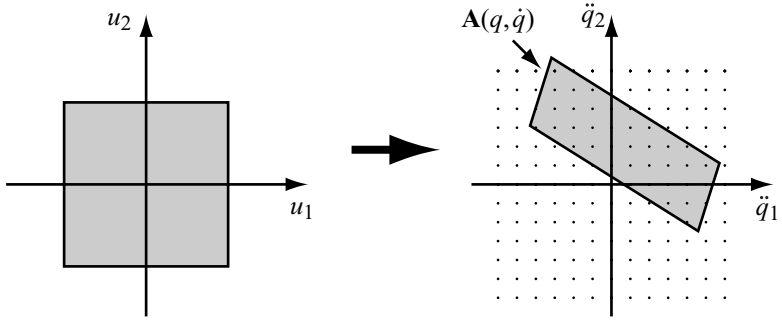

Figure 11.12 The equations of motion turn the feasible controls u into a parallelepiped $\mathbf{A}(q, \dot{q})$ of feasible accelerations $\ddot{q}$.

end. Another problem is that it might happen that there is *no* grid point that is feasible at both the beginning and end of the timestep. Worse yet, if $M^{-1}(q)$ ever loses rank, then $\mathbf{A}(q, \dot{q})$ collapses to zero volume, and no point on a fixed grid is likely to lie in it.

To prevent $\mathbf{A}(q, \dot{q})$ from collapsing, we assume an upper bound on the largest eigenvalue of $M(q)$ during any motion. This tells us how "skinny" $\mathbf{A}(q, \dot{q})$ can become, providing information on how to choose the $\ddot{q}$ grid spacing so that there are always grid points inside the region. For a given timestep h, we also have to choose a conservative approximation $\widehat{\mathbf{A}}(q, \dot{q}) \subset \mathbf{A}(q, \dot{q})$ such that any $\ddot{q}$ inside $\widehat{\mathbf{A}}(q(t), \dot{q}(t))$ stays inside $\mathbf{A}(q(t + \delta t), \dot{q}(t + \delta t))$ for all $\delta t \in [0, h]$ (figure 11.13). To construct the conservative approximation $\widehat{\mathbf{A}}(q, \dot{q})$, we need to know how quickly $M(q)$ can change, which can be bounded by global bounds on the derivatives of $M(q)$ with respect to time. In other words, properties of $M(q)$ and its derivatives must be used to avoid the problems outlined in the previous paragraph. Details can be found in [186].

The idea is to choose the $\ddot{q}$ grid spacing (which may be different for each $\ddot{q}_i$, $i = 1 \ldots n$) and the timestep h so that any feasible trajectory of the system using the full acceleration capabilities $\mathbf{A}(q, \dot{q})$ can be approximately tracked by trajectories using the discretized controls A, chosen to be the $\ddot{q}$ grid points lying inside $\widehat{\mathbf{A}}(q, \dot{q})$.[6] The allowable tracking error depends on a user-defined approximation parameter ϵ, $0 < \epsilon < 1$, where the allowable tracking error goes to zero as ϵ goes to zero.

As before, we can define a trajectory to be $\delta_v(c_0, c_1)$-safe if all real-space obstacles are avoided by a distance of at least $c_0 + c_1\|\dot{q}\|$ at all points along the trajectory. It has been shown that if there exists a $\delta_v(c_0, c_1)$-safe trajectory from $(q_{\text{start}}, \dot{q}_{\text{start}})$

6. Alternatively, A could consist only of $\ddot{q}$ grid points near the boundary of $\widehat{\mathbf{A}}(q, \dot{q})$.

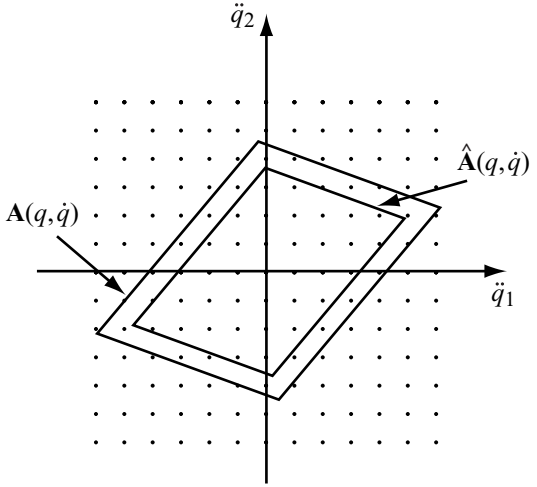

Figure 11.13 The parallelepiped $\mathbf{A}(q, \dot{q})$ represents instantaneously feasible accelerations and $\widehat{\mathbf{A}}(q, \dot{q})$ is a conservative approximation to the set of accelerations that are feasible over the entire timestep of duration h.

to $(q_{\text{goal}}, \dot{q}_{\text{goal}})$ taking time T_{opt}, then the procedure GRID_SEARCH outlined in algorithm 21 will find a $(1-\epsilon)\delta_v(c_0, c_1)$-safe trajectory from $(q^*_{\text{start}}, \dot{q}^*_{\text{start}})$ to $(q^*_{\text{goal}}, \dot{q}^*_{\text{goal}})$ [where the errors from the desired initial and final states are $O(\epsilon)$] taking no more than time $(1+\epsilon)T_{\text{opt}}$. The timestep h and $\ddot{q}$ grid-spacing are polynomial in ϵ, and the choice of these parameters involves lengthy calculations. The running time of the algorithm has been shown to be polynomial in ϵ and exponential in the degrees of freedom $n_{\mathcal{Q}}$ [134, 185, 205, 206].

The grid-search algorithm is attractive because it is possible to prove its completeness (using the concept of safe trajectories) and to understand how the running time depends on an approximation parameter ϵ. This allows the user to trade off computation time against the quality of the trajectory. There have been no implementations of the algorithm for more than a few degrees of freedom, however, because in practice the computation time and memory requirements grow quickly with the number of degrees of freedom.

The running time of the algorithm can be improved by using nonuniform discretization of the state space [362] or search heuristics that favor first exploring from nodes close to the goal state. The `RRT` and `EST` (see chapter 7) are methods for trajectory planning based on a heuristic that biases the search to evenly explore the state space. Like the grid-based algorithm, they discretize the controls and choose a

constant, finite timestep. Unlike the grid-based algorithm, they give up on any notion of optimality and settle for probabilistic completeness, attempting to quickly find any feasible trajectory.

Problems

1. For the RP manipulator of example 11.2.1, write a program that accepts a straight-line path and draws the velocity limit curve.
2. Implement the time-scaling algorithm for the RP arm of example 11.2.1 following a straight line path specified by the user. Try to write the program in a modular fashion, so systems with different dynamics can use many of the same routines.
3. In step 2 of the time-scaling algorithm, explain why there is no solution to the time-scaling problem if $\dot{s} = 0$ is reached. Give a simple example of this case.
4. In step 3 of the time-scaling algorithm, explain why there is no solution to the time-scaling problem if the forward integration reaches $s = 1$ or $\dot{s} = 0$ before crossing the velocity limit curve or the curve F. Give a simple example of this case.
5. In example 11.2.1, explain intuitively why $b_1(s) \neq 0$ but $b_2(s) = 0$, in terms of the arm dynamics and the manipulator path.
6. At a zero inertia point s, at least one term $a_i(s)$ is equal to zero. Explain the implications if $b_i(s)$ is also zero. Should the algorithm be modified to handle this case? Remember, we are assuming the robot is strong enough to maintain any configuration statically. Does it matter if the actuator limits are state-dependent?
7. We would like to find optimal motions for the RP robot arm of example 11.2.1. The Lagrangian defining the objective function is $\mathcal{L} = w_1 u_1^2 + w_2 u_2^2$, where w_1 and w_2 are positive weights. For a fixed time of motion t_f, write the Hamiltonian, the necessary condition for optimality (11.28), and the adjoint equation (11.30).
8. For the previous problem, choose a parameterization of the joint trajectories and use a nonlinear programming package to find optimal point-to-point trajectories for the RP arm. A good initial guess for the joint trajectories would be one that exactly satisfies the start and goal state constraints. Comment on any difficulties you encounter using nonlinear programming.
9. Implement the procedure GRID_SEARCH (algorithm 21) for a point moving in a planar world with polygonal obstacles.