CHAPTER 7

Trajectory generation

7.1 INTRODUCTION

In this chapter, we concern ourselves with methods of computing a trajectory that describes the desired motion of a manipulator in multidimensional space. Here, **trajectory** refers to a time history of position, velocity, and acceleration for each degree of freedom.

This problem includes the human-interface problem of how we wish to *specify* a trajectory or path through space. In order to make the description of manipulator motion easy for a human user of a robot system, the user should not be required to write down complicated functions of space and time to specify the task. Rather, we must allow the capability of specifying trajectories with simple descriptions of the desired motion, and let the system figure out the details. For example, the user might want to be able to specify nothing more than the desired goal position and orientation of the end-effector and leave it to the system to decide on the exact shape of the path to get there, the duration, the velocity profile, and other details.

We also are concerned with how trajectories are *represented* in the computer after they have been planned. Finally, there is the problem of actually computing the trajectory from the internal representation—or *generating* the trajectory. Generation occurs at *run time*; in the most general case, position, velocity, and acceleration are computed. These trajectories are computed on digital computers, so the trajectory points are computed at a certain rate, called the **path-update rate**. In typical manipulator systems, this rate lies between 60 and 2000 Hz.

7.2 GENERAL CONSIDERATIONS IN PATH DESCRIPTION AND GENERATION

For the most part, we will consider motions of a manipulator as motions of the tool frame, $\{T\}$, relative to the station frame, $\{S\}$. This is the same manner in which an eventual user of the system would think, and designing a path description and generation system in these terms will result in a few important advantages.

When we specify paths as motions of the tool frame relative to the station frame, we decouple the motion description from any particular robot, end-effector, or workpieces. This results in a certain modularity and would allow the same path description to be used with a different manipulator—or with the same manipulator, but a different tool size. Further, we can specify and plan motions relative to a moving workstation (perhaps a conveyor belt) by planning motions relative to the station frame as always and, at run time, causing the definition of $\{S\}$ to be changing with time.

As shown in Fig. 7.1, the basic problem is to move the manipulator from an initial position to some desired final position—that is, we wish to move the tool frame from its current value, $\{T_{\text{initial}}\}$, to a desired final value, $\{T_{\text{final}}\}$. Note that, in general, this motion involves both a change in orientation and a change in the position of the tool relative to the station.

Sometimes it is necessary to specify the motion in much more detail than by simply stating the desired final configuration. One way to include more detail in a path description is to give a sequence of desired **via points** (intermediate points between the initial and final positions). Thus, in completing the motion, the tool frame must pass through a set of intermediate positions and orientations as described by the via points. Each of these via points is actually a frame that specifies both the position and orientation of the tool relative to the station. The name **path points** includes all the via points plus the initial and final points. Remember that, although we generally use the term "points," these are actually frames, which give both position and orientation. Along with these *spatial* constraints on the motion, the user could also wish to specify *temporal* attributes of the motion. For example, the time elapsed between via points might be specified in the description of the path.

Usually, it is desirable for the motion of the manipulator to be *smooth*. For our purposes, we will define a smooth function as a function that is continuous and has a continuous first derivative. Sometimes a continuous second derivative is also

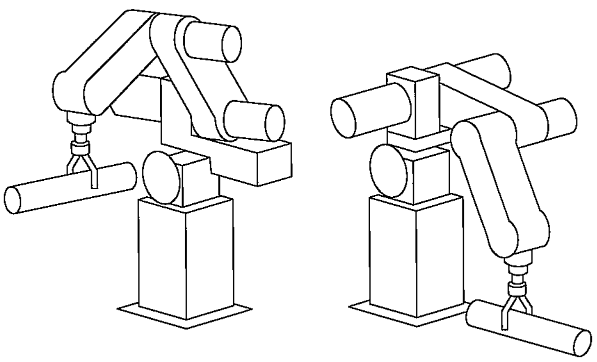

FIGURE 7.1: In executing a trajectory, a manipulator moves from its initial position to a desired goal position in a smooth manner.

desirable. Rough, jerky motions tend to cause increased wear on the mechanism and cause vibrations by exciting resonances in the manipulator. In order to guarantee smooth paths, we must put some sort of constraints on the spatial and temporal qualities of the path *between* the via points.

At this point, there are many choices that may be made and, consequently, a great variety in the ways that paths might be specified and planned. Any smooth functions of time that pass through the via points could be used to specify the exact path shape. In this chapter, we will discuss a couple of simple choices for these functions. Other approaches can be found in [1, 2] and [13–16].

7.3 JOINT-SPACE SCHEMES

In this section, we consider methods of path generation in which the path shapes (in space and in time) are described in terms of functions of joint angles.

Each path point is usually specified in terms of a desired position and orientation of the tool frame, $\{T\}$, relative to the station frame, $\{S\}$. Each of these via points is "converted" into a set of desired joint angles by application of the inverse kinematics. Then a smooth function is found for each of the n joints that pass through the via points and end at the goal point. The time required for each segment is the same for each joint so that all joints will reach the via point at the same time, thus resulting in the desired Cartesian position of $\{T\}$ at each via point. Other than specifying the same duration for each joint, the determination of the desired joint angle function for a particular joint does not depend on the functions for the other joints.

Hence, joint-space schemes achieve the desired position and orientation at the via points. In between via points, the shape of the path, although rather simple in joint space, is complex if described in Cartesian space. Joint-space schemes are usually the easiest to compute, and, because we make no continuous correspondence between joint space and Cartesian space, there is essentially no problem with singularities of the mechanism.

Cubic polynomials

Consider the problem of moving the tool from its initial position to a goal position in a certain amount of time. Inverse kinematics allow the set of joint angles that correspond to the goal position and orientation to be calculated. The initial position of the manipulator is also known in the form of a set of joint angles. What is required is a function for each joint whose value at t_0 is the initial position of the joint and whose value at t_f is the desired goal position of that joint. As shown in Fig. 7.2, there are many smooth functions, $\theta(t)$, that might be used to interpolate the joint value.

In making a single smooth motion, at least four constraints on $\theta(t)$ are evident. Two constraints on the function's value come from the selection of initial and final values:

$$\begin{aligned}\theta(0) &= \theta_0,\\ \theta(t_f) &= \theta_f.\end{aligned} \tag{7.1}$$

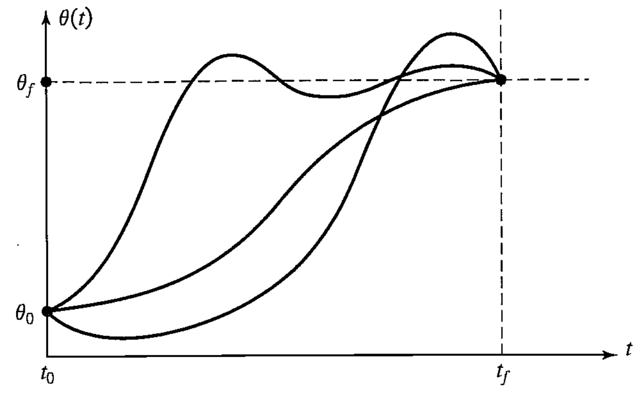

FIGURE 7.2: Several possible path shapes for a single joint.

An additional two constraints are that the function be continuous in velocity, which in this case means that the initial and final velocity are zero:

$$\begin{aligned}\dot{\theta}(0) &= 0,\\ \dot{\theta}(t_f) &= 0.\end{aligned} \tag{7.2}$$

These four constraints can be satisfied by a polynomial of at least third degree. (A cubic polynomial has four coefficients, so it can be made to satisfy the four constraints given by (7.1) and (7.2).) These constraints uniquely specify a particular cubic. A cubic has the form

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3, \tag{7.3}$$

so the joint velocity and acceleration along this path are clearly

$$\begin{aligned}\dot{\theta}(t) &= a_1 + 2a_2 t + 3a_3 t^2,\\ \ddot{\theta}(t) &= 2a_2 + 6a_3 t.\end{aligned} \tag{7.4}$$

Combining (7.3) and (7.4) with the four desired constraints yields four equations in four unknowns:

$$\begin{aligned}\theta_0 &= a_0,\\ \theta_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3,\\ 0 &= a_1,\\ 0 &= a_1 + 2a_2 t_f + 3a_3 t_f^2.\end{aligned} \tag{7.5}$$

Solving these equations for the a_i, we obtain

$$\begin{aligned}a_0 &= \theta_0,\\ a_1 &= 0,\end{aligned}$$

$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0), \tag{7.6}$$

$$a_3 = -\frac{2}{t_f^3}(\theta_f - \theta_0).$$

Using (7.6), we can calculate the cubic polynomial that connects any initial joint-angle position with any desired final position. This solution is for the case when the joint starts and finishes at zero velocity.

EXAMPLE 7.1

A single-link robot with a rotary joint is motionless at $\theta = 15$ degrees. It is desired to move the joint in a smooth manner to $\theta = 75$ degrees in 3 seconds. Find the coefficients of a cubic that accomplishes this motion and brings the manipulator to rest at the goal. Plot the position, velocity, and acceleration of the joint as a function of time.

Plugging into (7.6), we find that

$$\begin{aligned} a_0 &= 15.0, \\ a_1 &= 0.0, \\ a_2 &= 20.0, \\ a_3 &= -4.44. \end{aligned} \tag{7.7}$$

Using (7.3) and (7.4), we obtain

$$\begin{aligned} \theta(t) &= 15.0 + 20.0t^2 - 4.44t^3, \\ \dot{\theta}(t) &= 40.0t - 13.33t^2, \\ \ddot{\theta}(t) &= 40.0 - 26.66t. \end{aligned} \tag{7.8}$$

Figure 7.3 shows the position, velocity, and acceleration functions for this motion sampled at 40 Hz. Note that the velocity profile for any cubic function is a parabola and that the acceleration profile is linear.

Cubic polynomials for a path with via points

So far, we have considered motions described by a desired duration and a final goal point. In general, we wish to allow paths to be specified that include intermediate via points. If the manipulator is to come to rest at each via point, then we can use the cubic solution of Section 7.3.

Usually, we wish to be able to pass through a via point without stopping, and so we need to generalize the way in which we fit cubics to the path constraints.

As in the case of a single goal point, each via point is usually specified in terms of a desired position and orientation of the tool frame relative to the station frame. Each of these via points is "converted" into a set of desired joint angles by application of the inverse kinematics. We then consider the problem of computing cubics that connect the via-point values for each joint together in a smooth way.

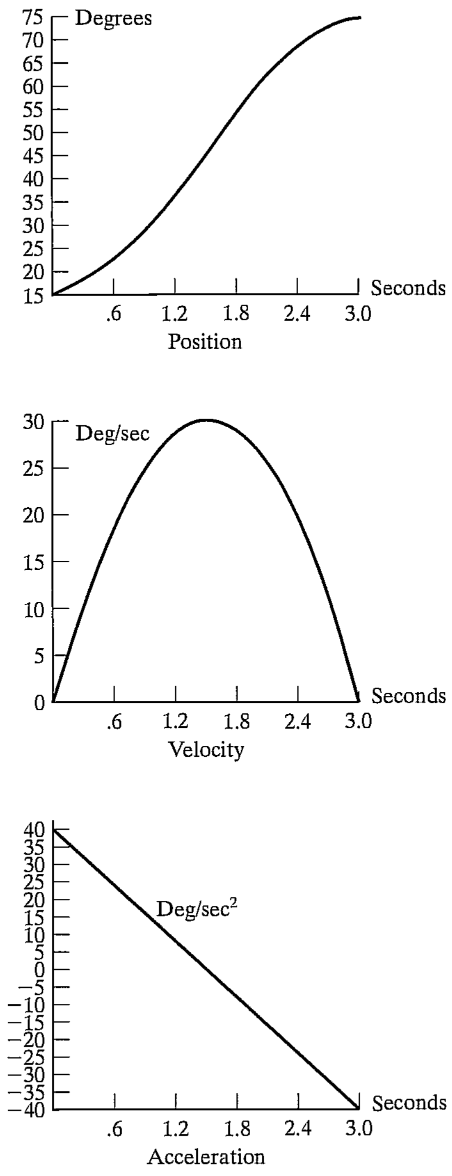

FIGURE 7.3: Position, velocity, and acceleration profiles for a single cubic segment that starts and ends at rest.

If desired velocities of the joints at the via points are known, then we can construct cubic polynomials as before; now, however, the velocity constraints at each end are not zero, but rather, some known velocity. The constraints of (7.3) become

$$\begin{aligned}\dot{\theta}(0) &= \dot{\theta}_0, \\ \dot{\theta}(t_f) &= \dot{\theta}_f.\end{aligned} \tag{7.9}$$

The four equations describing this general cubic are

$$\begin{aligned}
\theta_0 &= a_0, \\
\theta_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3, \\
\dot{\theta}_0 &= a_1, \\
\dot{\theta}_f &= a_1 + 2a_2 t_f + 3a_3 t_f^2.
\end{aligned} \tag{7.10}$$

Solving these equations for the a_i, we obtain

$$\begin{aligned}
a_0 &= \theta_0, \\
a_1 &= \dot{\theta}_0, \\
a_2 &= \frac{3}{t_f^2}(\theta_f - \theta_0) - \frac{2}{t_f}\dot{\theta}_0 - \frac{1}{t_f}\dot{\theta}_f, \\
a_3 &= -\frac{2}{t_f^3}(\theta_f - \theta_0) + \frac{1}{t_f^2}(\dot{\theta}_f + \dot{\theta}_0).
\end{aligned} \tag{7.11}$$

Using (7.11), we can calculate the cubic polynomial that connects any initial and final positions with any initial and final velocities.

If we have the desired joint velocities at each via point, then we simply apply (7.11) to each segment to find the required cubics. There are several ways in which the desired velocity at the via points might be specified:

1. The user specifies the desired velocity at each via point in terms of a Cartesian linear and angular velocity of the tool frame at that instant.
2. The system automatically chooses the velocities at the via points by applying a suitable heuristic in either Cartesian space or joint space.
3. The system automatically chooses the velocities at the via points in such a way as to cause the acceleration at the via points to be continuous.

In the first option, Cartesian desired velocities at the via points are "mapped" to desired joint rates by using the inverse Jacobian of the manipulator evaluated at the via point. If the manipulator is at a singular point at a particular via point, then the user is not free to assign an arbitrary velocity at this point. It is a useful capability of a path-generation scheme to be able to meet a desired velocity that the user specifies, but it would be a burden to require that the user always make these specifications. Therefore, a convenient system should include either option 2 or 3 (or both).

In option 2, the system automatically chooses reasonable intermediate velocities, using some kind of heuristic. Consider the path specified by the via points shown for some joint, θ, in Fig. 7.4.

In Fig. 7.4, we have made a reasonable choice of joint velocities at the via points, as indicated with small line segments representing tangents to the curve at each via point. This choice is the result of applying a conceptually and computationally simple heuristic. Imagine the via points connected with straight line segments. If the

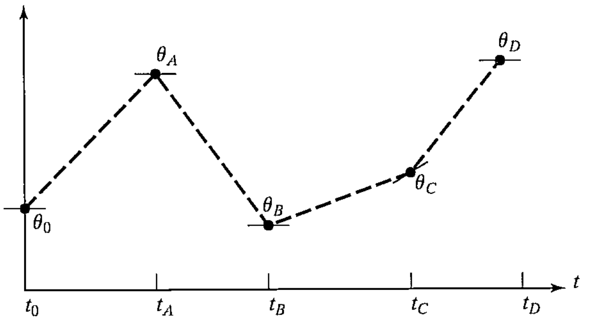

FIGURE 7.4: Via points with desired velocities at the points indicated by tangents.

slope of these lines changes sign at the via point, choose zero velocity; if the slope of these lines does not change sign, choose the average of the two slopes as the via velocity. In this way, from specification of the desired via points alone, the system can choose the velocity at each point.

In option 3, the system chooses velocities in such a way that acceleration is continuous at the via point. To do this, a new approach is needed. In this kind of spline, set of data[1] we replace the two velocity constraints at the connection of two cubics with the two constraints that velocity be continuous and acceleration be continuous.

EXAMPLE 7.2

Solve for the coefficients of two cubics that are connected in a two-segment spline with continuous acceleration at the intermediate via point. The initial angle is θ_0, the via point is θ_v, and the goal point is θ_g.

The first cubic is

$$\theta(t) = a_{10} + a_{11}t + a_{12}t^2 + a_{13}t^3, \tag{7.12}$$

and the second is

$$\theta(t) = a_{20} + a_{21}t + a_{22}t^2 + a_{23}t^3. \tag{7.13}$$

Each cubic will be evaluated over an interval starting at $t = 0$ and ending at $t = t_{fi}$, where $i = 1$ or $i = 2$.

The constraints we wish to enforce are

$$\begin{aligned}
\theta_0 &= a_{10}, \\
\theta_v &= a_{10} + a_{11}t_{f1} + a_{12}t_{f1}^2 + a_{13}t_{f1}^3, \\
\theta_v &= a_{20}, \\
\theta_g &= a_{20} + a_{21}t_{f2} + a_{22}t_{f2}^2 + a_{23}t_{f2}^3, \qquad (7.14) \\
0 &= a_{11},
\end{aligned}$$

[1]In our usage, the term "spline" simply means a function of time.

$$0 = a_{21} + 2a_{22}t_{f2} + 3a_{23}t_{f2}^2,$$
$$a_{11} + 2a_{12}t_{f1} + 3a_{12}t_{f1}^2 = a_{21},$$
$$2a_{12} + 6a_{13}t_{f1} = 2a_{22}.$$

These constraints specify a linear-equation problem having eight equations and eight unknowns. Solving for the case $t_f = t_{f1} = t_{f2}$, we obtain

$$\begin{aligned}
a_{10} &= \theta_0, \\
a_{11} &= 0, \\
a_{12} &= \frac{12\theta_v - 3\theta_g - 9\theta_0}{4t_f^2}, \\
a_{13} &= \frac{-8\theta_v + 3\theta_g + 5\theta_0}{4t_f^3}, \\
a_{20} &= \theta_v, \\
a_{21} &= \frac{3\theta_g - 3\theta_0}{4t_f}, \\
a_{22} &= \frac{-12\theta_v + 6\theta_g + 6\theta_0}{4t_f^2}, \\
a_{23} &= \frac{8\theta_v - 5\theta_g - 3\theta_0}{4t_f^3}.
\end{aligned} \tag{7.15}$$

For the general case, involving n cubic segments, the equations that arise from insisting on continuous acceleration at the via points can be cast in matrix form, which is solved to compute the velocities at the via points. The matrix turns out to be tridiagonal and easily solved [4].

Higher-order polynomials

Higher-order polynomials are sometimes used for path segments. For example, if we wish to be able to specify the position, velocity, *and* acceleration at the beginning and end of a path segment, a quintic polynomial is required, namely,

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5, \tag{7.16}$$

where the constraints are given as

$$\theta_0 = a_0,$$
$$\theta_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5,$$

$$\begin{aligned}\dot\theta_0 &= a_1,\\ \dot\theta_f &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4,\\ \ddot\theta_0 &= 2a_2,\\ \ddot\theta_f &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3.\end{aligned} \tag{7.17}$$

These constraints specify a linear set of six equations with six unknowns, whose solution is

$$\begin{aligned}a_0 &= \theta_0,\\ a_1 &= \dot\theta_0,\\ a_2 &= \frac{\ddot\theta_0}{2},\\ a_3 &= \frac{20\theta_f - 20\theta_0 - (8\dot\theta_f + 12\dot\theta_0)t_f - (3\ddot\theta_0 - \ddot\theta_f)t_f^2}{2t_f^3},\\ a_4 &= \frac{30\theta_0 - 30\theta_f + (14\dot\theta_f + 16\dot\theta_0)t_f + (3\ddot\theta_0 - 2\ddot\theta_f)t_f^2}{2t_f^4},\\ a_5 &= \frac{12\theta_f - 12\theta_0 - (6\dot\theta_f + 6\dot\theta_0)t_f - (\ddot\theta_0 - \ddot\theta_f)t_f^2}{2t_f^5}.\end{aligned} \tag{7.18}$$

Various algorithms are available for computing smooth functions (polynomial or otherwise) that pass through a given set of data points [3, 4]. Complete coverage is beyond the scope of this book.

Linear function with parabolic blends

Another choice of path shape is linear. That is, we simply interpolate linearly to move from the present joint position to the final position, as in Fig. 7.5. Remember that, although the motion of each joint in this scheme is linear, the end-effector in general does not move in a straight line in space.

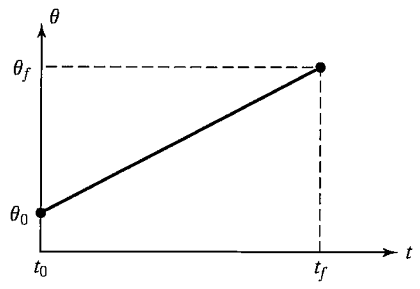

FIGURE 7.5: Linear interpolation requiring infinite acceleration.

However, straightforward linear interpolation would cause the velocity to be discontinuous at the beginning and end of the motion. To create a smooth path with continuous position and velocity, we start with the linear function but add a parabolic *blend* region at each path point.

During the blend portion of the trajectory, constant acceleration is used to change velocity smoothly. Figure 7.6 shows a simple path constructed in this way. The linear function and the two parabolic functions are "splined" together so that the entire path is continuous in position and velocity.

In order to construct this single segment, we will assume that the parabolic blends both have the same duration; therefore, the same constant acceleration (modulo a sign) is used during both blends. As indicated in Fig. 7.7, there are many solutions to the problem—but note that the answer is always symmetric about the halfway point in time, t_h, and about the halfway point in position, θ_h. The velocity at the end of the blend region must equal the velocity of the linear section, and so we have

$$\ddot{\theta} t_b = \frac{\theta_h - \theta_b}{t_h - t_b}, \tag{7.19}$$

where θ_b is the value of θ at the end of the blend region, and $\ddot{\theta}$ is the acceleration acting during the blend region. The value of θ_b is given by

$$\theta_b = \theta_0 + \tfrac{1}{2}\ddot{\theta} t_b^2. \tag{7.20}$$

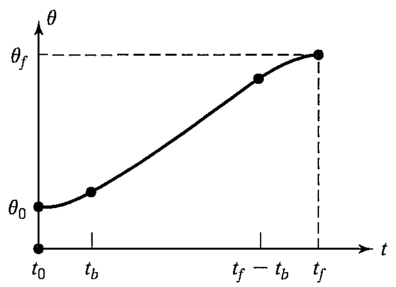

FIGURE 7.6: Linear segment with parabolic blends.

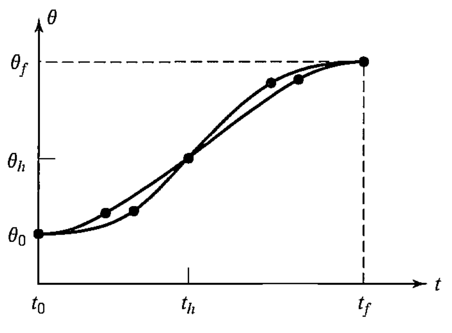

FIGURE 7.7: Linear segment with parabolic blends.

Combining (7.19) and (7.20) and $t = 2t_h$, we get

$$\ddot{\theta}t_b^2 - \ddot{\theta}tt_b + (\theta_f - \theta_0) = 0, \tag{7.21}$$

where t is the desired duration of the motion. Given any θ_f, θ_0, and t, we can follow any of the paths given by the choices of $\ddot{\theta}$ and t_b that satisfy (7.21). Usually, an acceleration, $\ddot{\theta}$, is chosen, and (7.21) is solved for the corresponding t_b. The acceleration chosen must be sufficiently high, or a solution will not exist. Solving (7.21) for t_b in terms of the acceleration and other known parameters, we obtain

$$t_b = \frac{t}{2} - \frac{\sqrt{\ddot{\theta}^2 t^2 - 4\ddot{\theta}(\theta_f - \theta_0)}}{2\ddot{\theta}}. \tag{7.22}$$

The constraint on the acceleration used in the blend is

$$\ddot{\theta} \geq \frac{4(\theta_f - \theta_0)}{t^2} \tag{7.23}$$

When equality occurs in (7.23) the linear portion has shrunk to zero length and the path is composed of two blends that connect with equivalent slope. As the acceleration used becomes larger and larger, the length of the blend region becomes shorter and shorter. In the limit, with infinite acceleration, we are back to the simple linear-interpolation case.

EXAMPLE 7.3

For the same single-segment path discussed in Example 7.1, show two examples of a linear path with parabolic blends.

Figure 7.8(a) shows one possibility where $\ddot{\theta}$ was chosen quite high. In this case we quickly accelerate, then coast at constant velocity, and then decelerate. Figure 7.8(b) shows a trajectory where acceleration is kept quite low, so that the linear section almost disappears.

Linear function with parabolic blends for a path with via points

We now consider linear paths with parabolic blends for the case in which there are an arbitrary number of via points specified. Figure 7.9 shows a set of joint-space via points for some joint θ. Linear functions connect the via points, and parabolic blend regions are added around each via point.

We will use the following notation: Consider three neighboring path points, which we will call points j, k, and l. The duration of the blend region at path point k is t_k. The duration of the linear portion between points j and k is t_{jk}. The overall duration of the segment connecting points j and k is t_{djk}. The velocity during the linear portion is $\dot{\theta}_{jk}$, and the acceleration during the blend at point j is $\ddot{\theta}_j$. See Fig. 7.9 for an example.

As with the single-segment case, there are many possible solutions, depending on the value of acceleration used at each blend. Given all the path points θ_k, the desired durations t_{djk}, and the magnitude of acceleration to use at each path point

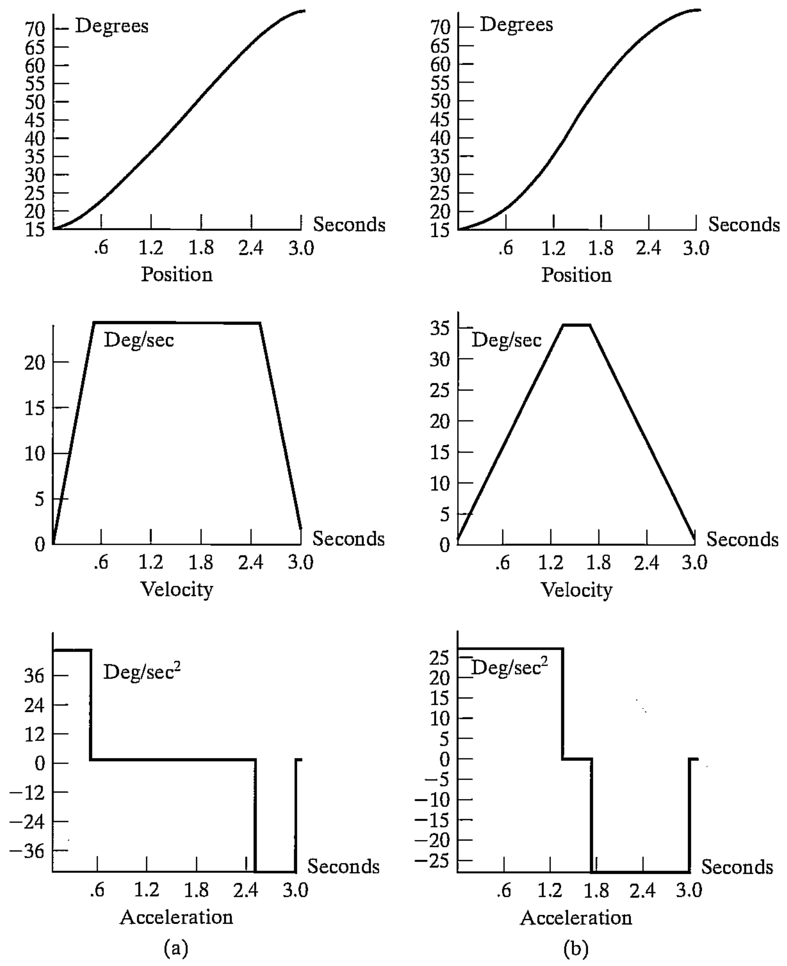

FIGURE 7.8: Position, velocity, and acceleration profiles for linear interpolation with parabolic blends. The set of curves on the left is based on a higher acceleration during the blends than is that on the right.

$|\ddot{\theta}_k|$, we can compute the blend times t_k. For interior path points, this follows simply from the equations

$$\begin{aligned}
\dot{\theta}_{jk} &= \frac{\theta_k - \theta_j}{t_{djk}}, \\
\ddot{\theta}_k &= SGN(\dot{\theta}_{kl} - \dot{\theta}_{jk})|\ddot{\theta}_k|, \\
t_k &= \frac{\dot{\theta}_{kl} - \dot{\theta}_{jk}}{\ddot{\theta}_k}, \\
t_{jk} &= t_{djk} - \frac{1}{2}t_j - \frac{1}{2}t_k.
\end{aligned} \tag{7.24}$$

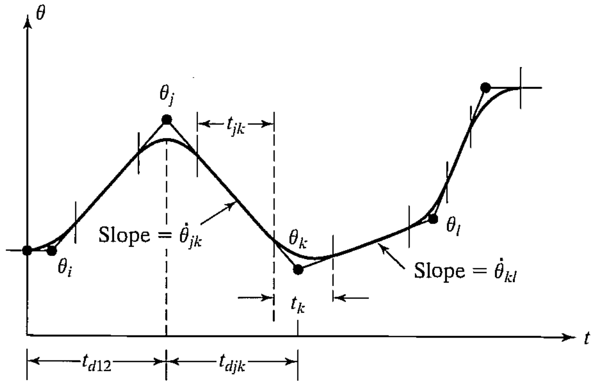

FIGURE 7.9: Multisegment linear path with blends.

The first and last segments must be handled slightly differently, because an entire blend region at one end of the segment must be counted in the total segment's time duration.

For the first segment, we solve for t_1 by equating two expressions for the velocity during the linear phase of the segment:

$$\frac{\theta_2 - \theta_1}{t_{12} - \frac{1}{2}t_1} = \ddot{\theta}_1 t_1. \tag{7.25}$$

This can be solved for t_1, the blend time at the initial point; then $\dot{\theta}_{12}$ and t_{12} are easily computed:

$$\begin{aligned} \ddot{\theta}_1 &= SGN(\theta_2 - \theta_1)|\ddot{\theta}_1|, \\ t_1 &= t_{d12} - \sqrt{t_{d12}^2 - \frac{2(\theta_2 - \theta_1)}{\ddot{\theta}_1}}, \\ \dot{\theta}_{12} &= \frac{\theta_2 - \theta_1}{t_{d12} - \frac{1}{2}t_1}, \\ t_{12} &= t_{d12} - t_1 - \frac{1}{2}t_2. \end{aligned} \tag{7.26}$$

Likewise, for the last segment (the one connecting points $n-1$ and n), we have

$$\frac{\theta_{n-1} - \theta_n}{t_{d(n-1)n} - \frac{1}{2}t_n} = \ddot{\theta}_n t_n, \tag{7.27}$$

which leads to the solution

$$\begin{aligned} \ddot{\theta}_n &= SGN(\theta_{n-1} - \theta_n)|\ddot{\theta}_n|, \\ t_n &= t_{d(n-1)n} - \sqrt{t_{d(n-1)n}^2 + \frac{2(\theta_n - \theta_{n-1})}{\ddot{\theta}_n}}, \end{aligned}$$

$$\dot{\theta}_{(n-1)n} = \frac{\theta_n - \theta_{n-1}}{t_{d(n-1)n} - \frac{1}{2}t_n}, \tag{7.28}$$

$$t_{(n-1)n} = t_{d(n-1)n} - t_n - \frac{1}{2}t_{n-1}.$$

Using (7.24) through (7.28), we can solve for the blend times and velocities for a multisegment path. Usually, the user specifies only the via points and the desired duration of the segments. In this case, the system uses default values for acceleration for each joint. Sometimes, to make things even simpler for the user, the system will calculate durations based on default velocities. At all blends, sufficiently large acceleration must be used so that there is sufficient time to get into the linear portion of the segment before the next blend region starts.

EXAMPLE 7.4

The trajectory of a particular joint is specified as follows: Path points in degrees: 10, 35, 25, 10. The duration of these three segments should be 2, 1, 3 seconds, respectively. The magnitude of the default acceleration to use at all blend points is 50 degrees/second^2. Calculate all segment velocities, blend times, and linear times.

For the first segment, we apply (7.26) to find

$$\ddot{\theta}_1 = 50.0. \tag{7.29}$$

Applying (7.26) to calculate the blend time at the initial point, we get

$$t_1 = 2 - \sqrt{4 - \frac{2(35-10)}{50.0}} = 0.27. \tag{7.30}$$

The velocity, $\dot{\theta}_{12}$, is calculated from (7.26) as

$$\dot{\theta}_{12} = \frac{35-10}{2-0.5(0.27)} = 13.50. \tag{7.31}$$

The velocity, $\dot{\theta}_{23}$, is calculated from (7.24) as

$$\dot{\theta}_{23} = \frac{25-35}{1} = -10.0. \tag{7.32}$$

Next, we apply (7.24) to find

$$\ddot{\theta}_2 = -50.0. \tag{7.33}$$

Then t_2 is calculated from (7.24), and we get

$$t_2 = \frac{-10.0 - 13.50}{-50.0} = 0.47. \tag{7.34}$$

The linear-portion length of segment 1 is then calculated from (7.26):

$$t_{12} = 2 - 0.27 - \tfrac{1}{2}(0.47) = 1.50. \tag{7.35}$$

Next, from (7.29), we have

$$\ddot{\theta}_4 = 50.0. \tag{7.36}$$

So, for the last segment, (7.28) is used to compute t_4, and we have

$$t_4 = 3 - \sqrt{9 + \frac{2(10-25)}{50.0}} = 0.102. \qquad (7.37)$$

The velocity, $\dot{\theta}_{34}$, is calculated from (7.28) as

$$\dot{\theta}_{34} = \frac{10-25}{3-0.050} = -5.10. \qquad (7.38)$$

Next, (7.24) is used to obtain

$$\ddot{\theta}_3 = 50.0. \qquad (7.39)$$

Then t_3 is calculated from (7.24):

$$t_3 = \frac{-5.10 - (-10.0)}{50} = 0.098. \qquad (7.40)$$

Finally, from (7.24), we compute

$$t_{23} = 1 - \tfrac{1}{2}(0.47) - \tfrac{1}{2}(0.098) = 0.716, \qquad (7.41)$$

$$t_{34} = 3 - \tfrac{1}{2}(0.098) - 0.012 = 2.849. \qquad (7.42)$$

The results of these computations constitute a "plan" for the trajectory. At execution time, these numbers would be used by the **path generator** to compute values of θ, $\dot{\theta}$, and $\ddot{\theta}$ at the path-update rate.

In these linear-parabolic-blend splines, note that the via points are not actually reached unless the manipulator comes to a stop. Often, when acceleration capability is sufficiently high, the paths will come quite close to the desired via point. If we wish to actually pass through a point, by coming to a stop, the via point is simply repeated in the path specification.

If the user wishes to specify that the manipulator pass *exactly* through a via point without stopping, this specification can be accommodated by using the same formulation as before, but with the following addition: The system automatically replaces the via point through which we wish the manipulator to pass with two *pseudo via points*, one on each side of the original (as in Fig. 7.10). Then path generation takes place as before. The original via point will now lie in the linear region of the path connecting the two pseudo via points. In addition to requesting that the manipulator pass exactly through a via point, the user can also request that it pass through with a certain velocity. If the user does not specify this velocity, the system chooses it by means of a suitable heuristic. The term **through point** might be used (rather than via point) to specify a path point *through* which we force the manipulator to pass exactly.

7.4 CARTESIAN-SPACE SCHEMES

As was mentioned in Section 7.3, paths computed in joint space can ensure that via and goal points are attained, even when these path points were specified by means of

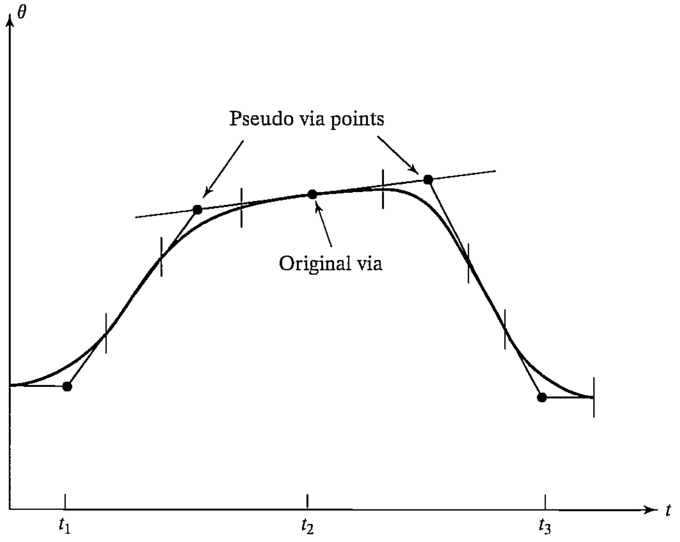

FIGURE 7.10: Use of pseudo via points to create a "through" point.

Cartesian frames. However, the spatial shape of the path taken by the end-effector is not a straight line through space; rather, it is some complicated shape that depends on the particular kinematics of the manipulator being used. In this section, we consider methods of path generation in which the path shapes are described in terms of functions that compute Cartesian position and orientation as functions of time. In this way, we can also specify the spatial shape of the path between path points. The most common path shape is a straight line, but circular, sinusoidal, or other path shapes could be used.

Each path point is usually specified in terms of a desired position and orientation of the tool frame relative to the station frame. In Cartesian-based path-generation schemes, the functions splined together to form a trajectory are functions of time that represent Cartesian variables. These paths can be *planned* directly from the user's definition of path points, which are $\{T\}$ specifications relative to $\{S\}$, without first performing inverse kinematics. However, Cartesian schemes are more computationally expensive to execute, because, at run time, inverse kinematics must be solved at the path-update rate—that is, after the path is generated in Cartesian space, as a last step the inverse kinematic calculation is performed to calculate desired joint angles.

Several schemes for generating Cartesian paths have been proposed in literature from the research and industrial robotics community [1, 2]. In the following section, we introduce one scheme as an example. In this scheme, we are able to use the same linear/parabolic spliner that we developed for the joint-space case.

Cartesian straight-line motion

Often, we would like to be able to specify easily a spatial path that causes the tip of the tool to move through space in a straight line. Obviously, if we specify many closely separated via points lying on a straight line, then the tool tip will appear to follow a straight line, regardless of the choice of smooth function that interconnects the via points. However, it is much more convenient if the tool follows straight-line paths between even widely separated via points. This mode of path specification and execution is called **Cartesian straight-line motion**. Defining motions in terms of straight lines is a subset of the more general capability of **Cartesian motion**, in which arbitrary functions of Cartesian variables as functions of time could be used to specify a path. In a system that allowed general Cartesian motion, such path shapes as ellipses or sinusoids could be executed.

In planning and generating Cartesian straight-line paths, a spline of linear functions with parabolic blends is appropriate. During the linear portion of each segment, all three components of position change in a linear fashion, and the end-effector will move along a linear path in space. However, if we are specifying the orientation as a rotation matrix at each via point, we cannot linearly interpolate its elements, because doing so would not necessarily result in a valid rotation matrix at all times. A rotation matrix must be composed of orthonormal columns, and this condition would not be guaranteed if it were constructed by linear interpolation of matrix elements between two valid matrices. Instead, we will use another representation of orientation.

As stated in Chapter 2, the so-called **angle–axis** representation can be used to specify an orientation with three numbers. If we combine this representation of orientation with the 3 × 1 Cartesian-position representation, we have a 6 × 1 representation of Cartesian position and orientation. Consider a via point specified relative to the station frame as ${}^S_AT$. That is, the frame $\{A\}$ specifies a via point with position of the end-effector given by ${}^SP_{AORG}$, and orientation of the end-effector given by ${}^S_AR$. This rotation matrix can be converted to the angle–axis representation $ROT({}^S\hat{K}_A, \theta_{SA})$—or simply ${}^SK_A$. We will use the symbol χ to represent this 6 × 1 vector of Cartesian position and orientation. Thus, we have

$$
{}^S\chi_A = \begin{bmatrix} {}^SP_{AORG} \\ {}^SK_A \end{bmatrix}, \tag{7.43}
$$

where ${}^SK_A$ is formed by scaling the unit vector ${}^S\hat{K}_A$ by the amount of rotation, θ_{SA}. If every path point is specified in this representation, we then need to describe spline functions that smoothly vary these six quantities from path point to path point as functions of time. If linear splines with parabolic blends are used, the path shape between via points will be linear. When via points are passed, the linear and angular velocity of the end-effector are changed smoothly.

Note that, unlike some other Cartesian-straight-line-motion schemes that have been proposed, this method does not guarantee that rotations occur about a single "equivalent axis" in moving from point to point. Rather, our scheme is a simple one that provides smooth orientation changes and allows the use of the same mathematics we have already developed for planning joint-interpolated trajectories.

One slight complication arises from the fact that the angle–axis representation of orientation is not unique—that is,

$$
({}^S\hat{K}_A, \theta_{SA}) = ({}^S\hat{K}_A, \theta_{SA} + n360^\circ), \tag{7.44}
$$

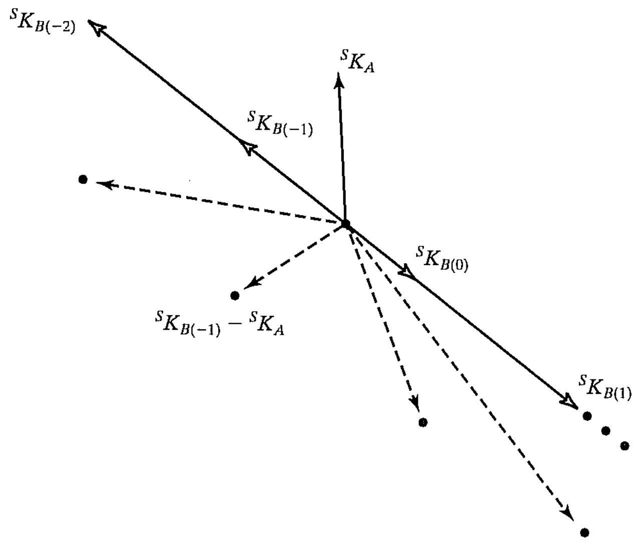

FIGURE 7.11: Choosing angle–axis representation to minimize rotation.

where n is any positive or negative integer. In going from a via point $\{A\}$ to a via point $\{B\}$, the total amount of rotation should be minimized. If our representation of the orientation of $\{A\}$ is given as ^SK_A, we must choose the particular ^SK_B such that $|^SK_B - ^S K_A|$ is minimized. For example, Fig. 7.11 shows four different possible ^SK_B's and their relation to the given ^SK_A. The difference vectors (broken lines) are compared to learn which ^SK_B which will result in minimum rotation—in this case, $^SK_{B(-1)}$.

Once we select the six values of χ for each via point, we can use the same mathematics we have already developed for generating splines that are composed of linear and parabolic sections. However, we must add one more constraint: The blend times for each degree of freedom must be the same. This will ensure that the resultant motion of all the degrees of freedom will be a straight line in space. Because all blend times must be the same, the acceleration used during the blend for each degree of freedom will differ. Hence, we specify a duration of blend, and, using (7.24), we compute the needed acceleration (instead of the other way around). The blend time can be chosen so that a certain upper bound on acceleration is not exceeded.

Many other schemes for representing and interpolating the orientation portion of a Cartesian path can be used. Among these are the use of some of the other 3×1 representations of orientation introduced in Section 2.8. For example, some industrial robots move along Cartesian straight-line paths in which interpolation of orientation is done by means of a representation similar to Z–Y–Z Euler angles.

7.5 GEOMETRIC PROBLEMS WITH CARTESIAN PATHS

Because a continuous correspondence is made between a path shape described in Cartesian space and joint positions, Cartesian paths are prone to various problems relating to workspace and singularities.

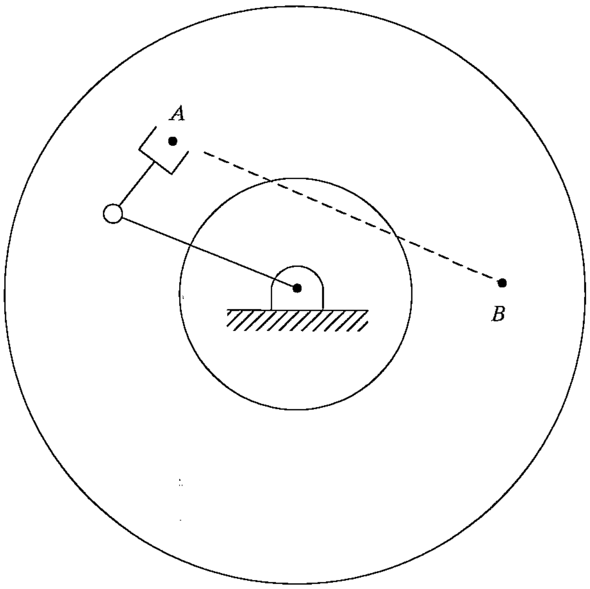

FIGURE 7.12: Cartesian-path problem of type 1.

Problems of type 1: intermediate points unreachable

Although the initial location of the manipulator and the final goal point are both within the manipulator workspace, it is quite possible that not all points lying on a straight line connecting these two points are in the workspace. As an example, consider the planar two-link robot shown in Fig. 7.12 and its associated workspace. In this case, link 2 is shorter than link 1, so the workspace contains a hole in the middle whose radius is the difference between link lengths. Drawn on the workspace is a start point A and a goal point B. Moving from A to B would be no problem in joint space, but if a Cartesian straight-line motion were attempted, intermediate points along the path would not be reachable. This is an example of a situation in which a joint-space path could easily be executed, but a Cartesian straight-line path would fail.[2]

Problems of type 2: high joint rates near singularity

We saw in Chapter 5 that there are locations in the manipulator's workspace where it is impossible to choose finite joint rates that yield the desired velocity of the end-effector in Cartesian space. It should not be surprising, therefore, that there are certain paths (described in Cartesian terms) which are impossible for the manipulator to perform. If, for example, a manipulator is following a Cartesian straight-line path and approaches a singular configuration of the mechanism, one or more joint velocities might increase toward infinity. Because velocities of the

[2]Some robot systems would notify the user of a problem before moving the manipulator; in others, motion would start along the path until some joint reaches its limit, at which time manipulator motion would be halted.

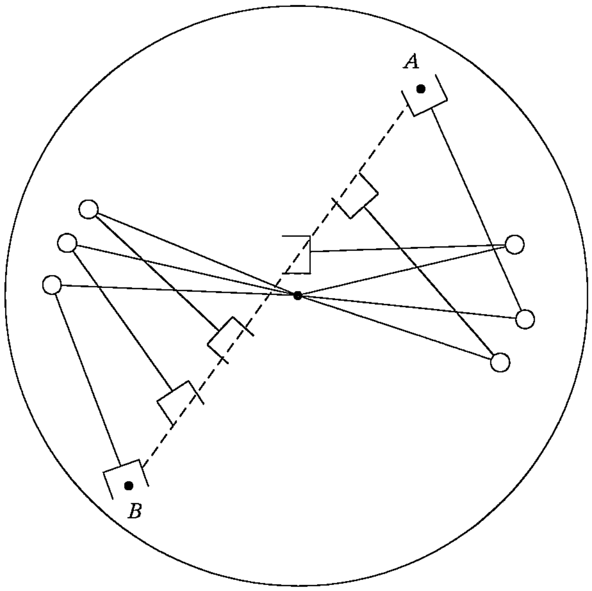

FIGURE 7.13: Cartesian-path problem of type 2.

mechanism are upper bounded, this situation usually results in the manipulator's deviating from the desired path.

As an example, Fig. 7.13 shows a planar two-link (with equal link lengths) moving along a path from point A to point B. The desired trajectory is to move the end tip of the manipulator at constant linear velocity along the straight-line path. In the figure, several intermediate positions of the manipulator have been drawn to help visualize its motion. All points along the path are reachable, but as the robot goes past the middle portion of the path, the velocity of joint one is very high. The closer the path comes to the joint-one axis, the faster this rate will be. One approach is to scale down the overall velocity of the path to a speed where all joints stay within their velocity capabilities. In this way, the desired temporal attributes of the path might be lost, but at least the spatial aspect of the trajectory definition is adhered to.

Problems of type 3: start and goal reachable in different solutions

A third kind of problem that could arise is shown in Fig. 7.14. Here, a planar two-link with equal link lengths has joint limits that restrict the number of solutions with which it can reach a given point in space. In particular, a problem will arise if the goal point cannot be reached in the same physical solution as the robot is in at the start point. In Fig. 7.14, the manipulator can reach all points of the path in some solution, but not in any one solution. In this situation, the manipulator trajectory planning system can detect this problem without ever attempting to move the robot along the path and can signal an error to the user.

To handle these problems with paths specified in Cartesian space, most industrial manipulator-control systems support both joint-space and Cartesian-space

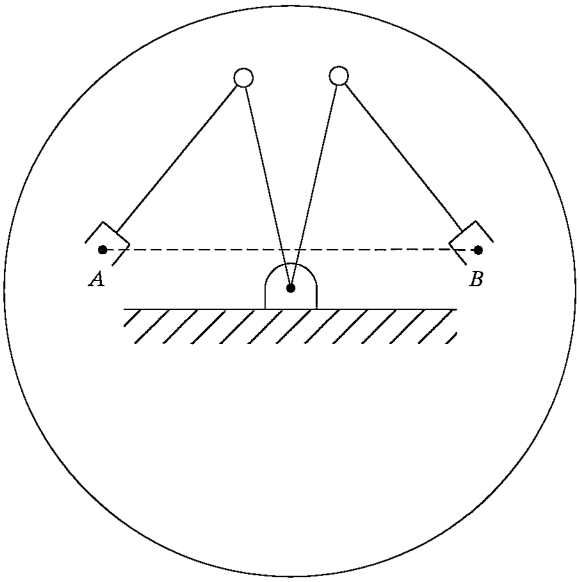

FIGURE 7.14: Cartesian-path problem of type 3.

path generation. The user quickly learns that, because of the difficulties with Cartesian paths, joint-space paths should be used as the default, and Cartesian-space paths should be used only when actually needed by the application.

7.6 PATH GENERATION AT RUN TIME

At **run time**, the path-generator routine constructs the trajectory, usually in terms of θ, $\dot{\theta}$, and $\ddot{\theta}$, and feeds this information to the manipulator's control system. This path generator computes the trajectory at the path-update rate.

Generation of joint-space paths

The result of having planned a path by using any of the splining methods mentioned in Section 7.3 is a set of data for each segment of the trajectory. These data are used by the path generator at run time to calculate θ, $\dot{\theta}$, and $\ddot{\theta}$.

In the case of cubic splines, the path generator simply computes (7.3) as t is advanced. When the end of one segment is reached, a new set of cubic coefficients is recalled, t is set back to zero, and the generation continues.

In the case of linear splines with parabolic blends, the value of time, t, is checked on each update to determine whether we are currently in the linear or the blend portion of the segment. In the linear portion, the trajectory for each joint is calculated as

$$\begin{aligned} \theta &= \theta_j + \dot{\theta}_{jk} t, \\ \dot{\theta} &= \dot{\theta}_{jk}, \\ \ddot{\theta} &= 0, \end{aligned} \tag{7.45}$$

where t is the time since the jth via point and $\dot{\theta}_{jk}$ was calculated at path-planning time from (7.24). In the blend region, the trajectory for each joint is calculated as

$$\begin{aligned} t_{inb} &= t - (\tfrac{1}{2}t_j + t_{jk}), \\ \theta &= \theta_j + \dot{\theta}_{jk}(t - t_{inb}) + \tfrac{1}{2}\ddot{\theta}_k t_{inb}^2, \\ \dot{\theta} &= \dot{\theta}_{jk} + \ddot{\theta}_k t_{inb}, \\ \ddot{\theta} &= \ddot{\theta}_k, \end{aligned} \tag{7.46}$$

where $\dot{\theta}_{jk}$, $\ddot{\theta}_k$, t_j, and t_{jk} were calculated at path-planning time by equations (7.24) through (7.28). This continues, with t being reset to $\frac{1}{2}t_k$ when a new linear segment is entered, until we have worked our way through all the data sets representing the path segments.

Generation of Cartesian-space paths

For the Cartesian-path scheme presented in Section 7.4, we use the path generator for the linear spline with parabolic blends path. However, the values computed represent the Cartesian position and orientation rather than joint-variable values, so we rewrite (7.45) and (7.46) with the symbol x representing a component of the Cartesian position and orientation vector. In the linear portion of the segment, each degree of freedom in χ is calculated as

$$\begin{aligned} x &= x_j + \dot{x}_{jk}t, \\ \dot{x} &= \dot{x}_{jk}, \\ \ddot{x} &= 0, \end{aligned} \tag{7.47}$$

where t is the time since the jth via point and $\dot{x}_{jk}$ was calculated at path-planning time by using an equation analogous to (7.24). In the blend region, the trajectory for each degree of freedom is calculated as

$$\begin{aligned} t_{inb} &= t - (\tfrac{1}{2}t_j + t_{jk}), \\ x &= x_j + \dot{x}_{jk}(t - t_{inb}) + \tfrac{1}{2}\ddot{x}_k t_{inb}^2, \\ \dot{x} &= \dot{x}_{jk} + \ddot{x}_k t_{inb}, \\ \ddot{x} &= \ddot{x}_k, \end{aligned} \tag{7.48}$$

where the quantities $\dot{x}_{jk}$, $\ddot{x}_k$, t_j, and t_{jk} were computed at plan time, just as in the joint-space case.

Finally, this Cartesian trajectory (χ, $\dot{\chi}$, and $\ddot{\chi}$) must be converted into equivalent joint-space quantities. A complete analytical solution to this problem would use the inverse kinematics to calculate joint positions, the inverse Jacobian for velocities, and the inverse Jacobian plus its derivative for accelerations [5]. A simpler way often used in practice is as follows: At the path-update rate, we convert χ into its equivalent frame representation, ${}^S_G T$. We then use the SOLVE routine (see Section 4.8) to calculate the required vector of joint angles, Θ. Numerical differentiation is then used to compute $\dot{\Theta}$ and $\ddot{\Theta}$.[3] Thus, the algorithm is

$$\begin{aligned} \chi &\rightarrow {}^S_G T, \\ \Theta(t) &= SOLVE({}^S_G T), \\ \dot{\Theta}(t) &= \frac{\Theta(t) - \Theta(t - \delta t)}{\delta t}, \\ \ddot{\Theta}(t) &= \frac{\dot{\Theta}(t) - \dot{\Theta}(t - \delta t)}{\delta t}. \end{aligned} \tag{7.49}$$

Then Θ, $\dot{\Theta}$, and $\ddot{\Theta}$ are supplied to the manipulator's control system.

7.7 DESCRIPTION OF PATHS WITH A ROBOT PROGRAMMING LANGUAGE

In Chapter 12, we will discuss **robot programming languages** further. Here, we will illustrate how various types of paths that we have discussed in this chapter might be specified in a robot language. In these examples, we use the syntax of **AL**, a robot programming language developed at Stanford University [6].

The symbols A, B, C, and D stand for variables of type "frame" in the AL-language examples that follow. These frames specify path points that we will assume to have been taught or textually described to the system. Assume that the manipulator begins in position A. To move the manipulator in joint-space mode along linear-parabolic-blend paths, we could say

```
move ARM to C with duration = 3*seconds;
```

To move to the same position and orientation in a straight line we could say

```
move ARM to C linearly with duration = 3*seconds;
```

where the keyword "linearly" denotes that Cartesian straight-line motion is to be used. If duration is not important, the user can omit this specification, and the system will use a default velocity—that is,

```
move ARM to C;
```

A via point can be added, and we can write

```
move ARM to C via B;
```

or a whole set of via points might be specified by

```
move ARM to C via B,A,D;
```

Note that in

```
move ARM to C via B with duration = 6*seconds;
```

the duration is given for the entire motion. The system decides how to split this duration between the two segments. It is possible in AL to specify the duration of a single segment—for example, by

```
move ARM to C via B where duration = 3*seconds;
```

The first segment which leads to point B will have a duration of 3 seconds.

7.8 PLANNING PATHS WHEN USING THE DYNAMIC MODEL

Usually, when paths are planned, we use a default or a maximum acceleration at each blend point. Actually, the amount of acceleration that the manipulator is capable

[3]This differentiation can be done noncausally for preplanned paths, resulting in better-quality $\dot{\Theta}$ and $\ddot{\Theta}$. Also, many control systems do not require a $\ddot{\Theta}$ input, and so it would not be computed.

of at any instant is a function of the dynamics of the arm and the actuator limits. Most actuators are not characterized by a fixed maximum torque or acceleration, but rather by a torque–speed curve.

When we plan a path assuming there is a maximum acceleration at each joint or along each degree of freedom, we are making a tremendous simplification. In order to be careful not to exceed the actual capabilities of the device, this maximum acceleration must be chosen conservatively. Therefore, we are not making full use of the speed capabilities of the manipulator in paths planned by the methods introduced in this chapter.

We might ask the following question: Given a desired spatial path of the end-effector, find the timing information (which turns a description of a spatial path into a trajectory) such that the manipulator reaches the goal point in minimum time. Such problems have been solved by numerical means [7, 8]. The solution takes both the rigid-body dynamics and actuator speed–torque constraint curves into account.

7.9 COLLISION-FREE PATH PLANNING

It would be extremely convenient if we could simply tell the robot system what the desired goal point of the manipulator motion is and let the system determine where and how many via points are required so that the goal is reached without the manipulator's hitting any obstacles. In order to do this, the system must have models of the manipulator, the work area, and all potential obstacles in the area. A second manipulator could even be working in the same area; in that case, each arm would have to be considered a moving obstacle for the other.

Systems that plan collision-free paths are not available commercially. Research in this area has led to two competing principal techniques and to several variations and combinations thereof. One approach solves the problem by forming a connected-graph representation of the free space and then searching the graph for a collision-free path [9–11, 17, 18]. Unfortunately, these techniques have exponential complexity in the number of joints in the device. The second approach is based on creating artificial potential fields around obstacles, which cause the manipulator(s) to avoid the obstacles while they are drawn toward an artificial attractive pole at the goal point [12]. Unfortunately, these methods generally have a local view of the environment and are subject to becoming "stuck" at local minima of the artificial field.

BIBLIOGRAPHY

[1] R.P. Paul and H. Zong, "Robot Motion Trajectory Specification and Generation," 2nd International Symposium on Robotics Research, Kyoto, Japan, August 1984.

[2] R. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories," in *Robot Motion*, Brady et al., Editors, MIT Press, Cambridge, MA, 1983.

[3] C. DeBoor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[4] D. Rogers and J.A. Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill, New York, 1976.

[5] B. Gorla and M. Renaud, *Robots Manipulateurs*, Cepadues-Editions, Toulouse, 1984.

[6] R. Goldman, *Design of an Interactive Manipulator Programming Environment*, UMI Research Press, Ann Arbor, MI, 1985.

[7] J. Bobrow, S. Dubowsky, and J. Gibson, "On the Optimal Control of Robotic Manipulators with Actuator Constraints," *Proceedings of the American Control Conference*, June 1983.

[8] K. Shin and N. McKay, "Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints," *IEEE Transactions on Automatic Control*, June 1985.

[9] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," AI Memo 605, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1980.

[10] T. Lozano-Perez, "A Simple Motion Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 3, June 1987.

[11] R. Brooks, "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:190–197, 1983.

[12] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, Vol. 5, No. 1, Spring 1986.

[13] R.P. Paul, "Robot Manipulators: Mathematics, Programming, and Control," MIT Press, Cambridge, MA, 1981.

[14] R. Castain and R.P. Paul, "An Online Dynamic Trajectory Generator," *The International Journal of Robotics Research*, Vol. 3, 1984.

[15] C.S. Lin and P.R. Chang, "Joint Trajectory of Mechanical Manipulators for Cartesian Path Approximation," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-13, 1983.

[16] C.S. Lin, P.R. Chang, and J.Y.S. Luh, "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots," *IEEE Transactions on Automatic Control*, Vol. AC-28, 1983.

[17] L. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, 12(4): 566–580, 1996.

[18] J. Barraquand, L. Kavraki, J.C. Latombe, T.Y. Li, R. Motwani, and P. Raghavan, "A Random Sampling Scheme for Path Planning," *International Journal of Robotics Research*, 16(6): 759–774, 1997.

EXERCISES

7.1 [8] How many individual cubics are computed when a six-jointed robot moves along a cubic spline path through two via points and stops at a goal point? How many coefficients are stored to describe these cubics?

7.2 [13] A single-link robot with a rotary joint is motionless at $\theta = -5°$. It is desired to move the joint in a smooth manner to $\theta = 80°$ in 4 seconds. Find the coefficients of a cubic which accomplishes this motion and brings the arm to rest at the goal. Plot the position, velocity, and acceleration of the joint as a function of time.

7.3 [14] A single-link robot with a rotary joint is motionless at $\theta = -5°$. It is desired to move the joint in a smooth manner to $\theta = 80°$ in 4 seconds and stop smoothly. Compute the corresponding parameters of a linear trajectory with parabolic blends. Plot the position, velocity, and acceleration of the joint as a function of time.

7.4 [30] Write a path-planning software routine that implements (7.24) through (7.28) in a general way for paths described by an arbitrary number of path points. For example, this routine could be used to solve Example 7.4.

7.5 [18] Sketch graphs of position, velocity, and acceleration for the two-segment continuous-acceleration spline given in Example 7.2. Sketch them for a joint for which $\theta_0 = 5.0°$, $\theta_v = 15.0°$, $\theta_g = 40.0°$, and each segment lasts 1.0 second.

7.6 [18] Sketch graphs of position, velocity, and acceleration for a two-segment spline where each segment is a cubic, using the coefficients as given in (7.11). Sketch them for a joint where $\theta_0 = 5.0°$ for the initial point, $\theta_v = 15.0°$ is a via point, and $\theta_g = 40.0°$ is the goal point. Assume that each segment has a duration of 1.0 second and that the velocity at the via point is to be 17.5 degrees/second.

7.7 [20] Calculate $\dot{\theta}_{12}, \dot{\theta}_{23}, t_1, t_2$, and t_3 for a two-segment linear spline with parabolic blends. (Use (7.24) through (7.28).) For this joint, $\theta_1 = 5.0°, \theta_2 = 15.0°, \theta_3 = 40.0°$. Assume that $t_{d12} = t_{d23} = 1.0$ second and that the default acceleration to use during blends is 80 degrees/second^2. Sketch plots of position, velocity, and acceleration of θ.

7.8 [18] Sketch graphs of position, velocity, and acceleration for the two-segment continuous-acceleration spline given in Example 7.2. Sketch them for a joint for which $\theta_0 = 5.0°$, $\theta_v = 15.0°$, $\theta_g = -10.0°$, and each segment lasts 2.0 seconds.

7.9 [18] Sketch graphs of position, velocity, and acceleration for a two-segment spline where each segment is a cubic, using the coefficients as given in (7.11). Sketch them for a joint where $\theta_0 = 5.0°$ for the initial point, $\theta_v = 15.0°$ is a via point, and $\theta_g = -10.0°$ is the goal point. Assume that each segment has a duration of 2.0 seconds and that the velocity at the via point is to be 0.0 degrees/second.

7.10 [20] Calculate $\dot{\theta}_{12}, \dot{\theta}_{23}, t_1, t_2$, and t_3 for a two-segment linear spline with parabolic blends. (Use (7.24) through (7.28).) For this joint, $\theta_1 = 5.0°, \theta_2 = 15.0°, \theta_3 = -10.0°$. Assume that $t_{d12} = t_{d23} = 2.0$ seconds and that the default acceleration to use during blends is 60 degrees/second^2. Sketch plots of position, velocity, and acceleration of θ.

7.11 [6] Give the 6×1 Cartesian position and orientation representation ${}^S\chi_G$ that is equivalent to ${}^S_GT$ where ${}^S_GR = ROT(\hat{Z}, 30°)$ and ${}^SP_{GORG} = [10.0\ \ 20.0\ \ 30.0]^T$.

7.12 [6] Give the ${}^S_GT$ that is equivalent to the 6×1 Cartesian position and orientation representation ${}^S\chi_G = [5.0\ \ -20.0\ \ 10.0\ \ 45.0\ \ 0.0\ \ 0.0]^T$.

7.13 [30] Write a program that uses the dynamic equations from Section 6.7 (the two-link planar manipulator) to compute the time history of torques needed to move the arm along the trajectory of Exercise 7.8. What are the maximum torques required and where do they occur along the trajectory?

7.14 [32] Write a program that uses the dynamic equations from Section 6.7 (the two-link planar manipulator) to compute the time history of torques needed to move the arm along the trajectory of Exercise 7.8. Make separate plots of the joint torques required due to inertia, velocity terms, and gravity.

7.15 [22] Do Example 7.2 when $t_{f1} \neq t_{f2}$.

7.16 [25] We wish to move a single joint from θ_0 to θ_f starting from rest, ending at rest, in time t_f. The values of θ_0 and θ_f are given, but we wish to compute t_f so that $\|\dot{\theta}(t)\| < \dot{\theta}_{max}$ and $\|\ddot{\theta}(t)\| < \ddot{\theta}_{max}$ for all t, where $\dot{\theta}_{max}$ and $\ddot{\theta}_{max}$ are given positive constants. Use a single cubic segment, and give an expression for t_f and for the cubic's coefficients.

7.17 [10] A single cubic trajectory is given by

$$\theta(t) = 10 + 90t^2 - 60t^3$$

and is used over the time interval from $t = 0$ to $t = 1$. What are the starting and final positions, velocities, and accelerations?

7.18 [12] A single cubic trajectory is given by

$$\theta(t) = 10 + 90t^2 - 60t^3$$

and is used over the time interval from $t = 0$ to $t = 2$. What are the starting and final positions, velocities, and accelerations?

7.19 [13] A single cubic trajectory is given by

$$\theta(t) = 10 + 5t + 70t^2 - 45t^3$$

and is used over the time interval from $t = 0$ to $t = 1$. What are the starting and final positions, velocities, and accelerations?

7.20 [15] A single cubic trajectory is given by

$$\theta(t) = 10 + 5t + 70t^2 - 45t^3$$

and is used over the time interval from $t = 0$ to $t = 2$. What are the starting and final positions, velocities, and accelerations?

PROGRAMMING EXERCISE (PART 7)

1. Write a joint-space, cubic-splined path-planning system. One routine that your system should include is

```
Procedure CUBCOEF (VAR th0, thf, thdot0, thdotf: real; VAR cc:
vec4);
```

where

$$\text{th0} = \text{initial position of } \theta \text{ at beginning of segment},$$

$$\text{thf} = \text{final position of } \theta \text{ at segment end},$$

$$\text{thdot0} = \text{initial velocity of segment},$$

$$\text{thdotf} = \text{final velocity of segment}.$$

These four quantities are inputs, and "cc", an array of the four cubic coefficients, is the output.

Your program should accept up to (at least) five via-point specifications—in the form of tool frame, $\{T\}$, relative to station frame, $\{S\}$—in the usual user form: (x, y, ϕ). To keep life simple, all segments will have the same duration. Your system should solve for the coefficients of the cubics, using some reasonable heuristic for assigning joint velocities at the via points. *Hint*: See option 2 in Section 7.3.

2. Write a path-generator system that calculates a trajectory in joint space based on sets of cubic coefficients for each segment. It must be able to generate the multisegment path you planned in Problem 1. A duration for the segments will be specified by the user. It should produce position, velocity, and acceleration information at the path-update rate, which will also be specified by the user.

3. The manipulator is the same three-link used previously. The definitions of the $\{T\}$ and $\{S\}$ frames are the same as before:

$$^{W}_{T}T = [x\ y\ \theta] = [0.1\ \ 0.2\ \ 30.0],$$

$$^{B}_{S}T = [x\ y\ \theta] = [0.0\ \ 0.0\ \ 0.0].$$

Using a duration of 3.0 seconds per segment, plan and execute the path that starts with the manipulator at position

$$[x_1\ y_1\ \phi_1] = [0.758\ \ 0.173\ \ 0.0],$$

moves through the via points

$$[x_2\ y_2\ \phi_2] = [0.6\ \ -0.3\ \ 45.0]$$

and

$$[x_3\ y_3\ \phi_3] = [-0.4\ \ 0.3\ \ 120.0],$$

and ends at the goal point (in this case, same as initial point)

$$[x_4\ y_4\ \phi_4] = [0.758\ \ 0.173\ \ 0.0].$$

Use a path-update rate of 40 Hz, but print the position only every 0.2 seconds. Print the positions out in terms of Cartesian user form. You don't have to print out velocities or accelerations, though you might be interested in doing so.

MATLAB EXERCISE 7

The goal of this exercise is to implement polynomial joint-space trajectory-generation equations for a single joint. (Multiple joints would require n applications of the result.) Write a MATLAB program to implement the joint-space trajectory generation for the three cases that follow. Report your results for the specific assignments given; for each case, give the polynomial functions for the joint angle, angular velocity, angular acceleration, and angular jerk (the time derivative of acceleration). For each case, plot the results. (Arrange the plots vertically with angle, velocity, acceleration, and then jerk, all with the same time scale—check out the *subplot* MATLAB function to accomplish this.) Don't just plot out results—give some discussion; do your results make sense? Here are the three cases:

a) *Third-order polynomial.* Force the angular velocity to be zero at the start and at the finish. Given $\theta_s = 120^\circ$ (start), $\theta_f = 60^\circ$ (finish), and $t_f = 1$ sec.

b) *Fifth-order polynomial.* Force the angular velocity and acceleration to be zero at the start and at the finish. Given $\theta_s = 120^\circ, \theta_f = 60^\circ$, and $t_f = 1$ sec. Compare your results (functions and plots) with this same example, but using a single third-order polynomial, as in problem (a).

c) *Two third-order polynomials with via point.* Force the angular velocity to be zero at the start and at the finish. Don't force the angular velocity to be zero at the via point—you must match velocity and acceleration at this point for the two polynomials meeting at that point in time. Demonstrate that this condition is satisfied. Given $\theta_s = 60^\circ$ (start), $\theta_v = 120^\circ$ (via), $\theta_f = 30^\circ$ (finish), and $t_1 = t_2 = 1$ sec (relative time steps—i.e., $t_f = 2$ sec).

d) Check the results of (a) and (b) by means of the Corke MATLAB Robotics Toolbox. Try function *jtraj*().