John Lloyd
Vincent Hayward
McGill Research Centre for Intelligent Machines
McGill University
Montréal, Québec, Canada

Trajectory Generation for Sensor-Driven and Time-Varying Tasks

Abstract

In on-line robot trajectory generation, a connecting polynomial is normally used to remove discontinuities in velocity and acceleration between adjacent path segments. This article presents a new technique for performing such transitions in which adjacent path segments are "blended" together, with excess acceleration being removed using an estimate of the initial path velocities. Because this method requires no advance knowledge of the path segments, it can handle situations where the paths are changing with time (as when tracking sensor or control inputs). The method can also be used to adjust the spatial shape of the transition curve (such as to have it pass around or through the "via point"), which may be necessary to handle constraints imposed by different types of manipulator tasks. When the blended paths are nonlinear, it is possible to set a tight bound on the resulting transition acceleration. The blend technique works directly for vector trajectories and can be modified to handle 3-D rotational trajectories. A simple trajectory generation algorithm is presented as an illustration.

1. Introduction

The *trajectory generator* is that part of a manipulator control system that accepts motion commands and produces a stream of set points (usually at a fixed sample rate) that can be tracked by a feedback controller. Motion commands typically prescribe constraints for the manipulator to satisfy, such as target positions, velocities, path shape, arrival times, and stiffnesses or compliant forces. The trajectory generator must then produce a sampled path that meets these constraints as closely as possible. A central problem in trajectory generation is that the specified task constraints often conflict with the kinematic or dynamic constraints of the manipulator itself. In particular, the final path must be smooth, with no discontinuities in the velocity and possibly higher derivatives.

Current approaches to trajectory generation can be roughly grouped into off-line and on-line techniques. If the trajectory is computed off-line (i.e., before the robot program is actually run), then time is available to compute a trajectory that addresses both task and manipulator constraints in some optimal fashion. Common optimality criteria include minimum time and minimum path error, in the presence of various constraints. This problem has been extensively studied in the literature. Lin et al. (1983) describes a spline-based method for generating time-optimal paths satisfying constraints in the velocity, acceleration, and jerk; Shin and McKay (1985, 1986) and Bobrow et al. (1985) describe ways for computing time-optimal trajectories along parametric paths subject to torque constraints.

Once computed, however, such trajectories are generally difficult to modify in response to real-time sensor information, although this problem may be tempered somewhat by relaxing the optimality criterion and forming a trajectory using localized splines (Thompson and Patel 1987). With on-line trajectory generation, the manipulator set points are computed in real time, usually at some known sample rate, at the same time they are sent to the controller. This maximizes the opportunity to respond to sensor-driven events, at the expense of creating paths that utilize only very local (and usually suboptimal) constraints. Recent increases in CPU power currently permit more sophisticated trajectories to be computed on-line.

A classic technique for on-line trajectory generation is to compute idealized path segments that satisfy program requirements but ignore the manipulator dynamics, and then join these together using polynomial fits applied across a *transition window* (Paul 1981, Taylor 1979).

We believe that this paradigm has more utility than is generally realized. In addition to permitting sensor responsiveness, it effectively decouples the problem of meeting both program and manipulator constraints: during the transition window, the program path constraints are relaxed, and the dynamics constraints predominate. Between transition windows, where the path segments may accelerate very little, the program constraints predominate. The amount of manipulator torque required for the transition is proportional to the inverse square of the length of the transition window (Hollerbach 1984), and

The International Journal of Robotics Research,
Vol. 12, No. 4, August 1993, pp. 380–393,

so can be easily controlled. The importance of decoupling these constraints stems from the fact that in a complex robot task, we frequently need to impose a far richer set of constraints on the manipulator motion than just traversing a path in the shortest time. For instance, it is often necessary to have the manipulator travel at a constant speed, or exert a certain force, or stop and wait for some event. When such needs are absent, it may be possible to "extend" the transition window so that it covers the entire motion and compute each path as a polynomial fit to the next goal position (Andersson 1988).

This article describes a new transition window technique that uses blend functions to connect manipulator path segments. Use of this paradigm offers the following advantages:

- Path segments may be nonlinear, and knowledge of their future behavior is not required.
- The spatial profile of the transition can be controlled by the adjustment of a pair of scalar parameters.

The first item is important whenever the manipulator trajectory is adjusted on-line by inputs from sensors or operator controls such as joysticks and hand controllers (the latter being common in shared control and telerobotics applications) (Hayati and Venkataraman 1989; Hirzinger and Dietrich 1986). In these situations, the precise manipulator trajectory is not known in advance. The second item is useful in situations where the transition shape must conform to certain task constraints. For instance, a transition associated with going around a corner usually "cuts" the corner on the inside. However, if the manipulator is tracking the outside of an angular solid, the transition shape must be adjusted so that it lies on the outside of the solid. Examples showing how the transition shape can be controlled are given in Section 5.

Our method of transition blending is applicable to trajectories described by vectors and also works for the 3-D rotational paths associated with Cartesian trajectories, using some modifications as described in Section 7. The methods outlined in this article have been fully implemented and form the core of the trajectory generators for the robot programming systems Multi-RCCL (Lloyd et al. 1988) and Kali (Hayward et al. 1989).

Section 2 reviews the conventional transition window paradigm. Transition blending is introduced in Sections 3 and 4. Section 5 describes how to control the spatial shape of the transition, Section 6 contains an example trajectory algorithm to illustrate the ideas of the paper, and Section 7 discusses the modifications necessary to handle 3-D rotations.

Throughout the article, vectors will be indicated with lower case boldface letters (e.g., $\mathbf{v}$), and matrices will be indicated with upper case boldface letters (e.g., $\mathbf{M}$).

2. Review of the Transition Window Technique

Suppose that a manipulator is following a particular path $\mathbf{x}_1(t)$ in some coordinate system and that at time t_s switches to a second path $\mathbf{x}_2(t)$. The time dependencies of these paths may be induced by both the trajectory generator (such as by interpolating between via points), and by external influences (such as by tracking a moving target). Very little is assumed about paths $\mathbf{x}_1$ and $\mathbf{x}_2$, except that individually they provide a smooth trajectory for the robot to follow (i.e., they have no discontinuities in position, velocity, and possibly higher derivatives).

If no transition is applied, then the switching between paths at $t = t_s$ will generally create a discontinuity in acceleration, velocity, and possibly position. The conventional remedy for this (Paul 1981; Taylor 1979) amounts to connecting $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ with a smooth polynomial that spans an interval $t \in [t_s - \tau, t_s + \tau]$, for some appropriate value of τ (Fig. 1). This involves (1) determining an appropriate length of time (2τ) for the transition and (2) forming the connecting polynomial.

A simple way to estimate the necessary transition time is to divide the magnitude of the velocity change by some desired reference acceleration a_r:

$$2\tau \approx \frac{\|\dot{\mathbf{x}}_2(t_s + \tau) - \dot{\mathbf{x}}_1(t_s - \tau)\|}{a_r}. \tag{1}$$

Different variations on this equation can be used. In joint coordinates, one may compute individual transition times for each joint and then use the maximum, whereas in Cartesian coordinates, one may compute separate transition times for the translational and rotational path components and then take the maximum of these. The acceleration limit itself (a_r) can be determined by considering actuator torque limits, applying some estimate of the manipulator dynamic capacity, and mapping back into

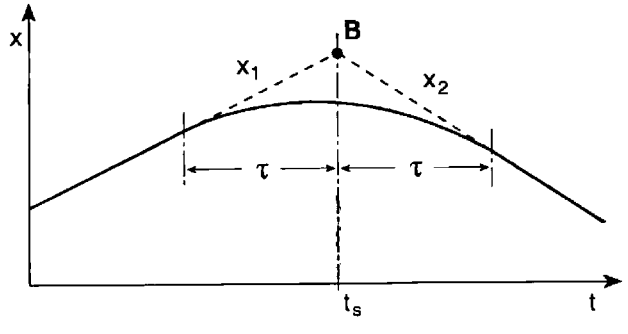


Fig. 1. Illustration of a path segment transition in one dimension. The path segments x_1 and x_2, which intersect at B, are indicated by hatched lines, and the final connected path is indicated by a solid line.

the appropriate coordinate system. This issue is important but is outside the scope of this article and is assumed solved without loss of generality.

To form the connecting polynomial $\mathbf{x}(t)$, it is convenient to define a new time coordinate s,

$$s \equiv \frac{t - t_s + \tau}{2\tau}, \tag{2}$$

so that the transition occurs during the interval $s \in [0, 1]$. For each of the path vector components i, the polynomial $\mathbf{x}(s)$ must satisfy the following boundary conditions with $\mathbf{x}_1(s)$, $\mathbf{x}_2(s)$, and their first and second derivatives:

$$x_i(0) = x_{1i}(0) \equiv p_{1i}, \quad x_i(1) = x_{2i}(1) \equiv p_{2i}, \tag{3}$$
$$\dot{x}_i(0) = \dot{x}_{1i}(0) \equiv v_{1i}, \quad \dot{x}_i(1) = \dot{x}_{2i}(1) \equiv v_{2i}, \tag{4}$$
$$\ddot{x}_i(0) = \ddot{x}_{1i}(0) \equiv a_{1i}, \quad \ddot{x}_i(1) = \ddot{x}_{2i}(1) \equiv a_{2i}. \tag{5}$$

These can be satisfied using a fifth-degree polynomial whose coefficients, described by the vector $\mathbf{c} = (c_5, c_4, c_3, c_2, c_1, c_0)^T$, can be found using a Hermite boundary condition matrix $\mathbf{H}$ (Foley and Van Dam 1984):

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 5 & 4 & 3 & 2 & 1 & 0 \\ 20 & 12 & 6 & 2 & 0 & 0 \end{pmatrix}. \tag{6}$$

If the boundary conditions are described by a vector $\mathbf{b} = (p_{1i}, v_{1i}, a_{1i}, p_{2i}, v_{2i}, a_{2i})^T$, $\mathbf{c}$ may be determined from

$$\mathbf{c} = \mathbf{H}^{-1}\,\mathbf{b}. \tag{7}$$

This can be expanded to yield

$$\begin{aligned} c_5 &= (a_{2i} - a_{1i})/2 + 6(p_{2i} - p_{1i}) - 3(v_{2i} + v_{1i}), \\ c_4 &= (3a_{1i} - 2a_{2i})/2 + 15(p_{1i} - p_{2i}) + 8v_{1i} + 7v_{2i}, \\ c_3 &= (a_{2i} - 2a_{1i})/2 + 10(p_{2i} - p_{1i}) - 6v_{1i} - 4v_{2i}, \\ c_2 &= a_{1i}/2, \\ c_1 &= v_{1i}, \\ c_0 &= p_{1i}, \end{aligned} \tag{8}$$

which can be used to construct a connecting polynomial for each coordinate i.

The method just described constitutes the basis for most current on-line trajectory generators. It is simple and easy to implement, but suffers from two deficiencies:

- To compute the connecting polynomial, it is necessary, at the start of the transition, to know $\mathbf{x}_2$ and its first two derivatives at $s = 1$. However, this may not be possible if the path is tracking external sensor signals or control inputs.
- There is no particularly easy way to control the transition's *shape* in either space or in time. If the two paths intersect at $s = 1/2$ (as in Figure 1), then the smoothed path undercuts this intersection. Although this is often desirable, task constraints may occasionally make it preferable to travel through the intersection, or even to overshoot it.

The next two sections will present a solution to both of these problems.

3. Path Segment Blending

We can avoid the problem of path uncertainty in the following way: instead of connecting the paths $\mathbf{x}_1$ and $\mathbf{x}_2$ by a fixed polynomial, we can simply blend them together using a convex average. During the transition interval, the final path $\mathbf{x}(s)$ is computed from

$$\mathbf{x}(s) = \mathbf{x}_1(s) + \alpha(s)(\mathbf{x}_2(s) - \mathbf{x}_1(s)), \tag{9}$$

where $\alpha(s)$ is a blend function that smoothly increases from 0 to 1 over the interval $s \in [0, 1]$ and satisfies the boundary conditions

$$\alpha(0) = 0, \quad \alpha(1) = 1, \tag{10}$$
$$\dot{\alpha}(0) = 0, \quad \dot{\alpha}(1) = 0, \tag{11}$$
$$\ddot{\alpha}(0) = 0, \quad \ddot{\alpha}(1) = 0. \tag{12}$$

These conditions are met if α is defined by the polynomial

$$\alpha(s) = 6s^5 - 15s^4 + 10s^3. \tag{13}$$

A one-dimensional example of path blending is illustrated in Figure 2. The blended path $\mathbf{x}(s)$ meets all of the boundary conditions specified in (4) without requiring any a priori knowledge of either path.

There is a problem, however, in that the blended path tends to accelerate (and then decelerate) more than necessary during the transition; notice the increase in path slope after the transition begins. It turns out that this can

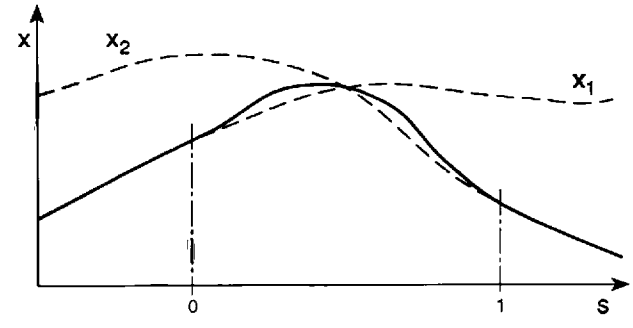


Fig. 2. Direct blend of paths x_1 and x_2.

be minimized by using an estimate of the transition velocity change (for which at least an approximate value should be available to set the transition time in the first place). This is shown with the following calculations:

Assume that we can compensate for the extra acceleration by adding to the blended path an additional term $\beta(s)\,\mathbf{u}$, where β is some polynomial in s and $\mathbf{u}$ is a fixed vector, so that

$$\mathbf{x}(s) = \mathbf{x}_1(s) + \alpha(s)(\mathbf{x}_2(s) - \mathbf{x}_1(s)) + \beta(s)\mathbf{u}. \tag{14}$$

What degree should β be? To avoid disturbing the existing boundary conditions, we must have $\beta(0) = \dot{\beta}(0) = \ddot{\beta}(0) = \beta(1) = \dot{\beta}(1) = \ddot{\beta}(1) = 0$. This requires a fifth-degree polynomial, and to allow β to do something useful as well requires at least one more degree, so we let β be of degree 6.

The relationship between the boundary conditions and the coefficients of β can be expressed using the (underdetermined) Hermite matrix

$$\mathbf{H}' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 30 & 20 & 12 & 6 & 2 & 0 & 0 \end{pmatrix}. \tag{15}$$

If there is one set of coefficients $\mathbf{c}_0$ that satisfies this matrix, then all other such sets $\mathbf{c}$ must satisfy

$$\mathbf{c} = \mathbf{c}_0 + \aleph(\mathbf{H}')\mathbf{k},$$

where $\mathbf{k}$ is an arbitrary vector whose length equals the dimension of the matrix's null space. In the present case, the boundary conditions are all zero, implying $\mathbf{c}_0 = 0$. The null space of $\mathbf{H}'$ has dimension 1, implying $\mathbf{k} = k$, and is spanned by $(1, -3, 3, -1, 0, 0, 0)$, which yields the following form for $\beta(s)$:

$$\beta(s) = k(s^6 - 3s^5 + 3s^4 - s^3).$$

k is a free parameter that may be used to adjust the characteristics of the transition curve. Because the term being sought is of the form $\beta(s)\,\mathbf{u}$, we can, without loss of generality, set $k = 1$ and adjust the magnitude of $\mathbf{u}$ instead. Now $\mathbf{u}$ will be determined so as to minimize the average values of $\|\ddot{\mathbf{x}}(s)\|$; this is reasonable, since the purpose of $\beta(s)\,\mathbf{u}$ is to remove excess accelerations. For purposes of this analysis, we will assume that the paths $\mathbf{x}_1$ and $\mathbf{x}_2$ are close to linear during the transition interval, which implies that they can be approximated by

$$\mathbf{x}_1(s) = \mathbf{b}_1 + \mathbf{v}_1 s, \tag{16}$$

$$\mathbf{x}_2(s) = \mathbf{b}_2 + \mathbf{v}_2 s, \tag{17}$$

where $\mathbf{b}_i$ and $\mathbf{v}_i$ are fixed. Let $\mathbf{v}_d \equiv \mathbf{v}_2 - \mathbf{v}_1$ be the difference in path velocities, and let $\mathbf{b}_d \equiv \mathbf{b}_2 - \mathbf{b}_1$ be the difference in path positions, so that (14) becomes

$$\mathbf{x}(s) = \mathbf{v}_1 s + \mathbf{b}_1 + \alpha(s)(\mathbf{v}_d s + \mathbf{b}_d) + \beta(s)\mathbf{u}. \tag{18}$$

For simplicity, the explicit dependence on s of $\alpha(s)$, $\beta(s)$, and their derivatives will be omitted in most of the remaining discussion. Eq. (18) can be differentiated twice to determine

$$\ddot{\mathbf{x}}(s) = (\ddot{\alpha}s + 2\dot{\alpha})\mathbf{v}_d + \ddot{\alpha}\mathbf{b}_d + \ddot{\beta}\mathbf{u}. \tag{19}$$

Minimizing the average acceleration is equivalent to minimizing the integral

$$\begin{aligned}\int_0^1 \|\ddot{\mathbf{x}}(s)\|^2\,ds = \int_0^1 \Big[&(\ddot{\alpha}s + 2\dot{\alpha})^2\|\mathbf{v}_d\|^2 + \ddot{\alpha}^2\|\mathbf{b}_d\|^2 \\ &+ \ddot{\beta}^2\|\mathbf{u}\|^2 \\ &+ 2\ddot{\alpha}(\ddot{\alpha}s + 2\dot{\alpha})\mathbf{v}_d \cdot \mathbf{b}_d \\ &+ 2\ddot{\beta}(\ddot{\alpha}s + 2\dot{\alpha})\mathbf{v}_d \cdot \mathbf{u} + 2\ddot{\alpha}\ddot{\beta}\,\mathbf{b}_d \cdot \mathbf{u}\Big]\,ds,\end{aligned}$$

which reduces, after some work, to

$$\begin{aligned}&1/7\,(60\|\mathbf{v}_d\|^2 + 120\mathbf{v}_d \cdot \mathbf{b}_d + 120\|\mathbf{b}_d\|^2) \\ &\qquad + 1/35\,(30\mathbf{v}_d \cdot \mathbf{u} + 2\|\mathbf{u}\|^2).\end{aligned} \tag{20}$$

The first term of (20) does not depend on $\mathbf{u}$ and therefore does not figure into the calculation. To minimize the second term, resolve $\mathbf{u}$ into two components $\mathbf{u}_v$ and $\mathbf{u}_\perp$, which are parallel and perpendicular, respectively, to $\mathbf{v}_d$. The right side then becomes

$$1/35\,(30\|\mathbf{v}_d\|\|\mathbf{u}_v\| + 2\|\mathbf{u}_v\|^2 + 2\|\mathbf{u}_\perp\|^2),$$

which is minimized by setting $\mathbf{u}_\perp = 0$ and $\mathbf{u}_v = -15/2\,\mathbf{v}_d = \mathbf{u}$.

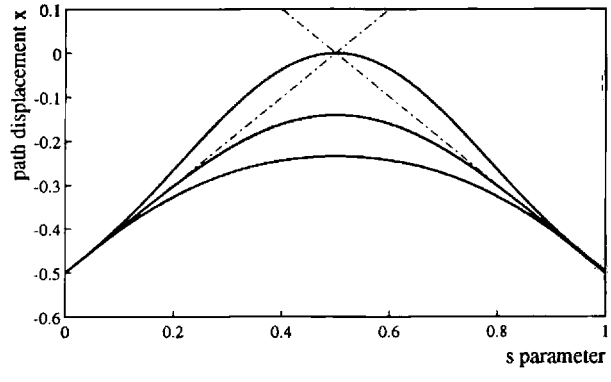


Fig. 3. Blend transitions for two linear path segments intersecting at $s = 1/2$ and with κ set to 0 (top), 6 (middle), and 15/2 (bottom).

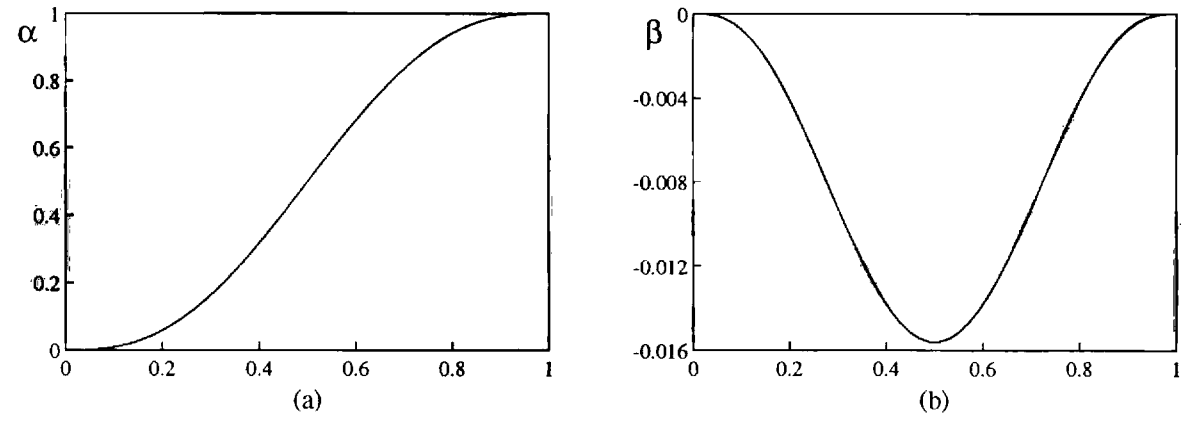


Fig. 4. (A), Plot of $\alpha(s)$. (B), Plot of $\beta(s)$.

Note that $\mathbf{v}_d$ is the transition velocity change mentioned earlier. Letting $\mathbf{u} = -\kappa\mathbf{v}_d$, the full formula for the transition blend becomes

$$\mathbf{x}(s) = \mathbf{x}_1(s) + \alpha(\mathbf{x}_2(s) - \mathbf{x}_1(s)) - \kappa\beta\mathbf{v}_d. \qquad (21)$$

κ is a parameter that controls the amount of acceleration compensation to apply; very qualitatively, it provides a sort of "damper" control on the transition curve. Typically, we will want to set κ to the optimal value of 15/2 derived above, but other values can be used: for instance, if κ is set to 6 and the path segments are linear and intersect at $s = 1/2$, the transition curve becomes a fifth-order polynomial identical to the one originally described in Paul (1981). If we don't want to apply compensation at all, κ can be set to 0. Figure 3 shows some transition curves with different values of κ. The $\alpha(s)$ and $\beta(s)$ blend functions are graphed in Figure 4.

In the case where $\mathbf{x}_1$ and $\mathbf{x}_2$ are not linear, $\mathbf{v}_d$ and $\mathbf{b}_d$ are set to the *initial* differences in velocity and position, and $\kappa\beta\mathbf{v}_d$ will then compensate for accelerations associated with the linear path components.

It should be noted that changing the value of κ changes the overall acceleration associated with the transition, making it necessary to adjust the transition time. This is discussed further in Section 5.3.

4. Blending Nonlinear Paths

Some of the analysis in the preceding section assumed that paths $\mathbf{x}_1$ and $\mathbf{x}_2$ are linear. If the paths are nonlinear, the transition will still be smooth, as the boundary conditions of (4) will still be satisfied. However, what will be the acceleration magnitude $\|\ddot{\mathbf{x}}(s)\|$? This is a reasonable question, as the primary purpose of the transition is to limit $\|\ddot{\mathbf{x}}(s)\|$.

If $\mathbf{x}_1$ and $\mathbf{x}_2$ are nonlinear, they are accelerating; in particular, given that we have required them to have no discontinuities in position or velocity, we can expand each of them as a Taylor series about $s = 0$:

$$\mathbf{x}_1(s) = \mathbf{b}_1 + \mathbf{v}_1 s + \mathbf{y}_1(s), \qquad (22)$$

$$\mathbf{x}_2(s) = \mathbf{b}_2 + \mathbf{v}_2 s + \mathbf{y}_2(s), \qquad (23)$$

where

$$\mathbf{y}_i(s) = \iint_0^s \ddot{\mathbf{x}}_i(\sigma)\, d\sigma^2.$$

This formulation separates the paths into linear and nonlinear components. The linear component (formed from $\mathbf{b}_i$ and $\mathbf{v}_i$) is that part that depends on the position and velocity of the paths at the start of the transition (when $s = 0$); because of this, it is also the part we can *predict*. The nonlinear component depends on the future acceleration profile and is the part of the path that cannot be predicted.[1]

By applying (21) to the paths in (22), differentiating twice, and letting $\mathbf{y}_d(s) \equiv \mathbf{y}_2(s) - \mathbf{y}_1(s)$, the acceleration is found to be

$$\ddot{\mathbf{x}}(s) = \ddot{\mathbf{x}}_L(s) + \ddot{\mathbf{y}}(s),$$

where

$$\ddot{\mathbf{x}}_L(s) = \ddot{\alpha}(\mathbf{b}_d + s\mathbf{v}_d) + (2\dot{\alpha} - \kappa\ddot{\beta})\mathbf{v}_d, \quad \text{(linear component)}$$

$$\ddot{\mathbf{y}}(s) = \ddot{\mathbf{y}}_1(s) + \alpha\ddot{\mathbf{y}}_d(s) + 2\dot{\alpha}\dot{\mathbf{y}}_d(s) + \ddot{\alpha}\mathbf{y}_d(s). \quad \text{(nonlinear component)}$$

The linear component $\ddot{\mathbf{x}}_L(s)$ is the quantity that is controlled when (1) (or a similar relation) is used to determine the transition time. An important question is how much *additional* acceleration can be imposed by the nonlinear component $\ddot{\mathbf{y}}(s)$. It turns out that if the individual

1. Prediction based on the extrapolation of second and higher order derivatives is possible in theory but is generally hard in practice because of the difficulty in estimating these quantities reliably, coupled with the sensitivity of such predictions to errors in the initial conditions.

path accelerations $\ddot{\mathbf{y}}_1(s)$ and $\ddot{\mathbf{y}}_2(s)$ are bounded, then the bound on $\|\ddot{\mathbf{y}}(s)\|$ is described by the following theorem:

THEOREM 1. Let $\mathbf{y}_1(t)$ and $\mathbf{y}_2(t)$ be two paths in space, and let $\mathbf{y}(t)$ be the blend of these two paths such that $\mathbf{y}(t) \equiv \mathbf{y}_1(t) + \alpha(s)(\mathbf{y}_2(t) - \mathbf{y}_1(t))$, for $t \in [0, T]$, with $\alpha(s) \equiv 6s^5 - 15s^4 + 10s^3$ and $s \equiv t/T$. If $\mathbf{y}_d(t) \equiv \mathbf{y}_2(t) - \mathbf{y}_1(t)$ satisfies the boundary conditions $\mathbf{y}_d(0) = \dot{\mathbf{y}}_d(0) = 0$, and $\|\ddot{\mathbf{y}}_1(t)\|, \|\ddot{\mathbf{y}}_2(t)\| \leq A$, then

$$\|\ddot{\mathbf{y}}(t)\| \leq (19/4)A,$$

where this bound is tight.

Note that the definition of $\mathbf{y}_i$ implies that the boundary conditions $\mathbf{y}_d(0) = \dot{\mathbf{y}}_d(0) = 0$ are satisfied and that $\ddot{\mathbf{x}}_1 = \ddot{\mathbf{y}}_1$ and $\ddot{\mathbf{x}}_2 = \ddot{\mathbf{y}}_2$. The proof of the theorem is rather tedious, and is therefore deferred to the Appendix.

Theorem 1 indicates that it is possible for the transition blend to amplify the accelerations of the paths being blended, and that if we want to be sure $\|\ddot{\mathbf{y}}(s)\|$ is no larger than $\|\ddot{\mathbf{x}}_L(s)\|$, then we should ensure that (roughly) $\|\ddot{\mathbf{x}}_1(s)\|, \|\ddot{\mathbf{x}}_2(s)\| \leq 1/5\|\ddot{\mathbf{x}}_L(s)\|$. The bound of the theorem is tight in the case where both paths are accelerating directly away from each other.

5. Adjusting the Transition Shape

For some applications, it may be desirable to adjust the actual "shape" of the path segment transitions. For instance, we may want to actually go through the intersection point of paths $\mathbf{x}_1$ and $\mathbf{x}_2$ or perhaps overshoot the intersection point altogether.

5.1. Halt and Start Transition Components

An easy way to go through the intersection point is to set $\kappa = 0$ in equation (21), although this has the disadvantage of causing the manipulator to speed up during the transition. In this section, we will develop a simple technique for adjusting the shape of the transition *without* causing the manipulator to speed up. For the purposes of analysis, it will again be assumed that the path segments are approximately linear.

It is useful to think of a transition as being the superposition of two actions: a *halt* transition, which brings the motion along path $\mathbf{x}_1$ to rest at some fixed target point $\mathbf{p}$, and a *start* transition, which initiates a motion along path $\mathbf{x}_2$ away from $\mathbf{p}$. This is illustrated in Figure 5, where $\mathbf{p}$ has been set equal to $\mathbf{x}_1(1/2)$ (i.e., the point along $\mathbf{x}_1$ that would have been reached at $s = 1/2$ had there been no transition). The notion can be expressed by deliberately expanding (21) to include $\mathbf{p}$:

$$\mathbf{x}(s) = \underbrace{\mathbf{x}_1(s) + \alpha(\mathbf{p} - \mathbf{x}_1(s)) + \kappa\beta\dot{\mathbf{x}}_1(0)}_{\text{halt component}}$$

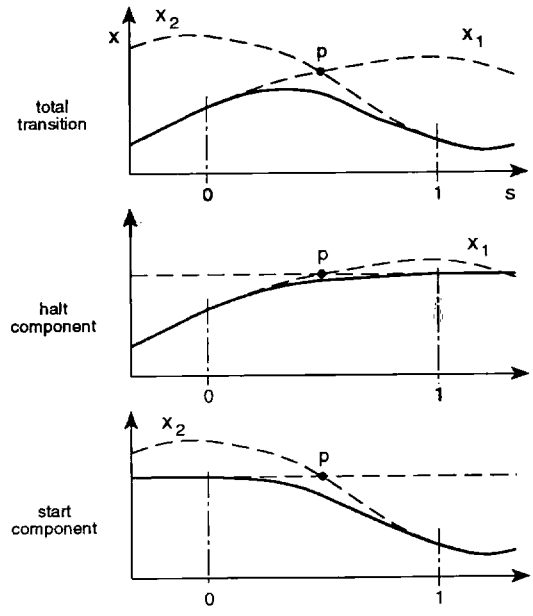


Fig. 5. The transition between paths x_1 and x_2 (top figure) *can be thought of as the combination of a halt component, which brings the motion along x_1 to rest at $p = x_1(1/2)$* (middle figure), *and a start component that initiates motion along x_2 away from p* (bottom figure).

$$+ \underbrace{\alpha(\mathbf{x}_2(s) - \mathbf{p}) - \kappa\beta\dot{\mathbf{x}}_2(0)}_{\text{start component}} - \mathbf{p}. \qquad (24)$$

It should be understood that $\mathbf{p}$ is somewhat artificial; even if $\mathbf{x}_1(s)$ is not heading toward a fixed point, $\mathbf{p}$ can be approximated by extrapolating $\dot{\mathbf{x}}(0)$.

Now consider what happens to the "halt" component if $\mathbf{x}_1$ intersects $\mathbf{p}$ at $s = 0$ instead of $s = 1/2$ (i.e., $\mathbf{p}$ is set to $\mathbf{x}_1(0)$): the resulting motion overshoots the target point and then slowly comes to rest there (Fig. 6). Alternatively, if $\mathbf{x}_1$ intersects $\mathbf{p}$ at $s = 1$, then the resulting motion speeds up to "catch" the target before coming to rest. We can also adjust the "start" component of the transition so that $\mathbf{x}_2$ intersects $\mathbf{p}$ at some s value other than $1/2$; Figure 6 shows the resulting motions when the intersection value is set to 0 and 1.

By controlling the timing of the halt and start transition components in this way, the shape of the transition can be adjusted. We define the *preview* parameters π_h and π_s to be the values of s for which the halt and start components intersect $\mathbf{p}$. Both of these have a nominal value of $1/2$. Qualitatively, each preview factor is a measure of how much the trajectory generator is allowed to "plan ahead" in computing the associated transition component.

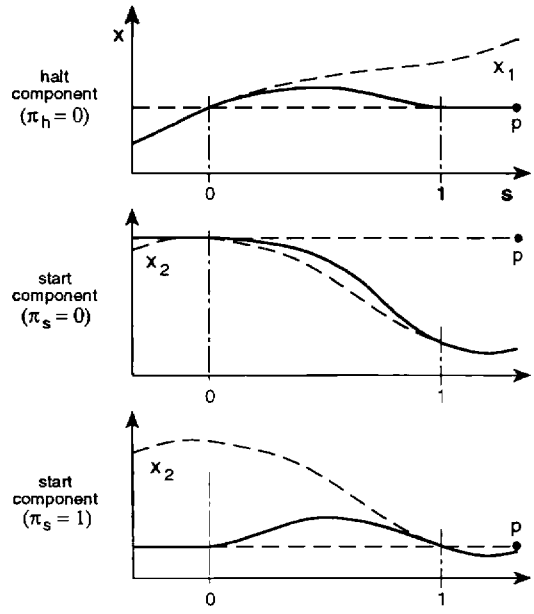
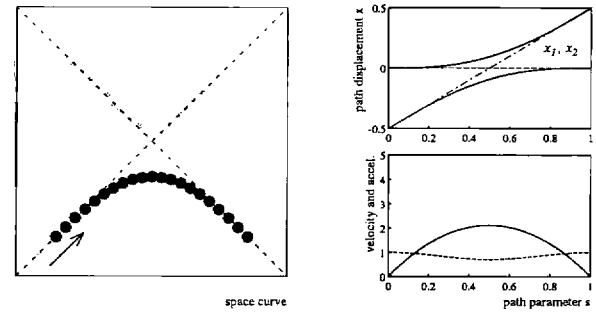


Fig. 6. Top, *A transition halt component where x_1 intersects the stationary position* **p** *at $s = 0$.* Middle, *A transition start component where x_2 intersects the stationary position at $s = 0$.* Bottom, *Start component where x_2 intersects the stationary position at $s = 1$.*

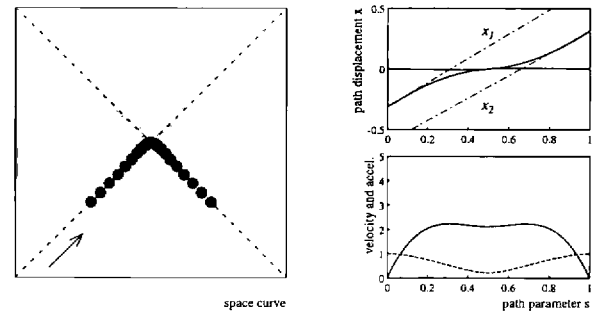
5.2. *Using the Preview Parameters*

The effect of different settings of π_h and π_s on the transition profile is illustrated in Figure 7. In each of these examples, the path segments are linear, have equal speeds, and intersect at right angles. All transitions are computed with $\kappa = 6$. The spatial trajectory is displayed in the large box on the left, with dots indicating the "manipulator" position at different times. Normal path motion and transitional motion are indicated by light gray and dark gray dots, respectively. The box at the upper right shows the displacements associated with the transition halt and start components, as a function of s, while the box at the lower right shows the magnitudes of the manipulator velocity (*dotted line*) and acceleration (*solid line*), also as functions of s.

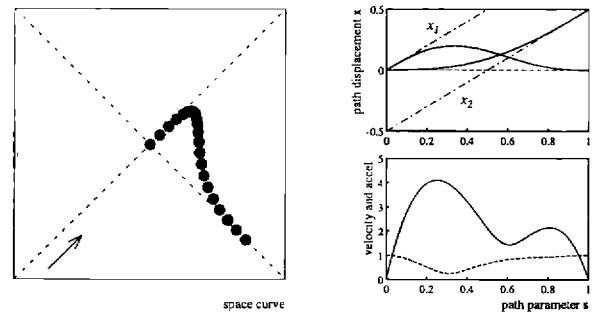
With $\pi_h = \pi_s = 0.5$, we have the conventional transition in Figure 7A. In Figure 7B, setting $\pi_h = 0.3125$ and $\pi_s = 0.6875$ causes the transition to travel through the path intersection point (the values depend on κ, which is 6 here). In Figure 7C, we let $\pi_h = 0$ and $\pi_s = 0.5$, which causes path $\mathbf{x}_1$ to be followed all the way to the end, where it then overshoots the intersection point as it begins the blend into path $\mathbf{x}_2$. The opposite case—undershooting to follow path $\mathbf{x}_2$ completely from the beginning—requires $\pi_h = 0.5$ and $\pi_s = 1$, as is shown in Figure 7D. By setting $\pi_h = 0$ and $\pi_s = 1$ (Figure 7E), it is possible to follow both paths exactly, at the expense of



(a) Transition profile with $\pi_h = 0.5$ and $\pi_s = 0.5$



(b) Transition profile with $\pi_h = 0.3125$ and $\pi_s = 0.6875$

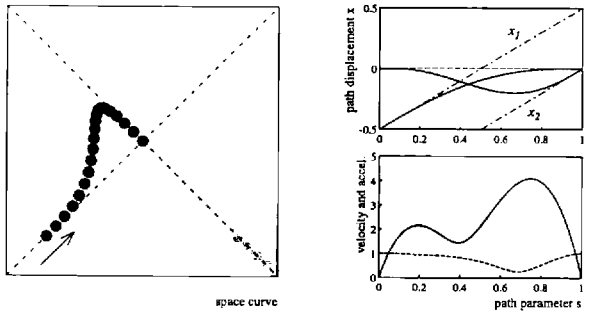


(c) Transition profile with $\pi_h = 0.0$ and $\pi_s = 0.5$

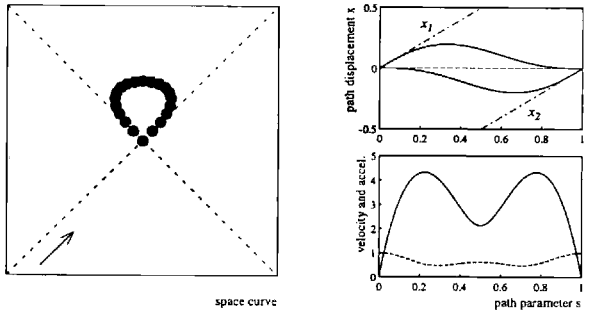
Fig. 7. Different transition profiles created with $\kappa = 6$ and different values of π_h and π_s for two straight motion segments intersecting at right angles. The left box shows the spatial shape of the transition, the top right box graphs the displacements of the halt and start components, and the bottom right box graphs the magnitudes of the velocity (dotted line) *and acceleration* (solid line).

overshooting and looping around. Notice that in this case, the position *and* the velocity specifications for both paths are followed precisely for their entire length, which can be useful in situations such as where a robot is required to deposit material at a constant rate. In the last figure, 7F, settings of $\pi_h = 0.6$ and $\pi_s = 0.4$ are used to produce a symmetrical transition for which the manipulator speed is close to constant.

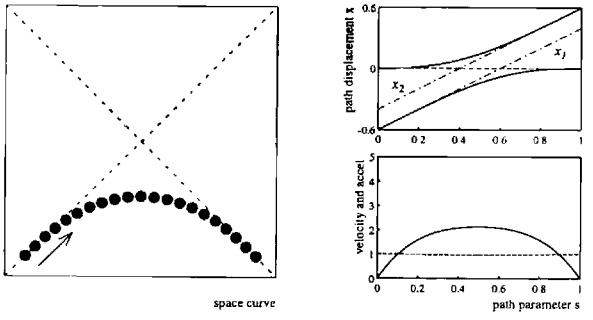
It is reasonable to use the following rules of thumb when setting preview values. If $\pi_h + \pi_s = 1$ and the initial and final path velocities are equal, then the space



(d) Transition profile with $\pi_h = 0.5$ and $\pi_s = 1.0$



(e) Transition profile with $\pi_h = 0.0$ and $\pi_s = 1.0$



(f) Transition profile with $\pi_h = 0.6$ and $\pi_s = 0.4$

Fig. 7. (continued)

curve will be symmetric about the intersection point. Generally, as $\pi_h \to 0$ and $\pi_s \to 1$, the speed during the transition will decrease. Specifically, for $\kappa = 6$, if $\pi_h > 1/2$ (or $\pi_s < 1/2$), an overshoot will occur in the initial (or final) path velocity.

Preview parameters can be implemented as follows. π_h is controlled by simply changing the time at which the transition begins. Setting π_h to 0 requires starting the transition earlier, while setting π_h to 1 requires starting it later. π_s is controlled by adding an offset to path $\mathbf{x}_2$ to change the time at which it intersects $\mathbf{p}$ (see Section 6.3).

5.3. *Adjusting the Transition Time Interval*

Adjusting the values of π_h and π_s can cause the transition acceleration to change, and the transition time calculation must account for this. To determine the effect, consider again the acceleration associated with two linear path segments. Setting $\mathbf{u} = -\kappa \mathbf{v}_d$ in (20) and letting M denote the integral square of the acceleration magnitude, we get

$$M = 2/35\,(150 - 15\kappa + \kappa^2)\|\mathbf{v}_d\|^2 + 120/7\,(\mathbf{v}_d \cdot \mathbf{b}_d + \|\mathbf{b}_d\|^2). \quad (25)$$

From the definition of π_h and π_s we have that $\mathbf{x}_1(\pi_h) = \mathbf{p}$ and $\mathbf{x}_2(\pi_s) = \mathbf{p}$, and, because the acceleration can be measured in any coordinate frame, we assume without loss of generality that $\mathbf{p} = 0$. Combining these relationships with (17) yields

$$\mathbf{b}_d = \pi_h \mathbf{v}_1 - \pi_s \mathbf{v}_2. \quad (26)$$

Substituting this into (25) gives an expression for M in terms of κ, π_h, π_s, and the initial path velocities:

$$M = 2/35\,(150 - 15\kappa + \kappa^2)\|\mathbf{v}_d\|^2 + 120/7\,(\pi_s \mathbf{v}_1 - \pi_h \mathbf{v}_2) \cdot [(\pi_s - 1)\mathbf{v}_1 - (\pi_h - 1)\mathbf{v}_2]. \quad (27)$$

Taking the square root of M (and dividing by the unity time interval for s) gives the root-mean-square acceleration, which we wish to compare to the reference acceleration a_r. However, where is τ? So far, all terms have been expressed with respect to the time coordinate s; τ appears when the necessary coordinate changes between t and s are applied. First, the input velocity terms in M must be converted from t to s, which requires multiplying each term by 2τ. Second, the output acceleration value must be converted from s back to t, which requires dividing by $4\tau^2$. The net expression can be solved for τ to yield

$$\tau = \frac{\sqrt{M}}{2a_r}. \quad (28)$$

As a side note, it can be shown that M has a minimum value with respect to π_h and π_s when $\pi_s = \pi_h = 1/2$, which are the canonical values.

6. An Example Trajectory Algorithm

To illustrate how the ideas in this article are used in practice, we present a simple trajectory generation algorithm.

For the sake of brevity, a number of simplifications are assumed. The time taken to travel along a given path segment is always determined from a desired velocity and cannot be explicitly specified; it is presumed that a future motion target is always available; there is no capacity to interrupt motions; and it is not possible to begin a new motion while a transition is in progress. A more practical trajectory generator would need to handle these cases. The algorithm also ignores details associated with the discrete time nature of the computation.

6.1. Motion Interpolation

The example trajectory generator constructs paths by linearly interpolating between target positions. Each such interpolation constitutes one *motion segment*. Let the current time be t, the time the current motion began be t_a, and the target position the manipulator is heading toward be $\mathbf{x}_b$. It should be noted that $\mathbf{x}_b$ may be changing in time *independently* of the trajectory generator, with a velocity described by $\mathbf{v}_b$. Let the difference between $\mathbf{x}_b$ and the previous target position (evaluated at time t_a) be $\mathbf{d}_{ba}$, and let the total time required for the motion segment be σ_b. Then the desired manipulator position $\mathbf{x}$ along the motion segment at each time t is given by

$$\mathbf{x} = \mathbf{x}_b - [1 - (t - t_a)/\sigma_b]\,\mathbf{d}_{ba}. \tag{29}$$

$\mathbf{d}_{ba}$ is the "drive" vector for the motion segment. Notice that the manipulator will directly track any variations in $\mathbf{x}_b$.

6.2. Estimating the Transition Time

At any given time, the trajectory generator is in one of two states: *cruise*, when (29) is used to move toward a target position, or *transition*, when the motion toward $\mathbf{x}_b$ is replaced with another motion toward the *next* target point $\mathbf{x}_c$.

When the trajectory generator is in the *cruise* state, it constantly estimates the τ required for the transition to the next motion. This is done using the current manipulator velocity $\mathbf{v}_1$ and an estimate of the posttransition velocity $\mathbf{v}_2$. $\mathbf{v}_1$ is the sum of the current target velocity $\mathbf{v}_b$ and the velocity with which the target is being approached:

$$\mathbf{v}_1 = \mathbf{v}_b + \mathbf{d}_{ba}/\sigma_b.$$

$\mathbf{v}_2$ is formed by adding the velocity of the next target ($\mathbf{v}_c$) to a vector in the direction of the next motion whose magnitude equals the desired travel speed v_r:

$$\mathbf{v}_2 = \mathbf{v}_c + v_r(\mathbf{x}_c - \mathbf{x})/\|\mathbf{x}_c - \mathbf{x}\|.$$

Equations (25), (26), and (28) are then used by the function *estimateTau* to produce a value for τ. As explained earlier, the reference acceleration a_r is presumed to be provided, possibly being updated by a separate algorithm that accounts for the robot dynamics.

If the actual length of the next path is very short, the manipulator may not have time to accelerate to the desired travel speed v_r, and the estimate for τ will be too large. To correct for this, $a_r\tau^2$ is compared to the estimated path length, and, if it is greater, τ is reestimated with a lower prescribed speed of $\sqrt{\|\mathbf{x}_c - \mathbf{x}\|/a_r}$.

6.3. Computing the Next Motion Parameters

Using the estimate for τ, the trajectory generator calculates the time t_0 at which the transition to the next motion should begin. The anticipated time of arrival at $\mathbf{x}_b$ is $t_h \equiv t_a + \sigma_b$, and t_0 precedes t_h by an amount specified by the preview parameter π_h:

$$t_0 = t_a + \sigma_b - 2\tau\pi_h.$$

When $t \geq t_0$, the transition to the next motion is started, and the new motion parameters are initialized. These parameters include t_s, $\mathbf{d}_{cb}$, and σ_c, which correspond to t_a, $\mathbf{d}_{ba}$, and σ_b, so that the new path towards $\mathbf{x}_c$ is computed from

$$\mathbf{x} = \mathbf{x}_c - [1 - (t - t_s)/\sigma_c]\,\mathbf{d}_{cb}. \tag{30}$$

t_s and $\mathbf{d}_{cb}$ must be determined so as to properly respect the preview parameters. In this case, the "target point" $\mathbf{p}$ of equation (24) is simply the value of the target position $\mathbf{x}_b$ at the time t_h.The new path intersects this target point at time t_s, which is offset from the transition start time by the preview parameter π_s:

$$t_s = t_0 + 2\tau\pi_s. \tag{31}$$

$\mathbf{d}_{cb}$ is computed so that the new path intersects the target point at $t = t_s$. Setting $t = t_s$ in (30), and remembering that the target point equals $\mathbf{x}_b$ at time t_h, we get

$$\mathbf{x}_b|_{t=t_h} = \mathbf{x}_c|_{t=t_s} - \mathbf{d}_{cb}.$$

Evaluating $\mathbf{d}_{cb}$ requires that $\mathbf{x}_b$ and $\mathbf{x}_c$ be determined at the future times t_h and t_s. This can be done by extrapolation, using their current velocities and positions. Because $t_h = t_0 + 2\tau\pi_h$, $t_s = t_0 + 2\tau\pi_s$, and the time of evaluation is t_0, $\mathbf{d}_{cb}$ can be computed with the formula

$$\mathbf{d}_{cb} = (\mathbf{x}_c - \mathbf{x}_b) + 2\tau(\pi_s\mathbf{v}_c - \pi_h\mathbf{v}_b).$$

The travel time for the next motion segment, σ_c, is computed by dividing the magnitude of the drive vector by the desired path velocity: $\sigma_c = \|\mathbf{d}_{cb}\|/v_r$. The difference in velocities, $\mathbf{v}_d$, between the old and new paths, which will be used by the β term during the transition blend, is reestimated by

$$\mathbf{v}_d = \mathbf{v}_c + \mathbf{d}_{cb}/\sigma_c - \mathbf{v}_1.$$

During the transition, the parameter s is computed from $s = (t - t_0)/2\tau$. Both the current path $\mathbf{x}_1$ and the next path $\mathbf{x}_2$ are computed using equations (29) and (30), and the results are blended together using the functions $\alpha(s)$ and $\beta(s)$.

```
estimateTau (v_r)                          // estimate τ for desired velocity v_r.
 { v_2 = v_c + v_r(x_c − x)/||x_c − x||;    // estimate post transition velocity.
   v_d = v_2 − v_1;                         // estimate differences in velocity
   b_d = π_h v_1 − π_s v_2;                 //    and position.
   M = 2/35 (150 − 15κ + κ^2)||v_d||^2+
         120/7 (v_d · b_d + ||b_d||^2);     // compute integral square of acceleration
   return (√M/2a_r);                        //    and use this to estimate τ.
 }

eachCycle ()                               // called once every trajectory cycle.
 { if (state == cruise)
    { v_1 = v_b + d_ba/σ_b.                 // estimate current velocity.
      τ = estimateTau (v_r);                // estimate transition time.
      if (a_r τ^2 > ||x_c − x||)
       { τ = estimateTau (√(||x_c − x||/a_r));   // correct for small distances.
       }
      t_0 = t_a + σ_b − 2τπ_h;              // compute next transition start time
      if (t ≥ t_0)                          //    and then start transition.
       { t_s = t_0 + 2τπ_s;                 // compute start time for next motion,
         d_cb = (x_c − x_b) + 2τ(π_s v_c − π_h v_b);   //    drive vector,
         σ_c = ||d_cb||/v_r;                //    and total motion time.
         v_d = v_c + d_cb/σ_c − v_1;        // compute velocity difference for blending.
         state = transition;
       }
    }
   else
    { if (s ≥ 1)                            // then transition is over.
       { x_b ← x_c;                         // x_c becomes next motion target.
         d_ba = d_cb, σ_b = σ_c, t_a = t_s; // shift drive and time parameters for next motion.
         x_c ← next target;                 // get next specified target
         set new v_r, π_h, π_s, κ values;   //    and control variables.
         state = cruise;
       }
    }
   if (state == cruise)
    { x = x_b − [1 − (t − t_a)/σ_b] d_ba;   // normal interpolation to target.
    }
   else
    { s = (t − t_0)/2τ;                     // compute transition parameter.
      x_1 = x_b − [1 − (t − t_a)/σ_b] d_ba; // compute motion along path 1.
      x_2 = x_c − [1 − (t − t_s)/σ_c] d_cb; // compute motion along path 2.
      x = x_1 + α(s)(x_2 − x_1) − κβ(s)v_d; // blend motions together.
    }
 }
```

Fig. 8. Algorithm for computing trajectories.

The transition ends when $s \geq 1$. At this point, the next target $\mathbf{x}_c$ becomes the current target $\mathbf{x}_b$, and the variables $\mathbf{d}_{ba}$, σ_b, and t_a assume the values of $\mathbf{d}_{cb}$, σ_c, and t_s. $\mathbf{x}_c$ is set to the next future target; new values for the control variables v_r, π_h, π_s, and κ are loaded; and the state of the trajectory generator is set to *cruise*.

The complete algorithm is shown in Figure 8.

7. Handling Rotations

When applied to Cartesian coordinates, the blend paradigm must handle 3-D rotational paths. Because rotational operations do not commute, rotations cannot be described as a vector quantity, and therefore the methods described so far cannot be applied directly. However, it is possible to achieve the same effect with rotational operators.

The basic idea utilizes the decomposition into *halt* and *start* components described in Section 5. The vector equation (24) can be rewritten as

$$\mathbf{x}(s) = \mathbf{x}_1(s) + \underbrace{\alpha(\mathbf{p} - \mathbf{x}_1(s)) + \kappa\beta\dot{\mathbf{x}}_1(0)}_{A} + \underbrace{\alpha(\mathbf{x}_2(s) - \mathbf{p}) - \kappa\beta\dot{\mathbf{x}}_2(0)}_{B}. \quad (32)$$

Because $\mathbf{p}$ intersects paths $\mathbf{x}_1$ and $\mathbf{x}_2$, then, to a first approximation, $\dot{\mathbf{x}}_1$ is parallel to $\mathbf{p} - \mathbf{x}_1$ and $\dot{\mathbf{x}}_2$ is parallel to $\mathbf{x}_2 - \mathbf{p}$. If parts A and B are now thought of as representing rotations instead of vectors, this implies that within each part, the axes of angular velocity and rotation are parallel. Within A and B, the blend functions may then be applied using scalar operations on a rotation about a single axis.

Some definitions are now needed. Let $\mathbf{Rot}(\theta, \mathbf{v})$ be a function that accepts an angle θ and a vector $\mathbf{v}$ describing an axis of rotation and returns the corresponding 3×3 rotation matrix.[2] Let $\Theta(\mathbf{R})$ and $\mathbf{vec}(\mathbf{R})$ be the inverse functions that return the angle and direction vector corresponding to a given rotation matrix $\mathbf{R}$. Assume that the two rotational paths we wish to blend are given by matrices $\mathbf{R}_1(s)$ and $\mathbf{R}_2(s)$ and that they intersect at a fixed orientation $\mathbf{R}_p$. Assume also that the initial angular velocity of the two paths is given by Ω_1 and Ω_2.

We now need to blend paths $\mathbf{R}_1$ and $\mathbf{R}_2$ together in a manner analogous to (32). Define the difference between $\mathbf{R}_1(s)$ and $\mathbf{R}_p$ by $\mathbf{R}_{1p}(s) = \mathbf{R}_1(s)^{-1}\mathbf{R}_p$, and the difference between $\mathbf{R}_p$ and $\mathbf{R}_2(s)$ by $\mathbf{R}_{p2}(s) = \mathbf{R}_p^{-1}\mathbf{R}_2(s)$. If $\mathbf{R}_1(s)$ and $\mathbf{R}_2(s)$ are linear (i.e., they have constant angular velocities), then $\mathbf{vec}(\mathbf{R}_{1p})$ is parallel to Ω_1 and $\mathbf{vec}(\mathbf{R}_{p2})$ is parallel to Ω_2. The blended path can now be computed as

$$\mathbf{R}(s) = \mathbf{R}_1(s)\,\mathbf{R}_A(s)\,\mathbf{R}_B(s), \quad (33)$$

where

$$\mathbf{R}_A(s) = \mathbf{Rot}\big[\alpha\,\Theta(\mathbf{R}_{1p}(s)) + \kappa\beta\|\Omega_1\|,\ \mathbf{vec}\,(\mathbf{R}_{1p}(s))\big],$$
$$\mathbf{R}_B(s) = \mathbf{Rot}\big[\alpha\,\Theta(\mathbf{R}_{p2}(s)) + \kappa\beta\|\Omega_2\|,\ \mathbf{vec}\,(\mathbf{R}_{p2}(s))\big].$$

In the nonlinear case, the parallel axis assumptions are still valid to a first approximation, and all the necessary boundary conditions of (4) still hold. In the linear case, (33) turns out to be equivalent to the method for rotational transitions described in Taylor (1979).

Rotational operators can be used to similarly generalize the other computations used in the algorithm of Section 6.

8. Conclusion

We have presented a method for computing the transition between manipulator trajectory path segments that is tolerant of time variations in those segments. This is particularly important in cases where the paths are being updated by real-time sensory information.

In essence, the method decomposes the calculation into two parts. The first part uses a smooth convex blend function to combine the paths segments and ensure that all the necessary boundary conditions are satisfied, without requiring any knowledge of the future. The second part uses an additional smooth function, combined with the predictive information provided by the initial path velocities, to minimize transition acceleration. The blend functions described here provide continuity up to the second derivative. If necessary, higher-order blend functions could be used to provide continuity for higher derivatives.

If the blended paths are themselves accelerating, and this acceleration is known to be bounded by A, then the additional acceleration this introduces into the transition is bounded by $(19/4)A$.

Although the blend paradigm is mainly designed for dealing with path uncertainties, it also provides a natural way to join motion segments whose paths are computed in different coordinate systems. For instance, it is convenient to "blend" together a motion computed in joint coordinates with one computed in Cartesian coordinates.

We have also found that if we decompose the trajectory blend into "halt" and "start" components, then it is possible to define two preview parameters, π_h and π_s, that can be set to various values in the range [0, 1] to control the spatial shape of the transition curve.

Appendix: Proof of the Bound on Blended Acceleration

The proof of Theorem 1 requires the following lemmas:

LEMMA 1. Let $\mathbf{y}(t)$ be a path in space, let $\dot{\mathbf{y}}(t)$ be its velocity, and define $\mathbf{v}_0 \equiv \dot{\mathbf{y}}(t_0)$. If $\|\ddot{\mathbf{y}}(t)\| \leq B$, then

$$\|\dot{\mathbf{y}}(t) - \mathbf{v}_0\| \leq B|t - t_0|.$$

Proof. From basic kinematics, we know that

$$\dot{\mathbf{y}}(t) = \mathbf{v}_0 + \int_{t_0}^{t} \ddot{\mathbf{y}}(\tau)\, d\tau.$$

Rearranging, taking norms, and applying the triangle inequality under the integral yields

$$\|\dot{\mathbf{y}}(t) - \mathbf{v}_0\| = \|\int_{t_0}^{t} \ddot{\mathbf{y}}(\tau)\, d\tau\| \leq B\int_{t_0}^{t} |d\tau|,$$

from which the result follows. □

LEMMA 2. Let $y(t)$ be a path in one dimension, with $y(0) = \dot{y}(0) = 0$, and define its velocity at some time

2. Any three-dimensional rotation can be expressed as a single rotation θ about some axis $\hat{\mathbf{v}}$.

$\tau \geq 0$ by $v_f \equiv \dot{y}(\tau)$. Then, if $|\ddot{y}(t)| \leq B$ and $v_f \geq 0$, $y(\tau)$ is bounded by

$$\begin{aligned} 1/2 v_f \tau - 1/4(B\tau^2 - v_f^2/B) &\leq y(\tau) \\ &\leq 1/2 v_f \tau \\ &\quad + 1/4(B\tau^2 - v_f^2/B). \end{aligned} \tag{34}$$

Proof. We will prove the upper bound first. Because $\dot{y}(0) = 0$, we know from Lemma 1 that

$$\dot{y}(t) \leq Bt \quad \text{for} \quad t \geq 0. \tag{35}$$

Applying the same lemma with t_0 set to τ yields the additional constraint

$$|\dot{y}(t) - v_f| \leq B|t - \tau|,$$

which for $t \leq \tau$ implies that

$$\dot{y}(t) \leq B(\tau - t) + v_f. \tag{36}$$

Constraints (35) and (36) are equal at the point $t = \sigma \equiv 1/2(\tau + v_f/B)$. For $t < \sigma$, (35) dominates, and for $t > \sigma$, (36) dominates. $y(\tau)$ can be determined from the piecewise integral

$$y(\tau) = \int_0^\sigma \dot{y}(t)\, dt + \int_\sigma^\tau \dot{y}(t)\, dt.$$

Applying the appropriate constraints under the integral sign gives

$$y(\tau) \leq \int_0^\sigma Bt\, dt + \int_\sigma^\tau [B(\tau - t) + v_f]\, dt,$$

and integrating and expanding the value of σ yields the upper bound.

To prove the lower bound, consider the change of coordinates defined by

$$x(t) \equiv y(\tau - t) - y(\tau) + v_f t.$$

The first and second derivatives of $x(t)$ are

$$\begin{aligned} \dot{x}(t) &= -\dot{y}(\tau - t) + v_f \\ \ddot{x}(t) &= \ddot{y}(\tau - t). \end{aligned}$$

From this we can verify that $x(0) = 0$, $\dot{x}(0) = 0$, $\dot{x}(\tau) = v_f$, and $|\ddot{x}(t)| \leq B$. This in turn implies that the upper bound in (34) holds for x. Then, for, $t = \tau$, we have

$$x(\tau) = -y(\tau) + v_f \tau \leq 1/2 v_f \tau + 1/4(B\tau^2 - v_f^2/B),$$

which can be solved for $y(\tau)$ to yield the lower bound. □

Proof of Theorem 1: First, scale the time coordinate to use s in place of t, so that $\mathbf{y}(s) = \mathbf{y}_1(s) + \alpha(s)(\mathbf{y}_2(s) - \mathbf{y}_1(s))$. Because $s = t/T$, the acceleration bounds are modified to $\|\ddot{\mathbf{y}}_1(s)\|, \|\ddot{\mathbf{y}}_2(s)\| \leq AT^2$. Next, define $\mathbf{x}(s) \equiv \mathbf{y}_d(s) = \mathbf{y}_2(s) - \mathbf{y}_1(s)$ and take the second derivative of $\mathbf{y}(s)$ to get

$$\ddot{\mathbf{y}}(s) = \ddot{\mathbf{y}}_1(s) + \alpha(s)\,\ddot{\mathbf{x}}(s) + 2\dot{\alpha}(s)\,\dot{\mathbf{x}}(s) + \ddot{\alpha}(s)\,\mathbf{x}(s), \tag{37}$$

where

$$\begin{aligned} \dot{\alpha}(s) &= 30(s^4 - 2s^3 + s^2), \\ \ddot{\alpha}(s) &= 60(2s^3 - 3s^2 + s). \end{aligned}$$

Note that $\|\ddot{\mathbf{x}}\| \leq 2AT^2$. For convenience, define $A_x \equiv 2AT^2$, so that $\|\ddot{\mathbf{y}}_1(s)\|, \|\ddot{\mathbf{y}}_2(s)\| \leq 1/2 A_x$.

For notational simplicity, the explicit dependency of variables on s will be omitted from most of the remaining discussion.

Applying norms to (37) yields

$$\|\ddot{\mathbf{y}}\| \leq \|\ddot{\mathbf{y}}_1 + \alpha\ddot{\mathbf{x}}\| + \|2\dot{\alpha}\dot{\mathbf{x}} + \ddot{\alpha}\mathbf{x}\|. \tag{38}$$

The first term on the right side is bounded by

$$\begin{aligned} \|\ddot{\mathbf{y}}_1 + \alpha\ddot{\mathbf{x}}\| &= \|(1 - \alpha)\ddot{\mathbf{y}}_1 + \alpha\ddot{\mathbf{y}}_2\| \\ &\leq (1 - \alpha)1/2\, A_x + \alpha\, 1/2\, A_x \\ &\leq A_x/2, \end{aligned}$$

as α and $1 - \alpha$ are both ≥ 0 for $s \in [0, 1]$.

We next need to establish a bound for $\|2\dot{\alpha}\dot{\mathbf{x}} + \ddot{\alpha}\mathbf{x}\|$. Denote this term by F, so that

$$\|\ddot{\mathbf{y}}\| \leq A_x/2 + F. \tag{39}$$

F will be bounded by bounding F^2. This will be done by formulating bounds that apply for any particular value of s, and then finding the maximum of these bounds for all $s \in [0, 1]$.

If, for some value of s, $\mathbf{x}$ and $\dot{\mathbf{x}}$ are both 0, then F is also 0, and so we do not need to consider this case further. Otherwise, if $\dot{\mathbf{x}}$ is 0 and $\mathbf{x}$ is not 0, then let $\hat{\mathbf{u}}$ be a fixed unit vector in the direction of $\mathbf{x}$ at that particular value of s. Let x be the component of $\mathbf{x}$ along this vector. We then have $x = \|\mathbf{x}\|$, $\dot{x} = 0$ (because $\dot{\mathbf{x}}$ is 0), and $\ddot{x} \leq A_x$ (because $\|\ddot{\mathbf{x}}\| \leq A_x$). Hence, for any value of $s \geq 0$, x is constrained by the upper bound of Lemma 2 with v_f set to 0, so that

$$x \leq 1/4 A_x s^2,$$

which implies

$$F^2 = \|2\dot{\alpha}\dot{\mathbf{x}} + \ddot{\alpha}\mathbf{x}\|^2 = \|\ddot{\alpha}\mathbf{x}\|^2 \leq (1/4\,\ddot{\alpha}\, A_x s^2)^2. \tag{40}$$

Now assume that $\mathbf{x}$ and $\dot{\mathbf{x}}$ are both nonzero. Let $\hat{\mathbf{u}}$ be a fixed unit vector in the direction of $\dot{\mathbf{x}}$ for a particular value of s. Let $\mathbf{x}_v$ be the projection of $\mathbf{x}$ onto $\hat{\mathbf{u}}$, and define $\mathbf{x}_\perp \equiv \mathbf{x} - \mathbf{x}_v$. From these definitions, F^2 can be expanded to

$$F^2 = \|2\dot{\alpha}\dot{\mathbf{x}} + \ddot{\alpha}\mathbf{x}_v\|^2 + \|\ddot{\alpha}\mathbf{x}_\perp\|^2. \tag{41}$$

Note that $\|\ddot{\mathbf{x}}_v\|, \|\ddot{\mathbf{x}}_\perp\| \leq \|\ddot{\mathbf{x}}\| \leq A_x$. Consider the $\mathbf{x}_\perp$ term first: because by construction there is no velocity component along $\mathbf{x}_\perp$, its magnitude is limited by the same bound in (40), and so

$$\|\ddot{\alpha}\mathbf{x}_\perp\|^2 \leq (1/4\,\ddot{\alpha}\,A_x s^2)^2. \tag{42}$$

Now consider the other term. By construction, both $\dot{\mathbf{x}}$ and $\mathbf{x}_v$ are parallel to $\hat{\mathbf{u}}$. Letting v and x_v denote the components of these vectors along $\hat{\mathbf{u}}$ gives

$$\|2\dot{\alpha}\dot{\mathbf{x}} + \ddot{\alpha}\mathbf{x}_v\|^2 = (2\dot{\alpha}v + \ddot{\alpha}x_v)^2.$$

By construction $v \geq 0$, and $\ddot{x} = \|\ddot{\mathbf{x}}_v\| \leq A_x$, and therefore the bounds of Lemma 2 hold for x_v, for any value of $s \geq 0$. Denote these bounds by x_{vl} and x_{vu}, so that

$$\begin{aligned} x_{vl} &\equiv 1/2\,vs - 1/4(A_x s^2 - v^2/A_x) \\ &\leq x_v \leq 1/2\,vs + 1/4(A_x s^2 - v^2/A_x) \equiv x_{vu}. \end{aligned} \tag{43}$$

Now, because $x_{vl} \leq x_v \leq x_{vu}$, it can be deduced that

$$|2\dot{\alpha}v + \ddot{\alpha}x_v| \leq \max(|2\dot{\alpha}v + \ddot{\alpha}x_{vl}|, |2\dot{\alpha}v + \ddot{\alpha}x_{vu}|).$$

Combining this with (41) and (42) gives

$$\begin{aligned} F^2 \leq \max[&(2\dot{\alpha}v + \ddot{\alpha}x_{vl})^2 + (1/4\,\ddot{\alpha}A_x s^2)^2, \\ &(2\dot{\alpha}v + \ddot{\alpha}x_{vu})^2 + (1/4\,\ddot{\alpha}A_x s^2)^2]. \end{aligned} \tag{44}$$

Note that this bound also subsumes the bound in (40) for the case where $\dot{\mathbf{x}} = 0$, so we can deal exclusively with (44).

Now from Lemma 1, and the fact that $v \geq 0$, we know that $v = lA_x s$ for some $l \in [0, 1]$. Substituting this into the expressions for v, x_{vl}, and x_{vu} in (44) gives

$$F(s)^2 \leq A_x^2 \max[L(s,l),\ U(s,l)], \tag{45}$$

where

$$\begin{aligned} L(s,l) &\equiv (2\dot{\alpha}ls + 1/2\,\ddot{\alpha}s^2[l - 1/2(1 - l^2)])^2 + (1/4\,\ddot{\alpha}s^2)^2, \\ U(s,l) &\equiv (2\dot{\alpha}ls + 1/2\,\ddot{\alpha}s^2[l + 1/2(1 - l^2)])^2 + (1/4\,\ddot{\alpha}s^2)^2. \end{aligned}$$

U and L are both polynomials in s and l, and we now wish to find their maximum values in the region $s, l \in [0, 1]$.[3]

Starting with $U(s,l)$, we first examine the global critical points where $\partial U/\partial l = \partial U/\partial s = 0$. There are 17 of these, but only one that is interior to the region $s, l \in [0, 1]$, for which the value of U is ≈ 2.74. Next, we examine critical points on the region boundaries. Taking $s = 0$ or $s = 1$ gives $U(0,l) = U(1,l) = 0$, so this can be ignored. Taking $l = 0$ or $l = 1$ yields $U(s,0)$ or $U(s,1)$, which are both polynomials in s. The maximum of both of these occurs for $U(s,1)$ at $s = 1/2$ and has a value of $225/64$.

For $L(s,l)$, there are also 17 global critical points, for which three satisfy $s, l \in (0, 1)$, and the largest of these values is ≈ 0.98. The equations of the region boundaries are the same as those for $U(s,l)$, and so these do not need to be considered again.

The largest value is hence $225/64$. Substituting this back into (45), taking the square root of both sides and combining with (39) yields

$$\|\ddot{\mathbf{y}}(s)\| \leq (19/8)A_x. \tag{46}$$

Finally, transforming s back to t gives

$$\|\ddot{\mathbf{y}}(t)\| = \|\ddot{\mathbf{y}}(s)/T^2\| \leq (19/4)A,$$

which establishes the bound.

To demonstrate that the bound is tight, consider a one-dimensional case where the two paths are accelerating away from each other as fast as possible. Recalling that $A_x = 2AT^2$, define $\mathbf{y}_1(s) = y_1(s) \equiv -1/4A_x s^2$ and $\mathbf{y}_2(s) = y_2(s) \equiv 1/4A_x s^2$. $\mathbf{x}(s)$ then becomes $x(s) = y_2(s) - y_1(s) = 1/2A_x s^2$, and $\dot{\mathbf{x}}(s)$ becomes $\dot{x}(s) = A_x s$. Substituting these values for x into (37) yields

$$\ddot{y}(s) = \ddot{y}_1(s) + \alpha(s)\ddot{x}(s) + 2\dot{\alpha}(s)A_x s + 1/2\ddot{\alpha}(s)A_x s^2.$$

At $s = 1/2$, where $\alpha(1/2) = 1/2$, $\dot{\alpha}(1/2) = 15/8$, and $\ddot{\alpha}(1/2) = 0$, we have

$$\ddot{y}(1/2) = \ddot{y}_1(1/2) + 1/2\ddot{x}(1/2) + 15/8A_x.$$

Now, if at $s = 1/2$ the acceleration of y_1 is suddenly reversed so that $\ddot{y}_1(1/2) = 1/2A_x$, we get

$$\ddot{y}(1/2) = (19/8)A_x,$$

which is equal to the bound (46). □

Acknowledgments

The research described in this article was supported by a grant from the National Science and Engineering Council of Canada, and by the Institute for Robotics and Intelligent Systems (Center of Excellence of Canada) as part of the project "Simulation, Control and Planning in Robotics."

References

Andersson, R. L. 1988. Aggressive trajectory generator for a robot ping-pong player. *IEEE Conference on Robotics and Automation*, Philadelphia, pp. 188–193.

3. This was done with the help of *Mathematica*, a symbolic algebra program distributed by Wolfram Research, Inc., 100 Trade Center Drive, Champaign, Illinois 61820–7237, USA.

Bobrow, J. E., Dubowsky, S., and Gibson, J. S. 1985. Time optimal control of robotic manipulators along a specified path. *Int. J. Robot. Res.* 4(3):3–17.

Foley, J. D., and Van Dam, A. 1984. *Fundamentals of Interactive Computer Graphics*. Reading, MA: Addison-Wesley.

Hayati, S., and Venkataraman, S. T. 1989. Design and implementation of a robot control system with traded and shared control capability. *IEEE Conference on Robotics and Automation*, Scottsdale, Arizona, pp. 1310–1315.

Hayward, V., Daneshmend, L., and Hayati, S. 1989 (Columbus, OH). An overview of Kali: A system to program and control cooperative manipulators. *Fourth International Conference on Advanced Robotics*, New York: Springer-Verlag, pp. 547–558.

Hirzinger, G., and Dietrich, J. 1986. Multisensory robots and sensor based path generation. *IEEE Conference on Robotics and Automation*, San Francisco, pp. 1992–2001.

Hollerbach, J. M. 1984. Dynamic scaling of manipulator trajectories. *ASME J. Dyn. Sys. Measurement Control* 106:102–106.

Lin, C. S., Chang, P. R., and Luh, J. Y. S. 1983. Formulation and optimization of cubic polynomial joint trajectories for industrial robots. *IEEE Trans. Auto. Control* AC-28(12):1066–1073.

Lloyd, J., Parker, M., and McClain, R. 1988. Extending the RCCL programming environment to multiple robots and processors. *IEEE Conference on Robotics and Automation*, Philadelphia, pp. 465–469.

Paul, R. P. 1981. *Robot Manipulators: Mathematics, Programming, and Control*. Cambridge, MA: MIT Press.

Shin, K. G., and McKay, N. D. 1985. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Trans. Auto. Control* AC-30(6):531–541.

Shin, K. G., and McKay, N. D. 1986. A dynamic programming approach to trajectory planning of robotic manipulators. *IEEE Trans. Auto. Control* AC-31(6):491–500.

Taylor, R. H. 1979. Planning and execution of straight line manipulator trajectories. *IBM J. Res. Dev.* 23:253–264.

Thompson, S. E., and Patel, R. V. 1987. Formulation of joint trajectories for industrial robots using B-splines. *IEEE Trans. Indust. Electron.* IE-34(2):192–199.