Proceedings of the 1991 IEEE
International Conference on Robotics and Automation
Sacramento, California - April 1991

Real-Time Trajectory Generation Using Blend Functions

John Lloyd and Vincent Hayward
McGill Research Center for Intelligent Machines
3480 University Street
Montréal, Québec H3A 2A7

Abstract

A technique for transitioning between path segments is described which is tolerant to dynamic changes arising from sensor inputs. The main idea is to "blend" the segments together in a way that does not require advance knowledge of the paths. It is also possible to decompose the transition into an action which "brings to rest" the motion along the initial path plus an action which "starts up" the motion along the final path. By adjusting the timing of these two components, one may control the shape of the transition, in both time and space, so as to satisfy different task constraints.

1 Introduction

In robot manipulator control systems, the role of the trajectory generator is to convert commands specified by a *programming* level into a stream of setpoints suitable for tracking by a feedback controller. A typical programming command will specify 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 setpoint paths that meet these constraints as closely as possible. A central problem in trajectory generation is that 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 velocity and possibly higher derivatives.

Current approaches to trajectory generation can be grouped first into offline and online techniques. If the trajectory is computed offline (i.e., before the setpoints are sent to the feedback controller), 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. [2, 9]. Once computed, however, such trajectories are generally difficult to modify in response to real-time sensor information. This problem may be tempered somewhat by using techniques, such as B-splines, that localize the trajectory's dependence on the constraints [11].

On-line trajectory generation, by contrast, computes the setpoints 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 constraints. The recent rapid advances in CPU power have improved the ability to perform online trajectory generation.

A common technique in on-line trajectory generation is to compute path segments that satisfy individual program requests, and then join these together using polynomial fits applied over transition windows [8, 10]. Often the path segments themselves are simply straight lines between via points. This separates the problem of meeting both program and manipulator constraints: during the transition window, the program path constraints are relaxed and the acceleration constraint predominates. Between transition windows, where the path segments are known to accelerate very little or not at all, the program path and velocity constraints predominate. The amount of manipulator acceleration induced by the transition is proportional to the inverse square of the length of the transition window [6], and so can be easily controlled. If one is willing to relax program velocity constraints (such as the ability to move at a constant speed), then the cruise period between transition windows may be disposed of entirely and the path may be constantly computed (and recomputed) as a polynomial fit to the next goal position [1].

What we will describe in this paper is a variation on the conventional transition window technique that

- ◇ permits the path segments to be changing dynamically in response to sensor inputs, and
- ◇ provides several parameter "knobs" that allow us to adjust the shape of the transition in both space and time.

The techniques described here can be applied equally well in either joint or Cartesian coordinates (or yet an-

other coordinate system). To simplify the presentation, most of the discussion will refer to paths of a single variable (x) vs. time. All of the methods generalize to paths described by vectors ($\mathbf{x}$). The generalization to rotations, which is necessary when working in Cartesian coordinates, is slightly more subtle; it has been worked out, but is not described here because of space limitations. Two variations of the scheme outlined in this paper have been implemented and currently form the core of the trajectory generation techniques of the Multi-RCCL system [7], and Kali [4, 5].

2 Transitioning by Polynomial Fitting

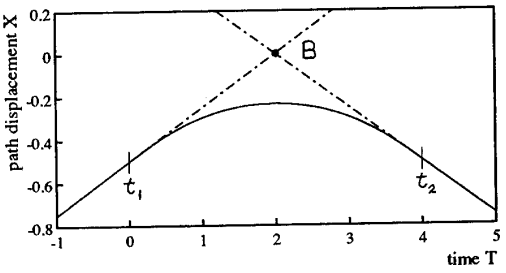

Figure 1: Illustration of a path segment transition, in one dimension. The path segments are indicated by hatched lines and the smoothed path is indicated by a solid line.

The transition window used to connect two path segments is illustrated by a one-dimensional example in Figure 1. Paths $X_1(t)$ and $X_2(t)$ both meet at the common point B. The transition window begins at time t_1 and ends at time t_2, and over this interval X_1 and X_2 should be smoothly connected by some curve $X(t)$. For simplicity, transition computations are usually done using a normalized time coordinate $s \in [0, 1]$, where

$$s = \frac{t - t_1}{t_2 - t_1}$$

Let $x(s)$, $x_1(s)$ and $x_2(s)$ correspond to $X(t)$, $X_1(t)$ and $X_2(t)$. Assuming that we want continuity down to the second derivative, then we have the following boundary conditions:

$$\begin{aligned} x(0) = x_1(0) \equiv p_1, \quad & x(1) = x_2(1) \equiv p_2 \\ \dot{x}(0) = \dot{x}_1(0) \equiv v_1, \quad & \dot{x}(1) = \dot{x}_2(1) \equiv v_2 \\ \ddot{x}(0) = \ddot{x}_1(0) \equiv a_1, \quad & \ddot{x}(1) = \ddot{x}_2(1) \equiv a_2 \end{aligned} \tag{1}$$

These can be satisfied using a 5th 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 readily found using a Hermite boundary condition matrix $\mathbf{H}$ [3].

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

Letting the boundary conditions be described by a vector $\mathbf{b} = (p_1, v_1, a_1, p_2, v_2, a_2)$, the coefficients may be determined from

$$\mathbf{c} = \mathbf{H}^{-1}\,\mathbf{b} \tag{3}$$

which yields

$$\begin{aligned} c_5 &= (a_2 - a_1)/2 + 6(p_2 - p_1) - 3(v_2 + v_1) \\ c_4 &= (3a_1 - 2a_2)/2 + 15(p_1 - p_2) + 8v_1 + 7v_2 \\ c_3 &= (a_2 - 2a_1)/2 + 10(p_2 - p_1) - 6v_1 - 4v_2 \\ c_2 &= a_1/2 \\ c_1 &= v_1 \\ c_0 &= p_1 \end{aligned} \tag{4}$$

This allows us to connect path 1 to path 2 smoothly. If the paths are linear (i.e., $a_1 = a_2 = 0$), and the paths intersect at the transition midpoint, then the boundary conditions can be matched using only a 4th degree polynomial [8].

Most windowed transition schemes utilize some form of the above relationships. Notice that in order to compute $x(s)$, we must have advance knowledge of the final boundary conditions p_2, v_2, and a_2. But if the paths are changing dynamically because of sensor input information, this may not be possible. What we need is a transition scheme that tracks changes in the paths x_1 and x_2.

3 Transitioning by Blending

One way to achieve our objective is to simply blend the two paths together, using a blend function $\alpha(s)$, so that

$$x(s) = \alpha(s)x_2(s) + (1 - \alpha(s))x_1(s) \tag{5}$$

To meet the boundary conditions at $x_1(0)$ and $x_2(1)$, $\alpha(s)$ should itself satisfy the following boundary conditions:

$$\begin{aligned} \alpha(0) = 0 \quad & \alpha(1) = 1 \\ \dot{\alpha}(0) = 0 \quad & \dot{\alpha}(1) = 0 \\ \ddot{\alpha}(0) = 0 \quad & \ddot{\alpha}(1) = 0 \end{aligned} \tag{6}$$

This is achieved by letting α have the form

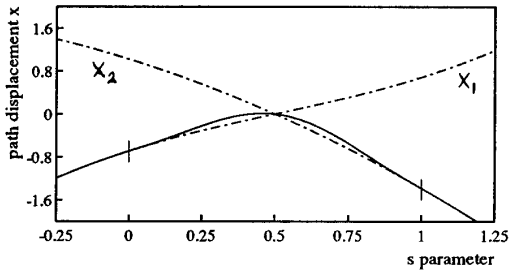

Figure 2: Direct blend of the paths x_1 and x_2.

$$\alpha(s) = 6s^5 - 15s^4 + 10s^3, \quad s \in [0,1] \tag{7}$$

The transition curve $x(s)$ is a one-dimensional Coon's patch along the time axis. Its appearance is shown in Figure 2. We are not yet done, however. Examining Figure 2, one may see that the blend curve has an undesirable tendency to speed up as it approaches the transition midpoint. Let us try to correct for this by adding an additional polynomial $\beta(s)$ to the transition curve.

What degree should $\beta(s)$ be? To leave the original boundary conditions undisturbed, its own position, velocity, and acceleration should be zero at both endpoints. Then, to do something useful, we will need at least one more degree of freedom. We therefore choose β to be of degree 6.

The relationship between the boundary conditions and the polynomials coefficients can be expressed using the following (underdetermined) Hermite matrix:

$$\mathbf{H}' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 30 & 20 & 12 & 6 & 2 & 0 & 0 \end{pmatrix} \tag{8}$$

If we have a set of coefficients $\mathbf{c}_0$ that satisfies this matrix, then for all other coefficients we have

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

where $\mathbf{k}$ is a parameter vector that matches 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 $\mathbf{k} \equiv k$, 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 with which we may adjust the characteristics of the transition curve.

Since the raw blend introduces acceleration into the transition, it would be reasonable to choose β so as to try to minimize the acceleration. For purposes of this analysis, we shall assume that the paths x_1 and x_2 are close to linear in the neighborhood of the transition interval, which implies that they can be approximated by

$$\begin{aligned} x_1 &= m_1 s + b_1 \\ x_2 &= m_2 s + b_2 \end{aligned} \tag{9}$$

Letting $\Delta m = m_2 - m_1$ and $\Delta b = b_2 - b_1$, we can expand (5) and find that

$$\begin{aligned} x(s) &= b_1 + m_1 s + 10\Delta b s^3 + (10\Delta m - 15\Delta b)s^4 + \\ &\quad (6\Delta b - 15\Delta m)s^5 + 6\Delta m s^6 \\ \ddot{x}(s) &= 60\Delta b s + (120\Delta m - 180\Delta b)s^2 + \\ &\quad (120\Delta b - 300\Delta m)s^3 + 180\Delta m s^4 \end{aligned} \tag{10}$$

Expanding $\ddot{\beta}(s)$ gives

$$\ddot{\beta}(s) = -6s + 36s^2 - 60s^3 + 30s^4. \tag{11}$$

We now combine these to compute the mean square acceleration as a function of k:

$$\begin{aligned} \int_0^1 (\ddot{x}(s) + k\ddot{\beta}(s))^2 \delta s &= \frac{120\Delta b(\Delta b + \Delta m) + 60\Delta m^2}{7} \\ &\quad + \frac{30\Delta m k + 2k^2}{35} \end{aligned} \tag{12}$$

which is minimized for $k = -15/2\Delta m$.

Note that Δm is simply $v_2 - v_1$, or the rate at which the two curves are moving toward each other at time $s = 0$. This means that we can compute k given only local knowledge of x_1 and x_2 at the beginning of the transition. Of course, if x_1 and x_2 turn out to be nonlinear, then the acceleration may not necessarily be minimized, and the final curve will distort in response to the nonlinearities.

We may not always want to set k to $-15/2\Delta m$. In particular, if we set k to $-6\Delta m$ instead, then we notice that the high order term for $x(s)$ in (10) cancels out, and the transition curve is described by a polynomial of degree 5. This implies that curve is less likely to wobble. In general, can define a parameter κ, such that

$$k = -\kappa\Delta m$$

which (very qualitatively) gives us a sort of "damper control" on the transition curve. The transition curve is then computed from

$$x(s) = \alpha(s)x_2(s) + (1 - \alpha(s))x_1(s) - \kappa\Delta m\beta(s) \tag{13}$$

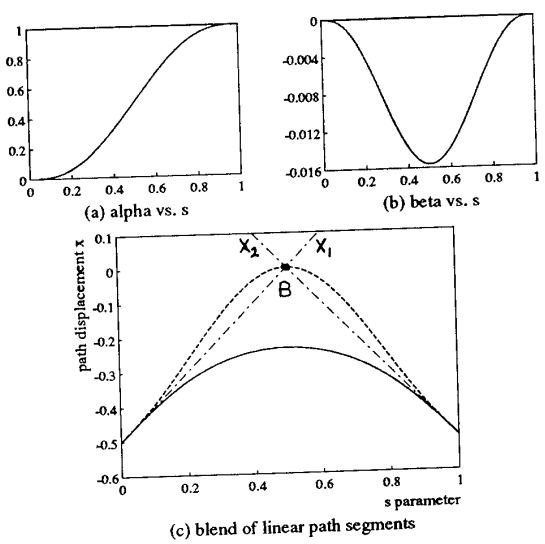

Figure 3: (a) Plot of α. (b) Plot of β. (c) Blends applied to the intersection of two linear path segments. The dotted line is the curve for which $\kappa = 0$, while the solid line is the curve for which $\kappa = 15/2$.

Throughout the rest of this paper, κ will usually be set to 6.

Plots of α, β, and $x(s)$ for $\kappa = 0$ and $\kappa = -15/2$ are given in Figure 3.

4 Controlling the Transition Shape

We may be interested in controlling other characteristics of the transition, and particularly the way in which the space curve wanders from the specified paths during the transition. It may be desirable of have the transition curve actually pass through (or over) the via point, or we may want to follow one of the path segments faithfully for the entire motion, and handle the transition with an overshoot at the end point. In this section, we will introduce a pair of parameters to give us that sort of control over the curve.

It is useful to think of a transition as having two components, where the first component brings the motion along path x_1 to a halt (at some target point B, which intersects path x_2), and the second component starts up the motion along path x_2, away from B. This can be expressed by expanding (13) in terms of B:

$$\begin{aligned} x(s) &= \alpha(s)B + (1-\alpha(s))x_1(s) \\ & \quad \alpha(s)x_2(s) + (1-\alpha(s))B - \kappa\Delta m\beta(s) \end{aligned} \tag{14}$$

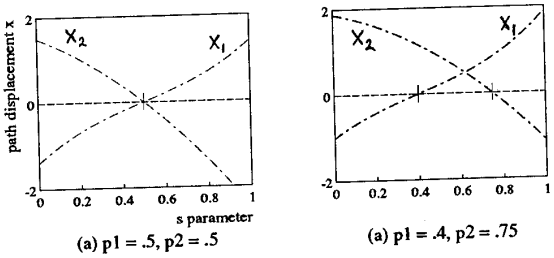

Figure 4: (a) Path intersections with $\rho_1 = \rho_2 = .5$. (b) Path intersections with $\rho_1 = .4$ and $\rho_2 = .75$.

Thus far, we have tended to assume that the path segments x_1 and x_2 meet B simultaneously near the midpoint of the transition (as illustrated in Figure 4(a)). However, this does not have to be the case. For each of the two path segments, we define a *preview* factor $\rho \in [0,1]$ which describes the value of s in the transition window for which the associated path meets B. ρ_1 and ρ_2 define the preview factors for x_1 and x_2, respectively. Figure 4(b) gives an example of a situation with $\rho_1 = .4$ and $\rho_2 = .75$.

Qualitatively, the preview factor is a measure of how much the trajectory generator is allowed to "plan ahead" in computing the transition for the associated curve. The effect of the parameters are most clearly illustrated by examining the associated position/time and space curves. Figure 5 shows a situation where we have two linear, orthogonal paths which intersect at $B = (0,0)$. x_1 describes a motion along the unit vector $\sqrt{2}/2(1,1)$, while x_2 describes a motion along $\sqrt{2}/2(1,-1)$. The left-hand figures plot the path parameter vs. time; the right-hand figures show the corresponding space curves. With $\rho_1 = \rho_2 = 0.5$, we have the conventional transition in Figure 5(a). In Figure 5(b), setting the preview factors to $\rho_1 = .3125$ and $\rho_2 = .6875$ causes the transition to travel through the via point B (the magic values depend on κ, which is 6 is these examples). In Figure 5(c), we let $\rho_1 = 0$ and $\rho_2 = .5$, which causes path x_1 to be followed all the way to the end, where it then overshoots as it begins the blend into path x_2. The opposite case, preshooting in order to follow path x_2 completely for its entire length, is shown in Figure 5(d). Finally, by setting $\rho_1 = 0$ and $\rho_2 = 1$, it is possible to follow both paths exactly, at the expense of overshooting both and looping around.

Figure 6 shows the velocity and acceleration magnitudes resulting from transitioning between two orthogonal path segments, with different preview values.

It is reasonable to use the following rules of thumb when setting preview values. If $\rho_1 + \rho_2 = 1$, and the

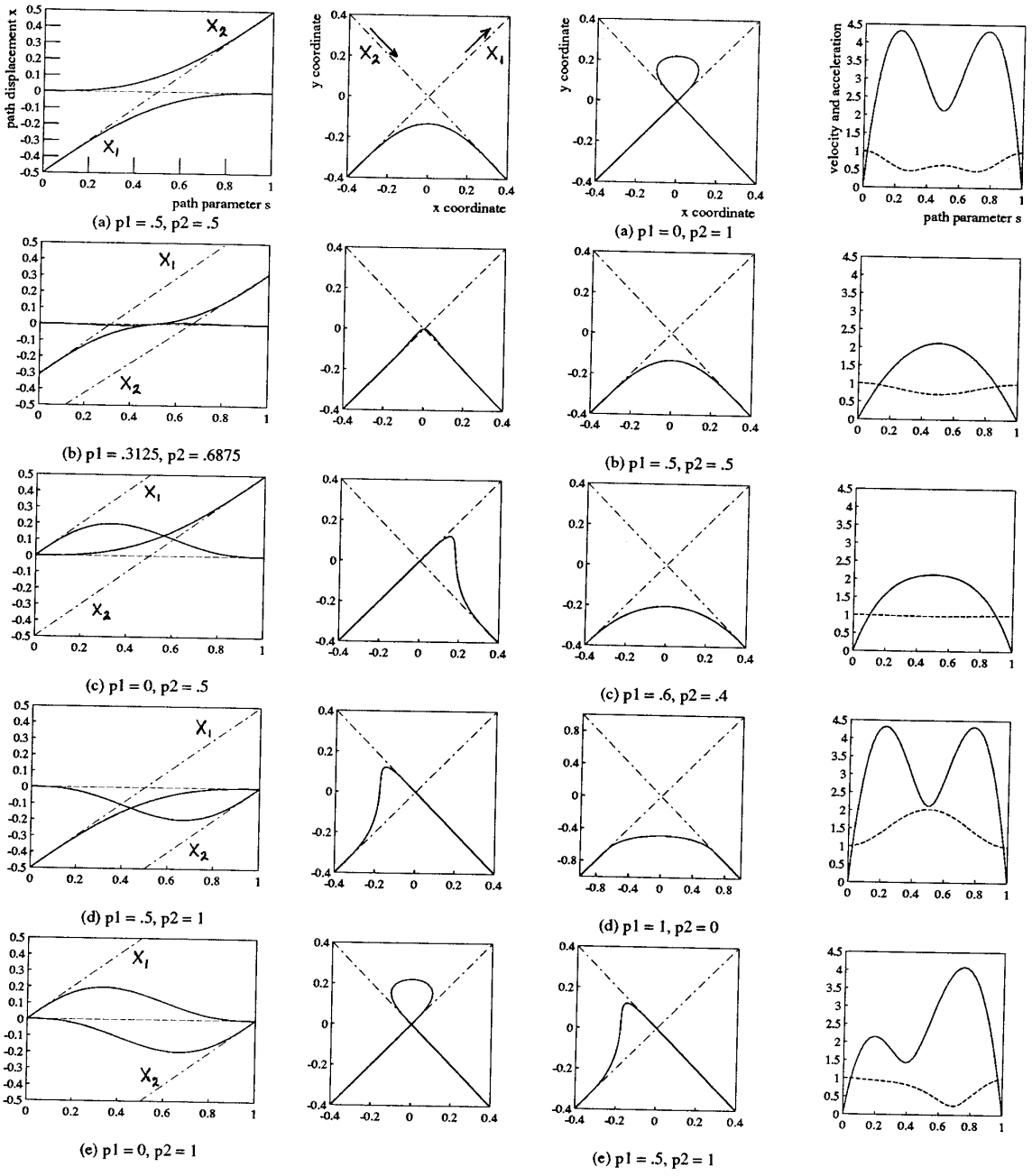

Figure 5: Position/time curves (on the left) and space trajectory curves (on the right) for transitions between two orthogonal linear path segments, showing the effect of different settings of the preview parameters. The figures on the left show the two path segments vs. time (hatched lines) and their corresponding transition components (solid lines). The figures on the right indicate the space curve for the two path segments (hatched lines) and the space curve generated by the transition (solid line).

Figure 6: Space trajectory curves are shown on the left, and plots of the magnitudes of the corresponding velocities (dotted lines) and accelerations (solid lines) are shown on the right, for different settings of the preview parameters.

initial and final path velocities are equal, then the space curve will be symmetric about the via point. Generally, as $\rho_1 \rightarrow 0$ and $\rho_2 \rightarrow 1$, the speed during transition will decrease. Specifically, for a κ value of 6, if $\rho_1 \in [.5, 1]$ (or $\rho_2 \in [0, .5]$), we will get an overshoot in the initial (or final) path velocity.

5 Applying the Transition Paradigm

The transition blending technique is quite simple and can be implemented for paths generated in any vectorized coordinates system (or in Cartesian coordinates if we use the rotation blending scheme described in the next section).

To implement the preview ρ_1, we simply begin the transition $-T\rho_1$ time units before the scheduled arrival time of path x_1 at its goal point B, where T is the total time allocated for the transition window. We then adjust the parameterization from which path x_2 will be computed to ensure that it meets B at a time $T\rho_2$ units from the start of the transition. In a straight-line motion interpolation scheme such as in [8], this would be done by appropriately setting the "drive" parameters (either joint or Cartesian). Notice that to do this we need to know the goal position B. But this may not always be possible if x_1 is dependent on sensor information. In that case, B may be estimated from x_1 and $\dot{x}_1$. Inaccuracies in this estimate may perturb the transition slightly, but will not affect the end-point conditions. Once we have path x_2, then Δm can be computed from from $\dot{x}_2 - \dot{x}_1$ at the beginning of the transition.

We may not always want to continue computing x_1 after the transition has started. This is true in cases where the computation is cumbersome, such as when x_1 and x_2 are originally computed in different coordinate systems and doing the full blend would require mapping from one into the other. In such cases we have found that it is usually acceptable to compute a "fake" x_1 path by simply extrapolating x_1 from its initial conditions.

6 Working with Rotations

When working in Cartesian coordinates, it is necessary to blend paths in rotation space. 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.

We are unable to describe the methods for doing this because of space limitations.

7 Conclusion

We have found a simple method for computing the transition between manipulator trajectory segments that is tolerant of dynamic changes in the path. This is of particular concern if the path is coupled to real-time sensory information.

We have also found that if we decompose the trajectory blend into a "stop" operation and a "start" operation, then we can define two "preview" parameters, ρ_1 and ρ_2, the setting of which gives us considerable control over the time/space shape of the transition curve. This decomposition also provides us with a natural way to do the blend computation for rotations.

References

[1] R. L. Andersson, "Aggressive Trajectory Generator for a Robot Ping-Pong Player". *IEEE Conference on Robotics and Automation*, Philadelphia, Pa., April 1988, pp. 188 – 193.

[2] Chun-Shin Lin, Po-Rong Chang, and J.Y.S.Luh, "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots". *IEEE Transactions on Automatic Control*, December 1983, pp. 1066 – 1073.

[3] J.D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, July 1984.

[4] Vincent Hayward, Laeeque Daneshmend, Ajit Nilakantan, "Model Based Trajectory Planning Using Preview". Report TR-CIM-88-9, McGill Research Center for Intelligent Machines, McGill University, Montreal, March 1988.

[5] Hayward, V., Daneshmend, L. K., Hayati, S., "An overview of Kali: a system to program and control cooperative manipulators". *Fourth International Conference on Advanced Robotics*, Columbus, Ohio, June 1989, Springer-Verlag pp. 547–558.

[6] Hollerbach, J. M., "Dynamic scaling of manipulator trajectories". *ASME J. Dynamic Systems, Measurement, and Control*, 1984. Vol. 106, pp. 102–106.

[7] John Lloyd, Mike Parker, and Rick McClain, "Extending the RCCL Programming Environment to Multiple Robots and Processors". *IEEE Conference on Robotics and Automation*, Philadelphia, Pa., April 24-29, 1988, pp. 465 – 469.

[8] Richard P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, Mass., 1981

[9] Kang G. Shin and Neil D. McKay, "Minimum-Time Trajectory Planning for Industrial Robots with General Torque Constraints". *IEEE Conference on Robotics and Automation*, San Fransisco, Ca., April 7-10, 1986, pp. 412 – 417.

[10] R.H. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories". *IBM Journal of Research and Development*, July 1979, pp. 253 – 264.

[11] Stuart E. Thompson and Rajnikant V. Patel, "Formulation of Joint Trajectories for Industrial Robots Using B-Splines". *IEEE Transactions on Industrial Electronics*, May 1987, pp. 192 – 199. (Vol. IE-34, No. 2)