Singularity-Robust Trajectory Generation for Robotic Manipulators

John E. Lloyd
Computer Science Dept., University of British Columbia
201-2366 Main Mall
Vancouver, B.C., V6T 1Z4, Canada
lloyd@cs.ubc.ca

Revised, September 2000

Technical Report TR-98-01
Computer Science Department
University of British Columbia

Abstract

A singularity-robust trajectory generator is presented which, given a prescribed manipulator path and corresponding kinematic solution, computes a feasible trajectory in the presence of kinematic singularities. The resulting trajectory is close to minimum time, subject to individual bounds on joint velocities and accelerations, and follows the path with precision. The algorithm has complexity $O(M \log M)$, where M is the number of robot joints, and works using "coordinate pivoting", in which the path timing near singularities is controlled using the fastest changing joint coordinate. This allows the handling of singular situations, including linear self-motions (e.g., wrist singularities), where the speed along the path is zero but some joint velocities are non-zero. To compute the trajectory, knot points are inserted along the path, dividing it into intervals, with the knot density increasing near singularities. An appropriate path velocity is then computed at each knot point, and the resulting knot-velocity sequence is integrated to yield the path timing. Examples involving the PUMA manipulator are shown.

1 Introduction

Robotic manipulators, when requested to follow prescribed spatial paths, frequently encounter kinematic singularities which can cause joint velocities (and higher time derivatives) to become unacceptably large. Singularities often coincide with a workspace boundary, but can occur inside the workspace as well. They reduce the useful size of a manipulator's workspace, and render certain types of operations (e.g., full arm extension using straight line motion) impossible.

While it is sometimes possible to avoid singularities, this complicates the motion planning process and limits the range of tasks which can be performed. However, recent work on the singularity problem (Section 2) has indicated that, with proper time scaling, it is generally possible to follow any prescribed spatial path near (and at) a singularity, without deviating from the path and without incurring large joint velocities.

Our paper builds on this result and presents a *singularity-robust trajectory generator*, which can take any inverse kinematic solution for a prescribed spatial path and produce a path timing (i.e., trajectory) which is close to minimum time, subject to keeping joint velocities and accelerations within prescribed bounds. The algorithm handles all types of singularities except those involving non-linear self-motions (Section 3.1), and is efficient and easy to implement. It should also be possible to extend the algorithm to handle general torque constraints imposed by the manipulator dynamics. An implementation is available from

http://www.cs.ubc.ca/spider/lloyd/singRobustTgen.html.

While the problem of this paper has already been studied with respect to velocity limits alone (Chiacchio & Chiaverini, 1995), our work is novel in that it also incorporates acceleration limits. The ensuing problem is somewhat trickier (for reasons described at the end of Section 2) and requires planning over the whole path. As such, the algorithm described here is well-suited to form a "black box" back end to a motion planner or programmer interface. Provided that time scaling is acceptable, path planning can then be performed by considering only workspace limits and physical obstacles, with the singularity-robust trajectory generator ensuring that joint velocities and accelerations are properly bounded whenever singularities are encountered.

2 Related Work

The most traditional way of handling singularities involves the manipulator Jacobian $\mathbf{J}$, which relates joint velocities $\dot{\mathbf{q}}$ to the spatial velocity $\mathbf{v}$ according to

$$\mathbf{v} = \mathbf{J}\,\dot{\mathbf{q}}. \tag{1}$$

At a singularity, $\mathbf{J}$ becomes rank deficient, and so inverting (1) to find the $\mathbf{v}$ required to follow a particular path may result in arbitrarily large values of $\dot{\mathbf{q}}$. One way of handling this is to introduce a damping factor into the inverse calculations which limits the magnitude of $\dot{\mathbf{q}}$ at the expense of deviating from the desired path (Nakamura & Hanafusa, 1986; Wampler II & Leifer, 1988). Because this can result in motions which are somewhat sluggish, a number of researchers have considered ways to dynamically compute and optimize the damping parameter (Maciejewski & Klein, 1989; Kircanski, 1993; Deo & Walker, 1993; Chiaverini *et al.*, 1994). Other Jacobian-based methods involve removing degenerate degrees of freedom from $\mathbf{J}$ (Aboaf & Paul, 1987), or using $\mathbf{J}^T$ in place of the inverse (or pseudo-inverse) of $\mathbf{J}$ (Wolovich & Elliott, 1984; Chiacchio *et al.*, 1991).

Somewhat more recently, it has been observed that by considering higher-order differential behavior, exact path tracking is generally possible at singularities if an appropriate path timing is supplied. If the path is parameterized by a scalar s, this entails making $\dot{s}$ (and usually higher derivatives) zero at the singular point. Examples of this for simple robots were shown in (Nielsen *et al.*, 1990; Chevallereau & Daya, 1994). The idea is equivalent to finding a reparameterization $\lambda = f(s)$

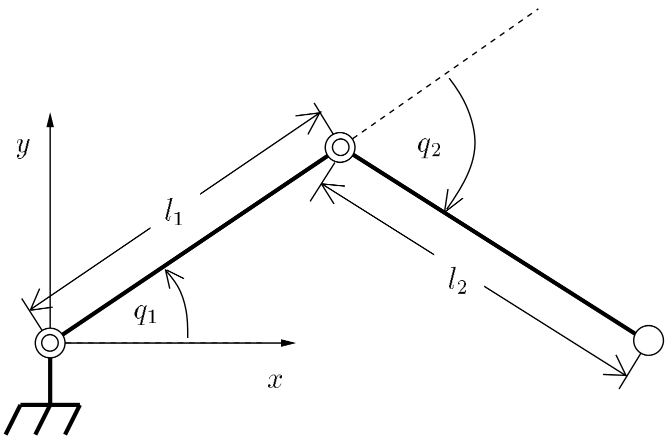

Figure 1: Planar 2R robot centered at the origin.

such that the path's inverse kinematic solution $\mathbf{q}(s)$ is smooth with respect to λ. In (Kieffer, 1994) it was shown that these reparameterizations always exist for non-redundant robots if $\mathbf{J}$ has a rank deficiency of one and the path's tangent has a component in the singular direction. Other reparameterization conditions were presented in (Chevallereau, 1996; O'Neil *et al.*, 1997). For non-redundant robots, it can be proven that a smooth reparameterization of the path always exists at any singularity with a finite root multiplicity (Lloyd, 1998).

Some authors have applied these ideas to on-line robot control. If $\mathbf{J}$ is square and has a rank deficiency of one, then the manipulator's motion can be controlled within the (one-dimensional) null space of $\mathbf{J}$ (Senft & Hirzinger, 1995; Nenchev *et al.*, 1996; Chang & Khatib, 1995). Alternatively, information from the manipulator's Hessian can sometimes be used to solve for $\dot{\mathbf{q}}$ (Tumeh & Alford, 1988; Pohl & Lipkin, 1991; O'Neil *et al.*, 1997). In a different approach, it is shown in (Lloyd & Hayward, 1998) that the reparameterization described in (Lloyd, 1996) can sometimes be applied to the workspace itself, such that all motions planned within this transformed workspace have well-behaved joint velocity profiles.

When using the above-mentioned techniques, it can be difficult to place explicit limits on acceleration, particularly if there are to be no deviations from the path and a time-efficient motion is desired. The reason why can be demonstrated using the planar 2R robot in Figure 1, by considering the behavior of joint q_2 when the robot is driven along a straight line into the singularity at the outer workspace boundary (Figure 2).

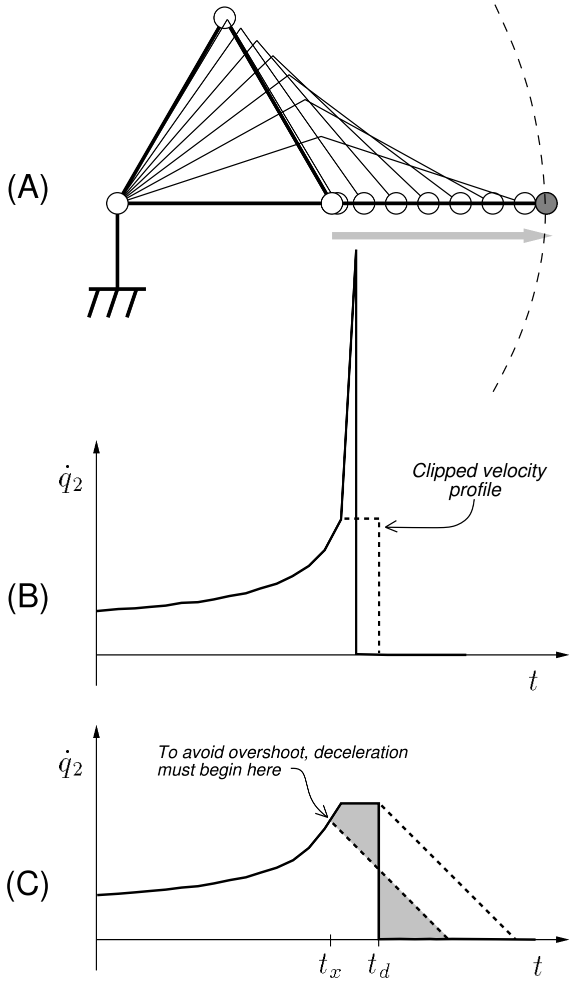

Figure 2: (A) Straight-line motion into the outer boundary singularity (dashed circular line). (B) Nominal velocity profile $\dot{q}_2(t)$ resulting when this motion is performed at constant speed (solid line), and when the motion is time-scaled so as to limit $\dot{q}_2$ to some maximum value (dotted line). (C) Effect of removing the discontinuity in $\dot{q}_2(t)$ by limiting $|\ddot{q}_2|$. If this is done directly at the discontinuity (time t_d), the resulting velocity profile (rightmost dotted line) will cause an overshoot in q_2. To avoid this, it is necessary to begin the deceleration at an earlier time t_x (leftmost dotted line), so that q_2 is brought to rest exactly as the singularity is reached. t_x must be located so that the areas of the two shaded regions match.

Doing this at constant speed produces a large spike in $\dot{q}_2$ (solid line, Figure 2.B). The high-velocity associated with this spike can be dealt with fairly easily: as the singularity is approached, one may scale the path velocity $\dot{s}$ in direct proportion to the minimum singular value of $\mathbf{J}$ (similar to the method of (Chiacchio & Chiaverini, 1995)). This has the effect of roughly "clipping" $|\dot{q}_2|$ to some maximum value (Figure 2.B, dotted line). However, this is not sufficient to ensure reasonable robot behavior, since $\dot{q}_2$ still experiences a discontinuity (corresponding to a spike in $\ddot{q}_2$) when the robot halts at the workspace boundary. It is therefore important to limit acceleration as well.

The difficulty with this is that limiting $|\ddot{q}_2|$ also affects q_2. For instance, suppose we limit $|\ddot{q}_2|$ directly at the velocity discontinuity, as shown in Figure 2.C. This creates a controlled deceleration profile, beginning at t_d, indicated by the rightmost dotted line. However, this results in q_2 overshooting its proper value at the singularity, causing deviation from the prescribed path (as evidenced by the area under the new velocity profile being larger than that under the original profile). In order to limit acceleration *and* preserve the correct joint displacement, it is necessary to start decelerating at some point t_x *before* arriving at the singularity (as indicated by the leftmost dotted line in Figure 2.C).

Placing tight limits on acceleration therefore necessitates the use of global trajectory planning. Our algorithm does this using techniques similar to those used for general time-optimal path-following (Bobrow *et al.*, 1985; Shin & McKay, 1985; Slotine & Yang, 1989), with velocity and acceleration constraints used instead of dynamics-based torque constraints. The resulting algorithm produces near minimum-time trajectories, in the presence of both ordinary singularities and those involving linear self-motions (Section 3.1). It is an improvement on some earlier work (Lloyd & Hayward, 1996) which did not handle self-motions.

3 Algorithm Overview

Note to the reader: a notation summary is given in Appendix G.

Let the spatial path to be followed be described by $\mathbf{X}(s)$, where $\mathbf{X}$ is a 4×4 homogeneous transform matrix and s is a scalar path parameter defined over some interval $[s_A, s_B]$. $\mathbf{X}(s)$ is assumed to be continuous. The robot has M joints whose values are given by $(q_1, \dots, q_M)^T$. The vector of joint coordinates is given by $\mathbf{q}$. For reasons that will become clear later, it is useful to extend $\mathbf{q}$ to include the path parameter s as coordinate q_{M+1}, so that

$$\mathbf{q} \equiv (q_1, \dots, q_M, q_{M+1})^T, \quad q_{M+1} \equiv s. \tag{2}$$

The algorithm assumes that an inverse kinematic solution for $\mathbf{X}(s)$ is provided; this will be denoted by $\mathbf{q}(s)$ (see Section 7.3 for comments about determining this solution). $\mathbf{q}'(s)$ is also required, although this can be computed numerically from $\mathbf{q}(s)$ and $\mathbf{q}(s+\epsilon)$ for some small ϵ (which has the advantage of producing values which are always finite, even at singularities).

Given $\mathbf{q}(s)$ and $\mathbf{q}'(s)$, the algorithm produces an efficient path timing $s(t)$ for which the resulting coordinate velocities $\dot{\mathbf{q}}$ and accelerations $\ddot{\mathbf{q}}$ adhere closely to the following limits:

$$|\dot{q}_j| \leq V_j, \quad |\ddot{q}_j| \leq A_j, \tag{3}$$

where V_j and A_j represent the (possibly different) velocity and acceleration bounds for each coordinate. Because $\mathbf{q}$ includes s as q_{M+1}, these constraints also impose limits on $\dot{s}$ and $\ddot{s}$, which will be specifically referred to as V_s and A_s. These limits provide control over task velocities and accelerations; they can be set to very high values if unneeded.

As the robot is made to follow $\mathbf{q}(s)$, the velocity and acceleration of each coordinate q_j depends on $\dot{s}$ and $\ddot{s}$, via the chain rule:

$$\dot{q}_j = q_j'(s)\,\dot{s} \quad \text{and} \quad \ddot{q}_j = q_j'(s)\,\ddot{s} + q_j''(s)\,\dot{s}^2. \tag{4}$$

Near singularities, $q_j'(s)$ and $q_j''(s)$ may become very large, implying that $\dot{q}_j$ and $\ddot{q}_j$ may also become very large for non-zero values of $\dot{s}$ and $\ddot{s}$ (at a self-motion singularity, $q_j(s)$ may even be discontinuous). In such situations, the algorithm uses the velocity of some *joint* coordinate, in place of $\dot{s}$, to control the path timing. This is known as *coordinate pivoting*, and is described in Section 4. Whichever coordinate is used to control the path timing, s or otherwise, is called the *driving coordinate*, and is denoted by x.

In standard time-optimal path-following procedures (Bobrow *et al.*, 1985; Shin & McKay, 1985), the timing $s(t)$ is actually produced implicitly by computing $\dot{s}$ as a function of s. The same thing is done here, using the driving coordinate x (whose identity will vary along the path) in place of s. The algorithm comprises two main steps:

1. *CreateKnots*: Knot points s_i are introduced along the path, dividing it into a sequence of intervals $[s_i, s_{i+1}]$. The knots are added by recursive bisection until each of the intervals satisfies conditions needed for good algorithm performance (Section 5). The resulting knot density is usually higher near singularities. A suitable driving coordinate x is determined for each interval.

2. *AssignVelocities*: Appropriate values for $\dot{x}$ are calculated at each knot, so as to produce a near-minimum time motion that closely satisfies $|\dot{q}_j| \leq V_j$ and $|\ddot{q}_j| \leq A_j$ (Section 6).

After drive coordinate velocities $\dot{x}$ have been assigned, the resulting knot-velocity sequence can be integrated to yield the path timing (Section 7.2). This is done assuming that $\ddot{x}$ is constant between knots, making this integration fairly easy to perform.

3.1 Examples

Our algorithm directly handles both ordinary singularities (i.e., those not associated with self-motion) and linear self-motion singularities (i.e., those for which the self-motion forms a straight line in joint space)*. The so-called "wrist" singularity is probably the most common example of a linear-self-motion singularity.

Both types of singularity can be illustrated by the planar 2R manipulator of Figure 1, with link lengths $l_1 = l_2 = 1$. Our attention will be restricted to the values of q_1 arising from straight-line motions along the x axis, as shown in Figure 3.

Ordinary singularities exist at $x = \pm 2$, where the x-axis intersects the outer workspace boundary and the two solution branches defined for $x \in (-2, 2)$ meet. A linear self-motion singularity exists at $x = 0$, where the manipulator folds up on itself and can spin freely about the q_1 axis without affecting the tip's position.

Now consider a path along the x axis defined by $x(s) = s, y(s) = 0$, for $s \geq -2$. The solution $q_1(s)$ depicted by the solid line in Figure 3 corresponds to the motion shown in Figure 4.A. At the outer boundary singularities ($s = \pm 2$), $q_1'(s)$ becomes infinite, and so by equation (4) any non-zero path speed $\dot{s}$ will result in infinite values for $\dot{q}_1$ and $\ddot{q}_1$. However, both $\dot{q}_1$ and $\ddot{q}_1$ can be kept bounded if $\dot{s}$ is brought to 0 appropriately at the singularity. Whether this means $\dot{q}_1$ will *also* have to be brought to 0 depends on the situation. For example, if the robot is brought to rest at the singularity, then clearly $\dot{q}_1$ will have to be brought to 0 as well. On the other hand, suppose the manipulator reverses direction at the singularity, by following the path $x(s)$ defined by

$$x(s) = \begin{cases} s & \text{if } s \leq 2, \\ 4 - s & \text{if } s > 2. \end{cases}$$

*Self-motions are joint-space manifolds along which the manipulator can execute a finite joint motion without incurring a change in end-effector position.

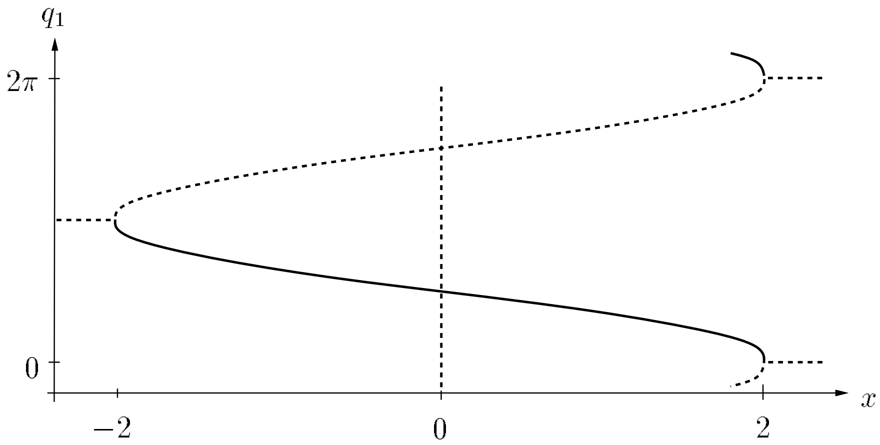

Figure 3: Solution set for q_1 (solid and dotted lines) of the planar 2R robot shown in Figure 1, for motion along the x axis. Workspace limits are exceeded for $x < -2$ or $x > 2$; the horizontal dotted lines in these regions indicate the "closest possible" solutions $q_1 \equiv \pi$ and $q_1 \equiv 0$ formed by projecting the desired position onto the workspace boundary.

By permitting $q_1(s)$ to switch branches at $s = 2$, we obtain the motion shown in Figure 4.B. Because $\dot{q}_1$ does not then change sign as it passes through the singularity, a timing exists which bounds $\dot{q}_1$ without bringing it to 0. A robust trajectory generator should be able to construct such timings, in which one or more $\dot{q}_j \neq 0$ while $\dot{s} = 0$. These timings are difficult to create using $\dot{s}$, which is why we employ coordinate pivoting.

The vertical dotted line at $x = 0$ in Figure 3 indicates the solution associated with the linear self-motion singularity. It is possible to use this to switch solution branches, by bringing the robot to rest at the singularity, moving along the self-motion solution, and then resuming along the new branch (Figure 4.C). The associated $q_1(s)$ will contain a discontinuity, which can be handled by our algorithm, resulting in the aforementioned branch-switching behavior. While it is usually preferable to avoid switching branches in this situation (as in Figure 4.A), task constraints, such as the presence of obstacles or joint limits, may dictate otherwise. Also, paths which pass *near* self-motion singularities may produce solutions $q_j(s)$ which are very steep and thereby numerically appear discontinuous. For these reasons, a robust trajectory generator should be capable of detecting and handling discontinuities in $q_j(s)$.

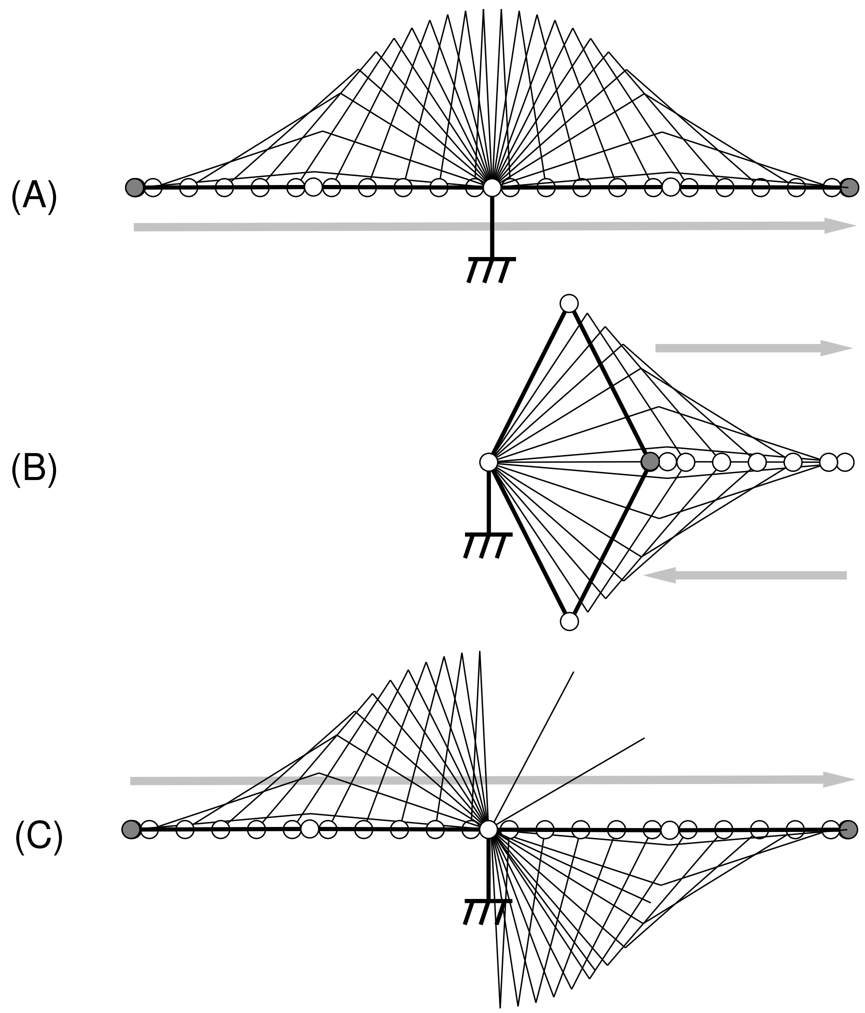

Figure 4: (A) Motion along the x axis corresponding to the solid line in Figure 3. (B) Motion out to and back from the outer boundary singularity, in which the q_1 solution branch is changed at the singularity. (C) Motion along the x axis in which the q_1 solution branch is changed at the self-motion singularity at the origin.

4 Coordinate Pivoting

As described in Section 3, the algorithm uses an alternate driving coordinate x to control the path timing across intervals where one or more $q_j'(s)$ becomes large. We call this *coordinate pivoting*.

To formalize this idea, let $\mathbf{q}_i \equiv \mathbf{q}(s_i)$ denote the value of $\mathbf{q}(s)$ at knot i, let ${}^i\Delta\mathbf{q} \equiv \mathbf{q}_{i+1} - \mathbf{q}_i$ denote the change in $\mathbf{q}$ across interval i (which lies between knots i and $i+1$), and let ${}^i\Delta q_j$ denote an individual element of ${}^i\Delta\mathbf{q}$. Then:

Definition 1. *The driving coordinate for interval i is that for which $|{}^i\Delta q_j|/\alpha_j$ is a maximum, where α_j is a normalizing factor (usually 2π for revolute joints or the workspace diameter for prismatic joints), and is represented by ${}^i x$.*

Within interval i, the coordinates $\mathbf{q}$ (including s) are treated as a function $\mathbf{q}({}^i x)$ of the driving coordinate ${}^i x$ (our ability to do this follows from ensuring that $s({}^i x)$ is strictly monotone, as described in Section 5.4). The function $\mathbf{q}({}^i x)$ is approximated using cubic Hermite interpolation of the interval endpoint values of $\mathbf{q}$ and $\mathbf{q}'({}^i x)$. Interpolation errors are bounded by making sure knots i and $i+1$ are sufficiently close together (Section 5.1). Individual endpoint values of $\mathbf{q}'({}^i x)$ can be determined from

$$q_j'({}^i x) = q_j'(s) / {}^i x'(s). \tag{5}$$

In the present algorithm implementation, derivatives on the right side of this equation are computed numerically by taking finite differences between $q_j(s)$ and $q_j(s+\epsilon)$ for some small ϵ; this ensures that all numbers remain finite even at singularities. Because ${}^i x$ is the coordinate with the largest variation over the interval, we can expect $q_j'({}^i x)$ to be modestly sized for sufficiently small intervals.

As an example, refer to Figure 3 and consider a motion along the x-axis defined by $x(s) = s$. With respect to coordinates s and q_1 only, we would expect q_1 to be the driving coordinate in right and left neighborhoods of $s = -2$ and $s = 2$, respectively, and also at $s = 0$ if $q_1(s)$ switches branches along the self-motion. The driving coordinate elsewhere would be s.

Coordinate pivoting is a central feature of our algorithm. It is analogous to pivoting in matrix computations, where one chooses to divide a matrix row or column by the element with largest magnitude.

The following notation will be useful in the sequel. The values of ${}^i x$ at the interval end points i and $i+1$ will be represented as

$${}^i x_i \equiv {}^i x(s_i) \quad \text{and} \quad {}^i x_{i+1} \equiv {}^i x(s_{i+1}).$$

Likewise, the driving coordinate velocity will be denoted by ${}^i\dot{x}$, with values at the interval endpoints given by ${}^i\dot{x}_i$ and ${}^i\dot{x}_{i+1}$. The change in ${}^i x$ across the interval will be represented as ${}^i\Delta x \equiv {}^i x_{i+1} - {}^i x_i$. The value of the derivative $\mathbf{q}'({}^i x)$ at knots i and $i+1$ will be represented as

$$ {}^i\mathbf{q}'_i \equiv \mathbf{q}'({}^i x_i) \quad \text{and} \quad {}^i\mathbf{q}'_{i+1} \equiv \mathbf{q}'({}^i x_{i+1}), $$

with individual elements specified by ${}^i q'_{i,j}$ and ${}^i q'_{i+1,j}$. The change in $q'_j({}^i x)$ across the interval will be given by

$$ {}^i\Delta q'_j \equiv {}^i q'_{i+1,j} - {}^i q'_{i,j}. \tag{6} $$

We will also utilize the *average* values of $\mathbf{q}'({}^i x)$ and $\mathbf{q}''({}^i x)$ over interval i, respectively defined by

$$ {}^i\overline{\mathbf{q}}' \equiv \frac{\mathbf{q}_{i+1} - \mathbf{q}_i}{{}^i\Delta x} \quad \text{and} \quad {}^i\overline{\mathbf{q}}'' \equiv \frac{{}^i\mathbf{q}'_{i+1} - {}^i\mathbf{q}'_i}{{}^i\Delta x}, \tag{7} $$

with individual elements specified by ${}^i\overline{q}'_j$ and ${}^i\overline{q}''_j$.

4.1 Controlling path timing with drive coordinates

Path timing within interval i is controlled by specifying ${}^i\dot{x}(t)$, with other velocities $\dot{q}_j$ then determined by $\dot{q}_j = q'_j({}^i x)\,{}^i\dot{x}$. The timing produced by the algorithm employs a constant value for ${}^i\ddot{x}$ over the interval, implying that ${}^i\dot{x}(t)$ is linear.

Because $s({}^i x)$ is strictly monotone (Section 5.4), ${}^i\dot{x}$ does not change sign within the interval, and so ${}^i\dot{x}$ can also be specified as a function of ${}^i x$. If we do this, then the constancy of ${}^i\ddot{x}$ implies that ${}^i\dot{x}^2({}^i x)$ is also linear, since

$$ \frac{d({}^i\dot{x}^2({}^i x))}{d\,{}^i x} = \frac{d({}^i\dot{x}^2(t))}{dt}\frac{dt}{d\,{}^i x} = 2\,{}^i\ddot{x}\,\frac{d\,{}^i x}{dt}\frac{dt}{d\,{}^i x} = 2\,{}^i\ddot{x}. \tag{8} $$

Moreover, with respect to the endpoint velocities ${}^i\dot{x}_i$ and ${}^i\dot{x}_{i+1}$, we have

$$ {}^i\ddot{x} = \frac{{}^i\dot{x}^2_{i+1} - {}^i\dot{x}^2_i}{2\,{}^i\Delta x}. \tag{9} $$

If there exist n contiguous intervals with the same drive coordinate x, then the timing over these intervals can be specified by a function $\dot{x}^2(x)$ which is linear within each interval and piecewise-linear over the whole interval set.

4.2 Maintaining velocity continuity

To preserve velocity continuity, the drive coordinate velocities ${}^{i-1}\dot{x}$ and ${}^{i}\dot{x}$ for intervals $i-1$ and i must match appropriately at knot i. Within intervals $i-1$ and i, $\dot{\mathbf{q}}$ is determined by

$$\dot{\mathbf{q}} = \mathbf{q}'({}^{i-1}x)\ {}^{i-1}\dot{x} \quad \text{and} \quad \dot{\mathbf{q}} = \mathbf{q}'({}^{i}x)\ {}^{i}\dot{x},$$

respectively. Continuity of velocities at knot i therefore requires that

$$ {}^{i-1}\mathbf{q}'_i\ {}^{i-1}\dot{x}_i = {}^{i}\mathbf{q}'_i\ {}^{i}\dot{x}_i. \tag{10}$$

Now, if $\mathbf{q}'({}^{i}x)$ is continuous at knot i (which it will be unless i is a corner; Section 5.6), and if we define $C_i \equiv {}^{i-1}x'({}^{i}x)$, then by the chain rule

$$ {}^{i-1}\mathbf{q}'_i\ C_i = {}^{i}\mathbf{q}'_i. \tag{11}$$

From equation (10) it then follows that $\dot{\mathbf{q}}$ will be continuous at knot i if and only if ${}^{i-1}\dot{x}_i$ and ${}^{i}\dot{x}_i$ satisfy

$$ {}^{i-1}\dot{x}_i = {}^{i}\dot{x}_i\ C_i. \tag{12}$$

5 Knot Creation

The algorithm begins by inserting a nominal number of equally spaced knots along the path. Additional knots are then added by recursive bisection until each of the following conditions are satisfied for each interval:

(a) The path error is within prescribed bounds;

(b) Joint differences ${}^{i}\Delta q_j$ are within prescribed bounds;

(c) Joint derivative differences ${}^{i}\Delta q'_j$ are within prescribed bounds;

(d) The (interpolated) function $s({}^{i}x)$ is strictly monotone within the interval.

The next four sections detail each of these conditions and explain their relevance to algorithm performance.

5.1 Bounding the path error

Condition (a) ensures that in interpolating $\mathbf{q}({}^i x)$ across the interval, we do not deviate from $\mathbf{X}(s)$ by more than some prescribed tolerance. This is a standard problem in path generation, and is handled using the same sort of test described in (Taylor, 1979): Let $\mathbf{K}(\mathbf{q})$ be the manipulator forward kinematic function (returning a 4×4 homogeneous end-effector transform) and let

$$x_h \equiv \frac{{}^i x_i + {}^i x_{i+1}}{2}$$

be the mid-interval value for ${}^i x$, with corresponding coordinate and path parameter values given by $\mathbf{q}(x_h)$ and $s(x_h)$. Then the spatial displacement between $\mathbf{K}(\mathbf{q}(x_h))$ and $\mathbf{X}(s(x_h))$ is used to estimate the interpolation error. If this displacement is described by a 4×4 homogeneous transform, according to

$$\begin{pmatrix} \mathbf{R}_h & \mathbf{p}_h \\ 0 & 1 \end{pmatrix} \equiv \mathbf{K}(\mathbf{q}(x_h))^{-1}\, \mathbf{X}(s(x_h)),$$

and the tolerances for translational and rotational errors are ϵ_p and ϵ_r, then Condition (a) is met when

$$\|\mathbf{p}_h\| \leq \epsilon_p \quad \text{and} \quad \|\log \mathbf{R}_h\| \leq \epsilon_r \tag{13}$$

(using the notation of (Murray *et al.*, 1994), where $\log \mathbf{R}_h$ is a 3-vector representing the direction and magnitude of the rotation implied by $\mathbf{R}_h$).

5.2 Bounding coordinate differences

Condition (b) helps ensure that the algorithm's path timing closely respects each coordinate's individual velocity constraint $|\dot{q}_j| \leq V_j$. It also helps ensure that the path solution will be close to minimum-time.

We first consider the former criterion. By properly assigning path coordinate velocities ${}^i\dot{x}$, the algorithm ensures that $|\dot{q}_j| \leq V_j$ is satisfied exactly at each knot point (Section 6). But what about within the interval between knot points? If $\ddot{q}_j$ happened to be constant over the interval, then $\dot{q}_j(t)$ would be linear, would therefore never exceed its endpoint values, and $|\dot{q}_j| \leq V_j$ would be satisfied over the whole interval. This means we might expect the velocity constraints to be *closely* adhered to if $\ddot{q}_j$ is *nearly* constant over the interval. It turns out that near-constancy of $\ddot{q}_j$ is implied by Condition (c). More precisely, we show in Appendix A that for intervals where $q_j({}^i x)$ is strictly monotone (which will be true for most intervals when the knots are sufficiently close together), the bounds associated with conditions (b) and (c) imply that $|\dot{q}_j| \leq {}^5\!/_4 V_j$, and so velocity limits are guaranteed within 25% over the whole interval. In practice, the behavior is much better, since the treatment in Appendix A relies on conservative assumptions.

Now consider the second criterion (that the resulting path timing be close to minimum time). Because $\ddot{q}_j$ is approximately constant across any interval (by Condition (c), described below), then if $|{}^i\Delta q_j|$ is too large, it may be necessary to keep $|\ddot{q}_j|$ well below its maximum value A_j in order to honor the velocity constraint $|\dot{q}_j| \leq V_j$. This effect can be prevented by reducing the size of ${}^i\Delta q_j$: for situations where $q_j({}^ix)$ is monotone within interval i, it can be shown (Appendix B) that if $\Delta\dot{q}_j$ equals the change in velocity across the interval, then the corresponding average acceleration $\bar{\ddot{q}}_j$ satisfies

$$|\bar{\ddot{q}}_j| \geq \frac{1}{2} \frac{|\Delta\dot{q}_j|^2}{|{}^i\Delta q_j|}, \tag{14}$$

where $\bar{\ddot{q}}_j$ is defined in terms of the $\dot{q}_j$ endpoint values ($\dot{q}_{i,j}$ and $\dot{q}_{i+1,j}$) and the interval transit time ${}^i\Delta t$:

$$\bar{\ddot{q}}_j = \frac{\dot{q}_{i+1,j} - \dot{q}_{i,j}}{{}^i\Delta t}. \tag{15}$$

Now if $\ddot{q}_j$ is approximately constant over the interval, it will be close to $\bar{\ddot{q}}_j$. Consequently, for any velocity change where $|\Delta\dot{q}_j| \geq V_j/2$, we can expect $|\ddot{q}_j|$ will be able to reach A_j if

$$|{}^i\Delta q_j| \leq \frac{V_j^2}{8A_j}. \tag{16}$$

This is the check used for Condition (b).

5.3 Bounding derivative differences

Condition (c) limits the local curvature of each $q_j({}^ix)$, which helps ensure that the algorithm's path timing closely adheres to each coordinate's individual acceleration constraint $|\ddot{q}_j| \leq A_j$. It also helps justify the algorithm's assumption that a constant drive coordinate acceleration ${}^i\ddot{x}$ across an interval will produce a roughly constant value for $\ddot{q}_j$.

If $\ddot{q}_j(t)$ *were* constant within the interval, then it would equal the average acceleration $\bar{\ddot{q}}_j$ defined by equation (15). Otherwise, let

$$E_{aj} \equiv \max_t |\bar{\ddot{q}}_j - \ddot{q}_j(t)| \tag{17}$$

characterize the maximum deviation, within the interval, of $\ddot{q}_j$ from its "ideal" value $\bar{\ddot{q}}_j$. Then by assuming that $q_j({}^i x)$ is approximately quadratic, it can be shown (Appendix C) that

$$E_{aj} < 2\,|{}^i\Delta q'_j|\,|{}^i\ddot{x}|. \tag{18}$$

By limiting $|{}^i\Delta q'_j|$ according to

$$|{}^i\Delta q'_j| \le \frac{A_j}{4A_x}, \tag{19}$$

where A_x is the maximum value for $|{}^i\ddot{x}|$, we can ensure that

$$E_{aj} \le A_j/2, \tag{20}$$

and so $\ddot{q}_j$ is guaranteed not to deviate from $\bar{\ddot{q}}_j$ by more than 50% of A_j. In practice, because inequality (18) relies on conservative assumptions, performance is generally much better than this.

If $q_j({}^i x)$ contains an inflection point within the interval, then $|{}^i\Delta q'_j|$ may not give a good indication of the local curvature of $q_j({}^i x)$. Because of this, we have found it better to check the following *two* inequalities in place of (19):

$$|{}^i q'_{i,j} - {}^i\bar{q}'_j| \le \frac{A_j}{8A_x} \tag{21}$$

and

$$|{}^i q'_{i+1,j} - {}^i\bar{q}'_j| \le \frac{A_j}{8A_x}. \tag{22}$$

Together, both these inequalities imply satisfaction of (19) and are the checks used for Condition (c).

5.4 Ensuring drive coordinate monotonicity

By requiring $s({}^i x)$ to be strictly monotone, Condition (d) ensures that we don't reverse direction along the path as an artifact of interpolating $\mathbf{q}({}^i x)$ within the interval. It also implies that ${}^i x$ will be monotone with respect to s, justifying our assumption that within an interval $\mathbf{q}$ can be treated as a function of ${}^i x$.

In cases where ${}^i x = s$, Condition (d) is trivially true. Otherwise, $s({}^i x)$ will be a cubic interpolated function of ${}^i x$, like every other coordinate. For notational ease, let ${}^i s'_i \equiv s'({}^i x_i)$ and ${}^i s'_{i+1} \equiv s'({}^i x_{i+1})$, and let

$${}^i\bar{s}' \equiv \frac{s_{i+1} - s_i}{{}^i\Delta x} \tag{23}$$

represent the average value of $s'({}^i x)$ within the interval. It can then be shown (Appendix D) that $s({}^i x)$ is strictly monotone if

$$|{}^i s_i' - {}^i\overline{s}'| < |{}^i\overline{s}'| \quad \text{and} \quad \mathrm{sgn}({}^i s_i') = \mathrm{sgn}({}^i\overline{s}') \tag{24}$$

and

$$|{}^i s_{i+1}' - {}^i\overline{s}'| < |{}^i\overline{s}'| \quad \text{and} \quad \mathrm{sgn}({}^i s_{i+1}') = \mathrm{sgn}({}^i\overline{s}'). \tag{25}$$

Satisfaction of both (24) and (25) are the checks used for Condition (d).

5.5 Knot creation summary

Each interval is recursively bisected until all the checks for conditions (a) through (d) are met. For purposes of the algorithm, it is useful to regroup these checks into four tests, as described in Table 1. In particular, Tests L and R collect the checks involving information local to the left and right of the interval, respectively, which facilitates the handling of corners or discontinuities, as described below.

Test name	Associated checks
Test A	(13)
Test B	(16)
Test L	(21), (24)
Test R	(22), (25)

Table 1: Tests for knot creation.

It is straightforward to show that repeated bisection will eventually ensure satisfaction of all tests, provided that $\mathbf{X}(s)$, $\mathbf{q}(s)$, and $\mathbf{q}'({}^i x)$ are continuous. However, while $\mathbf{X}(s)$ is continuous by assumption, discontinuities may sometimes arise in $\mathbf{q}(s)$ or $\mathbf{q}'({}^i x)$, generally at singularities. The handling of these is described in the next section, while the complete knot creation procedure is summarized in Algorithm 1.

5.6 Discontinuities and corners

Special treatment is required whenever discontinuities arise in $\mathbf{q}(s)$ or $\mathbf{q}'({}^i x)$ (with the latter referred to as *corners*).

Both possibilities are illustrated in Figure 3. For a motion along the x axis defined by $x(s) = s$ and $y(s) = 0$, $q_1(s)$ will contain corners at $s = \pm 2$ (where the regular solution branches meet the special "out of workspace" solutions defined by $q_1(s) \equiv 0$ or $q_1(s) \equiv \pi$), and a possible discontinuity at $s = 0$ (depending on whether or not solution branches are switched using the self-motion).

Corners are detected and handled as follows. Let ${}^i\Delta s \equiv s_{i+1} - s_i$ be the change in s across an interval. If, during recursive interval bisection, ${}^i\Delta s$ falls below a threshold ϵ_s, then if Test L (Section 5.5) fails, knot i is declared to be a corner, while if Test R fails, knot $i+1$ is declared to be a corner. If knot i is a corner, it implies that $\mathbf{q}({}^i x)$ has different left and right derivatives there, and equation (11) no longer holds. Instead, ${}^{i-1}\mathbf{q}'_i$ and ${}^i\mathbf{q}'_i$ are both recomputed independently, using adjacent curve information, as shown in Procedure 1. Because $\mathbf{q}'({}^i x)$ is discontinuous at any corner i, both ${}^i\dot{x}_i$ and ${}^{i-1}\dot{x}_i$ must be set to 0 there in order to maintain velocity continuity.

proc *makeCorner*(i) $\equiv$	
if $i > 1$ **then**	// if not first knot:
${}^{i-1}\mathbf{q}'_i := 2\,{}^{i-1}\overline{\mathbf{q}}' - {}^{i-1}\mathbf{q}'_{i-1}$	// compute left derivative
fi	
if $i < K$ **then**	// if not last knot:
${}^{i}\mathbf{q}'_i := 2\,{}^{i}\overline{\mathbf{q}}' - {}^{i}\mathbf{q}'_{i+1}$	// compute right derivative
fi	
$\mathcal{C} := \mathcal{C} \cup i$.	// add i to list of corner points

Procedure 1: $\mathcal{C}$ denotes the set of corner points and K gives the total number of knots.

Discontinuities in $\mathbf{q}(s)$ are detected when ${}^i\Delta s$ falls below ϵ_s and Test B is still not satisfied. When this happens, $\mathbf{q}$ is redefined within the interval as a linearly interpolated function of ${}^i x$:

$$\mathbf{q}({}^i x) := \mathbf{q}_i + \left(\frac{{}^i x - {}^i x_i}{{}^i\Delta x}\right) {}^i\Delta\mathbf{q}.$$

Additional evenly-spaced knots are added to the interval to ensure the satisfaction of Test B. The exact interpolation procedure for interval i, called *linearInterp*(i), is described in detail in Appendix E.

The handling of discontinuities is what permits the algorithm to function properly at linear self-motion singularities. If a discontinuity is caused by a *non-linear*

```
proc createKnots() ≡
   Start by creating n equally spaced knots
   i := 1, K := n
   C = {1, K}                          // first and last knots treated as corners
   while i < K do
         if ^iΔs > ε_s and (A), (B), (L), or (R) fails then // bisect
            Add new knot at s = (s_i + s_{i+1})/2
            K := K + 1, reindex knots > i
      else
            if (B) fails then          // some q_j(s) is discontinuous
               linearInterp(i)
            fi
            if (L) fails then          // some q_j(^ix) is discontinuous at i
               makeCorner(i)
            fi
            if (R) fails then          // some q_j(^ix) is discontinuous at i + 1
               makeCorner(i + 1)
            fi
            i := i + 1
         fi
   od
   for i := 1 to K − 1 do
      if i ∈ C and i + 1 ∈ C then
         linearInterp(i)               // add extra knot between adjacent corners
      fi
   od.
```

Algorithm 1: Knot creation procedure. A, B, L, and R refer to the test conditions defined in Section 5.5, and K is the total number of knots. The first knot 1 and the last knot K are treated as corners. The final loop uses *linearInterp()* to insert an additional knot between i and $i+1$ if they both happen to be corners; otherwise, because ${}^i\dot{x}_i$ and ${}^i\dot{x}_{i+1}$ will both be set to zero, and ${}^i\ddot{x}$ is constant, motion across the interval would not be possible.

self-motion solution, then linear interpolation across the discontinuity will result in a motion that wanders off the path. If such non-linear self-motions are anticipated, and path deviations are unacceptable, then the algorithm should be modified to either abort on discontinuities, or perform an appropriate non-linear interpolation that tracks the self-motion solution.

6 Velocity Assignment

After the path has been subdivided, the algorithm computes appropriate values for the driving coordinate velocity ${}^i\dot{x}_i$ at each knot. This is done so as to produce a near-minimum-time motion that adheres closely to the individual velocity and acceleration limits for each coordinate (3). The resulting timing satisfies each $|\dot{q}_j| \leq V_j$ exactly at each knot and each $|\ddot{q}_j| \leq A_j$ for an approximate average of $\ddot{q}_j$ between knots. As a result, the velocity and acceleration constraints are closely satisfied over the entire path. As mentioned in Section 3, these constraints apply to extended coordinates and so also include limits on $\dot{s}$ and $\ddot{s}$.

Rather than determining ${}^i\dot{x}$ directly, it is easier to compute the *coordinate energy* ${}^ie$, defined by ${}^ie \equiv {}^1\!/_2\, {}^i\dot{x}^2$. This is because, as discussed below, the acceleration constraints $|\ddot{q}_j| \leq A_j$ give rise to linear constraints on ${}^ie$. For a given value of ${}^ie$, the corresponding ${}^i\dot{x}$ can be recovered using

$$ {}^i\dot{x} = \operatorname{sgn}({}^i\Delta x)\sqrt{2\,{}^ie}, \tag{26} $$

which follows from the monotonicity of ${}^ix$ with respect to s (Section 5.4). It should be noted that ${}^ie$ is a purely *mathematical* energy, not a physical one.

The coordinate energy values at the left and right endpoints of interval i are given by

$$ {}^ie_i \equiv {}^1\!/_2\, {}^i\dot{x}_i^2 \quad \text{and} \quad {}^ie_{i+1} \equiv {}^1\!/_2\, {}^i\dot{x}_{i+1}^2. $$

Of course, since ${}^{i-1}\dot{x}_i = {}^i\dot{x}_i\, C_i$ (equation (12)), determining ${}^ie_i$ is equivalent to determining ${}^{i-1}e_i$, and it is easy to see that

$$ {}^{i-1}e_i = {}^ie_i\, C_i^2 \quad \text{and} \quad {}^ie_i = {}^{i-1}e_i / C_i^2. \tag{27} $$

Now, in computing coordinate energy values, the basic idea is to make each ${}^ie_i$ large (corresponding to a rapid motion) while maintaining velocity continuity and observing the velocity and acceleration constraints.

First, recall from Section 5.6 that if i is a corner point, then maintaining velocity continuity requires that ${}^i\dot{x}_i = 0$ and therefore ${}^ie_i$ must be set to 0.

Second, consider each coordinate's velocity constraint $|\dot{q}_j| \leq V_j$. To satisfy this exactly at each knot i requires

$$ {}^i e_i \leq B_i, \quad \text{where} \quad B_i \equiv \frac{1}{2} \min_j \left(\frac{V_j}{{}^i q'_{i,j}} \right)^2 . \tag{28} $$

This will also lead to the velocity constraint being closely satisfied *within* the interval, as discussed in Section 5.1.

Third, for each coordinate's acceleration constraint $|\ddot{q}_j| \leq A_j$, we use an approximate average of $\ddot{q}_j$ within interval i. To simplify notation, let $x \equiv {}^i x$, and observe from equation (9) that

$$ \ddot{x} = \frac{{}^i e_{i+1} - {}^i e_i}{{}^i \Delta x}. \tag{29} $$

Now we assume that within the interval, $q'_j(x)$ and $q''_j(x)$ are both approximately equal to their *average* values given by equations (7), so that from the chain rule we get

$$ \ddot{q}_j \approx {}^i \overline{q}'_j \, \ddot{x} + {}^i \overline{q}''_j \, \dot{x}^2. \tag{30} $$

Because our timing computations employ a constant value for $\ddot{x}$ across the interval, it is easy to show that the corresponding average value of $\dot{x}^2$ is given by

$$ \frac{{}^i \dot{x}_i^2 + {}^i \dot{x}_{i+1}^2}{2} = {}^i e_i + {}^i e_{i+1}. $$

Using this to replace $\dot{x}^2$ in (30), and replacing $\ddot{x}$ with (29), turns the constraint $|\ddot{q}_j| \leq A_j$ into

$$ -A_j \leq \left({}^i \overline{q}''_j - \frac{{}^i \overline{q}'_j}{{}^i \Delta x} \right) {}^i e_i + \left({}^i \overline{q}''_j + \frac{{}^i \overline{q}'_j}{{}^i \Delta x} \right) {}^i e_{i+1} \leq A_j. $$

This means that the pair $({}^i e_i, {}^i e_{i+1})$ must lie between two parallel lines in the ${}^i e_i$-${}^i e_{i+1}$ plane.

The intersection of all such acceleration constraints for each coordinate j, combined with the non-negativity of ${}^i e_i$ and ${}^i e_{i+1}$ and the constraints ${}^i e_i \leq B_i$ and ${}^i e_{i+1} \leq B_{i+1} C_{i+1}^2$ implied by (28) and (27), forms a convex polygonal region $\mathcal{E}_i$ in the first quadrant of the ${}^i e_i$-${}^i e_{i+1}$ plane (Figure 5). By choosing the drive coordinate velocities for knots i and $i+1$ so that the corresponding values of ${}^i e_i$

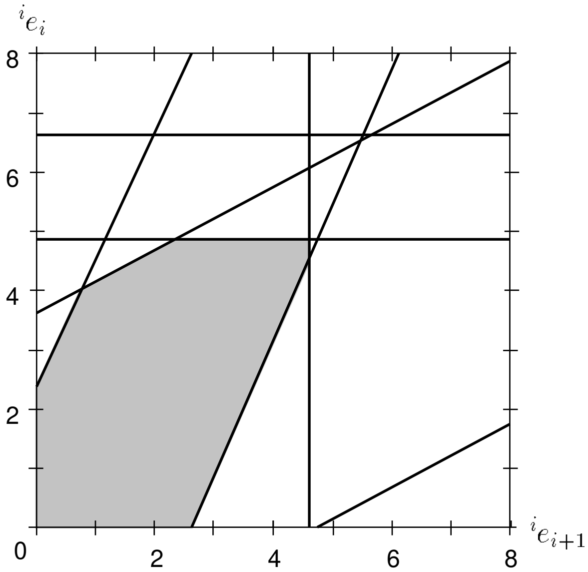

Figure 5: Region $\mathcal{E}_i$ in the ${}^ie_i$-${}^ie_{i+1}$ plane (gray) formed by the intersection of velocity and acceleration constraints.

and ${}^ie_{i+1}$ lie within this region, we ensure satisfaction of each coordinate's velocity constraint $|\dot{q}_j| \leq V_j$ exactly at i and $i+1$, and each coordinate's acceleration constraint $|\ddot{q}_j| \leq A_j$ for the approximation of $\ddot{q}_j$ given by (30).

Given the constraints imposed by the region $\mathcal{E}_i$, we now consider how to assign ${}^ie_i$ and ${}^ie_{i+1}$ so as to generate a near-minimum-time motion. Let ${}^i\mathbf{e} \equiv ({}^ie_i, {}^ie_{i+1})$. A value of ${}^i\mathbf{e}$ that is associated with a minimum-time motion should be associated with a minimal transit time across the interval. Since the interval transit time is inversely proportional to ${}^i\dot{x}_i + {}^i\dot{x}_{i+1}$, it can be minimized by maximizing $F \equiv \sqrt{{}^ie_i} + \sqrt{{}^ie_{i+1}}$. Ideally, then, $({}^ie_i, {}^ie_{i+1})$ should be as close as possible to the point ${}^i\mathbf{e}^* \equiv ({}^ie_i^*, {}^ie_{i+1}^*)$ within $\mathcal{E}_i$ that maximizes F. Because F has only one global minimum at the origin, ${}^i\mathbf{e}^*$ must lie on the boundary of $\mathcal{E}_i$, and so can be found by examining the edges of $\mathcal{E}_i$.

At first glance, it might appear that we can simply set each ${}^ie_i$ to ${}^ie_i^*$. Unfortunately, this won't work, because ${}^ie_i$ also corresponds to the element ${}^{i-1}e_i$ in the pair ${}^{i-1}\mathbf{e}$ associated with the *previous* interval, and so setting ${}^ie_i := {}^ie_i^*$ may conflict with the requirement ${}^{i-1}\mathbf{e} \in \mathcal{E}_{i-1}$. Instead, energies must be assigned using the following three step procedure:

1. *Initialization:* Each ${}^ie_i$ is set to the *minimum* of ${}^ie_i^*$ and the value corresponding to ${}^{i-1}e_i^*$ (note that this may imply ${}^i\mathbf{e} \notin \mathcal{E}_i$). Also, ${}^ie_i$ is set to 0 at every corner (to prevent velocity discontinuities).

2. *Forward Pass:* Each knot is examined in increasing order. If ${}^i\mathbf{e} \notin \mathcal{E}_i$, and this can be corrected by lowering the value of ${}^ie_{i+1}$, we do so.

3. *Reverse Pass:* Each knot is examined again in decreasing order. If ${}^i\mathbf{e} \notin \mathcal{E}_i$, and this can be corrected by lowering the value of ${}^ie_i$, we do so.

A schematic illustration of the procedure is shown in Figure 6. In effect, it behaves like a discrete version of the standard time-optimal path-following algorithms (Bobrow *et al.*, 1985; Shin & McKay, 1985; Slotine & Yang, 1989). That it produces an admissible result is shown by the following theorem:

Theorem 1. *Every coordinate energy pair* ${}^i\mathbf{e}$ *produced by the above three-step procedure satisfies* ${}^i\mathbf{e} \in \mathcal{E}_i$.

This is proven in Appendix F. A continuous-case version of the same procedure, which is less practical to implement but handles all constraints exactly and is provably minimum-time, is described in (Lloyd, 1995).

The complete velocity assignment procedure is summarized by Algorithm 2. In the actual implementation, we need to consider that intervals i and $i+1$ may have different driving coordinates, meaning that ${}^{i+1}e_{i+1}$ must be converted to ${}^ie_{i+1}$, and vice versa, using ${}^ie_{i+1} = {}^{i+1}e_{i+1}C_{i+1}^2$ (from equation (27)).

7 Discussion

7.1 Complexity

For this analysis, since $\mathbf{q}(s)$ is considered an algorithm input, we start by ignoring inverse kinematic costs and assume that $\mathbf{q}(s)$ can be determined at any s with a complexity proportional to the number of joints M. It can then be shown that each of the tests used in *createKnots()* has a complexity proportional to M. Since the knots themselves are created by recursive bisection subject to these tests, the overall complexity of *createKnots()* is therefore $O(KM)$, where K is the number of knots.

For the procedure *assignVelocities()*, this begins by creating a region $\mathcal{E}_i$ at each knot. Each $\mathcal{E}_i$ has $O(M)$ edges, corresponding to $M+1$ velocity and acceleration constraints, and so can be constructed in $O(M \log M)$ time (Preparata & Shamos, 1985). The initialization of each ${}^ie_i$ then requires computing ${}^i\mathbf{e}^*$, which can be done

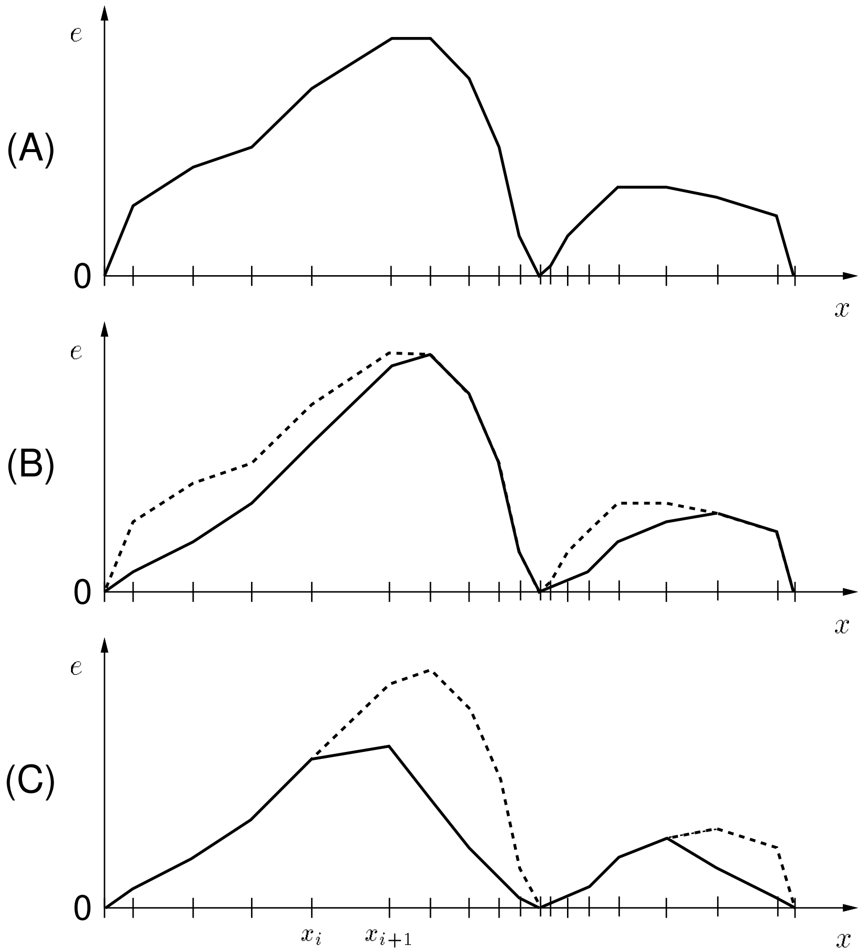

Figure 6: Energy assignment over a set of intervals with an identical driving coordinate $x \equiv {}^i x$. (A) Initial energy assignments are made, resulting in a piecewise-linear energy function $e(x)$. (B) Moving from left to right, the forward pass (solid line) reduces ${}^i e_{i+1}$ whenever this will correct ${}^i\mathbf{e} \notin \mathcal{E}_i$. The effect is to "clip" the slope of $e(x)$ in places where $e'(x)$ is "too positive". (C) Moving from right to left, the reverse pass (solid line) reduces ${}^i e_i$ whenever this will correct ${}^i\mathbf{e} \notin \mathcal{E}_i$. This "clips" the slope of $e(x)$ in places where $e'(x)$ is "too negative".

proc *assignVelocities*(K) $\equiv$

```
for i := 1 to K do                          // INITIALIZATION
    if i ∉ C then                           // knot is not a corner:
        ^i e_i := min(^{i-1}e*_i / C_i^2, ^i e*_i)   // regular initialization
    else
        ^i e_i := 0                         // zero velocity at corner
    fi
od
for i := 1 to K − 1                         // FORWARD PASS
    y := max{λ : (^i e_i, λ) ∈ E_i}
    if y < ^i e_{i+1}                       // can ensure ^i e ∈ E_i ...
        ^i e_{i+1} := y                     // ... by lowering ^i e_{i+1}
    fi
od
for i := K − 1 to 1 do                      // REVERSE PASS
    y := max{λ : (λ, ^i e_{i+1}) ∈ E_i}
    if y < ^i e_i                           // can ensure ^i e ∈ E_i ...
        ^i e_i := y                         // ... by lowering ^i e_i
    fi
od.
```

Algorithm 2: Complete velocity assignment procedure, using coordinate energies. K is the number of knots, and $\mathcal{C}$ is the set of corner points. Knots 1 and K are treated as corners. As there is no interval K, we simply define ${}^{K}e_K \equiv {}^{K-1}e_K$ and $C_K \equiv 1$. Note that ${}^{i}e_{i+1}$ is actually stored as ${}^{i+1}e_{i+1}$, so to obtain ${}^{i}e_{i+1}$, we need to compute ${}^{i+1}e_{i+1}C_{i+1}^2$, and when assigning y to ${}^{i}e_{i+1}$, we need to set ${}^{i+1}e_{i+1} := y/C_{i+1}^2$.

by inspecting each edge of $\mathcal{E}_i$ and so takes $O(M)$ time. Lastly, the computation of y in the forward and reverse passes of *assignVelocities()* is equivalent to intersecting $\mathcal{E}_i$ with a line segment and so also takes $O(M)$ time, implying that *assignVelocities()* has an overall complexity of $O(KM \log M)$.

The total algorithm complexity is thus $O(KM \log M)$. If we also consider inverse kinematic costs, and these have a complexity $C(M)$ which is greater than $M \log M$, then the total algorithm complexity becomes $O(KC(M))$.

7.2 Integration

The actual path timing is produced by integrating the coordinate energy profile produced by the algorithm. Because a constant drive coordinate acceleration is employed within each interval, the energy profile is piecewise-linear, and so this integration is quite easy to perform. Suppose we begin within interval i, at some initial value ${}^i x_b$ of the driving coordinate ${}^i x$, and we wish to determine where along the path we will be after some additional time T. The acceleration ${}^i \ddot{x}$ is given by equation (29), and so the initial energy ${}^i e_b$ at ${}^i x_b$ is given by

$$ {}^i e_b = {}^i e_{i+1} - {}^i \ddot{x} \, ({}^i x_{i+1} - {}^i x_b). $$

By using equation (26) to recover velocities, we can compute the time Δt remaining before reaching the end of the interval (at knot $i + 1$):

$$ \Delta t = \frac{2 \, |{}^i x_{i+1} - {}^i x_b|}{\sqrt{2 \; {}^i e_{i+1}} + \sqrt{2 \; {}^i e_b}}. $$

If $\Delta t \leq T$, then the final position is still within interval i, with a value of ${}^i x$ given by

$$ {}^i x = {}^i x_b + {}^i \dot{x}_b T + {}^1\!/_2 \, {}^i \ddot{x} \, T^2. $$

Otherwise, if $\Delta t > T$, we set $T := T - \Delta t$ and ${}^i x_b := {}^{i+1} x_{i+1}$ and repeat the calculation on interval $i + 1$.

7.3 Determining kinematic solutions

For input, the algorithm requires an inverse kinematic solution $\mathbf{q}(s)$ for the path $\mathbf{X}(s)$. This is particularly easy to produce when the manipulator is non-redundant and has a closed-form kinematic solution. Moreover, such closed-form solutions can sometimes be extended to provide artificial "best-possible" solutions when the path leaves the manipulator workspace (as in the example of Figure 7).

If a closed-form solution is not available, $\mathbf{q}(s)$ can still be determined by iteratively solving equation (1). While this requires special treatment at singularities, the problem is much easier than computing an admissible trajectory, because there is no need to be concerned about acceleration limits. Instead, it suffices to simply adjust the step size along the path so as to keep the resulting incremental changes in $\mathbf{q}$ sufficiently small, as per the methods described in (Chiacchio & Chiaverini, 1995). It would be reasonable to combine such an iterative solution procedure with the knot selection process described in Section 5, though we have not yet done this.

8 Experiments

The algorithm has been implemented and tested using a wide range of experiments involving PUMA and planar 2R manipulators. Three experiments with the PUMA are shown here.

Each example involves a spatial path that encounters a singularity at one or more points. "Stick figure" animation is used to illustrate the corresponding robot motion. The path parameter s is equal to the translational distance along each path, and nominal motions are undertaken with a maximum translational velocity and acceleration of $V_s = 400$ mm/s and $A_s = 2500$ mm/s^2. Two sets of velocity profiles $\dot{q}_j(t)$ are plotted for each example, the first being "raw" velocities showing the effects of the singularities when the path is followed at constant speed, and the second "controlled" velocities showing the results of the path timing produced by our algorithm. To help judge the algorithm's performance, these plots are augmented with a horizontal dotted line indicating V_j for each coordinate, along with a diagonal dotted line indicating the maximum slope corresponding to A_j. The controlled velocity plots also show $\dot{s}(t)$ (which equals the translational velocity) to show how the path speed is reduced near singularities.

For all examples it should be noted that the controlled velocity profiles will be "stretched out" relative to the raw profiles, because the path timing produced by the algorithm slows the manipulator down. This means that features in the controlled profiles will generally be delayed relative to their counterparts in the raw profiles. To help the reader match appropriate features, those portions of each profile which are associated with a singularity are shaded gray.

In the first example (Figure 7), the PUMA follows a parabolic path which leaves and then re-enters the workspace. Where the nominal path lies outside the workspace, the manipulator tracks it as closely as possible by following its projection onto the workspace boundary (our ability to compute this projection relies on the special geometry of the PUMA and may not be feasible for other manipulators). Tracking the path at constant speed results in large spikes in $\dot{q}_2(t)$ and $\dot{q}_3(t)$ around the "elbow" singularities encountered at the workspace boundary. Application of the algorithm, with $V_j = 100°/\text{s}$ and $A_j = 350°/\text{s}^2$ for each robot joint, results in a timing where these spikes are replaced with properly conditioned velocity profiles.

The second example (Figure 8) illustrates a situation, described in Section 3.1, where joint velocities do not have to be brought to rest at the singularity, even though motion along the path must stop. The PUMA is driven along a straight-line path into the outer workspace boundary with the elbow "up", and then pulled back along the same path with the elbow "down". If done at constant speed this results in very large spikes in $\dot{q}_2(t)$ and $\dot{q}_3(t)$. These are replaced with well-behaved velocity profiles by applying the algorithm with $V_j = 150°/\text{s}$ and $A_j = 500°/\text{s}^2$ for each robot joint. Because $\dot{q}_2$ and $\dot{q}_3$ do not change sign, they do not have to be brought to rest.

In the third example (Figure 9) the PUMA is driven along a parabolic trajectory that brings it close to its "ready" position. This involves a triple singularity, encompassing the "elbow", "shoulder", and "wrist" singularities, as evidenced by a discontinuity in $\dot{q}_1(t)$, a large spike in $\dot{q}_3(t)$, and even larger spikes in $\dot{q}_4(t)$ (other joints are also affected but not shown). The spikes in $\dot{q}_4$ correspond to a pair of discontinuities in $q_4(s)$ produced by the self-motion singularity of the wrist. Application of the algorithm, with $V_j = 180°/\text{s}$ and $A_j = 500°/\text{s}^2$ for each robot joint, resolves all the velocity profiles satisfactorily.

	Knots	Maximum Error		Compute time (ms)		
Figure	(K)	mm	degrees	*createKnots*	*assignVelocities*	total
7	131	1.3×10^{-4}	6×10^{-19}	19.8	9.2	29.0
8	68	2.3×10^{-5}	5×10^{-18}	8.9	4.9	13.8
9	80	7.8×10^{-3}	3×10^{-9}	24.6	12.8	37.4

Table 2: Computation statistics for the examples shown.

All examples used the path error tolerances $\epsilon_p = 0.01$ mm and $\epsilon_r = 0.1°$, and the knot-velocity profiles were integrated using a sample time of 50 ms. The number of knot points, maximum translational and rotational path errors, and required computation times are shown in Table 2. The path errors describe the maximum deviation of the realized path from the desired path $\mathbf{X}(s)$ (or its projection onto the workspace boundary, in the first example). Computations were done on a 350 MHz Pentium II workstation. All code is written in C++ and speed improvements are still possible.

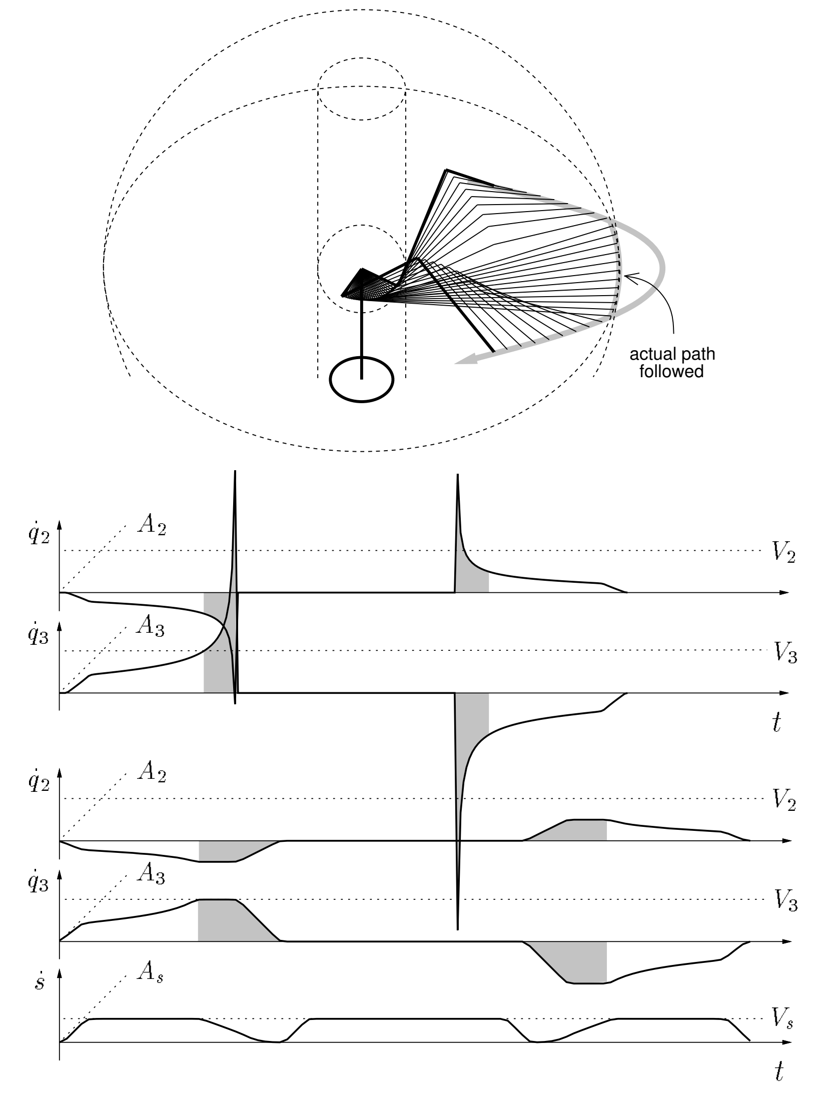

Figure 7: Parabolic path, in a horizontal plane centered at the shoulder, which leaves and then re-enters the workspace. Actual path follows the workspace boundary when necessary. Upper plots show $\dot{q}_2(t)$ and $\dot{q}_3(t)$ when this motion is done at constant speed, while lower plots show the (stretched out) velocities produced by algorithm time scaling. Gray shading shows those parts of each profile which corresponds to a singularity. Horizontal dotted lines show the velocity limits V_j and the diagonal dotted lines show the maximum slopes corresponding to A_j. The plot of $\dot{s}$ illustrates the slowing-down near the singularities.

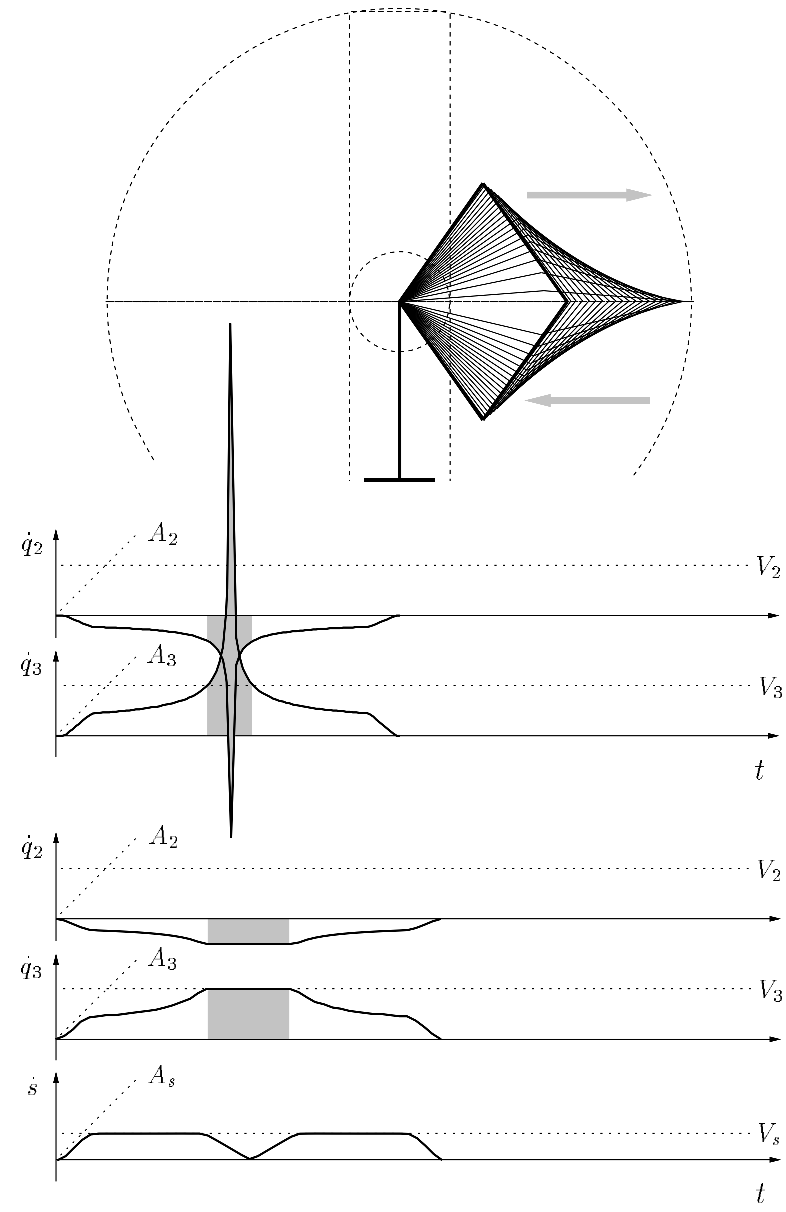

Figure 8: Motion out to and back from the PUMA workspace boundary. Upper plots show $\dot{q}_2(t)$ and $\dot{q}_3(t)$ when this motion is done at constant speed. Lower plots show results after algorithm time scaling has been applied. By changing configurations at the singularity, it is possible to maintain a finite joint speed while transiting the singularity, even though the speed along the path (indicated here by $\dot{s}$) is brought to 0.

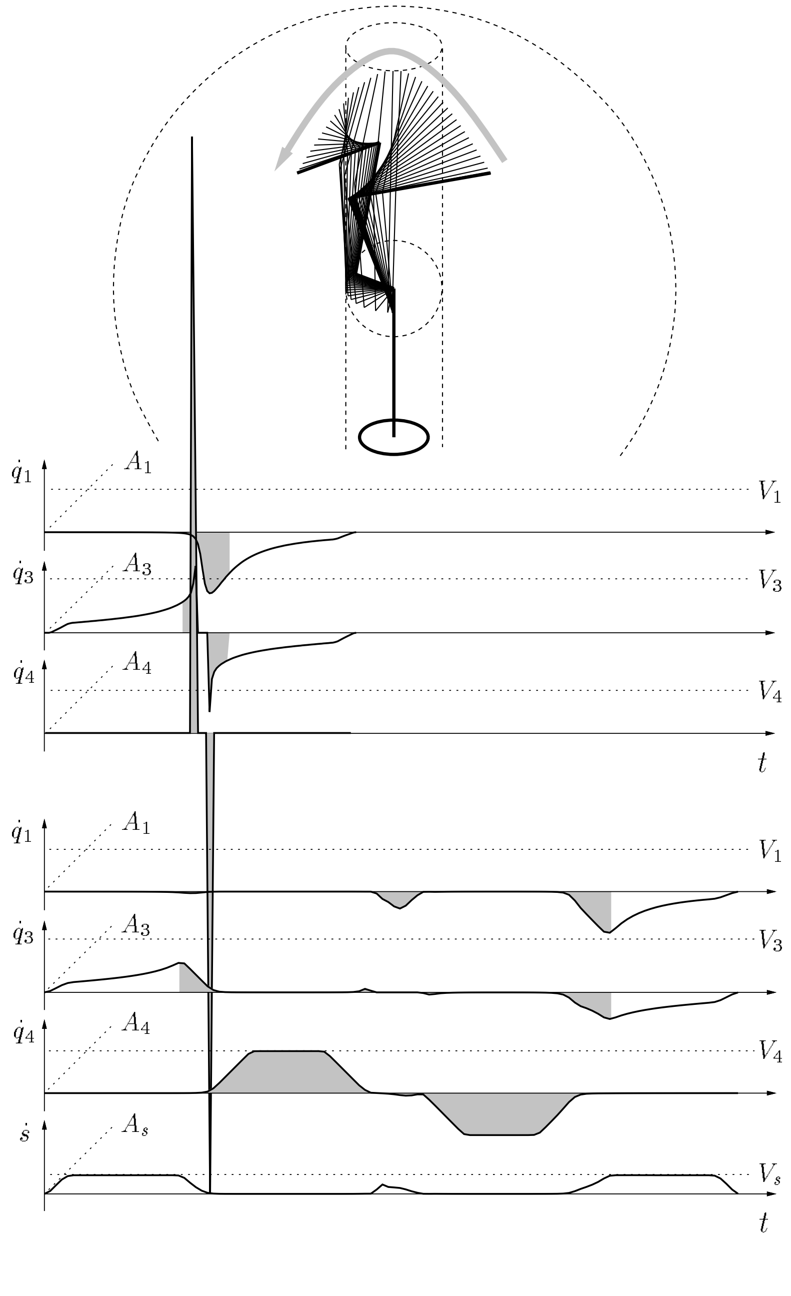

Figure 9: A parabolic motion taking the PUMA to the triple singularity near the "ready" position. Upper plots show $\dot{q}_1(t)$, $\dot{q}_3(t)$, and $\dot{q}_4(t)$ when this motion is done at constant speed. The extremely large spikes in $\dot{q}_4$ are due to the self-motion at the wrist singularity. Lower plots show results after algorithm time scaling has been applied.

These results, typical of other tests, indicate that the algorithm does in fact produce a timing which closely follows the constraints $|\dot{q}_j| \le V_j$ and $|\ddot{q}_j| \le A_j$, while also being near to minimum time. The latter statement can be verified by noting that in the resulting velocity profiles, one or more coordinates is always close to saturation with respect to either its velocity or acceleration constraint.

9 Conclusion

A singularity-robust trajectory generation algorithm has been presented which takes as input a specific path $\mathbf{X}(s)$ and a corresponding kinematic solution $\mathbf{q}(s)$, and produces a path timing (trajectory) which is close to minimum-time, subject to explicit individual constraints on velocity and acceleration for each joint. All types of singularities are handled except for those involving non-linear self-motion. The algorithm is computationally efficient, and has been verified through both analysis and experiment. Because no particular assumptions are made about how $\mathbf{q}(s)$ is produced, it should be applicable to a wide range of manipulator and mechanism types.

We have introduced the idea of coordinate pivoting, in which the fastest changing coordinate x is used locally to control the path timing. All other coordinates then have well-behaved derivatives with respect to x, which facilitates computation and analysis.

The algorithm works by inserting knots along the path, using recursive bisection, until the intervals between successive knots meet certain required conditions, generally resulting in higher knot densities near singularities. Appropriate drive coordinate velocities ${}^i\dot{x}_i$ are then determined for each knot, by computing the corresponding coordinate energies ${}^ie_i$. This is done using a three-step procedure, which tries to maximize the travel time associated with adjacent knot energies ${}^ie_i$ and ${}^ie_{i+1}$, while making sure that they remain within a convex polygonal region $\mathcal{E}_i$ so as to respect velocity and acceleration constraints. The resulting knot-velocity sequence can then be integrated to yield the path timing.

From a practical viewpoint, we believe that the problem of singularity-robust trajectory generation along fixed paths is now close to being solved. Further study of computational issues, including accuracies and tolerances, would be useful, and the algorithm should be extended to handle non-linear self-motions. Also, at present the computation of $\mathbf{q}(s)$ is not an integral part of the algorithm; it would be reasonable to change this so as to integrate Jacobian-based computation of $\mathbf{q}(s)$ with the knot selection process. Similarly, the algorithm currently produces a near minimum-time trajectory for a *fixed* input $\mathbf{q}(s)$; no attempt is made to improve the timing by modifying $\mathbf{q}(s)$, such as by changing branch selections at singularities, although such a feature could certainly be added. Modifications to produce smoother trajectories (for which the accelerations are continuous) may also be desirable.

Our trajectory generator makes arbitrary spatial paths containing singularities as realizable as trapezoidal-velocity trajectories for joint-interpolated motion. By providing a more general definition of $\mathcal{E}_i$, it should also be possible to take into account the manipulator's dynamics, thereby applying this algorithm to the general time-optimal path-following problem.

Acknowledgement

This work was supported by the Institute for Robotics and Intelligent Systems (IRIS) of Canada's Centers of Excellence Program (NCE), and by the Natural Sciences and Engineering Research Council of Canada (NSERC). We wish to thank the reviewer of an earlier paper who made several helpful suggestions and pointed out that non-linear self-motions require special treatment.

References

Aboaf, Eric W., & Paul, Richard P. 1987 (Mar. 31 –Apr. 3). Living with the Singularity of Robot Wrists. *Pages 1713–1717 of: Proceedings of the IEEE International Conference on Robotics and Automation*.

Bobrow, J.E., Dubowsky, S., & Gibson, J.S. 1985. Time-optimal Control of Robotic Manipulators Along Specified Paths. *International Journal of Robotics Research*, **4**(3), 3–17.

Chang, K. S., & Khatib, O. 1995 (Aug.). Manipulator Control at Kinematic Singularities: A Dynamically Consistent Strategy. *Pages 84–88 of: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Chevallereau, C. 1996 (Apr.). Feasible Trajectories for a Non-Redundant Robot at a Singularity. *Pages 1871–1876 of: Proceedings of the IEEE International Conference on Robotics and Automation*.

Chevallereau, C., & Daya, B. 1994 (May 8-13). A New Method for Robot Control in Singular Configurations with Motion in any Cartesian Direction. *Pages 2692–2697 of: Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4.

Chiacchio, Pasquale, & Chiaverini, Stefano. 1995. Coping with Joint Velocity Limits in First-Order Inverse Kinematics Algorithms: Analysis and Real-Time Implementation. *Robotica*, **13**(5), 515–519.

Chiacchio, Pasquale, Chiaverini, Stefano, Sciavicco, Lorenzo, & Siciliano, Bruno. 1991. Closed-Loop Inverse Kinematics Schemes for Constrained Redundant Manipulators with Task Space Augmentation and Task Priority Strategy. *International Journal of Robotics Research*, **10**(4), 410–425.

Chiaverini, Stefano, Siciliano, Bruno, & Egeland, Olav. 1994. Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. *IEEE Transactions on Control Systems Technology*, **2**(2), 123–134.

Deo, A. S., & Walker, I. D. 1993 (May 2-6). Adaptive Non-linear Least Squares for Inverse Kinematics. *Pages 186–193 of: Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1.

Kieffer, Jon. 1994. Differential Analysis of Bifurcations and Isolated Singularities for Robots and Mechanisms. *IEEE Transactions on Robotics and Automation*, **RA-10**(1), 1–10.

Kircanski, Manja V. 1993 (May 2-6). Inverse Kinematics Problem Near Singularities for Simple Manipulators: Symbolical Damped Least-Squares Solution. *Pages 974–979 of: Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1.

Lloyd, John E. 1995 (Jan.). *Robot Trajectory Generation for Paths with Kinematic Singularities*. Ph.D. thesis, Department of Electrical Engineering, McGill University, 3480 University Street, Montreal, Canada, H3A 2A7.

Lloyd, John E. 1996 (Apr.). Using Puiseux Series to Control Non-redundant Robots at Singularities. *Pages 1877–1882 of: Proceedings of the IEEE International Conference on Robotics and Automation*.

Lloyd, John E. 1998. Desingularization of Non-Redundant Serial Manipulator Trajectories Using Puiseux Series. *IEEE Transactions on Robotics and Automation*, **14**(4), 590–600.

Lloyd, John E., & Hayward, Vincent. 1996 (Apr.). A Discrete Algorithm for Fixed-path Trajectory Generation at Kinematic Singularities. *Pages 2743–2748 of: Proceedings of the IEEE International Conference on Robotics and Automation*.

Lloyd, John E., & Hayward, Vincent. 1998 (May). Removing the Singularities of Serial Manipulators by Transforming the Workspace. *In: International Conference on Robotics and Automation*.

Maciejewski, Anthony A., & Klein, Charles A. 1989. The Singular Value Decomposition: Computation and Applications to Robotics. *International Journal of Robotics Research*, **8**(6), 63–79.

Murray, R. W., Li, Z., & Shastry, S. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton.

Nakamura, Yoshihiko, & Hanafusa, Hideo. 1986. Inverse Kinematic Solutions with Singularity Robustness for Robot Manipulator Control. *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, **108**(3), 163–171.

Nenchev, D. N., Tsumaki, Y., Uchiyama, M., Senft, V., & Hirzinger, G. 1996 (Apr.). Two Approaches to Singularity-Consistent Motion of Nonredundant Robotic Mechanisms. *Pages 1883–1890 of: Proceedings of the IEEE International Conference on Robotics and Automation*.

Nielsen, Lars, de Wit, Carlos Canudas, & Hagander, Per. 1990 (Sept. 10-12). Controllability Issues of Robots near Singular Configurations. *Pages 283–290 of: Advances in Robot Kinematics, 2nd International Workshop*.

O'Neil, K. A., Cheng, Y. C., & Seng, J. 1997. Removing Singularities of Resolved Motion Rate Control of Mechanisms, Including Self-Motion. *IEEE Transactions on Robotics and Automation*, **13**(5), 741–751.

Pohl, Eric D., & Lipkin, Harvey. 1991 (June 19-22). A new method of robotic motion control near singularities. *Pages 405–410 of: Fifth International Conference on Advanced Robotics (91 ICAR)*.

Preparata, Franco P., & Shamos, Michael I. 1985. *Computational Geometry. An Introduction*. Springer-Verlag, New York.

Senft, V., & Hirzinger, G. 1995 (May). Redundant Motions of Non-Redundant Robots - A New Approach to Singularity Treatment. *Pages 1553–1558 of: Proceedings of the IEEE International Conference on Robotics and Automation*.

Shin, Kang G., & McKay, Neil D. 1985. Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints. *IEEE Transactions on Automatic Control*, **AC-30**(6), 531–541.

Slotine, Jean-Jacques, & Yang, Hyan S. 1989. Improving the Efficiency of Time-Optimal Path-Following Algorithms. *IEEE Transactions on Robotics and Automation*, **RA-5**(1), 118–124.

Taylor, Russell H. 1979. Planning and Execution of Straight Line Manipulator Trajectories. *IBM Journal of Research and Development*, **23**, 424–436.

Tumeh, Zuheir S., & Alford, Cecil O. 1988 (Apr. 24-29). Solving for Manipulator Joint Rates in Singular Positions. *Pages 987–992 of: Proceedings of the IEEE International Conference on Robotics and Automation*.

Wampler II, C. W., & Leifer, L. J. 1988. Applications of damped least-squares methods to resolved-rate and resolved-acceleration control of manipulators. *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, **110**(31), 31–38.

Wolovich, W. A., & Elliott, H. 1984 (Dec.). A Computational Technique for Inverse Kinematics. *Pages 1359–1363 of: Proceedings of the 23rd IEEE Conference on Decision and Control*.

A Adherence to Velocity Limits

We restrict our analysis to intervals where $q_j({}^i x)$ is strictly monotone. Now consider $\dot{q}_j(t)$ as the interval is traversed. Let $\dot{q}_{cj}(t)$ be the "ideal" trajectory that would result if $\ddot{q}_j$ were constant during the traversal. It turns out that knot creation conditions (b) and (c) imply the difference between $\dot{q}_j(t)$ and $\dot{q}_{cj}(t)$ is bounded:

Lemma 1. *Over an interval on which $q_j({}^i x)$ is strictly monotone,*

$$|\dot{q}_j(t) - \dot{q}_{cj}(t)| \leq \sqrt{E_{aj}|{}^i\Delta q_j|},$$

where E_{aj} is defined by equation (17).

Proof: For notational simplicity, we will suppress the subscript j and let $q \equiv q_j$. Without loss, it will be assumed that $i = 0$, with q_0 and q_1 denoting the values of q at the left and right interval endpoints, and $\Delta q \equiv q_1 - q_0$ denoting the change in q across the interval. The strict monotonicity of $q({}^i x)$ implies that Δq is non-zero and that its sign matches that of $\dot{q}(t)$ throughout the interval. Without loss, assume that $\Delta q > 0$, so that $\dot{q} \geq 0$.

The monotonicity of $q({}^i x)$ also implies that $\dot{q}(t)$ can be parameterized by q instead of by t. Furthermore, by the same derivation as in equation (8), we note that the derivative of $\dot{q}^2(q)$ is $2\ddot{q}$. For $\dot{q}_c(q)$, $\ddot{q}$ is constant and therefore equal to the average acceleration $\overline{\ddot{q}}$ (equation 15), and so

$$\dot{q}_c^2(q) = \dot{q}_0^2 + 2(q - q_0)\overline{\ddot{q}}, \tag{31}$$

where $\dot{q}_0 \equiv \dot{q}(q_0)$. Now, for $\dot{q}^2(q)$ corresponding to any other trajectory, let D be the maximum magnitude of the derivative of $\dot{q}^2(q) - \dot{q}_c^2(q)$. By equation (31) and the definition of E_a (equation (17)), it is clear that $D = \max_q |2\ddot{q} - 2\overline{\ddot{q}}| = 2E_a$. Then from the mean value theorem and the fact that $\dot{q}^2(q) = \dot{q}_c^2(q)$ at q_0 and q_1, it follows that

$$|\dot{q}_c^2(q) - \dot{q}^2(q)| \leq 2E_a(q - q_0)$$

and

$$|\dot{q}_c^2(q) - \dot{q}^2(q)| \leq 2E_a(q_1 - q).$$

Noting that $\Delta q = q_1 - q_0$, we obtain

$$|\dot{q}_c^2(q) - \dot{q}^2(q)| \leq \Delta q E_a. \tag{32}$$

Now, because $\dot{q}_c \geq 0$ and $\dot{q} \geq 0$, $|\dot{q}_c - \dot{q}| \leq |\dot{q}_c + \dot{q}|$, which implies

$$|\dot{q}_c(q) - \dot{q}(q)|^2 \leq |\dot{q}_c(q) - \dot{q}(q)||\dot{q}_c(q) + \dot{q}(q)| = |\dot{q}_c^2(q) - \dot{q}^2(q)|$$

and therefore

$$|\dot{q}_c(q) - \dot{q}(q)| \leq \sqrt{\Delta q E_a}.$$

$\square$

Combining this with inequalities (16) and (20) associated with knot creation conditions (b) and (c), and noting the result is independent of whether $\dot{q}$ is parameterized by q or by t, yields

$$|\dot{q}_c(t) - \dot{q}(t)| \leq V_j/4.$$

Then since $|\dot{q}_c(t)| \leq V_j$ (because $\dot{q}_c(t)$ is linear on the interval and $\dot{q}_j \leq V_j$ is satisfied at the interval end points), it follows that

$$|\dot{q}(t)| \leq {}^5\!/\!_4 V_j.$$

B Proof of Inequality (14)

Let the values of $\dot{q}_j$ at the left and right interval endpoints be denoted by $\dot{q}_{0j}$ and $\dot{q}_{1j}$. From basic kinematics, the average acceleration $\bar{\ddot{q}}_j$ over the interval then satisfies

$$\bar{\ddot{q}}_j = \frac{\dot{q}_{1j}^2 - \dot{q}_{0j}^2}{2\ {}^i\Delta q_j}. \tag{33}$$

Because $q_j({}^i x)$ is assumed to be monotone, and ${}^i x$ is itself monotone with respect to s and hence t, $\dot{q}_j$ must have the same sign as ${}^i\Delta q_j$ throughout the interval. In particular, $\dot{q}_{0j}$ and $\dot{q}_{1j}$ must both have the same sign, and so

$$|\dot{q}_{1j} + \dot{q}_{0j}| \geq |\dot{q}_{1j} - \dot{q}_{0j}|.$$

Since the change in velocity across the interval is given by $\Delta\dot{q}_j = \dot{q}_{1j} - \dot{q}_{0j}$, we then have

$$|\dot{q}_{1j}^2 - \dot{q}_{0j}^2| = |\Delta\dot{q}_j||\dot{q}_{1j} + \dot{q}_{0j}| \geq |\Delta\dot{q}_j|^2.$$

Substituting this into $|\bar{\ddot{q}}_j|$ as given by equation (33) leads directly to inequality (14).

$\square$

C Proof of Inequality (18)

Let $x \equiv {}^i x$ be the driving coordinate for interval i. For notational simplicity, assume without loss that $i = 0$, so that $x_i = x_0$ and $x_{i+1} = x_1$. Also, let $\Delta x \equiv x_1 - x_0$, and assume that $x_0 = 0$, so that $\Delta x = x_1$. Let the value of $q_j'(x)$ at the interval endpoints be denoted by $q_{0j}' \equiv q_j'(x_0)$ and $q_{1j}' \equiv q_j'(x_1)$. Finally, assume that the interval transit begins at time $t = 0$.

Our analysis will rely on the approximation that $q_j(x)$ is quadratic over the interval, implying that $q_j''(x)$ is constant and so can be denoted simply by q_j''. Since $x_0 = 0$, $q_j(x)$ is then given by

$$q_j(x) = q_j(0) + q_{0j}'x + {}^1\!/_2\, q_j'' x^2. \tag{34}$$

Because our timing computations employ a constant value of $\ddot{x}$ within the interval, we have

$$x(t) = \dot{x}_0 t + {}^1\!/_2\, \ddot{x} t^2. \tag{35}$$

If q_j happens to exhibit a constant acceleration $\ddot{q}_j$ across the interval, then this will be equal to the average acceleration $\bar{\ddot{q}}_j$ as defined by equation (15). In general, however, $\ddot{q}_j(t)$ is not constant, and we represent the difference between $\bar{\ddot{q}}_j$ and $\ddot{q}_j(t)$ by

$$\ddot{q}_{Ej}(t) \equiv \bar{\ddot{q}}_j - \ddot{q}_j(t). \tag{36}$$

Expanding $\ddot{q}_j(t)$, using the chain rule in conjunction with equations (34) and (35), gives

$$\begin{aligned} \ddot{q}_j(t) &= q_j'(x)\ddot{x} + q_j''\dot{x}^2 \\ &= [q_{0j}' + q_j''(\dot{x}_0 t + {}^1\!/_2\, \ddot{x}t^2)]\ddot{x} + q_j''(\dot{x}_0 + \ddot{x}t)^2. \end{aligned} \tag{37}$$

Because $q_j(x)$ is assumed quadratic, $q_j'' = (q_{1j}' - q_{0j}')/\Delta x$. Also, applying the chain rule to (15) and letting $\Delta t \equiv {}^i\Delta t$ gives

$$\bar{\ddot{q}}_j = \frac{q_{1j}'\dot{x}_1 - q_{0j}'\dot{x}_0}{\Delta t}.$$

Also, because $\ddot{x}$ is constant, we have

$$\ddot{x} = \frac{\dot{x}_1^2 - \dot{x}_0^2}{2\Delta x} \quad \text{and} \quad \Delta t = \frac{2\Delta x}{\dot{x}_0 + \dot{x}_1}.$$

Combining these with (36) and (37) yields

$$\ddot{q}_{Ej}(t) = \frac{(q'_{1j} - q'_{0j})(\dot{x}_1 - \dot{x}_0)F(t)}{8\Delta x^3}$$

where

$$F(t) \equiv 3(\dot{x}_0 - \dot{x}_1)(\dot{x}_0 + \dot{x}_1)^2 t^2 - 12\Delta x\, \dot{x}_0(\dot{x}_0 + \dot{x}_1)t + 4\Delta x^2(2\dot{x}_0 + \dot{x}_1). \tag{38}$$

Hence $\ddot{q}_{Ej}(t)$ is a quadratic function of t. Examination shows that this function has a single extremal point at $t = t_c = -\dot{x}_0/\ddot{x}$. Now since $\dot{x}(t) = x_0 + \ddot{x}t$, we see that $\dot{x}(t_c) = 0$. But since $\ddot{x}$ is constant and $\dot{x}$ is of uniform sign within the interval (Section 5.4), $\dot{x}$ cannot be 0 inside the interval, hence t_c cannot be inside the interval, and so the maximum and minimum values of $\ddot{q}_{Ej}(t)$ within the interval must occur at the endpoints, corresponding to

$$\begin{aligned} \ddot{q}_{Ej}(0) &= \frac{(q'_{1j} - q'_{0j})(\dot{x}_1 - \dot{x}_0)(2\dot{x}_0 + \dot{x}_1)}{2\Delta x}, \\ \ddot{q}_{Ej}(\Delta t) &= -\frac{(q'_{1j} - q'_{0j})(\dot{x}_1 - \dot{x}_0)(\dot{x}_0 + 2\dot{x}_1)}{2\Delta x}. \end{aligned} \tag{39}$$

Now, because $\dot{x}$ is of uniform sign, we have that $|2\dot{x}_0 + \dot{x}_1|$ and $|\dot{x}_0 + 2\dot{x}_1|$ are both less than $2|\dot{x}_0 + \dot{x}_1|$. In conjunction with the extremal values given by (39), this leads to

$$\begin{aligned} |\ddot{q}_{Ej}(t)| &< \frac{2|q'_{1j} - q'_{0j}||\dot{x}_1^2 - \dot{x}_0^2|}{|2\Delta x|} \\ &< 2|q'_{1j} - q'_{0j}||\ddot{x}| = 2|{}^i\Delta q'_j||\ddot{x}|. \end{aligned}$$

By equation (17), $E_{aj} = \max_t |\ddot{q}_{Ej}(t)|$, and so inequality (18) follows directly. $\square$

D Proof of Monotonicity Tests

We show here that relations (24) and (25) together imply that the cubic Hermite interpolation $s({}^ix)$ across interval i is strictly monotone.

First, note that the bisection process implies that ${}^i\Delta s$ is never 0, and therefore, by equation (23), $\mathrm{sgn}({}^i\overline{s}')$ is unambiguously ± 1. Without loss, assume that $\mathrm{sgn}({}^i\overline{s}') = 1$, so that satisfaction of (24) and (25) implies ${}^is'_i > 0$ and ${}^is'_{i+1} > 0$. For convenience, drop the preceding superscripts i, letting $\Delta x \equiv {}^i\Delta x$, $s'_i \equiv {}^is'_i$,

$s'_{i+1} \equiv {}^is'_{i+1}$, and $\overline{s}' \equiv {}^i\overline{s}'$, and also let $s''_i \equiv s''(x_i)$. Because $s(x)$ is determined by cubic Hermite interpolation, $s'''(x)$ is constant and can be represented simply as s'''. Then $\overline{s}'$ and s'_{i+1} are given by

$$\overline{s}' = \frac{s_{i+1} - s_i}{\Delta x} = s'_i + {}^1\!/_2\, s''_i \Delta x + {}^1\!/_6\, s''' \Delta x^2$$

and

$$s'_{i+1} = s'_i + s''_i \Delta x + {}^1\!/_2\, s''' \Delta x^2.$$

Solving these two equations for s''' and s''_i yields

$$s''' = \frac{6(s'_i + s'_{i+1} - 2\overline{s}')}{\Delta x^2} \tag{40}$$

and

$$s''_i = -\frac{2(2s'_i + s'_{i+1} - 3\overline{s}')}{\Delta x}. \tag{41}$$

Also, from $s''_{i+1} = s''_i + s''' \Delta x$, we obtain

$$s''_{i+1} = \frac{2(s'_i + 2s'_{i+1} - 3\overline{s}')}{\Delta x}. \tag{42}$$

Now, $s(x)$ must be strictly monotone within interval i if $s'(x)$ does not have a zero there. Given that

$$s'(x) = s'_i + s''_i(x - x_i) + {}^1\!/_2\, s'''(x - x_i)^2,$$

$s'(x)$ will have no zeros anywhere if the discriminant of this equation is negative:

$$s''^2_i - 2s''' s'_i < 0.$$

Substituting in equations (40) and (41), and solving for $\overline{s}'$, this becomes

$$\frac{s'_i + s'_{i+1} - \sqrt{s'_i s'_{i+1}}}{3} < \overline{s}' < \frac{s'_i + s'_{i+1} + \sqrt{s'_i s'_{i+1}}}{3}. \tag{43}$$

Now assume that the rightmost inequality of this expression does not hold, so that

$$\overline{s}' \geq \frac{s'_i + s'_{i+1} + \sqrt{s'_i s'_{i+1}}}{3}.$$

By substituting this into equations (41) and (42), we get

$$s_i'' \geq -\frac{2(s_i' - \sqrt{s_i' s_{i+1}'})}{\Delta x}, \quad s_{i+1}'' \leq \frac{2(s_{i+1}' - \sqrt{s_i' s_{i+1}'})}{\Delta x}.$$

Since s_i' and s_{i+1}' are both positive (by assumption), the above inequalities can be shown to imply that s_i'' and s_{i+1}'' cannot have opposite signs. But this means that $s'(x)$ can not have a root within interval i, since s_i' and s_{i+1}' both have the same sign and $s'(x)$ is quadratic. Therefore, we need only consider the leftmost inequality of (43). Now, because s_i' and s_{i+1}' are both positive, the inequalities in (24) and (25) can be expressed as

$$0 < s_i' < 2\overline{s}' \quad \text{and} \quad 0 < s_{i+1}' < 2\overline{s}'.$$

It can then be shown that these constraints are more than sufficient to satisfy the leftmost inequality of (43), and so $s(x)$ is strictly monotone within interval i. $\square$

E Interpolation Procedure

Procedure 2, given below, linearly interpolates across interval i in the event of a discontinuity in one or more $q_j(s)$.

By construction, ${}^i x$ remains the driving coordinate for each newly introduced subinterval, with $\mathbf{q}'({}^i x)$ constant and given by the original value of ${}^i\Delta\mathbf{q}/{}^i\Delta x$. Derivatives at the original interval endpoints i and $i+1$ at normally not changed, unless these points are corners, in which case the right derivative at i and/or the left derivative at $i+1$ is also set to ${}^i\Delta\mathbf{q}/{}^i\Delta x$.

F Proof of Theorem 1

The proof rests on the following observation. Since each interval's admissible energy region $\mathcal{E}_i$ is convex and contains $(0,0)$, if some pair ${}^i\mathbf{e} \equiv ({}^i e_i, {}^i e_{i+1})$ satisfies ${}^i\mathbf{e} \in \mathcal{E}_i$, then $\mu {}^i\mathbf{e} \in \mathcal{E}_i$ for all $\mu \in [0,1]$.

Now refer to the procedure. By construction, it is easy to see that energy values are only lowered, never raised, and that

$${}^i e_i \leq {}^i e_i^* \quad \text{and} \quad {}^i e_{i+1} \leq {}^i e_{i+1}^*,$$

where $({}^i e_i^*, {}^i e_{i+1}^*) = {}^i\mathbf{e}^* \in \mathcal{E}_i$. In particular, when ${}^i\mathbf{e}$ is examined during the forward pass, its initial value will be given by $(\mu_1 {}^i e_i^*, \mu_2 {}^i e_{i+1}^*)$, for $\mu_1 \in [0,1]$ and $\mu_2 \in [0,1]$. If ${}^i\mathbf{e} \notin \mathcal{E}_i$, the forward pass tries to correct this by lowering ${}^i e_{i+1}$. First,

proc *linearInterp*$(i) \equiv$

$n := \max_j \left\lceil \frac{8A_j|^i\Delta q_j|}{V_j^2} \right\rceil$ // number of new knots needed

if $n = 0$ **then** $n := 1$ **fi** // make sure n at least 1

$\Delta\mathbf{q} := {}^i\Delta\mathbf{q}, \Delta x := {}^i\Delta x$ // save for use below

if $i \in \mathcal{C}$ **then** // i is a corner:

${}^i\mathbf{q}'_i := \Delta\mathbf{q}/\Delta x$ // update right derivative at i

fi

if $i + 1 \in \mathcal{C}$ **then** // $i + 1$ is a corner:

${}^i\mathbf{q}'_{i+1} := \Delta\mathbf{q}/\Delta x$ // update left derivative at $i + 1$

fi

$K := K + n,$ reindex knots $> i$

for $k := i + 1$ **to** $i + n$ **do** // create and insert new knots

Add new knot k, with $\mathbf{q}_k := \mathbf{q}_i + \frac{k-i}{n+1}\Delta\mathbf{q}$

${}^k\mathbf{q}'_k := {}^{k-1}\mathbf{q}'_k := \Delta\mathbf{q}/\Delta x$ // set derivatives at k

od.

Procedure 2: Linear interpolation across a discontinuity in $q_j(s)$. The variable n is the number of equally-spaced additional knots needed to ensure satisfaction of Test B, as enforced by inequality (16). Requiring $n \geq 1$ means that at least one new knot will always be added, guaranteeing that motion will still be possible if both original endpoints are corners. K denotes the total number of knots.

suppose that after the forward pass, ${}^i\mathbf{e} \in \mathcal{E}_i$, and let the resulting value of ${}^ie$ be denoted by ${}^i\tilde{\mathbf{e}} \equiv ({}^i\tilde{e}_i, {}^i\tilde{e}_{i+1})$. When interval i is examined again during the reverse pass, ${}^ie_{i+1}$ may be even lower, so that ${}^i\mathbf{e}$ will be given by $({}^i\tilde{e}_i, \mu_3\,{}^i\tilde{e}_{i+1})$ for $\mu_3 \in [0, 1]$. Then since ${}^i\tilde{\mathbf{e}} \in \mathcal{E}_i$, we know that $(\mu_3\,{}^i\tilde{e}_i, \mu_3\,{}^i\tilde{e}_{i+1}) \in \mathcal{E}_i$, and so there exists at least one value to which the reverse pass can lower ${}^ie_i$ (if necessary) in order to ensure that ${}^i\mathbf{e} \in \mathcal{E}_i$.

On the other hand, suppose that after the forward pass ${}^i\mathbf{e} \notin \mathcal{E}_i$. ${}^i\mathbf{e}$ will then be unchanged and still equal to $(\mu_1\,{}^ie^*_i, \mu_2\,{}^ie^*_{i+1})$. Moreover, $\mu_1 > \mu_2$ (since otherwise the forward pass could have ensured ${}^i\mathbf{e} \in \mathcal{E}_i$ by lowering ${}^ie_{i+1}$ to $\mu_1\,{}^ie^*_{i+1}$). When interval i is examined by the reverse pass, ${}^ie_{i+1}$ may be still lower, so that ${}^i\mathbf{e}$ will be given by $(\mu_1\,{}^ie^*_i, \mu_3\,{}^ie^*_{i+1})$ for $\mu_3 \in [0, \mu_2]$. Then since ${}^i\mathbf{e}^* \in \mathcal{E}_i$, we know that $(\mu_3\,{}^ie^*_i, \mu_3\,{}^ie^*_{i+1}) \in \mathcal{E}_i$, and so there again exists at least one value to which the reverse pass can lower ${}^ie_i$ (if necessary) in order to ensure that ${}^i\mathbf{e} \in \mathcal{E}_i$.

$\square$

G Notation

M	Number of manipulator joints.
$\mathbf{J}$	Manipulator Jacobian.
$\mathbf{q}$	Joint coordinates, plus s as coordinate $M+1$.
$\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$	Velocities and accelerations of $\mathbf{q}$.
q_j, $\dot{q}_j$, $\ddot{q}_j$	Individual elements of $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$.
$\mathbf{X}(s)$	Desired spatial path.
$\mathbf{q}(s)$	Joint solution for the desired path.
s	Path parameter.
s_i	Path parameter knot point.
V_j, A_j	Coordinate-specific bounds for $\lvert\dot{q}_j\rvert$ and $\lvert\ddot{q}_j\rvert$.
V_s, A_s	Bounds for $\lvert\dot{s}\rvert$ and $\lvert\ddot{s}\rvert$.
${}^i\Delta\mathbf{q}$, ${}^i\Delta q_j$	Change in $\mathbf{q}$ and q_j across interval i.
${}^ix$	Driving coordinate for interval i.
${}^i\dot{x}$, ${}^i\ddot{x}$	Velocity and acceleration of ${}^ix$.
${}^ix_i$, ${}^ix_{i+1}$	Value of ${}^ix$ at interval endpoints i and $i+1$.
${}^i\dot{x}_i$, ${}^i\dot{x}_{i+1}$	Value of ${}^i\dot{x}$ at interval endpoints i and $i+1$.
$\mathbf{q}_i$	Value of $\mathbf{q}$ at knot i.
${}^i\mathbf{q}'_i$, ${}^i\mathbf{q}'_{i+1}$	Value of $\mathbf{q}'({}^ix)$ at interval endpoints i and $i+1$.
${}^iq'_{i,j}$, ${}^iq'_{i+1,j}$	Value of $q'_j({}^ix)$ at interval endpoints i and $i+1$.
${}^i\Delta q'_j$	Change in $q'_j({}^ix)$ across interval i.
${}^i\Delta x$, ${}^i\Delta s$	Change in ${}^ix$ and s across interval i.
${}^i\overline{\mathbf{q}}'$, ${}^i\overline{\mathbf{q}}''$	Average value of $\mathbf{q}'({}^ix)$ and $\mathbf{q}''({}^ix)$ over interval i.
${}^i\overline{q}'_j$, ${}^i\overline{q}''_j$	Average value of $q'_j({}^ix)$ and $q''_j({}^ix)$ over interval i.
$\overline{\ddot{q}}_j$	Average value of $\ddot{q}_j$ while transiting interval i.
${}^i\Delta t$	Transit time for interval i.
ϵ_p, ϵ_r	Maximum desired translational and rotational path errors.
${}^is'_i$, ${}^is'_{i+1}$	$s'({}^ix_i)$ and $s'({}^ix_{i+1})$.
${}^i\overline{s}'$	Average value of $s'({}^ix)$ within interval i.
$\Delta\dot{q}_j$	Change in $\dot{q}_j$ across interval i.
E_{aj}	Maximum deviation of $\ddot{q}_j$ from $\overline{\ddot{q}}_j$ over interval i.
C_i	Value of ${}^{i-1}x'({}^ix)$ at knot i.
ϵ_s	Tolerance in s for locating corner points.

$\mathcal{C}$	Set of corner points.
K	Total number of knot points.
${}^{i}e$	Coordinate energy $1/2\, {}^{i}\dot{x}^2$.
${}^{i}e_i$, ${}^{i}e_{i+1}$	${}^{i}e$ at interval endpoints i and $i+1$.
B_i	Upper bound on ${}^{i}e_i$ required to meet velocity constraints at knot i.
$\mathcal{E}_i$	Polygonal region formed by constraints on values of ${}^{i}e_i$ and ${}^{i}e_{i+1}$.
${}^{i}\mathbf{e}$	The two-tuple $({}^{i}e_i, {}^{i}e_{i+1})$.
${}^{i}e_i^*$, ${}^{i}e_{i+1}^*$	Values of ${}^{i}e_i$ and ${}^{i}e_{i+1}$ that maximize the transit time of interval i.
${}^{i}\mathbf{e}^*$	The two-tuple $({}^{i}e_i^*, {}^{i}e_{i+1}^*)$.