Proceedings of the 1998 IEEE
International Conference on Robotics & Automation
Leuven, Belgium • May 1998

Generating Robust Trajectories in the Presence of Ordinary and Linear-Self-Motion Singularities

John E. Lloyd
Computer Science Dept., University of British Columbia
Vancouver, B.C., Canada
lloyd@cs.ubc.ca

Vincent Hayward
Center for Intelligent Machines, McGill University
Montréal, P.Q., Canada
hayward@cim.mcgill.ca

Abstract

An algorithm is presented which computes feasible manipulator trajectories along fixed paths in the presence of kinematic singularities. The resulting trajectories are close to minimum time, given an inverse kinematic solution for the path and bounds on joint velocities and accelerations. The algorithm has complexity $O(M \log M)$, with respect to the number of joint coordinates M, and works using "coordinate pivoting", in which the path timing is generated locally with respect to whichever joint coordinate is changing the fastest. This allows the handling of singularities, including linear self-motions (e.g., wrist singularities), where the path speed is zero but other joint velocities are non-zero. Examples involving the PUMA manipulator are shown.

1 Introduction

It is very common for robotic manipulators, when following prescribed spatial paths, to encounter kinematic singularities which cause the joint velocities and accelerations to become unacceptably large. Fixed prescribed paths are a standard component of robot programs and are the typical output of automatic motion planners. We describe in this paper a robust trajectory generator which can take a kinematic solution for any specified path and produce an efficient timing that bounds the joint velocities and accelerations without deviating from the path. All types of singularities are handled except those involving non-linear self-motion (Section 3). An implementation of the algorithm is available at

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

Traditional ways of handling singularities by working with the manipulator Jacobian [1, 2, 3] can cause deviation from the prescribed path and incur sluggish performance near the singularity. However, the existence of path timings which permit motion at singularities without deviating from the path has been shown [4, 5, 6], and some authors have modified Jacobian-based approaches to this end [7, 8].

What the above techniques do not do is generate trajectories with tight bounds on joint acceleration. Indeed, this turns the problem into a variation of the time-optimal path-following planning problem [9, 10]. We believe that the algorithm described here is the first to produce near minimum-time trajectories, given bounds on the joint velocities and accelerations, for any kinematic solution containing ordinary and linear-self-motion singularities. It is an improvement on an earlier algorithm [11], which did not handle self-motions. The present algorithm also has lower complexity ($O(M \log M)$ vs. $O(M^3)$ with respect to the number of robot joints M), allows non-zero joint velocities at singularities, appears to require fewer data points along the path, and is simpler and more robust to program.

2 Overview

For input, the algorithm requires an inverse kinematic solution of the path to be followed. If the path is described by $\mathbf{X}(s)$, where s is a scalar parameter, then the algorithm must have access to the corresponding joint coordinates $\boldsymbol{\vartheta}(s) \equiv (\vartheta_1(s), \ldots, \vartheta_M(s))^T$ at any value of s. This is straightforward when the manipulator has a closed-form kinematic solution. If a closed form solution is not available, $\boldsymbol{\vartheta}(s)$ can still be determined using the manipulator Jacobian, with inverse conditioning and step-size adjustment at singularities (see [12] for details), and presented to the algorithm in interpolated form. Alternatively, it would be better to combine such an iterative solution procedure with the knot selection described in Section 5, though we have not yet done this.

Given $\boldsymbol{\vartheta}(s)$, the algorithm finds an efficient path timing $s(t)$ which keeps the resulting joint velocities $\dot{\boldsymbol{\vartheta}}$ and accelerations $\ddot{\boldsymbol{\vartheta}}$ close to the following individual bounds:

$$|\dot{\vartheta}_j| \leq V_j, \quad |\ddot{\vartheta}_j| \leq A_j. \tag{1}$$

Away from singularities, each $\dot{\vartheta}_j$ and $\ddot{\vartheta}_j$ can be determined from $\vartheta_j(s)$ using the chain rule

$$\dot{\vartheta}_j = \vartheta_j'(s)\,\dot{s} \qquad \text{and} \qquad \ddot{\vartheta}_j = \vartheta_j'(s)\,\ddot{s} + \vartheta_j''(s)\,\dot{s}^2.$$

The problem is that near singularities, $\vartheta_j'(s)$ and $\vartheta_j''(s)$ may become very large, and at self-motion singularities, $\vartheta_j(s)$ may even be discontinuous. To handle this, the algorithm uses *coordinate pivoting* (Section 4), in which some *other* coordinate velocity is used (locally) in place of $\dot{s}$ to control the path timing. Whichever coordinate is locally used to generate the timing is called the *driving coordinate*, and its value is denoted by x.

As with standard minimum-time path-following procedures [13, 14], the timing $s(t)$ is actually produced implicitly by computing path speed as a function of s. This is done in two steps:

1. Knot points s_i are added along the path, dividing it into a sequence of intervals $[s_i, s_{i+1}]$. These knots are added by recursive bisection until each interval is small enough to satisfy criteria needed for good algorithm performance (Section 5).

2. Appropriate driving coordinate velocities $\dot{x}$ are then computed for each knot i (Section 6).

Between knots, $\dot{x}$ is interpolated using a constant value of $\ddot{x}$. The resulting path-velocity profile can be integrated to yield $s(t)$ (see [12]).

3 Singularity 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 are illustrated by the example of Figure 1, which involves a planar 2R manipulator, centered at the origin. Figure 1.A shows all solutions of ϑ_1 for motion along the x axis.

Ordinary singularities occur at $x = \pm 2$, where the x-axis intersects the outer workspace boundary and the two main solution branches meet. For $x < -2$ and $x > 2$, the target position is outside the workspace and so we define the respective "closest possible" solutions $\vartheta_1(x) \equiv \pi$ and $\vartheta_1(x) \equiv 0$, each corresponding to an outstretched arm. At $x = \pm 2$, $\vartheta_1'(x)$ becomes infinite, and so an x-axis motion defined by $x(s) = s, y(s) = 0$, will result in infinite values for $\dot{\vartheta}_1$ and $\ddot{\vartheta}_1$ if $\dot{s} \neq 0$. However, both $\dot{\vartheta}_1$ and $\ddot{\vartheta}_1$ can be bounded if $\dot{s}$ is appropriately brought to 0 at the singularity. Whether $\dot{\vartheta}_1$ *also* needs to be brought to 0 depends on the situation. For example, if the robot is brought to rest at the singularity, then $\dot{\vartheta}_1$ will have to be brought to 0 also. On the other hand, suppose the manipulator is requested to follow the path $x(s) = s$ from $s = 1$ to the singularity at $s = 2$ with the elbow *up* (as in Figure 1.B), and then reverse direction at $s = 2$ and go back along the x-axis with the elbow *down* (as in the right side of Figure 1.C). Then $\dot{\vartheta}_1$ will not change sign as it passes through the singularity and so a timing exists which bounds $\dot{\vartheta}_1$ without bringing it to 0. It is, however, difficult to construct this timing using $\dot{s}$, which is why we employ coordinate pivoting instead.

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

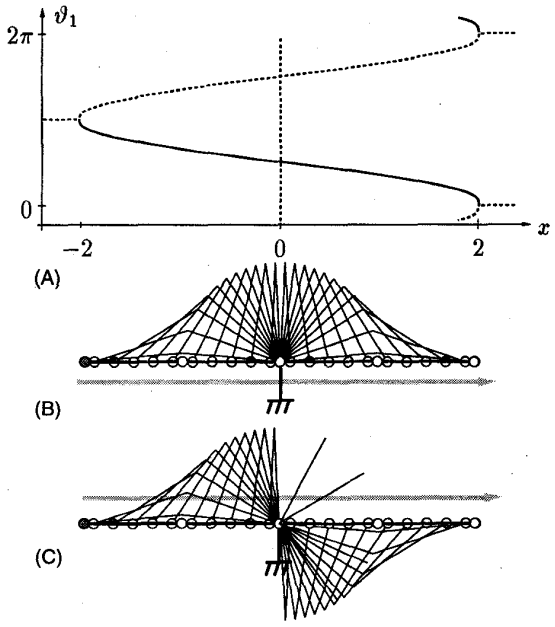

Figure 1: (A): Solutions of ϑ_1 (solid and dotted lines) for a planar 2R robot, with both link lengths equal to 1, following the x axis. (B): motion corresponding to the solid line solution. (C): motion resulting from switching solution branches (solid to dotted) using the self-motion singularity at the origin.

Figure 1.A also shows a linear self-motion singularity at $x = 0$, with the associated self-motion solution indicated by a vertical dotted line along which ϑ_1 can assume any value without changing the path position. It is possible to use this self-motion singularity 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 1.C). The associated $\vartheta_1(s)$ will contain a discontinuity, which can be handled by our algorithm, resulting in the branch-switching motion. While it is usually preferable to avoid switching branches (as in Figure 1.B), task constraints may dictate otherwise. Also, paths which pass *near* self-motion singularities may produce solutions $\vartheta_j(s)$ which numerically *appear* discontinuous.

4 Coordinate Pivoting

As mentioned above, the algorithm uses another coordinate to control the path timing in regions where one or more $\vartheta_j'(s)$ becomes large.

To formalize this idea, extend the set of M joint coordinates to include the path parameter s:

$$\boldsymbol{\vartheta} \equiv (\vartheta_1, \ldots, \vartheta_M, \vartheta_{M+1})^T, \quad \vartheta_{M+1} \equiv s.$$

Now consider the i-th interval defined by $[s_i, s_{i+1}]$. Let

${}^i\Delta\boldsymbol{\vartheta} \equiv \boldsymbol{\vartheta}(s_{i+1}) - \boldsymbol{\vartheta}(s_i)$ denote the change in $\boldsymbol{\vartheta}$ over the interval, and then determine the coordinate for which $|{}^i\Delta\vartheta_j|/\alpha_j$ is a maximum, where α_j is a normalizing factor†. This is called the *driving coordinate* for the interval, and its value will be denoted by the variable x.

Within the i-th interval, all other coordinates are approximated as functions of x, using cubic Hermite interpolation of $\boldsymbol{\vartheta}$ and $\boldsymbol{\vartheta}'(x)$ at the interval endpoints. Interpolation errors are bounded by keeping the knots sufficiently close together (Section 5). Endpoint values of $\vartheta_j'(x)$ can be determined from

$$\vartheta_j'(x) = \vartheta_j'(s)/x'(s). \tag{2}$$

Because x is the coordinate with the largest variation, we can generally expect that $|\vartheta_j'(x)| \leq 1$, provided the interval is small enough. The derivatives used in the right-hand side of (2) are computed numerically by taking finite differences between $\vartheta_j(s)$ and $\vartheta_j(s+\epsilon)$; this ensures that all numbers remain finite even at singularities. Path timing within the interval is specified using $\dot{x}$, with other velocities $\dot{\vartheta}_j$ defined by $\dot{\vartheta}_j = \vartheta_j'(x)\dot{x}$.

With respect to Figure 1.A, consider a motion defined by $x(s) = s$. Then (ignoring ϑ_2 for the sake of this example) ϑ_1 will be the driving coordinate in right and left neighborhoods of $s = -2$ and $s = 2$, respectively, and also at $s = 0$ if the chosen $\vartheta_1(s)$ contains a self-motion discontinuity. The driving coordinate elsewhere will be s.

The process of choosing x, which we call *coordinate pivoting*, is central to our algorithm. It is analogous to pivoting in matrix computations, where one divides a matrix row or column by the element with largest magnitude.

The following notation will be used in the sequel. The driving coordinate associated with the i-th interval will be denoted by ${}^ix$, and its values at i and $i+1$ by ${}^ix_i$ and ${}^ix_{i+1}$. Likewise, ${}^i\dot{x}_i$ and ${}^i\dot{x}_{i+1}$ will describe ${}^i\dot{x}$ at i and $i+1$. The change in $\boldsymbol{\vartheta}$ and $\boldsymbol{\vartheta}'({}^ix)$ over the interval will be denoted by ${}^i\Delta\boldsymbol{\vartheta}$ and ${}^i\Delta\boldsymbol{\vartheta}'$, with ${}^i\Delta x$ specifically denoting the change in ${}^ix$. We will also refer to the *average* values of $\boldsymbol{\vartheta}'({}^ix)$ and $\boldsymbol{\vartheta}''({}^ix)$ over an interval, respectively represented by

$${}^i\bar{\boldsymbol{\vartheta}}' \equiv \frac{\boldsymbol{\vartheta}({}^ix_{i+1}) - \boldsymbol{\vartheta}({}^ix_i)}{{}^i\Delta x} \tag{3}$$

and

$${}^i\bar{\boldsymbol{\vartheta}}'' \equiv \frac{\boldsymbol{\vartheta}'({}^ix_{i+1}) - \boldsymbol{\vartheta}'({}^ix_i)}{{}^i\Delta x}. \tag{4}$$

To ensure continuity of velocities, the driving coordinate velocities must match appropriately at the interval endpoints. Generally, at any knot i, if we define $C_i \equiv {}^{i-1}x'({}^ix)$, then $\boldsymbol{\vartheta}'({}^ix_i)$ will be related to $\boldsymbol{\vartheta}'({}^{i-1}x_i)$ by

$$\boldsymbol{\vartheta}'({}^ix_i) = \boldsymbol{\vartheta}'({}^{i-1}x_i)\, C_i \tag{5}$$

(though this is not true if i is a corner point; Section 5.1). Velocity continuity then requires that

$${}^{i-1}\dot{x}_i = {}^i\dot{x}_i\, C_i. \tag{6}$$

5 Adding Knot Points

The algorithm begins by adding a sufficient number of knots along the path. Starting with an initial set of knots, more knots are added by recursive bisection until each of the following conditions are satisfied:

(a) The path error is within bounds;

(b) Each $|{}^i\Delta\vartheta_j|$ is within a prescribed limit;

(c) Each $|{}^i\Delta\vartheta_j'|$ is within a prescribed limit;

(d) The (interpolated) function $s({}^ix)$ is monotone.

The conditions will only be summarized here; further details can be obtained from [12].

Condition (a) ensures that in interpolating $\boldsymbol{\vartheta}({}^ix)$ 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 [15].

Condition (b) helps ensure that the path solution will be close to minimum-time. Because $\ddot{\vartheta}_j$ is approximately constant across any interval (by condition (c)), then if $|{}^i\Delta\dot{\vartheta}_j|$ is too large, $|\ddot{\vartheta}_j|$ may be unable to reach its maximum A_j value without violating the velocity constraint $|\dot{\vartheta}_j| \leq V_j$. However, within intervals where $\vartheta_j({}^ix)$ is monotone and $\ddot{\vartheta}_j$ is assumed constant, it can be shown that $|\ddot{\vartheta}_j|$ will always reach A_j, for any velocity change whose magnitude exceeds $V_j/2$, if

$$|{}^i\Delta\dot{\vartheta}_j| \leq \frac{V_j^2}{8A_j}. \tag{7}$$

This is the test we usually use to bound each $|{}^i\Delta\dot{\vartheta}_j|$.

Condition (c) limits the local curvature of each $\vartheta_j({}^ix)$, which is necessary because the algorithm assumes that a constant ${}^i\ddot{x}$ within each interval results in a roughly constant value for $\ddot{\vartheta}_j$. If $\ddot{\vartheta}_{Ej}$ denotes the error in $\ddot{\vartheta}_j$ (i.e., the amount it deviates from a constant value), and we assume that $\vartheta_j({}^ix)$ is approximately quadratic over the interval, it is possible to show that $|\ddot{\vartheta}_{Ej}| < A_j/2$ (i.e., 50% of A_j) if

$$|{}^i\Delta\vartheta_j'| < \frac{A_j}{4A_x}, \tag{8}$$

† α_j adjusts for comparisons between coordinates with different units. Good values are 2π for revolute joints and the workspace diameter for prismatic joints. For $\vartheta_{M+1} \equiv s$, α_{M+1} can be set to the range of s.

where A_x is the maximum value for $|{}^i\ddot{x}|$. Since the derivation of (8) is fairly conservative, $|\ddot{\vartheta}_{Ej}|$ will usually be much smaller than $A_j/2$.

Condition (d) ensures that by using ${}^ix$ as a driving coordinate we don't reverse direction along the path as an artifact of interpolating $\vartheta({}^ix)$ within the interval. It also ensures that ${}^i\dot{x}$ is of uniform sign within the interval, an assumption used when assigning knot point velocities (Section 6).

5.1 Discontinuities and corners

The recursive subdivision of intervals to satisfy conditions (b) through (d) may fail to terminate if $\vartheta({}^ix)$ contains discontinuities or corners (i.e., discontinuities in $\vartheta'({}^ix)$). Figure 1 illustrates both: a possible discontinuity at $x = 0$ (depending on the choice of solution branches), and possible corners at $x = \pm 2$, where the main branches meet the artificial solutions $\vartheta_1(s) \equiv \pi$ for $x < -2$ and $\vartheta_1(s) \equiv 0$ for $x > 2$.

If $|{}^i\Delta s|$ falls below a small threshold ϵ_s and conditions (c) or (d) are still not satisfied, then the situation is resolved by declaring knot i and/or knot $i+1$ to be a corner. If knot i is a corner, it implies that $\vartheta'({}^ix_i)$ and $\vartheta'({}^{i-1}x_i)$ are no longer related by (5). Instead, we recompute each separately, based on adjacent curve information, according to

$$\vartheta'({}^ix_i) := 2\,{}^i\bar{\vartheta}' - \vartheta'({}^ix_{i+1})$$
$$\vartheta'({}^{i-1}x_i) := 2\,{}^{i-1}\bar{\vartheta}' - \vartheta'({}^{i-1}x_{i-1}).$$

If $|{}^i\Delta s|$ falls below ϵ_s and condition (b) is still not satisfied, then a discontinuity in $\vartheta({}^ix)$ is assumed. In this case, we linearly interpolate $\vartheta({}^ix)$ across the interval, adding enough extra knots so as to ensure satisfaction of condition (b). By construction, the driving coordinate of each interpolated knot will still be ${}^ix$. Usually, when a discontinuity is detected, the endpoints i and $i+1$ will also turn out to be corners.

This handling of discontinuities is what permits the algorithm to function properly at linear self-motion singularities. If the self-motion solution does *not* form a line in joint space, 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 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, at each knot, a driving coordinate velocity value ${}^i\dot{x}_i$. This is done so as to try and produce minimum-time motion while closely adhering to the constraints of (1).

Rather than working with driving coordinate velocities ${}^i\dot{x}$ 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 $|\ddot{\vartheta}_j| \le A_j$ imply linear constraints on ${}^ie$. Note that ${}^ie$ is a *mathematical* energy, not a physical one. The coordinate energy values corresponding to the interval endpoint velocities ${}^i\dot{x}_i$ and ${}^i\dot{x}_{i+1}$ are defined by ${}^ie_i \equiv {}^1/_2\,{}^i\dot{x}_i^2$ and ${}^ie_{i+1} \equiv {}^1/_2\,{}^i\dot{x}_{i+1}^2$.

To begin, we note that if i is a corner point (Section 5.1), then maintaining velocity continuity requires that ${}^i\dot{x}_i = 0$ and therefore ${}^ie_i$ must be set to 0.

Next, the constraints $|\dot{\vartheta}_j| \le V_j$ can be satisfied exactly at each knot i by requiring that

$${}^ie_k \le {}^iB_k, \quad \text{where} \quad {}^iB_k \equiv \frac{1}{2}\min_j \left(\frac{V_j}{\vartheta'_j({}^ix_k)}\right)^2. \tag{9}$$

Between knots, ${}^i\ddot{x}$ is constant, resulting in an approximately constant $\ddot{\vartheta}_j$ (condition (c), Section 5), and so $\dot{\vartheta}_j$ will stay roughly between its endpoint values and so closely adhere to $|\dot{\vartheta}_j| \le V_j$.

Finally, for the constraints $|\ddot{\vartheta}_j| \le A_j$, we start by observing from basic kinematics that

$${}^i\ddot{x} = \frac{{}^i\dot{x}_{i+1}^2 - {}^i\dot{x}_i^2}{2\,{}^i\Delta x} = \frac{{}^ie_{i+1} - {}^ie_i}{{}^i\Delta x}. \tag{10}$$

Now let $x \equiv {}^ix$. If we assume that $\vartheta_j(x)$ is approximately quadratic over the interval, with average values for $\vartheta'_j(x)$ and $\vartheta''_j(x)$ given by ${}^i\bar{\vartheta}'_j$ and ${}^i\bar{\vartheta}''_j$ (defined by equations (3) and (4)), then by the chain rule

$$\ddot{\vartheta}_j \approx {}^i\bar{\vartheta}'_j\,\ddot{x} + {}^i\bar{\vartheta}''_j\,\dot{x}^2.$$

If $\dot{x}^2$ is replaced with its average value on the interval, which equals ${}^ie_{i+1} + {}^ie_i$, and $\ddot{x}$ is replaced by (10), then the constraint $|\ddot{\vartheta}_j| \le A_j$ becomes

$$-A_j \le \left({}^i\bar{\vartheta}''_j - \frac{{}^i\bar{\vartheta}'_j}{{}^i\Delta x}\right){}^ie_i + \left({}^i\bar{\vartheta}''_j + \frac{{}^i\bar{\vartheta}'_j}{{}^i\Delta x}\right){}^ie_{i+1} \le A_j.$$

This means that the pair $({}^ie_i, {}^ie_{i+1})$ must lie between two parallel lines in the ${}^ie_i$-${}^ie_{i+1}$ plane. The intersection of all such constraints for each j, plus the constraints ${}^ie_i \le {}^iB_i$ and ${}^ie_{i+1} \le {}^iB_{i+1}$ implied by (9) and the fact that ${}^ie_i$ and ${}^ie_{i+1}$ are positive, forms a convex polygonal region $\mathcal{E}_i$ in the first quadrant of the ${}^ie_i$-${}^ie_{i+1}$ plane (Figure 2).

Let ${}^i\mathbf{e} \equiv ({}^ie_i, {}^ie_{i+1})$. In determining energies ${}^ie_i$ for each knot point, we want to ensure that ${}^i\mathbf{e} \in \mathcal{E}_i$, meaning that the constraints (1) are approximately satisfied. At the same time, we want the resulting motion to be close to

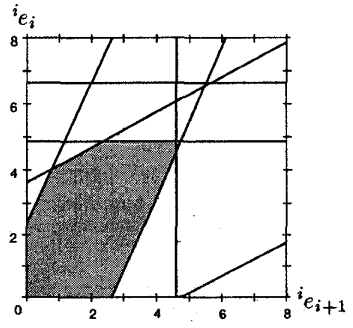

Figure 2: Region $\mathcal{E}_i$ in the $^{i}e_i$-$^{i}e_{i+1}$ plane (shaded gray) formed by the intersection of velocity and acceleration constraints.

minimum time, meaning that $^{i}\mathbf{e}$ should be associated with a minimal interval transit time. Since the interval transit time is inversely proportional to $^{i}\dot{x}_i + {}^{i}\dot{x}_{i+1}$, it can be minimized by maximizing $F \equiv \sqrt{^{i}e_i} + \sqrt{^{i}e_{i+1}}$. Ideally, then, we would like $^{i}\mathbf{e}$ to be close to the point $^{i}\mathbf{e}^* \equiv (^{i}e_i^*, {}^{i}e_{i+1}^*)$ within $\mathcal{E}_i$ that maximizes F. It is easy to show that $^{i}\mathbf{e}^*$ must lie on the boundary of $\mathcal{E}_i$, and so can be found by examining the edges of $\mathcal{E}_i$.

Computing $^{i}e_i$ is actually more complicated that simply assigning it the value $^{i}e_i^*$. That is because $^{i}e_i$ also corresponds to the element $^{i-1}e_i$ in the tuple $^{i-1}\mathbf{e}$ associated with the *previous* interval, and so setting $^{i}e_i := {}^{i}e_i^*$ may conflict with the requirement $^{i-1}\mathbf{e} \in \mathcal{E}_{i-1}$. Consequently, energies are assigned using a three step procedure:

1. *Initialization:* Each $^{i}e_i$ is set to the *minimum* of $^{i}e_i^*$ and the value corresponding to $^{i-1}e_i^*$ (note that this may mean $^{i}\mathbf{e} \notin \mathcal{E}_i$). Also, $^{1}e_1$ and $^{K}e_K$ are set according to initial conditions (typically to 0), and $^{i}e_i$ is set to 0 at every corner point (to prevent velocity discontinuities).

2. *Forward Pass:* A pass is made through the knots, in increasing order, which attempts to place each $^{i}\mathbf{e}$ within $\mathcal{E}_i$ by reducing the value of $^{i}e_{i+1}$.

3. *Reverse Pass:* Another pass is made through the knots, in decreasing order, which ensures that each $^{i}\mathbf{e}$ is within $\mathcal{E}_i$ by reducing the value of $^{i}e_i$.

It is possible to prove (using arguments contained in [12]) that at the conclusion of these steps, $^{i}\mathbf{e} \in \mathcal{E}_i$ for all knots i. The forward pass has the effect of clipping positive accelerations, while the reverse pass clips the negative accelerations.

In the above discussion, we ignored the fact that intervals $i-1$ and i may have different driving coordinates, meaning that, if i is not a corner, $^{i}e_i$ must be converted to $^{i-1}e_i$, and vice versa, using the velocity conversion of equation (6). In particular, it is easy to see that

$$^{i-1}e_i = {}^{i}e_i\, C_i^2 \quad \text{and} \quad {}^{i}e_i = {}^{i-1}e_i / C_i^2. \tag{11}$$

Taking such conversions into account, the complete energy assignment procedure for K knots is summarized below:

proc *assignEnergies*(K) $\equiv$
  for $i := 2$ **to** $K-1$ **do** // Initialize values
    $^{i}e_i := \min(^{i-1}e_i^*/C_i^2, {}^{i}e_i^*)$
  od
  Initialize $^{1}e_1$ and $^{K}e_K$ and set $^{i}e_i := 0$ at corners
  for $i := 1$ **to** $K-1$ // Forward pass
    $y := \max\{\lambda : (^{i}e_i, \lambda) \in \mathcal{E}_i\}$
    if $y < {}^{i+1}e_{i+1}C_{i+1}^2$
      $^{i+1}e_{i+1} := y/C_{i+1}^2$
    fi
  od
  for $i := K-1$ **to** 1 **do** // Reverse pass
    $y := \max\{\lambda : (\lambda, {}^{i+1}e_{i+1}C_{i+1}^2) \in \mathcal{E}_i\}$
    if $y < {}^{i}e_i$
      $^{i}e_i := y$
    fi
  od.

7 Complexity

For this analysis, since $\vartheta(s)$ is an algorithm input, we start by ignoring inverse kinematic costs and assume that $\vartheta(s)$ can be determined at any s with a complexity proportional to the number of joint coordinates M. Then knot creation, which proceeds by recursive bisection using tests with complexity proportional to M, has itself a complexity of $O(KM)$, where K is the number of knots.

Velocity assignment begins by creating a region $\mathcal{E}_i$ at each knot. Each $\mathcal{E}_i$ has $O(M)$ edges, corresponding to M velocity and acceleration constraints, and so can be constructed in $O(M \log M)$ time [16]. The initialization of each $^{i}e_i$ then requires $^{i}\mathbf{e}^*$, which can be computed 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 *assignEnergies()* is equivalent to intersecting $\mathcal{E}_i$ with a line segment and so also takes $O(M)$ time.

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

8 Experiments

The algorithm has been implemented and tested on a wide range of examples for the PUMA and planar 2R robots.

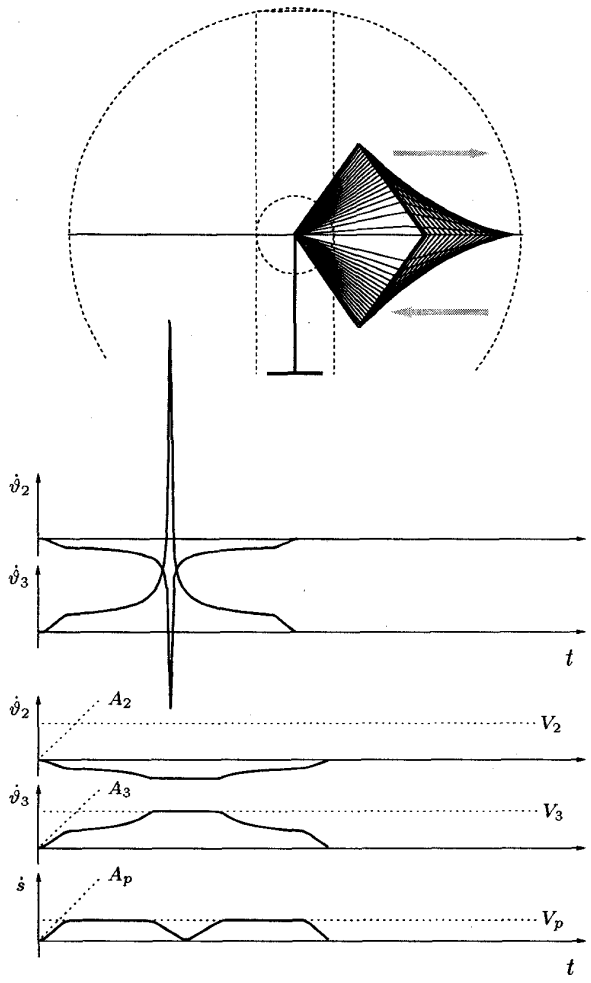

Figure 3: Motion out to and back from the PUMA workspace boundary. Upper plots show $\dot{\vartheta}_2(t)$ and $\dot{\vartheta}_3(t)$ when this motion is done at constant speed. Lower plots show results after algorithm time scaling has been applied. By changing configurations at the singularity, it is 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.

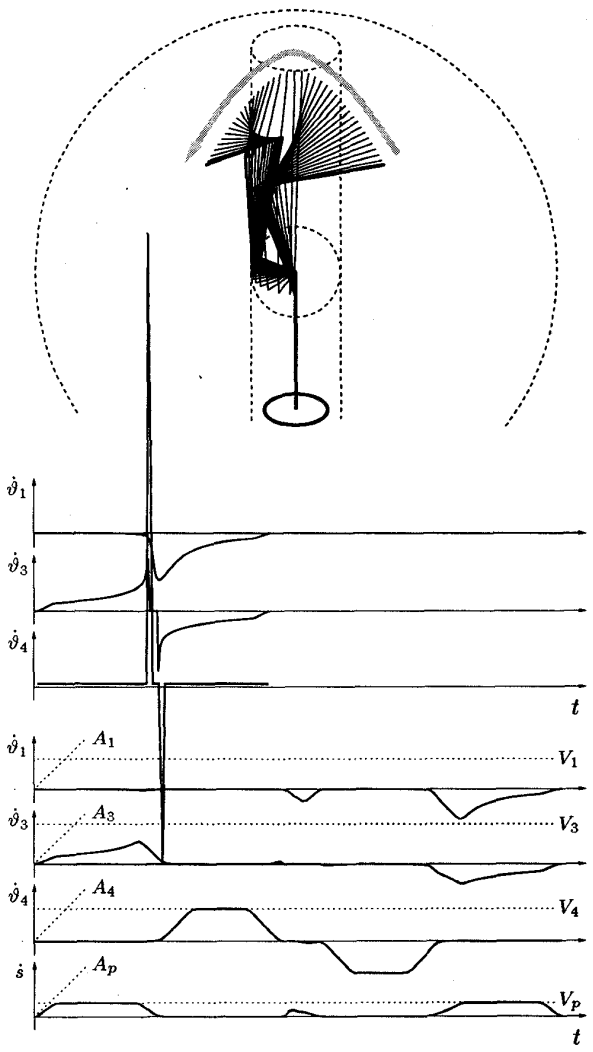

Figure 4: A parabolic motion taking the PUMA to the triple singularity near the ready position. Upper plots show $\dot{\vartheta}_1(t)$, $\dot{\vartheta}_3(t)$, and $\dot{\vartheta}_4(t)$ when this motion is done at constant speed. The extremely large spike in $\dot{\vartheta}_4$ is due to the self-motion at the wrist singularity. Lower plots show results after algorithm time scaling has been applied.

Two examples are shown here, both of which could not be handled by our earlier work in [11].

In the first example (Figure 3), 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 with constant speed this results in very large spikes in $\dot{\vartheta}_2$ and $\dot{\vartheta}_3$. Application of the algorithm, with $V_j = 150°/\text{s}$ and $A_j = 500°/\text{s}^2$ for the robot joints, removes these spikes and resolves the velocity profiles satisfactorily. Because there is no sign change in the velocities, it is possible to bound $\dot{\vartheta}_2$ and $\dot{\vartheta}_3$ without bringing them to rest.

In the second example (Figure 4) the PUMA is driven along a parabolic trajectory that brings it close to the triple-singularity at its "ready" position. Traversing the path at constant speed results in unmanageable profiles for $\dot{\vartheta}_1$, $\dot{\vartheta}_3$ and $\dot{\vartheta}_4$ (as well as other joints, not shown). The extremely large spikes in $\dot{\vartheta}_4$ are due to a pair of discontinuities in $\vartheta_4(s)$ resulting from the self-motion wrist singularity. Application of the algorithm, with $V_j = 180°/\text{s}$ and $A_j = 500°/\text{s}^2$ for the robot joints, resolves all the velocity profiles satisfactorily.

For both examples, s was equivalent to the path's spatial length. The nominal path speed was $\dot{s} = V_s = 400$ mm/s, and the algorithm enforced an acceleration bound of $|\ddot{s}| \leq A_s = 2500$ mm/s^2. The path error tolerance was 0.01 mm. Computations were done on a Sun Ultra-SPARC 2 workstation, capable of about 60 Mflops. The first example required 68 knots and took 19.5 msec to compute. The second example generated 180 knots and took 56.8 msec to compute. All code is written in C++ and significant speed improvements are still possible.

To help gauge performance, the velocity plots associated with the algorithm contain dotted horizontal and diagonal lines indicating V_j and A_j. As well, $\dot{s}$ is plotted to give an indication of how the nominal path speed is reduced near singularities. It will be noted that the constraints (1) are followed quite tightly, and that the trajectories are also close to minimum time, as evidenced by the fact that usually at least one coordinate is close to saturation with respect to either its velocity or acceleration constraint.

9 Conclusion

We believe that, from a practical point of view, the problem of robust trajectory generation along fixed paths containing singularities is now close to being solved. Further study of computational issues, along with accuracies and tolerances, would be useful. The algorithm should also be extended to handle non-linear self-motions. Also, at present we do not consider the computation of $\boldsymbol{\vartheta}(s)$ to be part of the algorithm; this should be changed so as to integrate Jacobian-based computation of $\boldsymbol{\vartheta}(s)$ with the knot selection process.

It should be noted that the algorithm produces a near minimum-time trajectory for a *fixed* input $\boldsymbol{\vartheta}(s)$. No attempt is presently made to improve the timing by modifying $\boldsymbol{\vartheta}(s)$ (such as by changing branch selections at singularities), although such abilities could certainly be introduced.

Our algorithm relies on the idea of coordinate pivoting, in which the coordinate x with the steepest derivative is used locally to control the path timing. All other coordinates then have well-behaved derivatives with respect to x, which makes computation and analysis considerably easier. In particular, it allows us to do path velocity assignment using the convex polygonal region $\mathcal{E}_i$, whereas the equivalent region in our earlier work [11] was formed by intersecting hyperbolas, and therefore much more tedious to work with.

By providing a more general definition of $\mathcal{E}_i$, it should be possible to take into account the manipulator's dynamics, thereby allowing this algorithm to also be applied to the general time-optimal path-following problem.

References

[1] C. W. Wampler II and L. J. Leifer, "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*, vol. 110, pp. 31–38, Mar. 1988.

[2] A. A. Maciejewski and C. A. Klein, "The singular value decomposition: Computation and applications to robotics," *International Journal of Robotics Research*, vol. 8, pp. 63–79, Dec. 1989.

[3] S. Chiaverini, B. Siciliano, and O. Egeland, "Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator," *IEEE Transactions on Control Systems Technology*, vol. 2, pp. 123–134, June 1994.

[4] L. Nielsen, C. C. de Wit, and P. Hagander, "Controllability issues of robots near singular configurations," in *Advances in Robot Kinematics, 2nd International Workshop*, (Linz, Austria), pp. 283–290, Sept.10-12 1990.

[5] J. Kieffer, "Differential analysis of bifurcations and isolated singularities for robots and mechanisms," *IEEE Transactions on Robotics and Automation*, vol. RA-10, pp. 1–10, Feb. 1994.

[6] C. Chevallereau, "Feasible trajectories for a non redundant robot at a singularity," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Minneapolis, Minnesota), pp. 1871–1876, Apr. 1996.

[7] D. N. Nenchev, Y. Tsumaki, M. Uchiyama, V. Senft, and G. Hirzinger, "Two approaches to singularity-consistent motion of nonredundant robotic mechanisms," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Minneapolis, Minnesota), pp. 1883–1890, Apr. 1996.

[8] K. A. O'Neil, Y. C. Cheng, and J. Seng, "Desingularization of resolved motion rate control of mechanisms," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Minneapolis, Minnesota), pp. 3147–3154, Apr. 1996.

[9] K. G. Shin and N. D. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, vol. AC-30, pp. 531–541, June 1985.

[10] J.-J. Slotine and H. S. Yang, "Improving the efficiency of time-optimal path-following algorithms," *IEEE Transactions on Robotics and Automation*, vol. RA-5, pp. 118–124, Feb. 1989.

[11] J. E. Lloyd and V. Hayward, "A discrete algorithm for fixed-path trajectory generation at kinematic singularities," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Minneapolis, Minnesota), pp. 2743–2748, Apr. 1996.

[12] J. E. Lloyd, "Singularity-robust trajectory generation for robotic manipulators," Tech. Rep. 98-02, Department of Computer Science, University of British Columbia, 201-2366 Main Mall, Vancouver, Canada, V6T 1Z4, Mar. 1998. Available from http://www.cs.ubc.ca/spider/lloyd/papers/singrob.ps.Z.

[13] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *International Journal of Robotics Research*, vol. 4, pp. 3–17, Fall 1985.

[14] K. G. Shin and N. D. McKay, "Minimun time trajectory planning for industrial robots with general troque constraints," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (San Fransisco, California), pp. 412–417, Apr.7-10 1986.

[15] R. H. Taylor, "Planning and execution of straight line manipulator trajectories," *IBM Journal of Research and Development*, vol. 23, pp. 424–436, 1979.

[16] F. P. Preparata and M. I. Shamos, *Computational Geometry. An Introduction*. Springer-Verlag, New York, 1985.