A Discrete Algorithm for Fixed-path Trajectory Generation at Kinematic Singularities[1]

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 for computing the necessary time-scaling to allow a non-redundant manipulator to follow a fixed Cartesian path containing kinematic singularities. The resulting trajectory is close to minimum-time, subject to bounds on joint velocities and accelerations. The algorithm assigns a series of knot points along the path, increasing the knot density in the vicinity of singularities. Appropriate path velocities are then computed for each knot point. Two experiments involving the PUMA manipulator are shown.

1. Introduction

The problem of kinematic singularities is a serious one for serial link manipulators assigned to execute prescribed Cartesian space tasks. Singularities are usually defined in terms of the manipulator Jacobian $\mathbf{J}$, which maps joint velocities $\dot{\boldsymbol{\vartheta}}$ into workspace velocities $\mathbf{v}$. At a singularity, $\mathbf{J}$ loses rank, and the process of inverting $\mathbf{J}$ to execute a prescribed $\mathbf{v}$ may result in extremely high joint velocities and accelerations.

A conventional way of handling singularities is to modify the calculation associated with the Jacobian inverse (or pseudo-inverse, for redundant manipulators), such as by adding a damping term [8, 13, 4, 5, 7]. Other techniques include directly eliminating degenerate degrees of freedom from $\mathbf{J}$ [1] or using the Jacobian transpose [3] in place of its inverse. These approaches to the singularity problem usually result in some deviation from the prescribed reference path. Also, it can be difficult to keep joint velocities, and more troublesome, accelerations, within bounds without incurring sluggish performance.

As an alternative, some recent work has focused on handling singularities exclusively by *time-scaling* the trajectory, without deviating from the desired path unless that path actually goes outside the workspace. If the path we wish to follow is given by $\mathbf{X}(s)$, where s is a scalar parameter, then the problem can be stated as follows:

Problem 1. *Suppose a non-redundant manipulator is to follow a path $\mathbf{X}(s)$ for which the corresponding inverse kinematic solution $\boldsymbol{\vartheta}(s)$ is known. Then find a path timing $s(t)$ such that, for each joint ϑ_j, the induced joint velocities and accelerations are bounded by $|\dot{\vartheta}_j| \leq V_j$ and $|\ddot{\vartheta}_j| \leq A_j$, regardless of the presence of singularities. Such a timing will be termed* **admissible**.

Additional bounds on $\dot{s}$ and $\ddot{s}$ can be added by treating s as an extra joint coordinate (such as joint "0", with $\vartheta_0 \equiv s$).

$\dot{\vartheta}_j$ and $\ddot{\vartheta}_j$ are related to $\vartheta_j(s)$ by the chain rule:

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

At singularities, one or more $\vartheta_j'(s)$ or $\vartheta_j''(s)$ may become infinite, with serious implications for $\dot{\vartheta}_j$ and $\ddot{\vartheta}_j$. Nevertheless, the feasibility of solving Problem 1 in certain cases involving 2R and 3R manipulators was studied in [9, 2]. The general solvability of Problem 1 for paths which are piecewise analytic is described [6]. Loosely stated, the idea is to make sure that $\dot{s}$ and $\ddot{s}$ approach zero as fast as the elements of $\vartheta_j'(s)$ or $\vartheta_j''(s)$ approach infinity.

We present in this paper a discrete algorithm for computing an approximate solution to problem 1. Its novelty lies in being able to handle general paths, produce solutions which are nearly time-optimal (subject to $|\dot{\vartheta}_j| \leq V_j$ and $|\ddot{\vartheta}_j| \leq A_j$), and do the required computations rather quickly. The algorithm is named DAO (for Discrete Approximate Optimal-admissible timing).

Rather than explicitly computing the path timing $s(t)$, the algorithm instead computes $\dot{s}$ as a function of s; $s(t)$ can then be obtained as the solution to the differential equation $ds/dt = f(s)$. This approach makes it easier to incorporate constraints on the joint accelerations. To obtain an $s(t)$ which is monotonically increasing, we require that $\dot{s} \geq 0$, with $\dot{s} = 0$ only at single points.

The reader may note that solving Problem 1 is, in principle, equivalent to solving the fixed-path minimum-time trajectory problem [12, 10]. Solutions to the latter usually consider actuator force/torque limits and the full manipulator dynamics, but do not work at singularities. Essentially, the DAO algorithm solves a simplified (i.e., unit dynamics) version of the fixed-path minimum-time problem, but does so robustly at singularities. Presumably the algorithm could be extended to incorporate dynamics, at the cost of increased computation. However, it meets our immediate objective, which is to make Cartesian paths containing singularities as realizable as trapezoidal-velocity trajectories for joint paths.

[1] Presented at the *IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 22-28, 1996

2. Algorithm Overview

The algorithm is assumed to work in conjunction with a discrete-time trajectory generator which produces joint position setpoints every T seconds. Assume that the path is defined over some s interval $[s_A, s_B]$. The algorithm selects K knot points $s_i \in [s_A, s_B]$ and then computes, for each one, a value of $\dot{s}$, denoted by ${}^t v_i$. This output is then interpolated by the trajectory generator to determine s for each trajectory sample, from which $\boldsymbol{\vartheta}(s)$ is computed by applying inverse kinematics to $\mathbf{X}(s)$. Interpolation is done by assuming that $\ddot{s}$ is constant between knot points; it is easy to show that this is equivalent to assuming that $\dot{s}^2(s)$ is piecewise-linear.

At each knot point s_i, the algorithm computes both $\boldsymbol{\vartheta}(s_i)$ and $\boldsymbol{\vartheta}'(s_i)$. The latter can be computed from

$$\boldsymbol{\vartheta}'(s_i) \;=\; \mathbf{J}(s_i)^{-1}\mathbf{u}(s_i), \tag{1}$$

where $\mathbf{u}(s_i)$ is the tangent to $\mathbf{X}(s_i)$. Near singularities, one or more $\vartheta_j'(s_i)$ may approach infinity. If $\mathbf{J}(s_i)^{-1}$ is available in symbolic form, this will correspond to a division by a small number in some of the expressions. Otherwise, robust matrix inversion techniques (such as the singular value decomposition) can be used to identify which $\vartheta_j'(s)$ blow up. At present, such cases are handled by simply clipping $\vartheta_j'(s_i)$ to some large value of appropriate sign. A possibly superior alternative, not yet implemented, would be to displace s_i slightly so that the affected $\vartheta_j'(s_i)$ values are large but not infinite.

The algorithm's effectiveness comes from increasing the knot point density near singularities, as described below.

The algorithm calculates the knot velocities ${}^t v_i$ to be as large as possible (to approach a time-optimal solution) while trying to approximately satisfy $|\dot{\vartheta}_j| \le V_j$ and $|\ddot{\vartheta}_j| \le A_j$. Specifically, the ${}^t v_i$ are calculated to try and ensure that

$$|\vartheta_j'(s_i)| \le V_j \tag{2}$$

and

$$|\bar{a}_{ji}| \le A_j, \tag{3}$$

where $\bar{a}_{ji}$ is the *average* acceleration between the knots s_i and s_{i+1}, under the assumption of constant $\ddot{s}$ between knot points. *Average* accelerations are used because (a) it obviates the need to calculate second-order solution derivatives and (b) it has been experimentally observed to yield good timing results.

The ${}^t v_i$ are computed in several stages:

1. Select the knot points s_i.
2. Assign initial knot velocities, denoted by ${}^b v_i$, to satisfy (2) and *approximately* satisfy (3).
3. Forward pass: for $i = 1$ to K, *try* to satisfy (3) by *reducing* the ${}^b v_i$, producing a new set of velocities ${}^f v_i$.
4. Backward pass: for $i = K$ down to 1, complete the enforcement of (3) by reducing the ${}^f v_i$. This yields the final velocities ${}^t v_i$.

By properly chosing the initial ${}^b v_i$, it can be proven that the subsequent forward and backward passes will then ensure satisfaction of (3).

3. Knot point selection

Knot selection is at the heart of the algorithm. The idea is to start with some nominal set of knots, and then increase the knot density appropriately in regions where the path solution is highly non-linear (e.g., near singularities).

What makes this work is the fact that the trajectory generator interpolating the algorithm output works in discrete time, and so the difference between successive position set points must always be finite (as noted in [11]).

Now observe that the *average* velocity $\bar{\dot{\vartheta}}_j$ of joint ϑ_j during the travel from knot s_i to s_{i+1} is given by

$$\bar{\dot{\vartheta}}_j = \frac{\vartheta_j(s_{i+1}) - \vartheta_j(s_i)}{\Delta t}. \tag{4}$$

This leads to the *velocity* rule for knot point selection: if the knots are spaced closely enough that

$$|\vartheta_j(s_{i+1}) - \vartheta_j(s_i)| \le V_j T, \tag{5}$$

where T is the trajectory sample interval, then if the transit time between knots is less than or equal to T, the associated *average* velocity will be within bounds.

The *acceleration* rule is a bit more complicated but similarly motivated; details of its rational can be found in [6]. Letting $\Delta s_i \equiv s_{i+1} - s_i$, enough knot points are added so that for each i, *either*

$$|\vartheta_j'(s_{i+1}) - \vartheta_j'(s_i)| \le \frac{A_j T^2}{\Delta s_i} \tag{6}$$

or

$$|\vartheta_j(s_{i+1}) - \vartheta_j(s_i)| \le \frac{A_j T^2}{2} \tag{7}$$

is satisfied. Relation (7) keeps the number of knots bounded in cases where $\vartheta_j'(s_{i+1})$ or $\vartheta_j'(s_i)$ become very large.

Knot point creation is currently implemented using a simple bisection strategy where new knots are inserted between existing ones until both the velocity rule (5), and one of the acceleration rules ((6) or (7)), are satisfied. As long as the path solution $\boldsymbol{\vartheta}(s)$ is continuous, the process is guaranteed to converge. Performance of the algorithm is also enhanced by placing a specific knot point very close to any path singularity which is encountered.

These knot selection rules are heuristicly based and have been experimentally confirmed to yield good results, without creating an unmanageable number of knots. An error analysis is presently being studied.

4. Constraints on v_i

In this section, we describe what constraints on individual knot velocities v_i are necessary to satisfy (2) and (3).

First, it is necessary that $v_i \leq b_i$, where

$$b_i = \min_j(Q_{ji}), \tag{8}$$

and

$$Q_{ji} = \begin{cases} \dfrac{V_j}{\dot{\vartheta}'_j(s_i)} & \text{if } \dot{\vartheta}'_j(s_i) \text{ is not close to } 0, \\ \sqrt{\dfrac{A_j}{2|\dot{\vartheta}''_j(s_i)|}} & \text{otherwise.} \end{cases}$$

From the chain rule $\dot{\vartheta}_j = \dot{\vartheta}'_j(s)\dot{s}$, it can be seen that the upper definition of Q_{ji} enforces (2). At zeros of $\dot{\vartheta}'_j(s_i)$, the chain rule expression for $\ddot{\vartheta}_j$ reduces to $2\dot{\vartheta}''_j(s)\dot{s}^2$, and so enforcement of (3) requires the lower definition of Q_{ji} (for which an estimate of $\dot{\vartheta}''_j(s_i)$ can be obtained by applying finite differences to nearby $\dot{\vartheta}'_j(s_i)$ values).

At singularities where one or more $\dot{\vartheta}'_j(s_i)$ approach infinity, b_i will approach zero. In such cases, setting b_i to zero does not by itself guarantee that $\dot{\vartheta}_j$ will be bounded (since $\dot{s}$ no longer specifies $\dot{\vartheta}_j$ uniquely [6]), but the action of the rest of the algorithm will in fact ensure that all $\dot{\vartheta}_j$ are brought to zero (although see Section 9 in this regard).

Next, consider the constraint (3). If Δt is the travel time between knots s_i and s_{i+1}, then the associated average acceleration $\bar{a}_{ji}$ is given by

$$\bar{a}_{ji} = \frac{\dot{\vartheta}_j(s_{i+1}) - \dot{\vartheta}_j(s_i)}{\Delta t}. \tag{9}$$

Because it is assumed that $\ddot{s}$ is constant between knot points,

$$\Delta t = \frac{2\Delta s_i}{v_i + v_{i+1}}. \tag{10}$$

For notational convenience, let $f_i \equiv \dot{\vartheta}'_j(s_i)$ for some specific coordinate ϑ_j, and let $\Delta f_i \equiv f_{i+1} - f_i$. Then substituting (10) into (9) and applying the chain rule $\dot{\vartheta}_j = \dot{\vartheta}'_j(s)\dot{s}$, we obtain

$$f_{i+1}v_{i+1}^2 + \Delta f_i v_{i+1} v_i - f_i v_i^2 - 2\Delta s_i \bar{a}_{ji} = 0. \tag{11}$$

For any given value of $\bar{a}_{ji}$, this represents a hyperbola in the v_i-v_{i+1} plane. We define an *admissible sub-region* $\mathcal{W}_{j,i}$ to be the set of (v_i, v_{i+1}) for which $|\bar{a}_{ji}| \leq A_j$. This region is bounded by the two hyperbolas corresponding to $\bar{a}_{ji} = \pm A_j$ in (11), as illustrated in Figure 1.

The region in the v_i-v_{i+1} plane for which constraint (3) is satisfied for *all* joints is called the *admissible region* $\mathcal{W}_i$ and is defined by

$$\mathcal{W}_i \equiv \bigcap_j \mathcal{W}_{j,i}.$$

Finally, a *complete admissible region* $\mathcal{W}_i^*$ can be defined which also satisfies the constraint (2), along with the additional requirement that each v_i be non-negative, by intersecting $\mathcal{W}_i$ with the square defined by $0 \leq v_i \leq b_i$ and $0 \leq v_{i+1} \leq b_{i+1}$ (see Figure 2). Any $(v_i, v_{i+1}) \in \mathcal{W}_i^*$ therefore satisfies both constraints (2) and (3).

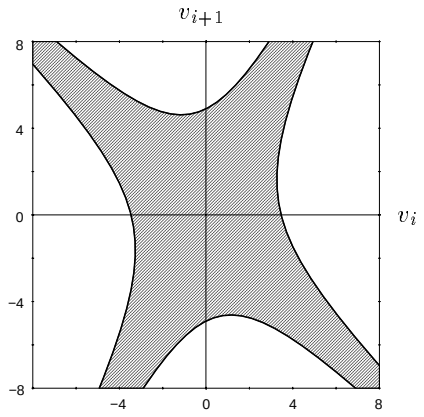

FIGURE 1. A sub-region $\mathcal{W}_{j,i}$ (shown in grey) corresponding to $f_i = 2$, $f_{i+1} = 1$, $A_j = 6$, and $\Delta s_i = 2$.

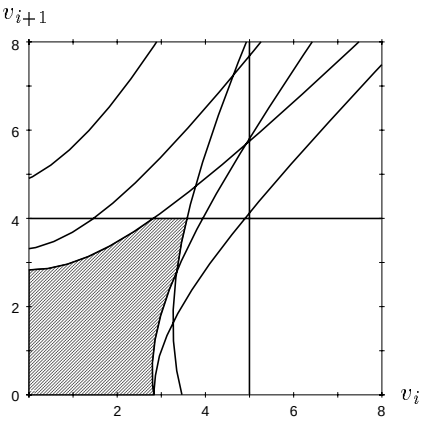

FIGURE 2. A complete admissible region $\mathcal{W}_i^*$ given by the intersection of several $\mathcal{W}_{j,i}$ and the square defined by $0 \leq v_i \leq b_i$ and $0 \leq v_{i+1} \leq b_{i+1}$, for $b_i = 5$ and $b_{i+1} = 4$.

The objective of the DAO algorithm can now be restated: try to make the output knot velocities ${}^t v_i$ as large as possible subject to $({}^t v_i, {}^t v_{i+1}) \in \mathcal{W}_i^*$. To do this, the following computations on the $\mathcal{W}_i$ and $\mathcal{W}_i^*$ are necessary:

1. Intersect $\mathcal{W}_i$ with a fixed v_i;
2. Intersect $\mathcal{W}_i$ with a fixed v_{i+1};
3. Compute the vertices of $\mathcal{W}_i^*$.

Details on these calculations are given in Appendix A.

5. **Determining initial velocities**

After the knots have been selected, each one is assigned an initial velocity ${}^b v_i$. This is done by first computing a point $(w_{i,1}, w_{i,2})$ in each $\mathcal{W}_i^*$ that maximizes $v_i + v_{i+1}$ (to help minimize the implied travel time between s_i and s_{i+1}). It can be shown [6] that such a point must lie on a vertex of

$\mathcal{W}_i^*$, the computation of which is described in section A.3. The ${}^b v_i$ are then calculated as follows:

```
b_v_1 := w_{1,1};
for i from 2 to K−1:
    b_v_i := min(w_{i,1}, w_{i−1,2});
b_v_K := w_{K−1,2};
```

Note that resulting pairs $({}^b v_i, {}^b v_{i+1})$ may not necessarily be contained in $\mathcal{W}_i^*$. However, since the seed $(w_{i,1}, w_{i,2})$ *is* contained in $\mathcal{W}_i^*$, it can be proven [6] that the subsequent forward and backward passes of the algorithm will yield final velocities ${}^t v_i$ whose pairs are also contained in $\mathcal{W}_i^*$.

6. Forward and Backward Passes

The forward pass computes a new set of velocities ${}^f v_i$:

```
f_v_1 := b_v_1;
for i from 1 to K − 1:
    y := max{v_{i+1} : 0 ≤ v_{i+1} ≤ b_v_{i+1}, (f_v_i, v_{i+1}) ∈ W_i};
    if (y ≠ ∅) then
        f_v_{i+1} := y;
    else
        f_v_{i+1} := b_v_{i+1};
```

In other words, for each i, counting up from 1, ${}^f v_{i+1}$ is nominally set to ${}^b v_{i+1}$. Then if $({}^f v_i, {}^f v_{i+1}) \notin \mathcal{W}_i$, and this can be corrected by lowering ${}^f v_{i+1}$, we do so. The computation involves determining the admissible v_{i+1} values for a given ${}^f v_i$, as described in Section A.1.

The backward pass repeats this in the reverse direction:

```
t_v_K := f_v_K;
for i from K − 1 down to 1:
    t_v_i := max{v_i : 0 ≤ v_i ≤ f_v_i, (v_i, f_v_{i+1}) ∈ W_i};
```

In other words, for each i descending from K, ${}^t v_i$ is nominally set to ${}^f v_i$. Then if $({}^t v_i, {}^t v_{i+1}) \notin \mathcal{W}_i$, this is corrected by lowering ${}^t v_i$. That such a correction is always possible follows from the above-mentioned proof in [6]. The computation involves determining the admissible v_i values for a given ${}^t v_{i+1}$, as described in Section A.2.

Note that since these passes only *reduce* velocities, $({}^t v_i, {}^t v_{i+1}) \in \mathcal{W}_i$ implies $({}^t v_i, {}^t v_{i+1}) \in \mathcal{W}_i^*$.

7. Algorithm Summary

Input: Continuous path solution $\boldsymbol{\vartheta}(s)$ defined on $[s_A, s_B]$.

Output: A set of path velocities ${}^t v_i$ defined for K knot points $s_i \in [s_A, s_B]$, implicitly specifying an approximately optimal admissible path timing $s(t)$.

Step D1. (Knot point selection). Create K knots s_i, with $s_1 = s_A$ and $s_K = s_B$, so as to satisfy the velocity and acceleration rules described in Section 3. If possible, insert a knot point close to each path singularity.

Step D2. (Initialization). Compute the initial knot velocities ${}^b v_i$ as described in Section 5.

Step D3. (Forward pass). Starting at $i = 1$, compute the knot velocities ${}^f v_i$ as described in Section 6.

Step D4. (Backward pass). Working backward from $i = K$, compute the ${}^t v_i$ as described in Section 6.

7.1. **Complexity.** If K is the number of knots and M the number of joint coordinates, then step D2 has the worst complexity, $O(KM^3)$, due to the $O(M^3)$ complexity of computing the vertices of each $\mathcal{W}_i^*$ (Section A.3). A more efficient calculation may be possible, but has not yet been investigated.

8. Experimental Results

Numerous experiments are described in [6] involving planar 2R and PUMA robots; only a couple involving the PUMA will be shown here. Both involve paths where s is the translational arc length, and were undertaken with a trajectory sample period of $T = 50$ msec, and $V_j = 60$ deg/sec and $A_j = 150$ deg/sec^2 for all robot joints. Constraints $|\dot{s}| \le V_0$ and $|\ddot{s}| \le A_0$ were also imposed by treating s as an additional joint coordinate ϑ_0, with $V_0 = 200$ mm/sec and $A_0 = 700$ mm/sec^2. Computations were done in 64-bit double precision, and large values of $\vartheta_j'(s_i)$ were clipped to 10^8. Each experiment is illustrated by a PUMA stick figure animation, and plots of selected velocities as functions of time, before and after application of the DAO algorithm. To make it easier to judge algorithm performance, velocity profiles are scaled as shown in Figure 3.

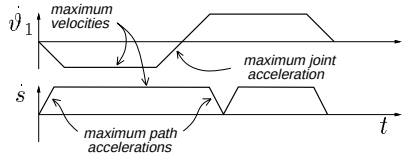

FIGURE 3. Velocity plots are scaled so that velocity and acceleration constraints appear as shown here. Acceleration limits correspond to a slope of ± 1 for robot joints and ± 2 for $\dot{s}$.

The example of Figure 4 involves the PUMA elbow singularity. Computations were done for $M = 4$ (ϑ_1 through ϑ_3, and $\vartheta_0 \equiv s$), required 206 knots, and took 181 msec on a Silicon Graphics "Indy" workstation with an R4600 CPU rated at 11 Mflops. The example of Figure 5 involves the PUMA shoulder singularity. Computations were again done for $M = 4$, required 217 knots, and took 149 msec.

These results, typical of a larger body of tests, indicate that the algorithm does in fact produce a timing which meets the ideal constraints $|\dot{\vartheta}_j| \le V_j$ and $|\ddot{\vartheta}_j| \le A_j$ very tightly, while being quite close to optimal. The latter statement can be verified by noting that in the resulting trajectories, one or more coordinates is always close to saturation with respect to either its velocity or acceleration constraint.

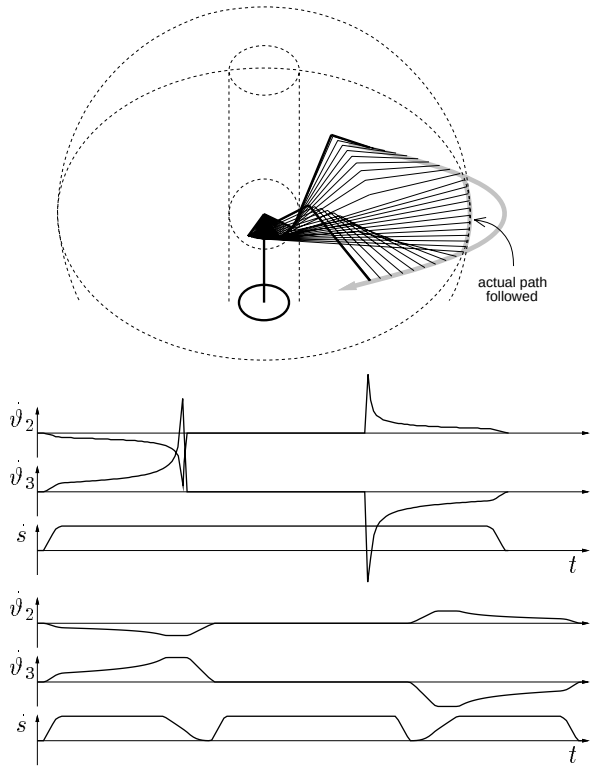

FIGURE 4. Parabolic reference path, in the plane $z = 0$, which leaves and then reenters the robot workspace. The actual path follows the boundary when the reference path goes outside of the workspace. Upper plots show $\dot{\vartheta}_2$, $\dot{\vartheta}_3$ corresponding to a constant path velocity $\dot{s}$; spikes correspond to the *elbow* singularity where the reference path leaves or enters the workspace. Lower plots show the modified velocities produced by the DAO algorithm.

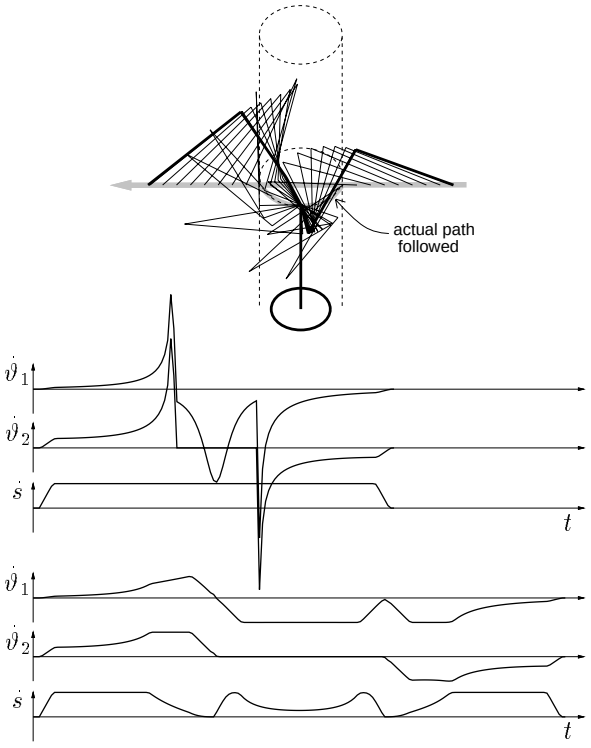

FIGURE 5. A straight line reference path, in the plane $z = 150$, which cuts through the cylindrical void in the center of the workspace. When the reference path is inside the void, the actual path is projected onto the boundary cylinder. Upper plots show $\dot{\vartheta}_1$, $\dot{\vartheta}_2$ corresponding to a constant path velocity; spikes correspond to the *shoulder* singularity where the path intersects the cylinder. Lower plots show the modified velocities produced by the DAO algorithm.

9. **Conclusion**

The DAO algorithm demonstrates the practical feasibility of handling singularities in fixed-path trajectories by time scaling alone. The present implementation appears to give excellent results in terms of achieving near-optimal solutions which honor the velocity and acceleration constraints.

There is one aspect in which the algorithm is *not* optimal: at a singularity where one or more $\vartheta_j'(s)$ approaches infinity, all joints are brought to rest, whereas in some (less common) cases, the optimal solution calls for the "most singular" joints to have non-zero velocities. This problem, discussed in [6], should be corrected.

Other work on the method can be done along the following lines: (a) improving the knot selection process, both in terms of theoretical understanding and trying to reduce the number of knots (since we have no reason to believe that the number of knots selected is optimal), and (b) simplifying the computations and making them more robust, including possibly replacing all derivative calculations with ones involving only finite differences of $\boldsymbol{\vartheta}(s)$.

Generalizations of the DAO algorithm to include the full manipulator dynamics, and hence actuator force/torque constraints, may also prove useful.

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).

References

[1] E. W. Aboaf and R. P. Paul, "Living with the Singularity of Robot Wrists." *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, pp. 1713 - 1717.

[2] C. Chevallereau and B. Daya, "A New Method for Robot Control in Singular Configurations with Motion in any Cartesian Direction." *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 2692 - 2697 (Vol. 4).

[3] Pasquale Chiacchio, Stefano Chiaverini, Lorenzo Sciavicco, Bruno Siciliano, "Closed-Loop Inverse Kinematics Schemes for Constrained Redundant Manipulators with Task Space Augmentation and Task Priority Strategy". *International Journal of Robotics Research*, August 1991, pp. 410 - 425 (Vol. 10, No. 4).

[4] S. Chiaverini, O. Egeland, and R. K. Kanestrom, "Achieving User-defined Accuracy with Damped Least-squares Inverse Kinematics."

Fifth International Conference on Advanced Robotics (91 ICAR), Pisa, 1991, pp. 672 - 677.

[5] A. S. Deo and I. D. Walker, "Adaptive Non-linear Least Squares for Inverse Kinematics." *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pp. 186 - 193 (Vol. 1).

[6] J. E. Lloyd, *Robot Trajectory Generation for Paths with Kinematic Singularities*. Ph. D. dissertation, Department of Electrical Engineering, McGill University, January 1995.

[7] A. A. Maciejewski and C. A. Klein, "Numerical Filtering for the Operation of Robotic Manipulators through Kinematically Singular Configurations." *Journal of Robotic Systems*, December 1988, pp. 527-552 (Vol. 5, No. 6).

[8] Y. Nakamura and H. Hanafusa "Inverse Kinematic Solutions with Singularity Robustness for Robot Manipulator Control." *Journal of Dynamic Systems, Measurement, and Control*, September 1986 pp. 163-171 (Vol. 108, No. 3)

[9] L. Nielsen, C. Canudas de Wit, and P. Hagander, "Controllability Issues of Robots near Singular Configurations." *Advances in Robot Kinematics, 2nd International Workshop*, Linz, 1990, pp. 283 - 290.

[10] F. Pfeiffer and R. Johanni, "A Concept for Manipulator Trajectory Planning." *IEEE Journal of Robotics and Automation*, April 1987, pp. 115 - 123 (Vol. RA-3, No. 2).

[11] E. D. Pohl and H. Lipkin, "A New Method of Robotic Motion Control near Singularities." *Fifth International Conference on Advanced Robotics (91 ICAR)*, Pisa, 1991, pp. 405 - 410.

[12] K. G. Shin and N. D. McKay, "Minimum-time Control of Robotic Manipulators with Geometric Path Constraints." *IEEE Transactions on Automatic Control*, June 1985, pp. 531-541 (Vol. AC-30, No. 6).

[13] C. W. Wampler II and L. J. Leifer, "Applications of Damped Least-squares Methods to Resolved-rate and Resolved-acceleration Control of Manipulators." *Journal of Dynamic Systems, Measurement, and Control*, March 1988, pp. 31-38 (Vol. 110, No. 1).

Appendix A.

A.1. **Intersecting $\mathcal{W}_i$ with a fixed v_i.** This amounts to finding all the v_{i+1} that are admissible for a fixed v_i. One can compute the answer for each sub-region $\mathcal{W}_{j,i}$ and then intersect the results. In [6] it is shown that, for a particular $\mathcal{W}_{j,i}$, the admissible v_{i+1} are contained within two intervals $[A - B^+, A - B^-]$ and $[A + B^-, A + B^+]$, where

$$B^+ = \frac{\sqrt{(f_i + f_{i+1})^2 v_i^2 + C_i}}{2|f_{i+1}|}$$

$$B^- = \frac{\sqrt{(f_i + f_{i+1})^2 v_i^2 - C_i}}{2|f_{i+1}|}$$

$$A = \frac{-\Delta f_i v_i}{2 f_{i+1}}, \quad \text{and} \quad C_i = 8|f_{i+1}|\Delta s_i A_j$$

If $(f_i + f_{i+1})^2 v_i^2 - C_i \leq 0$, this reduces to a single interval $[A - B^+, A + B^+]$. If $f_{i+1} = 0$, the interval becomes

$$\left[-v_i - \frac{2\Delta s_i A_j}{|f_i| v_i}, -v_i + \frac{2\Delta s_i A_j}{|f_i| v_i} \right].$$

A.2. **Intersecting $\mathcal{W}_i$ with a fixed v_{i+1}.** This amounts to finding all the v_i that are admissible for a fixed v_{i+1}. The computations are the same as those described in the previous section, except with v_i and v_{i+1} interchanged, f_i replaced by $-f_{i+1}$, and f_{i+1} replaced by $-f_i$.

A.3. **Computing the vertices of $\mathcal{W}_i^*$.** These are points on the boundary of $\mathcal{W}_i^*$ corresponding to the intersections of the hyperbolic curves and straight lines comprising the boundary. At present, the vertices are computed in a brute force way, by finding all such intersections and then discarding those not on the boundary. First, the lines $v_i = 0$, $v_i = b_i$, $v_{i+1} = 0$, and $v_{i+1} = b_{i+1}$ have 4 intersection points. Intersecting these with the hyperbolic boundaries of the $\mathcal{W}_{j,i}$, using the equations of sections A.1 and A.2, yields an additional $10M$ points, where M is the number of joint coordinates. Now for the intersections of the hyperbolas themselves: Suppose there exist two regions $\mathcal{W}_{j,i}$ and $\mathcal{W}_{k,i}$, corresponding to coordinates ϑ_j and ϑ_k, whose boundaries are described by the two pairs of hyperbolas represented by

$$f_{i+1} v_{i+1}^2 + \Delta f_i v_{i+1} v_i - f_i v_i^2 = \pm 2\Delta s_i A_j,$$

$$g_{i+1} v_{i+1}^2 + \Delta g_i v_{i+1} v_i - g_i v_i^2 = \pm 2\Delta s_i A_k,$$

where $f_i \equiv \vartheta_j'(s_i)$ and $g_i \equiv \vartheta_k'(s_i)$. It can then be verified that all the intersection points (v_i, v_{i+1}) between the two boundaries are given by

$$v_{i+1} = \frac{\sigma(f_i H_k - g_i H_j)\sqrt{2\Delta s_i}}{\sqrt{[f_{i+1} g_i - f_i g_{i+1}][(g_i + g_{i+1})H_j - (f_i + f_{i+1})H_k]}}$$

$$v_i = \frac{\sigma(g_{i+1} H_j - f_{i+1} H_k)\sqrt{2\Delta s_i}}{\sqrt{[f_{i+1} g_i - f_i g_{i+1}][(g_i + g_{i+1})H_j - (f_i + f_{i+1})H_k]}}$$

where $\sigma = \pm 1$, and $H_j = \pm A_j$ and $H_k = \pm A_k$. This gives eight solutions, only two of which (it can be shown) may correspond to vertices of $\mathcal{W}_i^*$. Among all M sub-regions $\mathcal{W}_{j,i}$ we then have at most $M(M-1)$ such points, and the total number of intersections among all bounding lines and curves is therefore bounded by $M^2 + 9M + 4$.

Checking if a point is on the boundary is done by bounding box tests, interval merging, and (lastly) checking to see if it is contained within all the sub-regions $\mathcal{W}_{j,i}$ by seeing if it satisfies the associated equations

$$|f_{i+1} v_{i+1}^2 + \Delta f_i v_{i+1} v_i - f_i v_i^2| \leq 2\Delta s_i A_j.$$

Since there are M such equations, the overall complexity of computing the vertices is $O(M^3)$.