Trajectory Generation Implemented as a Non-linear Filter

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

Computer Science Department
University of British Columbia

August 16, 1998

Technical Report TR-98-11
Computer Science Department
University of British Columbia

201-2366 Main Mall, University of British Columbia, Vancouver, B.C. V6T 1Z4
Phone: (604) 822-5109 FAX: (604) 822-5485
E-mail: lloyd@cs.ubc.ca

Abstract

A primary purpose of robotic trajectory generation is to produce a timed path which is sufficiently well behaved that it can be tracked by a manipulator. However, the creation of good paths becomes somewhat problematic in situations where a manipulator is required to follow a target whose position is varying erratically (for instance, if the target is specified using a position sensor held in an operator's hand). This paper presents a simple solution for such situations, in which the "trajectory generator" is implemented as a non-linear filter which tries to bring its output (manipulator setpoints) to the input (target position) as quickly as possible, subject to constraints on velocity and acceleration. The solution to this problem in one dimension is quite easy. For multiple dimensions, the problem can be handled by applying one-dimensional solutions to a pair of appropriately chosen coordinate axes. An interesting feature of the approach is that it can handle spatial rotations as well as vector quantities.

1 Introduction

In recent years, there has been a growing tendency to use robots in an interactive or sensor-driven manner. Some examples of the first include robot positioning using joysticks, mouse/screen interfaces, or position sensors held in an operator's hand; visual servoing is a good example of the second.

In many of these situations, the robot is required to follow a particular reference point in space, based on the wishes of either an operator or some higher level control system. The problem is that these reference inputs can sometimes move about quite quickly, way beyond the capabilities of the manipulator. In the case of input devices, the discrepancy between the dynamics of the input device and the robot can be enormous. An extreme example is trying to have a robot track the position of a mouse cursor on a computer screen; the mouse cursor can describe a trajectory that the robot has no hope of tracking accurately.

The solution to this, of course, is to smooth the reference inputs in such a way that they can be tracked, with reasonable accuracy, by the manipulator and its control system. In more conventional robotic applications, this is undertaken by a trajectory generator, which takes a sequence of target points and produces a feasible timed path [12, 7, 4, 13]. In the case of a prescribed spatial path, timing can be added to produce a minimum-time solution that meets the manipulator's dynamic constraints [10, 11, 9].

Although trajectory generation is a well-studied subject, almost all treatments assume that the path or target points(s) are known at least a short time in advance, and don't change. By contrast, this paper presents what is essentially a trajectory *filter*, which tries to bring the manipulator to rest at the current desired target position, as fast as possible, subject to some dynamical constraints. This filter runs in discrete time, with a fixed cycle time T, and outputs one position and velocity setpoint ($\mathbf{x}_i$,$\mathbf{v}_i$) per cycle. These output setpoints can then be tracked by a lower level control system. The setpoints themselves, and the current target $\mathbf{x}_d$, can be any vector quantity, which is sufficient for describing joint coordinates or spatial translations. Spatial rotations can also be handled, with some modifications to the filter's algorithm (Section 5).

The filter's function can now be stated more precisely:

Problem 1 *Given existing position and velocity setpoints ($\mathbf{x}_{i-1}, \mathbf{v}_{i-1}$), calculate the setpoints ($\mathbf{x}_i, \mathbf{v}_i$) for the next cycle, in such a way as to try and bring $\mathbf{x}$ to rest at the current target $\mathbf{x}_d$ as quickly as possible, subject to constraints on velocity and acceleration.*

The idea is that these calculations are repeated from scratch each sample period, with the only knowledge of the past being the current setpoint state $\mathbf{x}_{i-1}, \mathbf{v}_{i-1}$. Consequently, it doesn't matter how or if $\mathbf{x}_d$ varies from cycle to cycle. Only the current value of $\mathbf{x}_d$ is considered in the calculation.

A more general statement of the problem would also include a target velocity $\mathbf{v}_d$. However, since this makes the required calculations more complex, and is not necessary for many applications (particularly those which are simply position based), in this work we shall simply assume that $\mathbf{v}_d = 0$.

Problem 1, in its full generality, becomes something like the unconstrained minimum-time path problem [9], for which it can be hard to generate even off-line solutions. Fortunately, we aren't necessarily interested in trajectories which are fully optimal; instead, our primary concern is that they be realizable. Therefore, we assume that the necessary dynamic constraints can be met by imposing bounds on the velocity and acceleration:

$$\|\mathbf{v}\| \le V, \quad \|\dot{\mathbf{v}}\| \le A. \tag{1}$$

This is often quite reasonable for joint space coordinates, and works for Cartesian coordinates also, provided that the robot motion is restricted to a part of the workspace where the manipulator Jacobian is relatively well conditioned (see the comment on this in Section 8).

Besides facilitating implementation, the constraints of (1) are often desirable in their own right because they tend to induce straight-line constant-speed motions for cases when the target is actually at rest. For Cartesian motions in particular, this represents a fairly intuitive behavior.

The operation of the filter is based on a one-dimensional solution to Problem 1, described in Section 3. The generalization to multiple dimensions is given in Section 4. Subsequent sections describe handling rotations, coordination between filters, and stability issues.

2 Related Work

This work is similar in spirit to that described in [2], which uses polynomials to create smooth paths to targets which can change as often as every 50 msec. While those paths are smoother than the ones generated here, they are less optimal with respect to the constraints, and, most critically, are prone to overshoot (a common problem with polynomial methods, as noted in [3]). Another related work is [6], in which joint-based motions are generated on the fly for purposes of catching a moving target whose position is observed by a sensor system.

In [5], the problem of creating smooth *transitions* between time-varying paths is considered, but the conditioning of the paths themselves is not. In cases where a path to be tracked is well-behaved at a large scale but ill-behaved at a fine scale (perhaps due to sampling effects), it may be possible to do the necessary smoothing using local filters. For instance, in [1], an $\alpha - \beta - \gamma$ filter is used to smooth target inputs from a vision sensor.

3 Filtering in One Dimension

Not surprisingly, of Problem 1 has a fairly direct solution in one dimension. The vector quantities reduce to scalars, so that setpoints become x_i, v_i, and the constraints (1) become

$$|v| \le V, \quad |\dot{v}| \le A. \tag{2}$$

The basic idea is straight forward: given initial setpoints x_{i-1}, v_{i-1} and a current target value x_d, a velocity profile $v(t)$ is computed which brings x to rest at x_d in minimum time, subject to (2). If the velocity profile is assumed to start at $t = 0$, the next pair of setpoints x_i, v_i can be computed from

$$x_i = x_{i-1} + \int_0^T v(t)dt, \quad v_i = v(T).$$

Again, the profile $v(t)$ is recomputed once every cycle, and x_d may vary arbitrarily between cycles.

In fact, in order to facilitate coordination between multiple trajectory filters (as described in Section 6), it is useful to implement the solution to a slightly more general problem: determine a velocity profile $v(t)$ which takes x to x_d in minimum time, subject to (2), *unless* this minimum time is less than some prescribed time t_d, in which case $v(t)$ is stretched out so that its duration equals t_d. Solving the original problem amounts to taking $t_d = 0$.

The remainder of this section will describe the computation of $v(t)$.

3.1 Profile with zero initial velocity

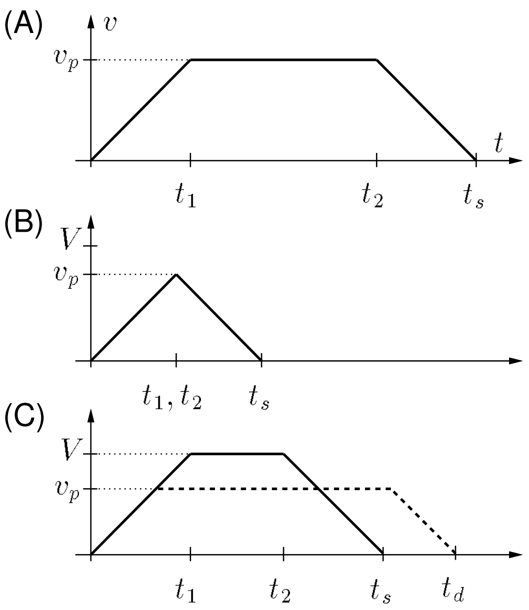

Figure 1: Standard trapezoidal velocity profiles.

If $v_{i-1} = 0$, then $v(t)$ is simply a standard trapezoidal velocity profile, as illustrated by Figure 1. In Figure 1(A), an acceleration $\dot{v} = a$ is initially applied until time t_1, bringing v to some peak value v_p; this is followed by a constant velocity slew phase until time t_2, when a constant deceleration $\dot{v} = -a$ is then applied which brings v back to zero at time t_s. Letting $\Delta x \equiv x_d - x_{i-1}$, it is easy to see that the total time t_s associated with the profile is given by

$$t_s = \frac{v_p}{a} + \frac{\Delta x}{v_p}. \tag{3}$$

It is well known that setting $v_p = V$ and $a = A$ results in a least-time trajectory subject to the constraints (2); formal proofs of this are often found in introductory books on optimal control theory. Assuming for the moment that $\Delta x > 0$, the only minor catch occurs when $\Delta x < V^2/A$, which implies that the target is too close to allow v to accelerate all the way to the velocity limit V. In this situation, shown in Figure 1(B), the peak velocity is given by $v_p = \sqrt{A\Delta x}$, there is no slew phase (i.e., $t_1 = t_2$), and equation (3) still holds. Finally, if t_s turns out to be *less* than the prescribed time t_d, t_s can be extended to time t_d by reducing v_p, according to

$$v_p = \frac{At_d - \sqrt{A^2 t_d^2 - 4\Delta x A}}{2}, \tag{4}$$

as shown by the dotted line in Figure 1(C). (It can be verified that $t_s < t_d$ implies that the square root in (4) is real-valued). This is not the only way to stretch out t_s, but it is the one which allows the acceleration values to remain at $\pm A$; this, in turn, will allow the general profile discussed in Section 3.2 to consist of at most three segments.

Since $v(t)$ consists of an acceleration, slew, and deceleration phase, with the acceleration magnitude equal to A and the final velocity v equal to 0, the slew velocity v_p alone provides all the information needed to specify $v(t)$ for a given Δx and v_{i-1}. However, for computational purposes, it is useful to supplement this with the profile duration t_s, and so $v(t)$ will generally be specified by the pair $\{v_p, t_s\}$.

Within this paper, profiles corresponding to $v_{i-1} = 0$ will be called *static*. A function *staticProf()* can be defined which produces $\{v_p, t_s\}$ for particular values of Δx and t_d. This is summarized in Algorithm 3.1, where the procedure has been generalized to handle $\Delta x < 0$.

3.2 Profile with non-zero initial velocity

The velocity profile for the general situation (where $v_{i-1} \neq 0$) has the same basic structure as the static case: an initial acceleration/deceleration phase until time t_1, a constant velocity slew phase until time t_2, and a final deceleration/acceleration phase. The acceleration values are $\pm A$, and v_p again uniquely characterizes the profile for a given target x_d and initial conditions x_{i-1}, v_{i-1}.

In the discussion that follows, we will assume, without loss, that $i - 1 = 0$, so that initial conditions are given by x_0 and v_0, and the filter's output setpoints are x_1 and v_1. As before, we let $\Delta x \equiv x_1 - x_0$.

In determining the shape of a general profile, it is helpful to first consider a minimum-time profile which simply brings v_0 to rest. This will correspond to a

funct *staticProf*$(\Delta x, t_d) \equiv$
  if $|\Delta x| > V^2/A$
    then
      $v_p := \mathrm{sgn}(\Delta x)V$
    else
      $v_p := \mathrm{sgn}(\Delta x)\sqrt{A|\Delta x|}$
  fi
  $t_s := |v_p|/A + \Delta x/v_p$
  if $t_s < t_d$
    then
      $v_p := {}^1\!/_2\,\mathrm{sgn}(\Delta x)\left[At_d - \sqrt{A^2t_d^2 - 4|\Delta x|A}\right]$
      $t_s := t_d$
  fi
  return$\{v_p, t_s\}$.

Algorithm 1: Computing a static velocity profile.

constant deceleration (or acceleration, if $v_0 < 0$) for a time given by $|v_0|/A$, and will entail a change in position given by ${}^1\!/_2\,\mathrm{sgn}(v_0)v_0^2/A$. We then consider how to modify this profile in order to accommodate the remaining required position change, given by

$$\Delta x^* \equiv \Delta x - \frac{\mathrm{sgn}(v_0)v_0^2}{2A}.$$

There are several cases, which are now described. Reference will be made to Figure 2, and without loss, it will be assumed that $\Delta x \geq 0$.

3.2.1 case with $v_0 > 0$ and $\Delta x^* > 0$

Here, we start by setting v_p to $\min(V, v_0)$, and then handle the remaining displacement Δx^* by adding a slew phase (at $v = v_p$) to the original deceleration profile (as shown by the solid line in Figure 2(A)). If $v_p < v_0$ (i.e., if $V < v_0$) then the initial acceleration phase will actually be a deceleration. This can occur in the multi-dimensional computation (Section 7), or if changes are made to V "on the fly". The resulting profile time is given by

$$t_s = \frac{v_0}{A} + \frac{\Delta x^*}{v_p}.$$

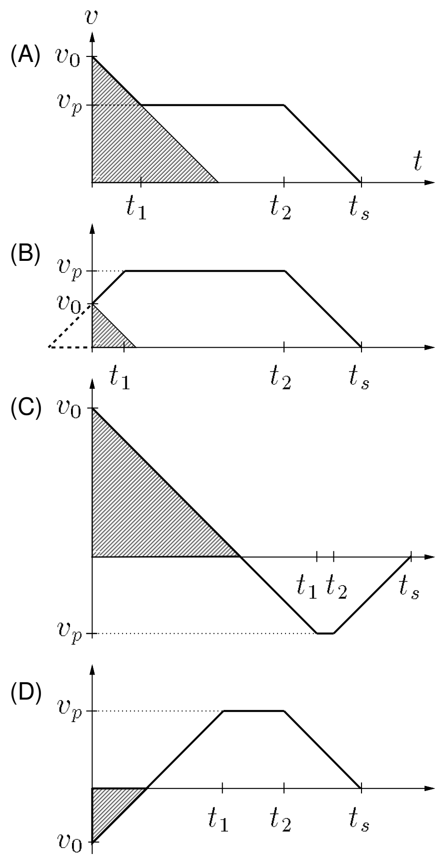

Figure 2: Handling a general velocity profile. In each illustration, the change in position resulting from bringing v_0 directly to rest is indicated by the grey shaded region.

If $t_d > t_s$, t_s can then be extended by lowering v_p according to

$$v_p = \frac{A\Delta x^*}{At_d - v_0}.$$

Otherwise, if $t_d \leq t_s$, and $v_p < V$, we try to improve the efficiency of the profile by *increasing* v_p up to a maximum value of V. What makes this easy to compute is that the corresponding profile can be considered as part of a longer *static* profile (dotted line, Figure 2(B)), corresponsing to a total displacement given by $\Delta x^+ = \Delta x + {}^1\!/_2 A v_0^2$ and a net desired time of $t_d + v_0/A$.

3.2.2 case with $v_0 > 0$ and $\Delta x^* \leq 0$

The occurrence of this case, shown in Figure 2(C), means that x is approaching the target too quickly and will overshoot. All that can be done is to bring v_0 to rest as quickly as possible, and then apply a static profile, with desired time $t_d - v_0/A$ and displacement Δx^*, to correct for the overshoot.

3.2.3 case with $v_0 \leq 0$

Shown in Figure 2(D), this is similar to the previous case, except now, if $v_0 < 0$, x is initially moving *away* from the target. After v_0 is brought to rest, the return to the target can be acheived by applying a static profile with desired time $t_d - |v_0|/A$ and displacement Δx^*.

A complete procedure for producing general profiles is contained within the function *filterOne()*, shown in Algorithm 3.2.3, which implements the trajectory filter in one dimension. With regard to the above discussion, the cases of Sections 3.2.2 and 3.2.3 have been combined, and the necessary changes to handle $\Delta x < 0$ have been introduced.

The integration used to compute x_1 is fairly trivial because of the piecewise-linear nature of $v(t)$. Some care needs to be taken to handle the case where $v_p = 0$, which occurs when bringing v_0 to zero puts x exactly on target. This obviates the need for a final acceleration/deceleration phase, so that $t_2 = t_s$.

4 Filtering in Multiple Dimensions

This section describes how to implement a multiple dimensional filter using a one dimensional filter. The notation $\Delta \mathbf{x} \equiv \mathbf{x}_d - \mathbf{x}_0$ will be used to describe the displacement from the current position setpoint $\mathbf{x}_0$ to the current target $\mathbf{x}_d$.

The first thing that should be noted is that the multi-dimensional problem always reduces to a *two* dimensional problem. This is because the trajectory should lie in the plane formed by the vectors $\Delta \mathbf{x}$ and $\mathbf{v}_0$; any departure from this plane would be extraneous and would take extra time. If $\Delta \mathbf{x}$ and $\mathbf{v}_0$ are parallel, then the problem simplifies even further into one dimension.

While an exact solution to the one-dimensional problem is fairly simple to produce, an exact solution to the multi-dimensional one is not. Instead, what we will present here is an approximate solution which exhibits reasonable behavior.

The basic approach is to treat the two-dimensional problem as simply a pair of one-dimensional problems; i.e., solve Problem 1 separately along each axis. Conceptually, this is similar to using ∞-norms in place of 2-norms in equation 1; not

funct *filterOne*$(\Delta x, x_0, v_0, t_d)$ $\equiv$
// First, compute $\{v_p, t_s\}$ describing $v(t)$:
$\Delta x^* = \Delta x - {}^1\!/_2 \operatorname{sgn}(v_0) v_0^2 / A$
if $\Delta x^* v_0 > 0$
  then
    $v_p := \operatorname{sgn}(v_0) \min(V, |v_0|)$
    $t_s := |v_0|/A + \Delta x^* / v_p$
    if $t_s < t_d$
      then
        $v_p := \frac{A \Delta x^*}{A t_d - |v_0|}$
        $t_s := t_d$
    elsif $v_p < V$
      then
        $\Delta x^+ := \Delta x + {}^1\!/_2 \operatorname{sgn}(v_0) v_0^2 / A$
        $\{v_p, t_s\} :=$ *staticProf*$(\Delta x^+, t_d + |v_0|/A)$
        $t_s := t_s - |v_0|/A$
    fi
  else
    $\{v_p, t_s\} :=$ *staticProf*$(\Delta x^*, t_d - |v_0|/A)$
    $t_s := t_s + |v_0|/A$
fi
// Then compute x_1, v_1 from $v(t)$:
$x_1 := x_0 + \int_0^T v(t) dt;\ v_1 = v(T)$
return$\{x_1, v_1\}$.

Algorithm 2: One-dimensional trajectory filter, computing new setpoints x_1, v_1.

coincidentally, ∞-norms are by their nature much easier to compute. However, ∞-norms have the particularly annoying drawback of being sensitive to orientation: the same vector will have different ∞-norm values depending on its orientation with respect to whatever coordinate system is being used. Likewise, decomposing Problem 1 into one-dimensional problems means that some axes are special, and so trajectories will not, in general, be invariant under rotation.

To get around this difficulty, we use a radial coordinate system, in which one *radial* axis is parallel to $\Delta\mathbf{x}$ and a second *perpendicular* axis is chosen at right angles to this (within the plane formed by $\Delta\mathbf{x}$ and $\mathbf{v}$). Such a coordinate system will thus not be fixed, but will instead vary, from one trajectory cycle to another, depending on the direction of $\Delta\mathbf{x}$.

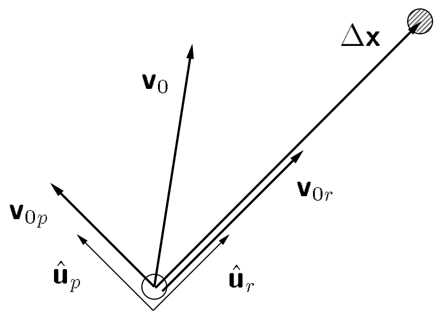

Figure 3: Radial coordinates used for the multi-dimensional filter. White and shaded circles represent $\mathbf{x}_0$ and $\mathbf{x}_d$, respectively

At the beginning of each cycle, the filter computes a new radial coordinate system (see Figure 3). This involves determining unit vectors $\hat{\mathbf{u}}_r$ and $\hat{\mathbf{u}}_p$ for the radial and perpendicular directions. If $\Delta\mathbf{x} \neq 0$, then $\hat{\mathbf{u}}_r := \Delta\mathbf{x}/\|\Delta\mathbf{x}\|$. Otherwise, if $\mathbf{v}_0 \neq 0$, $\hat{\mathbf{u}}_r := \mathbf{v}_0/\|\mathbf{v}_0\|$. If $\Delta\mathbf{x} = 0$ and $\mathbf{v}_0 = 0$, then the target has been reached, and $\mathbf{x}_1$ and $\mathbf{v}_1$ can be set to $\mathbf{x}_0$ and 0.

For the perpendicular direction, observe that $\mathbf{v}_0$ can be resolved into two components

$$\mathbf{v}_0 = \mathbf{v}_{0r} + \mathbf{v}_{0p}$$

where $\mathbf{v}_{0r}$ is parallel to the radial axis and $\mathbf{v}_{0p}$ is perpendicular to it. If $\|\mathbf{v}_{0p}\| = 0$, the problem reduces to a one-dimensional one and the perpendicular direction is not used. Otherwise, $\hat{\mathbf{u}}_p$ is set to $\mathbf{v}_{0p}/\|\mathbf{v}_{0p}\|$. Motion along the radial axis and motion perpendicular to it can now be considered as two one-dimensional subproblems. New position and velocity setpoints can be computed along each direction, with the results being summed to form $\mathbf{x}_1$ and $\mathbf{v}_1$.

Consider the radial direction first. Let x_{1r} and v_{1r} be the desired setpoint components in the radial direction. These can be computed using *filterOne()* with $\Delta x = \|\Delta \mathbf{x}\|$, $x_0 = 0$, and $v_0 = \|\mathbf{v}_{0r}\|$.

Now consider the perpendicular direction, with x_{1p} and v_{1p} being the desired setpoint components. If $\|\mathbf{v}_{0p}\| = 0$, then the perpendicular direction can be ignored, with $x_{1p} = v_{1p} = 0$. Otherwise, we *remove the perpendicular velocity component and return to the radial axis as fast as possible*. It is not sufficient to simply remove the velocity component, because that would leave $\mathbf{x}$ a distance $^1\!/_2\|\mathbf{v}_{0p}\|^2/A$ away from the radial axis. Given the above rule, x_{1p} and v_{1p} can be computed using *filterOne()* with $\Delta x = 0$, $x_0 = 0$, and $v_0 = \|\mathbf{v}_{0p}\|$, which will always result in a velocity profile like that shown in Figure 2(C). Because we want to remove the perpendicular velocity as fast as possible, the desired reference time is ignored by setting t_d to zero.

The radial and perpendicular setpoints are summed to yield $\mathbf{x}_1$ and $\mathbf{v}_1$, as shown in the complete description given in Algorithm 4.

5 Handling Rotations

Spatial rotations must be treated specially because they are not describable by vector quantities. However, the trajectory filter described above can stiil be applied.

It will be useful to describe rotations using matrix exponential notation [8]. Recall that any spatial rotation represented by a 3×3 matrix $\mathbf{R} \in SO(3)$ is equivalent to a rotation by some angle θ about an axis represented by a unit vector $\hat{\mathbf{w}}$. These representations are related by

$$\mathbf{R} \;=\; e^{[\hat{\mathbf{w}}]\theta} \quad \text{and} \quad \log \mathbf{R} \;=\; [\hat{\mathbf{w}}]\theta, \tag{5}$$

where $[\hat{\mathbf{w}}]$ is a 3×3 skew-symmetric matrix formed from the components of $\hat{\mathbf{w}}$.

For rotations, one may represent the initial and output setpoints by $\mathbf{R}_0, \boldsymbol{\omega}_0$ and $\mathbf{R}_1, \boldsymbol{\omega}_1$, respectively, and the target by $\mathbf{R}_d$, where $\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_d \in SO(3)$, and $\boldsymbol{\omega}_0, \boldsymbol{\omega}_1 \in \mathbf{R}^3$ are rotational velocity vectors.

The rotational displacement to the target is given by $\Delta\mathbf{R} \equiv \mathbf{R}_0^{-1}\mathbf{R}_d$. In accordance with (5), this displacement is associated with an axis $\hat{\mathbf{w}}$ and an angular displacement θ. Now, just as in Section 4, ω_0 can be resolved into components which are parallel and perpendicular to $\hat{\mathbf{w}}$. In fact, if the vector $\Delta\mathbf{w}$ is defined so that $\Delta\mathbf{w} \equiv \theta\hat{\mathbf{w}}$, then *filterMulti()* can be used directly, subject to the following changes:

funct *filterMulti*$(\Delta\mathbf{x}, \mathbf{x}_0, \mathbf{v}_0, t_d) \equiv$
  if $\|\Delta\mathbf{x}\| < \epsilon$
    then
      if $\|\mathbf{v}_0\| < \epsilon$
        then
          return$\{\mathbf{x}_0, 0\}$
        else
          $\hat{\mathbf{u}}_r := \mathbf{v}_0 / \|\mathbf{v}_0\|$
      fi
    else
      $\hat{\mathbf{u}}_r := \Delta\mathbf{x} / \|\Delta\mathbf{x}\|$
  fi
  $\mathbf{v}_{0r} := (\mathbf{v}_0 \cdot \hat{\mathbf{u}}_r)\hat{\mathbf{u}}_r$
  $\mathbf{v}_{0p} := \mathbf{v}_0 - \mathbf{v}_{0r}$
  if $\|\mathbf{v}_{0p}\| > \epsilon$
    then
      $\hat{\mathbf{u}}_p := \mathbf{v}_{0p} / \|\mathbf{v}_{0p}\|$
    else
      $\hat{\mathbf{u}}_p := 0$
  fi
  // Do radial computation:
  $\{x_{1r}, v_{1r}\} =$ *filterOne*$(\|\Delta\mathbf{x}\|, 0, \mathbf{v}_0 \cdot \hat{\mathbf{u}}_r, t_d)$
  $\mathbf{x}_1 := \mathbf{x}_0 + x_{1r}\hat{\mathbf{u}}_r$
  $\mathbf{v}_1 := v_{1r}\hat{\mathbf{u}}_r$
  // Do perpendicular computation if necessary:
  if $\hat{\mathbf{u}}_p \neq 0$
    then
      $\{x_{1p}, v_{1p}\} =$ *filterOne*$(0, 0, \|\mathbf{v}_{0p}\|, 0)$
      $\mathbf{x}_1 := \mathbf{x}_1 + x_{1p}\hat{\mathbf{u}}_p$
      $\mathbf{v}_1 := \mathbf{v}_1 + v_{1p}\hat{\mathbf{u}}_p$
  fi
  return$\{\mathbf{x}_1, \mathbf{v}_1\}$.

Algorithm 3: Multi-dimensional trajectory filter, computing new setpoints $\mathbf{x}_1, \mathbf{v}_1$.

1. Replace $\Delta\mathbf{x}, \mathbf{x}_0, \mathbf{v}_0, \mathbf{x}_1$ and $\mathbf{v}_1$ with $\Delta\mathbf{w}, \mathbf{R}_0, \boldsymbol{\omega}_0, \mathbf{R}_1$ and $\boldsymbol{\omega}_1$
2. Replace $\mathbf{x}_1 := \mathbf{x}_0 + x_{1r}\hat{\mathbf{u}}_r$ with $\mathbf{R}_1 := \mathbf{R}_0 \; e^{[\hat{\mathbf{u}}_r]x_{1r}}$
3. Replace $\mathbf{x}_1 := \mathbf{x}_1 + x_{1p}\hat{\mathbf{u}}_p$ with $\mathbf{R}_1 := \mathbf{R}_1 \; e^{[\hat{\mathbf{u}}_p]x_{1p}}$

A couple of comments are in order. First, because rotations do not commute, the value of $\mathbf{R}_1$ will change (in general) if the multiplication statements of items 2 and 3 are interchanged. However, if the incremental rotations described by $e^{[\hat{\mathbf{u}}_r]x_{1r}}$ and $e^{[\hat{\mathbf{u}}_p]x_{1p}}$ are sufficiently small, then this effect should not be significant. For the same reasons, the computed velocity setpoints will not be reachable by uniform acceleration along $\hat{\mathbf{u}}_r$ and $\hat{\mathbf{u}}_p$, but again, this effect will not be significant for small displacements.

Small displacements are ensured if the filter sample rate is high enough. This can be estimated from the velocity limit V, which in this case refers to angular velocity. Since displacements are unlikely to be much in excess of VT (but see Section 7), and a displacement of $\pi/10$ can be considered "small", one might wish to ensure that $T \leq \pi/(10V)$, where V is given in rad/sec.

6 Coordinating with Other Filters

The trajectory filter uses single norm bounds to represent the velocity and acceleration constraints. This is frequently appropriate, as in the case of either Cartesian translations or rotation. However, a single norm bound is generally *not* appropriate to apply to both Cartesian translations and rotations *together* (because the units are generally different). Likewise, different bounds may be required for different robot joints (or groups of joints, such as the distal joints vs. the proximal joints).

One must use a different filter for each object requiring a different set of velocity/acceleration bounds. If one then wishes these objects to move in a coordinated fashion, the following two pass procedure may be used: first, the profile times corresponding to $t_d = 0$ are determined for each object. Then, letting t_M be the maximum such profile time, a second pass is made over all the objects, in which the filter is called with $t_d = t_M$.

7 Performance and Stability

The filter described here has been implemented and is currently being used in live control situations involving a CRS A460 anthropomorphic manipulator. A common application involves having the robot track, in some Cartesian plane, the position of a mouse-driven cursor. The usual cycle rate is 100 Hz. Two choices are available to the operator: a continuous mode, in which the cursor position is tracked constantly, or a discrete mode, in which the cursor is used in conjunction with a "mouse click" to change the current target point. The filter is equally effective in both situations. Figures 4 and 5 give some illustration of the filter's behavior in the latter and former cases, respectively.

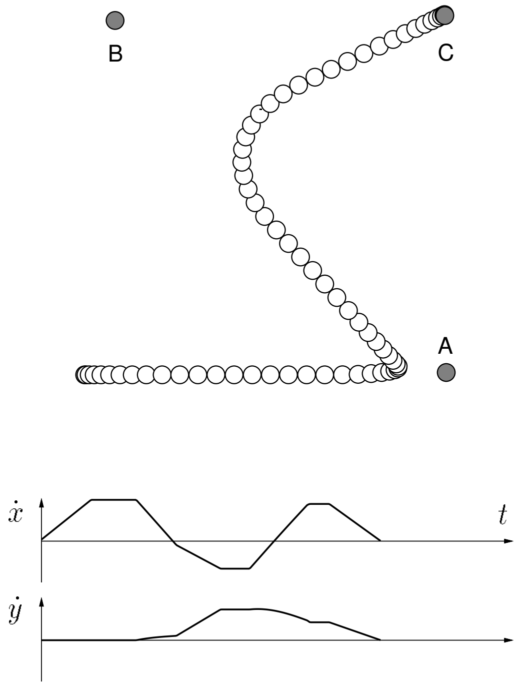

Figure 4: Path produced by the filter with a target that first appears at A, and then later shifts to B and then to C. Corresponding velocity profiles for $\dot{x}$ and $\dot{y}$ (in global coordinates) are shown below.

In general, we have observed that whenever a target becomes stationary, the perpendicular velocity components decay fairly quickly, and the motion turns into a straight line motion toward the target (although, as one might expect, this effect is less pronounced if the acceleration bound is low).

We conclude this section with some discussion of stability: notably, does the velocity remain bounded, and does the output always converge to a stationary target? If the filter operated in a fixed coordinate frame, there would clearly be no question of this: by construction, the velocity would remain bounded according to $\|\mathbf{v}\|_\infty \leq V$, and since the velocity profiles would not change for a stationary tar-

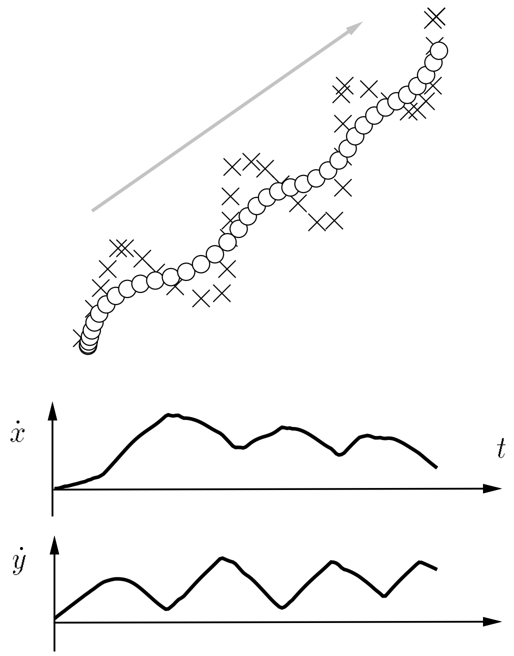

Figure 5: Path produced by the filter for a target position (indicated by the crosses) that is constantly changing. Target path was hand generated by dragging a mouse cursor across a computer screen. Velocity profiles are shown below.

get, convergence would occur within the maximum segment time t_s of either the radial or perpendicular component.

On the other hand, by working in a radial coordinate system, the velocity profiles will generally *not* remain the same from one cycle to the next (unless the perpendicular velocity has been eliminated and the motion has turned into a straight-line path to the target). Also, $\mathbf{v}$ is in effect bounded with an ∞-norm, whose value will change as the radial coordinate system rotates. As for converging to a stationary target, we should affirm that there do not exist conditions under which $\mathbf{x}$ will go into a limit cycle about $\mathbf{x}_d$.

Because of the discrete nature of the problem and the non-linear switching characteristics of the filter, proofs of these types of properties are difficult. We have, however, established the following:

Property 1 *If* $\mathbf{x}$ *is a vector quantity, the acceleration* $\dot{\mathbf{v}}$ *resulting from the action of filterMulti() is bounded by* $\|\dot{\mathbf{v}}\| \leq \sqrt{2}A$.

Proof: Within the radial coordinate system for any particular cycle, we have (by construction) $\|\dot{\mathbf{v}}\|_\infty \le A$. The result is then immediate, since (in two dimensions) $\|\dot{\mathbf{v}}\| \le \sqrt{2}\|\dot{\mathbf{v}}\|_\infty$.

Property 2 *If* $\mathbf{x}$ *is a vector quantity, the maximum possible increase in velocity between successive applications of filterMulti() is* AT*; i.e,* $\|\mathbf{v}_1\| \le \|\mathbf{v}_0\| + AT$*, where* T *is the filter sample interval.*

Proof: Given radial coordinate vectors $\hat{\mathbf{u}}_r$ and $\hat{\mathbf{u}}_p$ for a particular cycle, let $v_{0r} \equiv \mathbf{v}_0 \cdot \hat{\mathbf{u}}_r$ and $v_{0p} \equiv \mathbf{v}_0 \cdot \hat{\mathbf{u}}_p$ be the radial and perpendicular components of $\mathbf{v}_0$. Let v_{1r} and v_{1p} be similarly defined for $\mathbf{v}_1$ (with respect to the same coordinates). A characteristic of the filter is to always reduce the perpendicular velocity. More precisely: from the definition of $\hat{\mathbf{u}}_p$, $v_{0p} \ge 0$, and since the perpendicular velocity profile is always of the type shown in Figure 2(C) (with $\Delta x = 0$), it can be verified that $v_{1p} \in [\max(-V, -v_{0p}/\sqrt{2}), v_{0p})$, and thus $|v_{1p}| \le |v_{0p}|$. For the radial components, acceleration limits imply that $|v_{1r}| \le |v_{0r} + AT|$. Then, noting that $|v_{0r}| \le \|\mathbf{v}_0\|$ and $\|\mathbf{v}_1\|^2 = v_{1r}^2 + v_{1p}^2$, we have

$$\begin{aligned}\|\mathbf{v}_1\|^2 &\le (v_{0r} + AT)^2 + v_{0p}^2 \\ &\le \|\mathbf{v}_0\|^2 + 2AT|v_{0r}| + (AT)^2 \le (\|\mathbf{v}_0\| + AT)^2.\end{aligned}$$

A bound can also be proven for $\|\mathbf{v}\|$ (although there is no indication that this is tight):

Property 3 *If* $\mathbf{x}$ *is a vector quantity, the velocity* $\mathbf{v}$ *resulting from the action of filterMulti() is bounded by* $\|\dot{\mathbf{v}}\| \le \sqrt{2}V + AT$*.*

Proof: The idea is to show that $\|\mathbf{v}_1\| \le \|\mathbf{v}_0\|$ if $\|\mathbf{v}_0\| \ge \sqrt{2}V$. The main result then follows from Property 2.

$|v_{1p}| < |v_{0p}|$ (see previous proof), and so $\|\mathbf{v}_1\| > \|\mathbf{v}_0\|$ only if $|v_{1r}| > |v_{0r}|$. Now, if $v_{0r} \le 0$, then from the algorithm it can be established that $v_{1r} \in (v_{0r}, \min(V, |v_{0r}|)]$, and so $|v_{1r}| \not> |v_{0r}|$. Otherwise, if $v_{0r} > 0$, it can be established that $v_{1r} \in [-V, \max(V, v_{0r})]$, and hence $|v_{1r}| > |v_{0r}|$ is at best possible if $0 \le v_{0r} < V$. Furthermore, if $v_{1p} \le 0$, then $|v_{1p}| \le V$ (previous proof) and hence $\|\mathbf{v}_1\| = \sqrt{v_{1r}^2 + v_{1p}^2} \le \sqrt{2}V \le \|\mathbf{v}_0\|$.

Therefore, $\|\mathbf{v}_1\| > \|\mathbf{v}_0\|$ is at best possible if $v_{0p} > 0$ and $0 \le v_{0r} < V$. By hypothesis, $\|\mathbf{v}_0\| \ge \sqrt{2}V$, and since $\|\mathbf{v}_0\|^2 = v_{0r}^2 + v_{0p}^2$, it follows that $v_{0r} < V < v_{0p}$. Also, the requirement $|v_{1r}| > |v_{0r}|$ implies that $v_{1r} = v_{0r} + \Delta$, where $\Delta \in$

$(0, V - v_{0r}]$. Then $V < v_{0p}$ implies $\Delta < v_{0p} - v_{0r}$. Furthermore, $v_{1p} < v_{0p} - \Delta$, since an ability to increase v_{0r} by Δ implies an ability to decrease v_{0p} by Δ. Then:

$$\begin{aligned} \|\mathbf{v}_1\|^2 = v_{1p}^2 + v_{1r}^2 &\leq (v_{0p} - \Delta)^2 + (v_{0r} + \Delta)^2 \\ &\leq v_{0p}^2 + v_{0r}^2 + 2\Delta^2 - 2v_{0p}\Delta + 2v_{0r}\Delta \\ &\leq \|\mathbf{v}_0\|^2 + 2\Delta(\Delta - v_{0p} + v_{0r}). \end{aligned}$$

But $\Delta < v_{0p} - v_{0r}$ implies that $\Delta - v_{0p} + v_{0r} < 0$, and therefore $\|\mathbf{v}_1\|^2 < \|\mathbf{v}_0\|^2$, and we are done.

As for the possibility of $\mathbf{x}$ going onto orbit about $\mathbf{x}_d$, no evidence of this behavior has been observed, but we have not yet produced a formal proof. One approach would be to demonstrate that the perpendicular velocity component disappears after a finite number of steps, leaving only a one-dimensional problem, for which the convergence is immediate.

8 Conclusion

A simple non-linear filter has been presented which creates feasible trajectories of $\mathbf{x}$ to some target point $\mathbf{x}_d$, which may be varying in some completely unknown way from cycle to cycle. The key idea is to decouple the problem into a pair of one-dimensional problems in a radial coordinate system based on the remaining displacement to the target, an idea which can also be generalized to handle spatial rotations. The resulting velocities and accelerations are bounded by

$$\|\mathbf{v}\| \leq \sqrt{2}V + AT, \quad \|\dot{\mathbf{v}}\| \leq \sqrt{2}A, \tag{6}$$

where V and A are the ideal constraints given in (1).

The computational requirements for the algorithm are extremely small, as should be clear from its description.

Many questions about this approach remain, or which two prominent ones are described here.

First, what about higher order constraints, such as a jerk limitation $\|\ddot{\mathbf{v}}\| \leq J$? This could be effected by keeping $\dot{\mathbf{v}}$ as a third state. A three-dimensional radial coordinate system could then be formed from $\Delta\mathbf{x}$ and the appropriate perpendicular components of $\mathbf{v}$ and $\dot{\mathbf{v}}$. However, solving the one-dimensional version of this problem can be tedious, involving several cases and the solution of a 4-th order polynomial.

Second, for most manipulators, the true limits on velocity and acceleration can vary widely within the workspace. This effect is most pronounced for Cartesian motions. There is, however, no reason why A and V need to be fixed; they could instead be estimated locally within the workspace. The only catch in doing this is that when A drops, it means that it will take the filter longer to bring $\mathbf{x}$ to rest (resulting in overshoot hazards). Therefore, some estimate of the behavior of A over the path to the target would need to be considered.

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] P. K. Allen, A. Timcenko, B. Yoshimi, and P. Michelman, "Automated Tracking and Grasping of a Moving Object with a Robotic Hand-Eye System". *IEEE Transactions on Robotics and Automation*, April 1993, pp. 152–165 (Vol. RA-9, No. 2).

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

[3] R. H. Castain and R. P. Paul, "An On-Line Dynamic Trajectory Generator". *International Journal of Robotics Research*, Spring 1984, pp. 68 – 72 (Vol. 3, No. 1).

[4] C. S. Lin, P. R. Chang, and J.Y.S. Luh, "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots." *IEEE Transactions on Automatic Control*, December 1983, pp. 1066-1073 (Vol. AC-28, No. 12).

[5] J. E. Lloyd and V. Hayward, "Trajectory Generation for Sensor Driven and Time-Varying Tasks". *International Journal of Robotics Research*, August 1993, pp. 380-393 (Vol. 12, No. 4).

[6] Z. Lin, V. Zeman, and R.V. Patel, "On-line Robot Trajectory Planning for Catching a Moving Object". *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, Scottsdale, pp. 1726 – 1731.

[7] J. Y. S. Luh and C. S. Lin, "Optimum Path Planning for Mechanical Manipulators." *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, June 1981, pp. 142 – 151 (Vol. 102).

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

[9] M. Renaud and J. Y. Fourquet, "Time-Optimal Motions of Robot Manipulators Including Dynamics." *The Robotics Review 2*, (Khatib, Craig, and Lozano-Pérez, editors), MIT Press, 1992.

[10] 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).

[11] J. J. Slotine and H. S. Yang, “Improving the Efficiency of Time-optimal Path-following Algorithms.” *IEEE Transactions on Robotics and Automation*, February 1989, pp. 118 – 124 (Vol. RA-5, No. 1).

[12] R. Taylor, “Planning and Execution of Straight-line Manipulator Trajectories.” *IBM Journal of Research and Development*, July 1979, pp. 424 – 436 (Vol. 23, No. 4).

[13] S. E. Thompson and R.V. Patel, 1987. “Formulation of Joint Trajectories for Industrial Robots Using B-Splines”. *IEEE Transactions on Industrial Electronics*, Vol. IE-34, No. 2, pp. 192 – 199.