IEEE International Conference on Robotics and Automation, 1996

Proximate Time-Optimal Algorithm for On-Line Path Parameterization and Modification

Gerardo Pardo-Castellote* Robert H. Cannon Jr.†
Stanford Aerospace Robotics Laboratory
{pardo, cannon}@sun-valley.stanford.edu

Abstract

This paper presents an new, proximate-optimal solution to the path-constrained time-parameterization problem. This new algorithm has three distinguishing features: First, the run-time worst-case complexity of the proximate time-optimal algorithm is linear with respect to path-length and it is shown to be more efficient than any other truly time-optimal algorithm. Second, for a given robotic system, the algorithm's running-time is predictable as a function of the length of the path (allowing its use in combination with time-aware planners). Third, the algorithm easily supports the modification of on-going trajectories. The algorithm has been extensively tested and is operational in a number of robotic systems including a dual-arm workcell, an underwater robotic system, and the Marsokhod Rover vehicle. Experimental results presented illustrate the on-line use of the algorithm with a path planner to allow capture and delivery of objects from a moving conveyor belt.

1 Introduction

Automatic trajectory generation is integral to the practical utilization of robotic systems, especially semi-autonomous systems that use planners to compute the robot paths. If the system operates in a changing environment, these paths may need to be modified while a motion is in progress, hence the need for trajectory modification.

This work was motivated by the Stanford Intelligent Manufacturing Workcell [8, 10] shown in Figure 1. From a simple high-level command, this workcell can perform single-and-dual-armed object acquisition from a moving conveyor, and deliver the objects in a field cluttered with obstacles and other objects. An on-line planner [7] generates geometric paths, and the time-parameterization algorithm presented in this paper provides efficient, feasible trajectories for the manipulators.

Several general approaches to the trajectory-generation problem have been proposed: (a) Time-Parameterization of Geometric Paths (Decoupled Approach), (b) Combined Path-Planning and Time Parameterization (Combined Approach), (c) Reactive and Hybrid Methods. These methods are briefly described below. However, none of the existing methods is well-suited for the on-line generation and modification of robot trajectories.

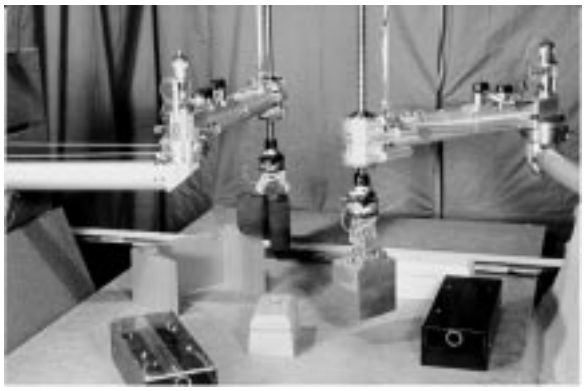

Figure 1: **Stanford Intelligent Manufacturing Workcell**

The workcell is shown capturing and delivering objects on a cluttered obstacle field. The arms are controlled from object-trajectories generated from planned via-point paths. Both via-point paths and trajectories are generated on-line. Interaction with a dynamic environment (e.g. capturing a moving object) requires the trajectories to be modified while the motion is in progress.

The Decoupled Approach was introduced by Bobrow, Dubowsky & Gibson [5], and Shin & McKay [16, 18, 17]. This approach breaks the overall problem in two parts: First a *geometric* path, described as a function of some parameter "s," $\mathbf{q} = \mathbf{f}(s)$ is obtained by some means (e.g. a path planner). Then the *geometric* path is time parameterized by finding the time history $s(t)$ that minimizes a pre-specified performance index subject to the dynamic constraints on the system[1]. This approach is conceptually simple and well suited to interface with standard geometric planning subsystems. However, the computational complexity of these algorithms is such that they have only been used off-line. To address their computational complexity, some authors have proposed trading strict optimality for efficiency [23, 3] while other authors have focused on increasing the efficiency of the optimal methods [21, 20].

The Coupled Approach was proposed by Gilbert & Johnson [4], Bobrow [2], and Shiller & Dubowsky [14, 15]. This approach, computes a collision-free path that is also optimal with respect to some performance index without going through an intermediate geometric path. This is the most general (and truly-optimal) approach because the shape of the path is also optimized in the process. However, its computational complexity is significantly higher than the decoupled approach, and requires

*Department of Electrical Engineering.
†Department of Aeronautics and Astronautics.

[1]This latter stage does not change the geometric path, hence the name decoupled.

co-development of the planner and time-parameterization modules which may preclude the use of many existing planners.

Reactive and Hybrid Methods never generate a path (or trajectory) explicitly. Rather, the desired state of the system is computed as a function of the current measured state and some external "virtual potential field". For instance the potential-function methods presented by Khatib [6] don't require á priori trajectory-generation. However, it is difficult to specify the desired (or pre-planned) system behavior. Nonetheless, there is promising research to allow construction of potential fields that guide a manipulator near a pre-specified path (see Rimon [12] and Arkin [1], and to provide incremental modification of existing paths [11].

This paper presents a new proximate-optimal algorithm of the decoupled approach category. The identification of a set of physically-meaningful, stricter constraints (beyond the ones imposed by the manipulator dynamics) allow the algorithm to parameterize the trajectory *efficiently* and *predictably*. The algorithm is **efficient** because, unlike all other decoupled algorithms, the integration is done on a single pass with no backtracking (each segment is integrated exactly once). This feature also makes the algorithm **predictable**: For a given robotic system, its running-time can be characterized as a linear function of the path length. Moreover, the algorithm presents a lower bound on the complexity of any optimal time-parameterization algorithm. These features make possible the modification (patching) of on-going trajectories (see Section 5). Experimental results can be found in Section 6, and in [8, 9].

2 Background

In this section, we summarize the classic formulation of the decoupled optimal time parameterization (DOTP) problem and collect some useful results. See [5, 16] for details and the derivation of the equations in this section.
The EOM of a manipulator with no friction can be written as:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{B}(\mathbf{q})[\mathbf{q},\mathbf{q}] + \mathbf{C}(\mathbf{q})[\mathbf{q}^2] + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \qquad (1)$$

Where $\boldsymbol{\tau}$ is the torque vector, $\mathbf{q}$ is the vector of generalized coordinates, $\mathbf{M}$, $\mathbf{B}$, $\mathbf{C}$, g are the mass matrix, matrix of coriolis terms, matrix of centripetal, and gravity matrix respectively.

Given a *geometric* path for this robot parameterized as a function of a parameter "s": $\mathbf{q} = \mathbf{f}(s)$ where $s \in [s_0, s_f]$, the DOTP is the search for the time evolution of the parameter $s = s(t)$, such that a pre-specified performance index $\mathcal{J}$[2] is minimized subject to the constraints imposed on the robot trajectory[3].

[2]Often minimum-time, but may also involve jerk, energy and other magnitudes.

[3]Maximum torque, velocities, accelerations, etc.

The approaches in the literature differ in their selection of performance index $\mathcal{J}$, the nature of the constraints imposed, and the method used to solve the optimization problem. This paper focuses on the case where total-travel time is the performance index and the only constraints are limits on actuator velocity, acceleration, and torque. We refer to this as the Decoupled Minimum-Time Time-Parameterization problem (DMTTP).

For a given geometric path $\mathbf{q} = \mathbf{f}(s)$, the dynamic equations of the robot (1) can be expressed as $\boldsymbol{\tau}(s,\dot{s}) = \mathbf{m}(s)\ddot{s} + \mathbf{c}(s)\dot{s}^2 + \mathbf{g}(s)$. From this equation, velocity, acceleration and torque limits may be expressed as the inequalities:

$$\begin{array}{l} 0 \le \dot{s} \le \dot{s}_{max}(s) \\ \alpha_{min}(s,\dot{s}) \le \frac{d\dot{s}}{ds} = \frac{\ddot{s}}{\dot{s}} \le \alpha_{max}(s,\dot{s}) \end{array} \qquad (2)$$

Furthermore, assuming that the torque limits are such that the manipulator can hold its own weight at any point along the trajectory: $\forall s : \boldsymbol{\tau}_{min}(s,0) \le \mathbf{g}(s) \le \boldsymbol{\tau}_{max}(s,0)$ we can assert that $\forall s : \alpha_{min}(s,0) \le 0 \le \alpha_{max}(s,0)$. The phase-space constraints in (2) can be viewed as a "wedge" associated with each point $(s,\dot{s})$ in phase space. This "wedge" represents the range of allowed slopes of any trajectory that goes through that phase-space point ($\dot{s} = \dot{s}(s)$) and locally satisfies the constraints. Several authors [5, 19] have noted that for each value of s there are values of $\dot{s}$ for which the wedge closes (i.e. $\alpha_{min}(s,\dot{s}) > \alpha_{max}(s,\dot{s})$) meaning there is no phase space trajectory that goes through that point $(s,\dot{s})$ and satisfies the constraints. Furthermore, in [19] the authors prove that under fairly general assumptions[4], the "allowed" region for $\dot{s}$ has the form $0 \le \dot{s}_{max}(s)$. We can therefore redefine $\dot{s}_{max}(s)$ to ensure that, in addition to (2), we also have:

$$0 \le \dot{s} \le \dot{s}_{max}(s) \quad \Rightarrow \quad \alpha_{min}(s,\dot{s}) \le \alpha_{max}(s,\dot{s})$$

Figure 3 illustrates these phase-space constraints for a planner-generated geometric path for one of the 4-DOF manipulators in the workcell.

Among the useful results associated with the DMTTP problem, the following lemma–proven in [5]–will be used in the description of the proximate-optimal algorithm.

Lemma 1 *Let* $\{[s_0,s_f], \dot{s}(s_0), \dot{s}(s_f), \alpha_{min}, \alpha_{max}, \dot{s}_{max}\}$ *be an instance on the DMTTP problem and* $\dot{s}^*(s)$ *be its solution. Then for any function* $\dot{s}(s)$ *that satisfies the constraints of the problem, we have* $\dot{s}(s) \le \dot{s}^*(s), \forall s \in [s_0,s_f]$.

3 Proximate DMTTP problem

This section presents the theoretical foundation of the proximate-optimal algorithm. The algorithm gains its efficiency from transforming the DMTTP problem into one with

[4]These result valid for a manipulator modelled without friction. The only assumption made in the paper is that the torque limits have a dependency on the joint velocities $\dot{\mathbf{q}}^i$ that is at most quadratic in them.

stricter constraints, and then computing the optimal solution to this modified problem. In many cases these stricter constraints will result in trajectories that are not significantly slower [5].

The key to deriving a predictable, $\mathcal{O}(L)$ algorithm is the identification of a criterion that allows the phase-space integration to be divided into several independent sections. A side benefit is that the algorithm will be highly parallel. Note that an instance of the DMTTP problem (and hence its solution $\dot{s}^*(s)$), is completely determined by the boundary conditions $\{\dot{s}_0, \dot{s}_f\}$ and the constraint functions $\{\alpha_{min}(s,\dot{s}), \alpha_{max}(s,\dot{s}), \dot{s}_{max}(s)\}$. Therefore, if we knew in advance any point in the optimal path $\{s_d, \dot{s}^*(s_d)\}$ with $s_0 < s_d < s_f$, we would be able to divide the problem into two independent ones: first solve for $s_0 \leq s \leq s_d$ and then for $s_d \leq s \leq s_f$. In other words, the value $\dot{s}^*(s_d)$ provides the boundary condition at $s = s_d$ that allows the problem to be divided. Any point along the optimal phase-space trajectory, $\dot{s} = \dot{s}^*(s)$ can be used in this manner. Figure 3 shows one such phase-space trajectory. The following characterization of a subset of the points in the optimal trajectory allows early identification of these *decoupling* points:

Theorem 1 *Let* $\{[s_0, s_f], \dot{s}(s_0), \dot{s}(s_f), \alpha_{min}(s,\dot{s}), \alpha_{max}(s,\dot{s}), \dot{s}_{max}(s)\}$ *be an instance of the DMTTP problem and* $\dot{s}^*(s_d)$ *its solution. Assume that* $\forall s \in [s_0, s_f]$*:*

$$\dot{s} \leq \dot{s}_{max}(s) \quad \Rightarrow \quad \alpha_{min}(s,\dot{s}) \leq 0 \leq \alpha_{max}(s,\dot{s}) \qquad (3)$$

Then the following property (see Figure 2) holds: For all $s_1, s_2, s_d \in [s_0, s_f]$:

$$\left.\begin{array}{l} s_1 < s_d < s_2 \\ \dot{s}_{max}(s_d) = \min_{s_1 \leq s \leq s_2}\{\dot{s}_{max}(s)\} \\ \dot{s}^*(s_1) = \dot{s}_{max}(s_d) = \dot{s}^*(s_2) \end{array}\right\} \quad \Rightarrow \quad \dot{s}^*(s_d) = \dot{s}_{max}(s_d)$$

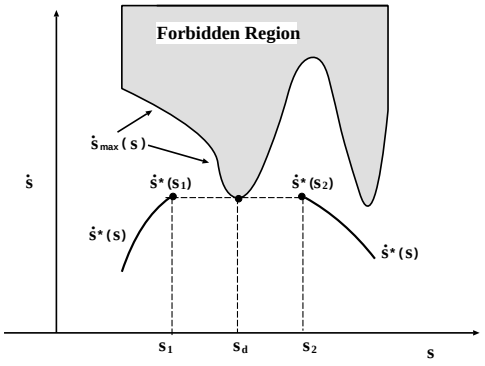

Figure 2: **Illustration of conditions of Theorem 1**

Proof. It suffices to show that the phase-space trajectory defined by Equation (4) below satisfies all the constraints.

$$\dot{s}(s) = u(s) \stackrel{\text{def}}{=} \begin{cases} \dot{s}_{max}(s_d) & \text{for } s_1 < s < s_2 \\ \dot{s}^*(s) & \text{otherwise} \end{cases} \qquad (4)$$

[5] This has been observed empirically. Further research is needed to precisely characterize the performance loss with respect to the true time-optimal path.

Once the above statement is proven, applying Lemma 1, we see that $\dot{s}^*(s_d) \geq u(s_d) = \dot{s}_{max}(s_d)$ which combined with the constraint $\dot{s}^*(s_d) \leq \dot{s}_{max}(s_d)$ implies $\dot{s}^*(s_d) = \dot{s}_{max}(s_d)$.

In view of its definition, we only need to show that $u(s)$ satisfies the constraints in the interval $]s_1, s_2[$. Now, in this interval, our hypothesis guarantees $\dot{s}(s) = \dot{s}_{max}(s_d) \leq \dot{s}_{max}(s)$, and since $\dot{s}(s)$ is constant $\frac{d\dot{s}}{ds} = 0$, and therefore, $\alpha_{min}(s,\dot{s}) \leq 0 = \frac{d\dot{s}}{ds} \leq \alpha_{max}(s,\dot{s})$.

Corollary 1 *The theorem holds even if we relax the equality* $\dot{s}^*(s_1) = \dot{s}_{max}(s_d) = \dot{s}^*(s_2)$ *to* $\dot{s}^*(s_1) \geq \dot{s}_{max}(s_d) \leq \dot{s}^*(s_2)$

Proof. Since $\dot{s}^*(s)$ is continuous and $\dot{s}^*(s_d) \leq \dot{s}_{max}(s_d)$, the intermediate value theorem guarantees that there are values $\tilde{s_1}, \tilde{s_2}$ such that $s_1 \leq \tilde{s_1} \leq s_d \leq \tilde{s_2} \leq s_2$ with $\dot{s}^*(\tilde{s_1}) = \dot{s}_{max}(s_d) = \dot{s}^*(\tilde{s_2})$. We can now apply the theorem to $\tilde{s_1}, \tilde{s_2}, s_d$.

The above theorem and its corollary provide a sufficient condition for a phase-space point $\{s, \dot{s}_{max}(s)\}$ to belong to the optimal phase-space trajectory $\dot{s}^*(s)$. This characterization only applies when we can guarantee that the following pre-condition holds:

$$\{\dot{s} \leq \dot{s}_{max}(s) \quad \Rightarrow \quad \alpha_{min}(s,\dot{s}) \leq 0 \leq \alpha_{max}(s,\dot{s})\}$$

This condition can always be enforced by redefining $\dot{s}_{max}(s)$ to be:

$$\dot{s}_{max}(s) \leftarrow \min \left\{ \begin{array}{l} \dot{s}_{max}(s) \\ \min\{\dot{s} \mid (\alpha_{min}(s,\dot{s}) = 0) \vee (\alpha_{max}(s,\dot{s}) = 0)\} \end{array} \right.$$

This more restrictive (proximate-optimal) constraint imposed on the allowable trajectories physically means that we require enough authority left in the actuators at every state in the trajectory that the system is able to change its speed along the trajectory in either direction: $\ddot{s} > 0$ (increase in speed), and $\ddot{s} < 0$ (decrease in speed)[6].

The fact that the optimal trajectory touches the boundary curve $\dot{s} = \dot{s}_{max}(s)$ at a finite number of points is also exploited in [5, 19, 21]. Reference [19] shows that for the case in which there are no limits in $\mathbf{q}$ (and therefore the boundary $\dot{s}_{max}(s)$ is given by the equation $\alpha_{min}(s,\dot{s}) = \alpha_{max}(s,\dot{s})$) the "switching points" satisfy the necessary condition $\frac{d\dot{s}_{max}(s)}{ds} = \alpha_{min}(s, \dot{s}_{max}(s))$. The algorithm presented in [21] exhaustively classifies these points and presents an efficient method to calculate them. The proximate-optimal approach differs from the above in that we have obtained a *sufficient* condition. This is key to reducing the algorithmic complexity as discussed in section 7.

4 Proximate-Optimal Algorithm

This section describes the proximate-optimal algorithm and proves its correctness in the continuous domain. The discrete

[6] This interpretation assumes the parameter "s" is either the path length or related by a strictly monotonic (increasing) function to the path length. This is the usual case.

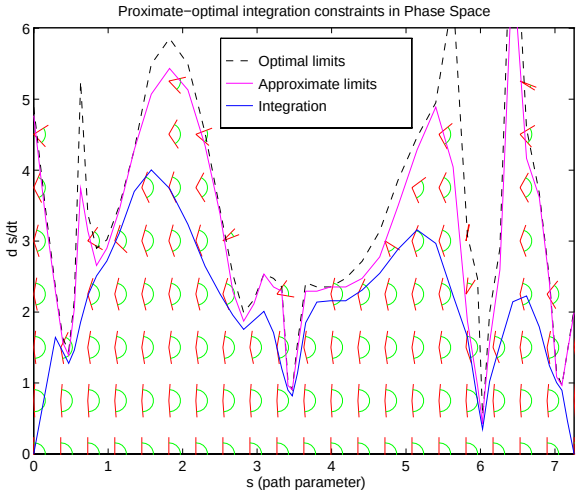

Figure 3: **Integration trajectory in phase space using proximate-optimal constraints**

This figure shows the constraints on the phase-space trajectories that correspond to the manipulator dynamic constraints for trajectories along the given geometric path. Velocity, acceleration, and torque limits map into two types of phase-space constraints: (1) an allowed region that verifies $\dot{s} \leq \dot{s}_{max}(s)$ and (2) "slope" limits $\alpha_{min}(s,\dot{s}) \leq \frac{d\dot{s}}{ds} \leq \alpha_{max}(s,\dot{s})$. At each point $(s,\dot{s})$ in phase-space, there is a "wedge" representing the range of allowed slopes of legal phase-space trajectories trough that point. The strict (optimal) limit occurs when the "wedge" "closes". The proximate-optimal limit precludes in addition the phase-space regions where the "wedge" does not allow zero slope phase-space trajectories.

implementation can be found in [9].

Assume an instance of the DMTTP problem:

$$\{[s_0, s_f], \dot{s}_0, \dot{s}_f, \alpha_{min}(s,\dot{s}), \alpha_{max}(s,\dot{s}), \dot{s}_{max}(s)\}$$

that satisfies the requirement (3) in Theorem 1.

The algorithm to integrate $\dot{s} = \dot{s}_a(s)$ consists of the three steps illustrated in Figure 4:

1. Initialization. Let

$$\mathcal{H} = \{s \in [s_0, s_f] \quad | \quad \dot{s}_{max}(s) \text{ is a local minimum}\}$$
$$\text{And, } s_l \leftarrow s_0,\ \dot{s}_l \leftarrow \dot{s}_0,\ s_r \leftarrow s_f,\ \dot{s}_r \leftarrow \dot{s}_f$$

2. Integration. Let

$$s_1 \leftarrow s_l,\ \dot{s}_a(s_1) \leftarrow \dot{s}_l,\ s_2 \leftarrow s_r,\ \dot{s}_a(s_2) \leftarrow \dot{s}_r$$
$$\mathcal{H} = \mathcal{H} \cap]s_l, s_r[$$
$$s_d \leftarrow \{s_p \in \mathcal{H} \quad | \quad \dot{s}_{max}(s_p) = \min_{s\in\mathcal{H}}\{\dot{s}_{max}(s)\}\}$$
$$\dot{s}_a(s_d) \leftarrow \dot{s}_{max}(s_d)$$

While $[s_1 < s_2] \wedge [\{\dot{s}_a(s_1) < \dot{s}_a(s_d)\} \vee \{\dot{s}_a(s_2) < \dot{s}_a(s_d)\}]$ **Do :**

If $\dot{s}_a(s_1) \leq \dot{s}_a(s_2)$ integrate forward (i.e. increasing s_1) along the maximum acceleration curve:

$$\frac{d\dot{s}_a}{ds}(s_1) = \begin{cases} \alpha_{max}(s_1, \dot{s}_a(s_1)) & \textbf{if} \quad \dot{s}_a(s_1) < \dot{s}_{max}(s_1) \\ \min\{\frac{d\dot{s}_{max}}{ds}(s_1), \alpha_{max}(s_1, \dot{s}_a(s_1)\ \}\ , & \textbf{otherwise} \end{cases} \tag{5}$$

Else $\dot{s}_a(s_1) > \dot{s}_a(s_2)$ and we integrate backward (decreasing s_2) along the maximum deceleration curve:

$$\frac{d\dot{s}_a}{ds}(s_2) = \begin{cases} \alpha_{min}(s_2, \dot{s}_a(s_2)) & \textbf{if} \quad \dot{s}_a(s_2) < \dot{s}_{max}(s_2) \\ \min\{\frac{d\dot{s}_{max}}{ds}(s_2), \alpha_{min}(s_2, \dot{s}_a(s_2))\ \}\ , & \textbf{otherwise} \end{cases}$$

At any point in the integration, if any element of $\mathcal{H}$ falls outside the (changing) interval $]s_1, s_2[$, we eliminate it from $\mathcal{H}$ and recompute s_d if required.

The condition $s_1 = s_2$ indicates the integration between s_l and s_r has completed and we return. The condition $[\dot{s}_a(s_1) \geq \dot{s}_a(s_d)] \wedge [\dot{s}_a(s_2) \geq \dot{s}_a(s_d)]$ indicates we can break the problem into two independent ones and we continue with the next step.

3. Separation. Here we know that $[\dot{s}_a(s_1) \geq \dot{s}_d] \wedge [\dot{s}_a(s_2) \geq \dot{s}_d]$. We divide the problem into the following two:

i) $s_l \leftarrow s_1,\ \dot{s}_l \leftarrow \dot{s}_a(s_1),\ s_r \leftarrow s_d,\ \dot{s}_r \leftarrow \dot{s}_d = \dot{s}_{max}(s_d)$

ii) $s_l \leftarrow s_d,\ \dot{s}_l \leftarrow \dot{s}_d = \dot{s}_{max}(s_d),\ s_r \leftarrow s_2,\ \dot{s}_r \leftarrow \dot{s}_a(s_2)$

And then invoke (recursively) the integration step on each one of the subproblems.

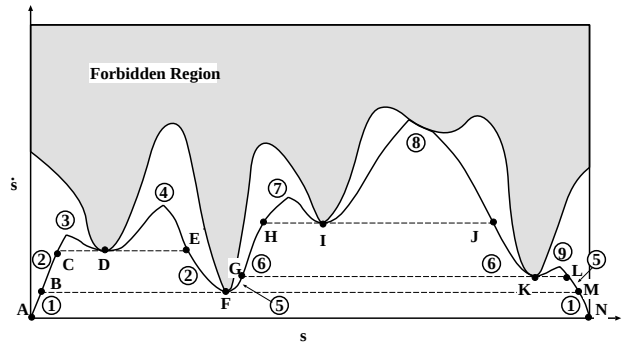

Figure 4: **Steps of proximate-optimal algorithm**

Phase-space illustration of the steps taken by the proximate-optimal algorithm to integrate a trajectory. Same numbered pieces are integrated simultaneously. Initially integration proceeds forward from A and backwards from N, keeping both branches balanced (1). As soon as the first local-minima is reached (F), the trajectory is broken into two independent pieces: B–F and F–M. B–F is integrated first (2), until another local minimum is reached at D. From here C–D is integrated (3) finishing that branch and D–E is integrated (4) completing the B–F branch. The right branch is integrated similarly after been separated at points K and I. Notice the possible existence of intervals where the integral follows the boundary of the forbidden region (8).

These steps are sketched in Figure 4. An example with the workcell manipulators can be seen in Figure 3. Each local minimum of the solution $\dot{s}_a(s)$ served as a decoupling point during the integration.

4.1 Algorithm correctness

This section proves that the algorithm integrates the "optimal[7]" phase-space trajectory i.e. $\dot{s}_a(s) = \dot{s}^*(s)\ \forall s \in [s_0, s_f]$.

It suffices to show that, given boundary conditions $(s_l, \dot{s}_l)$ and $(s_r, \dot{s}_r)$ in the optimal trajectory, the integration step always generates points in the optimal trajectory. Once we show this, Theorem 1 guarantees that the separation step generates boundary conditions in the optimal trajectory.

It is clear by construction that $\dot{s}_a(s) \geq \dot{s}'(s)$ for any function $\dot{s}'(s)$ that satisfies the constraints and the boundary conditions. In view of Lemma 1 it is sufficient to prove that $\dot{s}_a(s)$ itself satisfies the constraints.

Clearly by construction $\dot{s}_a(s) \leq \dot{s}_{max}(s)\ \forall s \in [s_l, s_r]$, so this constraint is always satisfied. This is because of the way the integration (see Equation (5)) changes the value of $\frac{d\dot{s}_a}{ds}$, to always be smaller than $\frac{d\dot{s}_{max}}{ds}(s)$ whenever $\dot{s}_a(s)$ intersects the boundary curve $\dot{s}_{max}(s)$.

So all we need to prove is that $\frac{d\dot{s}_a}{ds}$ does not violate the slope limits. This will proven through contradiction. Let $s_v \in [s_l, s_r]$ be the first value of s for which $s_a(s)$ violates the slope constraints. Given the way we choose $\frac{d\dot{s}_a}{ds}(s)$ to be either $\alpha_{max}(s_v, \dot{s}_a(s_v))$ or $\alpha_{min}(s_v, \dot{s}_a(s_v))$ whenever $\dot{s}_a(s_v) < \dot{s}_{max}(s_v)$, the only way the constraint $\alpha_{min}(s_v, \dot{s}_a(s_v)) \leq \dot{s}_a(s_v) \leq \alpha_{max}(s_v, \dot{s}_a(s_v))$ can be violated is at a point where $\dot{s}_a(s_v) = \dot{s}_{max}(s_v)$. Given how the integration (Equation (5)) chooses $\frac{d\dot{s}_a}{ds}(s_v)$, a slope violation can only occur if either during forward integration we have:

$$s_1 = s_v \quad \text{and} \quad \frac{d\dot{s}_a}{ds}(s_v) = \frac{d\dot{s}_{max}}{ds}(s_v) < \alpha_{min}(s_v, \dot{s}_a(s_v))$$

or during backward integration we have:

$$s_2 = s_v \quad \text{and} \quad \frac{d\dot{s}_a}{ds}(s_v) = \frac{d\dot{s}_{max}}{ds}(s_v) > \alpha_{max}(s_v, \dot{s}_a(s_v))$$

This will be shown to be impossible for the forward integration case; the backward integration case being analogous. First we must realize that the terminating conditions of the integration step and our selection of $\mathcal{H}$ and s_d guarantee that the invariant $\forall s \in]s_1, s_2[: \dot{s}_a(s_1) \leq \dot{s}_{max}(s) \leq \dot{s}_a(s_2)$ holds at all times during the integration. Then it is obvious that $\dot{s}_a(s_v) = \dot{s}_{max}(s_v)$, $s_1 < s_2$ and $\frac{d\dot{s}_{max}}{ds}(s_v) < \alpha_{min}(s_v, \dot{s}_a(s_v)) \leq 0$ violate this invariant. This contradicts our assumptions, and therefore there is no point s_v where the constraint is violated.

5 Trajectory Modification

Figure 5 illustrates our concept of patching an ongoing trajectory: The remaining piece of the geometric path beyond a certain point (called the *patch point*) is replaced by a new piece (the *patch path*). As a result, the trajectory which had already started needs to be modified. The modification starts at the *merge point*, located between the current and the patch point. The original trajectory remains unchanged until the *merge time* (time when the trajectory reaches the merge point). From there on, the patch trajectory is followed.

[7] Optimal with respect to the modified constraints.

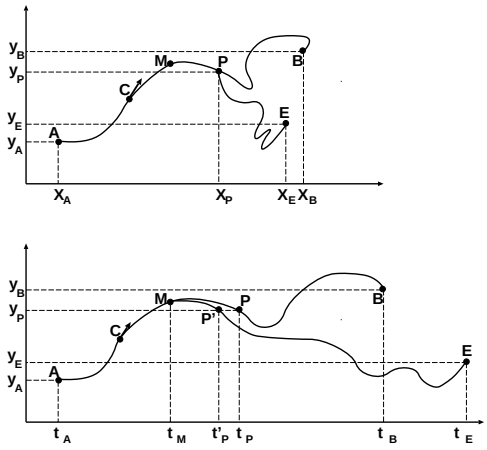

Figure 5: **Patching of an ongoing trajectory**

The top diagram shows the original and patched geometric paths in two dimensions (X and Y). The diagram underneath illustrates the corresponding trajectories for one of the coordinates (there is a corresponding one for every coordinate). The initial geometric path joined points A and B. The trajectory is currently at point C, when a new patch replaces the P-B section by the P-E patch. The resulting trajectory takes the system from C to E. We see in the bottom diagram that, even though the geometric path is the same until point P is reached, the trajectories start to differ at some intermediate point M between C and P. Point P is called the **patch point**, *point M is the* **merge point**, *and t_M the* **merge time**. *Notice how the patch point P is reached at a time "t'_p" different from the original t_p.*

Trajectories cannot be arbitrarily patched. In particular, given the initial geometric path and trajectory, the patch to the geometric path, and the current state, the trajectory-modification algorithm must determine (a) whether the patch is feasible without violating the dynamic constraints on the system, and (b) an appropriate (optimal) merge time.

The proximate-optimal algorithm provides efficient mechanisms to address the above issues. The operation of the trajectory-modification algorithm is essentially identical to the regular proximate-optimal algorithm except we start integrating backwards from the end of the trajectory, and whenever a decoupling point is reached, the left (earlier) piece of trajectory is computed first. The process can be seen in the phase-space plots of Figure 6.

Two things are worth noting in the example of Figure 6: First, the determination of the feasibility of the path is almost immediate due to the existence of a fairly strict constraint (local minimum K) in the original trajectory between the current and patch points (C-P piece). Second, the original trajectory is never recomputed (i.e. the piece from C to P is used without change or extra effort). In general, only the piece from the last decoupling point before the patch (point L) to the patch may require re-computation.

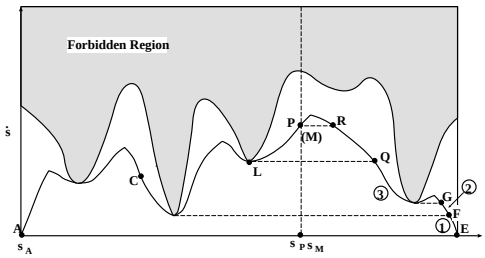

Figure 6: **Trajectory modification algorithm**

This example illustrates several optimizations that often occur when the patch is added far ahead of the current point. The patch point **P** *and the trajectory up to that point (A to P) are already computed, and the trajectory is being followed. The system is currently at point* **C** *(already accounting for the computational delay). The trajectory-modification algorithm starts at* **E** *along (1). As soon as the integral curve reaches* **F***, we know that the patch is feasible because there is a horizontal line that reaches the original trajectory between C and P without crossing the boundary region. The integration proceeds along (2) and is immediately separated into two pieces at point G. The left piece is selected first (3). The integration proceeds, until the point* **Q** *where a horizontal line can be drawn to* **L** *(the last decoupling point in the original trajectory before the patch point P). The remaining piece is integrated until either we can join the integral curve with the patch point with a horizontal line (point R), or else the integral curve intersects the original trajectory. The different independent pieces left are integrated as usual. Note that in this case the merge point* **M** *happens to just be the patch point P.*

6 Experimental Results

To compute these trajectories, the proximate-optimal algorithm uses the equations of motion of the two 4-DOF SCARA manipulators. The torque limits for the shoulder and elbow actuators is $9.5 Nm$. The remaining degrees-of-freedom (Z and Yaw) are not shown in the figures for simplicity but they are also parameterized.

Figure 7 illustrates the result of time-parameterizing a planner-generated path for two manipulators simultaneously. These paths are just like the single-arm paths, except the via-points correspond to an 8^{th}-dimensional space (four degrees-of-freedom per manipulator), and the dynamic equations of motion (EOM) correspond to the concatenation of the EOM for each manipulator[8].

Trajectory Modification results are difficult to illustrate with the dual-arm manipulator because the paths are quite short, and the resulting modifications become too cluttered to be easily visualized. For this reason, we present results for an imaginary system with equations of motion that correspond to those of a $1\,Kg$ mass. Only force limits of $1.5\,N$ are used[9].

Figure 9 illustrates the first modification to the on-going trajectory. As seen in the third plot, the trajectory is modified at

[8]This is a general feature of the time-parameterization algorithm, all it really needs is the sequence of via-points in some suitable "configuration-space," along with the corresponding equations-of-motion mapping from position, velocity and acceleration in this space to actuator torques.

[9]This is equivalent to acceleration limits of $1.5\,m/s^2$.

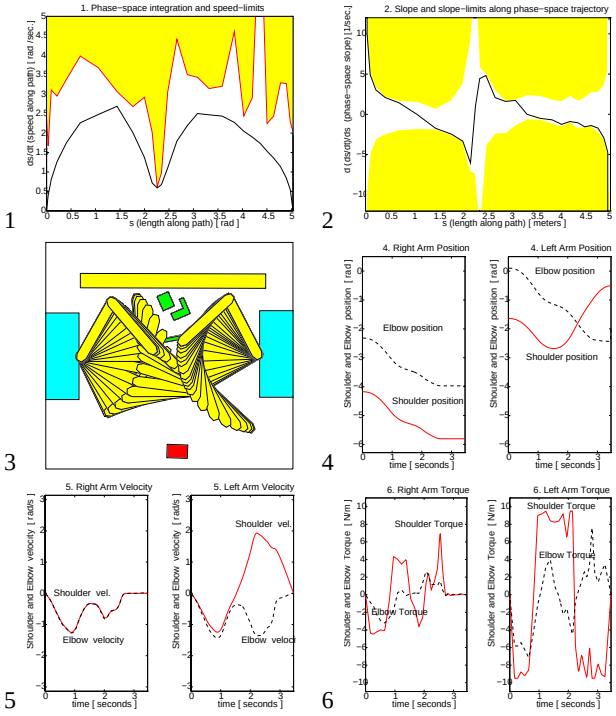

Figure 7: **Time-parameterization of a dual-arm path**

A coordinated motion for both manipulators is shown in the 3rd figure. The first two plots illustrate that the phase-space trajectory satisfies all limits and, as expected, switches between the two slope limits (second plot). The last two plots illustrates that all accelerations are within their limits of $\pm 9.5\ Nm$*. Note that the optimal trajectory would always keep at least one actuator saturated (at the maximum or minimum torque). The proximate-optimal algorithm comes close to this result as seen in the last plot.*

time $7\ sec.$, and the modification changes the trajectory beyond $12.3\ sec$. This modification causes the algorithm to recompute the remaining beyond the current point. This computation is depicted in the phase-space plots (first and second plot) of Figure 9. The modified trajectory beyond the *patch point* has smaller curvatures, and as a result, the phase-space limits (first plot) are higher. The velocity and accelerations of the modified trajectory are shown in the fifth and sixth plots, and they exhibit the expected alternation between maximum and minimum acceleration for each degree-of-freedom.

7 Complexity and Predictability

The algorithm presented (and its discrete implementation) have running times that are proportional to the length of the path (i.e it is $\mathcal{O}(L)$). Moreover, given the equations of motion, run-time of proximate-optimal algorithm is predictable, as a function of the length of the path. These characteristics stem from the fact that, as described in Figure 4, the proximate-optimal algorithm integrates along the path exactly once, without iterating or backtracking (a more formal proof can be found

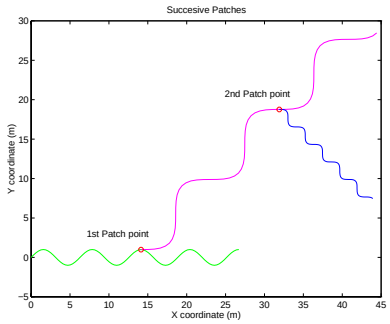

Figure 8: **Initial path and two subsequent modifications**

Illustration of the initial path and each one of the patches as given to the time-parameterization algorithm. These patches occur while the trajectory is being executed.

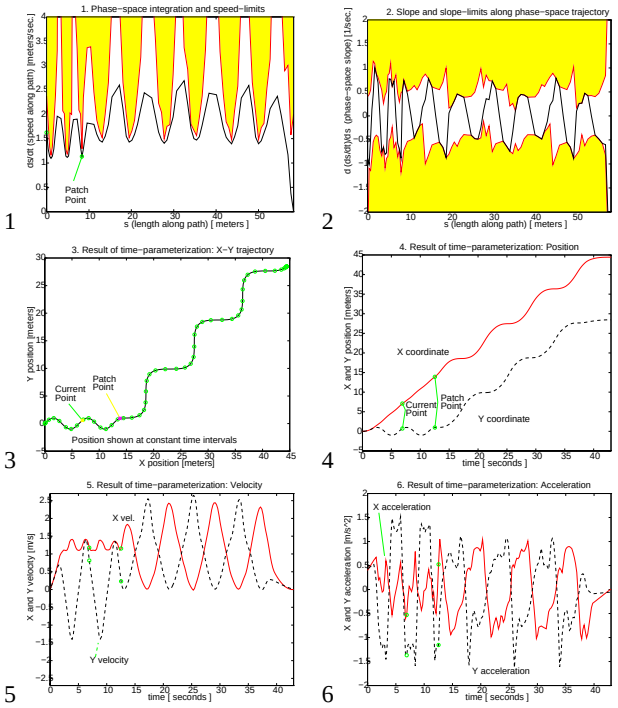

Figure 9: **Results after first modification to on-going trajectory**

The 3rd plot illustrates the current point in the trajectory when it is modified beyond the patch point. *The constraints (first two plots) only change beyond the patch point. The trajectory, however, may change at any point beyond the current point. The full trajectory, after the modification is made, is shown in plots 4, 5, and 6.*

in [9]).

Figure 10 shows the execution-time dependency with the number of via points and degrees of freedom for sequences of via points of increasing length. The sequences of via points used to evaluate the computational complexity of the algorithm

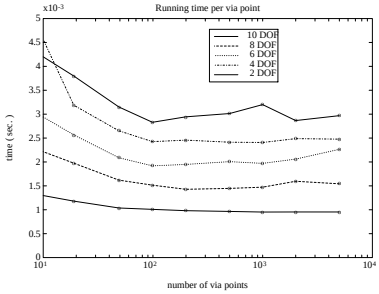

Figure 10: **Running time versus number of via points for different number of degrees of freedom**

This figure illustrates that the time-complexity of the algorithm is linear with the number of via points (or path length). We show that the execution time per via point is approximately constant over a 3 order of magnitude change in the number of via points. The timing corresponds to a Sparc station 2.

must truly represent paths of increased length[10]. In addition for the comparisons to be meaningful, the paths must be ergodic in their geometric properties. Such paths have been generated by filtering a random walk.

Lack of space precludes us from addressing the worst-case complexity of the approaches proposed in the literature. Rather, a brief justification and comparison is presented in Figure 11.

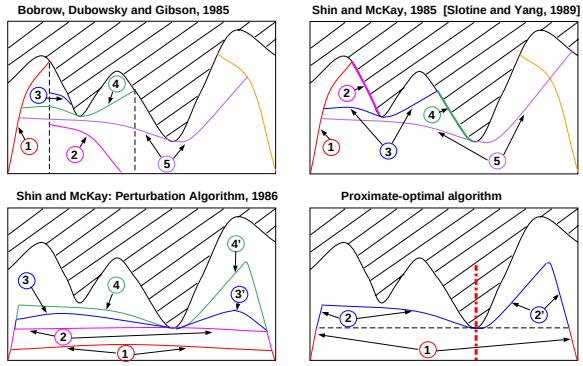

Figure 11: **Comparison of approaches to time-parameterization**

This figure compares the proximate-optimal algorithm with several classical optimal algorithms. Each call number indicates a sequential stage in the algorithm. Notice that the proximate-optimal algorithm is the **only** *one that never integrates the same region twice. From this comparison it is clear that it is "minimal" in the number of integration steps. Note that, for the top two algorithms, whenever the accelerating (decelerating) trajectory intersects the boundary region, it is necessary to search for a suitable switching point along the boundary region, and then backtrack until the original integral is met. Backtracking can take us past the region previously integrated, all the way back to the initial trajectory. As a result, the algorithm may end-up integrating the same piece over and over resulting on $\mathcal{O}(L^2)$ worst-case complexity (see Appendix F of [9]).*

[10] As opposed to representing the same geometric path sampled with greater density of via-points.

8 Conclusions

This paper has described a new (proximate-optimal) algorithm to time-parameterize geometric paths described as sequences of via points. The trajectory is proximate-time-optimal, subject to dynamic constraints that can be any combination of (configuration dependent) velocity, acceleration, or torque limits.

The algorithm gives up strict optimality by imposing more strict (yet physically meaningful) constraints on top of the regular dynamic constraints. The proximate-optimal algorithm achieves efficient, predictable performance (run-time linear with respect to the number of via points), that enables its use online. The algorithm is also well suited to allow modifications (patching) of trajectories already in progress.

The predictability and performance of the algorithm has been evaluated experimentally using the "canonical" sequences of via points. For the 4 DOF manipulators in the workcell, the computational time is about 15 ms per via point in a Sparc-Station 2 machine.

This algorithm is used to create all the trajectories (single-arm, dual-arm, and object) that correspond to pre-planned paths in the robotic workcell. Experimental results are presented for several trajectories computed for the workcell manipulators. The algorithm has since been used in a variety of other systems such as underwater robotic systems [22], and the Marsokhod rover [13].

References

[1] R. C. Arkin. The Impact of Cybernetics on the Design of a Mobile Robot System: A Case Study. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1245–1257, November 1990.

[2] J.E. Bobrow. Optimal robot path planning using the minimum-time criterion. *IEEE Journal of Robotics and Automation*, 4(4):443–451, August 1988.

[3] J. Butler and M. Tomizuka. A suboptimal reference generation technique for robotic manipulators following specified paths. *Transactions of the ASME, Journalof Dynamic Systems, Measurement, and Control*, 114:524–527, September 1992.

[4] Elmer G. Gilbert and Daniel W. Johnson. Distance functions and their application to robot path planning in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, RA-1(1):21–30, September 1985.

[5] S. Dubowsky J.E. Bobrow and J.S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(3), Fall 1985.

[6] Oussama Khatib. Real time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1), 1986.

[7] Tsai-Yen Li and Jean-Claude Latombe. On-Line Motion Planning for Two Robot Arms in a Dynamic Environment. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, Nagoya, Japan, May 1995.

[8] G. Pardo-Castellote, S. A. Schneider, and R. H. Cannon Jr. System Design and Interfaces for Intelligent Manufacturing Workcell. In *Proceedings of the International Conference on Robotics and Automation*, Nagoya, Japan, May 1995. IEEE, IEEE Computer Society.

[9] Gerardo Pardo-Castellote. *Experiments on the Integration and Control of an Intelligent Manufacturing Workcell*. PhD thesis, Stanford University, Department of Electrical Engineering, Stanford, CA 94305, June 1995. Also published as SUDAAR 675.

[10] Gerardo Pardo-Castellote, Tsai-Yen Li, Yoshihito Koga, Robert H. Cannon Jr., Jean-Claude Latombe, and Stan Schneider. Experimental Integration of Planning in a Distributed Control System. In Tsuneo Yoshikawa and Fumio Miyazaki, editors, *Experimental Robotics III: The third International Symposium*, volume 200 of *Lecture Notes in Control and Information Sciences*, pages 50–61. Springer-Verlag, Kyoto Japan, October 28-30 1993.

[11] Sean Quinlan. *Real-Time Modification of Collision-Free Paths*. PhD thesis, Stanford University, Department of Computer Science, Stanford, CA 94305, 1994. Also published as STAN-CS-TR-94-1537.

[12] E. Rimon and D.E. Koditschek. Exact Robot Navigation Using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–18, Oct. 1992.

[13] S. A. Schneider, V. Chen, J. Steele, and G. Pardo-Castellote. The ControlShell Component-Based Real-Time Programming System, and its Application to the Marsokhod Martian Rover. In *Proceedings of 2nd ACM SIGPLAN on Languages, Compilers, and Tools for Real-Time Systems*, La Jolla, CA, June 1995. IEEE, IEEE Computer Society.

[14] Z. Shiller and S. Dubowsky. Robot path planning with obstacles, actuator, gripper and payload constraints. *International Journal of Robotics Research*, 8(6), December 1989.

[15] Z. Shiller and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, 7(6):785–797, December 1991.

[16] Kang G. Shin and Neil D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, AC-30(6):531–541, June 1985.

[17] Kang G. Shin and Neil D. McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *IEEE Transactions on Automatic Control*, AC-31(6):491–500, June 1986.

[18] Kang G. Shin and Neil D. McKay. Minimum-time trajectory planning for industrial robots with general torque constraints. In *IEEE International Conference on Robotics and Automation*, pages 412–415, San Francisco, California, April 6-10 1986.

[19] Kang G. Shin and Neil D. McKay. Selection of near-minimum time geometric paths for robotic manipulators. *IEEE Transactions on Automatic Control*, AC-31(6):501–511, June 1986.

[20] S. Singh and M.C. Leu. Optimal trajectory generation for robotic manipulators using dynamic programming. *Transactions of the ASME*, 109:88–96, June 1987.

[21] J. E. Slotine and H. S. Yang. Improving the efficiency of time-optimal path-following algortihms. *IEEE Transactions on Robotics and Automation*, 5(1):118–124, April 1989.

[22] H. H. Wang, R. L. Marks, S. M. Rock, and M. J. Lee. Task-Based Control Architecture for an Untethered, Unmanned Submersible. In *Proceedings of the 8th Annual Symposium of Unmanned Untethered Submersible Technology*, pages 137–147. Marine Systems Engineering Laboratory, Northeastern University, September 1993.

[23] J. T. Wen and A. Desrochers. Sub-time-optimal control strategies for robotic manipulators. In *IEEE International Conference on Robotics and Automation*, San Francisco, California, April 6-10 1986.