Fast Smoothing of Manipulator Trajectories using Optimal Bounded-Acceleration Shortcuts

Kris Hauser* and Victor Ng-Thow-Hing†
* School of Informatics and Computing, Indiana University, Bloomington, IN 47405 USA
† Honda Research Institute, Mountain View, CA 94041 USA

***Abstract*— This paper considers a shortcutting heuristic to smooth jerky trajectories for many-DOF robot manipulators subject to collision constraints, velocity bounds, and acceleration bounds. The heuristic repeatedly picks two points on the trajectory and attempts to replace the intermediate trajectory with a shorter, collision-free segment. Here, we construct segments that interpolate between endpoints with specified velocity in a time-optimal fashion, while respecting velocity and acceleration bounds. These trajectory segments consist of parabolic and straight-line curves, and can be computed in closed form. Experiments on reaching tasks in cluttered human environments demonstrate that the technique can generate smooth, collision-free, and natural-looking motion in seconds for a PUMA manipulator and the Honda ASIMO robot.**

I. Introduction

Autonomous robots that interact with humans or human environments must have the capability to quickly generate safe and natural-looking motion. So far, it has been a challenge to simultaneously satisfy the three objectives of speed, safety, and esthetics for high-DOF robots performing complex tasks in unstructured environments. Sample-based planners (e.g., PRM, RRT, etc., see Chapter 7 of [1]) are widely used to plan collision-free paths for high-DOF robots. They are often fast, but they produce jerky, unnatural paths. This paper presents a fast smoothing algorithm that postprocesses paths to produce a dynamic trajectory that respects velocity and acceleration bounds and avoids collision.

Standard sample-based planners compute piecewise linear paths that can be executed precisely by stopping the robot at every vertex along the path. This is slow and looks unnatural, so smoothing is often performed before execution. Spline fitting approaches can overshoot or undershoot the original path and lead to collision. Numerical trajectory optimization can be used, but is computationally expensive because the feasibility of the path must be checked at each iteration [2].

This paper presents a variant of a *shortcutting* heuristic commonly used in robotics and animation. The heuristic iteratively picks two points along the existing path, constructs a segment between them, and checks it for collision [3]. If it is collision-free, the segment replaces the subpath between the two points. Our technique differs from traditional approaches in two ways. First, instead of operating in configuration space, our algorithm operates in configuration/velocity state space. Second, we use smooth, dynamically-feasible trajectory segments as shortcuts. These shortcuts interpolate between points in state space in a time-optimal fashion, given bounded joint velocities and accelerations. The output of the algorithm is a smooth trajectory that respects collision, velocity, and acceleration constraints.

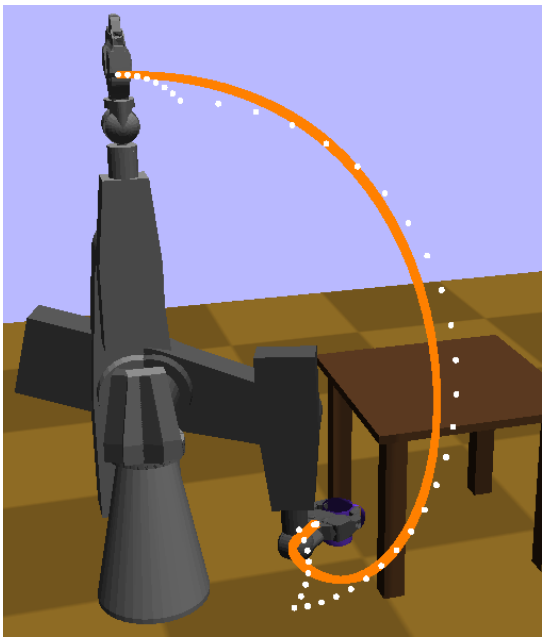


Fig. 1. A manipulator reaches under a table to grasp a cup. The white dotted curve depicts the original end effector path. The orange curve depicts the smoothed path after 100 randomly-attempted shortcuts. Execution time is reduced from 9.4s to 4.0s.

The primary contribution of this work is an analytical derivation of time-optimal, bounded-velocity, bounded-acceleration trajectories that interpolate between endpoints with specified velocities. For a single joint, the time-optimal interpolant is known to be composed of parabolic and linear arcs and can be derived in closed form. We interpolate multiple joints by finding the joint with the longest execution time T, and then solve for the minimum-acceleration interpolants for the remaining joints with end time T fixed. We also present a method for exact collision checking of these trajectories based on the adaptive technique of [4].

We test the approach in moderately cluttered environments using a PUMA760 manipulator (in simulation) and the arms of the Honda ASIMO humanoid (both in simulation and on the real robot). Experiments show that high-quality, smooth paths with low execution time can be produced with only a handful of shortcuts, which take just a few seconds of computation time on a 3 GHz PC (Fig. 1). The approach also

lends itself to a convenient "on-line" smoothing implementation that optimizes the path during execution. C++ code for the algorithms in this paper are available for download at `http://cs.indiana.edu/~hauserk/software.htm`.

II. Related Work

High-quality motion generation is a topic of interest in computer animation, and a common technique is to adapt high-quality example motions to new characters, tasks, and environments [5], [6]. Another technique constructs trajectories using a generative model of "natural-looking" motion, derived from example motions or biomechanical principles [7]–[9]. Hard motion constraints can be handled by using the models to bias the sampling strategy of a sample-based planner [10], [11]. A similar approach was applied to humanoid robot locomotion [12].

Especially for non-human-like robots, "natural-looking" can be defined in terms of a cost function that includes obstacle potential fields [13], [14] or physical criteria like execution time, torque, or energy consumption [15], [16]. Then, cost is minimized using iterative numerical techniques (e.g., gradient descent) [17]. These techniques are typically too slow for real-time use; the resulting optimization problems can be extremely large, and each iteration is expensive because of the large number of collision queries that need to be tested (at possibly hundreds or thousands of discrete points along the path). To put this in perspective, querying collisions along the path shown in Fig. 1 takes approximately 0.15 s on a 3 GHz PC.

A *shortcutting* heuristic tries to replace portions of a path with shorter segments such as straight lines, and is commonly used in robotics and computer animation [10], [11] . It is fast, is easily implemented, and often produces high-quality paths in practice [3]. These techniques do not achieve optimality, or even local optimality, but in practice can produce short paths quickly. They can also produce good initial trajectories for further optimization using numerical techniques.

The approach of shortcutting with time-optimal trajectories has been applied to car-like vehicles using Reeds-Shepp curves [18]. A similar technique was used for smoothing trajectories of aerial vehicles by placing trim curves at waypoints [19]. Our main contribution is the closed-form solution of time-optimal trajectories for acceleration- and velocity- bounded systems, which is applicable to a wide variety of robot manipulators.

III. Assumptions and Notation

Let $\mathcal{C} = \mathbb{R}^d$ denote the d-dimensional configuration space, and let $\mathcal{F}$ denote the subset of configurations that are collision-free and respect joint limits. Vector-valued quantities are denoted in bold (e.g., $\mathbf{x}$), and superscripts denote joint indexing (e.g., $\mathbf{x}^k$).

A trajectory $\mathbf{u}(t)$ is represented as a curve with piecewise-constant acceleration (and is therefore piecewise composed of parabola and straight lines). A trajectory is considered to be feasible if

1) Configurations $\mathbf{u}(t)$ lies entirely in $\mathcal{F}$.

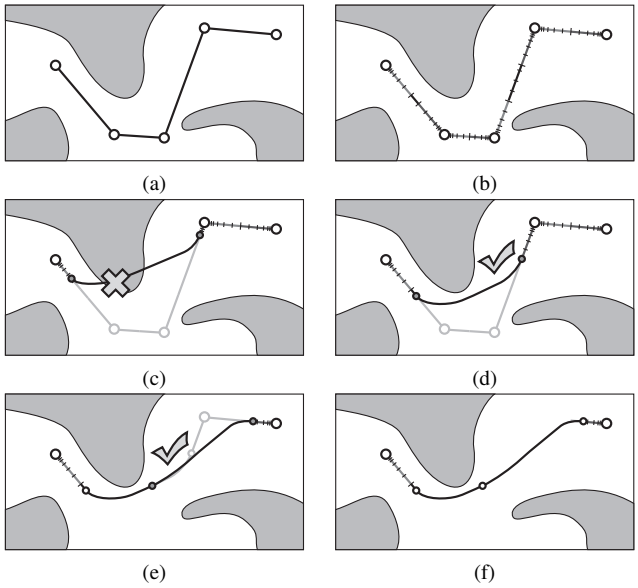


Fig. 2. Illustrating the smoothing algorithm. (a) A jerky path produced by a sample-based planner. (b) Converted into a trajectory that stops at each milestone. (c) Attempting a random shortcut. The feasibility check fails. (d),(e) Two more shortcuts, which pass the feasibility check. (f) The final trajectory executed by the robot.

2) Velocities $\mathbf{u}'(t)$ are bounded by box limits $|\mathbf{u}'(t)| \leq \mathbf{v}_{max}$. (Here, absolute value and inequality are taken element-wise).
3) Accelerations $\mathbf{u}''(t)$ are bounded by box limits $|\mathbf{u}''(t)| \leq \mathbf{a}_{max}$.

Our objective is to reduce the execution time of an input trajectory as much as possible while retaining feasibility.

Some comments are warranted about these assumptions. First, the velocity and acceleration bounds are somewhat idealized constraints. The true *physical* limits are a complex, nonlinear function of motor torque and power characteristics, joint configuration, and dynamic interactions between joints. Although some applications may demand that a robot be pushed to its limits, in practice artificial velocity and acceleration bounds are usually justified for reasons of safety and servo stability.

Second, reducing execution time is typically not a desirable objective alone, because time-optimal trajectories will graze obstacles with little or no room for error. Some separation margin may be needed to prevent collisions in the presence of uncertainties and disturbances, or the objective could weight between execution time and obstacle proximity. Such modifications can be easily implemented in our method.

IV. Shortcutting Algorithm

The algorithm, summarized in Fig. 3, performs N iterations of shortcutting on a piecewise-linear path $\mathbf{x_1}, \ldots, \mathbf{x_n}$ in $\mathcal{F}$. Fig. 2 illustrates three iterations of the algorithm.

Step 1 converts the path to a trajectory that stops at every milestone, and will be described in Sec. IV-A. Steps 3–5 selects two random states $(\mathbf{a}, \mathbf{c})$ and $(\mathbf{b}, \mathbf{d})$ along the trajectories. The `Shortcut` subroutine in line 6 computes the time-optimal interpolant $\mathbf{s}(t)$ between states $(\mathbf{a}, \mathbf{c})$ and $(\mathbf{b}, \mathbf{d})$

Input: piecewise linear path $\mathbf{x_1}, \ldots, \mathbf{x_n}$, iteration count N
Output: a smoothed, dynamically-feasible trajectory from $\mathbf{x_1}$ to $\mathbf{x_n}$.
1. $\mathbf{u} \leftarrow \texttt{StartStop}(\mathbf{x_1}, \ldots, \mathbf{x_n})$
2. Repeat for N iterations:
3. Sample t_a and t_b randomly from $[0, T]$
4. $(\mathbf{a}, \mathbf{c}) \leftarrow (\mathbf{u}(t_a), \mathbf{u}'(t_a))$
5. $(\mathbf{b}, \mathbf{d}) \leftarrow (\mathbf{u}(t_b), \mathbf{u}'(t_b))$
6. $\mathbf{s} \leftarrow \texttt{Shortcut}((\mathbf{a}, \mathbf{c}), (\mathbf{b}, \mathbf{d}))$
7. If $\texttt{CollisionFree?}(\mathbf{s})$ then
8. Replace the section of $\mathbf{u}$ from t_a to t_b with $\mathbf{s}$
9. Return $\mathbf{u}$.

Fig. 3. Pseudocode for the shortcutting algorithm.

as described in Sec. IV-B. Line 7 tests if $\mathbf{s}(t)$ is collision free as described in Sec. IV-E, and if so, $\mathbf{s}(t)$ is spliced into the path in line 8.

A. Time-Optimal Straight-Line Trajectories

The first step of the algorithm converts a piecewise linear path to a time-optimal trajectory that stops at each vertex. This requires a routine algebraic computation, and we include it here for completeness.

Let the interpolation parameter s be a monotonic function $s(t)$, such that the trajectory $\mathbf{u}(t) = \mathbf{a} + s(t)(\mathbf{b} - \mathbf{a})$ tracks the straight-line path (see Fig. 4.a). Then the velocity and acceleration of $s(t)$ must not exceed, respectively, $v_s = \min_k \mathbf{v}^k_{max}/|\mathbf{b}^k - \mathbf{a}^k|$ and $a_s = \min_k \mathbf{a}^k_{max}/|\mathbf{b}^k - \mathbf{a}^k|$ where the minimums are taken over joints $k = 1, \ldots, d$. The optimal $s(t)$ travels along 1) a parabolic arc with acceleration a_s, 2) may reach the maximum velocity v_s and travel along a straight line, and 3) a parabolic arc with acceleration $-a_s$. The interpolation curve is completely determined the inflection points.

First, we consider the case where $s'(t)$ never exceeds v_s. Then, the inflection point occurs at $t_P = \sqrt{|(x_2 - x_1)/a_s}$. If $a_s t_P \leq v_s$, then this trajectory is valid. Otherwise $s(t)$ must contain a linear section. Then, the first inflection point is at $t_1 = v_s/a_s$, the second occurs after an additional time $t_2 = 1/v_s - 1/a_s$ has elapsed, and after decelerating for duration t_1 the trajectory terminates at $T = 2t_1 + t_2$. Fig. 5a illustrates the solution for $s(t)$ with $v_s = a_s = 1$.

B. Time-Optimal, Bounded-Acceleration, Bounded-Velocity Interpolants

The fastest dynamically feasible trajectory between $(\mathbf{a}, \mathbf{c})$ and $(\mathbf{b}, \mathbf{d})$ consists of parabolic arcs and straight lines. Since each joint variable is assumed to be independent, the minimum execution time is determined by the slowest single-joint trajectory. We first determine this time T, and the interpolate the rest of the joints with T fixed. Of the many possible interpolants that finish at time T, we pick the minimum-acceleration interpolant. We will describe these unidimensional interpolants in the next section.

More precisely, let $f(x_1, x_2, v_1, v_2, v_{max}, a_{max})$ compute the time of the time-optimal interpolant between x_1 and x_2, with beginning and ending velocity v_1 and v_2 respectively, under maximum velocity v_{max} and acceleration a_{max}. Let $g(x_1, x_2, v_1, v_2, v_{max}, T, t)$ compute the state at time t of the minimum-acceleration interpolant, given a fixed final time T. (The next section will describe f and g in more detail.) To construct multidimensional interpolants, we first compute the optimal time

$$T = \max_k f(\mathbf{a}^k, \mathbf{b}^k, \mathbf{c}^k, \mathbf{d}^k, \mathbf{v}^k_{max}, \mathbf{a}^k_{max}), \tag{1}$$

and then compute the acceleration-optimal joint trajectories

$$\mathbf{s}^k(t) = g(\mathbf{a}^k, \mathbf{b}^k, \mathbf{c}^k, \mathbf{d}^k, \mathbf{v}^k_{max}, T, t). \tag{2}$$

Examples of these curves are illustrated in Fig. 4.

C. Univariate Time-Optimal Interpolants

Here we derive the function $f(x_1, x_2, v_1, v_2, v_{max}, a_{max})$ that computes the execution time of a univariate, time-optimal, velocity- and acceleration-bounded trajectory. We compute it by enumerating all bang-bang controls [20] that connect the initial and final states, and picking the one with the lowest execution time. For each control, we have computed inflection points analytically through an elementary, but somewhat tedious derivation. We omit the derivations from the following discussions.

We define four motion primitives: the parabolas P^+ and P^- accelerating at a_{max} and $-a_{max}$, respectively, and the lines L^+ and L^- traveling at v_{max} and $-v_{max}$, respectively. There are four possible classes of motion primitive combinations that may be optimal: P^+P^-, P^-P^+, $P^+L^+P^-$, and $P^-L^-P^+$. We examine each class for a valid execution time T, and find the class with the minimal execution time.

For class P^+P^-, to find the inflection time t_P when the trajectory stops accelerating and starts decelerating, we compute a solution t of the quadratic equation

$$a_{max}t^2 + 2v_1 t + (v_1^2 - v_2^2)/(2a_{max}) + x_1 - x_2 = 0 \tag{3}$$

that also satisfies $0 \leq t \leq (v_2 - v_1)/a_{max}$. If no solution exists, the class is declared invalid. If a solution exists, the total time is $T = 2t_P + (v_1 - v_2)/a_{max}$. We must also check that the maximum speed of the trajectory $v_1 + t_P a_{max}$ does not exceed v_{max}. The P^-P^+ solution is given by negating a_{max} in the above equations.

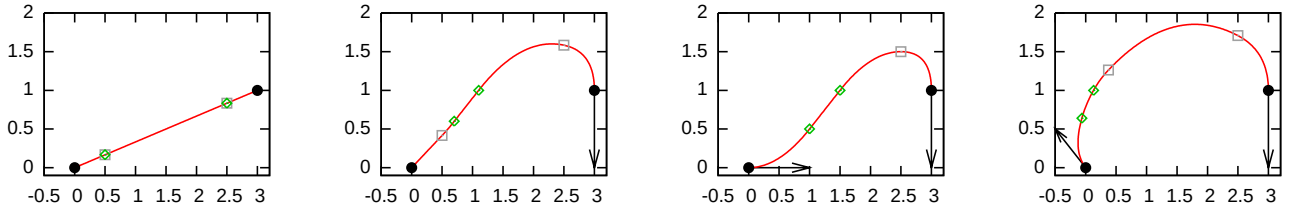
For class $P^+L^+P^-$, we compute the duration t_L of the linear portion,

$$t_L = (v_2^2 + v_1^2 - 2v_{max}^2)/(2v_{max}a_{max}) + (x_2 - x_1)/v_{max}, \tag{4}$$

the duration $t_{P1} = (v_{max} - v_1)/a_{max}$ in the first parabola, and the duration $t_{P2} = (v_2 - v_{max})/a_{max}$ in the second. If any of t_L, t_{P1}, or t_{P2} are negative, the class is invalid. Otherwise, the execution time is given by

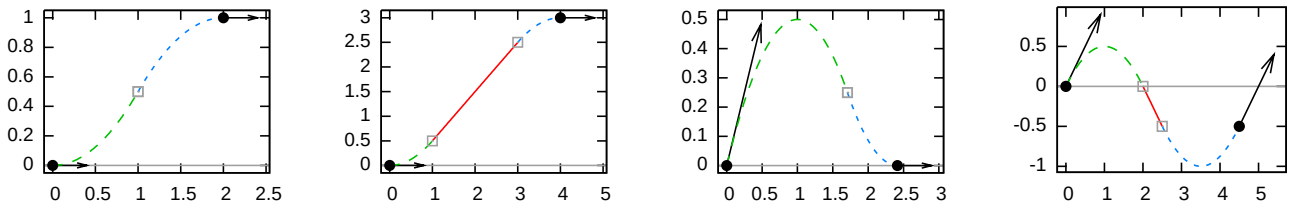
$$T = t_{P1} + t_L + t_{P2}. \tag{5}$$

The $P^-L^-P^+$ solution is given by negating a_{max} and v_{max} in the above equations.



(a) $\mathbf{v}_a = \mathbf{v}_b = (0,0)$ (b) $\mathbf{v}_a = (0,0), \mathbf{v}_b = (0,-1)$ (c) $\mathbf{v}_a = (1,0), \mathbf{v}_b = (0,-1)$ (d) $\mathbf{v}_a = (-0.5,0.5), \mathbf{v}_b = (0,-1)$

Fig. 4. 2D time-optimal trajectories from (0,0) to (3,1) under $v_{max} = (1,1)$ and $a_{max} = (1,1)$ and varying start and final velocity $\mathbf{v}_a$ and $\mathbf{v}_b$. Squares depict inflection points in the x parameter, and diamonds depict inflection points in the y parameter.



(a) P^+P^- curve from 0 to 1, starting and stopping at rest. (b) $P^+L^+P^-$ curve from 0 to 3, starting and stopping at rest. (c) P^-P^+ curve from 0 to 0, with initial velocity 1 and stopping at rest. (d) $P^-L^-P^+$ curve starting at 0 with velocity 1 and ending at -0.5 with velocity 1.

Fig. 5. Univariate time-optimal trajectories under $v_{max} = 1$ and $a_{max} = 1$. Time is plotted on the horizontal axis. Squares depict inflection points between parabolic and linear trajectory segments.

D. Minimum-Acceleration Interpolants

Now we derive the minimum-acceleration trajectory given a fixed end time T. First, we compute the minimal acceleration $a = h(x_1, x_2, v_1, v_2, v_{max}, T)$, and then the optimal trajectory $g(x_1, x_2, v_1, v_2, v_{max}, T, t)$ is defined in a straightforward manner from a. Again, we use the same combinations of motion primitives: P^+P^-, P^-P^+, $P^+L^+P^-$, and $P^-L^-P^+$, and find the class that has minimum acceleration.

For classes P^+P^- and P^-P^+, we compute solutions a to the equation

$$T^2a^2 + \sigma(2T(v_1+v_2) + 4(x_1-x_2))a - (v_2-v_1)^2 = 0 \quad (6)$$

whose switch time $t_S = 1/2(T + (v_2 - v_1)/a)$ satisfies the condition $0 \leq t_S \leq T$. Here, σ is defined as $+1$ for class P^+P^- and -1 for class P^-P^+. We must also check that the maximum speed of the trajectory $|v_1 + at_S|$ does not exceed v_{max}.

For class $P^+L^+P^-$, we have

$$a = \frac{v_{max}^2 - v_{max}(v_1+v_2) + 0.5(v_1^2+v_2^2)}{Tv_{max} - (x_2 - x_1)} \quad (7)$$

We then compute the durations t_L, t_{P1}, and t_{P2} as in the previous section, but replace a_{max} with a, and check that they are all positive. For class $P^-L^+P^+$, we negate v_{max} in the above equation.

E. Trajectory Collision Checking

A simple, inexact method for trajectory collision checking is to discretize the curve to resolution ϵ and test each point for collision. If ϵ is small, the checker will be slow, but if ϵ is large, the checker may miss some collisions. The risk of a missed collision increases after a large number of smoothing steps as the trajectory approaches closer to obstacles. One possible solution grows the geometry of the robot or obstacles by a small but sufficiently large margin, so that penetrations of distance ϵ do not cause actual collisions. Another solution is *exact* collision checking by attempting to cover the path with collision-free neighborhoods [4].

We extend the adaptive, recursive bisection technique of [4] to handle parabolic paths as well as straight paths. (see Fig. 6). Given a trajectory segment $\{\mathbf{u}(t) \mid t_1 \leq t \leq t_2\}$ we compute the maximum workspace distance λ traversed by any point on the robot's geometry as the robot executes the segment. We also compute the robot-environment distances δ_1 and δ_2 at $\mathbf{u}(t_1)$ and $\mathbf{u}(t_2)$, respectively (Fig. 6b). If $\lambda \leq \delta_1 + \delta_2$, then the entire segment is guaranteed to be collision free, and it can be excluded from further collision testing (Fig. 6d). If $\lambda > \delta_1 + \delta_2$, then the segment is bisected at $(t_1 + t_2)/2$ and the algorithm recurses on the two halves.

The method uses a subroutine to compute a bound on the workspace distance traveled by a point on the robot. For example, for a serial robot with d revolute joints and with link length at most L, a straight line motion $\Delta\mathbf{q}$ can move the robot geometry by at most a distance of $\sum_{k=1}^{d} |\Delta\mathbf{q}^k| kL$. For piecewise parabolic paths, we compute a bounding box around the entire path, and replace the term $\Delta\mathbf{q}$ with the diagonal vector of the bounding box. There are various optimizations that speed up the basic algorithm, such as using non-cubic neighborhoods, computing different neighborhoods for different constraints, and using approximate distance queries, all of which are described in more detail in [4]. In our experiments, the exact collision checker typically performs approximately 2–10 times slower than discretizing the path at a fine resolution. As expected, it is slowest when the robot passes close to obstacles, for example, near the goal configuration of Figure 1.

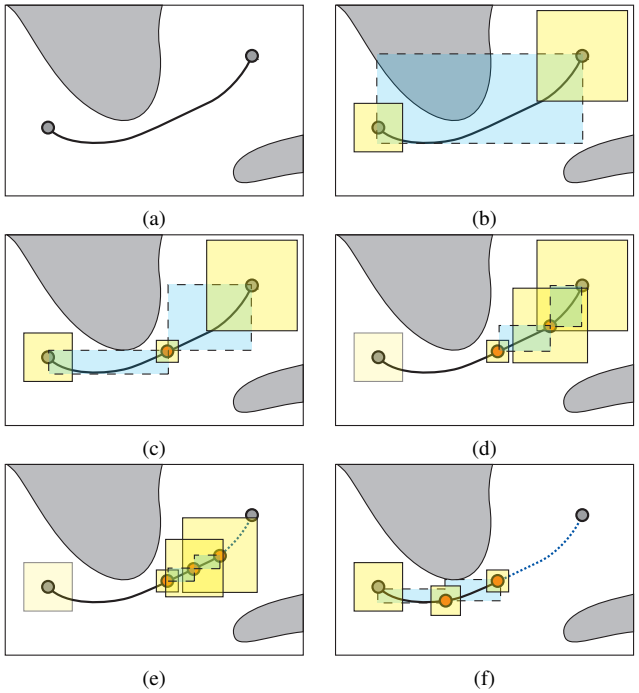


Fig. 6. Illustrating exact collision testing on (a) a trajectory segment. (b) Collision-free neighborhoods (boxes) and a path bounding box (dashed box) are computed. (c) The segment is bisected and recursion begins. (d) Bisecting the right half, the algorithm finds that the rightmost quarter lies inside a collision-free neighborhood of the rightmost endpoint. (e) Bisecting the third quarter, the fifth and sixth eighths are found to be collision-free. (f) Recursion returns to the left half and continues.

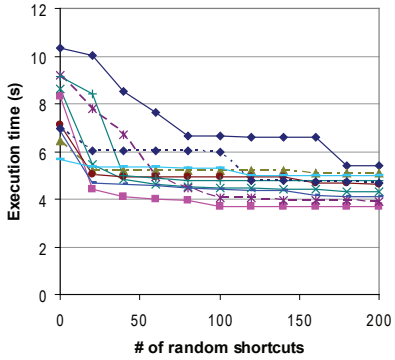


Fig. 7. Trajectory durations as the algorithm progresses for 10 different initial paths, on the example of Fig. 1.

V. Experiments

Experiments were performed on a simulated reaching task for a CAD model of a PUMA robot with approximately 8,000 polygons and an environment with over 20,000 polygons (Fig. 1). We generated 10 initial paths with the same start and end configurations as Fig. 1 using the SBL motion planner [21]. If these paths are followed exactly, their optimal execution times range from 5.7 s to 10.4 s. After 200 shortcutting iterations, execution times are reduced by an average of 40%, and the resulting trajectories range from 4.1 s to 6.6 s (Fig. 7). On a 3 GHz PC, average computation time is 0.6s for initial planning, 2.6s for smoothing, and 3.2s total. Because of our analytical construction, the shortcuts take negligible time to construct; smoothing time is overwhelmingly dominated by collision checking.

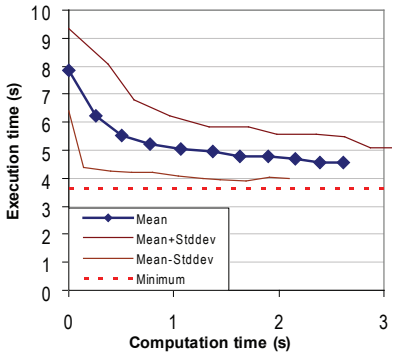


Fig. 8. Timing experiments on the example of Fig. 1. Mean, standard deviation, and minimum are plotted for 10 different initial paths.

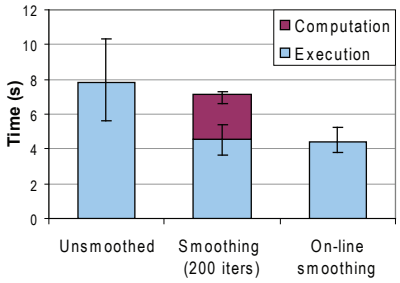


Fig. 9. On-line smoothing on the example of Fig. 1. Mean, minimum, and maximum are plotted over 10 different initial paths.

Fig. 8 shows that the smoothing algorithm reduces the trajectory execution time quickly at first, but further iterations have a lower rate of return. If a system is implemented in a sequential fashion such that smoothing must be completed before beginning execution, then it is unclear when to stop smoothing in order to achieve an ideal balance between computation and execution time. But, shortcutting algorithms can be conveniently implemented in an *on-line* technique that executes and smoothes the path in parallel, which makes choosing a termination criterion unnecessary. In this technique, shortcuts are drawn at random only from the path after the current time, plus some small padding. Experiments in Fig. 9 show that this method produces high-quality paths without needing to pause the robot.

We also integrated the smoothing algorithm into a system that enables ASIMO to push objects over a table [22]. Fig. 10 shows frames from a smoothed reaching motion performed on the real robot. In timing experiments, 50 iterations of shortcutting averaged 0.9s of computation time and reduced execution time by an average of 2.4s over 10 motions generated by SBL. Overall this was a 46% reduction relative to the execution time of the initial paths, and the motions were judged by observers to qualitatively look much more natural. Videos of our experiments can be found at `http://www.cs.indiana.edu/~hauserk/videos/icra2010/`.

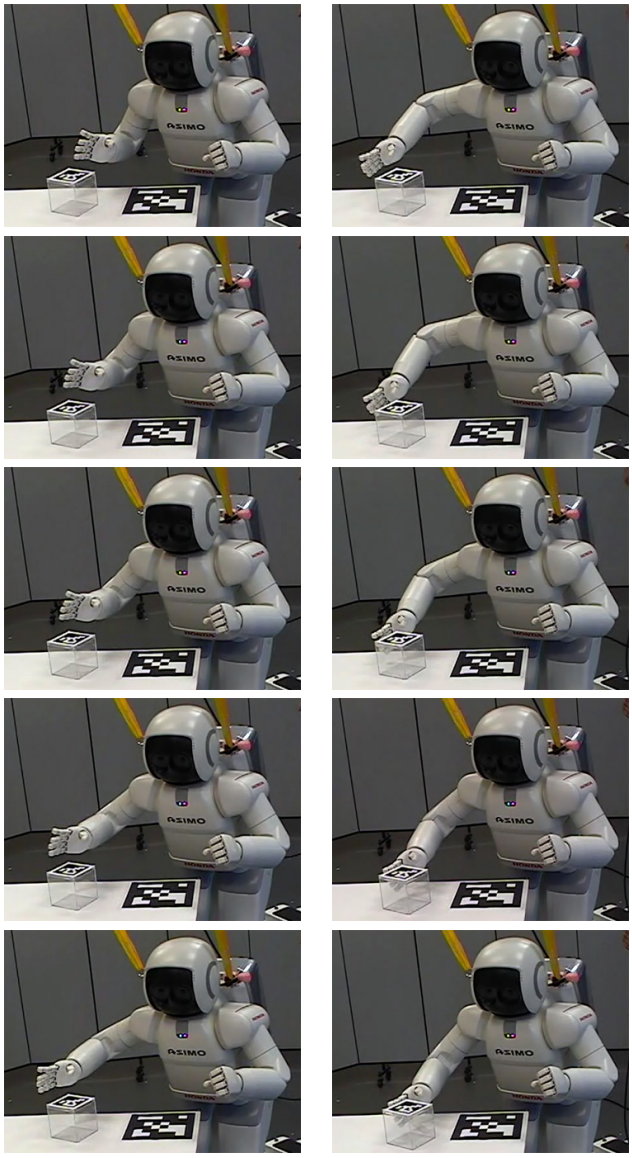


Fig. 10. The ASIMO robot reaching to push a block using a smoothed arm trajectory. Sequence proceeds from top to bottom, left to right).

VI. Conclusion

This paper presented a fast algorithm for smoothing collision-free trajectories for many-DOF robot manipulators. It uses a shortcutting heuristic that draws smooth velocity- and acceleration-bounded shortcuts between random points on the trajectory, and if the shortcut is collision-free, replaces the intermediate portion of the trajectory with the shortcut. The primary contribution of the paper is a closed-form derivation of the time-optimal velocity- and acceleration-bounded curves that interpolate between two endpoints with specified velocity. Experimental results on a PUMA manipulator and the Honda ASIMO robot demonstrate that the algorithm can smooth trajectories in moderately cluttered environments in seconds. It can also be implemented as an on-line algorithm that smoothes the trajectory during execution.

Like other postprocessing approaches, the algorithm may converge to a suboptimal solution if given a poor input path. Integrating smoothing into planning may be able to overcome some of these problems. Other work could attempt to reduce the time per shortcut with faster collision checking, or improve the convergence rate with nonuniform or adaptive shortcutting strategies.

References

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[2] S. Dubowsky, M. Norris, and Z. Shiller, "Time optimal trajectory planning for robotic manipulators with obstacle avoidance: A cad approach," in *Int. Conf. Rob. Auto.*, vol. 3, Apr 1986, pp. 1906–1912.

[3] R. Geraerts and M. H. Overmars, "Creating High-quality Paths for Motion Planning," *Int. J. of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.

[4] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking of robot paths," in *WAFR*, Nice, France, Dec 2002.

[5] A. Witkin and Z. Popović, "Motion warping," in *SIGGRAPH*, Los Angeles, CA, 1995, pp. 105–108.

[6] M. Gleicher, "Retargetting motion to new characters," in *SIGGRAPH*, 1998, pp. 33–42.

[7] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 522–531, 2004.

[8] C. K. Liu, A. Hertzmann, and Z. Popović, "Learning physics-based motion style with nonlinear inverse optimization," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1071–1081, 2005.

[9] M. Meredith and S. Maddock, "Adapting motion capture data using weighted real-time inverse kinematics," *Comput. Entertain.*, vol. 3, no. 1, 2005.

[10] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann, "Planning collision-free reaching motions for interactive object manipulation and grasping," *Computer Graphics Forum*, vol. 22, no. 3, pp. 313–322, 2003.

[11] K. Yamane, J. J. Kuffner, and J. K. Hodgins, "Synthesizing animations of human manipulation tasks," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 532–539, 2004.

[12] K. Hauser, T. Bretl, K. Harada, and J.-C. Latombe, "Using motion primitives in probabilistic sample-based planning for humanoid robots," in *WAFR*, New York, NY, 2006.

[13] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. J. of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[14] M. Toussant, M. Gienger, and C. Goerick, "Optimization of sequential attractor-based movement for compact behaviour generation," in *Proc. IEEE. Conf. on Humanoid Robots*, 2007.

[15] Z. Shiller and S. Dubowsky, "Global time optimal motions of robotic manipulators in the presence of obstacles," in *IEEE Int. Conf. Rob. Auto.*, Apr 1988, pp. 370–375 vol.1.

[16] J. E. Bobrow, "Optimal robot path planning using the minimum-time criterion," *IEEE J. of Robotics and Automation*, vol. 4, no. 4, pp. 443–450, 1988.

[17] J. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control and Dynamics*, vol. 21, no. 2, pp. 193–207, 1999.

[18] P. Jacobs, J. Laumond, and M. Ta ix, "Efficient motion planners for nonholonomic mobile robots," in *IEEE/RSJ Int'l. Conf. Intel. Robots and Systems*, Osaka, Japan, 1991.

[19] E. Anderson, R. Beard, and T. McLain, "Real-time dynamic trajectory smoothing for unmanned air vehicles," *IEEE Trans. on Control Systems Technology*, vol. 13, no. 3, pp. 471–477, May 2005.

[20] L. Sonneborn and F. V. Vleck, "The bang-bang principle for linear control systems," *SIAM J. Control*, vol. 2, pp. 151–159, 1965.

[21] G. Sánchez and J.-C. Latombe, "On delaying collision checking in PRM planning: Application to multi-robot coordination," *Int. J. of Rob. Res.*, vol. 21, no. 1, pp. 5–26, 2002.

[22] K. Hauser, V. Ng-Thow-Hing, and H. Gonzales-Baños, "Multi-modal planning for a humanoid robot manipulation task," in *Intl. Symposium on Robotics Research*, Hiroshima, Japan, 2007.