Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Multi-axis High-order Trajectory Planning

Ben Ezair[1] Tamir Tassa[1] Zvi Shiller[2]

***Abstract*— This paper presents a trajectory planning algorithm for multi-axis systems. It generates smooth trajectories of any order subject to general initial and final conditions, and constant state and control constraints. The algorithm is recursive, as it constructs a high order trajectory using lower order trajectories. Multi-axis trajectories are computed by synchronizing independent single-axis trajectories to reach their respective targets at the same time.**

The algorithm's efficiency and ability to handle general initial and final conditions make it suitable for reactive real time applications. Its ability to generate high order trajectories makes it suitable for applications requiring high trajectory smoothness. The algorithm is demonstrated in several examples for single- and two-axis trajectories of order $2-6$.

I. Introduction

The trajectory planning problem, in the context of robot motion, is the problem of generating a trajectory in the robot's state space that connects given initial and final states, subject to state and control constraints, and is optimal with respect to some given cost function. A trajectory is essentially a time-parameterized path between two points in the configuration space. While path planning has traditionally been concerned with generating the shortest path that avoids obstacles, trajectory planning is concerned in addition with the robot's dynamic behavior by imposing constraints on the robot's velocity, acceleration, jerk and possibly higher derivatives. Bounding the motion derivatives yields smooth trajectories, which can be tracked with smooth control inputs that do not excite high vibration modes. In addition, they increase tracking accuracy [11]. The number of bounded derivatives in the trajectory is called the order of the trajectory.

Several approaches for trajectory generation have been developed. One approach uses polynomials or other functions to approximate the desired trajectories. Piazzi and Visioli [13] optimize cubic splines to minimize jerk for a specified motion time. Petrinec and Kovacic [12] use 4th and 5th order polynomials to produce smooth multi-axis trajectories. Macfarlane and Croft [10] compute trajectories that are represented by fifth-order polynomials.

Another approach for trajectory generation is to divide the trajectory into segments where the value of the highest derivative is constant in each segment. Liu et al. [9] present an algorithm that produces a third order trajectory that is constructed by dividing the trajectory to seven segments. This approach is used to produce multi-axis trajectories by synchronizing several single-axis trajectories [1], [2], [5]. Haschke et al. [4] emphasize the online capabilities of their algorithm that is designed to produce a third order halting trajectory. Works by Kroger et al. [6], [7] also focus on online algorithms, using a thorough analysis of possible acceleration profiles to handle more general initial and final conditions. Lambrechts et al. [8] produce fourth order trajectories. Nguyen et. al. [11] developed an algorithm that generates trajectories of arbitrary order with zero initial and final conditions and symmetric state and control constraints. It is based on dividing the trajectory into a recursive structure of S-curve segments. The use of S-curves forms also the basis for the algorithm which we present herein.

[1]Ben Ezair and Tamir Tassa are with The Department of Mathematics and Computer Science, The Open University, Israel. ben_e@hotmail.com & tamirta@openu.ac.il

[2]Zvi Shiller is with the Department of Mechanical Engineering and Mechatronics, Ariel University Center of Samaria. shiller@ariel.ac.il

A. Our algorithm

This paper presents a novel algorithm for planning trajectories of arbitrary order between arbitrary initial and final states (position and its time derivatives), subject to arbitrary constant state and control constraints, which is geared towards minimizing motion time. The generality of our approach makes the algorithm suitable for both online and offline trajectory planning. The algorithm is recursive, as it reduces the original problem of order m to problems of order $m-1$, until it reaches basic problems that can be solved directly. The algorithm is also modular, as it may accept any external solver for the basic trajectory generation problem which is solved directly in order to terminate the recursion. The algorithm is efficient, as demonstrated in several experiments (see Table II in Section II-B.2).

Finally, the basic algorithm is extended to generate multi-axis trajectories by synchronizing single-axis trajectories to reach their respective targets at the same time.

Table I compares our algorithm with a few comparable algorithms that were presented in the above described studies. Most of the comparable algorithms are limited in the order of the trajectories that they may produce, or in the initial and final conditions that they may accept. Our algorithm's main advantage is its generality and flexibility, as it is applicable to a wider range of scenarios than the other algorithms.

II. Single-axis trajectories

We wish to compute a pair $\langle T, x(t)\rangle$, where $x(t)$ denotes the position of a moving object along a given axis, such that (a) $x(t)$ satisfies given initial and final conditions at $t=0$ and $t=T$,

$$x(0)=x_s^0\,,\ x^{(i)}(0)=x_s^i\,,\quad 1\le i\le m-1\,, \tag{1}$$

$$x(T)=x_f^0\,,\ x^{(i)}(T)=x_f^i\,,\quad 1\le i\le m-1\,, \tag{2}$$

Ref.	Order	Initial & Final Conditions	Online	Optimal
[6]	2	general	yes	yes
[2]	3	zero acceleration	yes	yes
[4]	3	ends at rest	yes	yes
[7]	3	zero final acceleration	yes	yes
[8]	4	rest to rest	no	no
[11]	any	rest to rest	no	no
Ours	any	general	yes	no

TABLE I

COMPARISON OF TRAJECTORY GENERATION ALGORITHMS

where $x^{(i)}(t)$, $i \geq 1$, is the i-th order derivative of $x(t)$; (b) it is constrained by constant lower and upper bounds,

$$x^i_{min} < 0 < x^i_{max}\,, \quad 1 \leq i \leq m \tag{3}$$

$$x^i_{min} \leq x^{(i)}(t) \leq x^i_{max}\,, \quad t \in [0,T]\,,\ 1 \leq i \leq m\,; \tag{4}$$

and (c) the time $T = \int_0^T 1dt$ is minimized. The number $m \geq 1$ of constrained derivatives is called the order of the problem. A pair $\langle T, x(t)\rangle$ that satisfies the initial and final conditions, (1)–(2), and the lower and upper bounds (4) is called a feasible solution. A solution is optimal if is feasible and minimizes T.

This single-axis trajectory planning problem may be viewed as a time optimal control problem of a linear system of ordinary differential equations with m state variables (being the position function $x(t)$ and its first $m-1$ derivatives) and a single control variable (being the m-th derivative $x^{(m)}(t)$), subject to initial and final conditions and state and control constraints. The structure of the optimal control for such problems can be shown to have a bang-zero-bang structure [3].

The solution for the case $m = 1$ is trivial, consisting of a constant velocity motion. The solution for $m = 2$ was derived in [6]. Our approach in solving higher order problems is recursive, as it reduces a problem of order m to problems of order $m-1$, repeatedly, until $m = 2$, in which case the problem can be solved directly.

A. A single-axis trajectory planning algorithm

1) Overview: The algorithm for computing high order trajectories is motivated by the observation that integrating a bang-zero-bang control profile yields an S-curve structure. A typical S-curve can be divided into three segments: (I) acceleration from the initial state; (II) cruising at a constant velocity; and (III) deceleration to the final state. This structure, as illustrated in Figure 1 for $m = 3$, repeats recursively since the velocity profile, as well as the profiles of higher derivatives, consist of two or more S-curve segments.

The recursive algorithm looks for a solution with an S-curve position profile. The main loop attempts to find the best value for the constant velocity in segment II. Given a candidate value v for that velocity, the algorithm computes the velocity profile in segments I and III by invoking recursion. Specifically, it solves in each of those segments a reduced order trajectory planning problem for the velocity profiles. Once the velocity profiles in all three segments are found, the algorithm checks that the corresponding position profile is a feasible solution. When the resulting solution is non-feasible, the algorithm reduces $|v|$; when the resulting solution is feasible, the algorithm increases $|v|$ in order to reduce motion time. The algorithm terminates when the optimal value of v is found within some predetermined accuracy, and it outputs the found position profile $x(t)$.

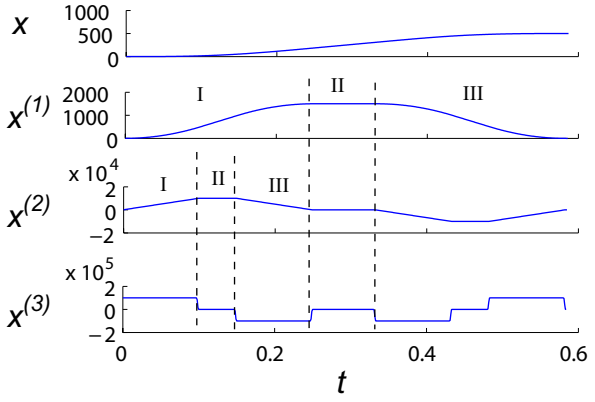

Fig. 1. The recursive structure of the trajectory

2) Detailed description: We proceed to describe the operation of Algorithm 1 that implements the above procedure. The algorithm accepts as inputs the problem order, the initial and final conditions, and the lower and upper constraints. It outputs a feasible solution $\langle T, x(t)\rangle$ which is time-efficient and in some cases optimal.

If $m = 2$ the algorithm outputs the analytic solution (Step 1). Otherwise, we set Δx to be the distance to be traveled (Step 2) and start a binary search for v within the allowed range of values $[x^1_{min}, x^1_{max}]$. The variables v_{min} and v_{max} hold the lower and upper limits of the search range; they are initialized in Step 3. The variable $\hat{v}$ holds the last value of v that produced a feasible solution. It is initialized to an illegal value $(x^1_{max} + 1)$ in Step 3, and so is v.

During the binary search (Steps 4-18), we consider the midpoint of the current range as the candidate value for v (Step 6). Given a candidate value for v, the trajectory planning problem in the acceleration and deceleration segments (I and III) are well defined and can be solved by invoking recursion. Let $v_1(t)$ be the velocity profile in segment I, from the initial value x^1_s to the cruising velocity v, and let $\tau_{1,v}$ denote the duration of that segment. Then in Step 7 we compute $\langle \tau_{1,v}, v_1(t)\rangle$ by solving a problem of order $m-1$ for $x'(t)$ along that segment. The initial conditions for that reduced order problem are $(x^1_s, \ldots, x^{m-1}_s)$; its final conditions are $(v, 0, \ldots, 0)$ (since we wish to reach the velocity v with all higher derivatives zero); and the lower and upper bounds on the derivatives are in accord with those of the original problem. Similarly, we invoke recursion in Step 8 to compute $v_3(t)$, the velocity profile in segment III, from v to the final value x^1_f, and the corresponding duration $\tau_{3,v}$.

Next, we compute the distance covered in segments I and III, Δx_1 and Δx_3 (Step 9). Δ is the remaining distance

that needs to be traveled in the intermediate segment II in order to complete a journey of length Δx. Since the velocity along segment II is constant and equals v, the duration of that segment should be $\tau_{2,v} = \Delta/v$ (Step 10). If $\tau_{2,v}$ is nonnegative, then this tested value of v leads to a valid trajectory; in that case we record that value of v in the variable $\hat{v}$ (Step 11).

The search ends once the lower and upper limits of the search are sufficiently close (Steps 12-14). In that case, we set v, v_{min} and v_{max} to be the last value of v that produced a valid solution. If $\hat{v}$ still equals its initial value $x^1_{max} + 1$ (a forbidden value for v, as it is outside the allowed range $[x^1_{min}, x^1_{max}]$), then the search failed to find a valid v. This may occur if the problem parameters define a range of legitimate v values that is smaller than ε, and, consequently, cannot be captured using a binary search with such accuracy. (We note that instead of using the same value of ε for all levels, we may define for each level i, $1 \le i \le m$, a different value ε_i.) Otherwise, if $\hat{v}$ is a legal value, then the algorithm performs another iteration. Since v, v_{min}, and v_{max} equal the last valid value of v, the subsequent setting of $last\_v$ and v in Steps 5-6 will cause the algorithm to perform the next iteration with $v = \hat{v}$ and then terminate the loop when it examines the termination condition in Step 18.

In case the lower and upper limits of the search are still far apart, we check the value of Δ to determine how to proceed with the search: if $\Delta > 0$, then we examine profiles with higher values of v (Step 15); if $\Delta < 0$, we consider lower values of v (Step 16); if $\Delta = 0$, we terminate the search by setting $last\_v$ to equal v (Steps 17). The search ends when $last\_v = v$. After determining the value v, we compute T and construct the profile of x' as the concatenation of three segments – $v_1(t)$, v,$v_3(t)$ (Steps 19-20). Finally, we integrate $x'(t)$ to obtain $x(t)$ (Step 21).

3) A note on optimality: Algorithm 1 uses a simple greedy approach in the search of a solution with a minimal motion time. The solution is optimal for rest-to-rest motions of order $m \le 3$. (The proof of this claim is deferred to the full version of this paper.) Although Algorithm 1 attempts to minimize motion time, the solution is not necessarily optimal, because the algorithm is based on two assumptions that are not always satisfied:

(A1) The duration of segment II is a continuous and monotonic function of v.

(A2) The velocity during segment II is constant, implying that during this phase all higher derivatives are zero.

The first assumption affects the way the algorithm updates v (steps 15-16). If this assumption is not satisfied, the algorithm may choose a value of v that will result in a non-optimal motion time. The second assumption is more central to Algorithm 1, as it allows us to subdivide the trajectory into two S curves that can be joined together with a simple constant velocity motion. However, this assumption is not always true, e.g. in cases where the optimal trajectory either always accelerates or always decelerates. In such cases, the algorithm would return a solution that is of a different structure than that of the optimal trajectory.

Algorithm 1 ComputeTrajectory

Input:
(1) The system order $m \ge 2$.
(2) Initial and final states: x^i_s, x^i_f, $0 \le i \le m-1$.
(3) Bounds: x^i_{min}, x^i_{max}, $1 \le i \le m-1$.

Output: A feasible solution $\langle T, x(t)\rangle$.

1: **if** $m = 2$ **then** return the analytic solution and stop.
2: $\Delta x = x^0_f - x^0_s$.
3: $v_{min} = x^1_{min}$, $v_{max} = x^1_{max}$, $v = \hat{v} = x^1_{max} + 1$.
4: **repeat**
5: $\quad last\_v = v$.
6: $\quad v = (v_{max} + v_{min})/2$.
7: $\quad \langle \tau_{1,v}, v_1(t)\rangle \leftarrow$ ComputeTrajectory$[m-1,$ $(x^1_s, \ldots, x^{m-1}_s), (v, 0, \ldots, 0),\ \{(x^2_{min}, x^2_{max})\}^m_{i=2}]$
8: $\quad \langle \tau_{3,v}, v_3(t)\rangle \leftarrow$ ComputeTrajectory$[m-1,$ $(v, 0, \ldots, 0), (x^1_f, \ldots, x^{m-1}_f),\ \{(x^2_{min}, x^2_{max})\}^m_{i=2}]$
9: $\quad \Delta x_1 = \int_0^{\tau_{1,v}} v_1(t)dt$; $\Delta x_3 = \int_0^{\tau_{3,v}} v_3(t)dt$.
10: $\quad \Delta = \Delta x - \Delta x_1 - \Delta x_3$; $\tau_{2,v} = \Delta/v$.
11: $\quad$ **if** $\tau_{2,v} \ge 0$, $\hat{v} = v$.
12: $\quad$ **if** $|v_{max} - v_{min}| \le \varepsilon$ **then**
13: $\quad\quad v = v_{min} = v_{max} = \hat{v}$.
14: $\quad\quad$ **if** $(\hat{v} = x^1_{max} + 1)$ **then** stop and output "Failed".
15: $\quad$ **elseif** $\Delta > 0$ **then** $v_{min} = v$
16: $\quad$ **elseif** $\Delta < 0$ **then** $v_{max} = v$
17: $\quad$ **else** $last\_v = v$ **endif**
18: **until** $last\_v = v$
19: $T = \tau_{1,v} + \tau_{2,v} + \tau_{3,v}$.
20: $x'(t) = \begin{cases} v_1(t) & [0, \tau_{1,v}] \\ v & [\tau_{1,v}, \tau_{1,v} + \tau_{2,v}] \\ v_3(t - \tau_{1,v} - \tau_{2,v}) & [\tau_{1,v} + \tau_{2,v}, T] \end{cases}$
21: Return $\langle T, x(t)\rangle$, where $x(t) = \int_0^t x'(\tau)d\tau + x^0_s$.

Both assumptions make the algorithm efficient by limiting the number of possible trajectory forms we need to consider. This, in turn, greatly simplifies the search for segment II that connects segments I and III. It is possible to remove these assumptions, while keeping the recursive structure of the algorithm, and produce the optimal trajectory by exhaustively searching for the initial and final conditions of segment II, at the obvious cost of increasing the computational complexity.

B. Experiments

1) Examples of trajectories: We tested Algorithm 1 for high order trajectories (with orders up to $m = 7$) with zero and non-zero initial and final conditions. Figure 2 shows trajectories computed by the algorithm for various values of m, $\Delta x = 50$, zero initial and final conditions ($x^i_s = x^i_f = 0$, $1 \le i \le m-1$); the state constraints in this example were $|x^{(i)}| \le 10^{2+i}$. These results show that motion time and smoothness increase with the trajectory order, because of the added limits on higher derivatives. The $m = 2$ profile in Figure 2 is the fastest, but it is not smooth as already its acceleration profile is discontinuous. The $m = 6$ profile, on the other hand, is the slowest, but it exhibits discontinuities only in its sixth derivative.

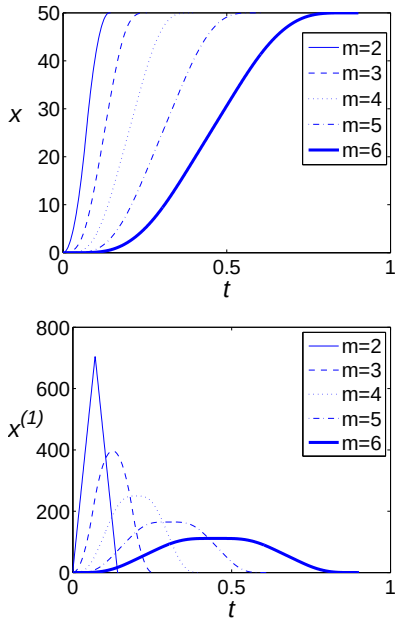

Fig. 2. Trajectory position (top) and velocity (bottom) for $m = 2, 3, 4, 5, 6$

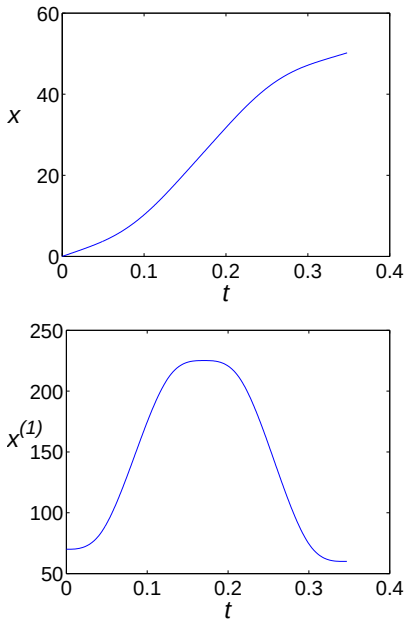

Fig. 3. 4th order trajectory position (top) and velocity (bottom) with nonzero initial and final conditions

Figure 3 shows a solution for the same setting as in Figure 2, for $m = 4$, except that the initial and final conditions on the velocity are nonzero: $x_s^1 = 70$ and $x_f^1 = 60$.

All of the these solutions share the familiar bang-zero-bang pattern.

2) Runtimes: The algorithm was implemented in C++ and was executed as a normal priority process on an Intel Pentium D 3.0 GHz processor, using a normal Microsoft windows XP system.

Table II shows the average runtimes of Algorithm 1 for various values of m, when executed with the same inputs as used to generate the trajectories in Figure 2. The parameter ε_i was set so that the accuracy is 0.01%, i.e., $\frac{x^i_{max} - x^i_{min}}{\varepsilon_i} = 0.0001$ for all i. For each m, the average runtime was computed by averaging several runs of the algorithm. As can be seen in Table II, the runtime changes exponentially with respect to m, since the algorithm is recursive in m. As m is always a small integer, that exponential dependence on m poses no practical problem.

We note that Algorithm 1 may be parallelized, as Steps 7 and 8 are independent of each other and could be executed in parallel. It is therefore possible to reduce the runtime by a factor of up to 2^{m-2} on a multi-core CPU, depending on the number of processes that can be executed in parallel.

Order	Number of runs	Average runtime [s]
3	1000	0.000074
4	1000	0.002141
5	10	0.0625
6	10	1.9515
7	10	80.064

TABLE II

RUNTIMES (SECONDS) FOR SEVERAL PROFILE ORDERS

III. MULTI-AXIS TRAJECTORIES

The single-axis trajectory planning algorithm can be used for solving multi-axis trajectory planning problems. We wish to compute a pair $\langle T, (x_1(t), \ldots, x_n(t)) \rangle$, where $(x_1(t), \ldots, x_n(t))$ is a function that connects two points in the Euclidean space $\mathbb{R}^n$ in minimal time, subject to the following constraints: (a) $x_j(t)$ satisfies given initial and final conditions at $t = 0$ and $t = T$,

$$x_j(0) = x_s^{j,0}\,, \quad x_j^{(i)}(0) = x_s^{j,i}\,, \tag{5}$$

$$x_j(T) = x_f^{j,0}\,, \quad x_j^{(i)}(T) = x_f^{j,i}\,, \tag{6}$$

where $1 \le i \le m-1$ and $1 \le j \le n$ (hereinafter the index j denotes the axis while i denotes the derivative order); (b) it is constrained by constant lower and upper bounds,

$$x_{min}^{j,i} \le x_j^{(i)}(t) \le x_{max}^{j,i}\,, \qquad t \in [0, T]\,, \tag{7}$$

where $1 \le i \le m$ and $1 \le j \le n$; and (c) the time $T = \int_0^T 1 dt$ is minimized.

To solve the multi-axis trajectory planning problem, we begin by first solving the n independent single-axis problems. For each axis $1 \le j \le n$, we get a single-axis trajectory, $x_j(t)$, that satisfies the initial and final conditions and kinematic bounds along that axis, and completes the journey in minimal time. The goal is now to combine those n single-axis trajectories, each reaching its final position at a possibly different time, into one multi-axis trajectory, $(x_1(t), \ldots, x_n(t))$. This is done by identifying the slowest axis, and then "stretching" the trajectories along the other axes so that they all reach their respective target at the same

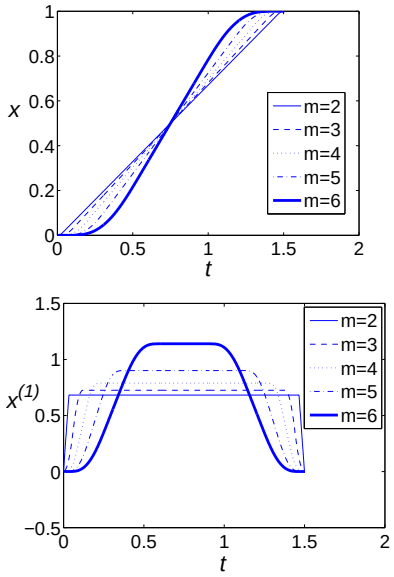

Fig. 4. Trajectories — position (top) and velocity (bottom) for $m = 2, \ldots, 6$, with $T_{\text{ext}} = 1.5$

time. The stretching procedure may be repeated until all single-axis trajectories reach their target at the same time.

In order to stretch a trajectory that was generated by Algorithm 1, we slightly modify the function ComputeTrajectory that the algorithm implements into a new function, called ComputeTrajectory-TimeLimit. That function receives the same inputs as ComputeTrajectory, and one additional positive scalar parameter denoted T_{ext}. It then proceeds to generate a trajectory that complies with the given inputs and takes minimal time that is no less than T_{ext}. To that end, if the modified algorithm generates a trajectory that reaches its goal in less than T_{ext}, it slows down the motion by decreasing the absolute value of v, the constant velocity during segment II. Specifically, if the value of v for the faster-than-T_{ext} solution is positive, the algorithm lowers the upper bound of the binary search so that it examines smaller values for v; if, on the other hand, the value of v for the faster-than-T_{ext} solution is negative, the algorithm sets it as the lower bound of the binary search to explore higher values for v. To achieve the above described functionality, the only modification that needs to be introduced is adding the next command after Step 11: **if** $(\tau_{2,v} > 0)$ and $(\tau_{1,v}+\tau_{2,v}+\tau_{3,v} < T_{\text{ext}})$ **then** $\Delta = -\Delta$.

To illustrate the effect of calling the modified function ComputeTrajectory-TimeLimit with a positive T_{ext}, we ran the algorithm with various values of m, $\Delta x = 1$, zero initial and final conditions ($x_s^i = x_f^i = 0$, $1 \le i \le m-1$), state constraints $|x^{(i)}| \le 2 \cdot 10^{i-1}$, and set $T_{\text{ext}} = 1.5$. The resulting trajectories, for $m = 1, \ldots, 6$, all with travel time of $T = 1.5$, are shown in Figure 4. Note that all trajectories use a cruising velocity well below the upper velocity constraint in order to comply with the given lower bound $T_{\text{ext}} = 1.5$ on the motion time.

Algorithm 2 solves the multi-axis problem, for any number of axes, iteratively by searching for the shortest common motion time. It saves in T_{max} the duration of the currently slowest trajectory, and in $sync$ the number of axes along which it already found a feasible solution with motion time T_{max} (or at least a motion time $T \in [T_{max}, T_{max} + \theta]$, where θ is a small parameter that determines the desired level of accuracy). To that end, after initializing those two variables (Step 1), it starts a cyclic loop over all axes (Steps 2-9) in search of the smallest value of T_{max} for which there is a feasible solution along each of the n axes with motion time $T \in [T_{max}, T_{max} + \theta]$. In order to synchronize the single-axis trajectories, Algorithm 2 computes a trajectory along each axis by invoking the modified Algorithm 1 (namely, the function ComputeTrajectory-TimeLimit) with T_{ext} that equals the current slowest motion time (Step 4). If ComputeTrajectory-TimeLimit succeeds in finding a feasible solution with $T \in [T_{max}, T_{max} + \theta]$, it records that success by increasing $sync$ (Step 5). Otherwise, the found feasible solution ends in time $T > T_{max} + \theta$; in that case, T_{max} is reset to T, and $sync$ is reset to 1 (Step 6). The loop ends only when $sync = n$ (Step 9), since then all single-axis trajectories have the same duration (up to a tolerable difference of θ). The algorithm then stops and returns the found feasible multi-axis solution (Step 10).

Algorithm 2 SynchronizeTrajectories

Input:
(1) The system order $m \ge 1$.
(2) The number $n \ge 1$ of trajectories that need to be synchronized.
(3) An accuracy parameter for the motion time, $\theta \ge 0$.
(4) Initial values: $x_s^{j,i}$, $0 \le i \le m-1$, $1 \le j \le n$.
(5) Final values: $x_f^{j,i}$, $0 \le i \le m-1$, $1 \le j \le n$.
(6) Bounds: $x_{min}^{j,i} \le 0 \le x_{max}^{j,i}$, $1 \le i \le m$, $1 \le j \le n$.

Output:
(1) Total motion time, $T > 0$.
(2) Trajectories $x_j(t)$, $1 \le j \le n$, that satisfy the input constraints, each spanning the time T.

1: $T_{max} = 0$; $sync = 0$.
2: $j = 1$.
3: **repeat**
4: $\langle T, x_j(t)\rangle \leftarrow$ ComputeTrajectory-TimeLimit$[m$, $(x_s^{j,0}, \ldots, x_s^{j,m-1}), (x_f^{j,0}, \ldots, x_f^{j,m-1})$, $\{(x_{min}^{j,i}, \ldots, x_{max}^{j,i})\}_{i=1}^{m}, T_{\text{ext}} = T_{max}]$
5: **if** $T - T_{max} \le \theta$ **then** $sync = sync + 1$
6: **else** $T_{max} = T$, $sync = 1$
7: $j = j + 1$.
8: **if** $j = n + 1$ **then** $j = 1$
9: **until** $sync = n$
10: Return $\langle T_{max}, (x_1(t), \ldots, x_n(t))\rangle$.

Example. This example demonstrates the use of Algorithm 2 to generate a trajectory that passes through four points in the plane with specified velocities and accelerations. The resulting trajectory demonstrates the algorithm's ability to produce a high-order continuous path.

Let $A = (0,0)$, $B = (20,0)$, $C = (20,20)$, and $D = (0,20)$ be four points in the $x-y$ plane. We wish to move a

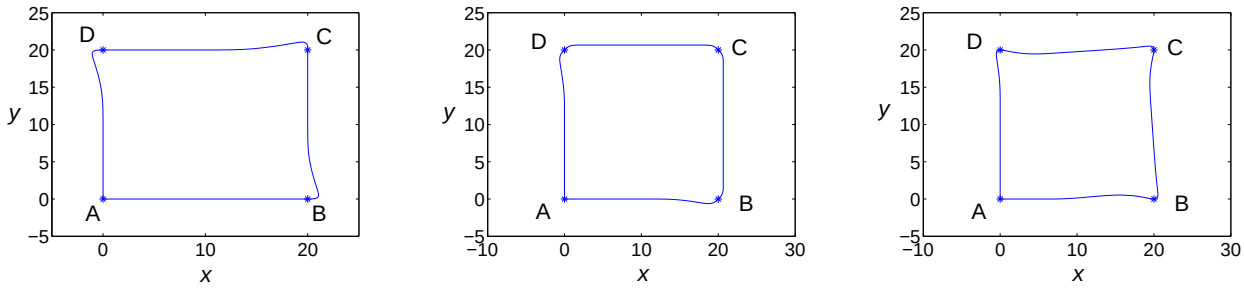

Fig. 5. Trajectories along a square path as described in Example 2: Scenario 2 (left), 3, and 4 (right).

body through these points, $A \to B \to C \to D \to A$, starting and finishing at rest. We consider trajectories of order $m = 3$ with the following bounds along each of the four motion segments: $|x^{(i)}| \leq 10^{2+i}$, $1 \leq i \leq m$.

We examine four scenarios that differ in the inner corner velocities and accelerations, at B, C and D. In Scenario 1, the body reaches a full stop in each inner corner before continuing its motion. In Scenario 2, the velocity in each inner corner is 50 in the direction leading to the corner, and the acceleration there is zero. In Scenario 3, the corner velocities are counterclockwise 45^o rotations of the corresponding corner velocities in Scenario 2 (so that the velocity at B, for example, is $(50/\sqrt{2}, 50/\sqrt{2})$ instead of $(50, 0)$ as it was in Scenario 2); the acceleration in each corner is set to zero. This adjustment of the velocity to the right-angle turn in each corner results in a shorter overall motion time with respect to Scenario 2. Finally, Scenario 4 is identical to Scenario 2 except for the acceleration values in the inner corners. These acceleration values are designed so that the moving body begins accelerating for the next motion segment earlier, in order to reduce the overall motion time. The acceleration values are $(-2000, 2000)$ at B, $(-2000, -2000)$ at C, and $(2000, -2000)$ at D. The trajectories in Scenarios 2, 3 and 4 are shown in Figure 5. (The trajectory in Scenario 1 is not shown since it is a perfect square.)

As expected, the motion time in Scenario 1 is the longest, $T_1 = 0.743$. In Scenario 2, where the body is not forced to stop in each inner point, it is $T_2 = 0.701$. In Scenario 3, in which the corner velocities are better adjusted to the counterclockwise turns in each corner, the motion time reduces to $T_2 = 0.683$. Finally, in Scenario 4, with the added benefit of acceleration conditions, the body completes the journey in time $T_4 = 0.620$.

IV. Conclusion

This paper presented a trajectory planning algorithm for single and multi-axis a trajectories, subject to general initial and final conditions and derivative bounds. It is based on a recursive process that reduces the original high order trajectory problem to lower order problems. The recursion is applied until reaching low orders ($m = 1$ or $m = 2$) for which a direct solution is available. The resulting algorithm is simple and efficient, as was demonstrated in our runtime results. The proposed algorithm can be used off-line to produce high order trajectories, as well as on-line in applications where efficiency and reactiveness are essential.

In this paper we focused on multi-axes trajectories with no concern to geometrical constraints, apart from the initial and final positions. Extending our algorithm to account for geometrical constraints, such as imposed by obstacles or by a specified path, is a subject of future research.

References

[1] I.H. Aguilar and D. Sidobre. On-line trajectory planning of robot manipulators end effector in cartesian space using quaternions.

[2] X. Broquère, D. Sidobre, and I. Herrera-Aguilar. Soft motion trajectory planner for service manipulator robot. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2808–2813. IEEE, 2008.

[3] A.E. Bryson and Y.C. Ho. *Applied Optimal Control*. Blaisdell Publishing Co., Cambridge, MA, 1969.

[4] R. Haschke, E. Weitnauer, and H. Ritter. On-line planning of time-optimal, jerk-limited trajectories. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3248–3253. IEEE.

[5] I. Herrera-Aguilar and D. Sidobre. Soft motion trajectory planning and control for service manipulator robot.

[6] T. Kroger, A. Tomiczek, and F.M. Wahl. Towards on-line trajectory computation. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 736–741. IEEE, 2006.

[7] T. Kroger and F.M. Wahl. Online trajectory generation: basic concepts for instantaneous reactions to unforeseen events. *Robotics, IEEE Transactions on*, 26(1):94–111, 2010.

[8] P. Lambrechts, M. Boerlage, and M. Steinbuch. Trajectory planning and feedforward design for high performance motion systems. In *American Control Conference, 2004. Proceedings of the 2004*, volume 5, pages 4637–4642. IEEE.

[9] S. Liu. An on-line reference-trajectory generator for smooth motion of impulse-controlled industrial manipulators. In *Advanced Motion Control, 2002. 7th International Workshop on*, pages 365–370. IEEE, 2002.

[10] S. Macfarlane and E.A. Croft. Jerk-bounded manipulator trajectory planning: Design for real-time applications. *Robotics and Automation, IEEE Transactions on*, 19(1):42–52, 2003.

[11] K.D. Nguyen, I.M. Chen, and T.C. Ng. Planning algorithms for s-curve trajectories. In *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*, pages 1–6. IEEE, 2007.

[12] K. Petrinec and Z. Kovacic. Trajectory planning algorithm based on the continuity of jerk. In *Control & Automation, 2007. MED'07. Mediterranean Conference on*, pages 1–5. IEEE, 6.

[13] A. Piazzi and A. Visioli. Global minimum-jerk trajectory planning of robot manipulators. *Industrial Electronics, IEEE Transactions on*, 47(1):140–149, 2000.