Planning High Order Trajectories with General Initial and Final Conditions and Asymmetric Bounds

Ben Ezair[1,3] Tamir Tassa[1] Zvi Shiller[2‡]

Abstract

This paper presents a trajectory planning algorithm for linear 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 orders $2-6$.

1 Introduction

This paper addresses the problem of planning time-efficient trajectories for linear multi-axis systems with arbitrary initial and final states, subject to constant constraints on any number of trajectory derivatives. In the context of robot motion, the trajectory planning problem consists of generating a smooth trajectory that connects given initial and final conditions, is optimal with respect to some cost function, and satisfies given bounds on a number of its time derivatives. The number of derivatives considered, also called the trajectory order, usually reflects the order of the robot dynamic model, whereas the bounds on the time derivatives reflect state and control constraints of the robot system.

Pre-computed feasible trajectories can be used to accurately guide a dynamic system to the destination state by serving as the desired inputs to the system's joint controllers. A carefully selected trajectory can prevent the joint controllers from reaching saturation, a common cause for tracking errors. In addition, they are essential when coordinated motion of several joints is desired. When used in a feed-forward fashion, they can significantly reduce tracking errors by reducing the magnitude of the tracking error needed to drive the system along the trajectory [22].

*[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. shiller@ariel.ac.il

‡[3]The research of the first author was partially supported by The Open University of Israel's Research Fund (grant no. 32046).

High order trajectories are needed to account for actual and unmodeled system dynamics. Unmodeled dynamics may arise from unmodeled actuators and drivers, and from unmodeled flexible modes. For both cases, high order smooth trajectories are desired. In the case of unmodeled actuator and driver dynamics, a smooth trajectory may be easier to follow when using it as a control input, since it may not demand a response that is beyond the capabilities of the actuator/driver system. For example, using a third order trajectory to drive a DC motor assumes that the motor voltage serves as the control input. Hence, by choosing a third order trajectory in this context it is assumed that the motor can react instantaneously to abrupt switches in the voltage signal. In practice, however, the driver may not be able to generate instantaneous voltage changes, thus causing some delay in the system response, or even completely ignoring (filtering) particularly short voltage pulses. Such delays and inaccuracies are the cause of tracking errors. Tracking errors could be avoided if the trajectory was of a higher order, for which the third derivative, and hence the voltage signal, is smooth. A further discussion of the benefits of higher order trajectories, including experimentation, can be found in [16, 13].

In the case of a flexible system, a high order smooth trajectory may not excite high frequency flexible modes, thus resulting in slower, yet vibration-free motion. It should be noted that the clear benefits of high order smooth trajectories come at the cost of slower motion. The appropriate balance between motion time and smoothness should be set in accord with the characteristics of the application at hand. Some applications may demand extreme accuracy and, hence, should use a high order trajectory; in other applications, a fast second or third order trajectory may suffice. One of the advantages of our algorithm is that this choice can be easily made by specifying the desired trajectory order.

One challenge in trajectory planning is to generate a high order smooth trajectory on the fly, during motion, to account for the current motion state and to respond to events that are identified during motion [11]. This, in turn, requires the algorithm to be highly efficient and to be able to accept arbitrary initial and final conditions.

The algorithm proposed herein generates time-efficient trajectories between arbitrary initial and final states for a linear system of any given order, subject to constant and asymmetric state and control constraints. Given the motion constraints, the algorithm generates a feasible trajectory while attempting to minimize motion time. The algorithm is recursive, in the sense that it reduces the original problem of order m to smaller problems of order $m-1$. The recursion is based on a binary search, which contributes to the algorithm's computational efficiency. Multi-axis trajectories are computed by synchronizing single-axis trajectories to reach their respective targets at the same time.

1.1 Related work

Early work on trajectory planning of multi-axis systems was based on decomposing the problem into path planning and trajectory planning. It consists of first generating a feasible path, then computing the time optimal velocity profile along the specified path [2, 18, 24, 25, 26], and finally modifying the path to obtain the time optimal trajectory [1, 23]. The optimal velocity profile is computed by switching between the maximum and minimum allowed acceleration values along the path, for a second order system [2]. Adding velocity constraints results in a *bang-zero-bang* structure of the acceleration profile [21]. The optimization along

a specified path was extended to account for non-linear third order systems, subject to general jerk constraints [28]. While this approach allowed solving difficult multi-axis problems with non-linear second or (at most) third order dynamics and any obstacle constraints, it is off-line in nature. Higher order dynamics are often needed to produce smooth trajectories that account for high order actuator dynamics and high order vibration modes. Computing smooth high order trajectories poses a special challenge in online motion planning, namely, applications where the trajectory is updated "on the fly" to account for tracking errors and changes in the environments. There are several different approaches used by more recent works for generating smooth trajectories. One approach uses polynomials or other functions to approximate the desired trajectories, often optimizing a parametric curve in order to achieve near optimal results [15, 17, 19]. Piazzi and Visioli [19] optimize cubic splines to minimize jerk for a specified motion time. Petrinec and Kovacic [17] use fourth and fifth order polynomials and various heuristics to produce smooth multi-axis trajectories. Macfarlane and Croft [15] compute trajectories that are represented by fifth order polynomials.

Another approach for trajectory generation relies on Pontryagin's maximum principle [20], which for time optimal control for linear systems with state constrains suggests a bang-zero-bang structure for the control input. Hence, the sought after trajectory is divided into segments, where the value of the highest derivative is constant in each segment, equaling its upper or lower bounds, or zero.

Liu [14] presents an algorithm that produces a third order trajectory that is constructed by dividing the trajectory to seven segments. They limit the trajectory to be one of several forms that are possible for a seven segment trajectory. This allows them to reduce the problem to two steps: first identifying the best form out of a finite set, then solve the equations for that form to calculate the exact trajectory. This basic approach can be extended to produce multi-axis trajectories by synchronizing several single-axis trajectories [9, 3, 10]. They use a similar method for each individual axis but also lower the derivative bounds of faster axes in order to synchronize them with the slower ones.

Haschke et al. [8] also produce a multi-axis third order trajectory based on a seven segment approach. They emphasize the online capabilities of their algorithm that is designed to produce a third order halting trajectory. The main tool used here is a manipulation of the acceleration profile to slow the trajectory. This allows slowing down single-axis trajectories so that they comply with the derivative bounds, and not overshoot the target position; it can also be used to synchronize them with slower axes. Works by Kroger et al. [11, 12] also focus on online algorithms for second and third order trajectories, using a thorough analysis of possible acceleration profiles to handle more general initial and final conditions. These works in many ways formalize the steps used by other algorithms. The first step of identifying the general form of the trajectory is accomplished through the use of decision trees that map out all possible forms for the requested trajectory. Each of the possible forms can then be used to define a system of equations, that is solved to get all possible solutions, amongst which the optimal solution can be found. This also allows a more thorough approach to be used to synchronize multiple axes: since all possible solutions for each axis are known, the best motion time to synchronize all axes can be easily identified.

Lambrechts et al. [13] produce fourth order trajectories. Here too the basic idea is dividing the trajectory into segments where the highest derivative has one of three constant

values. A fourth order trajectory, however, requires more segments than a third order trajectory, and is much more complex. They solve this by limiting themselves to dealing with rest to rest motion, and assuming a single specific trajectory form. This form has eight segments with a non-zero snap (the fourth derivative of position), and all of these segments have the exact same time length. By progressively reducing the length of these segments, so that they comply with the motion derivative limits and the target position, they then calculate the desired fourth order trajectory. Nguyen et.al. [16] 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. This recursive structure is used to construct an algorithm that is a generalization to an arbitrary order of the fourth order trajectory generation algorithm described in [13]. The use of recursive S-curves forms is also the basis for the algorithm which we present herein.

1.2 Our algorithm

This paper presents a novel algorithm for planning trajectories for linear systems of any order between arbitrary initial and final states (position and its time derivatives), subject to given constant state and control constraints; the algorithm 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 modular, as it may accept any external solver for the basic trajectory generation problem which is solved directly in order to terminate the recursion. (Here, we offer to stop the recursion at order $m=2$, for which a simple analytical solution exists. However, it can also continue until $m=1$, for which the basic solution is trivial, or until any other order for which a direct and efficient solution exists or will become available.)

The algorithm is efficient, as demonstrated in several experiments (see Table 2 in Section 2.3). Finally, the basic algorithm is extended to generate multi-axis trajectories by synchronizing independent single-axis trajectories to reach their respective targets at the same time.

Table 1 compares our algorithm with the leading comparable algorithms that consider similar settings as ours; i.e., algorithms that deal with linear systems subject to constant constraints and attempt to minimize motion time. (All of those algorithms were reviewed in Section 1.1.)

All of the comparable algorithms are limited either 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.

It should be noted that algorithms for non-linear systems exist, e.g. [5, 7, 28], however, they are limited (at least for now) to order $m \leq 3$, use other cost functions, or are limited to a specified path (usually using an arc length parametrization). Algorithms treating non-linear systems, which are typically computationally intensive, may not be suitable for multi-axis non-linear systems that need to react on-line to fast changing environments. One approach to handling such cases is to approximate the non-linear system with a linear

model, and then use an efficient on-line trajectory generator. The resulting trajectory may not be optimal, but it would be smooth, of the desired order, not constrained to a specified path, and satisfying approximate (constant) state and control constraints. So, in effect, linear trajectory generators may be useful for both linear and non-linear systems. It is in this context that we propose our algorithm as a contribution to the existing class of "linear" trajectory generators.

Ref.	Order	Initial & Final Conditions	Constraints	Online	Optimal
[11]	2	general	symmetric	yes	yes
[3]	3	zero acceleration	symmetric	yes	yes
[8]	3	ends at rest	symmetric	yes	yes
[12]	3	zero final acceleration	symmetric	yes	yes
[13]	4	rest to rest	symmetric	no	no
[16]	any	rest to rest	symmetric	no	no
Ours	any	general	asymmetric	yes	no

Table 1: Comparison of trajectory generation algorithms

2 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 $t = T$,

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

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

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

$$x_{min}^i < 0 < x_{max}^i\,,\quad 1 \le i \le m \tag{3}$$

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

and (c) the time $T = \int_0^T 1dt$ is minimized. The number $m \ge 1$ 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 [4].

The solution for the case $m = 1$ is trivial, consisting of a constant velocity motion. A solution for $m = 2$ is given in [11]. 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.

2.1 A single-axis trajectory planning algorithm

2.1.1 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 (see Figure 1 for the case $m = 3$). 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, repeats recursively since the velocity profile, as well as the profiles of higher derivatives, consist of two or more S-curve segments.

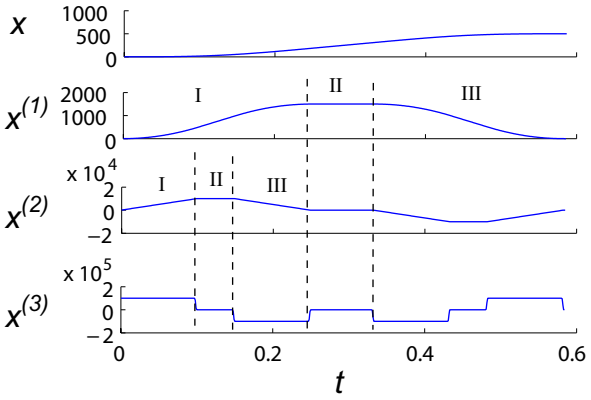

Figure 1: The recursive structure of the trajectory

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 $x(t)$.

2.1.2 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).

2.1.3 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 and further discussion can be found in Appendix A. 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. 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.

2.2 Complexity

The main computational effort in Algorithm 1 is the binary search, performed in steps 4-18, for the optimal value of v in the interval $[x^1_{min}, x^1_{max}]$. All other operations performed by the algorithm (including the solutions for $m = 1$ and $m = 2$) take constant time. Since the binary search terminates when the interval size becomes smaller than or equal to ε_1, it executes at most $\log_2 \frac{x^1_{max}-x^1_{min}}{\varepsilon_1}$ iterations. In each of those iterations, the algorithm invokes two recursive calls for solving problems of order $m-1$; in addition, it performs the computations in Steps 9-17. The time complexity of the latter computations can be bounded by a constant. If this constant is denoted d, and the time for solving the problem of order m is denoted T_m, then:

$$T_m = \left(\log_2 \frac{x^1_{max} - x^1_{min}}{\varepsilon_1}\right) \cdot (2T_{m-1} + d)\,. \tag{5}$$

This search repeats for each level i in the recursion in the range $[x^i_{min}, x^i_{max}]$; it terminates when the interval size becomes smaller than or equal to ε_i. Solving the recursive equation (5) yields T_m in terms of T_2, d, the interval lengths $(x^i_{max} - x^i_{min})$, and ε_i:

$$T_m = T_2 \cdot \prod_{i=1}^{m-2} (2C_i) + d \cdot \left[\sum_{j=1}^{m-2} \left(2^{j-1} \prod_{i=1}^{j} C_i\right)\right]\,, \tag{6}$$

where

$$C_i = \log_2 \frac{x^i_{max} - x^i_{min}}{\varepsilon_i}\,, \tag{7}$$

Let $\hat{C} = \max_{1\le i\le m-2} C_i$. Then, by Eq. (6):

$$T_m \le T_2 \cdot (2\hat{C})^{m-2} + d\hat{C} \cdot \left[\sum_{j=1}^{m-2} (2\hat{C})^{j-1}\right] = T_2 \cdot (2\hat{C})^{m-2} + d\hat{C} \cdot \frac{(2\hat{C})^{m-2} - 1}{2\hat{C} - 1}\,.$$

As $\hat{C} > 1$ for any reasonable setting of ε_i, we conclude that

$$T_m \leq (2\hat{C})^{m-2} \cdot \left(T_2 + \frac{d\hat{C}}{2\hat{C}-1} \right) \leq (2\hat{C})^{m-2}(T_2 + d)\,.$$

Thus

$$T_m \leq \left(2 \max_{1 \leq i \leq m-2} \log_2 \frac{x^i_{max} - x^i_{min}}{\varepsilon_i} \right)^{m-2} (T_2 + d)\,. \tag{8}$$

The total time taken by the algorithm is determined by the range of values that have to be searched at each level $x^i_{max} - x^i_{min}$, the desired accuracy in that level ε_i, and the problem order m. The more accurate is the search (lower ε_i), the longer is the computation time. Similarly, the larger is the range, the longer is the search. However, since this is a binary search, the overall dependence on these factors is only logarithmic. In contrast, the dependence of the runtime on m is exponential. However, as m denotes the order of the problem, its value in practical applications is typically very small (note that most studies thus far concentrated on $m \leq 3$ and only few studies considered orders up to $m = 5$). For such values of m, and even higher ones, the algorithm is still computationally practical, as demonstrated by our experimentation in the next section.

2.3 Experiments: Single-axis trajectories

Algorithm 1 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. We tested the algorithm for high order trajectories (with orders up to $m = 7$) with zero and non-zero initial and final conditions. The bounds used for the motion derivatives in these examples were chosen high (and low) so that motion derivatives reach their maximal (and minimal) values. Tight bounds on the high derivatives might result in frequent switches of the highest derivative, which might not let the lower derivatives reach (and sustain) their extreme values.

Figure 2 shows trajectories computed by the algorithm for various values of m, $\Delta x = 50$, zero initial and final conditions ($x^i_s = x^i_f = 0$, $1 \leq i \leq m-1$); the state constraints in this example were $|x^{(i)}| \leq 10^{2+i}$. 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.

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: $x^1_s = 70$ and $x^1_f = 60$. All of these solutions share the familiar bang-zero-bang pattern.

Figure 4 shows an asymmetric third order trajectory that was computed by our algorithm. For this trajectory we used $\Delta x = 50$, and zero initial and final conditions ($x^i_s = x^i_f = 0$, $1 \leq i \leq m-1$); the derivative constraints were $-1000 \leq x^{(1)} \leq 200$, $-10000 \leq x^{(2)} \leq 2000$, and $-100000 \leq x^{(3)} \leq 20000$. The capability of our algorithm to support asymmetric constraints may be useful in situations where such asymmetry is part

of the system dynamics: e.g., in a standard car where acceleration is created by a motor, while deceleration is achieved through the use of a mechanical brake.

Figure 5 shows another trajectory computed by the algorithm. For this fourth order trajectory we used $\Delta x = 20$, and zero initial and final conditions ($x_s^i = x_f^i = 0$, $1 \leq i \leq m-1$). The derivative constraints for this case were: $-10 \leq x^{(1)} \leq 10$, $-1 \leq x^{(2)} \leq 1$, $-10 \leq x^{(3)} \leq 10$, and $-10 \leq x^{(4)} \leq 10$. Using tight bounds on the high derivatives resulted in most derivatives not reaching their extreme values, except the acceleration for which the bounds were particularly tight.

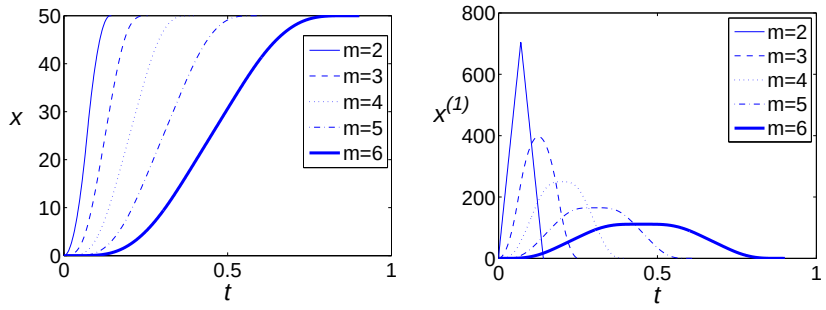

Figure 2: Trajectory position (left) and velocity (right) for $m = 2, 3, 4, 5, 6$

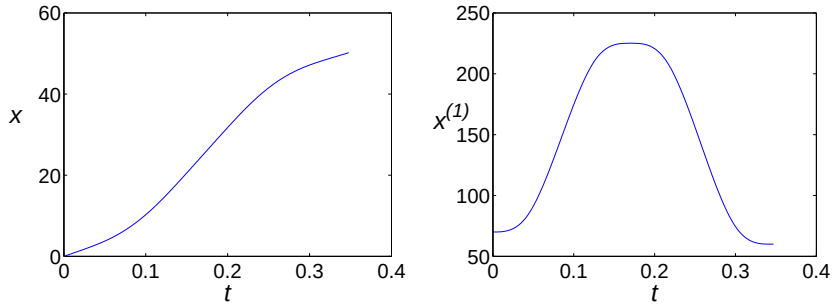

Figure 3: Fourth order trajectory position (left) and velocity (right) with nonzero initial and final conditions

Table 2 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., $\frac{x_{max}^i - x_{min}^i}{\varepsilon_i} = 0.0001$ for all i. For each m, the average runtime was computed by averaging several runs of the algorithm.

As can be seen in Table 2, the runtime changes exponentially with respect to m, as discussed in Section 2.2. However, as m is typically a small integer, the algorithm remains computationally practical. In particular, the runtimes for $m \leq 5$ are practical for both offline and online applications. The runtime for $m = 6$ might call for code optimization in order to be practical for online applications, but it is certainly practical for offline applications. The

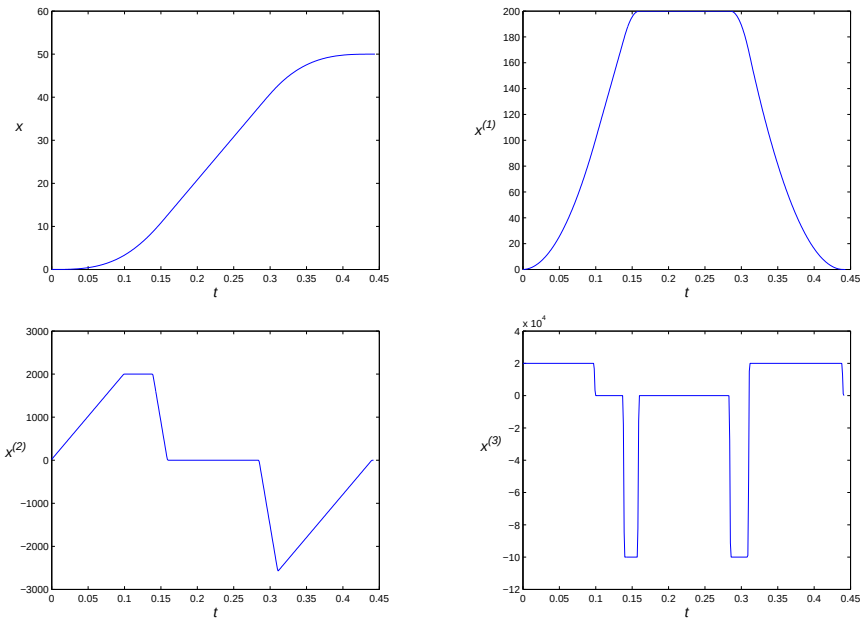

Figure 4: A third order trajectory with asymmetric bounds (position and all derivatives).

runtime for $m = 7$ (an order which seems currently to be beyond the need of any practical application) renders the algorithm still practical for offline applications.

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 2: Runtimes (seconds) for several profile orders

3 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

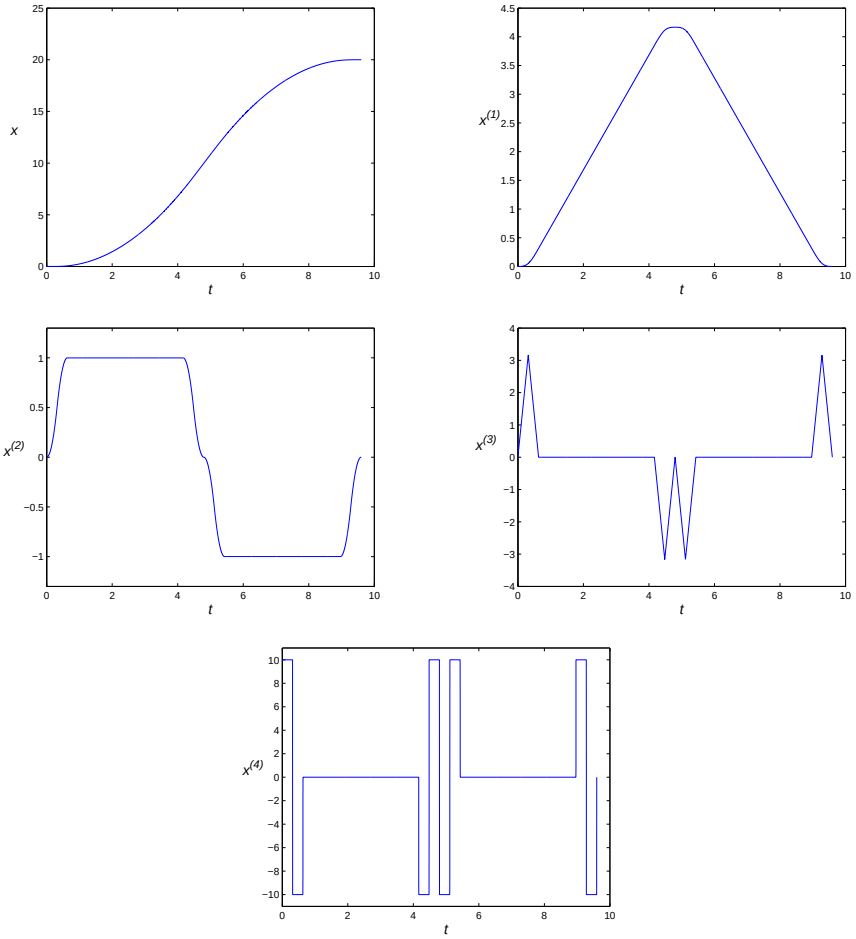

Figure 5: A fourth order trajectory with low bounds for the higher derivatives (position and all derivatives).

$t = T$,

$$x_j(0) = x_s^{j,0}\,, \quad x_j^{(i)}(0) = x_s^{j,i}\,, \tag{9}$$

$$x_j(T) = x_f^{j,0}\,, \quad x_j^{(i)}(T) = x_f^{j,i}\,, \tag{10}$$

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

$$x_{min}^{j,i} \leq x_j^{(i)}(t) \leq x_{max}^{j,i}\,, \qquad t \in [0, T]\,, \tag{11}$$

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

To solve the multi-axis trajectory planning problem, we begin by first solving the n independent single-axis problems. For each axis $1 \leq j \leq n$, we get a single-axis trajectory, $x_j(t)$, that satisfies the initial and final conditions and kinematic bounds along that axis,

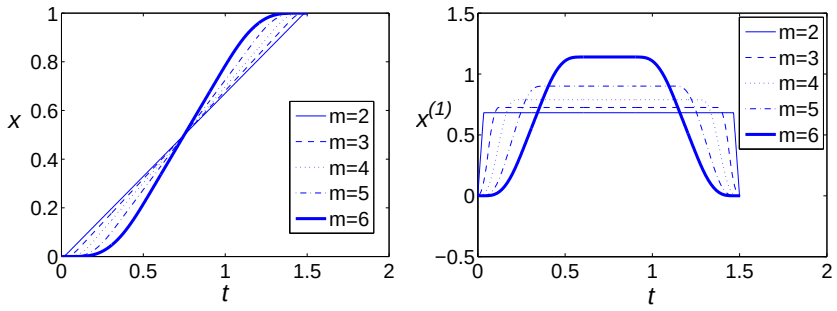

Figure 6: Trajectories — position and velocity for $m = 2, \ldots, 6$, with $T_{\text{ext}} = 1.5$

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 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 this 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 6. 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.

It should be noted that in this case, the algorithm succeeded in generating a trajectory that ends at the exact time $T_{\text{ext}} = 1.5$. There are, however, cases where the algorithm generates a trajectory that completes the journey in time significantly larger than T_{ext}. Such situations occur since the set of all *feasible times* for a single-axis trajectory planning problem is not necessarily continuous, as shown in [12]. Namely, if the optimal solution for

a given single-axis problem is, say, $\langle 10, x(t) \rangle$, it does not imply that a solution exists for every $T \geq 10$. The range of possible completion times may be discontinuous, and may take the form, say, $[10, 15] \cup [20, \infty)$.

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 this 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).

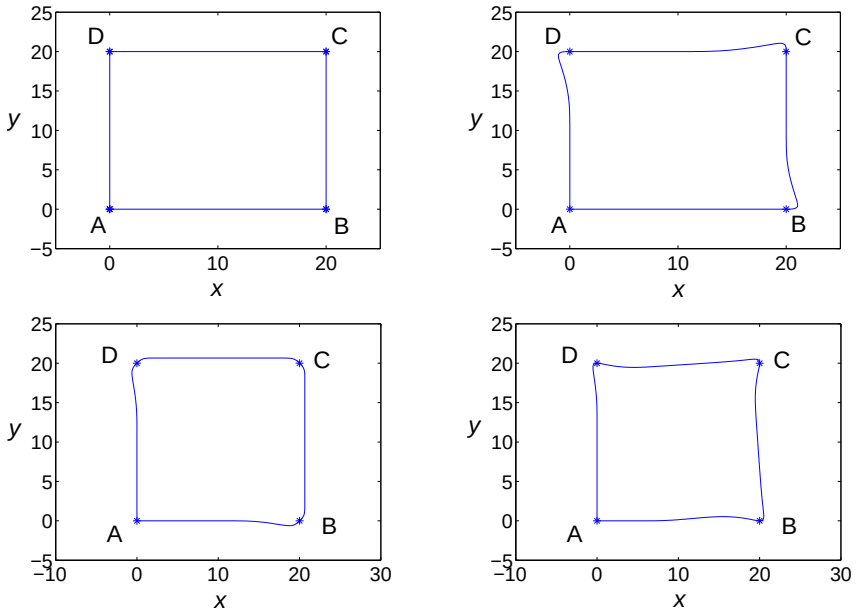

Figure 7: Trajectories along a square path as described in Example 1: Scenario 1 (top left), 2 (top right), 3 (bottom left), and 4 (bottom right).

Algorithm 1 ComputeTrajectory

Input:

(1) The system order $m \geq 2$.

(2) Initial and final states: x_s^i, x_f^i, $0 \leq i \leq m-1$.

(3) Bounds: x_{min}^i, x_{max}^i, $1 \leq i \leq m$.

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

if $m = 2$ **then** return the analytic solution and stop.
$\Delta x = x_f^0 - x_s^0$.
$v_{min} = x_{min}^1$, $v_{max} = x_{max}^1$, $v = \hat{v} = x_{max}^1 + 1$.
repeat
  $last\_v = v$.
  $v = (v_{max} + v_{min})/2$.
  $\langle \tau_{1,v}, v_1(t) \rangle \leftarrow$ ComputeTrajectory$[m-1,$
  $(x_s^1, \ldots, x_s^{m-1}), (v, 0, \ldots, 0), \{(x_{min}^i, x_{max}^i)\}_{i=2}^m]$
  $\langle \tau_{3,v}, v_3(t) \rangle \leftarrow$ ComputeTrajectory$[m-1,$
  $(v, 0, \ldots, 0), (x_f^1, \ldots, x_f^{m-1}), \{(x_{min}^i, x_{max}^i)\}_{i=2}^m]$
  $\Delta x_1 = \int_0^{\tau_{1,v}} v_1(t)dt$; $\Delta x_3 = \int_0^{\tau_{3,v}} v_3(t)dt$.
  $\Delta = \Delta x - \Delta x_1 - \Delta x_3$; $\tau_{2,v} = \Delta / v$.
  if $\tau_{2,v} \geq 0$, $\hat{v} = v$.
  if $|v_{max} - v_{min}| \leq \varepsilon$ **then**
    $v = v_{min} = v_{max} = \hat{v}$.
    if $(\hat{v} = x_{max}^1 + 1)$ **then** stop and output "Failed".
  elseif $\Delta > 0$ **then** $v_{min} = v$
  elseif $\Delta < 0$ **then** $v_{max} = v$
  else $last\_v = v$ **endif**
until $last\_v = v$
$T = \tau_{1,v} + \tau_{2,v} + \tau_{3,v}$.

$$x'(t) = \begin{cases} v_1(t) & [0, \tau_{1,v}] \\ v & [\tau_{1,v}, \tau_{1,v} + \tau_{2,v}] \\ v_3(t - \tau_{1,v} - \tau_{2,v}) & [\tau_{1,v} + \tau_{2,v}, T] \end{cases}$$

Return $\langle T, x(t) \rangle$, where $x(t) = \int_0^t x'(\tau)d\tau + x_s^0$.

Algorithm 2 SynchronizeTrajectories

Input:

(1) The system order $m \geq 1$.

(2) The number $n \geq 1$ of trajectories that need to be synchronized.

(3) An accuracy parameter for the motion time, $\theta \geq 0$.

(4) Initial values: $x_s^{j,i}$, $0 \leq i \leq m-1$, $1 \leq j \leq n$.

(5) Final values: $x_f^{j,i}$, $0 \leq i \leq m-1$, $1 \leq j \leq n$.

(6) Bounds: $x_{min}^{j,i} \leq 0 \leq x_{max}^{j,i}$, $1 \leq i \leq m$, $1 \leq j \leq n$.

Output:

(1) Total motion time, $T > 0$.

(2) Trajectories $x_j(t)$, $1 \leq j \leq n$, 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** $T - T_{max} \leq \theta$ **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 $\langle T_{max}, (x_1(t), \ldots, x_n(t)) \rangle$.

3.1 Examples of multi-axis trajectories

3.1.1 Example 1

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 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 these scenarios are shown in Figure 7.

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 $T_4 = 0.620$.

3.1.2 Example 2

This example demonstrates the use of Algorithm 2 to generate trajectories for a typical task of a mobile robot that needs to pass through a few specified points at specified velocities and accelerations. Figure 8 show three trajectories that pass through the points: $(0, 0)$, $(2, 3)$, $(4, 1)$, and $(5, 5)$ at velocities $(25, 7)$, $(5, 3)$, $(22, 25)$, and $(14, -25)$. At each point, the velocity vector is marked by a line. The three trajectories differ in their order of their control input: third ($m = 3$), fourth ($m = 4$), and fifth ($m = 5$). The bounds on the derivatives for both axes were set to: $|x^{(1)}| \leq 1000$, $|x^{(2)}| \leq 10000$, $|x^{(3)}| \leq 100000$, $|x^{(4)}| \leq 5000000$, $|x^{(5)}| \leq 100000000$. The rather large bounds for the higher derivatives were chosen to allow the lower derivatives to reach higher values and get an overall faster motion. It should be noted that large is a relative term here as for lower order trajectories these values are effectively set to infinity.

All trajectories pass through the specified points at the specified velocities. They differ, however, in their smoothness and geometric path due to the larger number of switches of the higher order trajectories.

The fastest motion time of $0.3s$ was achieved by the third order trajectory; the motion times for the fourth and fifth order trajectories were $0.41s$ and $0.67s$, respectively.

Figure 9 shows the acceleration along the y-axis for the third and fifth order trajectories. The high order trajectory is smoother, exerting, as a result, lower accelerations along the path. This explains the differences in the geometric shape of the paths shown in Figure 8: the paths generated by lower order trajectories are visually smoother than those generated by high order trajectories, because the higher accelerations applied by the lower order trajectory made it possible to connect the specified velocities with smooth curves, followed at high speeds; the higher order trajectories applied lower accelerations, which required to slow down before making the turn and accelerating to velocity at the next point. This illustrates the use of high order trajectories to limit the effort applied by the control system while achieving the desired motion. In a real application, this limited effort will likely result in a more accurate motion, smaller power consumption, and lower wear on the system.

For completeness, Figure 10 shows all derivatives for the x and y axes in the exemplified third order trajectory.

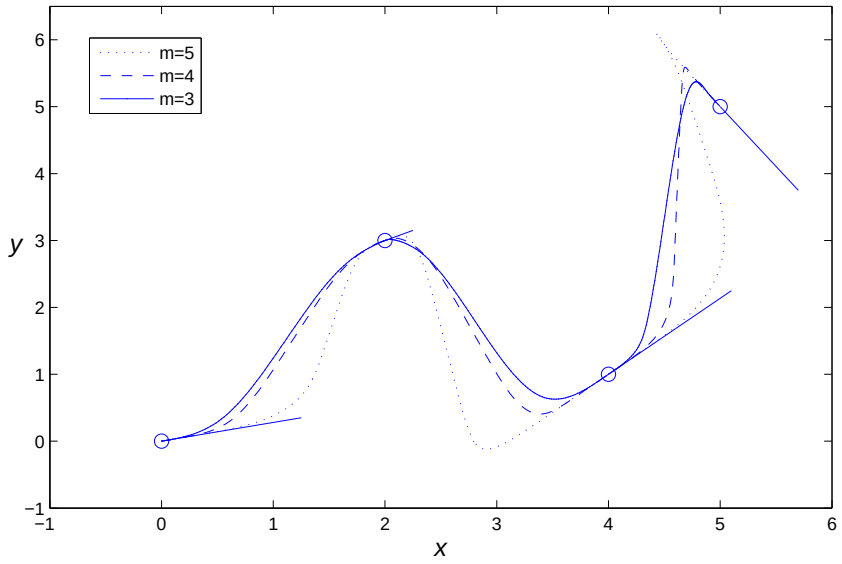

Figure 8: Trajectories along a path.

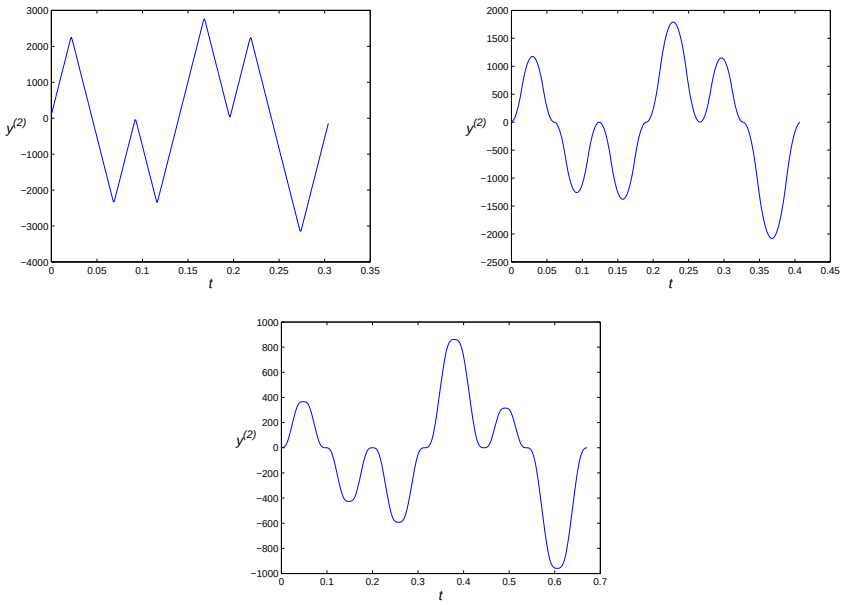

Figure 9: Acceleration profiles for the third (top left) fourth (top right) and fifth (bottom) order trajectories in Example 2.

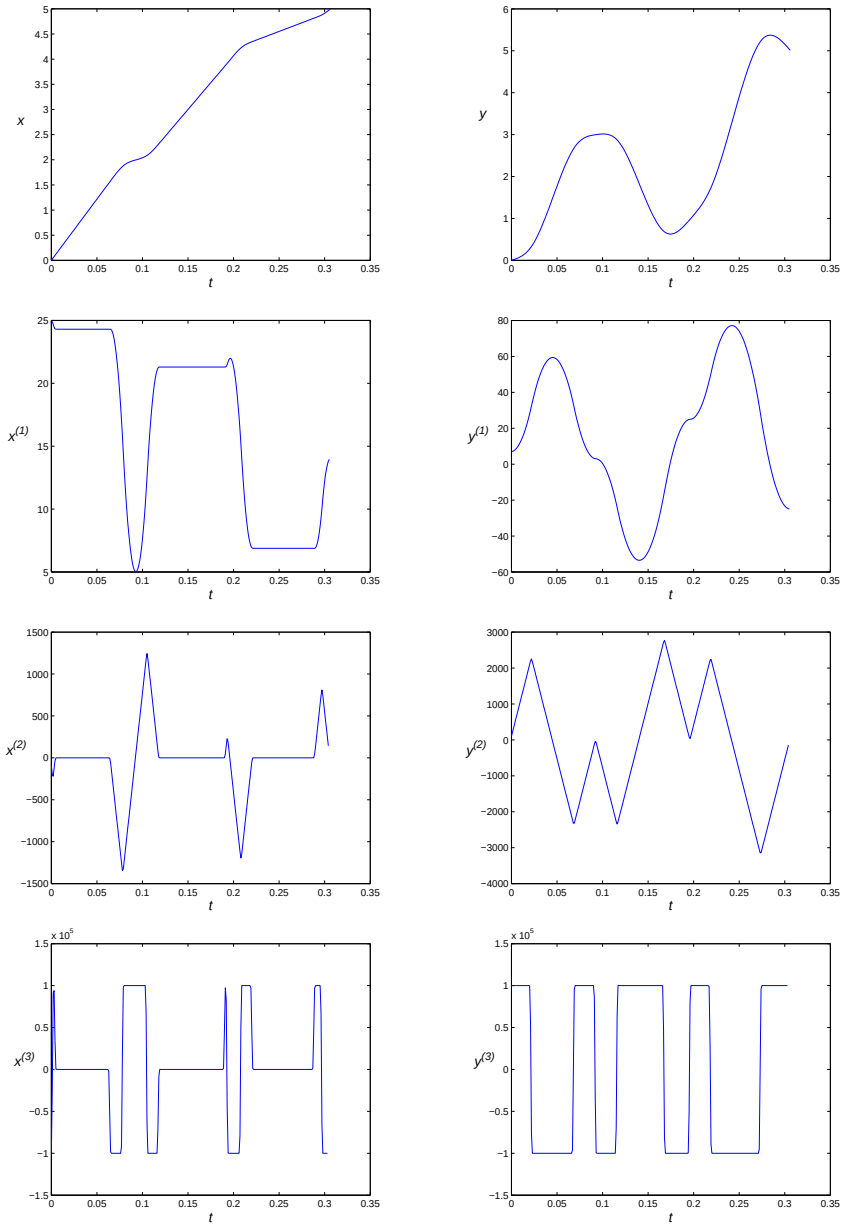

Figure 10: All derivatives for the third order trajectory in Example 2.

3.1.3 Example 3

In this example, a simple high level planner uses our algorithm to generate the trajectories for a planar three-wheeled non-holonomic mobile robot. The underlying simulation engine used here is the ODE given in [27]. An executable file for running the simulation can be found at [6].

The trajectories are shown in Figure 11. Figure 12 shows a diagram of the robot. The parameters of this robot are given in Table 3. It is controlled by specifying the longitudinal speed $v = \omega r$ and the steering rate $\beta^{(1)}$, where ω is the rotational speed of the rear wheels, r is the wheel radius, and β is the steering angle of the front wheel. The bounds used for the two control inputs are:

- $|\beta^{(1)}| < 5$ [rad/s], $|\beta^{(2)}| < 10$ [rad/s^2], $|\beta^{(3)}| < 10$ [rad/s^3]
- $|\omega| < 5$ [rad/s], $|\omega^{(1)}| < 10$ [rad/s^2], $|\omega^{(2)}| < 10$ [rad/s^3]

A high level planner then uses our algorithm to compute a sequence of control inputs that would drive the robot to the target position from any given state. The trajectory input parameters are determined by a simple state machine, which selects between three motion primitives: (a) a straight line motion, (b) a right or left curve at a constant speed, and (c) a transition from a curve to a straight line. Alternating between these three motion primitives, one can reach the target from any state. The handling of non-holonomic constraints in this example is done indirectly by the interaction between the high level planner and our algorithm. The high level planner reacts (online) to the current state and uses our algorithm to generate trajectory segments in an order that ensures reaching the goal. This example demonstrates the use of our algorithm to generate a series of concatenated smooth trajectories online as directed by a high level motion planner.

Component	Dimensions [m]	Weight [Kg]
Body	$0.7 \times 0.5 \times 0.2$	10
Wheels	0.1 radius	0.2

Table 3: Robot parameters

The attached simulation shows the robot moving towards a randomly selected target from a given initial state. The simulation demonstrates that by generating control inputs as trajectories of high order we can improve tracking accuracy as well as produce a smoother ride. The simulation also shows that trajectories are not limited to simple XY-paths but can be incorporated in many different applications.

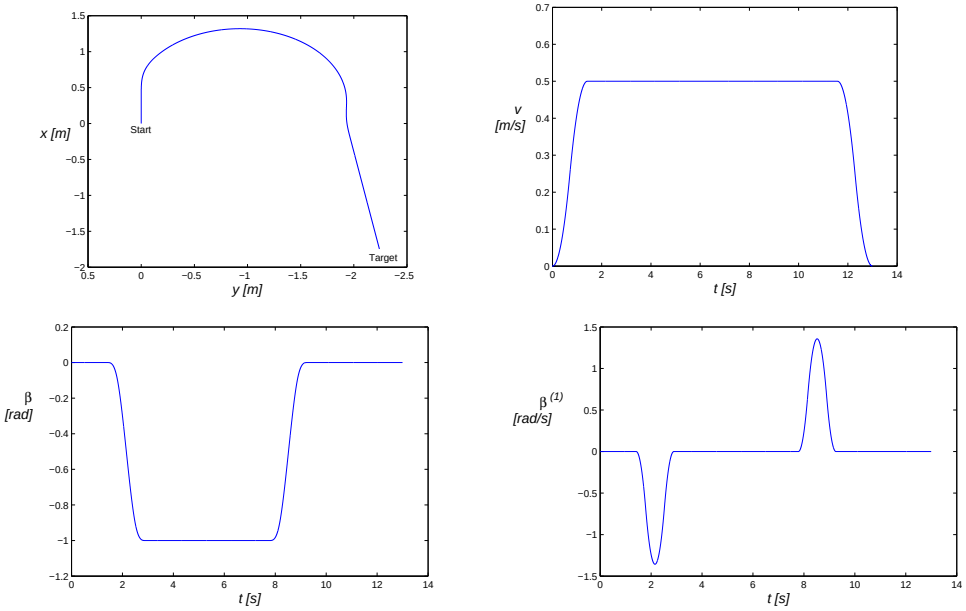

Figure 11: The trajectories for β, v and the resulting XY path from $(0, 0)$ to the target.

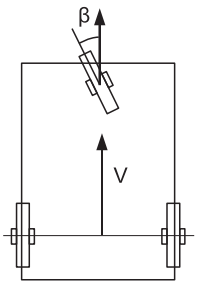

Figure 12: Top view of the simulated robot.

3.1.4 Example 4

This example demonstrates our algorithm for the 3 axis robot shown in Figure 13. The two links are both of $1.4m$ length. The underlying simulation engine used here is again the ODE in [27].

The robot is controlled by specifying the angular velocity of its three joint angles: θ_1, θ_2, θ_3. The robot's task is to move its tip from rest at the initial position (where all joint angles are 0) to rest at point C, while passing through points A, B at specified velocities. The trajectory is planned in the joint space.

The intermediate velocities were specified in the joint space as ± 0.2 [rad/s^2] at each point (the sign was dictated by the direction of motion towards the next point). Our algorithm generated three trajectories, one for each joint angle, that pass through the specified via points and speeds. The bounds used for generating the fourth order trajectories are:

- $|\theta_1^{(1)}| < 0.5$ [rad/s], $|\theta_1^{(2)}| < 1$ [rad/s^2], $|\theta_1^{(3)}| < 2.5$ [rad/s^3], $|\theta_1^{(4)}| < 5$ [rad/s^4]
- $|\theta_2^{(1)}| < 0.5$ [rad/s], $|\theta_2^{(2)}| < 1$ [rad/s^2], $|\theta_2^{(3)}| < 2.5$ [rad/s^3], $|\theta_2^{(4)}| < 5$ [rad/s^4]
- $|\theta_3^{(1)}| < 0.5$ [rad/s], $|\theta_3^{(2)}| < 1$ [rad/s^2], $|\theta_3^{(3)}| < 2.5$ [rad/s^3], $|\theta_3^{(4)}| < 5$ [rad/s^4]

Figure 15 shows the resulting trajectory in the workspace, along with the intermediate Cartesian velocities at the intermediate points. The generation time for the entire three dimensional three point trajectory was about 30 milliseconds, which is consistent with Table 2 and the discussion in Section 3.2.

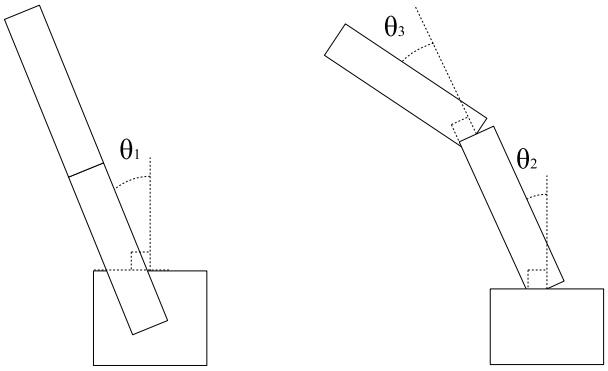

Figure 13: Top (left) and side (right) views of the robot in Example 4.

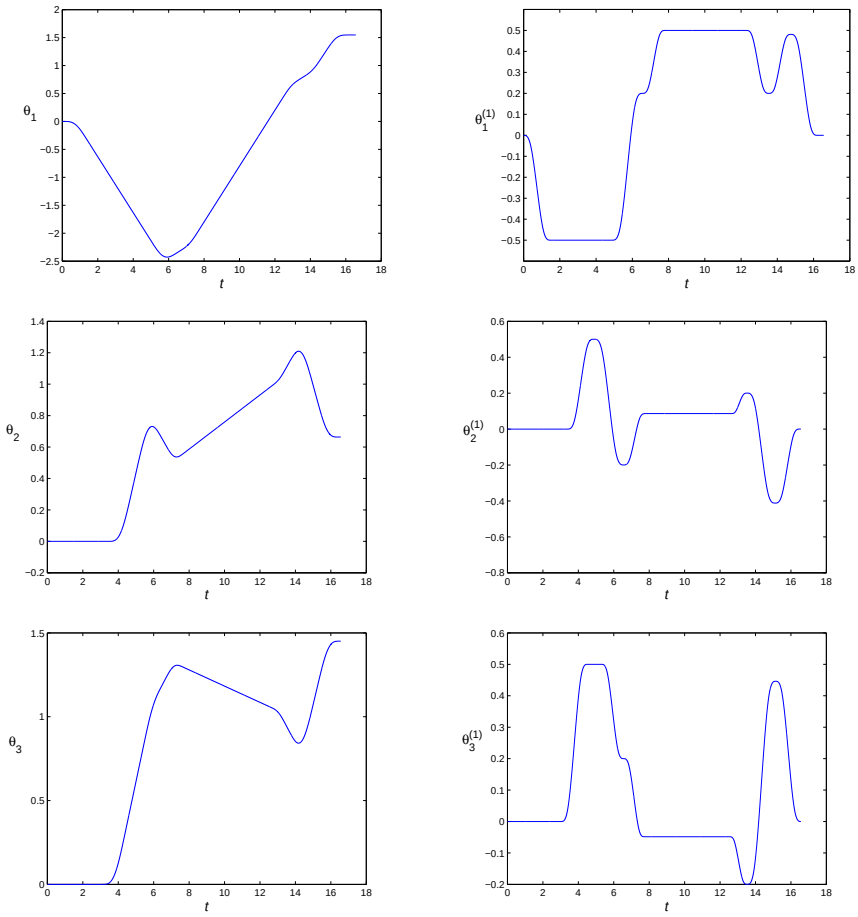

Figure 14: The joint angles and velocities for Example 4.

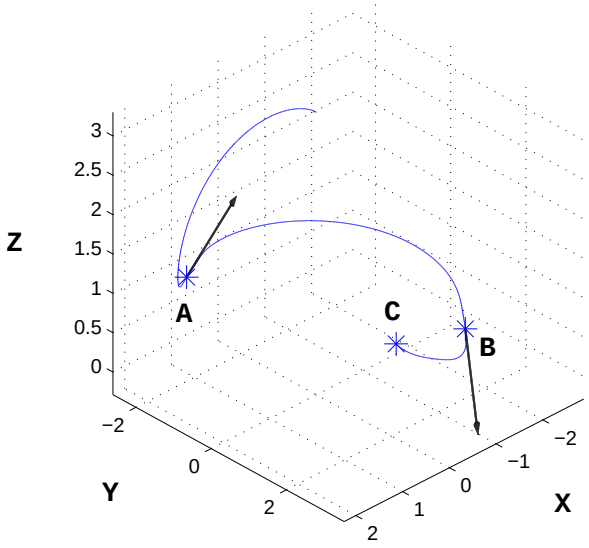

Figure 15: The path followed by the robot's tip and the Cartesian velocities at the via points in Example 4.

3.2 Multi-axis runtimes

The typical time needed to calculate a multi-axis trajectory with n axes, is the time it would take to calculate $2n-1$ single-axis trajectories. This is because n single-axis trajectory calculations are needed to find the slowest axis, and additional $n-1$ runs to synchronize the faster ones. This typical number may be exceeded however in cases when the other axes can not all be synchronized to the runtime of the slowest axis. In such cases (as discussed before) a new slowest axis will be found and all trajectories will have to be synchronized to it. The number of times this could happen for a single-axis however is limited to the number of discontinuities in the range of possible completion times for the single-axis problems. As discussed in [12], this is limited to a certain number depending on the order of the trajectory. So the worst runtime for the multi-axis case will still be polynomial with regards to the number of axes, since the process can only repeat a limited number of times for each axis with $n-1$ single-axis runs required each time.

Typical runtimes for the multi-axis case are shown in Table 4. Except for the initial and final conditions, this calculation uses the same setting as in the third order trajectory in Table 2. The initial and final conditions are set to: $x_s^{j,0}=0$, $x_f^{j,0}=100j$, $x_s^{j,1}=x_f^{j,1}=5$, $x_s^{j,2}=x_f^{j,2}=0$. These results show the typical dependency on the number of axes. The multi-axis runtimes are roughly $2n-1$ times what it would take to calculate a single-axis trajectory.

#Axes	Number of runs	Average runtime [s]
2	1000	0.000234
3	1000	0.000375
4	1000	0.000516

Table 4: Runtimes (seconds) for several multi-axis problems

4 Conclusions

This paper presented a trajectory planning algorithm for single- and multi-axis 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] J.E. Bobrow. Optimal robot path planning using the minimum time criterion. *IEEE Trans. Rob. Autom.*, 4(4):443–450, 1988.

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

[3] 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.

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

[5] D. Costantinescu and E.A. Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of Robotic Systems*, 17(5):233–249, 2000.

[6] B. Ezair, T. Tassa, and Z. Shiller. Robot application simulation. `http://www.ariel.ac.il/sites/shiller/ravlab/online_trajectory/simulation.zip`, 2013.

[7] C. Guarino Lo Bianco and O. Gerelli. Online trajectory scaling for manipulators subject to high-order kinematic and dynamic constraints. *Robotics, IEEE Transactions on*, 27(6):1144–1152, 2011.

[8] 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.

[9] I. Herrera-Aguilar and D. Sidobre. On-line trajectory planning of robot manipulators end effector in cartesian space using quaternions. *In 15th International Symposium on Measurement and Control in Robotics, Belgium, November*, 2005.

[10] I. Herrera-Aguilar and D. Sidobre. Soft motion trajectory planning and control for service manipulator robot. *Workshop on Physical Human-Robot Interaction in Anthropic Domains at IROS*, pages 13–22, 2006.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] F. Pfeiffer and R. Johanni. A concept for manipulator trajectory planning. *IEEE Trans. on Robotics and Automation*, RA-3(3):115–123, 1987.

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

[20] L.S. Pontryagin. The mathematical theory of optimal processes. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. Interscience, 1962.

[21] Z. Shiller. On singular time-optimal control along specified paths. *IEEE Transactions on Robotics and Automation*, 10(4), August 1994.

[22] Z. Shiller and H. Chang. Trajectory preshaping for high-speed articulated systems. In *ASME Journal of Dynamic Systems, Measurement and Control*, volume 117 No. 3, pages 304–310, September 1995.

[23] Z. Shiller and S. Dubowsky. Time-optimal path-planning for robotic manipulators with obstacles, actuator, gripper and payload constraints. *Intl. J. Rob. Research*, 8(6):3–18, Dec. 1989.

[24] Z. Shiller and H.H. Lu. Computation of path constrained time optimal motions with dynamic singularities. *Journal of Dynamic Systems, Measurement and Control*, 114:34–40, 1992.

[25] K.G. Shin and N.D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Trans. Aut. Ctrl.*, AC-30(6):531–541, June 1985.

[26] J. Slotine and H. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics and Automation*, 5(1):118124, 1989.

[27] R. Smith. Open dynamics engine. `http://www.ode.org`, 2007.

[28] M. Tarkiainen and Z. Shiller. Time optimal motions of manipulators with actuator dynamics. In *Proc. 1993 IEEE International Conference on Robotics and Automation*, volume 2, pages 725–730, 1993.

A On the optimality of the solution

We prove herein the optimality of the solution produced by Algorithm 1 for $m \leq 3$, in the case where the initial and final conditions are zero, $x_s^1 = x_f^1 = x_s^2 = x_f^2 = 0$.

Lemma A.1. *Consider the trajectory planning problem* $\mathbf{MP} := \mathbf{MP}(m; x_s, x_f, x_{min}, x_{max})$ *with zero initial and final conditions. Then the travel time of an optimal solution is strictly increasing with respect to the distance that needs to be traveled,* $|\Delta x| = |x_f^0 - x_s^0|$.

Proof. Without loss of generality, we assume that $x_s^0 = 0$ and that $x_f^0 > 0$. (The case $x_f^0 < 0$ can be reduced to the case $x_f^0 > 0$.) Let $\langle T, X(t)\rangle$ be an optimal solution of $\mathbf{MP} := \mathbf{MP}(m; x_s, x_f, x_{min}, x_{max})$, where $x_f = (x_f^0, 0, \ldots, 0)$, and let $\langle \hat{T}, \hat{X}(t)\rangle$ be an optimal solution of $\hat{\mathbf{MP}} := \mathbf{MP}(m; x_s, \hat{x}_f, x_{min}, x_{max})$ where $\hat{x}_f = (\hat{x}_f^0, 0, \ldots, 0)$ and $\hat{x}_f^0 > x_f^0$. Assume, towards contradiction, that $\hat{T} \leq T$. Set $k = x_f^0/\hat{x}_f^0 < 1$, and define the function $Z(t) = k\hat{X}(k^{-1/m}t)$. We claim, and prove below, that $\langle k^{1/m}\hat{T}, Z(t)\rangle$ is a solution of $\mathbf{MP}$. However, since $k^{1/m}\hat{T} < \hat{T} \leq T$ that would contradict the optimality of $\langle T, X(t)\rangle$.

To show that $\langle k^{1/m}\hat{T}, Z(t)\rangle$ is a solution of $\mathbf{MP}$, we first observe that it satisfies the required initial and final position requirements:

$$Z(0) = k\hat{X}(0) = 0\,; \quad Z(k^{1/m}\hat{T}) = k\hat{X}(k^{-1/m}k^{1/m}\hat{T}) = k\hat{X}(\hat{T}) = k\hat{x}_f^0 = x_f^0\,.$$

Its derivatives are given by:

$$Z^{(i)}(t) = k^{1-i/m} \cdot \hat{X}^{(i)}(k^{-1/m}t)\,, \quad 1 \leq i \leq m-1\,.$$

It is easy to see that all those derivatives vanish at $t = 0$ and $t = k^{1/m}\hat{T}$ since all derivatives of $\hat{X}$ vanish at $t = 0$ and at $t = \hat{T}$. It remains to show that the derivatives are properly bounded along the interval $[0, k^{1/m}\hat{T}]$. Indeed, as $k < 1$:

$$\max_{t\in[0,k^{1/m}\hat{T}]} k^{1-i/m} \cdot \hat{X}^{(i)}(k^{-1/m}t) = k^{1-i/m} \cdot \max_{t\in[0,\hat{T}]} \hat{X}^{(i)}(t) \leq k^{1-i/m} \cdot x_{max}^i < x_{max}^i\,,$$

whence Z respects the upper bounds x_{max}. Similarly, it respects also the lower bounds x_{min}. The proof is thus complete. □

Lemma A.2. *Consider the trajectory planning problem* $\mathbf{MP} := \mathbf{MP}(m \leq 2; x_s, x_f, x_{min}, x_{max})$, *where* $x_s^1 = x_f^1 = 0$, *and let* $\langle T, X(t)\rangle$ *be an optimal solution. Let* $Y(t)$ *be a function that satisfies the initial conditions* x_s, *and its derivatives are bounded by* x_{max} *and* x_{min}. *Then if* $Y(T) \leq X(T)$, *it holds that* $Y(t) \leq X(t)$ *for all* $t \in [0, T]$.

Proof. Assume, towards contradiction, that $Y(t) > X(t)$ for some $t \in [0, T]$. Then, as $Y(0) = X(0)$ and $Y(T) \leq X(T)$, the difference $Y(t) - X(t)$ must have a positive maximum in $(0, T)$, say at $t = t_0$. Define

$$Z(t) = \begin{cases} Y(t) & t \in [0, t_0]\,, \\ X(t) + Y(t_0) - X(t_0) & t \in [t_0, T]\,. \end{cases}$$

We claim, and prove below, that $\langle T, Z(t)\rangle$ is a solution to $\hat{\mathbf{MP}} := \mathbf{MP}(m; x_s, \hat{x}_f, x_{min}, x_{max})$ where $\hat{x}_f = (\hat{x}_f^0, 0)$ and $\hat{x}_f^0 > x_f^0$. However, that is impossible in view of Lemma A.1.

Clearly, Z satisfies the initial conditions x_s, since Y does. As for the final conditions, $Z(T) = \hat{x}_f^0 := X(T) + Y(t_0) - X(t_0) > X(T) = x_f^0$, and $Z'(T) = X'(T) = 0$ (recall that $m = 2$ so the initial and final conditions are only on the position and velocity). It remains to show that the derivatives of Z are properly bounded. Its derivatives are bounded by x_{min} and x_{max} on $[0, t_0]$ since Y is, and they are bounded on $[t_0, T]$ since X is. When $m = 1$, that completes the proof (since in that case it is only needed to show that the first derivative is bounded, but there is no need to show that it is continuous). When $m = 2$, we need to show that Z' is continuous at t_0. That is indeed the case since, as the function $Y(t) - X(t)$ reaches a maximum value at t_0, it holds that $Y'(t_0) = X'(t_0)$. □

Lemma A.3. *Consider the two problems* $\mathbf{MP}_i = \mathbf{MP}(m \leq 2; x_s, x_f, x_{min}, x_{max})$, $i = 1, 2$, *where* $x_{f_i} = (e_i, 0)$. *Let* $\langle T_i, X_i(t) \rangle$ *be an optimal solution to* $\mathbf{MP}_i$, $i = 1, 2$. *Then if* $e_1 \geq e_2$, *it holds that* $\int_0^{T_1} X_1(t)dt \geq \int_0^{T_2} X_2(t)dt$.

Proof. Assume that $e_1 \geq e_2$. Then, by Lemma A.1, $T_1 \geq T_2$. Let us extend the function X_2 to the interval $[0, T_1]$ as follows:

$$\hat{X}_2(t) = \begin{cases} X_2(t) & t \in [0, T_2]\,, \\ e_2 & t \in [T_2, T_1]\,. \end{cases}$$

It is easy to see that $\langle T_1, \hat{X}_2(t) \rangle$ is another solution to $\mathbf{MP}_2$. (When $m = 1$ this is simple; when $m = 2$, we have $X_2'(T_2) = 0$, whence $\hat{X}_2(t)$ has a continuous derivative along $[0, T_1]$.) By Lemma A.2 on $X_1(t)$ and $\hat{X}_2(t)$, we get that $\hat{X}_2(t) \leq X_1(t)$ for all $0 \leq t \leq T_1$. Hence,

$$\int_0^{T_1} X_1(t)dt \geq \int_0^{T_1} \hat{X}_2(t)dt \geq \int_0^{T_2} X_2(t)dt\,.$$

□

Finally, we prove that when $m \leq 3$ and the initial and final conditions are zero, Algorithm 1 produces a solution which approximates an optimal solution.

Theorem A.4. *Consider the trajectory planning problem* $\mathbf{MP}(m \leq 3; x_s, x_f, x_{min}, x_{max})$ *with zero initial and final conditions,* $x_s^1 = x_f^1 = x_s^2 = x_f^2 = 0$. *Let* $\langle T, X(t) \rangle$ *be an optimal solution for which* $v_0 := \max_{[0,T]} X'$ *is maximal (from among all optimal solutions for the problem). Then the solution* $x(t) = x_\varepsilon(t)$ *generated by Algorithm 1 converges to* $X(t)$ *when* $\varepsilon \to 0$.

Proof. The theorem is true for $m = 1$ since the solution that it returns in that case is clearly optimal. We proceed by induction. Without loss of generality, we assume that $\Delta x = x_f^0 - x_s^0 > 0$. (The case $\Delta x < 0$ can be reduced to the case $\Delta x > 0$.) In such cases, the algorithm will reject all values of $v \leq 0$ since for such values the resulting velocity will be always non-positive, whence cannot travel a positive distance Δx.

Let v_0 be the maximum of $X'(t)$ on $[0, T]$. Consider now the sequence of values of v that are tested in the binary search by Algorithm 1. For some of those values the algorithm manages to construct a valid solution (when $\Delta/v \geq 0$), while for others it fails (when $\Delta/v < 0$). If v is a value for which the algorithm managed to find a valid solution, we shall denote that solution by $\langle T_v, x_v(t) \rangle$. We will show that $v \to v_0$ and that $\langle T_v, x_v(t) \rangle \to \langle T, X(t) \rangle$. To this end, we prove the following claims:

C1: If v is a value that the algorithm accepts, then $\langle T_v, x_v(t)\rangle$ is the optimal solution from among all solutions for which the first derivative is upper bounded by v.

C2: In the binary search, every $v > v_0$ will be rejected by the algorithm.

C3: In the binary search, every $0 < v \le v_0$ will be accepted by the algorithm.

C4: When $v \to v_0$, $\langle T_v, x_v(t)\rangle \to \langle T, X(t)\rangle$.

• Proving C1: The solution $\langle T_v, x_v(t)\rangle$ constructed by the algorithm for the value v has a derivative $x'_v(t)$ with the following structure:

$$x'_v(t) = \begin{cases} v_1(t) & [0, \tau_{1,v}]\,, \\ v & [\tau_{1,v}, \tau_{2,v} + \tau_{1,v}]\,, \\ v_3(t - \tau_{2,v} - \tau_{1,v}) & [\tau_{2,v} + \tau_{1,v}, T_v]\,. \end{cases}$$

Since, by induction, the algorithm returns the optimal solution for $m-1$, then $\langle \tau_{1,v}, v_1(t)\rangle$ is an optimal solution for the reduced **MP** problem described in Step 7 of the algorithm.

Assume that $\langle T_g, g(t)\rangle$ is another solution **MP** for which $T_g \le T_v$ and $\max g' \le v$. We shall prove that in such a case g coincides with x_v. By Lemma A.2, for x'_v and g' (as X and Y respectively), $g'(t) \le x'_v(t)$ for all $t \in [0, \tau_{1,v}]$. Hence,

$$\int_{[0,\tau_{1,v}]} g'(t)dt \le \int_{[0,\tau_{1,v}]} x'_v(t)dt\,. \tag{12}$$

Applying Lemma A.2 for $x'_v(T_v - t)$ and $g'(T_g - t)$ on $[0, \tau_{3,v}]$ we infer that $g'(T_g - t) \le x'_v(T_v - t)$ for all $t \in [0, \tau_{3,v}]$, whence

$$\int_{[0,\tau_{3,v}]} g'(T_g - t)dt \le \int_{[0,\tau_{3,v}]} x'_v(T_v - t)dt\,. \tag{13}$$

Consider now the interval

$$I = [0, T_g] \setminus \Big([0, \tau_{1,v}] \bigcup [T_g - \tau_{3,v}, T_g]\Big)\,. \tag{14}$$

Since $T_g \le T_v$, that interval is contained in the interval $[\tau_{1,v}, T_v - \tau_{3,v}]$ along which $x'_v(t) = v$. Hence, as $g' \le v$,

$$\int_I g'(t)dt \le \int_I x'_v(t)dt\,. \tag{15}$$

Adding up inequalities (12)–(15), we infer that

$$\int_0^{T_g} g'(t) \le \int_{[0,\tau_{1,v}]} x'_v(t)dt + \int_{[0,\tau_{3,v}]} x'_v(T_v - t)dt + \int_I x'_v(t)dt\,. \tag{16}$$

The integral on the left of (16) is the total distance covered by g along the time interval $[0, T_g]$; therefore it equals Δx. The sum of integrals on the right of (16) equals

$$\int_0^{T_v} x'_v(t)dt + \delta = \Delta x + \delta \quad \text{where} \quad \delta := \left(\int_I x'_v(t)dt - \int_{\tau_{1,v}}^{T_v - \tau_{3,v}} x'_v(t)dt\right)\,.$$

The definition of the interval I, (14), and the assumption $T_g \leq T_v$, imply that I is contained in $[\tau_{1,v}, T_v - \tau_{3,v}]$. Since $x'_v = v > 0$ along the latter interval, it follows that $\delta \leq 0$ and $\delta = 0$ if and only if $T_g = T_v$. To summarize, the left hand side in inequality (16) equals Δx and the right hand side equals $\Delta x + \delta$, for $\delta \leq 0$. The only way for the inequality to hold is if $\delta = 0$, i.e. $T_g = T_v$. Therefore, g' and x'_v are two continuous functions on $[0, T_v]$ where $g'(t) \leq x'_v(t)$ for all $t \in [0, T_v]$ and their integrals along the interval coincide. This can occur if and only if $g' = x'_v$ for all t. Since $g(0) = x_v(0) = 0$, we infer that g and x_v coincide.

• Proving C2: Assume, towards contradiction, that the algorithm produced a valid solution $\hat{X}(t)$ with maximal velocity $v > v_0$. By C1, $\hat{X}(t)$ is at least as good as all other solutions whose maximal velocity is bounded from above by v. But $X(t)$ is such a solution. Hence, since $X(t)$ is an optimal solution, so is $\hat{X}(t)$. But that contradicts our assumption that $X(t)$ is an optimal solution that maximizes $\max X'$.

• Proving C3: In testing a value v, the algorithm finds, by recursion, the optimal solution to the problem

$$\mathbf{MP}(m-1; (0,0), (v,0), (x^1_{min}, \ldots, x^m_{min}), (x^1_{max}, \ldots, x^m_{max}))\,.$$

Let us denote that solution by $\langle \tau_{1,v}, v_{1,v} \rangle$. Similarly, $\langle \tau_{3,v}, v_{3,v} \rangle$ is the optimal solution to the problem

$$\mathbf{MP}(m-1; (v,0), (0,0), (x^1_{min}, \ldots, x^m_{min}), (x^1_{max}, \ldots, x^m_{max}))\,.$$

Let $y(t)$ denote the concatenation of v_{1,v_0} and v_{3,v_0}, i.e.,

$$y(t) = \begin{cases} v_{1,v_0}(t) & t \in [0, \tau_{1,v_0}]\,, \\ v_{3,v_0}(t - \tau_{1,v_0}) & t \in [\tau_{1,v_0}, \tau_{1,v_0} + \tau_{3,v_0}]\,. \end{cases}$$

Since τ_{1,v_0} is smaller than the time it takes X' to reach v_0 (owing to the optimality of $\langle \tau_{1,v_0}, v_{1,v_0} \rangle$), and, similarly for the decreasing part of the profile, the time interval on which y is defined is smaller than T (the interval on which X is defined). Hence, by Lemma A.1, the distance covered by $Y(t) = \int_0^t y(\tau) d\tau$ does not exceed Δx, the distance covered by X.

By Lemma A.3, as $v \leq v_0$,

$$\int_0^{\tau_{1,v}} v_{1,v}(t) dt \leq \int_0^{\tau_{1,v_0}} v_{1,v_0}(t) dt\,.$$

In similarity, the integral of $v_{3,v}$ in the third interval will be at most the integral of v_{3,v_0}. Therefore, the distance covered by $v_{1,v}$ and $v_{3,v}$ does not exceed Δx. But then $\Delta = \Delta(v) \geq 0$, whence the algorithm would accept v.

• Proving C4: By C2 and C3, the sequence v that the algorithm accepts must converge to v_0. By C3, the algorithm would accept the value v_0; the corresponding constructed solution, $\langle T_{v_0}, x_{v_0} \rangle$, is the optimal solution $X(t)$, as implied by C1. Hence, we only need to show the continuous dependence of $\langle T_v, x_v \rangle$ on v. But this is trivial, since the explicit solution for x'_v on the first and third intervals depend continuously on v) and hence, when $v \to v_0$, $T_v \to T_{v_0}$ and $x'_v(t) \to x'_{v_0}(t)$ for all t. By integration, we conclude that $x_v(t) \to x_{v_0}(t)$ for all t. □

Corollary A.5. *Under the conditions of Theorem A.4, the optimal solution* $\langle T, X(t)\rangle$ *is unique.*

Proof. Assume that there exists more than one optimal solution and let v_0 be as defined in Theorem A.4. By C3 in the proof of the theorem, Algorithm 1 would accept the value v_0 and return a solution $\langle T_{v_0}, x_{v_0}\rangle$. By C1 in the proof, any other solution to the problem whose first derivative is bounded by v_0 must coincide with $\langle T_{v_0}, x_{v_0}\rangle$. Hence, $\langle T_{v_0}, x_{v_0}\rangle$ is the unique optimal solution. □

A.1 Non-optimality in the general case

We showed in which cases the algorithm's solution is guaranteed to be optimal. Here, we examine the cases when the solution found by the algorithm may not be optimal. There are two assumptions made to make the search for the solution faster and more efficient. These assumptions, however, are not always true, and there are (somewhat rare) instances when they may mislead the algorithm into missing the optimal solution.

The first assumption may cause the optimal solution to be missed when dealing with third order (or higher) profiles with non-zero initial or final conditions. When we search for the optimal cruising velocity (the value v in the algorithm), we are looking for the velocity that will yield the lowest Δ, which is the distance covered during the constant velocity phase. By minimizing Δ, we cover more of the total distance while accelerating (or decelerating), which results in a faster profile. However, an underlying assumption that the algorithm makes implicitly is that Δ is monotonically non-increasing with respect to v, the cruising velocity. This is indeed true for zero initial and final conditions, but it is not always so otherwise. Figure 16 shows Δ as a function of the cruising velocity for a third order profile, which uses the same parameters as the third order profile in figure 2, except for $W = 15$, $x_s^1 = 250$, and $x_f^1 = 100$. These nonzero initial and final conditions give rise to two singularities in $\Delta(v)$, as the figure clearly shows. More importantly, the profile is not monotonic, as assumed by the algorithm. In particular, while a monotonic function attains the value 0 at most once, a non-monotonic function may attain it several times — five times in the function shown. Hence, while the algorithm uses a simple binary search to find the supposedly single zero of $\Delta(v)$, here, the existence of several zeros may lead the algorithm into an arbitrary one of those zeros, and not necessarily the one which yields minimal motion time.

The second assumption may affect third order profiles with non-zero initial or final conditions, or problem instances with $m \geq 4$. This time, the problem is in assuming that there is a cruising phase at all (even a degenerate one with 0 time). This assumption is shared by other works that produce higher than third order profiles, e.g. [16]. By assuming a cruising phase, we know that there is a point in the profile where all derivatives other than the velocity are 0. This greatly simplifies the search as we only have to search for an optimal cruising velocity rather than a state vector of length m. However, if the optimal trajectory always accelerates (or decelerates) then there will be no cruising phase. Also, if $m \geq 4$, the velocity profile may attain a true maximum (or minimum) velocity, and while in such points the acceleration is indeed zero (as it is the first derivative of the velocity), at least one higher derivatives will not be.

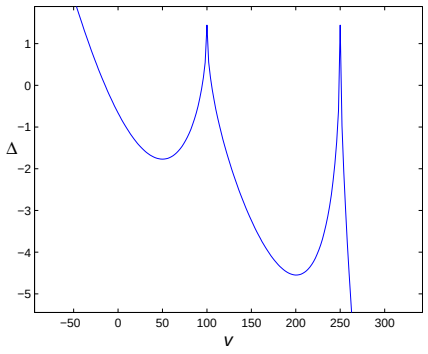

Figure 16: Δ as a function of v for an $m = 3$ profile

Hence, in each of these cases there will not be a cruising phase of any length. Figure 17 shows two $m = 4$ profiles. Solution 1 is the same profile as the $m = 4$ profile shown in Figure 2. Solution 2 is a superior solution, produced by searching for a solution with a true maximum for the velocity. Solution 1 completes the journey in time 0.4, rather than in time 0.38 as the other solution. The most striking thing shown in Figure 17 is the fact that the optimal velocity profile attains its maximum without having a cruising phase.

It should be noted that for third order trajectories these assumptions are not really needed, and a solution that always finds the optimal solution (like those described in [12] or [3]) can simply be used as the base step. However these methods will not work for higher order profiles.

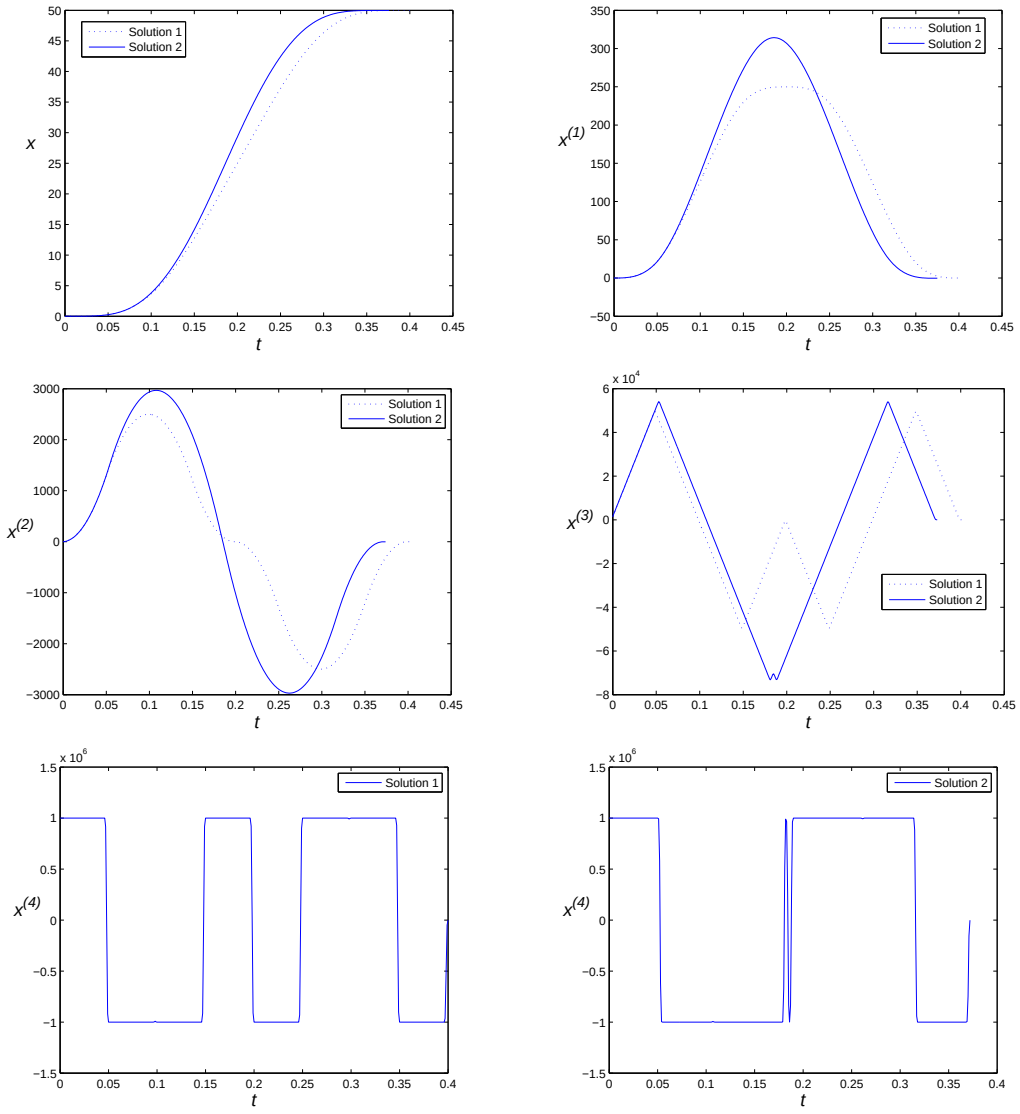

Figure 17: Two solutions for the same fourth order problem. Solution 1 is the one produced by Algorithm 1; Solution 2 is a better one.