MASTER

Time-optimal trajectory generation for n-DOF manipulators
Considering their full non-linear dynamics

Smits, C.A.

Award date:
2021

Department of Mathematics and Computer Science
Algorithms, Geometry, and Applications Research Group

Time-optimal trajectory generation for n-DOF manipulators

Considering their full non-linear dynamics

C. A. Smits

Supervisors:
Dr. K. A. Buchin
F. M. Oosterhof MSc

Committee members:
Dr. K. A. Buchin
F. M. Oosterhof MSc
Dr. I. Kostitsyna

Eindhoven, August 2021

Acknowledgements

This thesis has been developed in cooperation with the Eindhoven University of Technology and Prodrive Technologies. My thanks go out to Prodrive Technologies for allowing me to use their resources to complete this thesis. I am profoundly grateful to my colleague and supervisor Florian Oosterhof at Prodrive Technologies, who has been an excellent guide and helping me understand the complex material and concepts. Furthermore, I want to thank my supervisor Kevin Buchin from Eindhoven University of Technologiy, who has given useful feedback throughout the project, and guided me towards the completion of this thesis.

Abstract

Trajectory for n-DOF manipulators without considering the manipulator dynamics may lead to an over or under estimation of the capabilities, and potentially result in (short term) failure of the machine. Traditional techniques rely on carefully constructing the movement of the manipulator, but this is not scalable for more complex tasks in dynamic environments. In the past, several algorithms have been developed which utilise a model of the manipulator, and can handle constraints up to the second order. Two of such algorithms are implemented and their practical considerations discussed. They are then applied to several use cases to show their effectiveness. Furthermore, a novel smoothing method is described which is able to reduce high-jerk motion from a reference trajectory. This method is then also applied and shown to work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

It is impossible to imagine current production processes without some form of robotics. With the advantage of high-precision positioning, a higher productivity can be achieved in general. However, these robotic manipulators are an expensive piece of equipment. Therefore, optimal usage of these machines is crucial to result in a net gain for use cases.

There exist a wide variety of robotics manipulators. A robot is an actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks [1]. Their usage in industrial applications has increased significantly in the last ten years [2]. Their use cases extend a wide variety of industrial sectors, the largest sector being the automotive industry, followed closely by the electronics sector. Some examples of specific usages include, but are not limited to: welding, glue dispensing, order picking and spray painting operations. Typically, the tool situated at the tip of the manipulator can be changed, which makes these machines so versatile. Their high reliability, speed and precision makes them perfectly suitable for assembly line manufacturing. Since they are able to operate autonomously, they are also suitable for operating in hostile environments such as working with radioactive materials, or performing tasks in space [3].

When robots were first introduced into industrial processes, engineers "taught" the robots fixed trajectories. This could be done by specifying waypoints, or directly controlling the actuators themselves in a predefined way. This was sufficient for those applications, since they were relatively simple: the robots operated in a static environment with known geometry. Over time, the robots were required to perform more complex tasks. However, for such more complex tasks the environment cannot be assumed to be static, it is instead dynamic. An example of this is a human-robot work cell, in which a human operates alongside a manipulator. Moreover, "teaching" the robot trajectories becomes impractical for tasks that do not have fixed trajectories, for example bin-picking tasks. For those, the decisions are not fixed but depend on some external factor, such as a vision system deciding what and how to pick up an object. It was therefore required to formulate a new approach to more generic trajectory generation.

To increase productivity, not only safer manipulators are required, they must be used to their full potential. Time-optimal trajectories, trajectories that require the least amount of operation time for the manipulator, are an integral part to achieve this goal. It has been an active area of research since last decade, and it continued being so. Conventional optimal control theory is well developed for linear systems. However, it is difficult to apply these concepts to (highly) non-linear dynamic systems with complex constraints, such as robotic manipulators [4]. These constraints are necessary for a manipulator to function properly, even for an extended lifetime. There exist a wide variety of constraints: constraints which put a limit on the movement of the joints or end-effector, such as limits on velocity, acceleration, jerk, etc. Other constraints put limits on the actuator torque, or actuator current. Some constraints put limitations on the end-effector tool, such as a grasp stability constraint. Apart from these constraints, other optimisation goals than time-optimality exist, e.g. computational complexity or energy-optimal. The specific constraints depend on the application of the manipulator.

1.2 Problem formulation

The general problem of robotics planning admits a wide variety of techniques. While there exist many different goals for such algorithms, this thesis focuses on time-optimal trajectory generation. In general, the input to such algorithms is some task description which defines what the manipulator is expected to do, i.e. move from A to B. The output is then a trajectory. We make a clear distinction between a (geometric) *path* and a *trajectory*. A geometric path is a description which defines the shape of the movement of the manipulator, and does not carry any relevant information about the movement such as velocity and acceleration. A trajectory, on the other hand, is a path augmented with timing information such as velocity and acceleration.

To plan true time-optimal trajectories, an algorithm should consider both the path and the timing information, i.e. the path (timing) parameterisation, at the same time. This is often referred to as the coupled approach[1]. The shortest path does not guarantee being part of the optimal trajectory, simply because of the dynamics involved in multi-axis systems. It is often the case that a longer path is required, because it can be traversed faster, obtaining a shorter total traversal time.

While the coupled approach yields the most desired result, it is notoriously hard in practice. To this end, the decoupled approach has been developed. In the decoupled approach, the path planning is separated from the trajectory generation. This does not yield true time-optimal trajectories in general, but it reduces the computation complexity greatly. By splitting these two stages, the challenges can be solved each in their own domain. For practical considerations, these stages can be separately developed, and different implementations for specific use cases can be switched at will. In the remainder of this thesis, the decoupled approach is assumed.

It is now possible to define the stages from input to operation in more detail, and provide more context to each separate stage.

Task definition The goal of this first stage is to collect a complete description of the task, and all applicable constraints. This includes for example beginning and end positions of the manipulator, a geometric description of the environment, limits on actuator dynamics.

Geometric path planning This stage is responsible for generating a geometric path through the task space, or configuration space, of the manipulator. Challenges that occur at this stage are respecting the geometric constraints by obstacle avoidance and avoiding self-intersection. Another challenge which often occurs is singularity avoidance: points at which the velocity is often required to be very low, in order to satisfy the constraints. Hence, avoiding such points is beneficial for the total traversal time. The precise method of how a path is generated can differ greatly, and change depending on the requirements of the application. A path can be defined by a series of waypoints, predefined segments, or in terms of actuator configurations.

Trajectory generation At this stage, also commonly referred to as *time scaling*, the geometric path is converted into a trajectory. This involves adding timing information such as timestamps, velocity, acceleration, and higher derivatives if required to the path. The resulting velocity, acceleration, etc. should respect the system constraints as defined in the task definition.

Execution Finally, the resulting trajectory is executed by the controller of the manipulator such that the movement is realised. While this is not directly related to path and trajectory generation, this stage comes with its own challenges as well. For example path tracking: keeping

[1]Some authors have called this “global search”, but the majority agrees on the terminology presented in this thesis.

the manipulator on, or close to, the designed path. When a manipulator moves, it creates vibrations in the physical system. If not taken care of, these can cause resonance, reduce tracking precision, and exert undesired stress on the components. Typically, a controller is specifically created for a type of robot and tuned individually.

Although each stage comes with a set of interesting challenges, this thesis focuses on the trajectory generation stage of this general framework. It is therefore assumed from this point onward, unless otherwise stated, that a geometric path for each joint has been generated by some arbitrary path planning algorithm.

The topic of this thesis, and therefore the problem formulation, is the study of the feasibility and practical considerations of known algorithms which are developed and improved upon in the last four decades. While some algorithms are technically feasible and computationally possible, their complexity and computation time make them unsuitable in practice. The contributions of this thesis are threefold:

i An extensive literature survey is provided containing varying methods of achieving time-optimal, or near time-optimal, trajectories for various types of constraints and purposes. Moreover, other works are reviewed which consider challenges regarding the practical feasibility of techniques.

ii A practical implementation and comparison of various known techniques from previous literature to give insight into the development of such algorithms, and the practical considerations thereof.

iii A description of a novel technique for third-order constraints (i.e. joint jerk limits) for a path with specific assumptions to be elaborated on later.

1.3 Thesis outline

The outline for the rest of this thesis is as follows: Chapter 2 gives an literature overview of previously obtained results and developed techniques, which are divided into several categories. Chapter 3 is about algorithms to solve for second order constraints. Two algorithms from the literature are explained and implemented: TOPP-NI and TOPP-RA. Practical considerations are also discussed. Chapter 4 explorers a novel smoothing technique, to “smooth” a second order compliant trajectory such that it respects third order bounds, i.e. bounds on the jerk. In chapter 5, these algorithms are applied to several use cases which consist of chosen robot configurations and movements. The outputs of each algorithm are shown. Finally, chapter 6 concludes this thesis.

Chapter 2

Literature review

This chapter contains an overview of the previous literature on the topic of time optimal motion for robotic manipulators. As was explained in the introduction, there are essentially two types of algorithms which solve this problem: the coupled and the decoupled approach. Although it has been mentioned that this thesis focuses primarily on the trajectory generation step in the decoupled approach, some attention is given to the other approaches related to trajectory planning algorithms. There are many characterising features by which the trajectory generation, i.e. time optimal motion generation, can be subdivided: e.g. the manipulator model (fully actuated vs redundantly actuated manipulators) and the type of constraints. In general, three families of algorithms can be distinguished: numerical integration, dynamic programming and convex optimisation. The relevant literature for these families is discussed in the following subsections.

2.1 Numerical integration

The idea of incorporating the non-linear dynamics of a robotic manipulator was pioneered by Bobrow et al. [5, 6], Shin and McKay [7], and Pfeiffer and Johanni [8]. They were the first authors that developed algorithms which computed a time-optimal trajectory with those type of constraints. The new key insight that allowed this development is that they used a re-parameterisation function γ. This function γ takes as input a timestep, and returns the position along the path, precisely defining the trajectory velocity, acceleration, etc. The constraints are then formulated in terms of this function, allowing a solution to be found. This new space is called the *phase space*, or *phase plane*. In this new context, the re-parameterisation function is numerically integrated to obtain an optimal solution. Within the phase plane, one can distinguish admissible and inadmissible regions. The boundary between these regions is often referred to as the Maximum Velocity Curve, or MVC for short. A valid trajectory is a line that connects the beginning and end positions through the admissible region. Bobrow assumed in his paper that the inadmissible region is simply connected [6], but Shin and McKay showed that this is not the case when viscous friction forces are taken into account in the dynamic model of the manipulator [7]. The three authors developed similar algorithms based on the same technique, but their precise formulations differ. Practical implementations of these algorithms have been demonstrated to work in CAD(-like) programs [4, 9].

These algorithms rely on so called switch points, points at which the trajectory switches from maximum acceleration to minimum acceleration. Slotine et al. [10] developed a new method in which they first precompute these points. This heavily reduced the cost of searching for those points in the original approaches. However, Shiller and Lu showed that these algorithms failed at so called *singular critical points* [11, 12]. A critical point is a point on the MVC at which there does not exist a unique acceleration. This happens when the inertia term of a joint becomes zero. Singular critical points occur when a following the maximum acceleration profile pushes the trajectory into the inadmissible region. To combat this issue, Shiller and Lu developed their own algorithm which puts more computational effort into dealing with this type of critical points. Some years later, Ghasemi et al. developed a technique to compute these critical points for parallel manipulators [13]. Later, Pham revisited the research on singular critical points and developed a complete solution which resulted in a more robust algorithm [14]. This algorithm has also been presented as an open-source implementation in C++/Python[1].

Having showed that it is feasible to perform trajectory generation with non-linear dynamics, other authors started to experiment with new types of constraints. For example, Dubowsky et al. developed a set of constraints that minimised the disturbance of a manipulator mounted on a spacecraft [3]. Ma et al. developed a set of algorithms based around heat and kinematic characteristics for (non-)redundant serial manipulators [15, 16, 17, 18, 19]. Cho et al. developed an algorithm with impulsive constraints [20]. This is useful when a manipulator is carrying sensitive or dangerous objects. Zlajpah extended Bobrow's algorithm by also including other factors such as joint velocities and task requirements [21]. Constantinescu et al. developed an algorithm which handles the first derivative of the actuator torques, i.e. the torque rates [22]. In their view, this limits the jerky motion and severe vibrations in the arm, which may lead to failure of the operation, and in even worse situations the machine itself.

Another direction researchers have spend time on is approximation algorithms. Pardlo-Castellote et al. developed a proximate time-optimal path parameterisation algorithm [23]. Although the solution is not truly optimal, the worse-case runtime is linear with respect to the path length and is predictable as a function of the path length. It first computes the local minima using [10], and then integrates each segment once. Later, Matmüller et al. build upon this technique and develop a proximate time-optimal algorithm which takes jerk constraints into account [24].

Previous papers spent their attention mostly on second order dynamics, and the third order case has received less attention. Shiller and Tarkiainen noted that the optimal control of such models is usually discontinuous in the actuator torques [25]. This yields physically unrealistic trajectories due to delays caused by actuator dynamics. Even worse, the transient response of (neglected) actuator dynamics can possibly cause significant tracking errors. To obtain a more realistic solution, an optimised model is required. Shiller et al. developed an algorithm which takes the third order dynamics into account [25]. However, this significantly increases the running time due to the phase space becoming three dimensional. A year later, Shiller published another algorithm in which he modelled the actuator dynamics using a time-energy cost function [26]. More recently, Pham et al. developed TOPP3, a robust algorithm which is capable of handling third order constraints. In their original paper, they have studied the structure of third order solutions and identified two main difficulties with third order trajectory generation: how to smoothly connect optimal profiles and how to address singularities which stop profile integration prematurely. According to them, their algorithm addresses those issues [27].

A promising new method was found last decade which combines sampling-based planning and time-optimal trajectory generation to make the TOPP problem significantly more tractable. Admissible Velocity Propogation (AVP) computes all feasible trajectories by finding an interval [28, 29]. Given a path and an interval of reachable velocities at the beginning of that path, AVP efficiently computes the interval of all reachable velocities at the end of the path, while respecting the system constraints [30].

2.2 Convex optimisation

The second family of algorithms used to solve the TOPP problem is convex optimisation. With this approach, TOPP is formulated as a large convex optimisation problem, whose optimisation variables are the control (e.g. acceleration, jerk). While general mathematical optimisation is hard [31], there exist classes of convex optimisation problems that admit a polynomial-time algorithm [32]. It is therefore a simple method in execution, since off-the-shelves algorithms can be applied which are efficient and reliable.

[1]`https://github.com/quangounet/TOPP`

The difficulty with this approach, however, is that the constraints must remain, or become, convex. Verscheure et al. transformed the problem into a convex optimal control problem with a single state using various convexity preserving extensions [33]. They developed a similar algorithm for the online case [34]. While these methods guarantee the efficient computation of optimal solutions, they fail to address many practical applications such as torque rate constraints, velocity-dependent torque constraints or viscous friction effects. This is primarily due to the fact that these constraints give rise to non-convex formulations. To combat this issue, an extension on their original paper has been proposed using sequential convex programming [35]. A similar technique is used in a paper by Hauser on time-optimisation for robots under contact constraints, which can be encountered in legged locomotion [36]. Palleschi et al. applied McCormick envelopes to perform efficient and reliable convex relaxation, because of which they are able to use jerk as the control input [37].

More recently, Pham et al. developed an algorithm based on reachability analysis, a standard concept in control theory, to solve for the second-order case [38]. This technique is heavily related to Admissible Velocity Propogation (AVP). Their algorithm computes the reachable sets of control, i.e. the velocities, which is the same concept as AVP. They claim that the admissible control can be computed quickly and robustly using linear programs. This technique is explained in more detail later.

2.3 Dynamic programming

Another method that is used in the literature to solve the TOPP problem is applying dynamic programming to the phase space [39]. According to Shin and McKay, dynamic programming places few restrictions on the cost function that is to be minimised [40], which is the case for numerical integration techniques. The algorithm they developed is similar to their numerical integration technique [7], and they explain how jerk constraints can be incorporated into the algorithm. However, this requires the algorithm to search through the three dimensional phase space, instead of the two dimensional phase plane. Singh and Leu improved this technique by removing the need for the path to be parameterised by a scalar parameter, but instead it can be represented by a sequence of points [41].

2.4 Other approaches

2.4.1 Coupled approach

Coupled approaches have the advantage of generating truly time-optimal trajectories, with the negative side of high computational cost. While this is not the focus of this thesis, we shed a bit of light on the research done in this area.

Gilbert et al. created a (coupled) algorithm which applied "simple" dynamics to make the algorithm feasible, but they mostly focused on collision avoidance [42]. The joint motions are optimised whilst ensuring that at each moment in time, obstacles are avoided. An alternative approach is developed by Bobrow, in which the motion of each joint is paramterised first in space and then in time [43]. To find the time-optimal motion along a specified path, Bobrow uses his own developed algorithm a few years prior [6]. Around the same time, Shiller and Dubowsky presented their coupled approach which computes global time-optimal solution [44]. Their algorithm uses hierarchical search to reduce the set of feasible paths. Paths are pruned based on lower-bound estimators on the traversal time.

2.4.2 Practical considerations

It is important to understand the structure of the optimal solution when working on a practical implementation of these algorithms. Bobrow et al. assumed in their original paper [6] that at least one actuator is saturated at all times when considering optimal control. Chen at al. showed, however, that there is one, and only one, actuator saturated at a time [45].

Time-optimal trajectories require the maximum allowable control. Combined with the above fact, there is always one actuator operating at its maximum torque. There is therefore no margin for error to cope with disturbances or model inaccuracies [46]. Trajectories are planned without considering the dynamics due to the tracking controller, and, trajectories are planned without considering the requirements for tracking accuracy. This inevitably may result in deviations from the path, a highly undesirable artifact in precision positioning systems. Dahl and Nielsen therefore developed a path velocity controller which modifies a nominal solution, coming from a time-optimal algorithm [47]. This method has also been experimentally evaluated [48].

Chapter 3

Second order solution

In this chapter, the second-order case is discussed. Constraints using derivatives up to the second order are considered, e.g. acceleration bounds, torque constraints and direct velocity limits. First, the problem formulation is given for this scenario. Then, two implementations are discussed: TOPP-NI, an algorithm developed by Bobrow et al. using numerical integration techniques [6]; and TOPP-RA, a more recent algorithm developed by Pham et al. which employs reachability analysis and linear programming [38].

3.1 Problem formulation

3.1.1 Input

Path and trajectory

We consider an n-DOF manipulator, whose configuration is commonly denoted by $q \in \mathbb{R}^n$. Since we are dealing with paths, it is more natural to treat q as a function, defined as $q : [0, s_{\text{end}}] \to \mathbb{R}^n$ for some value s_{end}. This function is a path through the configuration space, which defines how the robotic manipulator moves. $q(s)$ for some $0 \leq s \leq s_{\text{end}}$ then simply refers to a specific position along the path. The decoupled scenario is assumed and therefore q is completely known. We furthermore assume that q is twice differentiable, and therefore q' and q'' exist, which we will refer to the path velocity and path acceleration respectively. Since the constraints are of the second order, we do not require q to be thrice differentiable and hence jerk constraints are not considered.

We make a distinction here between a path and a trajectory. The former is a geometric description of how the manipulator moves through the configuration space. The latter is a path with timesteps associated with each position. The goal is to find a trajectory which has the same geometric shape as the path, and satisfies all the constraints. A time-optimal trajectory is then a trajectory which minimises the total time that is needed to traverse the path. Let $\hat{q}$ denote the trajectory.

Constraints

Constraints are put on the trajectory $\hat{q}$ to bound the kinodynamic motion. In the second order case, these constraints may consist of up to the second derivative of $\hat{q}$. Two simple constraints are for example (direct) trajectory velocity bounds, and trajectory acceleration bounds. These simply put a hard limit on the direct movement of each joint.

$$\begin{aligned} \hat{q}'_{\min}(t) \leq \hat{q}'(t) \leq \hat{q}'_{\max}(t) \\ \hat{q}''_{\min}(t) \leq \hat{q}''(t) \leq \hat{q}''_{\max}(t) \end{aligned} \tag{3.1}$$

As explained in the introduction, when the manipulator moves, other forces start to contribute to the overall motion of the system. To model these forces, one can use the torque bounds on a fully-actuated manipulator:

$$M(\hat{q})\hat{q}'' + \hat{q}'^{\top}C(\hat{q})\hat{q}' + g(\hat{q}) = \tau \tag{3.2}$$

where M is the mass matrix, C is the Coriolis and centrifugal tensor, g is the vector of gravity forces and τ is a vector of joint torques. For each individual joint i, a limit can be defined as follows:

$$\tau_{i,\min} \leq \tau_i(t) \leq \tau_{i,\max} \tag{3.3}$$

These constraints can actually be put in a more generalised form: the generalised second order constraints:

$$A(\hat{q})\hat{q}'' + \hat{q}'^{\top}B(\hat{q})\hat{q}' + f(\hat{q}) = \tau \tag{3.4}$$

where A, B and f are continuous mappings from $\mathbb{R}^n$ to $\mathbb{R}^{m\times n}$, $\mathbb{R}^{n\times m\times n}$ and $\mathbb{R}^m$ respectively and m being the number of constraint inequalities [49]. This is a broad definition and can account for a large variety of constraint types. By appropriately setting these matrices and tensors, the direct velocity and acceleration bounds can be modeled, as well as the model for a fully-actuated manipulator in Equation 3.2. Other constraints types not considered in this thesis fit this generalised model too, e.g. torque bounds for redundantly-actuated manipulators, or contact stability under a Coulomb friction model.

Initial and final conditions

Finally, it is desirable to be able to set the initial and final velocity of the trajectory. This is particularly useful when the algorithm is used to find a trajectory for an already moving manipulator, or to bring the manipulator to a particular position with a specific velocity. As will be shown, the two algorithms discussed later handle these initial and final velocities differently.

3.1.2 Output

The objective is to construct a trajectory $\hat{q}$ which has the same geometric shape as q, and satisfies all the input constraints. This is achieved by constructing a function γ:

$$\gamma : [0, t_{\text{end}}] \to [0, s_{\text{end}}] \tag{3.5}$$

such that the trajectory $\hat{q}(t) = q(\gamma(t))$ for $0 \leq t \leq t_{\text{end}}$ minimises t_{end}. Intuitively, this function acts as a sort of "time-dilation" function, mapping from the time domain to a configuration of the manipulator along the path. This then defines the trajectory velocity and acceleration at any timestep along the trajectory.

While the path and trajectory velocity and acceleration seem to be disconnected, they can be linked by means of this function γ. It is not only useful, but it is required for the algorithms to be discussed later. The trajectory velocity can be derived by differentiating $\hat{q}$ once:

$$\hat{q}'(t) = q'(\gamma(t))\gamma'(t) \tag{3.6}$$

And differentiating $\hat{q}$ twice (using the chain and product) rule yields:

$$\hat{q}''(t) = q''(\gamma(t))\gamma'^{2}(t) + q'(\gamma(t))\gamma''(t) \tag{3.7}$$

Note that $\hat{q}'$ and $\hat{q}''$ are commonly called $\dot{q}$ and $\ddot{q}$ in the literature. The dot refers to the differentiating $\hat{q}$ to the time parameter, which yields the trajectory velocity and acceleration. We attempt to disambiguate between the separate concepts of path and trajectory here a bit further.

3.1.3 Reinterpretation in the phase space

The problem definition in its current form provides a clear intuition of what needs to be achieved. However, it does not admit a practical algorithm. Instead, the problem is usually transformed to the so called *phase space* (γ, γ'), which in the case of (at most) second order constraints can also be referred to as the *phase plane.* Instead of looking at the graph $(t, \gamma'(t))$, we look at the graph $(s, v(s))$ where $v(s) = \gamma'(s)$ for the time t such that $\gamma(t) = s$. There are two major advantages in doing so [25]:

1. The final point of the movement $(\hat{q}(t_{\text{end}}), t_{\text{end}})$ is not known beforehand in the original form. This is because t_{end} is precisely a part of the result of the algorithm. In the phase plane, however, the start and end points are (γ_0, γ'_0) and $(\gamma_{\text{end}}, \gamma'_{\text{end}})$. Here, γ_0 and γ_{end} are the beginning and end of the path parameter, and γ'_0 and γ'_{end} are the initial and final velocity which can now also be an input to the algorithm. A move from rest to rest then has $(\gamma_0, 0)$ and $(\gamma_{\text{end}}, 0)$ as beginning and end positions in the phase plane.

2. Instead of having to search through the $n + 1$ dimensional space $(\hat{q}, t_{\text{end}})$ to find a solution, one can search through the two dimensional phase plane. This greatly reduces the the curse of dimensionality and thereby the computational complexity.

To transform the orignal definition into the phase plane, it is required to transform the constraints into constraints on γ' and γ''. From this point on we shall refer to γ' as the pseudo velocity and γ'' as the pseudo acceleration. It is called “pseudo” velocity/acceleration because it relates the path velocity/acceleration with the trajectory velocity/acceleration in Equation 3.6 and 3.7 respectively, but it is not any joint or end-effector velocity/acceleration directly.

Transforming the direct velocity and acceleration constraints are the easiest to derive using Equation 3.6 and 3.7 and their limits $\hat{q}'_{\min}, \hat{q}'_{\max}, \hat{q}''_{\min}$ and $\hat{q}''_{\max}$. For each joint $0 \leq i \leq n$, compute the minimum pseudo velocity and pseudo acceleration as follows

$$\gamma'_{i,\min} = \frac{\hat{q}'_{i,\min}}{q'_i} \qquad \gamma''_{i,\min} = \frac{\hat{q}''_{i,\min} - q''_i \gamma'^2}{q'_i} \tag{3.8}$$

The actual minimum pseudo velocity and acceleration is then the maximum over these values. This yields

$$\gamma'_{\min} = \max_i \gamma'_{i,\min} \qquad \gamma''_{\min} = \max_i \gamma''_{i,\min} \tag{3.9}$$

Similarly for the maximum pseudo velocity and acceleration. Putting this all together, we obtain the following constraints for the pseudo velocity:

$$\max_i \left(\frac{\hat{q}'_{i,\min}}{q'_i} \right) \leq \gamma' \leq \min_i \left(\frac{\hat{q}'_{i,\max}}{q'_i} \right) \tag{3.10}$$

And similarly for the pseudo acceleration:

$$\max_i \left(\frac{\hat{q}''_{i,\min} - q''_i \gamma'^2}{q'_i} \right) \leq \gamma'' \leq \min_i \left(\frac{\hat{q}''_{i,\max} - q''_i \gamma'^2}{q'_i} \right) \tag{3.11}$$

Note that the limits on the pseudo velocity only depend on the current position on the path, while the limits on the pseudo acceleration depend both on the position and the current pseudo velocity γ'.

Transforming the generalised second order constraints in Equation 3.4 can be done by substituting Equations 3.6 and 3.7 into it. This yields a bound on the pseudo acceleration. The derivation is as follows:

$$\begin{aligned} A(\hat{q})(q''\gamma'^2 + q'\gamma'') + (q'\gamma')^\top B(\hat{q}) q'\gamma' + f(\hat{q}) &= \tau \\ Aq''\gamma'^2 + Aq'\gamma'' + q'^\top B(\hat{q}) q'\gamma'^2 + f(\hat{q}) &= \tau \\ \gamma'' Aq' + \gamma'^2 (Aq'' + q'^\top B(\hat{q}) q') + f(\hat{q}) &= \tau \end{aligned} \tag{3.12}$$

At this point, the equation is of the following form for some t such that $s = \gamma(t)$:

$$a(s)\gamma'' + b(s)\gamma'^2 + c(s) = \tau \tag{3.13}$$

Where

$$\begin{aligned} a(s) &= Aq' \\ b(s) &= Aq'' + q'^{\top} B(\hat{q}) q' \\ c(s) &= f(\hat{q}) \end{aligned} \tag{3.14}$$

This equation relates the desired quantities with each other. Note that the quantities $a(s), b(s)$ and $c(s)$ do not depend on the function γ. Plugging in the torque bounds from Equation 3.3 yields:

$$\tau_{\min} \leq a(s)\gamma'' + b(s)\gamma'^2 + c(s) \leq \tau_{\max} \tag{3.15}$$

And from this point it is now possible to define bounds on the pseudo acceleration for each joint $0 \leq i \leq n$ separately in terms of s and γ', the position along the path and the pseudo velocity. Let $\alpha_i(s, \gamma')$ be the minimum allowed acceleration that satisfies the above constraint, and $\beta_i(s, \gamma')$ be the maximum allowed acceleration, then for a valid trajectory $\alpha_i(s, \gamma') \leq \gamma_i'' \leq \beta_i(s, \gamma')$ must hold for any position along the path. The definitions for these bounds are:

$$\begin{aligned} \alpha_i(s, \gamma') &= \begin{cases} \tau_{i,\min} - b_i\gamma'^2 - c_i/a_i & \text{if} \quad a_i > 0 \\ \tau_{i,\max} - b_i\gamma'^2 - c_i/a_i & \text{if} \quad a_i < 0 \end{cases} \\ \beta_i(s, \gamma') &= \begin{cases} \tau_{i,\max} - b_i\gamma'^2 - c_i/a_i & \text{if} \quad a_i > 0 \\ \tau_{i,\min} - b_i\gamma'^2 - c_i/a_i & \text{if} \quad a_i < 0 \end{cases} \end{aligned} \tag{3.16}$$

Which are derived by manipulation of Equation 3.15. The final bounds are obtained by taking the maximum and minimum on α and β as follows:

$$\begin{aligned} \alpha(s, \gamma') &= \max_i \alpha_i(s, \gamma') \\ \beta(s, \gamma') &= \min_i \beta_i(s, \gamma') \end{aligned} \tag{3.17}$$

These constraints put limits on the control, i.e. the pseudo acceleration, and thereby also on the admissible space in the phase plane which are characterised by so called “limiting curves”. These curves form a boundary above which no admissible control exists. Any trajectory through this part of the phase plane will result in motion that violates one or more constraints. From the already discussed constraints, we can directly derive some of those limiting curves, some of which are also extensively used in previous literature. One of these widely defined curves is the Maximum Velocity Curve (MVC), which is derived from the acceleration vector fields. When the minimum allowed acceleration α exceeds the maximum allowed acceleration β, i.e. $\alpha > \beta$, then there does not exist admissible control γ'' which keeps the manipulator on the path. It is those regions that form the inadmissible part of the phase plane. The MVC is then precisely the curve for which $\alpha = \beta$ holds.

Figure 3.1 shows an example of how such a phase plane with two solutions might look like. The shaded area is the inadmissible region, which has the MVC as boundary. The two solutions shown, i.e. T_1 and T_2, connect the initial and final positions γ_0 and γ_{end}. Note that T_1 reaches higher values for γ', which means that the velocity for this solution is higher than that of T_2. However, the solution for T_1 might still not be time-optimal, since it could have reached a higher velocity while staying below the MVC. This depends on the particular shape of the vector fields α and β of course. The following section explains how time-optimal solutions can be found.

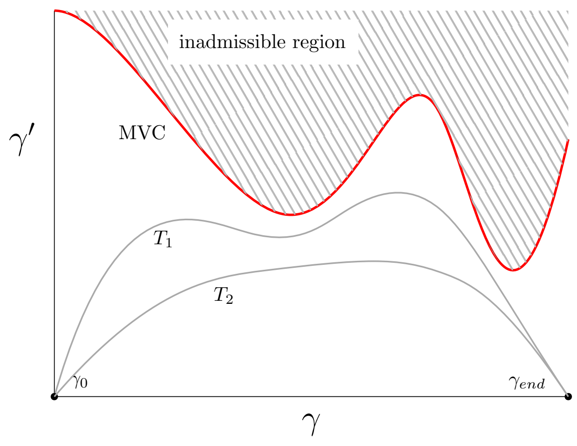

Figure 3.1: Example phase plane showing two solutions T_1 and T_2, and MVC (red) with the inadmissible region.

3.2 Implementations

This section contains a practical explanation of several techniques, how they can be used in a real-world scenario. Two variants of the algorithms previously mentioned are discussed: one based on numerical integration and on based on linear programming.

3.2.1 TOPP-NI

The idea of using numerical integration to solve the TOPP problem was first introduced by Bobrow et al. [6]. The basic concept of numerical integration is to integrate along a maximum acceleration and a minimum acceleration vector fields and carefully choosing the points when to switch between one of the other. The goal is to find the largest possible acceleration profile, which keeps the velocity profile within the maximum allowed velocity at each point on the path. The key insight is that the control is a "bang-bang" type, which means that either the maximum or minimum permissible acceleration is used. Any other control would yield a sub-optimal solution [50, 51].

In the previous section, the constraints in the phase plane were given. More specifically, α and β define two vector fields inside the phase plane that this algorithm utilises to construct the maximum acceleration and minimum acceleration curves. As long as $\beta \geq \alpha$, there exists a feasible acceleration profile that keeps the manipulator on the path. The "bang-bang" control type requires that no motion can deviate from those vector fields. The solution in the phase plane then consists of a combination of maximum and minimum acceleration curves. The main goal of the algorithm is to find the so called "switch points", the points at which the profile switches from maximum to minimum acceleration.

An example of a basic trajectory in the phase plane is shown in Figure 3.2. On the horizontal axis, we have γ (the position) and on the vertical axis we have γ', the pseudo velocity. The blue segment represents the curve following maximum acceleration, and the yellow minimum acceleration. At the top, there is a switch point.

Computation of trajectories

Before the strategy to find these switch points is discussed, an explanation to construct the maximum and minimum acceleration curves is given. The construction of the curves is done by numerical integration of Newton's equations of motion using the Euler method [52]. These equations require a time step, which will be fixed to be Δt. Now, consider a point (s_i, γ_i'), suppose we wish to integrate using the maximum acceleration vector field, or profile. The next point can then

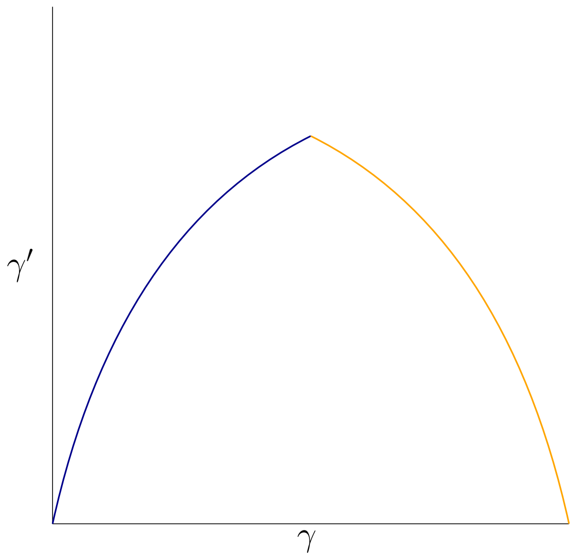

Figure 3.2: A simple trajectory consisting of maximum acceleration (blue) followed by minimum acceleration (yellow).

be computed by:

$$\begin{aligned} s_{i+1} &= s_i + \Delta t\gamma' + \frac{1}{2}\Delta t^2\alpha(s_i, \gamma'_i) \\ \gamma'_{i+1} &= \gamma' + \Delta t\alpha(s_i, \gamma'_i) \end{aligned} \tag{3.18}$$

Similarly, following the minimum acceleration profile can be computed by swapping α by β in the above equations. This is the forward integration case. We also require a method to integrate backward, i.e. starting from a point (s_i, γ'_i) and computing (s_{i-1}, γ'_{i-1}). This is done in a similar fashion:

$$\begin{aligned} s_{i-1} &= s_i - \Delta t\gamma' + \frac{1}{2}\Delta t^2\alpha(s_i, \gamma'_i) \\ \gamma'_{i-1} &= \gamma' - \Delta t\alpha(s_i, \gamma'_i) \end{aligned} \tag{3.19}$$

Dealing with switch points

To find the switch points, first integrate forwards following the maximum acceleration profile from the starting position (s_0, γ'_0) until either: s_i passes the final position s_{end} or the MVC is hit. Similarly, integrate backwards following the minimum acceleration profile from the final position $(s_{\text{end}}, \gamma'_{\text{end}})$ until $s_i < 0$, or the MVC is hit. In the most simple scenario, both curves do not cross the MVC and must at some point have crossed each other. This point is then a switch point. The solution then consists of: following the maximum acceleration profile until the switch point, then switching to the minimum acceleration profile until the end. This is graphically shown in Figure 3.3 case A.

There are six possible cases for a forward and backward trajectory to interact, which are shown in Figure 3.3. The simplest case A has already been discussed. In case C and D, one trajectory does hit the MVC (and enter the inadmissible area), but the switch point (i.e. the intersection between the forward and backward trajectory) lies before that. In these cases, the solution presented in the previous paragraph still holds. In cases E and F, the two trajectories do not hit the MVC, but the integration does not connect the initial and final end velocities properly. In E, the system constraints do not allow the velocity to increase to the required end velocity, and in case F the initial velocity is too high and the constraints do not allow for a suitable deceleration.

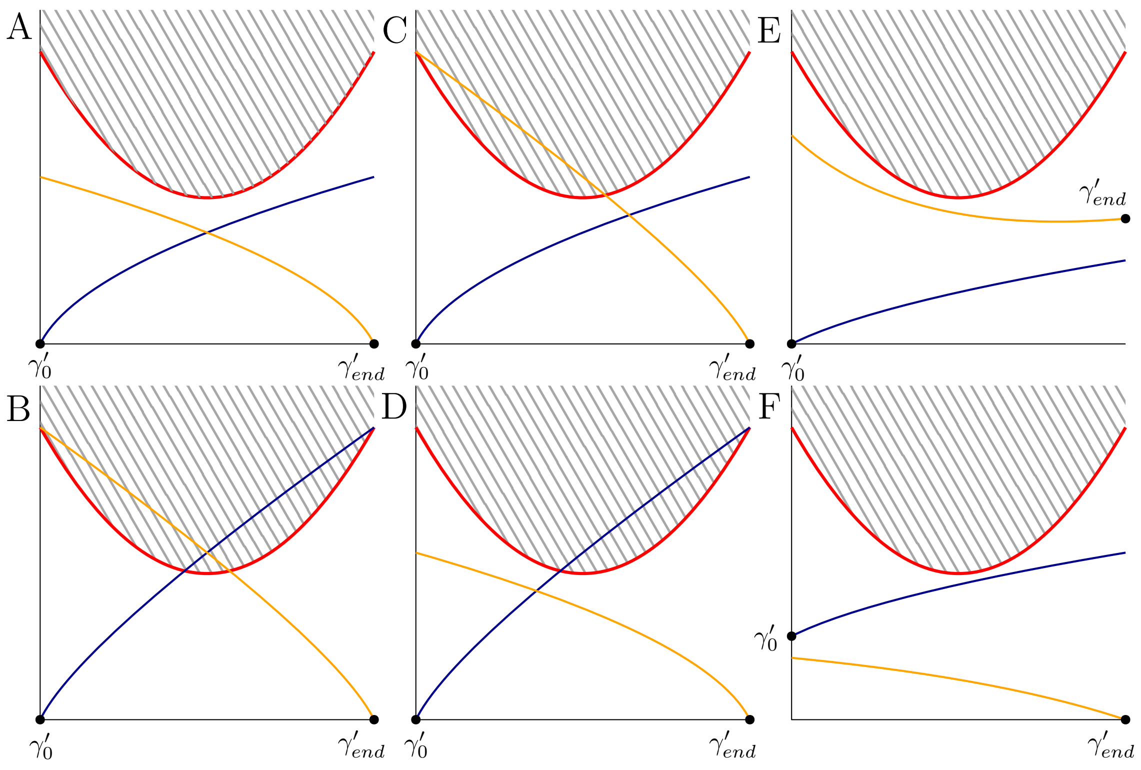

Figure 3.3: Six possible cases for forward and backward trajectories to “interact”.

In these cases, there does not exist a solution to the problem, and either the initial velocity, final velocity or constraints must be changed.

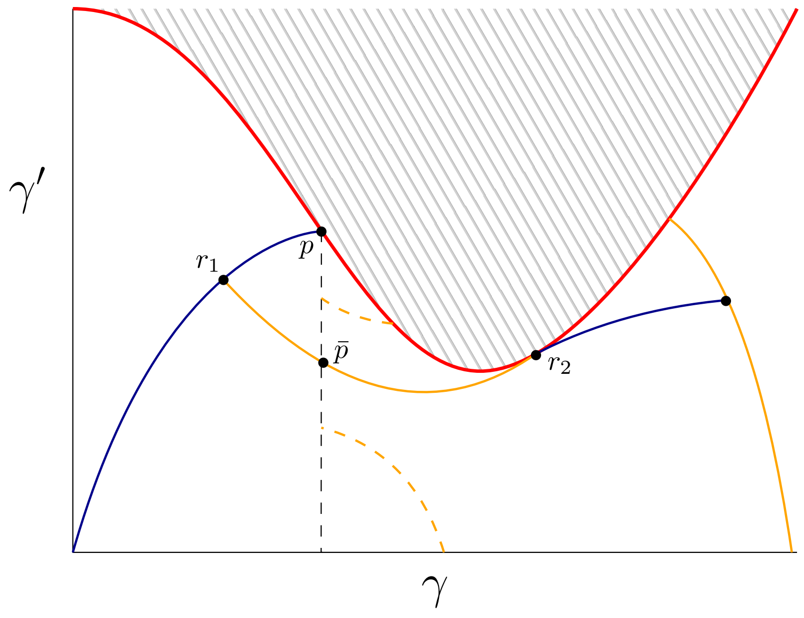

Figure 3.4: Schematic representation of how switch points are handled. The dotted yellow lines represent two trajectories starting at the same γ position as $\bar{p}$, but either hitting the MVC or reaching $\gamma' = 0$ before the end point.

In case B, both trajectories first hit the MVC and then intersect. A schematic overview of how to handle this case is given in Figure 3.4. To handle these switch points, the following steps are taken:

1. **Detection:** As explained earlier, we must deal with a switch point when the forward curve has hit the MVC, i.e. $\alpha > \beta$, and both the forward trajectory and backward trajectory have not yet crossed each other. Let $p = (\gamma_p, \gamma'_p)$ be the point at which the forward trajectory hits the MVC, as shown in Figure 3.4.

2. **Drop:** A point below p, called $\bar{p}$, is selected such that when integrating forward from $\bar{p}$ following the minimum acceleration profile, the trajectory hits the MVC tangentially. Let this point be r_2, which is a switch point. The two dashed yellow trajectories shown in Figure 3.4 represent: one trajectory starting too high and hitting the MVC; one trajectory starting too low and reaching $\gamma' = 0$ before the final position γ_{end}. Note that the switch point need not to be at a local minimum of the MVC.

3. **Connect:** To connect the initial trajectory stopping at p, and the new trajectory starting in $\bar{p}$, the algorithm integrates backward following the minimum acceleration profile starting in $\bar{p}$ until the initial trajectory is crossed. Let this intersection point be called r_1, which is also a switch point.

4. **Resume:** The algorithm can then continue on the sub-problem given by r_2 and γ_{end}. It is guaranteed that the algorithm cannot get stuck behind the MVC at the switch point r_2 when following the maximum acceleration profile [6].

This completes the scheme on how to handle switch points. The precise methods of how to find $\bar{p}$ and the intersection between trajectories is discussed in more detail later.

Algorithm

A summary of the algorithm is given in Algorithm 1. It starts by computing the initial forward and backward trajectories. When those do not intersect, the while-loop body is executed, which resolves the switch points. A point $\bar{p}$ is found below p. The trajectory starting at $\bar{p}$ and following the minimum acceleration profile stops at a point r_2 where the trajectory hits the MVC tangentially. The trajectory piece between $\bar{p}$ and r_2 is called C_2. The trajectory piece connecting C_2 to the initial trajectory is given by C_1 and the new forward trajectory is computed as C_3. The algorithm then repeats until all switch points are resolved. Note that appending a segment to the solution requires a bit more work than just to copy arrays. The final solution must have an monotonically increasing position. In some cases, the intersection between the already existing solution and the to be appended segment must be computed to correctly extend the solution.

Finding point $\bar{p}$

To find the point $\bar{p}$ is a non-trivial task. While the γ coordinate is fixed, the pseudo velocity coordinate lies in the interval between p and the γ-axis. Assume that we can sample the phase plane with infinite precision, and hence also integrate with infinite precision. A trajectory starting above $\bar{p}$ must either: also hit the MVC tangentially, or cross the MVC. Similarly, a trajectory starting below $\bar{p}$ must either: also hit the MVC tangentially, or never cross the MVC. Suppose the last property does not hold, then it means that no trajectory starting from $\bar{p}$ can decelerate down to a stand-still. From this analysis, it is possible to find the point $\bar{p}$ using binary search.

However, in any practical application it is not possible to sample the phase plane with infinite precision. It is therefore not possible to proof rigorously that binary search will always result in a solution. Suppose the following hypothetical example: we consider two candidates for $\bar{p}$ namely $\bar{p_1}$ and $\bar{p_2}$, where $\bar{p_1}$ lies above $\bar{p_2}$. Integrating forward from $\bar{p_1}$ only considers some set $\mathcal{X}_1$ of the phase plane, while the integration forward from $\bar{p_2}$ considers a set $\mathcal{X}_2$ in the phase plane. Let $\mathcal{X}_1 \cap \mathcal{X}_2 = \varnothing$. It is now possible that the trajectory starting at $\bar{p_1}$ does not cross the MVC, does

Algorithm 1: TOPP-NI

```
Step 1: compute initial forward and backward trajectories
$F \leftarrow$ integrateForward($\beta$, $(0, \gamma'_0)$);
$B \leftarrow$ integrateBackward($\alpha$, $(\gamma_{\mathtt{end}}, \gamma'_{\mathtt{end}})$);
$\mathcal{S} \leftarrow \{F\}$;
Step 2: while the forward and backward trajectories do not intersect, resolve switch points
while $F$ and $B$ do not intersect do
    $p \leftarrow$ last point of $F$;
    $\bar{p}, r_2, C_2 \leftarrow$ resolveSwitchPoint($p$);
    $C_1 \leftarrow$ integrateBackward($\alpha$, $\hat{p}$);
    $C_3 \leftarrow$ integrateForward($\beta$, $r_2$);
    Step 3: append the switch point trajectory pieces to the solution
    Append $C_1, C_2$ and $C_3$ to $\mathcal{S}$;
    $F \leftarrow C_3$;
end
Step 4: append final trajectory piece to the solution
Append $B$ to $\mathcal{S}$;
```

not hit it tangentially and simply drops to zero velocity, while the trajectory starting at $\bar{p_2}$ does cross the MVC. A schematic representation of this argument is shown in Figure 3.5.

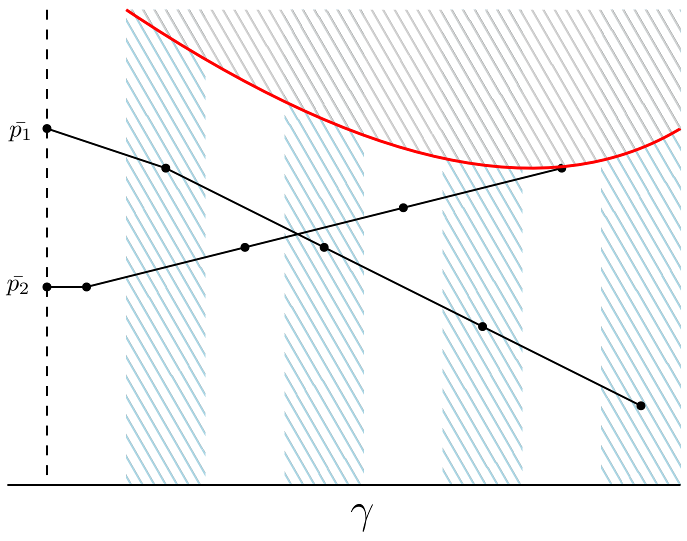

Figure 3.5: Schematic representation of the analysis why binary search may fail to find a suitable trajectory. The blue shaded area is $\mathcal{X}_1$ and the white shaded area is $\mathcal{X}_2$.

While this is a theoretical issue, if a sufficiently small integration timestep is chosen, the occurrence of this happening is minimised. Nonetheless, attention must be given to check for such a scenario, e.g. limiting the binary search algorithm by a constant number of passes.

To practically compute whether a curve hits the MVC tangentially is a non-trivial task as well. A curve could get arbitrarily close to the MVC, but the algorithm needs to decide at some point that the curve is tangent. There are essentially two methods to determine whether a point on the curve is tangent to the MVC: either compute it directly or find it by trial-and-error. The former has been investigated by Slotine et al. [10], the latter is employed in the original algorithm by Bobrow et al. [6], which is also the method explained here. The algorithm has three requirements to decide that a point r_2 hits the MVC tangentially:

1. r_2 is ε-close to the MVC, that is, $\beta - \alpha \leq \varepsilon$ in r_2.
2. The minimum acceleration vector at r_2 is roughly in the same direction as the MVC in the vicinity of r_2.
3. Integrating the trajectory forward following the minimum acceleration profile crosses $\gamma' = 0$ line at some point to the right of r_2. Or, while integrating forward, the trajectory crosses the backward trajectory, to which we needed to connect.

The first two conditions require the point to be close to the MVC, and to follow a trajectory which lies tangent to the MVC. The third condition follows from the binary search argument.

Polyline intersection

A subroutine to compute the intersections between polylines is used to find the intersection between two trajectories. This subroutine is hidden in some of the steps of the algorithm, however, a couple of notes can be made on this subroutine. An algorithm exists which computes all the intersections between a set of n line segments in $O(n \log n + I \log n)$ time, where I is the number of intersections [53]. However, an intersection is always computed between two trajectories (no self-intersections) and all trajectories are monotonically increasing polylines in γ. It is therefore possible to compute the intersections in linear time.

Runtime analysis

The analysis of this algorithm is not straightforward: the iteration cycles of the while loop is linear with the number of switch points, and the runtime of the while loop body depends on two integration steps and the switch point resolution subroutine. The latter is based on binary search, but this is not performed on a discrete domain. To limit the computation time needed, the binary search can be limited to a bounded number of iterations, after which the most suitable candidate trajectory is returned (or failure in case no candidate was found). Another factor which influences the runtime is the choice of Δt. Increasing this value results in a quicker runtime, but at the cost of decreasing the resolution, and hence quality, of the solution.

3.2.2 TOPP-RA

A different method to approach TOPP problem is based on Reachability Analysis [49], a standard concept from control theory. The key insight with this technique is based on the following: Assume an interval I_s of (squared) velocities at some point s along the path. Why the velocities are squared is explained in further detail in the next section. The *reachable set* is another interval $I_{s+\Delta}$ of (squared) velocities that is reachable following admissible control from I_s. Similarly, the controllable set $I_{s-\Delta}$ is the set of (squared) velocities such that there exists admissible control leading to a velocity in I_s. These sets can be computed quickly and robustly by solving a few small Linear Programs (LP). To obtain a solution, the phase plane is discretised into N segments, at which the controllable sets are computed recursively. This technique can therefore be put in the characteristic group of Convex Optimisation algorithms. One of the biggest advantages of TOPP-RA over TOPP-NI is that the initial and final velocity are allowed to be intervals, instead of a single value.

In short, the algorithm performs two passes: a backward pass and a forward pass. Given the desired end velocity, the backward pass computes the controllable sets, i.e. it computes at each discretised point in the phase plane the admissible control at the point. If any set is empty, then there does not exist a sequence of admissible control which steers the system to the end velocity. In this case, the algorithm reports failure. Otherwise, the algorithm proceeds to the forward pass. In the forward pass, the algorithm chooses the control greedily, i.e. it chooses the highest possible control which steers the system from one controllable set to the next, until the final position has been reached.

The algorithm is now explained in more detail, starting with a definition of the control and how a trajectory is formed in the phase plane. Then an explanation is given of how to compute this control. Finally, a final pass is required to generate the timing information from the solution, to obtain the desired output.

Definition of state and control

The phase plane is discretised into N segments: $0 = s_0 < s_1 < \ldots < s_N = s_{\text{end}}$, where s_i is referred to as the i-stage. The acceleration over the interval $[s_i, s_{i+1}]$ is denoted by u_i, which is assumed to be constant, and the squared velocity is denoted by $x_i = \gamma_i'^2$. u_i is referred to as the control, whereas x_i is referred to as the state: the acceleration makes changes to the velocity between each interval. We denote the difference in path position between to subsequent i-stages by $\Delta_i = s_{i+1} - s_i$. We have the following relation:

$$x_{i+1} = x_i + 2\Delta_i u_i \quad \text{for } 0 \le i \le N-1 \tag{3.20}$$

Proof. $\gamma(t)$ is a monotonically increasing function in t, thus there is a unique value $s \in [0, s_{\text{end}}]$ for every value of t. Therefore, the inverse $\gamma^{-1}(\text{s})$ exists. We define the pseudo velocity for a point s in terms of γ by $V(s) = \gamma'(\gamma^{-1}(s))$. Similarly, the pseudo acceleration is defined as $A(s) = \gamma''(\gamma^{-1}(s))$. For the inverse, we have:

$$\begin{aligned}(\gamma^{-1})'(s) &= \frac{1}{\gamma'(\gamma^{-1}(s))} \\ &= \frac{1}{V(s)}\end{aligned} \tag{3.21}$$

Because of the inverse function theorem, and the definition of $V(s)$. Let us now differentiate $V(s)$, yielding:

$$V'(s) = \gamma''(\gamma^{-1}(s)) \cdot (\gamma^{-1})'(s) \tag{3.22}$$

We will now define a function $X(s) = V^2(s)$, differentiate and simplify:

$$\begin{aligned}X'(s) &= 2V(s) \cdot V'(s) \\ &= 2V(s) \cdot \gamma''(\gamma^{-1}(s)) \cdot (\gamma^{-1})'(s) \\ &= 2V(s) \cdot A(s) \cdot \frac{1}{V(s)} \\ &= 2A(s)\end{aligned} \tag{3.23}$$

The second derivative of $X(s)$ is given by $X''(s) = 2A'(s)$. In fact, this is zero since we have made the assumption that the acceleration is constant over an interval. The second, and consecutive derivatives are hence zero. Let us now consider the Taylor expansion of $X(s)$ evaluated at $s_{i+1} = s_i + \Delta_i$ around the point s_i. This is given by:

$$X(s_i) + \frac{X'(s_i)}{1!}(s_i + \Delta_i - s_i) + \frac{X''(s_i)}{2!}(s_i + \Delta_i - s_i)^2 + \ldots \tag{3.24}$$

Since the second and consecutive derivatives are all zero, this infinite series reduces to:

$$X(s_i) + 2\Delta_i A(s_i) \tag{3.25}$$

Which is precisely the desired result. □

Computation of control

The algorithm only considers admissible state-control pairs. These are pairs (x_i, u_i) for some stage i for which the input constraints hold. The definition of the set containing all i-stage admissible state-control pairs is given by:

$$\Omega_i = \{(u, x) \mid \tau_{\text{min,i}} \le a_i u + b_i x + c_i \le \tau_{\text{max, i}}\} \tag{3.26}$$

Where a_i, b_i and c_i are the i-stage quantities of Equation 3.14. The i-stage set of admissible control is the projection of Ω_i on the control axis, given by:

$$U_i(x) = \{u \mid (u, x) \in \Omega_i\} \tag{3.27}$$

And similarly, the admissible states is a projection of Ω_i on the state axis

$$X_i = \{x \mid (u, x) \in \Omega_i\} \tag{3.28}$$

Note that the definition of X in the previous section is completely separate from here.

TOPP-RA defines two concepts: the i-stage reachable set and the i-stage controllable set. The reachable set is a set of states that, given some beginning interval of states, a sequence of admissible control can steer the system towards. More formally, for the zeroth-stage interval I_0, the i-stage reachable set, denoted by $\mathcal{L}_i(I_0)$, is the set of states $x \in X_i$ such that there exists a sequence of admissible control that steers the system from a state $x_0 \in I_0$ to x. In the case the sequence only consists of one step, then this is referred to as the reach set $\mathcal{R}_i(I)$ of some interval I. This set is specifically about moving from stage i to the stage $i+1$, i.e. only one step.

The dual notion of the i-stage reachable set is the i-stage controllable set. The controllable set is a set of admissible states that steer the system towards some final interval of velocities. More formally, for an interval I_N, the controllable set at stage i is the set of states $x \in X_i$ such that there exists sequence of admissible control that steers the system from x to a state $x_N \in I_N$. The i-stage reachable set is denoted by $\mathcal{K}_i(I_N)$. This set also has a one step version: given a set of states I, the one step set is a set of states $x \in X_i$ and an admissible control $u \in U_i(x)$ such that there exists a state $\bar{x} \in I$ for which $\bar{x} = x + 2\Delta_i u$ holds. The one step set is denoted by $\mathcal{Q}_i(I)$.

Since the admissible set Ω_i is convex, the reachable and controllable sets are intervals (and similarly also the one step versions). They can thus be completely defined by their upper and lower bounds. Pham et al. has formulated these computations as small linear programs (LPs). The following two LPs compute the lower and upper bound for the one step reach set of an interval I set respectively:

$$\begin{aligned} \text{minimise} \quad & \bar{x} + 2\Delta_i u \\ \text{subject to} \quad & (\bar{x}, u) \in \mathbb{R}^2 \\ & \tau_{\min} \le a_i u + b_i \bar{x} + c_i \le \tau_{\max} \\ & \bar{x} \in I \end{aligned}$$

$$\begin{aligned} \text{maximise} \quad & \bar{x} + 2\Delta_i u \\ \text{subject to} \quad & (\bar{x}, u) \in \mathbb{R}^2 \\ & \tau_{\min} \le a_i u + b_i \bar{x} + c_i \le \tau_{\max} \\ & \bar{x} \in I \end{aligned}$$

The LPs to compute the lower and upper bound for the one step controllable set $\mathcal{Q}_i(I)$ are again very small and look very similar to the previous two LPs. The following LP computes the lower bound on the controllable set of an interval I:

$$\begin{aligned} \text{minimise} \quad & x \\ \text{subject to} \quad & (x, u) \in \mathbb{R}^2 \\ & \tau_{\min} \le a_i u + b_i x + c_i \le \tau_{\max} \\ & x + 2\Delta_i u \in I \end{aligned}$$

And similarly the following LP computes the upper bound on the controllable set of an interval I:

$$\begin{aligned} \text{maximise} \quad & x \\ \text{subject to} \quad & (x,u) \in \mathbb{R}^2 \\ & \tau_{\min} \leq a_i u + b_i x + c_i \leq \tau_{\max} \\ & x + 2\Delta_i u \in I \end{aligned}$$

The latter two LPs are used to compute the lower and upper bound for the controllable sets K_i in Algorithm 2, inside the subroutine `backwardLP`. While these LPs only consider the torque bounds, it is not too difficult to add direct velocity and acceleration bounds. The bounds given by Equation 3.10 and Equation 3.11 can be incorporated into these LPs.

Extract solution

The algorithm computes at each discretised point the (pseudo) position, velocity and acceleration, but the timing information is not yet computed. To obtain this information, the second-order equations of motion can be used to calculate the time period of position intervals:

$$s_{i+1} = s_i + \Delta_t \gamma_i' + \frac{1}{2}\Delta_t^2 \gamma_i'' \tag{3.29}$$

Constant acceleration in the i-stage interval is assumed. Using the quadratic formula it is possible to solve this equation for Δ_t. While this yields in some cases two solutions (i.e. when $\gamma'' \neq 0$), the correct solution is always the smallest strictly positive solution, because the timestep must always be positive. It remains to be proven why the smallest positive solution is the correct solution for t.

Proof. Let us first rename some variables to make this proof easier to read, i.e. let $v_i = \gamma_i'$, $a_i = \gamma_i''$ and $\Delta_t = t_i$:

$$s_{i+1} = s_i + v_i t_i + \frac{1}{2} a_i t_i^2 \tag{3.30}$$

We will consider three distinct cases: $a_i < 0$, $a_i = 0$ and $a_i > 0$.

Case $a_i = 0$**:** In this case, Equation 3.30 reduces to $s_{i+1} = s_i + v_i t_i$, which yields precisely one strictly positive solution for t_i since $s_i - s_{i+1} > 0$ by construction of the discretised intervals.

Case $a_i > 0$**:** There are two solutions for t which can be found using the quadratic formula:

$$t = \frac{-v + \sqrt{v^2 - 2(s_i - s_{i+1})a_i}}{a_i} \vee t = \frac{-v - \sqrt{v^2 - 2(s_i - s_{i+1})a_i}}{a_i} \tag{3.31}$$

Note that $s_i - s_{i+1} = -\Delta_i$ as defined earlier, making the discriminant $D = v^2 + 2\Delta_i a_i$ precisely equal to the condition in the linear programs. It must therefore be positive by construction, and thus there must exist two solutions for t. In this case, the acceleration is positive and therefore we have that $v < \sqrt{v^2 - 2(s_i - s_{i+1})a_i}$. Using this result, we obtain that one solution for t is negative (since the numerator becomes negative) and the other solution for t is positive, which is the selected solution in this case.

Case $a_i < 0$**:** Using the same analysis as for the case $a_i > 0$ yields that there are two positive solutions for t. Further analysis is thus required. For a solution to be suitable in our case, the derivative of Equation 3.30 should be positive, since we assume that the position always increases. In the case when $a_i < 0$, Equation 3.30 is a mountain parabola, hence for one solution the derivative must be positive, and negative for the other solution. The derivative is given by: $2a_i t + v_i$. Let t_1 and t_2 be the two solutions given by:

$$t_1 = \frac{-v + \sqrt{v^2 - 2(s_i - s_{i+1})a_i}}{a_i}, t_2 = \frac{-v - \sqrt{v^2 - 2(s_i - s_{i+1})a_i}}{a_i} \tag{3.32}$$

Evaluating the derivative at t_1 yields the solution $\sqrt{v^2 - 2(s_i - s_{i+1})a_i}$ and evaluating the derivative at t_2 yields $-\sqrt{v^2 - 2(s_i - s_{i+1})a_i}$. Thus, t_1 is the suitable solution in this case, which is indeed the smallest strictly positive solution.

□

Algorithm

A summary of the steps is given in Algorithm 2. As previously discussed, the algorithm consists of a backward pass and a forward pass. In the backward pass, the subroutine `backwardLP` computes the controllable sets K_i using the LPs defined earlier. The set K_{i+1} is used for I in the LPs. This results in the lower and upper bound for the i-stage controllable set, which is then used to compute the $i - 1$ controllable set, until the 0-stage. Then, the `forwardLP` subroutine computes the control greedily using the following LP:

$$\begin{aligned} \text{maximise} \quad & u \\ \text{subject to} \quad & (x, u) \in \mathbb{R}^2 \\ & \tau_{\min} \le a_i u + b_i x + c_i \le \tau_{\max} \\ & x + 2\Delta_i u \in K_{i+1} \end{aligned}$$

This LP computes the highest possible acceleration, which respects the system bounds, and keeps the trajectory within the already computed controllable sets. Note that the initial controllable set K_n can be an interval having lower and upper bounds too.

Algorithm 2: TOPP-RA

```
$K_n \leftarrow \{\gamma_n'^2\}$ ;
for $i \in \{n-1, \dots, 0\}$ do
    $K_i \leftarrow$ backwardLP($K_{i+1}$) ;
end
if $K_0 = \varnothing \vee \gamma_0'^2 \notin K_0$ then
    Report failure ;
end
$x_0 \leftarrow \gamma_0'^2$ ;
for $i \in \{0, \dots, n-1\}$ do
    $u_i \leftarrow$ forwardLP($x_i$, $K_{i+1}$) ;
    $x_{i+1} \leftarrow x_i + 2\Delta_i u_i$ ;
end
```

Runtime analysis

The number of iterations of this algorithm is linear in N, however, the computational complexity of each iteration depends on the number of constraints. This is theoretically faster than the previously discussed algorithm TOPP-NI. This is because the numerical integration, the time expensive step, has been replaced by a fixed number of linear programs. The linear programs scale with respect to the number of constraints, which in this case remains constant throughout the algorithm. Another advantage is that TOPP-RA does not only yield a trajectory, but the controllable sets themselves can possibly also be used as they form an upper bound on the admissible control. This is also commonly known as Admissible Velocity Propogation, AVP. This can be advantageous to existing geometric planners [38]. Another advantage of TOPP-RA over TOPP-NI is the fact that it can handle parametric uncertainties without much modification.

3.2.3 Comparison

TOPP-NI and TOPP-RA essentially solve the same problem, but achieve this using a different method. TOPP-NI relies heavily on numerical integration in the phase plane, while TOPP-RA uses linear programming to find the optimal control. There are advantages to both methods: For certain use cases where the number of switch points is expected to be low, the TOPP-NI algorithm may be faster as it does not require the overhead of setting up and executing N linear programs. On the other hand, for more complex use cases, TOPP-RA may be significantly faster since the runtime does not depend on the number of switch points, but only on the number of iterations and constraints.

In the introduction of this chapter, q is assumed to be twice differentiable, i.e. to be C^2-continuous. In fact, it is sufficient to require q to be piece wise C^2-continuous. Both TOPP-NI and TOPP-RA require an initial and final velocity as input. It is therefore possible to solve for each segment of q, and string together the solutions for those pieces to obtain the final solution. Actually, TOPP-RA has a major advantage over TOPP-NI in this regard, since it can handle an interval of initial and final velocities while TOPP-NI can only handle a single initial and final velocity.

Chapter 4

Third order solution

Trajectory planning with second order constraints has received much attention. It has been shown that these algorithms can compute a time-optimal trajectory with reasonable computational cost. However, because the nature of these trajectories are bang-bang, the acceleration jumps between its maximum and minimum, which results in infinite jerk. Of course, in practical environments the jerk cannot be infinite. The consequence of this discrepancy is reduced accuracy in path tracking, i.e. the manipulator is likely to deviate from the generated path and trajectory. Constraining the jerk has several advantages:

- A jerk constrained trajectory is smoother than a acceleration constrained trajectory. Smoother trajectories can be tracked more accurately [54]. Moreover, some tasks require a smooth trajectory. Take for instance a manipulator which must move tubes with liquid. Second order trajectories would almost surely cause spillage whereas smooth, third order, trajectories are more suitable.
- Third order trajectories increase the life-span of the manipulators, its actuators and components. Jerky motion tends to cause vibrations in the manipulator by exciting resonant frequencies [54, 55]. Jerk bounded trajectories therefore prevent excessive mechanism wear and prolong the life-span of the manipulator. This makes the manipulator overall more cost-effective since the amount of maintenance, repairs and replacements is reduced.
- In some industrial applications, third order constraints are also a part of the problem definition. For example, electric motors have bounds on their input voltages, which in turn translate to torque rate constraints. Another example is fluids flowing through pistons and being compressed [27].

The most common ways of solving third order trajectories either disregard the system dynamics [54], or require much more computational effort due to the fact that the search space is increased by one dimension [40]. Convex optimisation becomes harder to perform since the introduction of third order dynamics often break the convexity of the equations. Near-optimal trajectories are therefore also considered [24], or planners which subdivide the trajectory into carefully constructed splines or polynomials of the desired degree [56, 57]. Time-optimal third order trajectories do therefore not yet admit a time-efficient algorithm, however, for some industrial applications third order constrained trajectories outweigh the need for time-optimality.

In the previous chapter it has been shown how to efficiently construct a time-optimal second order trajectory. At certain locations, the third order constraints are most likely violated, such as at switch points where the acceleration flips from its allowed maximum to its minimum. Graphically, this is represented in the phase plane as a sharp turn of the trajectory graph. Suppose we could "smooth" out this corner, such that the resulting curve respects the third order constraints. An example is given in Figure 4.1. This chapter explains this novel idea, its implementation and drawbacks.

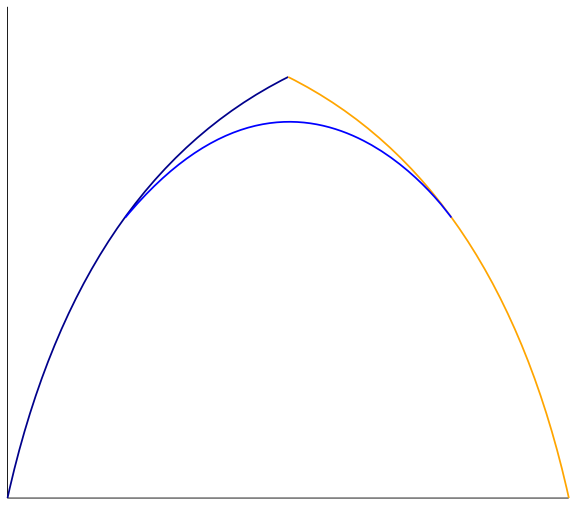

Figure 4.1: Concept of smoothing sharp bends in a trajectory. The blue line represents the "smoothing" segment, connecting the maximum and minimum velocity profiles shown in dark blue and yellow.

4.1 Trajectory smoothing

The principle of smoothing a trajectory is rather simple. Take an initial reference trajectory and transform it until it conforms to the desired constraints. In this case, these are the third order constraints. The previous chapter discussed how to construct time-optimal second order trajectories. These will form the initial reference trajectory to the following smoothing operation.

4.1.1 Tangent Circle Smoothing

Jerk constraints forces the trajectory to be smooth, both as a physical trajectory, and the graphic representation of their velocity. Visually, this means that the trajectory in the phase plane cannot have sharp bends or corners. In other words, the curvature of the trajectory is constrained. Let us now consider a reference trajectory with a point, or an interval, at which the third order bound is violated. Suppose we could gently slide a circle with a radius equal to the maximum allowed curvature into this area, such that it hits the trajectory on the left side of this point or interval, and that it hits the trajectory to the right side. This circle then lies tangent to both the left side and the right side. Instead of the trajectory going around the sharp bend at which the third order constraints are violated, it could follow the circular arc connecting the two tangent points. Since the circle has an admissible curvature, the jerk violation is removed and the trajectory is hence smoothed. This is the basic principle behind the Tangent Circle Method (TCM).

As will be shown in detail later, the smoothing will be applied to the trajectory curve in the (t, γ') plane. Only this plane admits a usable bound on the curvature. Changing the trajectory in this plane changes the traveled distance of the manipulator. This is because the area under the curve corresponds to the distance traveled, and since the smoothing operation strictly lowers the trajectory, the traveled distance is decreased. This loss in distance must be sufficiently compensated for. This is actually not too difficult: we can insert a straight line piece at the highest point of the circular trajectory. The loss in distance can be computed, and therefore the compensation length of this straight segment can be determined. A summary of TCM is given in Algorithm 3. The input are the two segments of the trajectory, A and B, which constitutes the segment on the left of the interval in which the jerk constraint is violated, and the segment on the right. A further explanation of each step is given next.

Algorithm 3: TCM

```
Step 1: compute radius of maximum curvature circle
$R_{\min} \leftarrow (1+\gamma''^2_{\max})^{\frac{3}{2}}/\gamma'''_{\max}$ ;
Step 2: find center of circle and compute circular arc
$A_{\text{par}} \leftarrow$ computeParallel($R_{\min}, A$) ;
$B_{\text{par}} \leftarrow$ computeParallel($R_{\min}, B$) ;
$C \leftarrow$ polyLineIntersection($A_{\text{par}}, B_{\text{par}}$) ;
$T_{\text{arc}}, a_{\text{end}}, b_{\text{begin}} \leftarrow$ computeArc($C$) ;
Step 3: compute compensation distance
$L_0 \leftarrow \gamma(b_{\text{begin}}) - \gamma(a_{\text{end}})$ ;
$L \leftarrow$ chordArea($a_{\text{end}}, b_{\text{begin}}, C$) + quadArea($a_{\text{end}}, b_{\text{begin}}$) ;
$L_{\text{comp}} \leftarrow (L_0 - L)/$top-y($T_{\text{arc}}$) ;
Step 4: Regenerate trajectory data
$T_{\text{new}} \leftarrow$ regenerateTrajectoryData($A, B, T_{\text{arc}}, L_{\text{comp}}$) ;
```

Step 1: compute maximum curvature

The first step of the algorithm is to compute the radius of the circle which has the maximum allowed curvature, depending on the jerk constraint. For a twice differentiable parametric representation of a curve $\ell(t) = (x(t), y(t))$, the curvature is defined as follows:

$$k = \frac{|x'y'' - y'x''|}{(x'^2 + y'^2)^{\frac{3}{2}}} \tag{4.1}$$

To obtain the curvature of the trajectory in the (t, γ') plane, the derivatives of x and y can be substituted by derivatives of t and γ respectively as follows:

$$\begin{aligned} k &= \frac{|1 \cdot \gamma''' - \gamma'' \cdot 0|}{(1^2 + \gamma''^2)^{\frac{3}{2}}} \\ &= \frac{|\gamma'''|}{(1 + \gamma''^2)^{\frac{3}{2}}} \end{aligned} \tag{4.2}$$

t is a linear function, and therefore the second and third derivative are respectively 1 and 0. Note that the γ''' term appears alone, and the denominator is bounded by $1 \leq (1 + \gamma''^2)^{\frac{3}{2}}$. The radius is related to curvature by:

$$R = \frac{1}{k} = \frac{(1 + \gamma''^2)^{\frac{3}{2}}}{|\gamma'''|} \tag{4.3}$$

To bound the radius, we consider the maximum allowed jerk $\gamma'''_{\max}$ and the maximum acceleration $\gamma''_{\max}$ present in the reference trajectory. The minimum curvature is then given by:

$$R_{\min} = \frac{(1 + \gamma''^2_{\max})^{\frac{3}{2}}}{\gamma'''_{\max}} \tag{4.4}$$

This concludes this step of the algorithm. However, it may be useful to understand why this method does not work in the phase plane as explained in the previous chapter. It is precisely because this relation between $\gamma'''_{\max}$ and $R_{\min}$ does not exist in a useful way. Suppose we substitute for x and y the quantities γ and γ' in Equation 4.1 respectively, then we obtain the following equation:

$$k = \frac{|\gamma' \cdot \gamma''' - \gamma''^2|}{(\gamma'^2 + \gamma''^2)^{\frac{3}{2}}} \tag{4.5}$$

It is not possible to place a suitable bound on k using this equation and therefore it is impossible to define a maximum curvature.

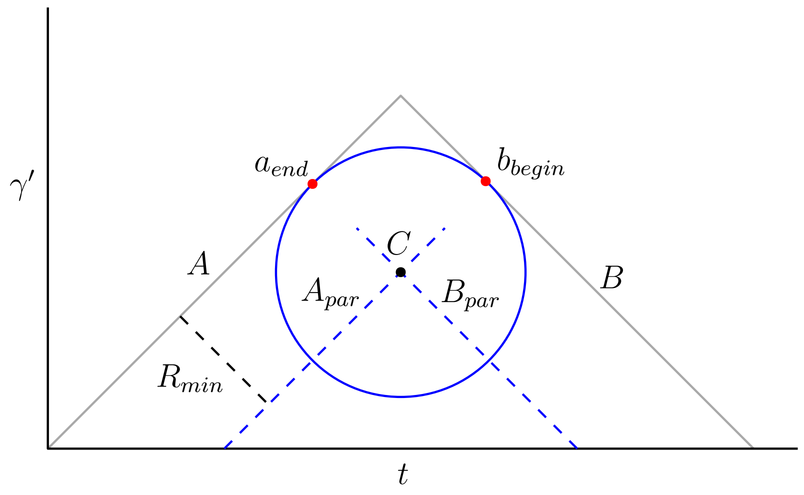

Figure 4.2: Example construction of the center of the circle. The dashed blue lines, A_{par} and B_{par}, represents the parallel lines to the reference trajectory given in gray.

Step 2: Compute circular arc

The second step of the smoothing algorithm consists of computing the circular arc connecting segment A and B. This circle must lie tangent to both trajectories A and B, otherwise a smooth transition from either of those trajectories to or from the circular arc cannot be made. To find all the locations at which the circle is at the right distance and tangent to those segments, two parallel lines A_{par} and B_{par} must be constructed at a distance of R_{min}. This is shown visually in Figure 4.2. The reference trajectory is given in gray and the parallel lines are shown as blue dashed lines. At the intersection of these two parallel lines, the center of the circle can be placed such that it is tangent to both A and B. Since the trajectories are polylines, i.e. consisting of straight line segments, a parallel line is easily computed by simply taking the normal to the segment, traveling a distance of R_{min} and placing a point for the parallel polyline. Finally, given the center of the circle, the arc is constructed starting from a_{end}, the point connecting A and the arc, and ends in b_{begin}, the point connecting the arc to B.

Note that in some extreme cases, the parallel lines might not intersect and cause difficulties in computing the center of the circle. Those edge cases require further analysis, but should not be too hard to solve. An example of such a scenario is when the distance R_{min} is larger than the length of segment A. In this particular case, circle should lay tangent to B and cross the first point of A to form a solution. These edge cases are not considered.

Step 3: compute compensation distance

As explained in the introduction to this method, the graph of the trajectory in the (t, γ') plane is locally lowered and therefore the distance traveled is decreased. To account for this, a straight line segment can be added to counteract this loss. A suitable place for this straight line segment is to insert it at the top of the arc. This keeps the arc tangent to the initially found points on A and B, and the length of this piece is easily computed. The loss in distance is the previously travelled distance minus the new travelled distance. The previously travelled distance is easily available, since the reference trajectory contains the distance parameter γ for each point, it is therefore equal to:

$$L_0 = \gamma(b_{\text{begin}}) - \gamma(a_{\text{end}}) \tag{4.6}$$

Where $\gamma(\cdot)$ acts as a accessor function. The new traveled distance, without the straight line segment for now, is the area under the curve between a_{end} and b_{begin}. This is equal to the area of the quadrilateral connecting these points and their perpendicular position on the γ axis, and the

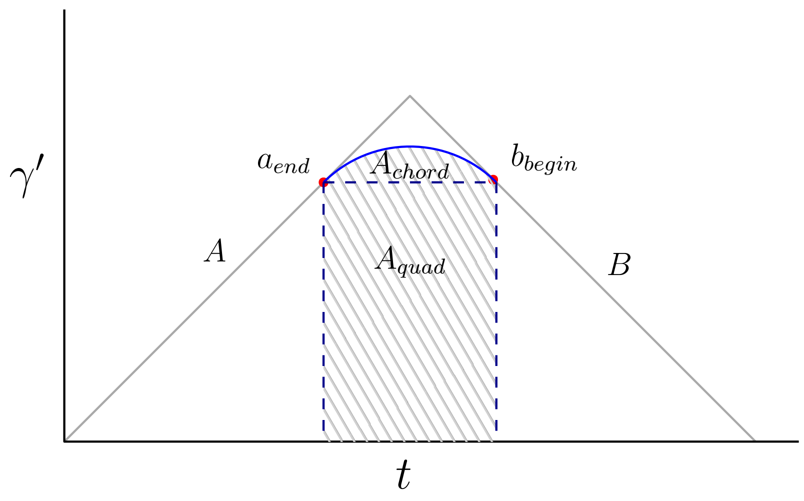

Figure 4.3: Computation of the area under the circular arc, consisting of the area of the quadrilateral A_{quad}, and the area of the chord A_{chord}.

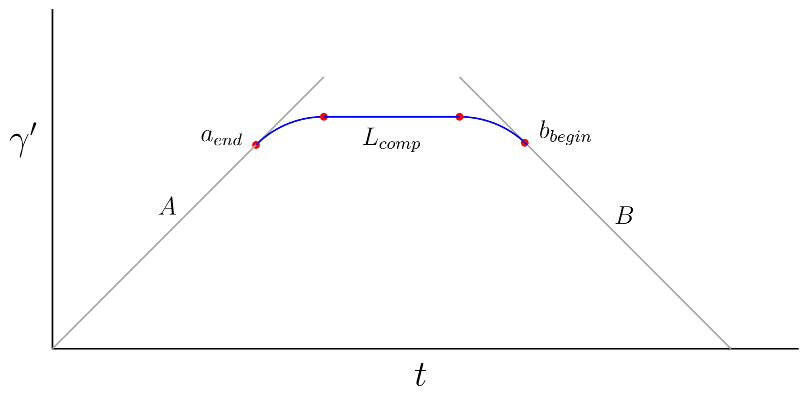

Figure 4.4: The completed trajectory with inserted compensation straight line segment, pushing the right trajectory segment B further to the right.

area of the circular chord, which is given by:

$$A_{\text{chord}} = \frac{1}{2} R_{\text{min}}^2 (\angle a_{\text{end}} C b_{\text{begin}} - \sin(\angle a_{\text{end}} C b_{\text{begin}})) \tag{4.7}$$

This is shown visually in Figure 4.3, where the shaded area represents the area of the quadrilateral plus the chord. The length of the straight line segment, given in time, is then the loss in distance, divided by the pseudo velocity at the top of the circular arc, as is computed in the algorithm. Figure 4.4 shows how the straight line segment is inserted, which pushes the right trajectory segments to the right.

Step 4: regenerate trajectory data

The final step of the algorithm is to complete the trajectory data. Each point of the reference trajectory contains the timestamp, position, pseudo velocity and pseudo acceleration. The inserted arc and straight line segment only contain the timestamp and velocity. Therefore, the missing data must be computed. This is not too difficult, as this information can be derived from the existing information. The new position γ_i for a point i is given by:

$$\gamma_i = \gamma_{i-1} + (t_i - t_{i-1}) \cdot \gamma'_{i-1} \tag{4.8}$$

And the acceleration at point i is given by:

$$\gamma_i'' = \frac{\gamma_{i+1}' - \gamma_i'}{t_{i+1} - t_i} \tag{4.9}$$

4.1.2 Runtime analysis

The input to this algorithm is a reference trajectory split around the interval at which the jerk bound is violated. We therefore do not consider the runtime require to generate this reference trajectory. For this, we refer the reader to the previous chapter about the second order algorithms. Most steps in Algorithm 3 require constant time, as they are simple calculations. The parallel line computation requires linear time, and the polyline intersection algorithm requires $O(n \log n + I \log n)$ time, where n is the number of line segments and I is the number of intersections [53]. The number of intersections is, however, most likely not in the order of n because of the construction of the parallel polylines.

4.1.3 Further remarks

The proposed method is suitable to smooth out trajectories with high jerk, such as encountered at switch points. The bound on the curvature is not tight, however, which means that the jerk of the smoothed out segment may lie between the desired bounds, but not at the limit. The resulting trajectory is therefore not third order optimal, but this method merely reduces the overall jerk of a solution. This can both be an advantage is that was the goal, or an disadvantage if faster trajectories are still required.

This method works well when the acceleration changes from maximum to minimum. In such cases, the smoothed trajectory lies beneath the reference trajectory, having a lower velocity at the smoothed out segment. This means that the smoothed trajectory can never cross the MVC, considering that the reference trajectory never crossed the MVC. When this procedure is applied to a segment where the acceleration goes from minimum to maximum, then the smoothed out segment lies above the reference trajectory. In such a case, it could violate the MVC bound. For these cases, further analysis is required.

A related field of research is Dubins paths [58]. A Dubins path is a curve which connects two points in euclidean space with constrained curvature. It has been proven that such paths exist and consist only of circular arcs of maximum curvature and straight line pieces. This is related to the smoothing procedure explained in this chapter: two (to be determined) points must be connected by a curve with a maximum curvature. The results and algorithms developed in this area could unfortunately not be applied to this case, due to several reasons which we were not (yet) able to overcome: the start and end point are not known beforehand, the smoothed trajectory must be monotonically increasing in the horizontal direction and the velocity γ', i.e. the highest point, must ideally be maximised.

Chapter 5

Experiments

In this chapter, the algorithms and methods discussed in previous chapters are applied to experimental and realistic input data, i.e. geometric paths for robotic manipulators and their corresponding constraints. We will show different visualisations of the various steps in the algorithm, and that indeed the input constraints are respected in the output trajectory. All experiments are performed using Python 3.6 on a desktop running 64-bit Windows 10 build 19043.1110, Intel Core i5-4460 (3.20 GHz) as CPU and with 8GB of RAM.

5.1 Application on use cases

In this section, the TOPP-RA algorithm is applied to several selected manipulators and input paths. The intermediate results of the algorithm, i.e. the trajectories in the phase plane, are shown, as well as output statistics to show that the trajectories indeed respect the system constraints.

5.1.1 Input definition

Several manipulators are considered, for which their shape and paths are explained in this section. For each manipulator, their Denavit–Hartenberg parameters (DH parameters) [59] are given. The DH parameters, or also known as the DH convention, are a four parameters which describe how to attache reference frames to links of robotic manipulators. The specific mass distribution, and hence inertia and Coriolis tensors, are not given as the specific values are not relevant for the experiments. The Robotics Toolbox developed by Peter Corke[1] is used to compute these matrices, and to compute the configuration of the robot given the DH parameters.

Basic arm

This manipulator consists of a single rotating axis, moving a weight at the end of the arm. The constraints on the torque are sufficiently high such that the trajectory does not get near the maximum velocity curve. This is to provide a clear example of the "bang-bang" control structure. The DH parameters, while rather trivial for this robotic manipulator, are given in Table 5.1. The trajectory is also fairly simple: the arm makes a 180° rotation. The input functions are thus given by:

$$q(s) = s \cdot \pi \qquad q'(s) = \pi \qquad q''(s) = 0 \tag{5.1}$$

The torque limits are set sufficiently high. This use case will also be used to demonstrate the direct velocity bounds using Equation 3.10. Let the maximum velocity be $\hat{q}'_{i,\min} = 1$. The limit on the pseudo velocity is then calculated as $\hat{q}'_{i,\min}/q'_i$ for each position. Since the first path derivative is a constant, this results in constant upper bound for γ', namely $1/\pi$.

[1]`https://petercorke.com/toolboxes/robotics-toolbox/`

Joint	Type	θ	d	a	α
1	Rotational	0	0	0.1	0

Table 5.1: DH parameters for basic arm manipulator

3D printer

A common 3D printer setup consists of three prismatic joints (i.e. translational axis), allowing the printer head to move in three dimensions. Depending on the specific use case, a rotational axis can be attached to the end to allow for rotation of the tool tip. This is not required for simple 3D printing tasks, and has not been included in this experiment. The DH parameters are given in Table 5.2. The programmed movement is the tool tip performing a circular motion translated from its origin. This is described by:

$$q_1(s) = a\cos(2\pi s) + b \qquad q_2(s) = a\sin(2\pi s) + b \tag{5.2}$$

$$q_1'(s) = 2\pi \cdot -a\sin(2\pi s) + b \qquad q_2'(s) = 2\pi \cdot a\cos(2\pi s) + b \tag{5.3}$$

$$q_1''(s) = (2\pi)^2 \cdot -a\cos(2\pi s) + b \qquad q_2''(s) = (2\pi)^2 \cdot -a\sin(2\pi s) + b \tag{5.4}$$

for some parameters a, b and $0 \leq s \leq 1$. The other joints do not move and therefore their functions return zero.

Joint	Type	θ	d	a	α
1	Prismatic	$\pi/2$	9.5	-0.5	$\pi/2$
2	Prismatic	$\pi/2$	-1	1	$\pi/2$
3	Prismatic	$-\pi/2$	0.5	-1	$\pi/2$
4	Prismatic	$\pi/2$	0.5	0	π

Table 5.2: DH parameters for 3D printer manipulator

Panda

Finally we consider the Panda robotic arm developed by FRANKA EMIKA[2]. It is a 7-DOF manipulator aimed for usages in production lines, hospitals, laboratories, logistics platforms and more. The DH parameters for this robot are given in Table 5.3. Note that the angle offsets need not to be strictly specified. The movement of this manipulator is a simple linear interpolation between two configurations $C_1 = (0,0,0,0,0,0,0)$ and $C_2 = (\pi/2, -0.3, \pi/4, -3.2, 3, 2, \pi/4)$. This movement moves the manipulator between the default configuration to a carefully selected ending configuration such that the MVC limit is reached in the phase plane graph.

Joint	Type	θ	d	a	α
1	Rotational	θ_1	0.33	0	0
2	Rotational	θ_2	0	0	$-\pi/2$
3	Rotational	θ_3	0.316	0	$\pi/2$
4	Rotational	θ_4	0	0.0825	$\pi/2$
5	Rotational	θ_5	0.384	-0.0825	$-\pi/2$
6	Rotational	θ_6	0	0	$\pi/2$
7	Rotational	θ_7	0	0.088	$\pi/2$

Table 5.3: DH parameters for Panda manipulator

[2]https://www.generationrobots.com/en/403317-panda-robotic-arm.html

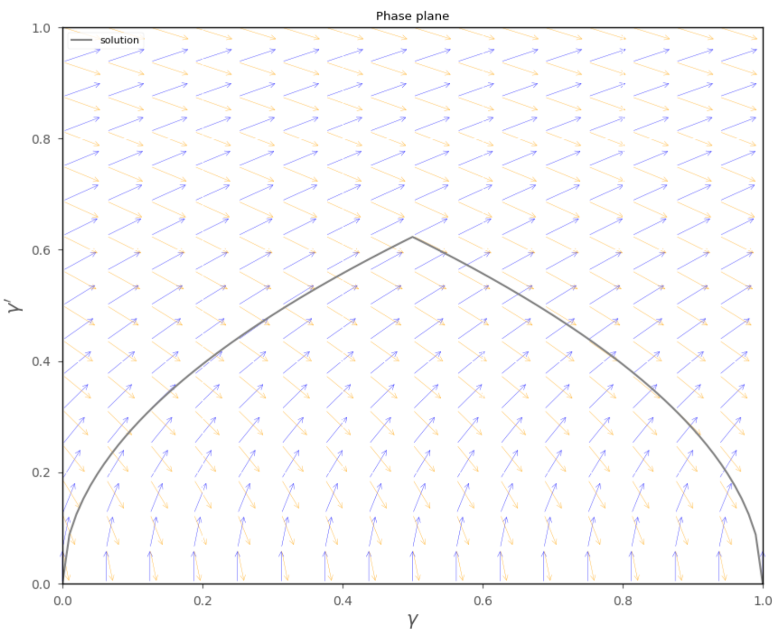

Figure 5.1: Phase plane visualisation of the basic arm.

5.1.2 Results

The TOPP-RA algorithm is applied to all manipulators discussed above. The number of intervals N is set to 1000 to allow for a sufficient accuracy and resolution of the output. For each manipulator, the phase plane will be shown, as well as the derivatives of the joints and the torques. Finally, graphs of the function γ are shown.

The images of the phase plane consists of three elements: the profile vectors, the maximum velocity curve (MVC) and the trajectory itself. The blue and yellow arrows show the direction of respectively the maximum and minimum velocity profiles at discretised positions. The curve shown in gray represents the solution found by the algorithm. The MVC may appear jagged instead of a smooth line. This is because the MVC is also computed by means of sampling at discretised positions since there is no closed form formula describing this curve.

For the basic arm manipulator shown in Figure 5.1, the forces stay well within the bounds and therefore the MVC is not visible in shown range for γ'. At first, the trajectory moves up following maximum acceleration until some point $\gamma' \approx 0.6$, after which the minimum acceleration profile is followed until the manipulator has reached its goal. This clearly depicts the “bang-bang” control structure of an optimal solution. Figure 5.2 shows the phase plane for the basic arm with the direct velocity constraint. Note that the MVC is actually situated right on the straight gray trajectory, but because of sampling it appears higher. The pseudo velocity reaches the limit which we already computed. The phase plane graph of the 3D printer as shown in Figure 5.3 shows shows more interesting features. Firstly, the MVC is clearly visible and constraining the trail of the solution. It consists of several peaks and valleys, which could trap the trajectory if it did not switch at the right moment. These switch points are easily distinguishable as well because their sharp bends. Interestingly, the MVC appears to be symmetric. This is not too surprising given the fact that the geometric path for this manipulator is also symmetric, i.e. a circle. The phase plane of the Panda manipulator is shown in Figure 5.4. This graph is similar to the 3D printer graph as it shown several clear peaks and valleys in the MVC, which the trajectory carefully avoids as intended.

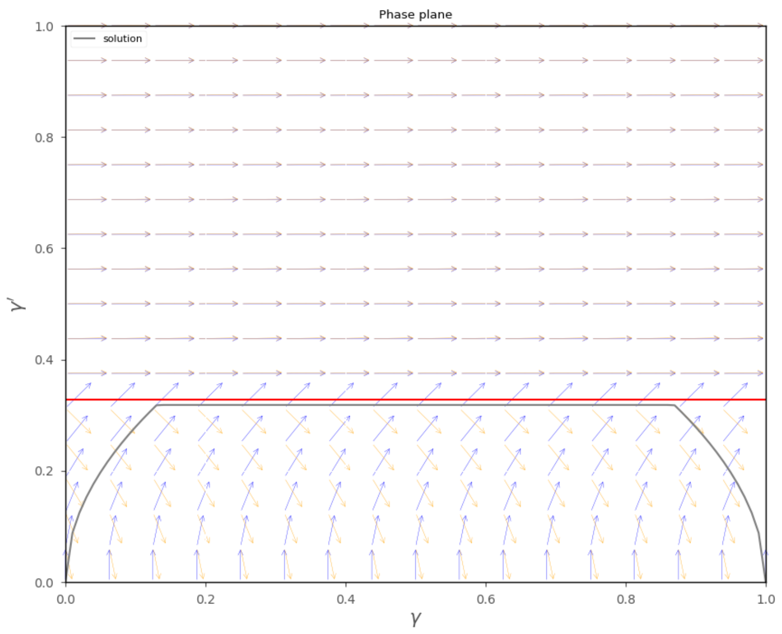

Figure 5.2: Phase plane visualisation of the basic arm, with a direct velocity constraint.

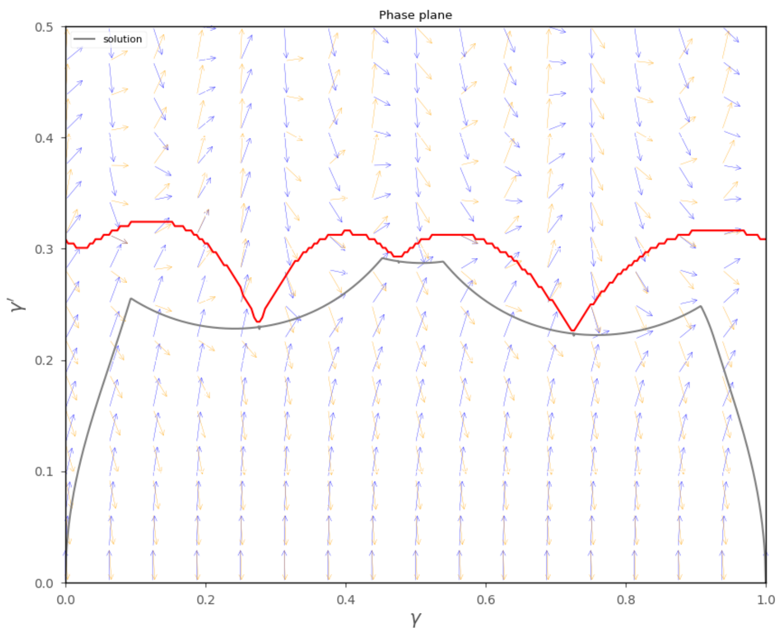

Figure 5.3: Phase plane visualisation for the 3D printer manipulator.

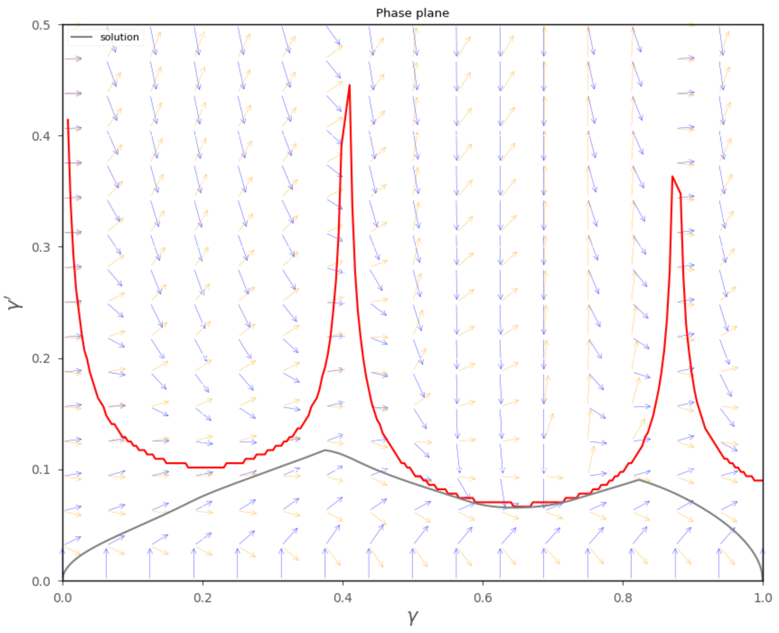

Figure 5.4: Phase plane visualisation for the Panda manipulator.

The graphs of the joint derivatives, i.e. the position, velocity and acceleration, show the second order nature of the solution and again the "bang-bang" control structure. Figure 5.5 shows the joint derivatives for the basic arm manipulator. This shows a typical second order graph, where the acceleration is a constant straight line (with one discontinuity), the velocity is linear and the position follows a quadratic curve. The joint derivatives graph for the case when then direct velocity constraint is added (Figure 5.6) clearly shows the desired maximum velocity. The velocity increases linearly until it has reached its maximum and continues until deceleration. The 3D printer, shown in Figure 5.7, shows a more interesting graph for the joint derivatives. The first and last actuator perform no action and therefore their joints do not move. Joint 3 is most of its time operating at its maximum. The short period in the middle of the action when it is not operating at its maximum, joint 2 is operating at its upper bound. As is clearly shown, always one joint is operating at its maximum and therefore the manipulator is operating optimally. The sudden spikes in the graph can be explained by numerical instability. In the joint torques graph of the Panda manipulator shown in Figure 5.8, the joint velocity and acceleration are less important. What is interesting to see is that the position increases non-linearly. The input was a linear interpolation between two configurations, and thus the algorithm has indeed changed the timing for the path in a non-linear fashion.

The joint derivatives graph does not show the precise joint torque limits as was defined in the input. For this we have plotted the joint torque in Figure 5.9 for the basic arm, Figure 5.10 for the 3D printer and in Figure 5.11 for the Panda manipulator. It is clearly visible that in each of these graphs, at any point in time at least one joint operates at its maximum. The discontinuities can again be explained because of numerical instability. One can also observe that (almost) at most one joint is operating at its maximum allowed torque at any time. Disregarding possible approximation errors, this is precisely as Chen et al. have shown in their work [45].

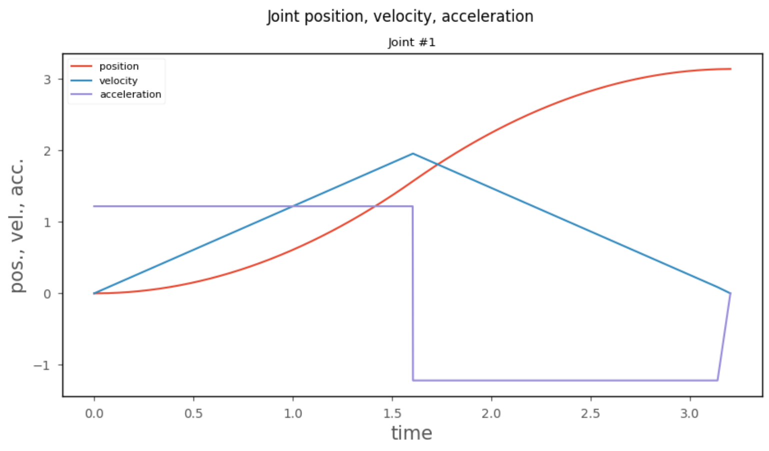

Figure 5.5: Joint derivatives for the basic arm; position (red), velocity (blue) and acceleration (purple).

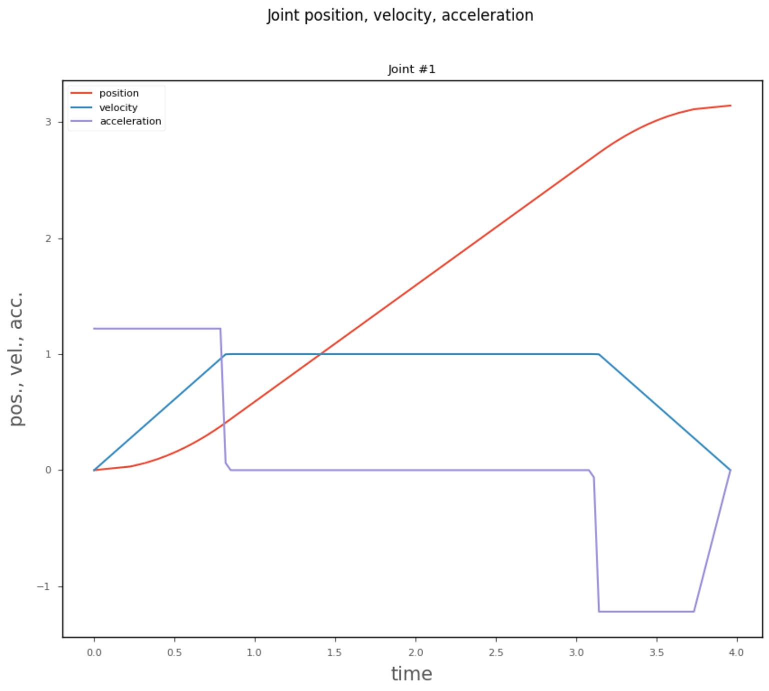

Figure 5.6: Joint derivatives for the basic arm with direct velocity constraint; position (red), velocity (blue) and acceleration (purple).

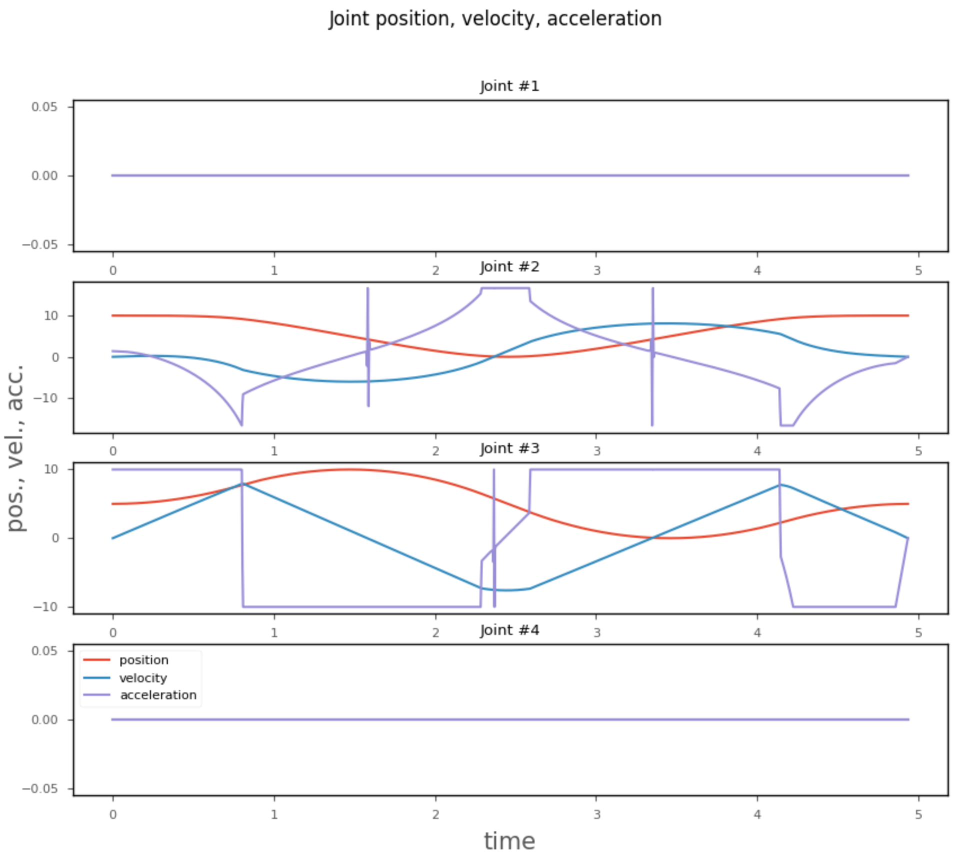

Figure 5.7: Joint derivatives for the 3D printer; position (red), velocity (blue) and acceleration (purple).

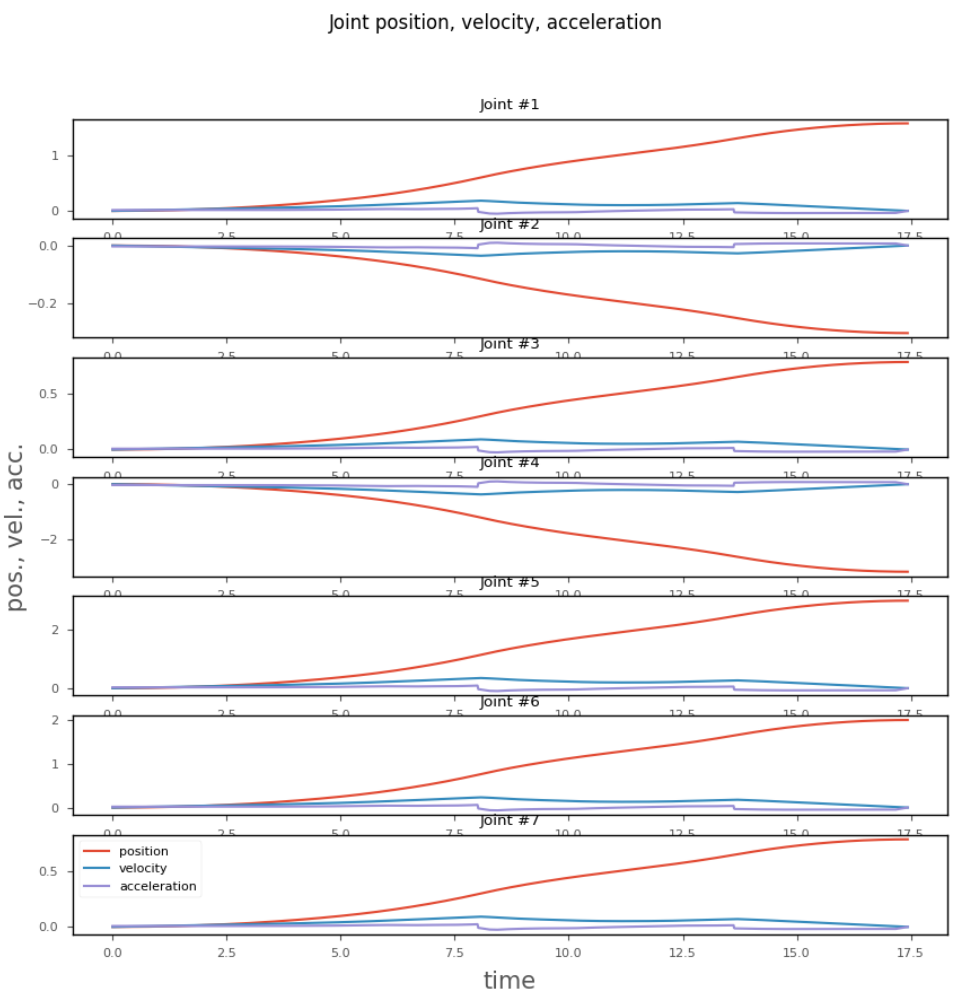

Figure 5.8: Joint derivatives for the Panda manipulator; position (red), velocity (blue) and acceleration (purple).

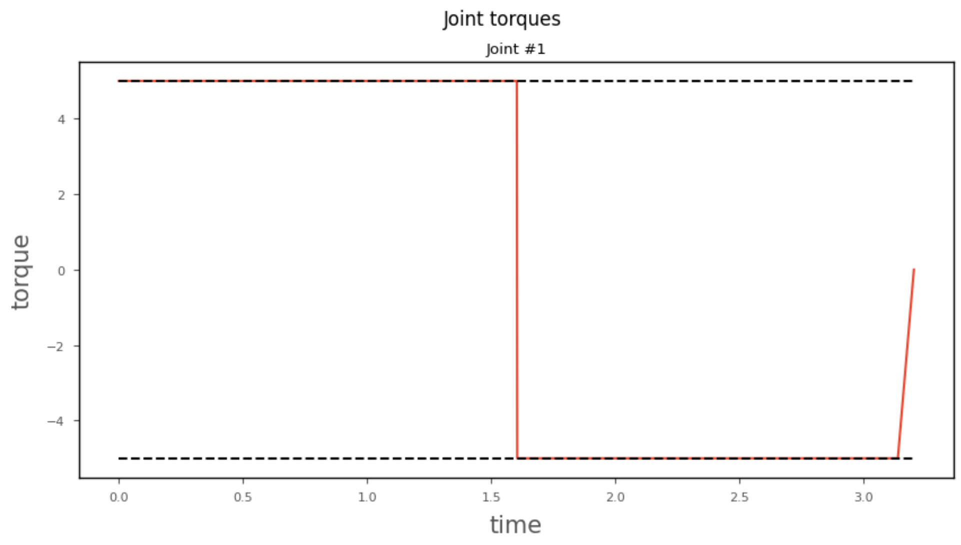

Figure 5.9: Joint torques for the basic arm; torque (red) and limits (dashed, black)

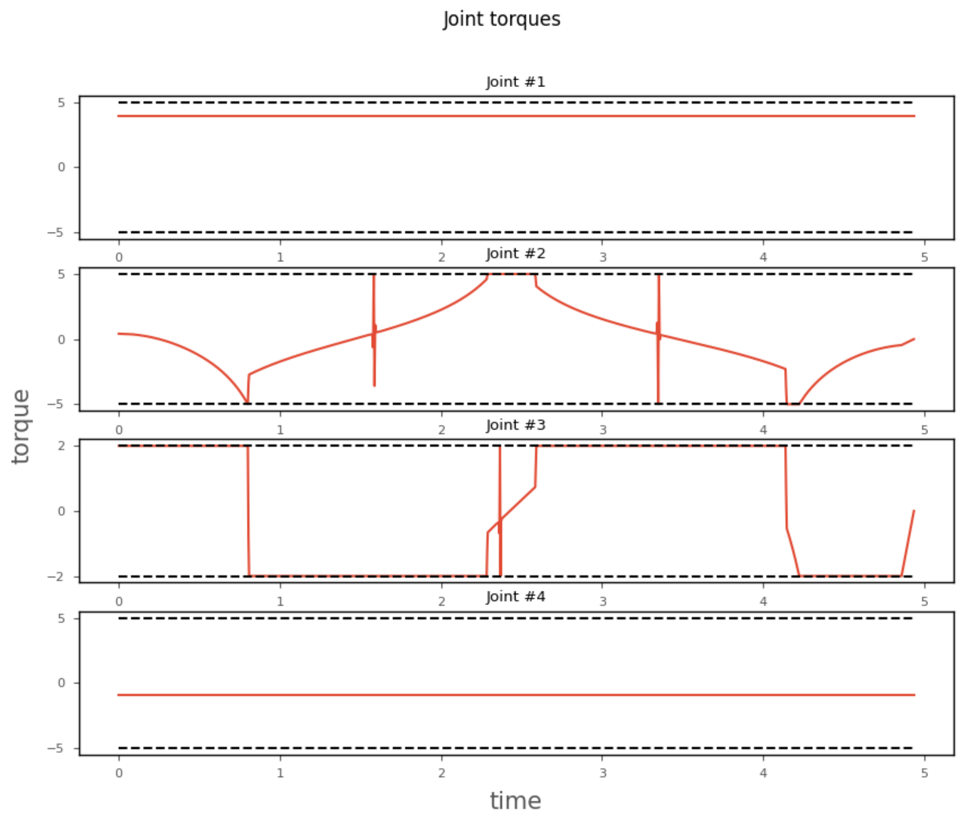

Figure 5.10: Joint torques for the 3D printer; torque (red) and limits (dashed, black)

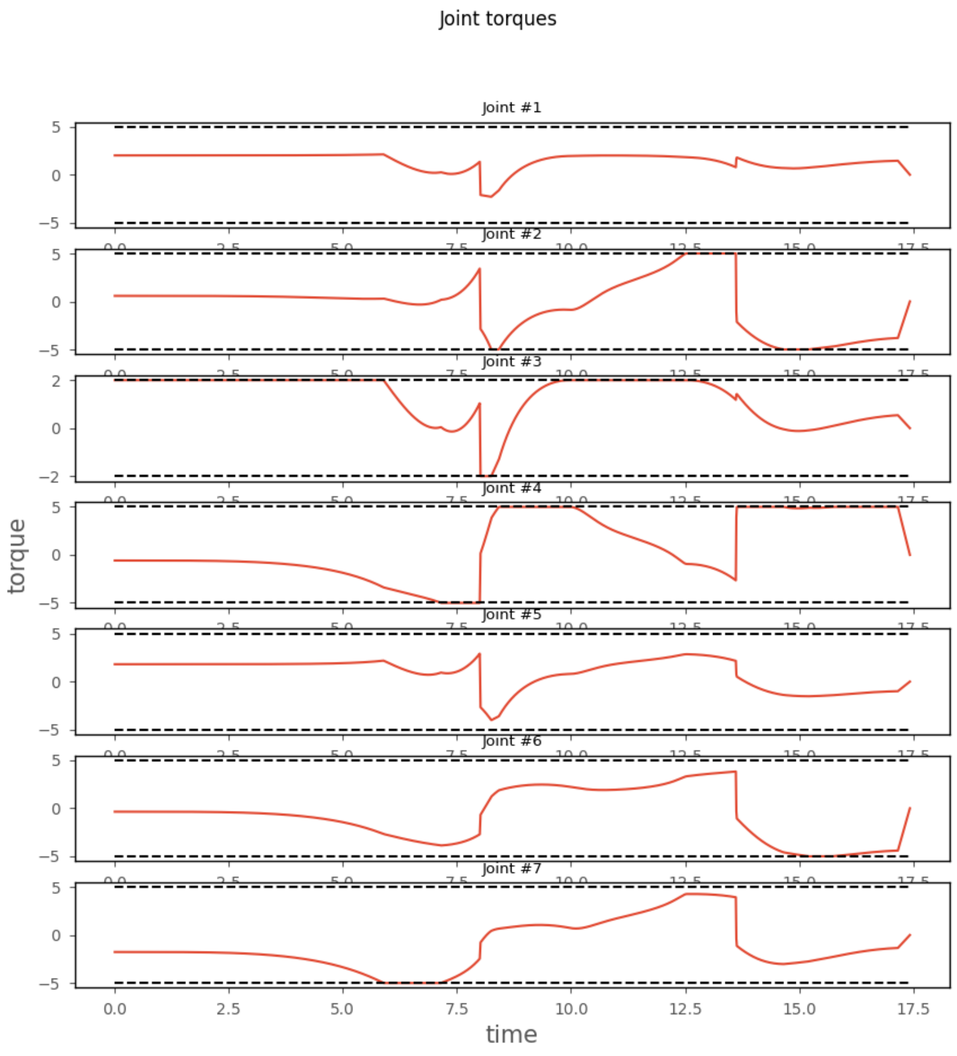

Figure 5.11: Joint torques for the Panda manipulator; torque (red) and limits (dashed, black)

5.2 Comparison TOPP-NI, TOPP-RA

To investigate the performance of each algorithm, TOPP-NI and TOPP-RA are run on two sets of inputs with varying parameter settings. The inputs are taken from the previous section: the basic arm and the 3D printer. For TOPP-NI, the parameter Δt is varied while for TOPP-RA, the parameter N is varied. To compare the results, the total running time (in milliseconds) and total trajectory traversal time T_{end} (also in milliseconds) are recorded, and averaged over five runs. It is expected that TOPP-NI will take significantly longer to run than TOPP-RA, but the total traversal time computed by both algorithms is expected not to differ by a significant amount.

Use case	-	$\Delta t = 0.1$	$\Delta t = 0.05$	$\Delta t = 0.01$	$\Delta t = 0.005$
Basic arm	Runtime	129	265	1 864	5 033
	T_{end}	3 300	3 250	3 210	3 215
3D printer	Runtime	4 432	9 459	53 934	123 094
	T_{end}	5 300	5 100	4 990	4 965

Table 5.4: Performance results for TOPP-NI algorithm

Use case	-	$N = 50$	$N = 100$	$N = 1000$	$N = 10000$
Basic arm	Runtime	165	219	2 218	22 011
	T_{end}	3 210	3 187	3 202	3 210
3D printer	Runtime	526	589	5 805	63 491
	T_{end}	4 915	4 918	4 937	4 945

Table 5.5: Performance results for TOPP-RA algorithm

The results are shown in Table 5.4 for TOPP-NI and in Table 5.5 for TOPP-RA. A couple of interesting remarks can be made on this data. Firstly, both TOPP-NI and TOPP-RA appear to have a linear relationship between their parameter (i.e. Δt and N) and the total runtime. Another interesting observation is that TOPP-NI has a lower runtime on the basic arm use case, but it has a much higher runtime on the 3D printer use case than TOPP-RA. This difference can be explained by the number of switch points, since the dominant factor adding runtime in TOPP-NI is the subroutine responsible for handling those points. The total traversal time T_{end} also seems to be more consistent for different values for the parameter for TOPP-RA than for TOPP-NI. This is most likely explained by numerical instability of the integration in TOPP-NI when using a larger integration interval.

To show that both algorithms produces the same result, one can compare the (t, γ) graph for each manipulator. If the algorithms produces the same output, their graph should look the same. This has been done in Figure 5.12 for the 3D printer manipulator. The blue line corresponds to the output of the TOPP-NI algorithm and the orange line for the TOPP-RA algorithm. As one can see, their graphs look almost perfectly similar, indicating that they indeed produce the same output, with some margin on the error.

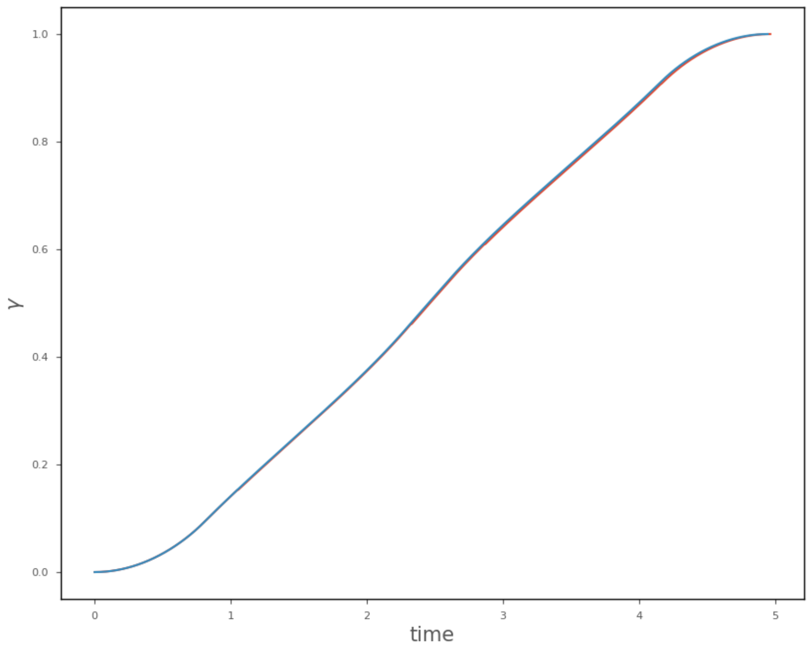

Figure 5.12: Comparison of (t, γ) graph of TOPP-NI and TOPP-RA.

5.3 Smoothing

To show the practical application of the smoothing algorithm, we first require a reference trajectory. For this, we use the TOPP-RA algorithm on the basic arm manipulator use case. The other use cases include some edge cases which are not implemented in this thesis.

Applying the smoothing subroutine on the reference trajectory in the (t, γ) plane results in the picture shown in Figure 5.13. The gray reference trajectory shows a spike at the middle of the movement, which is precisely the switch point in the phase plane shown in Figure 5.1. The blue trajectory consists of a circular arc, a straight line segment, and another circular arc. Since the new, blue, trajectory reaches a smaller pseudo velocity, the total traversal time must increase. This is clearly visible in the graph.

The torque bounds of this new trajectory are shown in Figure 5.14. The only and slight violation of this bound at $t = 0.9$ occurs when the trajectory switches between the reference trajectory and the circular arc. This can easily be fixed by exactly computing a suitable switch point, instead of reusing existing points on these trajectories, as is done in the current implementation. Note that the torque is not at its maximum at any time. This is because the radius of the circular arc is computed using the maximum acceleration present in the trajectory, and the radius may be smaller over an interval where the acceleration is smaller than the maximum acceleration.

Finally, we compare the (t, γ) graph of the reference trajectory and the smoothed trajectory. This is shown in Figure 5.15. The reference trajectory shown in red reaches a steeper slope, while the smoothed blue trajectory does not. It is again visible that the smoothed trajectory has a larger traversal time.

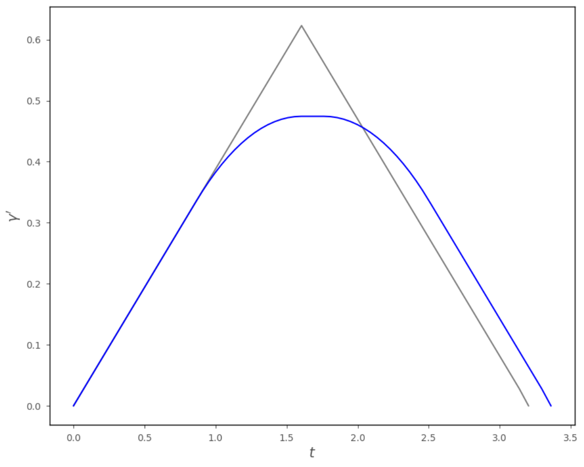

Figure 5.13: Reference trajectory (gray) and smoothed trajectory (blue) shown in (t, γ') plane.

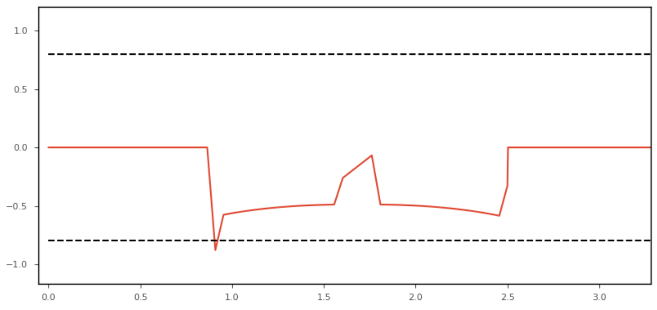

Figure 5.14: Torque of smoothed trajectory. Bounds are displayed as black, dashed, lines.

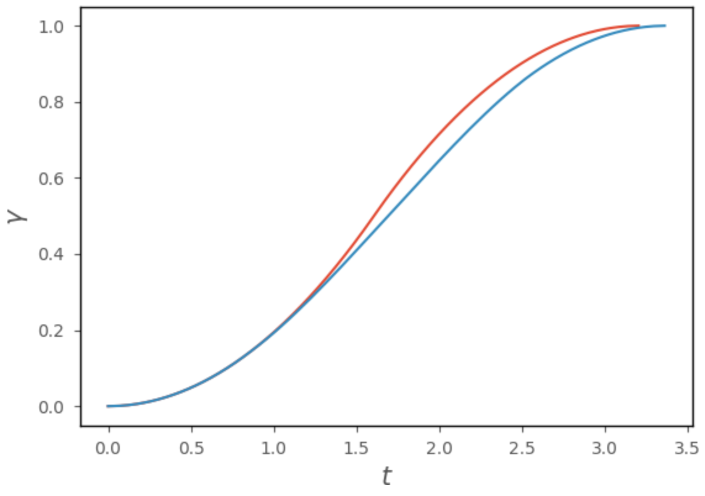

Figure 5.15: (t, γ) graph of the reference trajectory (red), and the smoothed trajectory (blue).

Chapter 6

Conclusion

In this chapter we summarise the most important results, as well as explain a direction for further research and improvements.

6.1 Summary

In this thesis, we have focused on time-optimal trajectory generation for n-DOF manipulators, considering their full non-linear dynamics. A literature survey has been performed which includes most of the major works done within this research area. It has been observed that second order constraints have received more attention in the literature than third (or higher) order constraints. The second order case admits a computationally efficient time-optimal algorithm, but for the third order constraints this is only achievable by approximation.

Two algorithms which handle second order constraints are investigated and implemented: TOPP-NI and TOPP-RA. The practicalities of these algorithms are studied, and they have been applied to several use cases. Important details regarding the implementation have been discussed. For TOPP-NI, this entails the subroutine to find the tangent trajectory. For TOPP-RA, the trajectory regeneration subroutine has been discussed and proven to yield correct results. It has been shown that the solution produced by these algorithms indeed respect the system bounds in the practical examples. The "bang-bang" control structure was clearly visible in the output.

For the third order constraints, a smoothing algorithm has been developed based on bounding the curvature in the (t, γ') plane. Jerky motion is identified by sharp bends and corners in this plane, which intuitively can be smoothed out by constructing circular arcs. A bound on the curvature has been identified, which is used in this construction. This method has then been applied to a use case to show its effectiveness. The resulting trajectory is not time-optimal anymore, but near time-optimal, which is in some situations more desirable. Edge cases appearing in this method have been discussed, but a complete solution has not been given.

6.2 Improvements

The results obtained in this thesis leave room for further improvements. We present here a list of several directions for future ideas and works:

- To improve the runtime of the TOPP-NI algorithm, the results from Slotine et al. [10] can be used to precompute the switch points. This could potentially lead to an algorithm which makes use of multi-threading, solving subproblems in parallel.
- The smoothing procedure as presented in this thesis is merely a concept which can be build upon. Edge cases were not yet considered, such as when the parallel lines do not have an intersection point. There also exists a potential failure when the smoothing procedure is

applied to a valley in the trajectory, where it avoids a valley in the MVC. In such a scenario, the smoothed trajectory could potentially penetrate the inadmissible area.

- The smoothing procedure is faintly related to Dubins paths. It may be possible to combine results from that area of research to the problem of trajectory smoothing. Due to problems discussed in Section 4.1.3, it was not (yet) possible to work out this idea. Perhaps by careful manipulation of the constraints, it may be possible to use Dubins paths to find a suitable, smoothed, trajectory.
- No error bounds have been given for the results in this thesis. Since all these methods rely on some form of sampling, error bounds can be used to better understand the practical limitations of these algorithms. Furthermore, for high-precision machines, they may even be strictly required.

Bibliography

[1] International Organization for Standardization. Iso/tc 299 robotics. Technical report, International Organization for Standardization, 2018. 1

[2] International Federation of Robotics. Executive summary world robotics 2020 industrial robots. Technical report, International Federation of Robotics, 2020. 1

[3] S. Dubowsky, E. Vance, and M. Torres. The control of space manipulators subject to spacecraft attitude control saturation limits. 1989. 1, 5

[4] Z. Shiller and S. Dubowsky. On the optimal control of robotic manipulators with actuator and end-effector constraints. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 614–620, 1985. 1, 4

[5] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. On the optimal control of robotic manipulators with actuator constraints. In *1983 American Control Conference*, pages 782–787, 1983. 4

[6] J. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4:17 – 3, 1985. 4, 6, 8, 12, 15, 16

[7] Kang Shin and N. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, 1985. 4, 6

[8] F. Pfeiffer and R. Johanni. A concept for manipulator trajectory planning. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1399–1405, 1986. 4

[9] S. Dubowsky, M. Norris, and Z. Shiller. Time optimal trajectory planning for robotic manipulators with obstacle avoidance: A cad approach. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1906–1912, 1986. 4

[10] J.-J.E. Slotine and H.S. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics and Automation*, 5(1):118–124, 1989. 4, 5, 16, 42

[11] Z. Shiller and H.-H. Lu. Robust computation of path constrained time optimal motions. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 144–149 vol.1, 1990. 4

[12] Zvi Shiller and Hsueh-Hen Lu. Computation of path constrained time optimal motions with dynamic singularities. *Journal of Dynamic Systems Measurement and Control-transactions of The Asme - J DYN SYST MEAS CONTR*, 114, 03 1992. 4

[13] Mohammad Hassan Ghasemi and Mohammad Jafar Sadigh. A direct algorithm to compute the switching curve for time-optimal motion of cooperative multi-manipulators moving on a specified path. *Advanced Robotics*, 22(5):493–506, 2008. 5

[14] Quang Cuong Pham. A general, fast, and robust implementation of the time-optimal path parameterization algorithm. *IEEE Transactions on Robotics*, 30, 12 2013. 5

[15] S. Ma. Time optimal control of manipulators with limit heat characteristics of actuators. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*, volume 1, pages 338–343 vol.1, 1999. 5

[16] Shugen Ma and M. Watanabe. Minimum time path-tracking control of redundant manipulators. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 1, pages 27–32 vol.1, 2000. 5

[17] Shugen Ma and M. Watanabe. Minimum-time control of coupled tendon-driven manipulators. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 1, pages 215–220 vol.1, 2000. 5

[18] Shugen Ma and M. Watanabe. Time optimal control of kinematically redundant manipulators with limit heat characteristics of actuators. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 1, pages 152–157 vol.1, 2001. 5

[19] Shugen Ma and Mitsuru Watanabe. Time optimal path-tracking control of kinematically redundant manipulators. *Jsme International Journal Series C-mechanical Systems Machine Elements and Manufacturing - JSME INT J C*, 47:582–590, 06 2004. 5

[20] Bang-Hyun Cho, Byoung-Suck Choi, and Jang-Myung Lee. Time-optimal trajectory planning for a robot system under torque and impulse constraints. In *30th Annual Conference of IEEE Industrial Electronics Society, 2004. IECON 2004*, volume 2, pages 1058–1063 Vol. 2, 2004. 5

[21] L. Zlajpah. On time optimal path control of manipulators with bounded joint velocities and torques. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1572–1577 vol.2, 1996. 5

[22] Daniela Constantinescu and Elizabeth Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified path. *Journal of Robotic Systems - J ROBOTIC SYST*, 17:233–249, 05 2000. 5

[23] G. Pardo-Castellote and R.H. Cannon. Proximate time-optimal algorithm for on-line path parameterization and modification. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1539–1546 vol.2, 1996. 5

[24] Jan Mattmüller and Damian Gisler. Calculating a near time-optimal jerk-constrained trajectory along a specified smooth path. *The International Journal of Advanced Manufacturing Technology*, 45(9):1007, Apr 2009. 5, 23

[25] Z. Shiller and M. Tarkiainen. Time optimal motions of manipulators with actuator dynamics. *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 725–730 vol.2, 1993. 5, 10

[26] Z. Shiller. Time-energy optimal control of articulated systems with geometric path constraints. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2680–2685 vol.4, 1994. 5

[27] Hung Pham and Quang Cuong Pham. On the structure of the time-optimal path parameterization problem with third-order constraints. pages 679–686, 05 2017. 5, 23

[28] Quang Cuong Pham, Stéphane Caron, and Yoshihiko Nakamura. Kinodynamic planning in the configuration space via admissible velocity propagation. 06 2013. 5

[29] Quang Cuong Pham, Stéphane Caron, Puttichai Lertkultanon, and Yoshihiko Nakamura. Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots. *The International Journal of Robotics Research*, 36, 11 2016. 5

[30] Stephen D. Butler, Mark Moll, and L. Kavraki. A general algorithm for time-optimal trajectory generation subject to minimum and maximum constraints. In *WAFR*, 2016. 5

[31] Katta G. Murty and Santosh N. Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39(2):117–129, Jun 1987. 5

[32] Y Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming.* Society for Industrial & Applied Mathematics, U.S., USA, 1st edition, 1994. 5

[33] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009. 6

[34] Diederik Verscheure, Moritz Diehl, Joris Schutter, and Jan Swevers. On-line time-optimal path tracking for robots. pages 599 – 605, 06 2009. 6

[35] Frederik Debrouwere, Wannes Van Loock, Goele Pipeleers, Quoc Tran Dinh, Moritz Diehl, Joris De Schutter, and Jan Swevers. Time-optimal path following for robots with convex–concave constraints using sequential convex programming. *IEEE Transactions on Robotics*, 29(6):1485–1495, 2013. 6

[36] Kris K. Hauser. Fast interpolation and time-optimization on implicit contact submanifolds. In *Robotics: Science and Systems*, 2013. 6

[37] Alessandro Palleschi, Manolo Garabini, Danilo Caporale, and Lucia Pallottino. Time-optimal path tracking for jerk controlled robots. *IEEE Robotics and Automation Letters*, 4(4):3932–3939, 2019. 6

[38] Hung Pham and Quang Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34:645 – 659, 06 2018. 6, 8, 21

[39] Donald E. Kirk. *Optimal control theory: an introduction.* Dover Publications, 1st edition, 2004. 6

[40] Kang Shin and N. McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *IEEE Transactions on Automatic Control*, 31(6):491–500, 1986. 6, 23

[41] S. K. Singh and M. Leu. Optimal trajectory generation for robotic manipulators using dynamic programming. *Journal of Dynamic Systems Measurement and Control-transactions of The Asme*, 109:88–96, 1987. 6

[42] E. Gilbert and D. Johnson. Distance functions and their application to robot path planning in the presence of obstacles. *IEEE Journal on Robotics and Automation*, 1(1):21–30, 1985. 6

[43] J.E. Bobrow. Optimal robot plant planning using the minimum-time criterion. *IEEE Journal on Robotics and Automation*, 4(4):443–450, 1988. 6

[44] Z. Shiller and S. Dubowsky. Global time optimal motions of robotic manipulators in the presence of obstacles. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 370–375 vol.1, 1988. 6

[45] Y. Chen and A.A. Desrochers. Structure of minimum-time control law for robotic manipulators with constrained paths. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 971–976 vol.2, 1989. 7, 33

[46] J. Kieffer, A.J. Cahill, and M.R. James. Robust and accurate time-optimal path-tracking control for robot manipulators. *IEEE Transactions on Robotics and Automation*, 13(6):880–890, 1997. 7

[47] O. Dahl and L. Nielsen. Torque limited path following by on-line trajectory time scaling. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 1122–1128 vol.2, 1989. 7

[48] O. Dahl. Path-constrained robot control with limited torques-experimental evaluation. *IEEE Transactions on Robotics and Automation*, 10(5):658–669, 1994. 7

[49] Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34(3):645–659, 2018. 9, 17

[50] Moshe Niv and David M. Auslander. Optimal control of a robot with obstacles. In *1984 American Control Conference*, pages 280–287, 1984. 12

[51] Shiller Z. Dubowsky S., Norris M.A. Time-optimal robotic manipulator task planning. In *RoManSy 6*, 1987. 12

[52] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* Society for Industrial and Applied Mathematics, USA, 1st edition, 1998. 12

[53] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008. 17, 28

[54] S. Macfarlane and E.A. Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, 19(1):42–52, 2003. 23

[55] Frank Merat. Introduction to robotics: Mechanics and control. *Robotics and Automation, IEEE Journal of*, 3:166 – 166, 05 1987. 23

[56] Matthias Oberherber, Hubert Gattringer, and Andreas Mueller. Successive dynamic programming and subsequent spline optimization for smooth time optimal robot path tracking. *Mechanical Sciences*, 6:245–254, 10 2015. 23

[57] Liang Liu, Chaoying Chen, Xinhua Zhao, and Yangmin Li. Smooth trajectory planning for a parallel manipulator with joint friction and jerk constraints. *International Journal of Control, Automation and Systems*, 14(4):1022–1036, Aug 2016. 23

[58] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 2021/08/07/ 1957. Full publication date: Jul., 1957. 28

[59] J. Denavit. A kinematic notation for lower pair mechanisms based on matrices. 1955. 29