Robotic Motion Planning using Numerical Optimization on Bézier Curves

John Wilhelm

Bachelor of Science
Computer Science and Mathematics
School of Informatics
University of Edinburgh
2020

Abstract

For robots and agents to be able to achieve their goals, they must be able to create plans in their task space, for getting from an arbitrary current configuration to the desired configuration, however, these plans must be carefully crafted in order to that they are physically and dynamically feasible. For example, the plans must avoid making a robot collide with physical objects in the space in which it is operating or making the joints of a robot accelerate faster than they were designed to. In particular, this piece of work will focus on the generation of trajectories which are smooth to the second degree and avoid static regions of physical infeasibility, and the means of generation of these trajectories will be that of numerical optimization. Given the NP-hard nature of non-convex optimization, the optimization problems considered in this paper will be defined on Bézier curves, given that utilizing this class of polynomial spline can remove elements of non-convexity of the problem being investigated. Initially in this work, the duration of a solution trajectory will be heuristically allocated, and with the timing variables fixed, formulations of the problem in the form of three different Quadratic Programms will be derived. Thereafter, given that heuristically allocating the duration of the solution curve is a process completely decoupled from the problem objective of producing smooth curves, two methods will be investigated to allow the timing variables to enter into the numerical optimization. In particular, utilizing a non-convex Trust-Region solver will be investigated along with the methodology of formulating the problem as a Bi-level optimization problem. Quantitative data is presented to show that the Trust-Region method is not suitable for this application.

Acknowledgements

I would like to thank Steve Tonneau for his assistance with the project. I would also like to thank Michael Herrmann for his constructive comments throughout the year.

Table of Contents

Chapter 1

Introduction

1.1 Background and Motivation

Many problems in the field of optimal motion planning for autonomous agents remain open and as such are a keen area of contemporary research [13, 31]. An example motion planning problem would be generating a plan for the robotic arm on the exterior of the International Space Station [14] to move from one configuration to the next, while avoiding regions of intersection of the arm with itself or the main capsule. Another motion planning task would be that of generating a plan for a drone to navigate through an urban area, from a start point to a final point while respecting the constraints of the problem, such as not colliding with buildings and potentially a maximum time constraint to complete the maneuver. Such problems seem particularly distinct on the surface, however both can be reduced to a more abstract form of simply searching continuous curve lying wholly within some arbitrary subset of Euclidean-space, which starts at one point and at some point in time in the future ends at some other predefined point, and perhaps respects a set of constraints (e.g. maximum time for the completion of the trajectory or maximum velocity constraints). The trajectory planning problem involving the robotic arm can be viewed in this form if the abstract space above is simply considered to be the configuration space of the robot, which gets mapped to the problem definition above via forward and inverse dynamics [8]. The search problem involving the drone can viewed directly as the abstract search problem defined above (i.e. the abstract search space is precisely task space of the robot). The interesting nature of this search problem is increased when trajectory generation is endowed with a cost function, that is, a mapping from a solution trajectory to the set of non-negative real numbers which defines an ordering of optimally on solution trajec-

tories. For example, due to the phenomenal monetary cost [14] associated with repairs on the International Space Station, a suitable cost function may be one which encourages smooth, low acceleration trajectories in order to increase the lifetime of the robot. Having a reference trajectory without large changes in velocity in a short period of time is crucial in this case, simply due to the physical constraints on the motors [34, 23]. In the autonomous drone scenario, the cost may be designed to punish trajectories which contain regions of high velocity in order to increase safety. The algorithms outlined in this paper will be designed to solve precisely the problem of searching for the optimal solution trajectory in the arbitrary subset of Euclidean Space, which further respects the constraints of the problem. In particular, in this paper the cost function of interest will be one which smooth, low-acceleration trajectories.

An initial source of motivation for solving this problem is found when one considers the benefits of having robots integrated with society. For example in the case of robotic drones, note that transport by air is clearly faster than transport on land in a congested city environment [26]. The applications of this technology are plentiful, ranging from applications in the medical domain (e.g. delivering defibrillators when rapid action is required) to applications in the retail domain. Another one of the main sources of motivation for solving this problem is embedded in the field of social impact of robots. For example, controlling a multi-joint fixed base robot arm would be would be within the capabilities of the planners presented in this paper. Such technologies could play a role in the creation of high quality prosthetic limbs for humans [16].

A keen angle of angle of attack for generating, feasible (respecting the constraints) near-optimal to optimal trajectories has been formulating the search problem as a numerical optimization problem [30, 21]. The reason for the popularity of turning to mathematical numerical optimization is that such optimization problems are defined in terms of a cost function and a set of constraints which must be satisfied by the solution [22]. Such a framework precisely aligns with the motion planning problem outlined above. One must note that a fundamental fact of optimization theory, is that non-convex optimization is a starkly harder problem than its convex counterpart, in general. The reason for this is that when a local minima is found in a convex problem, it must be the global optimum, by definition. Recent work shows great success for the numerical optimization method of trajectory generation particularly in search spaces with convex constraints, since these problems can be solved efficiently [15, 6]. Note that problems can be defined to be non-convex, and then converted to convex-optimization by relaxing the problem, e.g. via making a linear approximation of a

non-convex cost function. The art defining this topic of research is being able to define an optimization problem, which trades off how sub-optimal a solution is compared to the global optimal and the computation time of the numerical optimization algorithm for solving.

1.2 Summary of Approach

Note that the essential problem to be considered in this paper is to generate a parametric curve, wholly inside some arbitrary subset of Euclidean space, satisfying further constraints and optimally conditions. Given that the subset of Euclidean space is arbitrary, it may be non-convex, which will in turn cause non-convex constraints to enter into the optimization problem. For this reason, the methods presented in this paper will employ a pre-processing step of 'convexifying' the feasible space of the problem as much as possible, in order to decrease the degree of difficulty of the optimization problems. This is a common starting point in the literature for tackling such a problem (e.g. [18]). The obvious cost of such a modification is that the solution produced can no longer be claimed to be the optimal solution with respect to the initial constraints, however, this will reduce compute time, which in reality is usually more important than full optimally.

The methods presented in this paper will have a distinctly geometric flavor, namely through the use of Bézier curves [**?**]. A primer on this class of polynomial spline will be given in subsequent chapters, but two well-established properties of these curves which will be exploited with regularity in this paper. Firstly, after the defining points of a Bézier curve are established, it is easy to establish bounds on the value of of the polynomial on a bounded interval. Constant this to a classic polynomial where there is no formula to obtain bounds on the value of the function based purely on the defining coefficients. Furthermore, the fact that Bézier curves are closed under differentiation will be utilized. Given both of these features, bounds on the derivatives of a Bézier curve to the ith degree can be formulated with ease. A sequence of these Bézier curve will be optimized over, in order to construct the final solution.

A common approach to tackling the problem of trajectory generation is to first assign the amount of time which a each of the segments of the solution trajectory will take, usually based on some distance based heuristic, and then to optimize over the shape of the trajectory. However with this methodology, the timing variables are fully decoupled from the trajectory cost function, which in turn lead to a seriously sub-

optimal solution. Such an assumption will form a starting point for the methods of Bézier trajectory optimization, but promptly, this assumption will be lifted.

1.3 Previous Work

In this report, it has already been introduced that a critical component of any trajectory generation mechanism is to ensure that the constraints of the problem are not significantly violated. One method which has been used to 'ensure' constraints are not violated is to generate a candidate solution curve, and then to check whether it satisfies the constraints stipulated in the problem definition, at only finite number of points as described in [6]. In this paper, details are given of several papers which utilized these methods, e.g. [19, 27], but then progresses to give several adversarial examples to this method of constraint checking, e.g. generating a trajectory through a very thin wall. This is clearly serious flaws with this approach, and encourages this work to rely on mathematical bounds in order to check for constraint satisfaction - the only way in which the constraints can be guaranteed at every time-point. Note that the trade of for using this approach, as will be discussed in future chapters in the case of Bézier Curves, is that often these bounds outline sufficient but not necessary conditions for constraint satisfaction. Thus, these conservative formulation of the constraints may eliminate perfectly valid solution trajectories. However, work from the state-of-the-art (e.g. [35]), appeals to geometric properties of polynomial splines, rather than finite checking, so this indicates that using optimization on Bézier curves is a good direction for further work.

Another starting point for attacking this problem is to utilize a rapidly exploring random trees (RRT) to firstly solve for a solution which trajectory which is possible in terms of the physical constraints of the problem and then use a further optimization to satisfy and improve the dynamic performance, as detailed in [28]. Due to the convenient nature of decoupling of the physical and dynamic constraints of this solution, a similar approach will be appealed to in this dissertation, however, a form of A* search will be utilised in place of using an RRT.

1.4 Thesis Aims and Achievements

The aims of this project were as follows:

1. Analyse the algebraic properties of a smooth-trajectory cost function (provided), and its relationship with Bézier-curves to produce several numerical optimization formulations of how the smooth trajectory problem can be solved.

2. Implement all of the methods derived in Python, appealing to Numerical Optimization and Computer Algebra packages when appropriate.

3. Produce, a clear and fair quantitative analysis of the methods proposed in the paper, where theoretical results are unavailable.

The conclusions of the quantitative analysis were:

1. The Trust-Region method is four orders of magnitude slower than the Bi-level optimization method, when tested in a 3-dimensional problem environment.

2. Formulating the constraints of the problem as soft constraints via the Trust-Region method on average causes on average constraint violation of order of 10^5, when tested in a 3-dimensional problem environment.

3. In a 3-dimensional problem environment, utilizing the Bi-level optimization method over simply heuristically allocating the time for a trajectory is shown on average to reduce the cost of the solution trajectory by more that 53% of the way to global optimally.

All of these aims were achieved, and evidence of such will be provided herein. Note that the images and code examples included in this paper will be of searches in $\mathbb{R}^2$ and $\mathbb{R}^3$, however all of the algorithms outlined in this paper will be applicable to searches in any Euclidean space and further the code in the included with this report is fully general with respect to dimensionality of the search space. The code is available at `https://gitlab.com/SuperWilhelm/Beziertrajectoryplanning`, which I hope can be used for further research into the domain.

Chapter 2

Foundations of the methods

2.1 Formal Problem Definition

The problem of optimal motion planning can be defined succinctly in the following way. Consider a non-empty set $X \subseteq \mathbb{R}^n$ and two points $x_0, x_1 \in X$. Let $\gamma : [0,T] \to X$ denote any continuous curve such that $\gamma(0) = x_0$ to $\gamma(T) = x_1$, $T \in \mathbb{R}$, $T > 0$ and further where γ satisfies a set of constraints S. For the curve γ an example elements of the set S are:

$$T \leq 77 \tag{2.1}$$

$$\gamma\left(\frac{T}{2}\right) = \frac{x_0 + x_1}{4}, \tag{2.2}$$

$$\gamma''(t) < 3, \ \forall t \in [0,T] \tag{2.3}$$

Define the set Γ to be the set containing all of the feasible candidates for γ. The goal of optimal trajectory synthesis is to obtain the element of Γ which is optimal with respect to some cost function $C : \Gamma \to \mathbb{R}$, $C \geq 0$.

This paper will exclusively focus on the classic cost function in robotics the integral of the square norm of some nth derivative of the solution curve [29]. In the case of this paper, the integral of the square of the d_2 norm of the acceleration curve will be the cost function of interest:

$$C(\gamma) = \int_0^T ||\gamma''(t)||^2 dt \tag{2.4}$$

By definition, the above cost function will punish a trajectory which has drastic changes of speed in a short spaces of time. The interest of this cost function is that in robotics smooth trajectories are often very desirable. For example, in the domain of surgical

robots, it would be completely unnatural to observe a robot realizing the commands of the surgeon by means of a sporadic motion trajectory.

2.2 Optimization Basics

As prefaced in the introduction, the motion planning outlined in this paper will be performed via numerical optimization methods. Let $f : A \to B$ be a function and $C \subseteq A$. All optimization problems can be expressed in the following form:

$$\min_x f(x) \tag{2.5}$$

$$s.t.\ x \in C \tag{2.6}$$

The solution to such a problem would simply be a value in $x \in C$ such that $f(x) \leq f(c)$ for all $c \in C$. The convexity or non-convexity of functions and sets will be of vital importance for understanding the methods and results described in this paper, since these properties play a key role in understanding the difficulty of an optimization problem, as well as understanding the different methods of solving optimization problems discussed in this paper.

Definition 1 (Convexity of a function [22])**.** *A function $f : A \to B$ is convex on a set $C \subseteq A$ if and only if for all points $a_1, a_2 \in C$ and $\lambda \in [0,1]$ the following inequality holds: $f(\lambda a_1 + (1-\lambda)a_2) \leq \lambda f(a_1) + (1-\lambda) f(a_2)$.*

For example, the function $f(x) = |x|$ would satisfy the definition convexity, but the function $f(x) = x^3 + 1$ would not. A visual proof of these claims is given in Figure 2.1

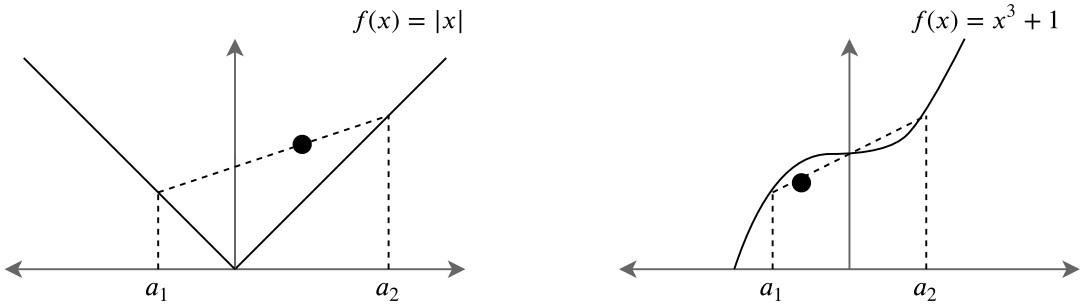

Figure 2.1: The left hand figure gives a visual proof that the function $f(x) = |x|$ is convex and the right had figure gives a visual counterexample to the definition of convexity for the function $f(x) = x^3 + 1$

.

Definition 2 (Convexity of a set [22])**.** *A set A is convex if and only if for all points $a_1, a_2 \in A$ and $\lambda \in [0,1]$ the following set containment holds: $\lambda a_1 + (1-\lambda)a_2 \in A$.*

Given the above definition, note that if A is some Euclidean space, convexity simply means that any straight line starting and ending at points in A, must lie wholly within A.

Definition 3 (Convexity of an optimization problem [22])**.** *Let $f : A \to B$ be a function and $C \subseteq A$. The optimization problem: $\min_x f(x)$ such that $x \in C$ is convex if and only if the set C is convex and the function $f : A \to B$ is convex on C.*

Note that if an optimization problem satisfies the definition of convexity, then it is the case that a local minima of the problem is a global optima.

2.3 Bézier Curves

Bézier Curves are a class of polynomial splines which have beautiful geometric and algebraic properties which play a crucial role alleviating the problem of $X \subseteq \mathbb{R}^n$, in the problem definition, being potentially non-convex. In this chapter, the class of polynomials will be defined, and their properties relevant to this paper will be proven.

Definition 4 (Bernstein Polynomials [11])**.** *The set of $n+1$ Bernstein basis polynomials of degree n each with domain $\mathbb{R}$, are defined as:*

$$b_{v,n}(t) = \binom{n}{v} t^v (1-x)^{n-v}, v \in \{0,..,n\} \tag{2.7}$$

Definition 5 (Bézier Curve [12])**.** *Let $\mathbf{x}_0, ..., \mathbf{x}_k \in \mathbb{R}^n$. The polynomial $\mathbf{B} : [0,1] \to \mathbb{R}^n$, defined by:*

$$\mathbf{B}(t) = \sum_{i=0}^{M} b_{i,n}(t) \cdot \mathbf{x}_i \tag{2.8}$$

is the Bézier Curve of degree k with control points $\mathbf{x}_0, ..., \mathbf{x}_M \in \mathbb{R}^n$.

Note that in the above definition $\mathbf{B}$ is a continuous (given that its simply a polynomial), parametric curve in $\mathbb{R}^n$, which is traversed over the interval, $[0,1]$. Constraining a curve to be traversed in one unit is particularly constraining, however the next definition introduces a re-parameterised generalization of the Bézier Curve.

Definition 6 (Duration parameterised Bézier Curve)**.** *Let* $\mathbf{x}_0, \ldots, \mathbf{x}_k \in \mathbb{R}^n$ *and* $T \in \mathbb{R}$, $T > 0$*. The polynomial* $\mathbf{B}_T : [0, T] \to \mathbb{R}^n$*, defined by:*

$$\mathbf{B}_T(t) = \mathbf{B}\left(\frac{t}{T}\right) \tag{2.9}$$

is the duration parameterized Bézier Curve of degree k with control points $\mathbf{x}_0, \ldots, \mathbf{x}_M \in \mathbb{R}^n$ *and duration T, where B is the Bézier Curve of degree k with control points* $\mathbf{x}_0, \ldots, \mathbf{x}_M \in \mathbb{R}^n$*.*

In order to introduce the first property of the duration parameterized Bézier curve, a further definition must be give. Thereafter, Theorem 1 will introduce and prove the property.

Definition 7. *The Convex-Hull of the points* $\mathbf{x}_0, \ldots, \mathbf{x}_k \in \mathbb{R}^n$ *is the set containing all points* $\mathbf{x}$ *such that there exists* $\lambda_0, \ldots \lambda_M \in [0, 1]$ *such that* $\mathbf{x} = \lambda_0 \mathbf{x}_0 + \ldots + \lambda_k \mathbf{x}_k$ *and* $\sum_{i=0}^{M} \lambda_i = 1$*. Such a sequence of* λ_i*'s is referred to as a convex combination of the* $\mathbf{x}_i$*'s.*

Theorem 1 (Convex-Hull Bound Property)**.** *The duration parameterized Bézier Curve of degree k with control points* $\mathbf{x}_0, \ldots, \mathbf{x}_k \in \mathbb{R}^n$ *and duration* $T \in \mathbb{R}$*,* $T > 0$*, is contained wholly within or on the boundary of* $Convex-Hull(\mathbf{x}_0, \ldots, \mathbf{x}_k)$*.*

Proof of Theorem 7. Let $\mathbf{B}_T$ denote the Bézier curve. Note that for $t \in [0, T]$:

$$\mathbf{B}_T(t) := \mathbf{B}\left(\frac{t}{T}\right) = \sum_{i=0}^{k} b_{i,k}\left(\frac{t}{T}\right)\mathbf{x}_i. \tag{2.10}$$

Clearly, it is true that the $\lambda_i(t) := b_{i,M}\left(\frac{t}{T}\right) \geq 0$ for $t \in [0, T]$, for all i. Additionally note that for all $t \in [0, T]$:

$$\sum_{i=0}^{k} \lambda_i(t) = \sum_{i=0}^{M} b_{i,M}\left(\frac{t}{T}\right) \tag{2.11}$$

$$:= \sum_{i=0}^{k} \binom{k}{i}\left(\frac{t}{T}\right)^i \left(1 - \frac{t}{T}\right)^{k-i} \tag{2.12}$$

$$= \left[\left(1 - \frac{t}{T}\right) + \frac{t}{T}\right]^k = 1, \tag{2.13}$$

by the binomial formula. Hence by definition of the Convex-Hull of a set of points $\mathbf{x}_0, \ldots, \mathbf{x}_M$, the result follows. □

Theorem 1 establishes that a duration parameterized Bézier curve is bounded by the convex hull of its control points. Next, a generalization of the two-dimensional convex polygon in $\mathbb{R}^n$ will be introduced, and then it will be established that, if the control points of a duration parameterized Bézier curve are chosen within this shape in $\mathbb{R}^n$, then the curve defined will lie wholly within the shape.

Theorem 2 (Polytope bound property)**.** *A duration parameterized Bézier curve* $\mathbf{B}_T$ *with control points* $\mathbf{x}_0, ..., \mathbf{x}_k \in \mathbb{R}^n$ *and duration* $T \in \mathbb{R}$*,* $T > 0$*, with control points all contained inside the polytope P, is contained wholly inside the polytope P.*

Proof. Let $t \in [0, T]$ be given. Note that by Theorem 1, the curve $\mathbf{B}_T$ is contained wholly withing the convex hull of its control points, $H := Convex - Hull(\mathbf{x}_0, ..., \mathbf{x}_k)$. Given this, $\mathbf{B}_T(t) \in H$. Then there exists $\lambda_0, ..., \lambda_N \geq 0$, such that $\mathbf{B}_T(t) = \lambda_0\mathbf{x}_0 + ... + \lambda_k\mathbf{x}_k$ and $\sum_{i=0}^{k} \lambda_i = 1$. All that is to show is that $\mathbf{B}_T(t)$ respects all of the linear constraints defining P. Let $a \cdot x \leq b$ with $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ be an arbitrary linear constraint defining P on the point $x \in \mathbb{R}^n$. Given that the control points $\mathbf{x}_0, ..., \mathbf{x}_k \in P$, $a \cdot \mathbf{x}_i \leq b$ for all i. Given this,

$$a \cdot \mathbf{B}_T(t) = a \cdot (\lambda_0\mathbf{x}_0 + ... + \lambda_k\mathbf{x}_k) \tag{2.14}$$

$$= \lambda_0(a \cdot \mathbf{x}_0) + ... + \lambda_k(a \cdot \mathbf{x}_k) \tag{2.15}$$

$$\leq \lambda_0 b + ... + \lambda_k b \tag{2.16}$$

$$= b. \tag{2.17}$$

Therefore, $\mathbf{B}_T$ lies wholly within the polytope P. □

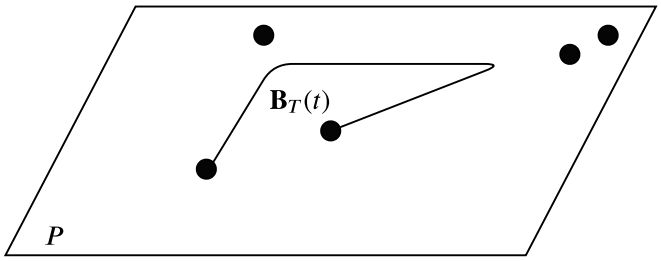

Figure 2.2: Illustration of Theorem 2 in $\mathbb{R}^2$. Simply if the control points of a duration parameterized Bézier curve, shown as black dots in the figure, are selected to be inside of a polytope P, then the parameterized Bézier curve must be contained wholly within P.

.

Theorem 2 above will be used extensively later in this report however, now we will turn our attention to calculating the derivative of a duration parameterized Bézier curve.

Theorem 3 (Derivative of Bézier Curve)**.** *The duration parameterized Bézier Curve of degree k with control points* $\mathbf{x}_0, ..., \mathbf{x}_k \in \mathbb{R}^n$ *and duration* $T \in \mathbb{R}$*,* $T > 0$*, has derivative with respect to t:*

$$\mathbf{B}'_T(t) = \frac{k}{T} \sum_{i=0}^{k-1} \binom{k-1}{i} \left(1 - \frac{t}{T}\right)^{(k-1)-i} \frac{t}{T}^{i} (\mathbf{x}_{i+1} - \mathbf{x}_i) \tag{2.18}$$

Proof. The result follows directly from The Product Rule for differentiation and basic algebraic manipulation. □

From the above result, it can be verified that the differentiation of duration parameterized Bézier curves is closed up to a constant not dependant on the variable of integration, t. To see this note that $\mathbf{B}'_T(t)$ in the above theorem is a duration parameterized Bézier curve, with duration T, degree $k-1$ and control points $\mathbf{x}_1-\mathbf{x}_0,\ldots,\mathbf{x}_k-\mathbf{x}_{k-1}$, multiplied by k/T. Given this it is as easy formulate an expression for the ith derivative of a duration parameterized Bézier curve. For example, an expression for the acceleration of the duration parameterized Bézier Curve of degree k with control points $\mathbf{x}_0,\ldots,\mathbf{x}_k \in \mathbb{R}^n$ and duration $T \in \mathbb{R}$, $T>0$ would be:

$$\mathbf{B}''(t) = \frac{k(k-1)}{T^2}\sum_{i=0}^{k-2}\binom{k-2}{i}\left(1-\frac{t}{T}\right)^{(k-2)-i}\left(\frac{t}{T}\right)^{i}(\mathbf{x}_{i+2}-2\mathbf{x}_{i+1}+\mathbf{x}_i). \tag{2.19}$$

2.4 Constructing the solution curve from a sequence of subcurves

Firstly, a definition of one requirement of any solution curve will be given, and thereafter, a general methodology for constructing a solution curve satisfying this definition will be given. Finally, it will be justified that the parameterization of such a solution curve is convex.

Definition 8. *A parametric curve $f:[0,T]\to\mathbb{R}^n$ is said to respect the physical constraints of the set $X\subseteq\mathbb{R}^n$ if $\mathfrak{I}(f)\subseteq X$.*

As described in Section 2.1, the goal of the methods being outlined in this paper is to connect two points $x_0,x_1\in X$ with a continuous curve $\gamma\colon[0,T]\to X$ such that the solution curve satisfies a set of constraints S. Recall that the set $X\subseteq\mathbb{R}^n$ is arbitrary in terms of convexity, by definition. The awkwardness of the potential non-convexity of the set X can be removed by instead of searching for a motion trajectory γ which lies within the potentially non-convex set X, if firstly a sequence of polytopes, $Q_0,\ldots,Q_{M-1}\subseteq X$ can be found with the property that $x_0\in Q_0$, $x_1\in Q_{M-1}$ and $Q_{\Delta i+1}\coloneqq Q_i\cap Q_{i+1}\neq\varnothing$. Then the solution curve γ can be constructed from a sequence of sub-curves $\gamma_0,\ldots,\gamma_{M-1}$ where γ_0 starts at x_0, γ_{M-1} ends at the point x_1 and γ_i ends where γ_{i+1} starts, inside the polytope $Q_{\Delta i+1}$. The usefullness of this construction is realised when $\gamma_0,\ldots,\gamma_{M-1}$ are

considered to be duration parameterized Bézier curves, the curve γ_i can be ensured to be contained within Q_i, by selecting the control points of the curve to be inside the polytope Q_i, given Theorem 2. Hence, selecting the control points of the γ_i's to have this property ensures γ respects the physical constraints of X.

In addition to ensuring that the solution curve satisfies the physical constraints of X, since polytopes can be verified to be convex, by noting that half-planes are convex and that the intersection of convex sets is convex, and since the duration of γ_i is selected from $\mathbb{R}_{>0}$, an interval in $\mathbb{R}$, the parameterization of γ_i is convex.

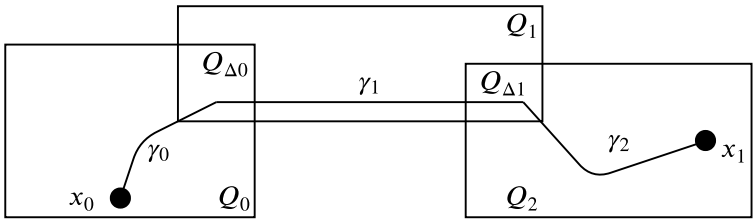

Figure 2.3: Illustration in $\mathbb{R}^2$ of notation used for constructing the solution curve γ from the polytope sequence $Q_0, Q_1, Q_2 \subseteq X$.

.

2.5 Obtaining the polytope sequence

Recall that in the previous section, sequence of polytopes, $Q_0, ..., Q_{M-1} \subseteq X$ with the property that $x_0 \in Q_0$, $x_1 \in Q_{M-1}$ and $Q_{\Delta i+1} := Q_i \cap Q_{i+1} \neq \varnothing$ was assumed as given. This section will outline the construction of such a sequence of polytopes.

Firstly, a set of polytopes $\tilde{Q} = \{\tilde{Q}_0, ...\tilde{Q}_k\}$ with each $\tilde{Q}_i \in \tilde{Q}$ satisfying $\tilde{Q}_i \subseteq X$ will be assumed as given. The construction of this set of polytopes $\tilde{Q}$ has been a keen topic of research in the literature (e.g. [5]), this construction will be considered out of the scope of this particular project. However, the utilization of this set will now be outlined.

Firstly, the intersection of all pairs of distinct polytopes will be considered. For example, consider $\{A, B\} \in \{\{A, B\} : A, B \in \tilde{Q}\}$. The polytopes A and B are defined by a set of half planes. Let $\tilde{A}x \leq a$ denote such linear constraints defining A and $\tilde{B}x \leq b$ similarly define the constraints defining B for vectors $x \in \mathbb{R}^n$. Clearly $x \in \mathbb{R}^n$, if and only if $\tilde{A}x \leq a$ and $\tilde{B}x \leq b$. In order to establish if there exists such an x, all subsets of the union of both of the linear constraints of size n can be enumerated. Then for each of sets of n linear inequations, we can attempt to solve for a unique solution to the corresponding set of linear equations. If one such subset firstly has a unique solution

and secondly the solution satisfies the linear inequalitues $\tilde{A}x \leq a$ and $\tilde{B}x \leq b$, then the intersection must be non-trivial. Otherwise, the intersection is assumed to be trivial. This is the method utilised in this paper, however it is only since, it will be assumed that all polytopes will be finite. However, this problem can be solved in general by noting that the Linear Program [22], defined as: $\min_x,\ \pi$ such that $\tilde{A}x \leq a$ and $\tilde{B}x \leq b$, will return a solution if and only if, the intersection of A and B is non-trivial.

An undirected graph G is constructed with $\tilde{Q}$ as the vertices and an edge is added to the graph between $A, B \in \tilde{Q}$ if and only if $A \cap B \neq \varnothing$ and $A \neq B$. The Euclidean distance between the 'center' of A, i.e. the average value of the vertives of A, and the center of B will be associated to this edge if included in G. An A^*-search can be performed over the graph G with the admissible heuristic d_2, to find a sequence of polytopes, $Q_0, ..., Q_{M-1} \subseteq X$ with the desired properties, if such a sequence exists.

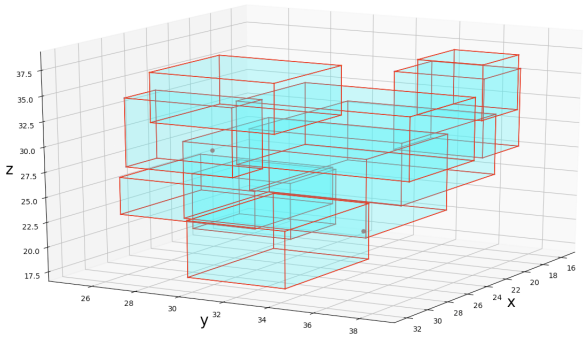

Figure 2.4: A random set $\tilde{Q}$ in $\mathbb{R}^3$, generated by the random generator for problems in $\mathbb{R}^3$ which I implemented. I also implemented classes defining Polytopes and linear constraints in $\mathbb{R}^n$. Note that the red dots denote the start and endpoints of the problem.

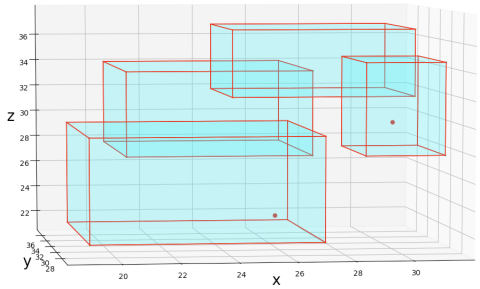

Figure 2.5: The resulting Q_0, Q_1, Q_2, Q_3, which the A^* search which I implemented found.

Chapter 3

Formulation of cost function and dynamic constraints on Bézier Curves

In this chapter details will be given of a convenient algebraic form of the cost function C, when the the solution curve γ is constructed from a finite sequence of duration parameterised Bézier curve. In addition to this, it will also be described how theory of time parameterized Beizer curves, as discussed in the previous chapter, can be utilised in order to ensure that a soltuion curve can be relized by accurately by a controller.

3.1 Cost Function

Again denote $\gamma_0, ..., \gamma_{M-1}$ be a sequence of duration parameterized Bézier curves of degree d in $\mathbb{R}^n$, where each γ_i is defined by the tuple of control points $x^{(i)} = (x_0^{(i)}, ..., x_d^{(i)})$ where each $x_\cdot^{(\cdot)} \in \mathbb{R}^n$ and a time allocation $t_i \in \mathbb{R}$, $t_i > 0$. Define

$$x^{(i)} = \left[{x_0^{(i)}}^T, {x_1^{(i)}}^T, ..., {x_d^{(i)}}^T \right]^T, \tag{3.1}$$

$$x = \left[{x^{(0)}}^T, {x^{(1)}}^T, ..., {x^{(M)}}^T \right]^T, \tag{3.2}$$

$$t = \left[t_0, ..., t_{M-1} \right]^T. \tag{3.3}$$

By the above definitions, $x^{(i)}$ is simply the control points of the ith duration parameterized Bézier curve stacked into a column vector and x is simply a column vector of all of the control points for all of the duration parameterized Bézier curves, and finally t can be described as a vector containing all of the time allocations.

Recall that the cost function which is being utilized in this paper is:

$$C(x,t) = \sum_{i=0}^{M-1} \int_0^{t_i} \|a_i(t')\|^2 dt', \tag{3.4}$$

where a_i is the function giving the acceleration vector of γ_i.

Further define C_i to simply be the cost associated with the ith duration parameterized Bézier curve, e.g.

$$C_i(x^{(i)},t_i) = \int_0^{t_i} \|a_i(t')\|^2 dt'. \tag{3.5}$$

I was able to observe the following result after my supervisor telling me to try to establish a closed form equivalence to C, which separates the state variables x and the timing variables t. I believe this to have been previously shown in the literature, e.g. [9], however, I did not utilize this when obtaining the following form of the cost function C.

Theorem 4. *The function C can be expressed in the form:*

$$C(x,t) = \frac{1}{2}x^T A(t)x + x^T b(t) + c(t),\ A : \mathbb{R}^M \to \mathbb{R}^{(n\cdot d\cdot M)\times(n\cdot d\cdot M)} \tag{3.6}$$

$$b : \mathbb{R}^M \to \mathbb{R}^{(n\cdot d\cdot M)} \tag{3.7}$$

$$c : \mathbb{R}^M \to \mathbb{R}. \tag{3.8}$$

Moreover, expressions for A, b and c can be obtained in closed form.

Proof. Given that C can be expressed in the form,

$$C(x,t) = \sum_{i=0}^{M-1} C_i(x^{(i)},t_i) \tag{3.9}$$

and further given that $x = \left[{x^{(0)}}^T, {x^{(1)}}^T, ..., {x^{(M)}}^T\right]^T$ the theorem must be considered true if it can be shown that the cost of the ith duration parameterised Bézier curve, γ_i can be expressed in the form:

$$C_i(x^{(i)},t_i) = \frac{1}{2}x^T A_i(t_i)x + x^T b_i(t_i) + c_i(t_i),\ A_i : \mathbb{R} \to \mathbb{R}^{(n\cdot d)\times(n\cdot d)} \tag{3.10}$$

$$b_i : \mathbb{R} \to \mathbb{R}^{(n\cdot d)} \tag{3.11}$$

$$c_i : \mathbb{R} \to \mathbb{R}. \tag{3.12}$$

This is because given the addative property of the cost function C given in Equation 3.9 and the definition of x, the following function definitions for A, b and c satisfy Equality 3.6[1]:

$$A(t) := diag(A_0(t_0), ..., A_{M-1}(t_{M-1})), \tag{3.13}$$

$$b(t) := [b_0(t_0)^T, ..., b_{M-1}(t_{M-1})^T]^T, \tag{3.14}$$

$$c(t) := c_0(t_0) + ... + c_{M-1}(t_{M-1}). \tag{3.15}$$

Recall that the ith curve, γ_i, is defined to have cost:

$$C_i(x^{(i)}, t_i) = \int_0^{t_i} ||a_i(t)||^2 dt. \tag{3.16}$$

The Bézier curve for this section of the solution can be expressed explicitly as:

$$\gamma_i(t) = \sum_{j=0}^{d} b_{j,d}\left(\frac{t}{t_i}\right) x_j^{(i)},\ t \in [0, t_i] \tag{3.17}$$

Which can be differentiated (via chain rule) with respect to t to obtain:

$$\gamma_i'(t) = \frac{d}{t_i} \sum_{j=0}^{d-1} b_{j,d-1}\left(\frac{t}{t_i}\right) (x_{j+1}^{(i)} - x_j^{(i)}),\ t \in [0, t_i] \tag{3.18}$$

This expression can be differentiated again to obtain a function returning the acceleration vector of γ_i:

$$a_i(t) := \gamma_i''(t) = \frac{d(d-1)}{{t_i}^2} \sum_{j=0}^{d-2} b_{j,d-2}\left(\frac{t}{t_i}\right) (x_{j+2}^{(i)} - 2x_{j+1}^{(i)} + x_j^{(i)}),\ t \in [0, t_i]. \tag{3.19}$$

As next the norm of the above vector function will be taken, it will be useful to have the above vector function in element-wise form. Recall that the number of dimensions is n. Denote the acceleration in the ith curve in the rth direction as $a_i^r(t)$, and further let $x_{j,r}^{(i)}$ denote the jth control point, of the ith curve in the rth dimension. Now the acceleration in the rth dimension of the ith curve can be expressed as:

[1] To verify the above sub-claim, note that:

$$\left(\frac{1}{2}x^T A x + b^T x + c\right) + \left(\frac{1}{2}y^T P y + q^T y + r\right) = \frac{1}{2}\tilde{x}^T diag(A, P)\tilde{x} + \tilde{x}^T [b^T, q^T]^T + c + r,\ where\ \tilde{x} = [x^T, y^T]^T$$

and further note that this result can be shown in general by induction.

$$a_i^r(t) := \frac{d(d-1)}{{t_i}^2} \sum_{j=0}^{d-2} b_{j,d-2}\left(\frac{t}{t_i}\right)(x_{j+2,r}^{(i)} - 2x_{j+1,r}^{(i)} + x_{j,r}^{(i)}),\ t \in [0,t_i]. \tag{3.20}$$

Note that now we can express $||a_i(t)||^2$ as:

$$||a_i(t)||^2 = \sum_{r=1}^{n} a_i^r(t)^2 \tag{3.21}$$

As to make progress in obtaining an C_i in quadratic form, an expanded expression describing $a_i^r(t)^2$ must be found. By definition:

$$a_i^r(t)^2 = \left(\frac{d(d-1)}{{t_i}^2} \sum_{j=0}^{d-2} b_{j,d-2}\left(\frac{t}{t_i}\right)(x_{j+2,r}^{(i)} - 2x_{j+1,r}^{(i)} + x_{j,r}^{(i)})\right)^2, t \in [0,t_i]. \tag{3.22}$$

$$= \frac{d^2(d-1)^2}{{t_i}^4} \left(\sum_{j=0}^{d-2} b_{j,d-2}\left(\frac{t}{t_i}\right)(x_{j+2,r}^{(i)} - 2x_{j+1,r}^{(i)} + x_{j,r}^{(i)})\right)^2, t \in [0,t_i] \tag{3.23}$$

Recall the standard result for real numbers z_i that:

$$\left(\sum_{i=0}^{N} z_i\right)^2 = \left(\sum_{i=0}^{N} z_i^2\right) + 2\left(\sum_{i<j}^{N} z_i z_j\right) \tag{3.24}$$

This can be directly applied to Equation 3.23 to give that:

$$a_i^r(t)^2 = \frac{d^2(d-1)^2}{{t_i}^4}\left(\left(\sum_{j=0}^{d-2} b_{j,d-2}\left(\frac{t}{t_i}\right)^2 \left(x_{j+2,r}^{(i)} - 2x_{j+1,r}^{(i)} + x_{j,r}^{(i)}\right)^2\right) +$$

$$2\left(\sum_{j<l}^{d-2} b_{j,d-2}\left(\frac{t}{t_i}\right) b_{l,d-2}\left(\frac{t}{t_i}\right)\left(x_{j+2,r}^{(i)} - 2x_{j+1,r}^{(i)} + x_{j,r}^{(i)}\right)\left(x_{l+2,r}^{(i)} - 2x_{l+1,r}^{(i)} + x_{l,r}^{(i)}\right)\right)\right) \tag{3.25}$$

Recalling that the real quantity of interest is the term,

$$C_i(x^{(i)}, t_i) = \int_0^{t_i} ||a_i(t)||^2 dt = \int_0^{t_i} \sum_{r=1}^{n} a_i^r(t)^2 dt = \sum_{r=1}^{n} \int_0^{t_i} a_i^r(t)^2 dt, \tag{3.26}$$

one should attempt to find an expression for:

$$\int_0^{t_i} a_i^r(t)^2 dt \tag{3.27}$$

Given the above expansion for $a_i^r(t)^2$ in Equation 3.25, an expression for this integral can be obtained:

$$\int_0^{t_i} a_i^r(t)^2 dt = \frac{d^2(d-1)^2}{{t_i}^4}\left(\left(\sum_{j=0}^{d-2}\left[\int_0^{t_i} b_{j,d-2}\left(\frac{t}{t_i}\right)^2 dt\right]\left(x_{j+2,r}^{(i)} - 2x_{j+1,r}^{(i)} + x_{j,r}^{(i)}\right)^2\right) + \right.$$

$$\left.2\left(\sum_{j<l}^{d-2}\left[\int_0^{t_i} b_{j,d-2}\left(\frac{t}{t_i}\right) b_{l,d-2}\left(\frac{t}{t_i}\right) dt\right]\left(x_{j+2,r}^{(i)} - 2x_{j+1,r}^{(i)} + x_{j,r}^{(i)}\right)\left(x_{l+2,r}^{(i)} - 2x_{l+1,r}^{(i)} + x_{l,r}^{(i)}\right)\right)\right) \tag{3.28}$$

Note that the above equation, only bi-linear in $x(i)$ will occur. Given that Equation 3.26 describes that C_i is the finite sum over r of the above equation, the cost function C_i is quadratic in $x^{(i)}$ as required.

In order to complete the proof, it must be shown that A, b and c can be attained in closed form. To show this, all that is required is to find a closed form solution to the integrals in the above equation. The general form of these integrals are:

$$\int_0^{t_i} b_{j,d-2}\left(\frac{t}{t_i}\right) b_{l,d-2}\left(\frac{t}{t_i}\right) dt,\ j,l \in \{0,..,d-2\} \tag{3.29}$$

A closed form solution to the above integral can easily be obtained in closed form. To this end, note that substituting the Bernstein polynomials for their definition and removing constants with respect to the variable of integration derives the following equivalence:

$$\int_0^{t_i} b_{j,d-2}\left(\frac{t}{t_i}\right) b_{l,d-2}\left(\frac{t}{t_i}\right) dt \tag{3.30}$$

$$= \int_0^{t_i} \binom{d-2}{j}\left(\frac{t}{t_i}\right)^j \left(1-\frac{t}{t_i}\right)^{d-2-j} \binom{d-2}{l}\left(\frac{t}{t_i}\right)^l \left(1-\frac{t}{t_i}\right)^{d-2-l} dt \tag{3.31}$$

$$= \binom{d-2}{j}\binom{d-2}{l} \int_0^{t_i} \left(\frac{t}{t_i}\right)^{j+l} \left(1-\frac{t}{t_i}\right)^{(2d-4)-(j+l)} dt \tag{3.32}$$

Note that the integrand in the final equation is very nearly a Bernstein polynomial of degree $2d-4$, with only a constant scaling being the difference. This constant can be introduced into the integral in the following way:

$$\binom{d-2}{j}\binom{d-2}{l} \int_0^{t_i} \left(\frac{t}{t_i}\right)^{j+l} \left(1-\frac{t}{t_i}\right)^{(2d-4)-(j+l)} dt \tag{3.33}$$

$$= \frac{\binom{d-2}{j}\binom{d-2}{l}}{\binom{(2d-4)-(j+l)}{j+l}} \int_0^{t_i} \binom{(2d-4)-(j+l)}{j+l}\left(\frac{t}{t_i}\right)^{j+l} \left(1-\frac{t}{t_i}\right)^{(2d-4)-(j+l)} dt \tag{3.34}$$

$$= \frac{\binom{d-2}{j}\binom{d-2}{l}}{\binom{(2d-4)-(j+l)}{j+l}} \int_0^{t_i} b_{j+l,2d-4}\left(\frac{t}{t_i}\right) dt \tag{3.35}$$

To compute the above integral consider a u-substitution with $u = \frac{t}{t_i}$ where the relevant derivative of u can be calculated to be $\frac{du}{dt} = \frac{1}{t_i}$. Thus,

$$\frac{\binom{d-2}{j}\binom{d-2}{l}}{\binom{2d-4-(j+l)}{j+l}} \int_0^{t_i} b_{j+l,2d-4}\left(\frac{t}{t_i}\right) dt \tag{3.36}$$

$$= \frac{\binom{d-2}{j}\binom{d-2}{l}}{\binom{2d-4-(j+l)}{j+l}} t_i \int_0^1 b_{j+l,2d-4}(u)\, du \tag{3.37}$$

By standard theory for Bernstein Polynomials, the value for the above integral on the unit interval is:

$$\frac{1}{(2d-4)+1} = \frac{1}{2d-3} \tag{3.38}$$

Given that it has been justified that C is quadratic in x and that such a quadratic form is obtainable in closed form, this proof is complete. □

3.2 Dynamic constraints

As previously mentioned, for any motion planner to be considered to be practical for use in real applications, it must be able to on request, ensure that the resulting plans satisfy certain conditions, which make them compatible with the dynamic constraints of the robot utilizing the planner. For example, a maximum velocity and acceleration at all time points on the curve, may be requested due to limitations of the control algorithms realizing the solution trajectories. Similarly, it may be requested that the motion plans produced from a planning algorithm are nth order continuous (i.e. the function is continuous along with it's the first n derivatives). The rationale for requesting such continuity constraints is again that the controller may be unable to execute the planned trajectory, and hence the trajectory executed by the robot may then be dangerously different form the solution trajectory produced by the planner. Below, a definition to refer to such constraints will be given and thereafter, a description of the formulation of the maximum velocity and acceleration constraints will be given, followed by a description of how nth order continuity will be enforced between two Bézier curves.

Definition 9. *A parametric curve $f : [0,T] \to \mathbb{R}^n$ is said to respect the dynamic constraints of a robot and its controller if it is sufficently smooth and has sufficiently bounded derivatives as dictated by the robot and its controller's limitations.*

3.2.1 Formulating continuity constraints

I was able to formulate continuity constraints for this problem, simply by appealing to the definition of duration parameterised Bézier Curves.

Let γ_0, γ_1 be a sequence of two dth degree duration parameterized Bézier curves, with control points $x_0^{(0)}, ..., x_m^{(0)}$ and $x_0^{(1)}, ..., x_m^{(1)}$, and time allocations t_0 and t_1 respectively. Further define γ to be the concatination of the curve γ_0 and γ_1. It will now be shown how to formulate constraints, to ensure that γ is nth order continuous. Note the formulation described below is naturally generalizable to trajectories γ constructed from a sequence of M duration parameterized Bézier curves.

Firstly note that since the curves γ_0 and γ_1 are simply polynomial splines, they are infinitely smooth. Hence the curve γ will be smooth at almost all points, with the exception points being the positions where the curves γ_0 and γ_1 start and terminate. To ensure that γ is nth order continuous, constraints representing nth order continuity of the curve at these points can be explicitly added to the set of constraints in the

optimization problem.

To see the formulation, firstly recall the explicit definition of the curves γ_0 and γ_1:

$$\gamma_0(t) = \sum_{i=0}^{d} b_{i,d}\left(\frac{t}{t_0}\right) x_i^{(0)}, t \in [0, t_0], \tag{3.39}$$

$$\gamma_1(t) = \sum_{i=0}^{d} b_{i,d}\left(\frac{t}{t_1}\right) x_i^{(1)}, t \in [0, t_1]. \tag{3.40}$$

It can be verified from the definition of the Bernstein Basis Polynomials the following identities follow,

$$\gamma_0(0) = x_0^{(0)},\ \gamma_0(t_0) = x_d^{(0)}, \tag{3.41}$$

$$\gamma_1(0) = x_0^{(1)},\ \gamma_1(t_1) = x_d^{(1)} \tag{3.42}$$

Ensuring (0th order) continuity at the 'hand-over' point of γ_0 and γ_1 is precisely to have $\gamma_0(t_0) = \gamma_1(0)$. Hence, by the above identities, this can be enforced simply by having the constraint that:

$$x_d^{(0)} = x_0^{(1)}. \tag{3.43}$$

Further to this, given that the problems of interest in this paper are defined to have start point x_0 and endpoint x_1, these constraints can be satisfied by ensuring that:

$$x_0^{(0)} = x_0 \text{ and } x_d^{(1)} = x_1. \tag{3.44}$$

In addition to zeroth order continuity, it may be a requirement of the robot and its controller to be continuous to the first order. Recall that the expression for the velocity of a duration parameterized Bézier curves γ_0 and γ_1 are given by:

$$\gamma_0'(t) = \frac{d}{t_0} \sum_{i=0}^{d-1} b_{i,d-1}\left(\frac{t}{t_0}\right)(x_{i+1}^{(0)} - x_i^{(0)}), t \in [0, t_0], \tag{3.45}$$

$$\gamma_1'(t) = \frac{d}{t_1} \sum_{i=0}^{d-1} b_{i,d-1}\left(\frac{t}{t_1}\right)(x_{i+1}^{(1)} - x_i^{(1)}), t \in [0, t_1]. \tag{3.46}$$

Again from the definition of the Bernstein Basis Polynomials the following identities follow,

$$\gamma_0'(0) = \frac{d}{t_0}(x_1^{(0)} - x_0^{(0)}),\ \gamma_0'(t_0) = \frac{d}{t_0}(x_d^{(0)} - x_{d-1}^{(0)}), \tag{3.47}$$

$$\gamma_1'(0) = \frac{d}{t_1}(x_1^{(1)} - x_0^{(1)}),\ \gamma_1'(t_1) = \frac{d}{t_1}(x_d^{(1)} - x_{d-1}^{(1)}). \tag{3.48}$$

Analogously to the zeroth order case above, ensuring first order continuity at the 'hand-over' point of γ_0 and γ_1 is precisely to have $\gamma_0'(t_0) = \gamma_1'(0)$. Hence given the identities

3.47 and 3.48 first order continuity at this point can be enforced by ensuring that the following holds:

$$\frac{d}{t_0}(x_d^{(0)} - x_{d-1}^{(0)}) = \frac{d}{t_1}(x_1^{(1)} - x_0^{(1)}). \tag{3.49}$$

Furthermore, analogous to the zeroth order constraint that the curve γ must start at the defined start and end point of the problem, it may also be desired that the curve γ has zero velocity at its endpoints. That is to say, $\gamma_0'(0) = 0$ and $\gamma_1'(t_!) = 0$ and hence it is therefore required that:

$$\frac{d}{t_0}(x_1^{(0)} - x_0^{(0)}) = 0 \text{ and } \frac{d}{t_1}(x_d^{(1)} - x_{d-1}^{(1)}) = 0. \tag{3.50}$$

At this stage, a general formula for obtaining *n*th order continuity constraints can be observed, and so the derivation will be shown to no further degree, however, the second order constraints will be be stated below, given that they will be utilized in this paper. That is, ensuring continuity in the second order at the 'hand-over' point of γ_0 and γ_1, can be formulated as the following constraint:

$$\frac{d(d-1)}{t_0^2}(x_d^{(0)} - 2x_{d-1}^{(0)} + x_{d-2}^{(0)}) = \frac{d(d-1)}{t_1^2}(x_2^{(1)} - 2x_1^{(1)} + x_0^{(1)}). \tag{3.51}$$

Further to this, it can be ensured that γ has an acceleration of zero at its endpoints by ensuring that

$$\frac{d(d-1)}{t_0^2}(x_2^{(0)} - 2x_1^{(0)} + x_0^{(0)}) = 0 \text{ and } \frac{d(d-1)}{t_1^2}(x_d^{(1)} - 2x_{d-1}^{(1)} + 2x_{d-2}^{(1)}) = 0. \tag{3.52}$$

3.2.2 Formulating maximum velocity and acceleration constraints

I was able to formulate constraints on the maximum velocity and acceleration of the solution curve in each of the dimensions after firstly, trying to formulate exact constraints, but then after a discussion with my supervisor, I was guided to use the results of Theorem 2 and Theorem 3, in order to formulate a conservative bound, i.e. a sufficient but not necessary condition for the derivatives to be bounded in magnitude as required. The formulation and its justification will be given below. Note that the below methodoloty is easily generalizable to a sequence of *M*, duration parameterized Bézier curves.

Let γ_0 be a duration parameterized Bézier curve with control points $x_0^{(0)},...,x_m^{(0)}$ and duration t_0. Recall from Equation 3.45 that the velocity of this curve at timepoint t is given by:

$$\gamma_0'(t) = \frac{d}{t_0}\sum_{i=0}^{d-1} b_{i,d-1}\left(\frac{t}{t_0}\right)(x_{i+1}^{(0)} - x_i^{(0)}), t \in [0, t_0] \tag{3.53}$$

Now suppose that a maximum value for the velocity of the solution curve is given as $v_{max} \in \mathbb{R}_+$, and further note that Equation 3.53 is simply a duration parameterized Bézier curve. To see this note that γ_0' can be written in the following form:

$$\gamma_0'(t) = \sum_{i=0}^{d-1} b_{i,d-1}\left(\frac{t}{t_0}\right)a_i,\ t \in [0, t_0], \tag{3.54}$$

$$\text{where } a_i = \frac{d}{t_0}(x_{i+1}^{(0)} - x_i^{(0)}),\ i \in \{0,...,d-1\}. \tag{3.55}$$

Recall from Theorem 2, that to bound a duration parameterized Bézier curve inside a polytope is sufficient to select the control points of the curve inside the desired bounding polytope. In particular, in order to bound the velocity of γ_0 by v_{max}, the control points of its derivative curve, e.g. the a_i's should be selected within the nth dimensional hyper-cube centered at the origin of $\mathbb{R}^n$, with side lengths $2 \cdot v_{max}$.

Following analogous reasoning, acceleration of the curve γ_0 can be bounded by the value a_{max} that the control points of the curve describing its second derivative, e.g. the curve:

$$\gamma_0''(t) = \sum_{i=0}^{d-2} b_{i,d-2}\left(\frac{t}{t_0}\right)a_i,\ t \in [0, t_0], \tag{3.56}$$

$$\text{where } a_i = \frac{d(d-1)}{t_0^2}(x_{i+2}^{(0)} - 2x_{i+1}^{(0)} + x_i^{(0)}),\ i \in \{0,...,d-2\}. \tag{3.57}$$

are selected within the nth dimensional hyper-cube centered at the origin of $\mathbb{R}^n$, with side lengths $2 \cdot a_{max}$.

After formulating the constraints in this section, I added code to generate these constraints to my Python implementation.

Chapter 4

Heuristically Allocated time Methods

Detailed previously in this paper was a method for constructing a sequence of polytopes with the property that there exists a continuous curve contained wholly within this sequence connecting the start point of a search with the end point. Recall such a sequence was denoted as $Q_0, ..., Q_{M-1}$. Recall further that the intersection of two adjacent polytopes say Q_i and Q_{i+1} was defined as $Q_{\Delta i+1}$. At this stage the only question which is left to answer is that of how to utilise this sequence in order to produce an optimal trajectory, which respects any set of the dynamic constraints outlined in the previous chapter.

The pipeline for the motion planner presented in this paper will 'branch' at this point of the algorithm, and the interest of this chapter will be to outline 3 different methods for generating the final optimal solution. The quality characterising the methods in this chapter will be that the duration of each of the sub-curves used for constructing the final curve, i.e. the γ_i's, will be heuristically allocated before any numerical optimization takes place. In the next chapter, the duration of each of the sub-curves, γ_i will enter into the numerical optimization. Herein, let $\tilde{t} \in \mathbb{R}^M$ be a positive vector containing the heuristically allocated duration value for each of the sub-curves.

An example of how one could calculate such a $\tilde{t}$ will be given, after the following definition.

Definition 10. *A knot-point sequence is a sequence of points, $q_{\Delta 0}, ...q_{\Delta M}$ satisfying the properties that $q_{\Delta 0} = x_0$ the start point of the search, $q_{\Delta M} = x_1$ is the end point of the search and $q_{\Delta i} \in Q_{\Delta i}$, for all $i \in \{1, .., M-1\}$.*

Given the above definition, an example calculation of $\tilde{t}$ would be to take the knot-point sequence defined such that all of the free variables are at the center of the polytope which they are defined to be inside, i.e. at the average value of the vertices. Then

one could compute the distance between each of the adjacent points in the sequence and define a vector $\tilde{t}$ based on an average velocity assumption.

The motivation for allocating the variable t to a fixed value is that is the cost function and the constraints derived in the previous chapter take on a much easier form to work with in conjunction with numerical optimization. In particular, the problem which this thesis is trying to address can be formulated as various types of Quadratic Programs [22]. Note that the form of a Quadratic Program is as follows:

$$\min_{y} \frac{1}{2} y^T Q y + r^T y + s \tag{4.1}$$

$$s.t.\ \ Ay = b, \tag{4.2}$$

$$Dy \leq f, \tag{4.3}$$

$$y \in \mathbb{R}^q. \tag{4.4}$$

In the above formulation, $Q \in \mathbb{R}^{q \times q}$, $r \in \mathbb{R}^q$, $s \in \mathbb{R}$, $A \in \mathbb{R}^{p \times q}$, $b \in \mathbb{R}^p$, $D \in \mathbb{R}^{w \times q}$, $f \in \mathbb{R}^w$. There are two particularly useful properties of optimization problems which can be formulated in this was. Firstly, if there is one vector $y \in \mathbb{R}^q$ satisfying the both of the linear constraints 2.4 and 4.3, then there exists and optimal solution in $\mathbb{R}^q$ to the problem. Moreover, the second useful property of problems which can be formulated in this structure is that, if there is an optimal solution, then it can be solved for efficiently, by means of the Interior Points Method (IMP) [24].

The next three subsections will present three different formulations of the quadratic optimization problem. The first is an naive formulation which under utilizes the algebraic form of the cost function and constraints described above. The second formulation drops these over constraining assumptions and the third implicitly solves for the sub-curves, $\gamma_0, ..., \gamma_{M-1}$ via a beautifully geometric argument.

4.1 Naive Quadratic Programming inside each polytope

This section will detail the first formulation of the problem which I constructed. To this end, the method presented in this section will generate the curves $\gamma_0, ..., \gamma_{M-1}$ using M distinct quadratic optimizations, which will be concatenated to produce the solution curve γ. Assume that the time allocation vector $\tilde{t} \in \mathbb{R}^M$ is given as a result of some heuristic calculation (e.g. time to complete section assuming constant velocity). Recall further that the duration parameterized Bézier curve γ_i is to start at $q_{\Delta i}$ and end at the point $q_{\Delta i+1}$ for some knot-point sequence $q_{\Delta_0}, ..., q_{\Delta_M}$. However note that by definition

of a knot-point sequence, $q_{\Delta 0}, ..., q_{\Delta M}$ only the first value and last value in this sequence are defined to be fixed values. That is, the values $q_{\Delta 1}, ..., q_{\Delta M-1}$ are only defined thus far such that $q_{\Delta i} \in Q_{\Delta i}$. At this point, the strong assumption will be made that shorter curves will in turn incur lower costs.

4.1.1 Fixing the Knot-point sequence to minimize length of solution

A method must be established to solve for $q_{\Delta 1}, ..., q_{\Delta M-1}$ such that the minimize the length of the polyline through the sequence of points $q_{\Delta 0}, ..., q_{\Delta M}$, where length is defined via the d_2 distance metric. That is the cost function of the optimization problem is:

$$Cost(q_{\Delta 1}, ..., q_{\Delta M-1}) = \sum_{i=0}^{M-1} d_2(q_{\Delta i}, q_{\Delta i+1}) \tag{4.5}$$

Note that the above function is convex since the d_2 metric is convex and the sum of convex functions is always convex. Note further that all of the variables of optimization $q_{\Delta 1}, ..., q_{\Delta M-1}$ $q_{\Delta i}$ are constrained such that $q_{\Delta i} \in Q_{\Delta i}$. Such $Q_{\Delta i}$'s are polytopes, and so hence convex so the feasable set of this optimization problem is convex. Thus, given that the cost function and feasible set of this optimization problem is convex, this problem can be solved for the global optimal solution, e.g. via iterative gradient descent methods.

4.1.2 Solving quadratic program in each polytope to solve globally

Recall that it is required that the solution curve γ is both respects the physical and dynamic constraints of the problem. The physical constraints are that the image of the solution curve γ is a subset of X, and the dynamic constraints are that curve γ is smooth to the second degree, and respects the maximum velocity and acceleration constraints placed on it. Such constraints can be satisfied by considering each γ_i independently.

Consider and arbitrary γ_i. In order to ensure that the solution curve γ satisfies the physical constraints of the problem, it is sufficient to select all of the control points of the curve γ_i to be contained within the polytope, Q_i. Note that such a requirements can be formulated as linear inequality constraints.

Furthermore, the solution curve must also be dynamically feasible. Firstly, this requires that the solution curve, γ, is smooth to the second degree. One way in which

this can be ensured is to make the curve γ_i starts at the point $q_{\Delta i}$, and ends at $q_{\Delta i+1}$, and furthermore have zero velocity and acceleration at its endpoints. Recall from Equations 3.44, 3.50 and 3.52 that this requirement can be formulated as:

$$x_0^{(i)} = q_{\Delta i} \text{ and } x_d^{(i)} = q_{\Delta i+1}, \tag{4.6}$$

$$\frac{d}{t_i}(x_1^{(i)} - x_0^{(i)}) = 0 \text{ and } \frac{d}{t_i}(x_d^{(i)} - x_{d-1}^{(i)}) = 0, \tag{4.7}$$

$$\frac{d(d-1)}{t_i^2}(x_2^{(i)} - 2x_1^{(i)} + x_0^{(i)}) = 0 \text{ and } \frac{d(d-1)}{t_i^2}(x_d^{(i)} - 2x_{d-1}^{(i)} + 2x_{d-2}^{(i)}) = 0. \tag{4.8}$$

Given the above equations, the curve γ_i can be constrained to satisfy the required endpoint conditions by means of linear equality constraints, given that d, t_i are assumed as fixed.

Finally, it can be verified from the justification in Section 3.2.2 that the curve γ_i can be constrained to satisfy the maximum velocity and acceleration constraints of the problem via linear inequalities, given that the degree of the γ_i and t_i are assumed to be fixed. Given that each of the all curves γ_i satisfy the maximum velocity and acceleration constraints, it follows that γ, the solution trajectory, will satisfy these constraints.

Recall from Equation 3.10-3.12 that the cost of the curve γ_i can be expressed in the following form:

$$C_i(x^{(i)}, t_i) = \frac{1}{2}x^T A_i(t_i)x + x^T b_i(t_i) + c_i(t_i),\ A_i : \mathbb{R} \to \mathbb{R}^{(n\cdot d)\times(n\cdot d)} \tag{4.9}$$

$$b_i : \mathbb{R} \to \mathbb{R}^{(n\cdot d)} \tag{4.10}$$

$$c_i : \mathbb{R} \to \mathbb{R}. \tag{4.11}$$

Given that the value t_i is fixed the above cost is quadratic in $x^{(i)}$ and hence the optimization problem can be considered a quadratic program. Each of the curves γ_i can be solved for and the solution curve γ can be produced. After construction this formulation, I implemented it in Python, by utilizing the library CVXPY [7] and the computer algebra software SymPy [20]. An example problem can be solved via this method and visualized by running the running the following command in the root directory of the supplementary material:

```
python3 example.py -s naive_solver
```

4.2 Sophisticated Quadratic Programming inside each polytope

In the problem formulation given in the previous subsection, there was an assumption which is extremely strong, and perhaps could even be described as simply false. That assumption being that shorter curves map to lower costs. To this end, the allocation of the knot-points in the previous algorithm is completely decoupled from the cost function which the optimization is being performed with respect to, however it turns out that a simple fix to this problem is at hand. This formulation will propose a fix to this problem by considering the problem as one large quadratic program, in contrast to the M sub-quadratic programs utilized in the previous method. As it turns out, from such a reformulation, the ability to remove the requirement that the velocity and acceleration of the solution curve at the knot-points drops out.

Firstly, recall from the Section 3.1 that the evaluation of the cost function C at the sequence of duration parameterized Bézier curves $\gamma_0, ..., \gamma_{M-1}$ can be expressed in the form:

$$C(x,t) = \frac{1}{2}x^T A(t)x + x^T b(t) + c(t),\ A : \mathbb{R}^M \to \mathbb{R}^{(n\cdot d\cdot M)\times(n\cdot d\cdot M)} \tag{4.12}$$

$$b : \mathbb{R}^M \to \mathbb{R}^{(n\cdot d\cdot M)} \tag{4.13}$$

$$c : \mathbb{R}^M \to \mathbb{R}. \tag{4.14}$$

Given that the time allocation for each γ_i is fixed, the above cost function is quadratic in all of its free variables, in particular x. Given that the cost function has been shown to be quadratic, in order to show that the problem conforms to the specification of a Quadratic Program, all that is to show is that the constraints of the problem are linear in x.

Firstly note that all of the control points of the problem are to be constraints either in a polytope Q_i or they are to be constrained within some intersection of two polytopes $Q_{\Delta i}$. Clearly given that the intersection of two polytopes is itself a polytope, constraining the control points of the solution so that they are located in a valid position can be formulated via linear inequality constraints on x.

In terms of dynamic constraints, as in the previous section the maximum velocity and acceleration constraints for each γ_i can be formulated as linear inequality constraints on x. All that is to do now is enforce that there is continuity between the curves γ_i and γ_{i+1}. Unlike in the previous problem formulation where Equations 4.6 to 4.8

constrained each sub-curve to have fixed endpoints, with zero velocity and acceleration, the constraints detailed in Equation 3.43, 3.49 and 3.51 can be generalized, such that the following condition defines that there is second order continuity between each adjacent subcurve.

$$x_d^{(i)} = x_0^{(i+i)}, \tag{4.15}$$

$$\frac{d}{t_i}(x_d^{(i)} - x_{d-1}^{(i)}) = \frac{d}{t_{i+1}}(x_1^{(i+1)} - x_0^{(i+1)}), \tag{4.16}$$

$$\frac{d(d-1)}{t_i^2}(x_d^{(i)} - 2x_{d-1}^{(i)} + x_{d-2}^{(i)}) = \frac{d(d-1)}{t_{i+1}^2}(x_2^{(i+1)} - 2x_1^{(i+1)} + x_0^{(i+1)}), \tag{4.17}$$

$$\forall i \in \{0, ..., M-1\}. \tag{4.18}$$

Clearly the above constraints can be formulated as linear equality constraints on x, given that the vector t and the Bézier curve degree are assumed as fixed. Hence this problem which has been formulated is of the form of a Quadratic program which can be solved efficiently to global optimally.

After formulating the problem in this way I created a Python implementation of the a, by utilizing again the optimization library CVXPY [7] and the computer algebra software SymPy [20]. An example problem can be solved via this method and visualized by running the running the following command in the root directory of the supplementary material:

```
python3 example.py -s sophisticated_solver
```

4.3 Quadratic Programming De Casteljau Optimization

In this section the problem of solving for an optimal trajectory, γ, within the sequence of polytopes $Q_0, ..., Q_{M-1}$ will be solved via De Casteljau's Algorithm [4] coupled with Quadratic Programming. The advantage of this formulation over the formulation directly previous is that the solution curve is smooth, i.e. all of the solution curve's derivatives are continuous. In this chapter, De Casteljau's Algorithm will be outlined initially, and thereafter it will be shown how residuals of this algorithm can be used to solve for the optimal curve using Quadratic Programming.

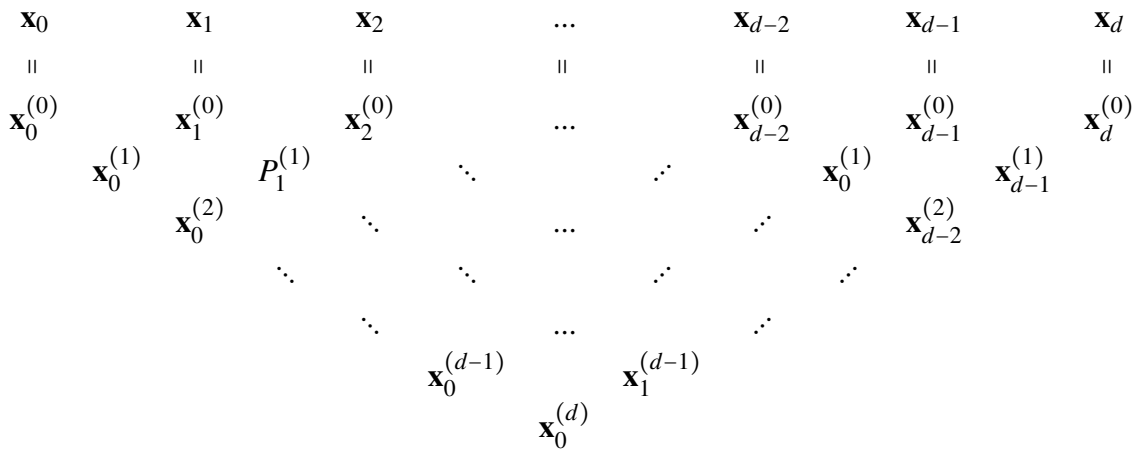

Figure 4.1: In the above diagram, the first two rows symbolize Equation 4.19. The remaining rows symbolize Equation 4.20. To this end, for an arbitrary symbol in this section of the diagram, its value is value is given by the weighted sum of the variables in the above row on the left and right side of the variable, with weights $(1 - t_0/T)$ and t_0/T respectively.

4.3.1 De Casteljau's Algorithm

Consider a duration parameterized Bézier curve, $\mathbf{B}_T$ of degree d with control points $\mathbf{x}_0, ..., \mathbf{x}_d \in \mathbb{R}^n$ and duration T. De Casteljau's Algorithm is a numerically stable algorithm which is used for evaluating the duration parameterized Bézier curve $\mathbf{B}_T$ at the point $t_0 \in [0, T]$. The algorithm is defined on the variables in the following recurrence relation:

$$\mathbf{x}_i^{(0)} := \mathbf{x}_i,\ i = 1, ..., d \tag{4.19}$$

$$\mathbf{x}_i^{(j)} := \mathbf{x}_i^{(j-1)} \left(1 - \frac{t_0}{T}\right) + \mathbf{x}_{i+1}^{(j-1)} \left(\frac{t_0}{T}\right),\ i = 1, ..., n - j,\ j = 1, ..., d \tag{4.20}$$

The double indexing involved in the previous recurrence relation is somewhat difficult to understand, and the relationships between the variables can be made significantly clearer by considering the variables in a diagramatic form, as given in Figure 4.1.
The claim of De Casteljau's Algorithm is that the value of the duration parameterized Bézier curve $\mathbf{B}_T$ at the point t_0 is given by $B(t_0) = \mathbf{x}_0^{(d)}$. This result can be shown by induction on d, however the proof is relatively simple, and the result is completely irrelevant to the optimization algorithm to be outlined in this section, so therefore it will be omitted. To this, the values of interest of the De Casteljau algorithm for a Bézier curve B and the timepoint t_0, for the sake of the optimization problem reformulation,

is not in fact the final forward-propagated final result, but a subset of the intermediate results of the calculation.

The interest of the above algorithm in the context of the optimization algorithm is the following theorem.

Theorem 5. *([2]) Consider a duration parameterized Bézier curve, $\mathbf{B}_T$ of degree d with control points $\mathbf{x}_0,\ldots,\mathbf{x}_d \in \mathbb{R}^n$ and duration T. If some value $t_0 \in [0,T]$ is selected, then residuals of the calculation in De Casteljau's Algorithm can be used to obtain the control points for dth degree duration parameterized Bézier curves $\mathbf{B}^1_{t_0}$ and $\mathbf{B}^2_{T-t_0}$ such that:*

$$\mathbf{B}_T(t) = \begin{cases} \mathbf{B}^1_{t_0}(t) & t \in [0,t_0] \\ \mathbf{B}^2_{T-t_0}(t-t_0) & t \in [t_0,T] \end{cases} \tag{4.21}$$

More specifically, it is claimed that, the appropriate control points are $\mathbf{x}_0^{(0)},\mathbf{x}_0^{(1)},\ldots,\mathbf{x}_0^{(n)}$ and $\mathbf{x}_0^{(n)},\mathbf{x}_1^{(n-1)},\ldots,\mathbf{x}_n^{(0)}$ for $\mathbf{B}^1_{t_0}$ and $\mathbf{B}^2_{T-t_0}$ respectively.

Proof. This result for a duration variable Bézier curve can be shown via a simple reparameterization of the argument for a standard Bézier curve as given in [2]. □

A visualization of the above result is given in Figure 4.2 below.

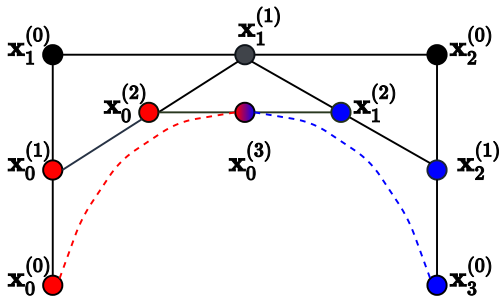

Figure 4.2: Let the dashed line in the above diagram represent a degree 3 duration parameterized Bézier curve, $\mathbf{B}_T(t)$ with control points $\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$. The above diagram shows the splitting of this curve at the point $t_0 = 0.5 \cdot T$, with $\mathbf{B}^1_{0.5\cdot T}$ and its control points shown in red and $\mathbf{B}^2_{0.5\cdot T}$ and its control points shown in blue.

The next theorem notes a critical property of this curve splitting algorithm, which will be seen to be essential for formulating the optimization problem, in the next subsection.

Theorem 6. *Consider a duration parameterized Bézier curve, $\mathbf{B}_T$ of degree d with control points $\mathbf{x}_0,\ldots,\mathbf{x}_d \in \mathbb{R}^n$ and duration T. If some value $t_0 \in [0,T]$ is selected, and*

the subdivision algorithm, as seen in Theorem 5 is applied to $\mathbf{B}_T$ *at the point* t_0 *to produce* $\mathbf{B}^1_{t_0}$ *and* $\mathbf{B}^2_{T-t_0}$*, then the control points of* $\mathbf{B}^1_{t_0}$ *and* $\mathbf{B}^2_{T-t_0}$ *are linear combinations of the control points of* $\mathbf{B}_T$*.*

Proof. Note from Theorem 5 that the control points of $\mathbf{B}^1_{t_0}$ and $\mathbf{B}^1_{t_0}$ are $\mathbf{x}_0^{(0)}, \mathbf{x}_0^{(1)}, ..., \mathbf{x}_0^{(n)}$ and $\mathbf{x}_0^{(n)}, \mathbf{x}_1^{(n-1)}, ..., \mathbf{x}_n^{(0)}$ for $\mathbf{B}^1_{t_0}$ and $\mathbf{B}^2_{T-t_0}$ respectively. However, by definition of these terms, as given in Equations 4.19 and 4.20, there terms are linear combinations of $\mathbf{x}_i$'s. □

Note that the theory included above for splitting duration parameterized Bézier curves $\mathbf{B}_T$ into two curves $\mathbf{B}^1_{t_0}$ and $\mathbf{B}^2_{T-t_0}$ at the point $t_0 \in [0, T]$ can be applied iterative define a method of splitting the curve $\mathbf{B}_T$, into M segments of duration t_i where $t \in \mathbb{R}^M$ and $\sum t_i = T$ and $t \geq 0$, $\mathbf{B}^0_{t_0}, \mathbf{B}^1_{t_1}$, ..., $\mathbf{B}^{M-1}_{t_{M-1}}$. Note that each of these subcurves will have control points which are linear combinations of the control points of $\mathbf{B}_T$, given Theorem 6.

4.3.2 Using the De Casteljau Algorithm to formulate Quadratic Program

In this section it will be outlined how the method of splitting Bézier curve via the De Casteljau Algorithm can be used to formulate the optimal trajectory generation problem.

Consider some heuristically calculated time allocation, $\tilde{t}$ to be given. Further let T be the sum of this vector. Let γ be a duration parameterized Bézier curve of degree d with control points $\mathbf{x}_0, ..., \mathbf{x}_d \in \mathbb{R}^n$ and duration T. Note that the curve γ will represent the solution curve. Further to this, consider $\gamma_0, ..., \gamma_{M-1}$ to be sub-curves of γ obtained via the method detailed in the previous section, where the times of splitting γ is defined as in $\tilde{t}$. Additionally, note by the observation at the end of the previous chapter that the control points of each of the curves γ_i can be expressed as linear combinations of the $\mathbf{x}_i$'s.

The curve γ is sure to be smooth since it is a duration parameterized Bézier curve, which is simply a polynomial. As described in the constraints section of this project, linear inequality constraints can be placed on the control points of γ in order to ensure that the curve respects maximum velocity and acceleration constraints. Finally each of the γ_i's can be constrained to remain inside the relevant polytopes, by constraining their control points with linear inequality constraints. Given that these control points

are linear in the $\mathbf{x}_i$'s, the constraints of constraining each of the γ_i's inside the correct polytope can be formulated via linear inequality constraints on the $\mathbf{x}_i$'s. With these constraints formulated, the solution curve γ can be guaranteed to reside within the polytope sequence, $Q_0, ..., Q_{M-1}$.

Finally via the result Theorem 4, the sequence of Bézier curves consisting simply of just γ with time allocation T is quadratic in $\mathbf{x}_0, ..., \mathbf{x}_d$. Hence given that it has been previously detailed how to formulate the constraints of this problem as linear equations and inequations in the $\mathbf{x}_i$'s, a Quadratic Program formulation of the problem has been obtained.

After formulating the problem in this way, I added an implementation of this solver to my Python codebase. This used CVXPY [7] for the Quadratic Programming and the computer algebra component SymPy [20] was again used. An example problem can be solved via this method and visualized by running the running the following command in the root directory of the supplementary material:

```
python3 example.py -s de_casteljau
```

Chapter 5

Time Optimized Methods

The objectives of the three methods to be outlined in this chapter are identical to the objectives of the methods in the previous chapter, i.e. generate the optimal trajectory with respect to with respect to the cost function C while respecting the physical dynamic constraints of the problem, however, the methods in this chapter will attempt to outperform previous methods, by introducing the time allocation vector t into the cost function. That is, unlike in the previous chapter, where the vector t was heuristically allocated, before an optimization in the state variables x took place the numerical optimization will now take place over the object (x,t).

Each of the three methods to be detailed in this section, will unitize the sequence of polytopes $Q_0,...,Q_{M-1}$ to construct the solution curve, γ, by concatenating sub-Bézier curves inside each of the polytopes, where continuity constraints and maximum derivitive constraints are formulated explicitly. Note that this is an identical formulation to that of the one seen in the chapter 'Sophisticated Quadratic Programming inside each polytope', apart from in the previous chapter, the constraint that $t = \tilde{t}$ was included where $\tilde{t}$ was the result of some time allocation heuristic.

In optimization theory, it is easy to define any problem which we wish, however, what the interest of this field is, is solving for the value of the problem defined. The next subsection will outline that the problem defined above can be easily solved optimally, however with the solution being of a degenerate variety. A further constraint will then be added to the problem and the next subsection thereafter will outline why the problem to be considered in this section is more 'difficult' than when t was fixed.

5.0.1 Trivial solution to the time variable optimization

Note that now a new variable, namely t, has been entered into the optimization problem, some high-level reasoning about trivial solutions is required, as to not over-complicate the problem. Recall that the optimization problem which the methods in this paper are to solve is to:

$$
\begin{aligned}
&\min C(x,t)\\
&s.t.\ (x,t) \in \{(x,t) : respects-max-vel-constraint(x,t) \wedge\\
&\qquad\qquad respects-max-acc-constraint(x,t) \wedge\\
&\qquad\qquad respects-0th-order-continuity-constraint(x,t) \wedge \quad (Problem\ 5.0.1.1)\\
&\qquad\qquad respects-1st-order-continuity-constraint(x,t) \wedge\\
&\qquad\qquad respects-2nd-order-continuity-constraint(x,t) \wedge\\
&\qquad\qquad respects-polytope-containment-constraints(x))\}\}\\
&\quad x \in \mathbb{R}^n, t \in \mathbb{R}^m\\
&\quad t > 0.
\end{aligned}
$$

Definition 11 (ε-optimality)**.** *An minimization optimization problem with cost function f and valid set X can be solved to ε optimally iff for all $\varepsilon > 0$ a point x_ε can be generated such that $|f(x_\varepsilon) - f^*| < \varepsilon$, $f^* := inf\{f(x) : x \in X\}$.*

Theorem 7 (Trivial Solution to time variable optimization)**.** *Problem 5.0.1.1 can be solved to globally ε-optimally.*

Proof of Theorem 7. This claim will be shown by construction of an ε optimal pair (x^*, t^*). Firstly, note that as the cost function C is defined to be the finite sum of integrals of non-negative functions therefore the cost function C is non-negative. Therefore, ε-optimality can be shown if, for any $\varepsilon > 0$ an (x^*, t^*) can be constructed such that $C(x^*, t^*) < \varepsilon$.

Let $\varepsilon > 0$ be given. Let $Q_0, ..., Q_{M-1}$ denote the sequence of polytopes and $\gamma_0, ..., \gamma_{M-1}$ denote the sequence of duration parameterized Bézier curves. Further, denote the intersection of Q_i and Q_{i+1} in this sequence as, $Q_{\Delta(i+1)}$. As a first step for constructing the desired (x^*, t^*) pair, inside each $Q_{\Delta(i+1)}$ select an arbitrary point, $q_{\Delta(i+1)}$. Further define $q_{\Delta 0}$ as the problem start point and $q_{\Delta(M+1)}$ as the problem end point. Let each sub-Bézier curve, i.e. γ_i, be of degree 5 (i.e. 6 control points). Let the curve γ_i have its first three control points fixed at $q_{\Delta i}$ and the remaining three control points fixed at

$q_{\Delta(i+1)}$. At this stage the vector x^* is now fixed, and the continuity constraints, along with the start and end point constraints and the polytope containment constraints are satisfied. All that is left is for the vector t^* to be fixed such that the maximum velocity and acceleration constraints are satisfied as well as the cost of the solution curve being strictly less than ε. Given that the cost C is the sum of the cost for each of the M γ_i's, namely sum of the C_i, it is sufficient to show that there exists a time allocation, t_i for the Bézier curve γ_i such that γ_i respects the maximum velocity and acceleration constraints, while having $C_i < \frac{\varepsilon}{M}$. This can be shown to be true by firstly by noting that given that the shape of the polynomial spline, γ_i is fixed, there will exist a time $t_i^{vel} \in \mathbb{N}$ such that the spline γ_i is traversed slowly enough to respect the maximum velocity constraint. Similarly $t_i^{acc} \in \mathbb{N}$ will exist, such that it defines a traversal time large enough for the maximum acceleration constraint to be respected on γ_i. Finally, note from Equation 3.26 and 3.28 and that given that all of the $x_{\_,\_}^{(\_)}$ are fixed (since x^* is fixed), the limit of C_i is zero as $t_i \to \infty$. Therefore, there exists an $t_i^{(cost)} \in \mathbb{N}$ such that traversing Bézier curve with fixed shape from x and time t_i^{cost} will result in $C_i < \frac{\varepsilon}{M}$. As to ensure that all of the requirements are satisfied, define $t_i^* = \min\{t_i^{vel}, t_i^{acc}, t_i^{cost}\}$ Naturally construct the vector t^* from the t_i^*'s and the construction of the ε-optimal (x^*, t^*) is complete. □

Given that this solution is clearly not valid, in terms of real life application of the algorithm (e.g. If this solver was to be applied to path planning for drones, the drone would likely run out of battery, before the solution trajectory gets fully realized). Given this, for this projects work on time variable optimization, a further constraint will be introduced, in particular, a maximum duration constraint on the solution curve γ, which is heuristically calculated.

5.0.2 Non-convexity of the problem, with optimization over state and time variables

With this modification in the formulation of the problem introduced in this chapter of letting the time allocation vector t be variable, the cost function of interest, C, is no longer quadratic in the variables of optimization, e.g. x and t. Recall that C can be expressed in the form:

$$C(x,t) = \frac{1}{2}x^T A(t)x + x^T b(t) + c(t). \tag{5.1}$$

Although the cost function, C, is non-quadratic in the variables of optimization, the problem of optimization may still be easy, if it can be shown that the constraints and the cost function form a convex optimization problem. However, this is not the case, and this can be verified by showing that there is a constraint in the problem formulation which is non-convex. Consider the second order continuity constraint, between two curves in one dimension. Let t_0 denote the duration of the first curve, and let t_1 denote the duration of the second curve. Further let a,b,c denote the final 3 control points of the first curve, in the given dimension and let c,d,e denote the first 3 control points of the second curve in a given dimension. Then, the constraint can be formulated as:

$$\frac{1}{t_0^2}(a-2b+c) = \frac{1}{t_1^2}(c-2d+e) \tag{5.2}$$

Denote the set of valid points of this constraint as the set as V, i.e.

$$V = \left\{ (t_0,t_1,a,b,c,d,e) : \frac{1}{t_0^2}(a-2b+c) = \frac{1}{t_1^2}(c-2d+e), t_0,t_1 a,b,c,d,e \in \mathbb{R} \right\}. \tag{5.3}$$

For this set to be convex, it must be the case that for all $v_1, v_2 \in V$ and $\lambda \in [0,1]$, $(1-\lambda)v_1 + \lambda v_2 \in V$. However this is clearly not the case, e.g. take $\lambda = 0.5$,

$$v_1 = (1,3,1,1,1,2,3), \text{ and} \tag{5.4}$$

$$v_2 = (1,4,2,1,1,1,17). \tag{5.5}$$

Hence given that one of the constraints of this problem is non-convex, the feasible set of the optimization problem is non-convex, in general. Given that the feasible set of the optimization cannot be assumed to be convex, the problem now falls outwith the scope of convex optimization, and alternative methods must be appealed to. At this stage, one should note that non-convex optimization is at least NP-hard, so it is mathematically impossible for there to be a stand alone algorithm to solve this problem to global optimally, with the current level of investigation on the characteristics of the optimization problem. Recall, that the reason why non-convex optimization is harder than convex optimization is that in non-convex optimization, local optima, are not guaranteed to be globally optimal. An illustrative example of these local optima on account of a non-convex feasable set, would be to optimize the the function $f(x) = x^2$ such that $x \in \mathbb{N}$. There is a local optima at each $x \in \mathbb{N}$.

Thus, there are currently two options, firstly one could investigate the problem definition (e.g. algebraic properties, Lipschitz Continuity) in order to reason about when

a local optimal is a global optimum (e.g. obtaining a closed form solution). A trivial yet illustrative example of this methodology would be to observe that the non-convex optimization problem; $f : \mathbb{R} \to \mathbb{R}, f(x) = 2 + \sin(x), x \neq 44$ and $f(x) = a, x = 44, a \in \mathbb{R}$, can be globally optimized using hard coded $if - then - else$ reasoning. Alternatively, one could simply accept that the solution obtained from an non-convex solver has no theoretical guarantees, and use quantities results to justify utilizing the algorithm.

The first two methods to be described in this chapter will take the form of the later methodology. That is in Section 5.1 and Section 5.2, solvers will be tasked with solving for local optima of the problem definition. These methods will appeal to the method of Trust-Region optimization [3] and Bi-level optimization [29]. In Section 5.3, an attempted method to solve the problem defined globally is outlined. This method attempts to utilize Semi-Definite programming [25].

5.1 Constrained Trust Region Optimization

The Constrained Trust Region Optimizaiton algorithm [3] is one of the standard algorithms which attempts to solve for local minima in large scale optimization problems, however, the utilization of this solver comes at a cost. This penalty comes in the form of not allowing for hard constraints and only allowing for soft constraints to be added into the cost function. More explicitly, suppose that the optimization problem which we are working with in this paper is written in the form:

$$\min_{x,t} \; C(x,t) \tag{5.6}$$

$$s.t. \; f(x,t) \leq 0, \tag{5.7}$$

$$g(x,t) \; = 0, \tag{5.8}$$

$$x \in \mathbb{R}^n, t \in \mathbb{R}^m. \tag{5.9}$$

Note that in the above formulation, x and t still represent the state variables and the timing variables. Furthermore, note that the constraint $f(x,t) \leq 0$, represents all of the inequality constraints of the problem, e.g. maximum acceleration constraints. Note that the output of the function f is a vector with each row representing corresponding to the value of one inequality constraint in the problem definition. Similarly, $g(x,t) = 0$ represents the equality constraints of the problem. For example, this could be the first order continuity constraints endowed on two sequential sub-Bézier curves.

Now a further two functions will be defined, for utilization in the new problem formulation. Introduce variables $u, v \in \mathbb{N}$ to represent the number of linear constraints in equations 5.7 and 5.8 respectively. Firstly $d_{f\leq} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ such that:

$$d_{f\leq}(x,t) = \sqrt{\sum_{i=1}^{u} f(x,t)_i^2 \cdot \delta(f(x,t)_i > 0)} \tag{5.10}$$

Secondly, define the function $d_{g=} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ such that:

$$d_{g=}(x,t) = \sqrt{\sum_{i=1}^{v} g(x,t)_i^2} \tag{5.11}$$

Note that one can simply consider the functions $d_{f\leq}$ and $d_{g=}$ as functions which punish violations of the constraints of the problem using the classic d_2 metric.

The required definitions are now in place to define the optimization problem which the trust region algorithm will consider. The problem has the form:

$$\min_{x,t} \; m(x,t) := C(x,t) + \lambda[\{d_{f\leq}(x,t)\}^2 + \{d_{g=}(x,t)\}^2] \tag{5.12}$$

$$s.t. \;\; x \in \mathbb{R}^n, t \in \mathbb{R}^m. \tag{5.13}$$

Note that in the above optimization problem $\lambda \in \mathbb{R}, \; \lambda > 0$ is a parameter which defines the the trade-off between minimizing the cost function C, and minimizing the violation of the constraints. Note that the function m is commonly referred to as the 'merit-function' of the initial problem [1].

In order to detail the solving method of this algorithm, one further concept must be introduced, namely the Taylor expansion of the function at a point. In particular, at the point (x_0, t_0) the function m can be approximated by the Taylor quadratic model [10]. For ease of notation let $\tilde{x} = [x^T, t^T]^T$ and $\tilde{x_0} = [x_0^T, t_0^T]^T$.

$$m(\tilde{x}) \approx m(\tilde{x_0}) + \nabla m(\tilde{x_0}) \cdot (\tilde{x} - \tilde{x_0}) + \frac{1}{2}(\tilde{x} - \tilde{x_0})^T \cdot \nabla^2 m(\tilde{x_0}) \cdot (\tilde{x} - \tilde{x_0}) \tag{5.14}$$

Note that the Jacobian, $\nabla m(\tilde{x})$, and the Hessian, $\nabla^2 m(\tilde{x})$, of the function m are required in the above approximation. In order to obtain these functions, the Jacobian and Hessian matrix functions for each of the constraints as well as the cost function C can be obtained via computer algebra. From this point the functions $\nabla m(\tilde{x})$ and $\nabla^2 m(\tilde{x})$ can be constructed, since m is constructed from the cost function C and the constraints.

The function quadratic approximation of an analytic function approximates the function well for points which are 'near' to where the quadratic approximation is centered, and is expected to approximate the function less well, as the distance increases from the center and the query point [10]. This fact will be taken into account by the trust region algorithm, which will be outlined in the next paragraph.

Now, the trust region algorithm for solving such an unconstrained optimization will be outlined. This is an optimization algorithm which is initialized at some staring tuple (x,t). Then a quadratic model of the function is formed at the current point (i.e. compute the second order Taylor Polynomial of the function at this point). The minimum point of the quadratic approximation is calculated. Note that this exists in closed form solution the minimization of the quadratic approximation [10], since the derivative of the right hand side of Equation 5.14 with respect to $\tilde{x}$ is:

$$\nabla m(\tilde{x_0}) + \nabla^2 m(\tilde{x_0}) \cdot \tilde{x} \tag{5.15}$$

The above formula can be set equal to zero and the critical value of $\tilde{x}$ can be solved for. A step in the direction of this minimum is taken by the optimization algorithm, only to the extent in which the the algorithm believes the quadratic model agrees with the error function. This step size is known to be the 'Trust-Region' and the algorithm to calculate this value is given in [3]. The algorithm then iterates by making a new quadratic model and step at each new point, until convergence. Given that this method has no optimally guarantees, its performance will need to be evaluated quantitatively. Such analysis is given in the next chapter.

After formulating the problem in the above form, I added an implementation of this solver to my Python codebase. This used the SciPy [32] implementation of the trust-region algorithm and SymPy [20] was again used for the computer algebra. An example problem can be solved via this method and visualized by running the running the following command in the root directory of the supplementary material:

```
python3 example.py -s non_convex_solver
```

Note that due to the formulation of the problem, constraint violation, may be observed.

5.2 Bilevel Optimization

The next method to be considered in this analysis is that of Bilevel-optimization. This is a method which aims to recover some of the nice properties of the optimization problem when the time allocation is fixed, while *not* actually enforcing that the time allocation of the problem is fixed. This is achieved by firstly partitioning the variables of optimization (x,t) into two sets; one for state variables and one for timing variables, i.e. simply the variables in the vector x and the variables in the vector t. As prefaced in the name of the algorithm, the solving of the problem is defined to have two levels of optimization, namely, the $upper-level$ optimization and the $lower-level$ optimization. To this end, in this particular example, the upper-level optimization is over the vector t while the lower-level optimization is over x assuming t is fixed, and therefore unlocking the nice optimization properties outlined in the previous chapter. Such a methodology will now be outlined. This formulation of the problem has been seen in [29].

Note firstly in the problem which is being attacked in this paper, there may be linear constraints placed on the timing vector t. For example, an inequality constraint restricting the total time of the solution curve, γ, may be included. Furthermore, it is feasible that one may wish to also include equality constraints on the timing variable t, and the framework supports such a constraint, so the math will be worked through with such a constraint. Let the equality and inequality constraints on t, be denoted as $At = b$ and $Et \leq f$. Further to this, note that there may also be constraints on the control points x and t. Assume that these can be written in the form $G(t)x = h(t)$ and $I(t)x \leq j(t)$. Such equality constraints may represent the continuity constraints between adjacent sub-Bézier curves, whereas the inequality constraints may represent the control point polytope containment constraints. The optimization problem can now be written out in full as:

$$\min_{x,t} \; C(x,t) \tag{5.16}$$

$$s.t.\; At = b,\; Et \leq f, \tag{5.17}$$

$$G(t)x = h(t),\; I(t)x \leq j(t), \tag{5.18}$$

$$x \in \mathbb{R}^n, t \in \mathbb{R}^m. \tag{5.19}$$

An identical re-expression of the above optimization problem would be [29]:

$$\min_{t} \ \sigma(t) = C(x^*, t) \tag{5.20}$$

$$s.t.\ At = b,\ Et \leq f, \tag{5.21}$$

$$t \in \mathbb{R}^m, \tag{5.22}$$

$$x^* = \underset{x}{\operatorname{argmin}}\ C(x,t) \tag{5.23}$$

$$s.t. G(t)x = h(t),\ I(t)x \leq j(t), \tag{5.24}$$

$$x \in \mathbb{R}^n. \tag{5.25}$$

The above equivalence between the two optimization problems is simply by definition. Simply note that the objective of the first optimization problem is to find the (x^*, t^*) pair which minimizes the cost function C while respecting all of the constraints. In the second optimization problem, *the upper-level optimization*, over t, is defined to find t^* which minimizes the cost C firstly respects the timing constraints and secondly such that x^* is chosen optimally, while respecting the constraints parameters by t. Therefore, by definition, the value of the problems, must be the same. For the remainder of this subsection, the second formulation will be used. Note further, the optimization over x will be referred to herein as *the lower-level optimization.*

To start off the analysis of the optimization problem, note that the lower-level optimization has already been seen in this paper, in the Section 4.1.2, namely, 'Sophisticated Quadratic Programming inside each polytope'. In this section, it was justified that when the time allocation of the problem is fixed, as it is in this case, the problem constitutes a Quadratic-Program which can be solved efficiently via the interior points method. Note that as an intermediate result for solving this optimization problem, Lagrange Multipliers [22] are produced. Suppose that the equality constraint associated with the quadratic program, i.e. Equation 5.24, is written in the following form $g(x,t) = 0$ and $f(x,t) \leq 0$ has l rows. Further suppose $g(x,t) = 0$ has k rows, and $f(x,t) \leq 0$ has l rows. Then the Lagrange multiplier associated with the equality constraint will be some vector $\lambda \in \mathbb{R}^k$ and similarly the Lagrange multiplier associated with the inequality constraint will be some $\mu \in \mathbb{R}^l$.

The ability to solve the lower-level-optimization efficiently alone, thus far, does not help in solving the upper-level-optimization. To this end, in the upper level optimization, a valid vector t can be generated and the value of the problem with this time allocation can be calculated. however as is, there is no information as to tell the upper level optimization how to adjust it's value of t. This is what is desired in opti-

mization theory - an intelligent method for selecting the next point of the search, with genuine evidence that the next point should be better than the previous. The method of finite-differences could be utilized in order to approximate the gradient as detailed in [17], however, this method only provides an approximation of the gradient. Recent work detailed in [29] provides a method for calculating the gradient of the function σ with respect to t, from the Karush–Kuhn–Tucker conditions for optimally. The desired result is calculated from the Lagrange Multilpiers associated with the solving of the the as well as the gradients of relevant constraints. The explicit formula for $\nabla_t \sigma(t)$ is:

$$\nabla_t \sigma(t) = \lambda^T \nabla_t g(x^*, t) + \mu^T \nabla_t f(x^*, t) + \nabla_t C(x^*, t) \tag{5.26}$$

Given that zeroth and first order information for σ is available, a Line Search [22] as was done in [29] will be appealed to in order to search over candidate time allocations in order to search for the best value of t.

After learning from [29] about this formulation of the problem, I added an implementation of this solver to my Python codebase. This used the SciPy [32] implementation of the line-search algorithm and SymPy [20] was again used for the computer algebra. An example problem can be solved via this method and visualized by running the running the following command in the root directory of the supplementary material:

```
python3 example.py -s bilevel_opt
```

5.3 Semi-Definite Programming

In this section, a failed, yet new research thread of this project will be detailed. In particular, the content of this research thread was an attempt to formulate the optimization problem being considered in this chapter as a Semi-Definite Program (SDP) [25]. In this section, initially, the form of a Semi-Definite program will be outlined, followed by a detailed description of two attempted formulations.

Firstly, two definitions will be given which are required to define the form of a Semi-definite program will be given.

Definition 12. *([33]) A matrix $A \in \mathbb{R}^{n\times n}$ is said to be positive semi-definite if, for all vectors $x \in \mathbb{R}^n$, $x^T Ax \geq 0$. Further if a matrix $A \in \mathbb{R}^{n\times n}$ is positive semi-definite, then this is denoted as $A \succeq 0$.*

Given the above definition, let the set $\mathbb{S}^n_+$ denote the of matrices in $\mathbb{R}^n$ which are symmetric and positive semi-definite. Additionally, let $\mathbb{S}^n$ denote the set of matrices in $\mathbb{R}^n$ which are symmetric.

Definition 13. *([33]) The inner product $\bullet$ on the set $\mathbb{R}^n$ is defined as $A \bullet B = trace(A^T B)$.*

Given the above definitions, the form of a Semi-definite programm (SDP) can be given. Note that an SDP is an optimization problem of the form:

$$\min_X C \bullet X, \tag{5.27}$$

$$s.t.\ A_i \bullet X = b_i,\ i \in \{1,...,m\}, \tag{5.28}$$

$$X \succeq 0, \tag{5.29}$$

$$\text{where } C, A_i \in \mathbb{S}^n, b \in \mathbb{R}^m, X \in \mathbb{S}^n_+. \tag{5.30}$$

Note, that optimization problems with the above structure can be solved systematically to global optimally using the Interior Points Method [24], hence formulating the time variable problem in this way is an extremely attractive prospect. My attempts to formulate the problem in such a way failed, however, my attempted formulations will now be outlined.

To outline the first formulation, let $t^{-1} \in \mathbb{R}^M$ denote the vector of the reciprocals of each of the t_i's, i.e. $t^{-1} = [t_0^{-1},...,t_{m-1}^{-1}]^T$. Then consider the starting point of the formulation where, the variable X is considered to represent the state and time variables

(x,t) in the following way:

$$X := [x^T, (t^{-1})^T]^T [x^T, (t^{-1})^T]. \tag{5.31}$$

That is, take X as the outer product of the vector $[x^T, (t^{-1})^T]^T$. Taking X as such allows for bilinear terms from the set of variables $\{x_0, ..., x_{M\times d\times n}, t_0^{-1}, ..., t_{M-1}^{-1}\}$ to appear in the cost function and the constraints and the constraints of the problem. This form of X has the potential to allow for first order continuity constraints to be formulated, as detailed in Equation 3.49 and Equation 3.50. However, given that it can be verified from the proof of Theorem 4, that the cost function C is constructed form a linear combination of terms of the form:

$$\frac{x_i x_j}{t_k^3} \tag{5.32}$$

and noting the definition of the inner product $\bullet$, it will not be possible to formulate the cost function C given this definition for X.

However, an alternative representation of the variables (x,t) can be considered inside the matrix X, which allow for the formulation of the cost function, C. Note that it can be verified from the proof of Theorem 4 that the terms detailed in Equation 5.32 will only appear for state and time variables in the same subcurve, γ_i. Hence define the matrix X_i as:

$$X_i = \frac{x^{(i)} x^{(i)T}}{t_i^3}. \tag{5.33}$$

Further to this, consider X to be the matrix with variable representation:

$$X = diag\{X_0, ..., X_{M-1}\}. \tag{5.34}$$

Such a matrix X now gives scope for representing the cost function C, however, note that all constraints of the problem must now depend on time. However, this is clearly not the case, e.g. consider the constraint of ensuring that all of the control points remain inside the correct polytope of the polytope sequence.

I considered at this point to have exhausted my ideas for how to construct X and so concluded my investigation into this potential solution.

Chapter 6

Experiments

In this chapter three experiments will be outlined which will help to investigate the value of methods presented in the previous chapters, where theory alone is insufficient. To this end, there will not be experiments presented comparing the three algorithms in Chapter 4, with each other. The first reason is that for a given problem, the 'Sophisticated Quadratic Programming' will always outperform the other two methods due to it being a less constrained optimization and therefore, this method will always perform at least as well as the other methods. In addition to the quality of the solution produced by these methods, the computation time for these methods is also not of interest given that the computer algebra results can be cached and the solving of formulated problems will be will take only a few milliseconds, and hence any comparison would be discussing the difference of a few milliseconds.

However, what is of interest and has not yet been covered by the theory in this report is how the time variable methods which were successfully formulated, e.g. the trust-region solver and the bilevel optimization, perform.

In the next subsection the domain which this thesis will use for evaluating the time variable methods will be described, and in the subsection thereafter, the three experiments will be specified in detail, and motivation for each of these three experiments will be given.

6.0.1 Domain for experiments

The experiments to be detailed in thesis will take place in $\mathbb{R}^3$, in order to establish how well the time-variable methods will perform the drone use-case, which has been previously motivated. In order to ensure that the results produced in this report are re-

liable, each experiment should be repeated several times. Therefore, it is essential that random problems in this domain can be produced. The methodology for generating random problems will now be outlined.

In order to generate a random problem in $\mathbb{R}^3$, firstly a Hyperrectangle is generated to be centered at the fixed point of $(100, 100, 100) \in \mathbb{R}^3$. The length of the three sides are sampled from independent uniform distributions with support $[1, 15]$. This polytope is then added to the problem definition, and next a point inside this polytope is sampled. Another Hyperrectangle is generated centered at this point from the same distribution as the previous, but independently. This new polytope is added to the problem definition and this process is repeated until 6 polytopes have been generated. The start point of the problem is defined to be $(100, 100, 100) \in \mathbb{R}^3$ and the endpoint of the problem is a point sampled uniformly from the final polytope. A reparameterization of this algorithm was used to generate Figure 2.4.

All of the experiments will be repeated 50 times. Further to this, the objects which the optimization will be over is a sequence of degree 8 duration parameterized Bézier curves. Each solver will be requested to respect the control point containment constrains, as well as being requested to produced solution trajectories which are smooth to the second degree. Additionally, in all experiments, no maximum velocity or acceleration constraints will be included, since including no bounds is just as arbitrary as fixing some bounds at specific values, however, randomly sampling these bounds is a potential avenue for future work. Finally, no maximum time to complete the trajectory is given to the solvers, so that the global optima of all of the problems, i.e. ε-optimality at zero, is known and can hence be used as a reference point for comparing performance of different methods.

6.0.2 Experiment Specifications

Given the framework form, three experiments with three distinct quantities of interest will be given, as well as motivation for these experiments.

The first experiment will be to run the Trust-Region algorithm and Bilevel Optimization method on the same 50 randomly generated problems and record the computation time of each of the numerical optimizations used in these algorithms. The motivation for this work is to establish which method is better suited to 'on-line' use.

The second experiment will be to run the Trust-Region algorithm in the domain specified above and for each random problem the maximum value of constraint viola-

tion will be recorded. The motivation for this experiment is to establish the extent to which this solver can ensure constraint satisfaction, in order to consider the safety of trajectories produced.

The final experiment will be to compare the solution quality of the trajectory at the initial iteration of the Bilevel-optimization algorithm vs the final trajectory produced, i.e. simply the solution of the sophisticated quadratic program formulation with time heuristic vs the solution quality of the trajectory generated at the completion of the Bilevel-optimization algorithm. The reasoning for undertaking this experiment is that the results can be analysed to investigate to what extent can optimizing over time, via Bi-level optimization, reduce the solution cost to the global minima.

The results of these experiments will be tabulated in the next chapter, and the chapter thereafter will present an analysis of the results.

Chapter 7

Results

In this chapter, the results of the experiments defined in the previous chapter will be given. To this end, note that Table 7.1 contains the results of the first experiment where the computation time of the Trust-Region method and the Bilevel optimization method were to be compared. Next, note that Table 7.2 contains the results of the second experiment, which was an investigation into the values of maximum constraint violation of the trust-region solver. Finally for the third experiment, comparing the extent to which the bi-level optimization method is able to improve on the cost of the sophisticated quadratic program formulation which uses heuristic methods to define the time allocation, the results are given in Table 7.3.

Method	Average Computation time (ms)	Standard deviation (ms)
Trust-Region	33820.06	31265.64
Bilevel Optimization	15.59	13.54

Table 7.1: This table outlines the results of the first experiment. In particular, the average solve time for the Trust-Region and Bilevel optimization are given along with their standard deviation.

Average Maximum constraint violation	Standard deviation
33820.06	31265.64

Table 7.2: This table outlines the results of the second experiment into the maximum constraint violation of the Trust-Region solver.

Average cost fraction after refinement	Standard deviation
0.4621	0.1168

Table 7.3: This table outlines the results of the third experiment. In particular, the table details the average quotient of the solution cost of the Bi-level optimizer at the final iteration and the first iteration.

Chapter 8

Discussion

In this section, firstly conclusions will be drawn from the set of empirical results outlined in the previous chapter. In the second section of this chapter, the report will be concluded with a critical review of this work undertaken in this project and the key achievements of this work.

8.0.1 Experimental Conclusions

The results of the first experiment show clearly that the Trust-Region method of optimization is not suitable for real-time trajectory planning, while in this experimental domain, the Bilevel optimization method was shown to produce computation time results, well inside the bounds for applicability in real time domains.

Next, the second experiment produces results which gives compelling evidence for the non-applicability of the trust-region to problems in this domain. To this end, the average maximum constraint violation for creating trajectories in the test domain was in the order of 10^5. I had hoped that the solver would be able to use the second order information to establish constraint satisfiability or even near constraint satisfiability, however, this is not the case given the numerical results of this experiment. This method must now be considered to generate unsafe trajectories.

From the results of the final experiment, it can be seen that appealing to the method of Bilevel optimization to refine the time allocation for the problem is worthwhile. To this the numerical results of this experiment illustrate that over the course of the algorithm, the cost of the solution curve gets reduced by more than half of the way to the global optimal value, precisely more that 53% of the way.

8.0.2 Report Conclusions

Firstly, this section will begin with a brief point of critical review of the work undertaken in this project. Thereafter, the main achievements of the project will be given.

A point of criticism of the work undertaken in this project is that formulating the constraints of the problem as 'soft', as was done in the the non-convex Trust-Region solver method violates key design principles which algorithms for real-time applications, e.g. drone path planning, should conform to. To this end and also ignoring the large computation time of this method, when the robot is operating there is no theoretical guarantees that it will be able to generate a plan which is near to satisfying the constraints. This is in contrast to the convex, heuristically allocated time methods, where if the problem is feasible then the global optimal solution will be returned by the planner. Thus with a soft constraint formulation, the robot, e.g. drone, may be left to operate without a feasible plan and it cannot be justified to operate a robot which may not be able to plan successfully or execute non-safe trajectories.
The main achievements of this work are enumerated below:

1. Three formulations of how the smooth trajectory problem can be solved, assuming a heuristically calculated time allocation were formulated during the course of this project.

2. Implementation produced of five different problem formulations in Python, appealing to Numerical Optimization and Computer Algebra packages when appropriate. The code is well structured and fully commented for maintainability and open-source use.

3. An argument, backed up by statistics has been produced to show the benefits of using the Bilevel optimization method of non-convex optimization over using a Trust-region solver on the problem definition.

Bibliography

[1] Constrained trust region. https://docs.scipy.org/doc/scipy/reference/optimize.minimize-trustconstr.html. Accessed: 2020-01-03.

[2] Why is the subdivision algorithm correct? https://pages.mtu.edu/ shene/COURSES/cs3621/NOTES/spline/Bezier/b-sub-correct.html. Accessed: 2020-04-04.

[3] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-Region Methods*. Society for Industrial and Applied Mathematics, USA, 2000.

[4] John Daintith and Edmund Wright. *A Dictionary of Computing*. Oxford University Press, Inc., USA, 6 edition, 2008.

[5] Robin Deits and Russ Tedrake. *Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming*, pages 109–124. Springer International Publishing, Cham, 2015.

[6] Robin Deits and Russell Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. 2015:42–49, 06 2015.

[7] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[8] R. Featherstone and D. Orin. Robot dynamics: equations and algorithms. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 826–834 vol.1, 2000.

[9] Fei Gao, William Wu, Yi Lin, and Shaojie Shen. Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial. *2018*

IEEE International Conference on Robotics and Automation (ICRA), pages 344–351, 2018.

[10] Jacek Gondzio. Lecture notes for fundamentals of optimization. *The University of Edinburgh*.

[11] Wolfram Research, Inc. Bernstein polynomial.

[12] Wolfram Research, Inc. Bezier curve.

[13] Christos Katrakazas, Mohammed A. Quddus, Wen hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. 2015.

[14] Dan King. Space servicing: Past, present and future. 2001.

[15] Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. 06 2014.

[16] Tushar Kulkarni and Rashmi Uddanwadiker. Overview: Mechanism and control of a prosthetic arm. *Molecular cellular biomechanics: MCB*, 12, 01 2016.

[17] Randall LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics Classics in Applied Mathemat)*. Society for Industrial and Applied Mathematics, USA, 2007.

[18] Changliu Liu, Chung-Yen Lin, and Masayoshi Tomizuka. The convex feasible set algorithm for real time optimization in motion planning. *SIAM Journal on Control and Optimization*, 56, 09 2017.

[19] D. Mellinger, A. Kushleyev, and V. Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *2012 IEEE International Conference on Robotics and Automation*, pages 477–483, 2012.

[20] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo,

Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[21] Jean-Paul Laumond Mylène Campana, Florent Lamiraux. A simple path optimization method for motion planning. 2015.

[22] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[23] B.-J Park, H.-J Lee, K.-K Oh, and C.-J Moon. Jerk-limited time-optimal reference trajectory generation for robot actuators. *International Journal of Fuzzy Logic and Intelligent Systems*, 17:264–271, 12 2017.

[24] Florian A. Potra and Stephen J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281 – 302, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[25] Motakuri V. Ramana and Panos M. Pardalos. *Semidefinite Programming*, pages 369–398. Springer US, Boston, MA, 1996.

[26] Bharat Rao, Ashwin Goutham Gopi, and Romana Maione. The societal impact of commercial drones. May 2016.

[27] T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed integer programming for multi-vehicle path planning. In *2001 European Control Conference (ECC)*, pages 2603–2608, 2001.

[28] S. Spedicato and G. Notarstefano. Minimum-time trajectory generation for quadrotors in constrained environments. *IEEE Transactions on Control Systems Technology*, 26(4):1335–1344, 2018.

[29] Weidong Sun, Gao Tang, and Kris Hauser. Fast UAV trajectory optimization using bilevel optimization with analytical gradients. *CoRR*, abs/1811.10753, 2018.

[30] Sung-Hee Lee, Junggon Kim, F. C. Park, Munsang Kim, and J. E. Bobrow. Newton-type algorithms for dynamics-based robot movement optimization. *IEEE Transactions on Robotics*, 21(4):657–667, 2005.

[31] S.H. Tang, Weria Khaksar, Napsiah Ismail, and Mohd khairol anuar Mohd ariffin. A review on robot motion planning approaches. *Pertanika Journal of Science and Technology*, 20:15–29, 01 2012.

[32] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[33] Alemseged Weldeyesus. Lecture notes for large scale optimization for data science. *The University of Edinburgh*.

[34] Gil Jin Yang, Raimarius Delgado, and Byoung Wook Choi. A practical joint-space trajectory generation method based on convolution in real-time control. *International Journal of Advanced Robotic Systems*, 13(2):56, 2016.

[35] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, PP:1–1, 07 2019.