Motion Planning for Cooperative Autonomous Robots using optimization Tools

Thomas David Pamplona Berry

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. António Manuel dos Santos Pascoal

December 2020

Acknowledgments

I would like to thank, first of all, my dissertation supervisor, Prof. Pascoal for his insight, support and sharing of knowledge that has made this thesis possible, and acknowledge too, the importance of his work in providing the foundations of this study.

My thanks are equally due to my collaborators, namely Isaac Kaminer, Naval Postgraduate School, and Venanzio Cichella, Iowa Technology Institute for their input throughout the year.

I would like to thank my friend Pitchaporn Paitong for her support.

Finally, I want to give a special thanks my parents and sister for their encouragement, care and support over all these years, and for always being there for me through thick and thin.

Abstract

Worldwide, there has been growing interest in the execution of missions of increasing complexity involving the use of several autonomous vehicles acting cooperatively without constant supervision of human operators. A key-enabling factor for the execution of such missions is the availability of advanced methods for cooperative motion planning that take explicitly into account temporal and spatial constraints, intrinsic vehicle limitations and energy minimization requirements.

Motivated by the current trends in this area of research, the motion planning techniques studied here are tasked with finding feasible and safe trajectories for a group of vehicles such that they reach a number of target points at the same time (the so called ”simultaneous arrival problem”) and avoid inter-vehicle as well as vehicle/obstacle collisions, subject to the constraint that the overall energy required for vehicle motion is minimized. Here, the motion planning problem is formulated as a continuous-time optimal control problem. Different numerical Direct Methods that approximate its solutions in a discretized setting are explored with a focus on Bernstein polynomials. These polynomials possess convenient properties that allow for efficient computation and enforcement of constraints along the vehicles' trajectories, such as maximum speed, angular rates, and minimum distance between trajectories along with the minimum distance between the vehicles and known obstacles.

Different mathematical tools to calculate cost and feasibility are evaluated. Finally, the results of simulations aimed at showing the efficacy of the complete motion planning algorithm developed for specific numbers of vehicles and different constraints are presented.

Keywords

Motion Planning, Cooperative Autonomous Vehicles, optimization, Bezier curves, Differentially Flat systems.

Resumo

A nível mundial, existe um aumento no interesse da execução de missões de complexidade crescentemente elevada que envolvem a utilização de vários veículos autónomos cooperativos sem supervisão constante de operadores humanos. Um fator chave para a execução de tais missões é a disponibilidade de métodos avançados para o planeamento de movimento cooperativo que leva explicitamente em conta restrições temporais e espaciais, limitações intrínsecas do veículo e requisitos de minimização de energia.

Motivadas pelas tendências atuais nesta área de pesquisa, as técnicas de planeamento de movimento aqui estudadas têm a tarefa de encontrar trajetórias viáveis e seguras para um grupo de veículos de modo a que atinjam vários pontos-alvo ao mesmo tempo (chamados ”problemas de chegada simultânea”) evitando colisões entre veículos, bem como entre veículos e obstáculos, tendo em conta restrições na energia consumida. Aqui, o problema de planeamento do movimento é formulado como um problema de controlo ótimo em tempo contínuo. Diferentes métodos numéricos diretos que aproximam variáveis de forma discreta são explorados com base em polinómios de Bernstein. Estes polinómios possuem propriedades convenientes que permitem o cálculo eficiente e aplicação de restrições ao longo das trajetórias dos veículos, como velocidade máxima, taxas angulares, distância mínima entre as trajetórias, bem como a distância mínima entre os veículos e obstáculos.

Serão avaliadas diferentes ferramentas matemáticas para calcular custos e viabilidade. Por último, são apresentados resultados de simulações que mostram a eficácia do algoritmo completo para números específicos de veículos e diferentes restrições.

Palavras Chave

Planeamento, Veículos Autonomos Cooperativos, Optimização, Curvas de Bezier, Sistemas Differentially Flat.

Contents

List of Figures

List of Tables

Listings

Acronyms

ODE	Ordinary Differential Equation
DOF	Degress of Freedom
AMV	Autonomous Marine Vehicle
TT	Trajectory-Tracking
PF	Path Following
NLP	Non-linear Problem
GJK	Gilbert–Johnson–Keerthi
EPA	Extended Polytope Algorithm
DEPA	Directed Extended Polytope Algorithm
IVP	Initial Value Problem
LQR	Linear Quadratic Regulator

1 Introduction

Contents

1.1 Motivation

Worldwide, there has been growing interest in the use of autonomous vehicles to execute missions of increasing complexity without constant supervision of human operators.

The WiMUST project [1] is an example of a project that demanded the use of such autonomous vehicles. The goal was to design a system of cooperating Autonomous Marine Vehicles (AMVs) able to perform innovative geotechnical surveying operations. Specifically, the WiMUST system consisted of an array of physically disconnected AMVs, acting as intelligent sensing and communicating nodes of a moving acoustic network. Together, the vehicles formed a geometry formation actively controllable according to the needs of a specific application.

Applications for the AMV formation handled by the WiMUST system include seabed mapping, seafloor characterization and seismic exploration. The overall system behaves as a distributed acoustic array capable of acquiring acoustic data obtained by illuminating the seabed and the ocean sub-bottom with strong acoustic waves sent by one (or more) acoustic sources installed onboard a support ship/boat (see figure 1.1). Advantages of multiple AMVs acting cooperatively as opposed to a single one include robustness against failure of a single node and improved seabed and sub-bottom resolution. They can also adapt better to unforeseen circumstances in the terrain by making better use of the larger environments that they can observe as the spatial distance between each AMV can be varied.

Another, unrelated, example of application that justifies the use of multiple autonomous vehicles was the Intel show at the 2018 Winter Olympics, where 1200 drones were used to put on a light show in the night sky. Each drone was mounted with a light bulb and, together, formed different shapes in the sky.

Complex systems like the two previously presented have in common the ability to properly plan motion for each vehicle that satisfies certain criteria such as simultaneous arrival of each vehicle in a formation while minimizing spent energy or time and avoiding collisions between vehicles and the environment.

Over the past decades, many approaches to solve motion planning problems have been proposed. Examples include bug algorithms, randomised algorithms such as PRM, RRT, RRT*, cell decomposition methods, graph-based approaches, planners based on learning, and methods based on optimal control formulations. Each technique has different advantages and disadvantages, and is best-suited for certain types of problems. Trajectory generation based on optimal control formulations stands out as particularly suitable for applications that require the trajectories to minimize (or maximize) some cost function while satisfying a complex set of vehicle and problem constraints. Finding closed-form solutions for Optimal Control problems can be difficult or even impossible to solve, and therefore numerical methods must be sought. Numerical methods can be divided into Direct and Indirect Methods. A numerical method for solving such complex Optimal Control problems consists of optimising trajectories parameterized by Bernstein polynomials [2]. Recent pioneering work on Direct Methods based on Bernstein polynomials shows interesting results [3]; specifically, they can solve complex problems with a high number of vehicles while keeping computational time relatively low. The use of these methods will be further explored here.

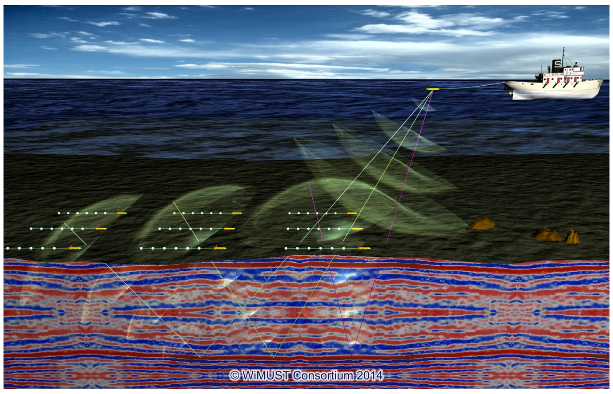


Figure 1.1: Artist's rendition of the WiMUST system for sub-bottom acoustic profiling with source-receiver decoupling

1.2 Background

The work discussed in this thesis focuses on motion planning for multiple cooperative vehicles. Motion Planning, as the name suggests, involves in planning of the motion of robots, such as mobile vehicles or robotic arms. Trajectory generation based on optimal control formulations are used, specifically, to solve the motion planning problems.

A trajectory is a time parameterized set of states of a dynamical system. These states can be position, their derivatives, and heading, among others. The states and inputs are related to each other by a set of dynamic equations. When dealing with multiple vehicles, the trajectory optimization problem takes into account the union of states and inputs of each vehicle such that the solution describes trajectories for all of the vehicles simultaneously.

As discussed in the previous section, numerical solutions to the trajectory generation problems must be sought. The numerical methods can be grouped into either Direct Methods or Indirect Methods.

Indirect Methods "use the necessary conditions of optimality of an infinite problem to derive a boundary value problem in ordinary differential equations" [4], the solutions of which must be found using analytic or numerical methods.

Direct Methods, on the other hand, are based on transcribing infinite optimal control problems into finite-dimensional Non-linear Problems (NLPs) using some kind of paramerisation (e.g., polynomial approximation or piecewise constant parameterization). They can be solved using ready-to-use NLP solvers (e.g. MATLAB) and do not require the computation of co-state and adjoint variables as Indirect

Methods do.

The focus on the work of this project is on the use of Direct Methods. Several parameterization methods are explored, such as piece-wise constants inputs, polynomials, and the use of Bernstein polynomials.

When solving an optimization problem, a model must be chosen for each vehicle. Different models are defined by different numbers of states and inputs and different dynamic equations. The choice of model will affect the complexity of the optimization problem. However, if the model is too simple, it may not accurately describe the real vehicle's dynamics. The Direct Methods that are tested focus on two models, the Medusa model [5] or the simpler Dubin's car model [6].

1.3 Objectives

The work presented in this thesis focuses on the use of Direct Methods.

There are several Direct Methods for trajectory optimization, for example, single and multiple shooting, collocation and quadratic programming. However, polynomial methods based on Bezier curves are particularly advantageous because they have favourable geometric properties which allow the efficient computation of the minimum distance between trajectories. As the complexity of the polynomials increases, the solutions converge to the optimal. [3]

An Optimal Control Problem's cost can be constructed based on several criteria such as consumed energy. For *cooperative* motion planning, the cost will have to be constructed differently because it will have to take into account the motion of the multiple vehicles at once, in particular, possible inter-vehicle collision.

The objectives of this work include

- testing of methods for obstacle avoidance;
- comparison of different parameterization methodologies;
- analysis of the complexity of increasing order and number of vehicles;
- viability testing for non differentially flat systems;
- testing of the use of log barrier functions for improved computation time.

1.4 Thesis Outline

This thesis is organized as follows. In Chapter 2, a number of direct optimization methods are presented, together with a simple motivating example that compares these methods. Chapter 3 introduces the two autonomous marine vehicle models that were used along with the simplifications adopted appropriate for the motion planning algorithms that were developed. Chapter 4 presents a formulation of the motion planning problem in the form of an equivalent optimization problem and the mathematical tools to solve it. Some results for particular motion planning problems are presented in chapter 5. Finally, chapter 6 summarizes the main achievements of the thesis and discusses topics that warrant future work.

2

Existing Trajectory optimization Methods for Motion Planning

Contents

In this chapter, Direct Methods for trajectory optimization will be reviewed. These methods are generic, i.e., they can be applied to a wide range of dynamic systems. Here, these methods are applied to the control of a vehicle in both one and two dimensions. A good understanding of motion planning for single vehicles will be necessary before extending to multiple vehicles.

2.1 The Optimization Problem

Motion planning for a single vehicle can be cast in the form of an optimal control problem of the form (see [4])

$$\begin{aligned}
\underset{x(.),u(.)}{\text{minimise}} \quad & J = \int_0^T L(x(t),u(t))dt + \Psi(x(T))\, dt & & \\
\text{subject to} \quad & x(0) = x_0, & & \\
& \dot{x} = f(x(t),u(t)), & & t \in [0,T] \\
& h(x(t),u(t)) \geq 0, & & t \in [0,T] \\
& r(x(T)) = 0 & &
\end{aligned} \tag{2.1}$$

where $x(t)$ is defined as $x : [0,T] \to \mathbb{R}^{n_x}$, $u(t)$ is defined as $u : [0,T] \to \mathbb{R}^{n_u}$, $f(x,t)$ is the system of equations for the dynamics, defined as $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, x_0 is the initial state, $h(x(t),\, u(t))$ represents the path constraints, $r(x(T)) = 0$ the terminal constraints, $L(x(t),u(t))$ is the running cost, defined as $L : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$, $\Psi(x(t))$ is the terminal cost defined as $\Psi : \mathbb{R}^{n_x} \to \mathbb{R}$ and finally T is known as the time "horizon", i. e., no matter what motion planning algorithm is used, the produced control law will only be valid within time $t \in [0;T]$.

An example of a path constraint is to keep the minimum distance to a known obstacle bounded below by a desired safety distance. If x encodes the position then $h(x)$ could be

$$h(x(t)) = \min_{t\in[0,T]} \|x - p_0\| - d_{\text{min}} \tag{2.2}$$

where p_0 is the location of an obstacle's centre of mass and d_{min} is the minimum accepted distance to that centre.

All of the available Direct Methods used in this Thesis have as an objective the formulation of a control law that optimises some form of the above cost function. These methods will be demonstrated with the control of a double integrator in 1-D and 2-D. A double integrator is a linear system where the second derivative of the position variable y, is the acceleration a, that is,

$$\ddot{y} = a \tag{2.3}$$

A general linear system is represented, in state-space form, by

$$\dot{x} = Ax + Bu \tag{2.4}$$

where $x = [x_1, \ldots, x_{n_x}]^T$ is the state of the system and $u = [u_1, \ldots, u_{n_u}]^T$ is the input. For the double integrator, the states and inputs are given by

$$\begin{cases} x_1 = y \\ x_2 = v \\ u = a \end{cases} \tag{2.5}$$

where x_1 and x_2 represent position and velocity, respectively, and u is the acceleration. The system's dynamics then become

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = u \end{cases} \tag{2.6}$$

or

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{2.7}$$

in matrix form, which means matrices A and B of equation 2.4 are given by

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.8}$$

.

The cost function 2.1 will be simplified to

$$J = \int_0^\infty x_1^2 + x_2^2 + \rho u^2 \; dt \tag{2.9}$$

where ρ is a scalar that penalizes "energy" consumption.

This dynamic system is diferentially flat (see section 2.8) with x_1 as the flat output, because it is possible to express x_2 as a derivative of x_1 and u as a double derivative of x_1.

Although this 1-D model is very simple, it can provide insight into real life applications. One obvious example is a train that follows a track. It cannot leave the track and so, environmental constraints, such as obstacles, cannot be fixed in time if they are between the train's initial and target position. A non-fixed constraint could be another train that is joining the same track, therefore, the first train cannot be at certain points at certain times.

The optimization problem (2.1), is infinite dimensional because the solution will be a function of time, which has an infinite number of points. The following Direct Methods will replace the infinite number of points of the function by approximated functions which are parameterized by a finite number of variables.

Before presenting the Direct Methods, results of the application of the Linear Quadratic Regulator are presented. This is important as the results serve as baselines to compare with the Direct Methods. The Linear Quadratic Regulator (LQR) is a feedback law which guarantees minimal cost for certain kinds of cost functionals.

2.2 Linear Quadratic Regulator

A Linear Quadratic Regulator is a technique applicable to linear dynamic systems of the form

$$\dot{x} = Ax + B \tag{2.10}$$

using the same definitions as in section 2.1.

The Linear Quadratic Regulator algorithm yields, under certain conditions, an appropriate state-feedback LQR controller that minimises a cost function of the form

$$J = \int_0^\infty x^T Qx + u^T Ru + 2x^T Nudt \tag{2.11}$$

The control law is of the form

$$u = -Kx \tag{2.12}$$

where K is given by

$$K = R^{-1}(B^T P(t) + N^T) \tag{2.13}$$

where $P(t)$ is the solution of the differential Riccati equation [7]

$$A^T P(t) + P(t)A - (P(t)B + N)R^{-1}(B^T P(t) + N^T) + Q = -\overline{P}(t) \tag{2.14}$$

with appropriate boundary conditions.

For the situation where the time horizon T is ∞, $P(t)$ will tend to a constant solution, P, and, as a result, P is found by solving the continuous time algebraic Riccati equation

$$A^T P + PA - (PB + N)R^{-1}(B^T P + N^T) + Q = 0 \tag{2.15}$$

The solution provided by this algorithm will be optimal and unique if the following conditions are fulfilled:

- (A, B) is controllable;
- for a system output $y = Cx$, with $Q = C^T C$, the pair (A, C) is observable;
- $R > 0$ and $Q > 0$.

Control via LQR is in closed-loop form, which has the advantage of being robust against parameter uncertainty and reducing the effect of external disturbances.

In order to visualise the resulting controlled double integrator evolves over time, the feedback law (2.12) is fed into the system (2.10), non zero initial conditions are provided, and the resulting Initial Value Problem (IVP) is solved. In other words, the objective is to find the soltion to

$$\begin{aligned} &\dot{x} = (A - BK)x \\ &\text{subject to} \quad x(0) = x_0 \end{aligned} \tag{2.16}$$

where x_0 is the initial conditions.

Here an example of control of the double integrator is presented in the context of linear quadratic regulator theory. Here, the initial conditions, which will be the same for all the subsequent 1-dimensional examples throughout this chapter, are given by

$$x_0 = \begin{bmatrix} x_{1_0} \\ x_{2_0} \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \tag{2.17}$$

Figure 2.1 shows the of an LQR controlled double integrator. The regulator used for this example was obtained with $Q = I$ and $R = \rho = 0.1$ as weights to obtain K in equation 2.13 so that the resulting cost matches 2.9.

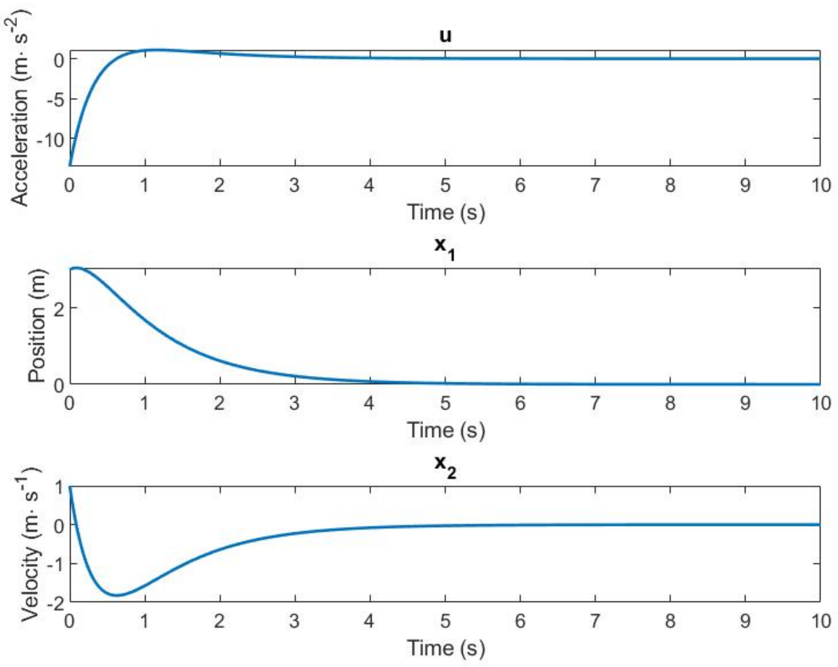


Figure 2.1: LQR-Controlled Double Integrator Solution

It can be seen that all variables tend asymptotically to zero. It should be noted that after around 6 seconds the system can already be considered stationary.

2.3 Single Shooting

Single Shooting is a direct method for motion planning. It consists of parameterising, for a given system, the control input signal $u(t)$ as a piece-wise constant function of time, as defined by

$$u(t) = q_i, \qquad t \in [t_i, t_{i+1}] \tag{2.18}$$

where q_i has the dimensions of the input, and the intervals $[t_i, t_{i+1}]$ denote the time-intervals between instants indexed by indices i and $i+1$.

The following step consists of solving the following Ordinary Differential Equation (ODE):

$$\begin{aligned} &x(0) = x_0, \\ &\dot{x}(t) = f(x(t), u(t; q)), \quad t \in [0, T]. \end{aligned} \tag{2.19}$$

to obtain the time evolution of x.

With the obtained trajectories, the optimization problem (2.1) can be extended and expressed as a function of the parameters $q = (q_0, q_1, \dots, q_{N-1})$ as optimization variables. The new optimization problem is written in the form

$$\begin{aligned} \underset{q}{\text{minimise}} \quad & \int_0^T L(x(t;q), u(t;q))dt + \Psi(x(T;q))\ dt \\ \text{subject to} \quad & x(0) = x_0, \\ & \dot{x} = f(x(t), u(t)), && t \in [0, T] \\ & h(x(t), u(t)) \geq 0, && t \in [0, T] \\ & r(x(T)) = 0 \end{aligned} \tag{2.20}$$

This method can produce good results when a sufficiently low step duration is used. However, an important concern when compared to other methods is the computation time. Convergence for a good solution takes a long time because the number of parameters that are used is relatively large. With an increasing number of search parameters comes a increasingly bigger complexity and, because the cost can only be calculated for the completed trajectory, calculating the gradients with respect to each parameter can be very difficult.

Trajectory planning for the double integrator with single shooting will be unconstrained. This is because the solution produced by the optimization problem will shape u, which will have no influence on the problem's initial conditions.

The solution for this unconstrained problem can be found using the Matlab function `fminsearch`.

During the optimization process, the variables x_1 and x_2 will be necessary for calculation of the cost. The integral 2.9 can be easily calculated without ever having to obtain an explicit expression for signals x_1 and x_2. This is because x_2 will be continuous piece-wise straight lines (by integration of constants) and x_1 will be continuous piece-wise parabolas (by integration of straight lines).

Figure 2.2 shows the solution of this shooting problem. The input u has a total of 30 coefficients, each having a duration of 20 ms.

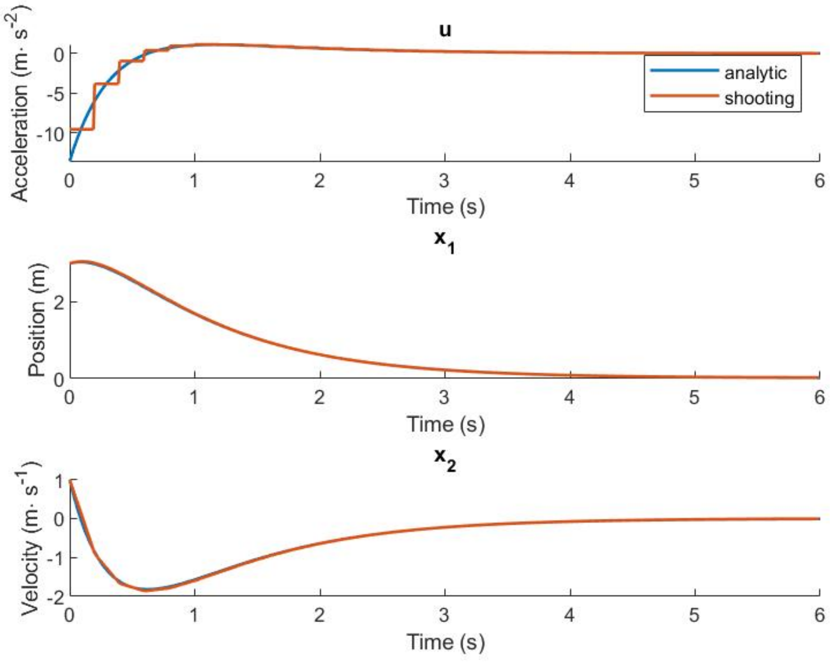


Figure 2.2: Direct optimization via a piecewise constant input.
Computation time: 2.44 s

2.4 Multiple Shooting

With multiple shooting, u is parameterized in the same way as single shooting (see 2.3). The main difference between multiple shooting and single shooting is that the ODE is calculated for each time interval separately. This ODE uses as initial value s_i and is expressed as

$$\dot{x}_i(t; s_i; q_i) = f(x_i(t; s_i, q_i), q_i), \qquad t \in [t_i; t_{i+1}] \tag{2.21}$$

Continuity of state x for each time-step has to be assured. As a result, an equality constraint has to be added to the problem. This constraint is given by

$$s_{i+1} = x_i(t_{i+1}, s_i, q_i), \qquad i = 0, 1, \ldots, N-1 \tag{2.22}$$

where the path cost l_i can now be calculated for each time interval $t \in [t_i; t_{i+1}]$ and is given by

$$l_i(s_i; q_i) = \int_{t_i}^{t_{i+1}} L(x(t; s_i; q_i), q_i) dt \tag{2.23}$$

The optimization problem can now be redefined as

$$\begin{aligned} \underset{q,s}{\text{minimise}} \quad & \sum_{i=0}^{N-1} l_i(s_i, q_i) + \psi(s_N) \\ \text{subject to} \quad & a(0) = x_0, \\ & s_{i+1} = x_i(t_{i+1}, s_i, q_i), \qquad i = 0, 1, \ldots, N-1, \\ & h(s_i, q_i) \geq 0, \qquad\qquad\qquad 1, \ldots, N, \\ & r(x(T)) = 0 \end{aligned} \tag{2.24}$$

For a solution u to be obtained with the same duration and order as in single shooting, the optimising algorithm will have to search through twice the amount of variables (an s_i for every q_i). However, the larger search space will be sparse due to the presence of constraints, and, as a result, easier to solve, compared to the small and dense optimization problem produced by single shooting [8].

Solutions for multiple shooting will be identical to single shooting and will differ only in computation time. As a result, examples with this method will not be given.

2.5 Quadratic Programming

Quadratic programming problems [9] have the form

$$\begin{aligned} \underset{\overline{x}(.)}{\text{minimise}} \quad & \frac{1}{2}\overline{x}^T H \overline{x} + f^T \\ \text{subject to} \quad & A \cdot \overline{x} \leq b, \\ & Aeq \cdot \overline{x} = beq, \\ & lb \leq \overline{x} \leq ub \end{aligned} \tag{2.25}$$

and are not specifically designed to tackle motion planning problems. The first step in applying quadratic programming to optimal control problems is to discretise the system's dynamics.

A discrete linear system can be represented as

$$\phi_{x_k} + \Gamma_{u_k} - x_{k+1} = 0, \tag{2.26}$$

where x_k and u_k are the state and input, respectively, at discrete time k, and Φ and Γ are state and input matrices, respectively of appropriate dimensions.

The cost function to optimise that is equivalent to 2.11 is given by

$$J_d = x_N^T Q x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k \tag{2.27}$$

where the matrices Q and R will be identical to the continuous time problem.

The goal of the quadratic programming optimiser is to obtain values x_k and u_k for all discrete-times that minimise 2.27. In order to formulate the optimization problem in the form of 2.25, all of the variables to optimise have to be flattened to the column vector

$$\overline{x} = \begin{bmatrix} x_1^0 \dots x_n^0 & u_1^0 \dots u_m^0 & (\dots) & x_1^{N-1} \dots x_n^{N-1} & u_1^{N-1} \dots u_m^{N-1} & x_1^N \dots x_n^N \end{bmatrix}^T, \tag{2.28}$$

where the superscript is the iteration number that goes from 0 to N, n is the dimension of x and m is the dimension of u. There will be a total of $(N-1)(n+m)+n$ variables to optimise.

To optimise 2.27, matrix H will be constructed by repetition of matrices Q and R as follows

$$H = \begin{bmatrix} [Q] & & & & & \\ & [R] & & & & \\ & & \ddots & & & \\ & & & [Q] & & \\ & & & & [R] & \\ & & & & & [Q] \end{bmatrix} \tag{2.29}$$

The system dynamics 2.26 will impose ”dynamic” linear constraints given by

$$\underbrace{\begin{bmatrix} [\phi] & [\Gamma] & [-I] & & \dots & \\ & & [\Phi] & [\Gamma] & [-I] & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}}_{\text{Aeq (dynamics)}} \overline{x} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\text{beq dynamics}} \tag{2.30}$$

where I is the identity matrix with size equal to the dimension of x.

The first and last samples of x will be restricted to the initial and final conditions, x_i and x_f

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \end{bmatrix}}_{\text{Aeq (initial state)}} \overline{x} = \underbrace{\mathbf{x}^i}_{\text{beq (initial state)}} \tag{2.31}$$

$$\underbrace{\begin{bmatrix} 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & 1 \end{bmatrix}}_{\text{Aeq (final state)}} \overline{x} = \underbrace{\mathbf{x}^f}_{\text{beq (final state)}} \tag{2.32}$$

The final matrices Aeq and beq that will be inserted into in the optimization software will be given by

$$\underbrace{\begin{bmatrix} \text{Aeq (dynamics)} \\ \text{Aeq (initial state)} \\ \text{Aeq (final state)} \end{bmatrix}}_{\text{Aeq}} \overline{x} = \underbrace{\begin{bmatrix} \vec{0} \\ x^i \\ x^f \end{bmatrix}}_{\text{beq}} \tag{2.33}$$

An additional inequality constraint may be used to bound values of $\overline{x}$. These are coded in the lb and ub vectors which correspond to the lower bound and upper bounds for $\overline{x}$. Later, in the application examples, this will be used to bound the values of u.

In order to control the double integrator using quadratic programming techniques, the system's discrete-time equivalent will have to be obtained first. Matrices Φ and Γ can be obtained via the Matlab function `c2d`. H will be as described with Q and R identical to the analytical solutions; f will be zero because there are no non-squared components of the cost function. Figure 2.3 shows the solution of the quadratic problem for an unconstrained input and identical initial conditions to the analytical example were used.

Figure 2.4 shows the solution to a problem where u is bounded by

$$\begin{aligned} ub &= U_{\text{max}} \begin{bmatrix} \infty & \infty & 1 & \dots & \infty & \infty & 1 & \infty & \infty \end{bmatrix} \\ lb &= U_{\text{min}} \begin{bmatrix} \infty & \infty & 1 & \dots & \infty & \infty & 1 & \infty & \infty \end{bmatrix} \end{aligned} \tag{2.34}$$

where U_{max} and U_{min} are given are given by ± 1. Here we can see a clear saturation on the input up to time 2.5 s after which the system evolves like in the unconstrained example.

2.6 Monomial Polynomials

Another way to describe trajectories is by using polynomials. If an optimal trajectory can be modelled by a polynomial, then it can be parameterized in terms of the polynomial's coefficients. As a result, the optimization problem tends to have a much smaller search space compared to, for example, shooting methods because with a relatively low order of polynomial, a whole trajectory over time $t \in [0, T]$ can be obtained will a relatively good cost.

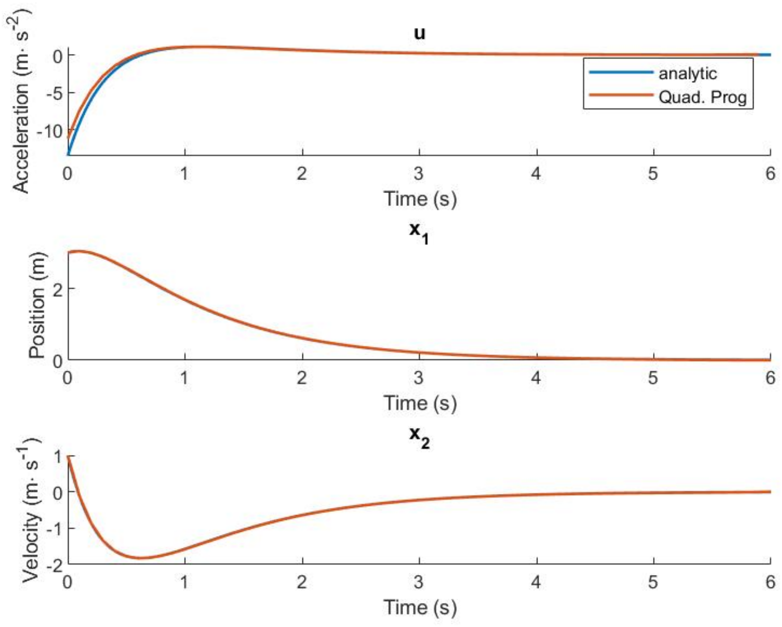


Figure 2.3: Quadratic Programming solution

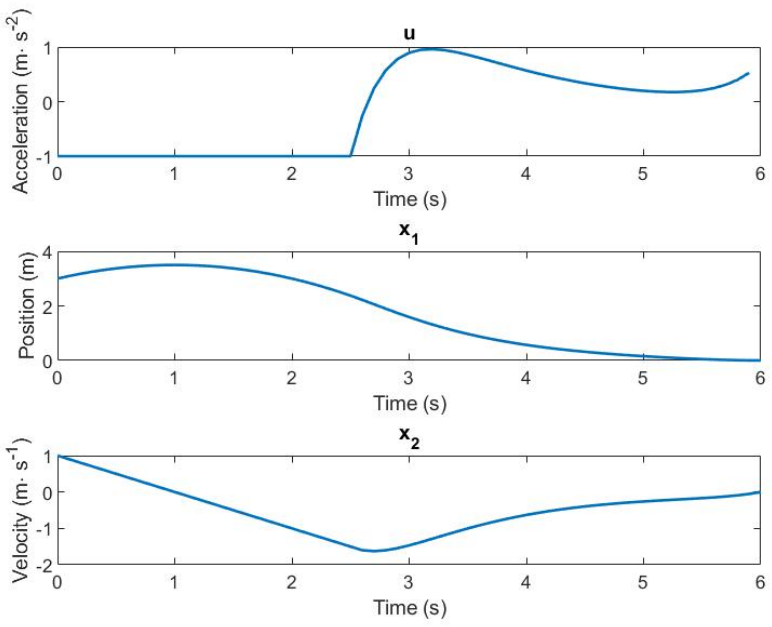


Figure 2.4: Quadratic Programming solution with bounded input

Polynomial methods are especially suited to deal with *differentially flat systems* (see section 2.8). In differentially flat systems, the state and input variables can be directly expressed, without integrating any differential equation, in terms of the flat output and a finite number of its derivatives [10]. This means that with coefficients that describe just the flat output it is possible to describe all of the other variables of the system, thus resulting in a

The solution provided by the Matlab function `fmincon` for the same double integrator problem of section 2.2 with this method will be a polynomial $\overline{p}$ for x_1. Once the optimal coefficients have been obtained the trajectories for x_1, x_2 and u can be obtained by polynomial derivation and evaluation. This is possible because the system is differentially flat. These polynomials are related in the system of equations

$$\begin{cases} u(t) = a_0 + a_1 t + a_2 t^2 + \ (\dots) \\ x_2(t) = v_0 + a_0 t + \frac{a_1}{2} t^2 + \frac{a_2}{3} t^3 + \ (\dots) \\ x_1(t) = p_0 + v_0 t + \frac{a_0}{2} t^2 + \frac{a_1}{6} t^3 + \frac{a_2}{12} t^4 + \ (\dots) \end{cases} \tag{2.35}$$

The cost function will be the same as the one used for the analytic method, and with the same initial conditions (see 2.9 and 2.17).

Figure 2.5 shows the solution of a polynomial optimization problem for a polynomial of order 7. Higher orders where attempted but take too long to converge.

In order to guarantee the initial and final conditions of the state variable x, the linear constraint that has to be introduced is given by

$$\underbrace{\begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ T^N & T^{N-1} & \dots & T & 0 \end{bmatrix}}_{\text{Aeq}} \overline{p} = \underbrace{\begin{bmatrix} x_1^i \\ x_2^i \\ 0 \end{bmatrix}}_{\text{beq}} \tag{2.36}$$

2.7 Bezier Curves

A Bezier curve is a parametric curve used in computer graphics and related fields. The curve, is named after Pierre Bezier, who used it in the 1960s for designing curves for the bodywork of Renault cars [11]. Other uses include the design of computer fonts and animation [11]. The mathematical basis for Bezier curves, the Bernstein polynomials, was established in 1912, but the polynomials were not applied to graphics until some 50 years later when mathematician Paul de Casteljau in 1959 developed de Casteljau's algorithm, a numerically stable method for evaluating the curves, and became the first to apply them to computer-aided design at French auto-maker Citroen [12].

What follows is a brief summary of the type of Bernstein polynomials that are exploited in this study.

A Bernstein polynomial is given by

$$P(\tau) = \sum_{k=0}^{n} \overline{p}_k B_{k,n}(\tau), \qquad \tau \in [0,1] \tag{2.37}$$

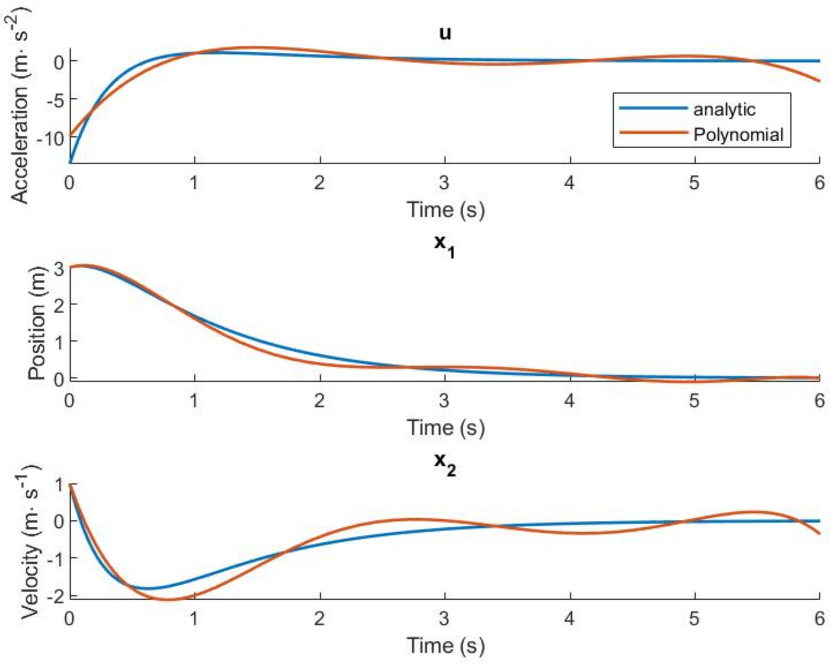


Figure 2.5: Direct optimization with Polynomials.
Polynomial order: 7.
Computation time: 1.33 s

where n is the order of the polynomial and τ is the time contraction of t, given by

$$\tau = \frac{t}{T}, \qquad t \in [0, T]. \tag{2.38}$$

The p_k are the polynomial coefficients or *control points*, and b_k^n are the *Bernstein Basis*, given by

$$B_k^n(\tau) = \binom{n}{k}(1-\tau)^{n-k}\tau^k \tag{2.39}$$

As a result, a Bernstein polynomial of order n is a linear combination of $(n+1)$ Bernstein basis with weights given by p_k. The Bernstein Basis, as a function of τ, for 6 different orders, n, are shown in figure 2.6.

For a polynomial of order n, the $i \in 0..n$ control points can be represented as a vector on a vector p.

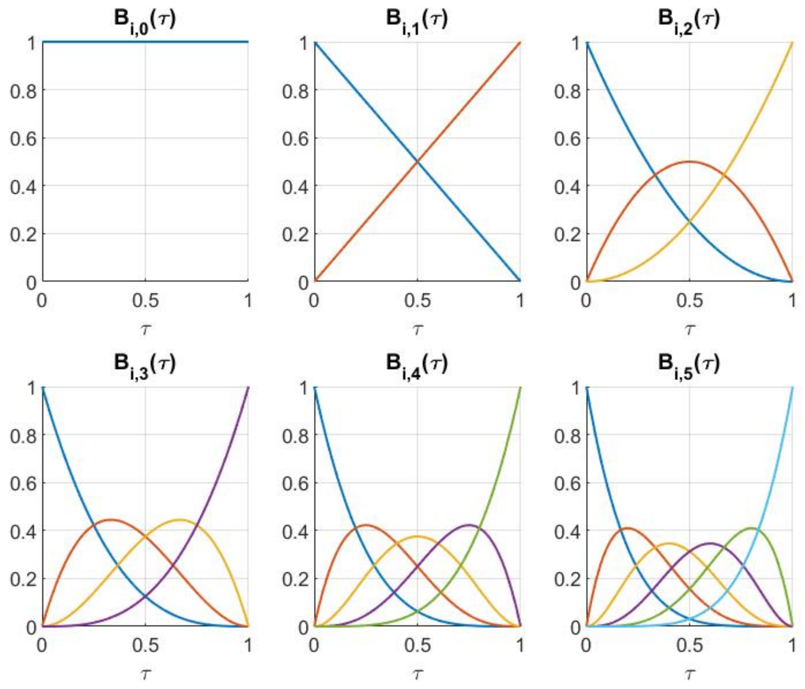


Figure 2.6: Representation of the Bernstein Basis for $\tau \in [0, 1]$ for orders $n = 0 \cdots 5$

For example, if p is given by

$$\overline{p} = \begin{bmatrix} 0 \\ 0.5 \\ 1 \\ 0.7 \\ 0.3 \\ -0.7 \\ -1 \\ -0.5 \\ -0.1 \end{bmatrix}, \tag{2.40}$$

the plot of the polynomial given by these coefficients is shown in figure 2.7. The control points on vector p are plotted along time in equidistant time nodes. What can be seen is that the control points "attract" the curve towards themselves.

Two or three Bernstein polynomials of the same order, i.e., same number of control points, can be plotted against each other within the time domain $\tau \in [0, 1]$. The control points of the curves together form $n + 1$ points in two or three dimensions. These two or three dimension polynomials are also known as Bezier curves. Figure 2.7 shows a 2-D Bezier Curve, the control points which resulted in it are plotted too. The control points for this curve had a specific order on the underlying Bernstein Polynomial. Changing the oder of the control points on the polynomial will, as a result, produce a different Bezier curve, such as the one in figure 2.9, despite the control points being the same on the 2-D plot.

One property than can be observed in figure 2.7 is that, within the domain $t \in [0, T]$, the polynomial

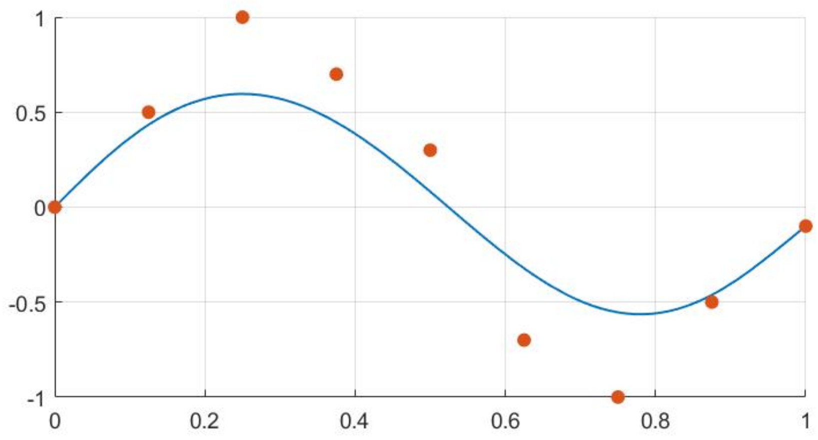


Figure 2.7: A Bernstein Polynomial

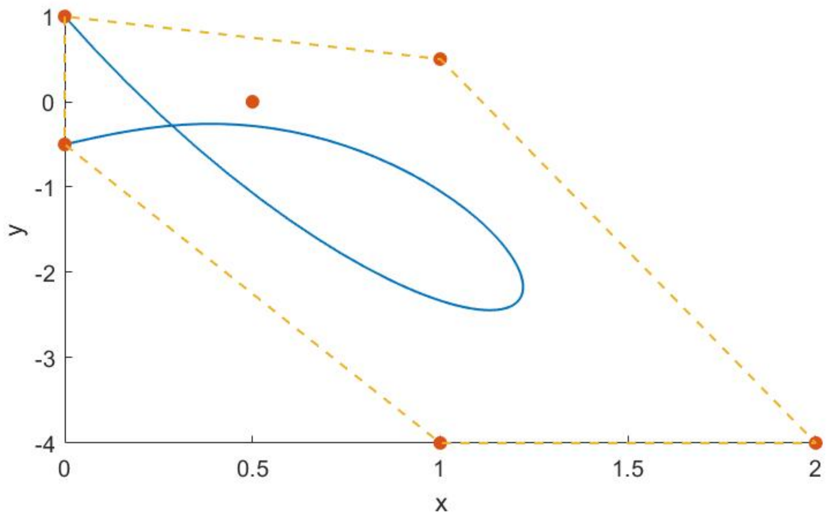


Figure 2.8: A 2-D Bernstein Polynomial

is limited by a convex hull [3] formed by the control points.

The initial and final values of $P(t)$ in the interval $[0, T]$ are given by

$$
\begin{aligned}
P(0) &= p_{n,0} \\
P(T) &= p_{n,n}
\end{aligned}
\tag{2.41}
$$

and the derivative and integral are given by

$$
\dot{P}(t) = \frac{n}{T} \sum_{k=0}^{n-1} (p_{k+1,n} - p_{k,n}) B_{k,n-1}(t) \tag{2.42}
$$

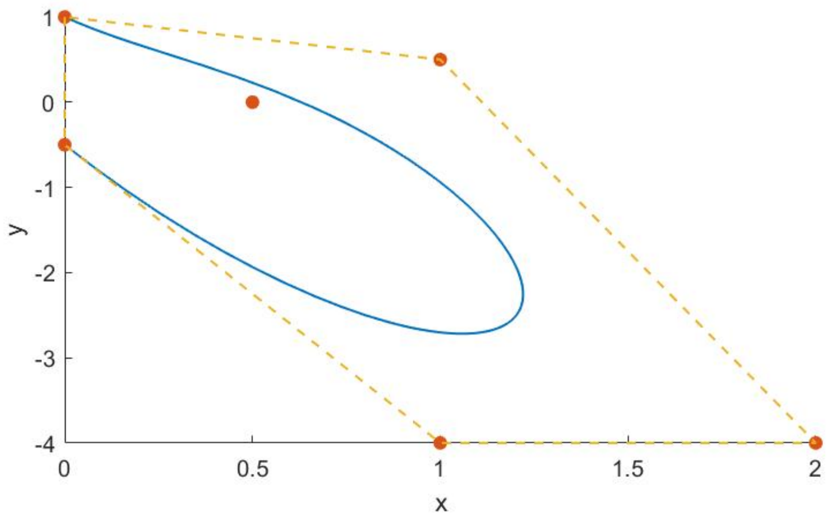


Figure 2.9: A 2-D Bernstein Polynomial with Reordered Control Points

$$\int_0^T P(t)dt = \frac{T}{n+1}\sum_{k=0}^{n} p_{k,n} \tag{2.43}$$

The control points for the derivative of a bernstein polynomial can be obtained my a multiplication of a derivation matrix by the control points of the original curve stored in the column vector, with the matrix given by

$$\boldsymbol{D}_{N-1} = \begin{bmatrix} -\frac{N}{T} & \frac{N}{T} & 0 & \dots & & 0 \\ 0 & -\frac{N}{T} & \frac{N}{T} & \dots & & 0 \\ \vdots & \ddots & \ddots & \ddots & & \\ 0 & & \dots & & -\frac{N}{T} & \frac{N}{T} \end{bmatrix} \in \mathbb{R}^{(N+1)\times N} \tag{2.44}$$

The control points of the Anti-Derivative/Primitive can be obtained similarly to the derivation by multiplying the control points to a primitive matrix [13], and adding a vector:

$$p = A \cdot \dot{p} + p_0 \tag{2.45}$$

where p_0 is the initial values of P and the matrix A is given by

$$\boldsymbol{A}_{N+1} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ \frac{T}{N+1} & 0 & \dots & 0 & 0 \\ \frac{T}{N+1} & \frac{T}{N+1} & \dots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \\ \frac{T}{N+1} & \frac{T}{N+1} & \dots & \frac{T}{N+1} & \frac{T}{N+1} \end{bmatrix} \tag{2.46}$$

A Bernstein polynomial of degree N and coefficients $p_{k,N} \in \mathbb{R}, k = 0, \dots, N$ can be expressed as a

Bernstein polynomial of degree M with $M > N$, with coefficients $p_{k,M} \in \mathbb{R}, k = 0, \ldots, M$ given by

$$p_{k,M} = \sum_{j=\max(0,k+M-N)}^{\min(N,k)} \frac{\binom{M-N}{k-j}\binom{N}{j}}{\binom{M}{k}} p_{j,N} \in \mathbb{R}^n, \tag{2.47}$$

which is known as degree elevation. In other words, it is possible find a polynomial with $M+1$ control points such that it's identical to a curve with order $N+1$ within time $t \in [0, T]$.

The control points for the degree elevation can also be obtained by multiplying the control points vector by a matrix $\boldsymbol{E}$ whose indices are given by

$$\boldsymbol{E}_{ij} = \frac{\binom{N}{j}\binom{M-N}{i-j}}{\binom{M}{i}} \tag{2.48}$$

Multiplication and addition is given by

$$f(t)g(t) = \sum_{i=0}^{m+n} \left(\sum_{i=\max(0,i-n)}^{\min(m,i)} \frac{\binom{m}{j}\binom{n}{i-j}}{\binom{m+n}{i}} f_{j,m} g_{i-j,n} \right) B_{i,m+n}(t) \tag{2.49}$$

$$f(t) \pm g(t) = \sum_{i=0}^{m} \left(f_{i,n} \pm \sum_{j=\max(0,i-m+n)}^{\min(n,i)} \frac{\binom{n}{j}\binom{m-n}{i-j}}{\binom{m}{i}} g_{j,n} \right) B_{i,m+n}(t) \tag{2.50}$$

The De Casteljau's algorithm [14] is a recursive method to evaluate polynomials in Bernstein form. A geometric interpretation of this algorithm presented as follows:

1. Connect the consecutive control points in order to create the control polygon of the curve;
2. Subdivide each line segment of this polygon with the ratio $t : (1-t)$ and connect the obtained points which results in a new polygon with one fewer control points;
3. Repeat the process until a single point is achieved – this is the point of the curve corresponding to the parameter t.

Figure 2.10 illustrates the breakdown of the control points into polygons and sub polygons.

The evaluation of the curve can be carried out analytically. For a Bernstein Polynomial $\boldsymbol{x}_N : [0, t_f] \to \mathbb{R}^n$, and a scaler $t_{div} \in [0, t_f]$, the Bernstein polynomial at t_{div} can be computed using the following recursive relation:

$$\begin{aligned} \overline{\boldsymbol{x}}_{i,N}^{[0]} &= \overline{\boldsymbol{x}}_{i,N}, \quad i = 0, \ldots, N \\ \overline{\boldsymbol{x}}_{i,N}^{[j]} &= \overline{\boldsymbol{x}}_{i,N}^{[j-1]} \frac{t_f - t_{div}}{t_f} + \overline{\boldsymbol{x}}_{i+1,N}^{[j-1]} \frac{t_{div}}{t_f} \end{aligned} \tag{2.51}$$

with $i = 0, \ldots, N-j$, and $j = 1, \ldots, N$. Then, the Bernstein polynomial evaluated at t_{div} is given by

$$\overline{\boldsymbol{x}}_N(t_{div}) = \overline{\boldsymbol{x}}_{0,N}^{[N]}. \tag{2.52}$$

Moreover, the Bernstein polynomial can be subdivided at t_{div} into two Nth order Bernstein polynomial with Bernstein coefficients

$$\overline{\boldsymbol{x}}_{0,N}^{[0]}, \overline{\boldsymbol{x}}_{0,N}^{[1]}, \ldots, \overline{\boldsymbol{x}}_{0,N}^{[N]}, \quad \text{and} \quad \overline{\boldsymbol{x}}_{0,N}^{[N]}, \overline{\boldsymbol{x}}_{0,N}^{[N-1]}, \ldots, \overline{\boldsymbol{x}}_{0,N}^{[0]}. \tag{2.53}$$

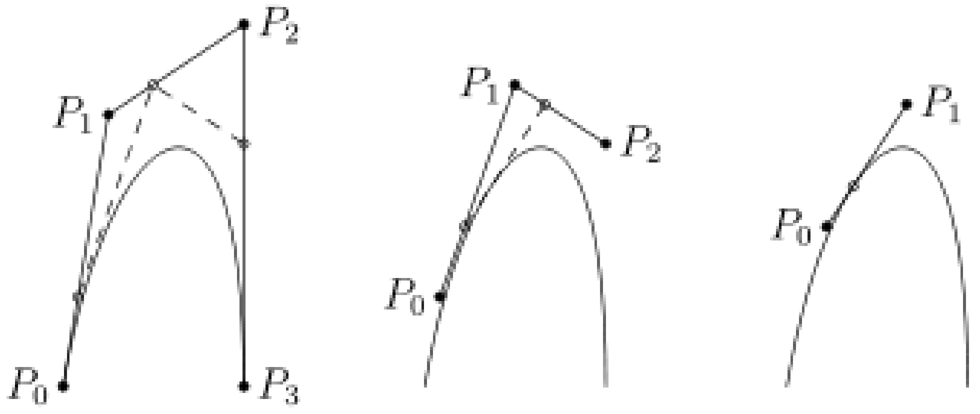


Figure 2.10: Visual representation of the De Casteljau algorithm

The convex hull property of these polynomials will prove useful in an algorithm that calculates distance between curves, which will be used in deconfliction of trajectories in multiple vehicle control. By just knowing the position of the control points in the space, it is possible to guarantee constraint satisfaction in the whole trajectory and not just in the control points.

Just as with monomial polynomials, when these Bernstein polynomials are applied to differentially flat systems, a reduction of the number of parameters to optimise is possible because with just a flat output, all state variables can be derived. If the system is not differentially flat, then parameters for all state variables need to be defined and optimised but will be subject to the constrained imposed by the system's dynamics.

The solution obtained by the Matlab function `fmincon` for the same double integrator problem of section 2.2 with this Bernstein polynomial based approach will provide the coefficients $\overline{p}$ for the Bernstein polynomial of state variable x_1.

Figure 2.11 shows the solution for the same quadratic optimization problem and with the same initial conditions as the previous sections. The red circles in the figure show the distribution of the obtained coefficients. The resulting polynomial has order 14.

In order to guarantee that the initial and final conditions are respected, the linear constraint that has to be introduced is given by

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ \frac{n}{T} & -\frac{n}{T} & 0 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix}}_{\text{Aeq}_{\text{1D}}} \overline{p} = \underbrace{\begin{bmatrix} x_0^i \\ x_1^i \\ 0 \end{bmatrix}}_{\text{beq}_{\text{1D}}} \tag{2.54}$$

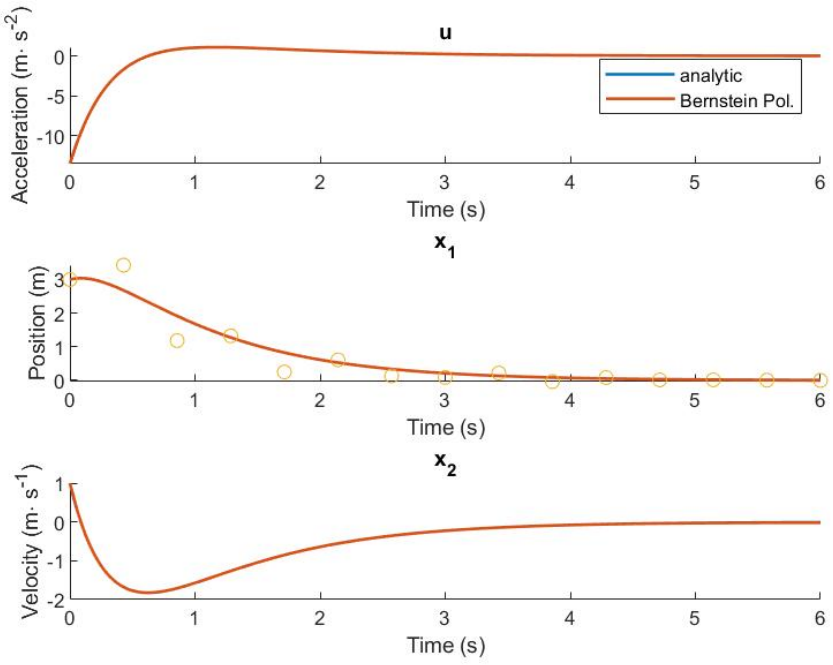


Figure 2.11: Direct Bernstein Solution

2.8 Differentially Flat Systems

A nonlinear system of the form

$$\dot{x} = f(x(t), u(t)), \quad x(t) \in \mathbb{R}^n \tag{2.55}$$

is said to be *differentially flat* [10] or simply *flat* if there exists a set of variables $y = (y_1, \ldots, y_m)$ called the *flat output*, such that

- the components of y are not differentially related over $\mathbb{R}$;
- every system variable, state or input, may be expressed as a function of the components of y and a finite number of their time-derivatives, and
- every component of y may be expressed as a function of the system variables and of a finite number of their time-derivatives.

The flat output has, in general, a clear physical interpretation and captures the fundamental properties of a given system and its determination allows to simplify considerably the control design. This implies that there is a fictitious *flat output* that can explicitly express all states and inputs in terms of the flat output and a finite number of derivatives, thus reducing the number of parameters to be optimised.

2.9 Remarks

Out of all of the presented Direct Methods, the preferred is the Bernstein polynomials method because it provides fast solutions for the simple double integrator problem and, as will be seen in later chapters, will prove advantageous for differentially and non-differentially flat dynamic systems.

3 Autonomous Vehicle Models

Contents

In this chapter equations that rule the motion of an autonomous marine vehicle are derived. But first, the coordinate frames will be defined (3.1). Afterwards, the motion equations for the Medusa model are presented (3.2). Finally, the simpler Dubin's car model is presented (3.3).

This chapter will focus primarily on dynamics in a two dimensional space.

3.1 Reference Frames and Notation

To derive the equations of motion for a marine vehicle it is standard practice to define two coordinate frames; Earth-fixed inertial frame $\{U\}$ composed by the orthonormal axes $\{x_U, y_U, z_U\}$ and the body-fixed frame $\{B\}$ composed by the axes $\{x_B, y_B, z_B\}$, as indicated in Figure 3.1.

- x_B is the longitudinal axis (directed from the stern to fore)
- y_B is the transversal axis (directed from to starboard)
- z_B is the normal axis (directed from top to bottom)

To simplify the model equations, the origin of the body-fixed frame is normally chosen to coincide with the centre of mass of the vehicle. The motion control of $\{B\}$ (that corresponds to the motion of the vehicle) is described relative to the inertial frame $\{U\}$.

In general, six independent coordinate are necessary to determine the evolution of the position and orientation (six Degress of Freedom (DOF)), three position coordinates (x, y, z) and using the Euler orientation angles (ϕ, θ, ψ). The six motion components are defined as *surge*, *sway*, *heave*, *roll*, *pitch*, and *yaw*, and adopting the SNAME [1] notation it can be written as in the Table 3.1 or in a vectorial form:

- $\eta_1 = [x, y, z]^T$ - position of the origin of $\{B\}$ with respect to $\{U\}$
- $\eta_2 = [\phi, \theta, \psi]^T$ - orientation of $\{B\}$ with respect to $\{U\}$.
- $\nu_1 = [u, v, w]^T$ - linear velocity of the origin of $\{B\}$ relative to $\{U\}$, expressed in $\{B\}$.
- $\nu_2 = [p, q, r]^T$ - angular velocity of $\{B\}$ relative to $\{U\}$, expressed in $\{B\}$.
- $\tau_1 = [X, Y, Z]^T$ - actuating forces expressed in $\{B\}$.
- $\tau_2 = [K, M, N]^T$ - actuating moments expressed in $\{B\}$

In compact form, this yields

$$\begin{cases} \eta = [\eta_1^T, \eta_2^T]^T \\ \nu = [\nu_1^T, \nu_2^T]^T \\ \tau_{RB} = [\tau_1^T, \tau_2^T]^T \end{cases} \tag{3.1}$$

[1] The Society of Naval Architects & Marine Engineers - http://www.sname.org/

Degree of Freedom	Forces and moments	Linear and angular velocities	Position and Euler angles
1. Motion in the x-direction (surge)	X	u	x
2. Motion in the y-direction (sway)	Y	v	y
3. Motion in the z-direction (heave)	Z	w	z
4. Rotation about the x-axis (roll)	K	p	ϕ
5. rotation about the y-axis (pitch)	M	q	θ
6. Rotation about the z-axis (yaw)	N	r	ψ

Table 3.1: Notation used for marine vehicles

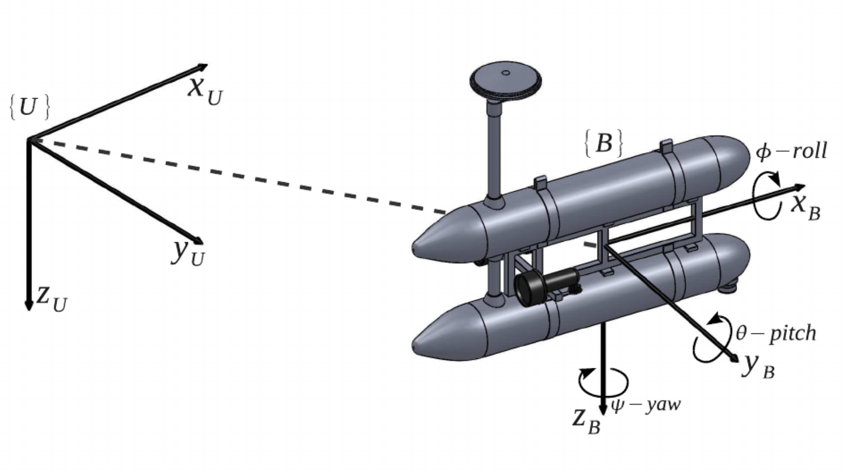


Figure 3.1: Coordinate frames

3.2 The Medusa Model

The kinematics involves only the geometrical aspects of motion, and relates the velocities with position. Using the coordinate frames notation in Section 3.1, the kinematic equation can be expressed as

$$\dot{\eta} = J(\eta)\nu \tag{3.2}$$

with

$$J(\eta) = \left[{}^{U}_{B}R\right] \tag{3.3}$$

where

$${}^{U}_{B}R(\Theta) = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix}, s\cdot = \sin(\cdot), c\cdot = \cos(\cdot) \tag{3.4}$$

is the rotation matrix from $\{B\}$ to $\{U\}$, [15], defined my means of three successive rotations (zyx-convention):

$${}^{U}_{B}R(\Theta) = R_{z,\psi}R_{y,\theta}R_{x,\phi} \tag{3.5}$$

and

$$T_\Theta(\Theta) = \begin{bmatrix} 1 & s\phi\ t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & c\phi/c\theta & c\phi/c\theta \end{bmatrix}, \theta \neq \pm 90 \tag{3.6}$$

is the Euler attitude transformation matrix that relates the body-fixed angular velocities (p, q, r) with the roll($\dot{\phi}$), pitch($\dot{\theta}$) and yaw($\dot{\psi}$) rates. Notice that $T_\Theta(\Theta)$ is not defined for the pitch angle $\theta = \pm 90$, resulting from using Euler angles to describe the vehicle's motion. To avoid this singularity, one possibility is to use a quaternion representation [15]. However, due to physical restrictions, the vehicle will operate far from this singularity ($\theta \approx 0$ and $\psi \approx 0$) which implies that we can use the Euler representation.

3.2.1 Dynamic Equations

Since the hydrodynamic forces and moments have simpler expressions if written in the body frame because they are generated by the relative motion between the body and the fluid, it is convenient to formulate Newton's law in $\{B\}$ frame. In that case, the rigid-body equation can be expressed as

$$M_{RB}\dot{\nu} + C_{RB}(\nu)\nu = \tau_{RB} \tag{3.7}$$

where M_{RB} is the rigid body inertia matrix, C_{RB} represents the Coriolis and centrifugal terms and τ_{RB} is a generalized vector of external forces and moments and can be decomposed as

$$\tau_{RB} = \tau + \tau_A + \tau_D + \tau_R + \tau_{dist} \tag{3.8}$$

where

1. τ - Vector of forces and torques due to thrusters/surfaces which usually can be viewed as the control input

2. τ_A + The force and moment vector due to the hydrodynamic added mass,

$$\tau_A = -M_A\dot{\nu} - C_A(\nu)\nu \tag{3.9}$$

3. τ_D - Hydrodynamics terms due to lift, drag, skin friction, etc.

$$\tau_D = -D(\nu)\nu \tag{3.10}$$

where $D(\nu)$ denotes the hydrodynamic damping matrix (positive definite).

4. τ_R - Restoring forces and torques due to gravity and fluid density,

$$\tau_R = -g(\eta) \tag{3.11}$$

5. τ_{dist} - External disturbances: waves, wind, etc.

Replacing (3.8) on (3.7), taking into account (3.9), (3.10), the dynamic equations can be written as

$$\underbrace{M_{RB}\dot{\nu} + C_{RB}(\nu)\nu}_{\text{rigid-body terms}} + \underbrace{M_A(\dot{\nu}) + C_A(\nu)\nu + D(\nu)\nu}_{\text{hydrodynamic terms}} + \underbrace{g(\nu)}_{\text{restoring terms}} = \tau + \tau_{dist} \tag{3.12}$$

which can be simplified to

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\nu) = \tau + \tau_{dist} \tag{3.13}$$

where $M = M_{RB} + M_a$, $C(\nu) = C_{RB}(\nu) + C_A(\nu)$.

3.2.2 Simplified Equations of Motion

This study will focus on movement along a 2-D plane, and so the dynamics and kinematics can be simplified such that there are only three degrees of freedom $[x, y, \psi]$.

The kinematics with take the simpler form

$$\begin{aligned} \dot{x} &= u\cos\psi - v\sin\psi, \\ \dot{y} &= u\sin\psi + v\cos\psi, \\ \dot{\psi} &= r. \end{aligned} \tag{3.14}$$

τ_u and τ_r are the external force in $surge$ (common mode) and the external torque about the z-axis (differential mode between thrusters), respectively, which can by obtained by

$$\begin{aligned} \tau_u &= F_{sb} + F_{ps}, \\ \tau_r &= l(F_{ps} - F_{sb}) \end{aligned} \tag{3.15}$$

where F_{sb} and F_{ps} are the starboard and port-side thruster forces, respectively, and l is the length of the arm with respect to the centre of mass.

By neglecting roll, pitch and heave motion, the equations for (u, v, r) without disturbances are

$$\begin{aligned} m_u\dot{u} - m_v vr + d_u u &= \tau_u, \\ m_v\dot{v} + m_u ur + d_v v &= 0, \\ m_r\dot{r} - m_{uv} uv + d_r r &= \tau_r, \end{aligned} \tag{3.16}$$

where

$$\begin{aligned} m_u &= m - X_{\dot{u}} \quad d_u = -X_u - X_{|u|u}|u| \\ m_v &= m - Y_{\dot{v}} \quad d_v = -Y_v - Y_{|v|v}|v| \\ m_r &= I_z - N_{\dot{r}} \quad d_r = -N_r - N_{|r|r}|r| \\ m_{uv} &= m_u - m_v \end{aligned} \tag{3.17}$$

All the previous equations are expressed without considering the influence of external factors like ocean currents. If a constant irrotational ocean current, v_c, is introduced, forming an angle $v = v_r + v_c$, where u_r and v_r are the components of the AMV velocity with respect to the current and u_c and v_c are the components of the ocean current velocity in the body frame. The previous dynamic equations (3.16) become

$$\begin{gathered} m_u\dot{u} - m_v(v_r + v_c)r + d_u u = \tau_u, \\ m_v\dot{v} + m_u(u_r + u_c)r + d_v v = 0, \\ m_r\dot{r} - m_{ub}(u_r + u_c)(v_r + v_c) + d_r r = \tau_r, \end{gathered} \tag{3.18}$$

where the expressions for masses and drags remain the same as in (3.17).

3.2.3 The ”Somewhat” Differentially Flat Property

If the states x, y and ψ are interpreted as flat outputs, then the Medusa model can be interpreted as differentially flat [13]. All other variables can be obtained with rearranging the kinematics and dynamics:

$$\begin{gathered} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} \Leftrightarrow \\ \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}^{-1} \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \Leftrightarrow \\ \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \end{gathered} \tag{3.19}$$

and the inputs from the dynamics:

$$\begin{gathered} \tau_u = m_u\dot{u} - m_v vr + d_u u, \\ \tau_r = m_r\dot{r} - m_{ub}uv + d_r r, \end{gathered} \tag{3.20}$$

These equations will always be valid, however, only for x, y and ψ that respect

$$m_v\dot{v} + m_u ur + d_v v = 0 \tag{3.21}$$

because this equation is what guarantees $\tau_v = 0$.

3.3 Dubin’s Car Model

The Medusa model can be even further simplified by a Dubin’s Car.

In geometry, the term Dubin’s path typically refers to the shortest curve that connects two points in the two-dimensional Euclidean plane (i.e. x-y plane) with a constraint on the curvature of the path and with prescribed initial and terminal tangents to the path, and an assumption that the vehicle travelling the path can only travel forward [6].

A Dubin's car is a simple model for a vehicle that transverses a Dubin's path.

The main difference compared with the Medusa Model is that this will does not possess the possibility of side slip, therefore, less variables and dynamics equations will be necessary which will translate to a smaller computation time.

A simple kinematic car model for the systems is:

$$\begin{aligned} \dot{x} &= u\cos\psi \\ \dot{y} &= u\sin\psi \\ \dot{\psi} &= r \end{aligned} \tag{3.22}$$

where (x, y) is the car's position, ψ is the car's heading, u is the car's speed, and r is the car's turn rate.

This model is differentially flat because for any continuous x and y, all of the other variables can be derived.

The tangent and rotational speeds can be obtained by

$$\begin{aligned} \begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} &= \begin{bmatrix}\cos(\psi) & -\sin(\psi)\\ \sin(\psi) & \cos(\psi)\end{bmatrix} \cdot \begin{bmatrix}u\\ 0\end{bmatrix} \Leftrightarrow \\ \begin{bmatrix}u\\ 0\end{bmatrix} &= \begin{bmatrix}\cos(\psi) & -\sin(\psi)\\ \sin(\psi) & \cos(\psi)\end{bmatrix}^{-1} \cdot \begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} \Leftrightarrow \\ \begin{bmatrix}u\\ 0\end{bmatrix} &= \begin{bmatrix}\cos(\psi) & \sin(\psi)\\ -\sin(\psi) & \cos(\psi)\end{bmatrix} \cdot \begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} \end{aligned} \tag{3.23}$$

where ψ can be obtained by integrating r. However, because r is not a flat output, ψ is derived as

$$\psi = \arctan\frac{\dot{y}}{\dot{x}} \tag{3.24}$$

Finally, tangent acceleration, τ_1 and torque, τ_2 can be derived from u and r:

$$\tau_1 = \dot{u} \tag{3.25}$$

$$\tau_2 = \dot{r} \tag{3.26}$$

which can be used as inputs to the system.

4

Cooperative Motion Planning: Problem Formulation and Adopted Solutions

Contents

In this chapter, implementation of a Direct Method based on the use of Bernstein polynomials is described. This chapter focuses on formalising the optimization problem for multiple vehicles. In chapter 5, solutions for particular optimization problems are presented based on the formulations of this chapter.

4.1 The Optimization Problem

The Motion Planning problem for multiple vehicles that are the focus of this project consists of a ”go-to” formation manoeuvre [16]. It consists of the simultaneous arrival of a formation of vehicles to desired locations whilst simultaneously avoiding collisions between each other and the environment.

Just as for optimization for a single vehicle, (see section 2.1) motion planning for multiple vehicles will also be based on the minimisation of an appropriate cost function. Here, however, the cost will be the result of the sum of the costs of each individual vehicle and an added constraint will be necessary that will take into account inter-vehicular collisions.

The optimal control formulation for this motion planning problem is defined as

$$\begin{aligned} \underset{x^{[i]}(.),u^{[i]}(.),i=1,\dots N_v}{\text{minimize}} \quad & \int_0^T \sum_{i=1}^{N_v} L_i(x^{[i]}(t), u^{[i]}(t))dt + \Psi(x^{[i]}(T)) \\ \text{subject to} \quad & x^{[i]}(0) = x_0^{[i]}, \\ & r^{[i]}(x^{[i]}(T)) = 0, \\ & \dot{x}^{[i]} = f^{[i]}(x^{[i]}(t), u^{[i]}(t)), \qquad t \in [0,T] \\ & h(x(t), u(t)) \geq 0, \end{aligned} \tag{4.1}$$

with the same definitions as in section 2.1 applied to each vehicle $i = 1 \dots N_v$ of of N_v vehicles but where $h(.)$ must also handle inter-vehicle constraints.

A problem that only has the integral term $\sum_{v=1}^{N_v} L_v(x^{(v)}(t), u^{(v)}(t))$ is said to be in *Lagrange form*, a problem that optimises only the boundary objective $\sum_{v=1}^{N_v} \Psi_v(x(T))$ is said to be in *Mayer form* and a problem with both terms is said to be in *Bolza form*. An example where only the Mayer form would be necessary could be a situation where the desired destination of the vehicles does not have enough room for them all to be arranged in their desired positions. Therefore, the goal of the optimiser is to find the closest to the desired positions.

A direct method based on Bernstein polynomials is used here. This means that some or all of the state variables/inputs are approximated by polynomials, each with the same order N. By using these polynomials all of the states and inputs for each vehicle combined will produce a total of $(N_v \times (n_u + n_x) \times (N+1))$ control points.

Let t_0, t_1, ..., t_N be a set of equidistant *time nodes* such that $t_j = j\frac{t_N}{N}$, with $T > 0$. By following the notation of Bernstein polynomials:

$$\begin{aligned} x^{[i]}(t) \approx x_N(t) = \sum_{j=0}^{N} \overline{x}_{j,N}^{[i]} b_{j,N}(t), \quad t \in [0,T] \\ u^{[i]}(t) \approx u_N(t) = \sum_{j=0}^{N} \overline{u}_{j,N}^{[i]} b_{j,N}(t), \quad t \in [0,T] \end{aligned} \tag{4.2}$$

with $x_N : [0,T] \to \mathbb{R}^{n_x}$ and $u_N : [0,T] \to \mathbb{R}^{n_u}$. In this equation above, $\overline{x}_{j,N} \in \mathbb{R}^{n_x}$ and $\overline{u}_{j,N} \in \mathbb{R}^{n_u}$ are Bernstein coefficients. Let $\overline{x}_N \in \mathbb{R}^{n_x \times (N+1)}$ and $\overline{u}_N \in \mathbb{R}^{n_u} \times (N+1)$ be defined as $\overline{x}_N = [\overline{x}_{0,N}, \ldots, \overline{x}_{N,N}]$, and $\overline{u}_N = [\overline{u}_{0,N}, \ldots, \overline{u}_{N,N}]$

The optimization problem now becomes

$$\begin{aligned} \underset{\overline{x}_N^{[i]}, \overline{u}_N^{[i]}, i=1,\ldots N_v}{\text{minimize}} \quad & \int_0^T \sum_{i=1}^{N_v} L^{[i]}(\overline{x}_N^{[i]}, \overline{u}_N^{[i]}) dt + \Psi(x_{N,N}) \\ \text{subject to} \quad & \overline{x}_{0,N}^{[i]} = x_0^{[i]}, \\ & \overline{x}_{N,N}^{[i]} = x_f^{[i]}, \\ & \sum_{k=0}^{N} \boldsymbol{D}_{j,k} \overline{x}_{k,N}^{[i]} = f^{[i]}(\overline{x}_{j,N}^{[i]}, \overline{u}_{j,N}^{[i]}), \qquad \forall j = 0, \ldots, N \\ & h(\overline{x}_N, \overline{u}_N) \geq 0, \end{aligned} \tag{4.3}$$

where $D_{j,k}$ is the (j,k)-th entry of the differentiation matrix $D_N = D_{N-1} E_{N-1}^N \in \mathbb{R}^{(N+1)\times(N+1)}$ that is obtained by combining the Bernstein differentiation matrix (2.44) with the Bernstein degree elevation matrix whose indices are given by (2.48).

What this new optimization problem means is that cost, and constraints dealt with $h(.)$ can treat the control points as actual sample of the respective polynomial functions. On the other hand, the dynamics assume that the control points are good enough approximations for the polynomial functions themselves. As a result, the equality constraint $\dot{x} = f(x,u)$ is applied to each control point instead of all infinite points of the respective polynomials.It also means that the cost can be obtained by applying the running and terminal costs to the control points.

Let (x,u), be a feasible solution to problem 4.1. If the the solution if is continuous, then problem 4.3 admits a feasible solution too. Moreover, let (x^*, u^*) be an optimal solution to 4.3 and $\{(x_N^*, u_N^*)\}_{N=N_1}^{\infty}$ be a sequence of optimal solution to problem 4.3, it can be proven that

$$\lim_{N\to\infty} (x_N^*(t), u_N^*(t)) = (x^*(t), u^*(t)), \tag{4.4}$$

which means that the higher the order N the closer to the true cost the solution is, and, as a result, the truer is the cost too. Proof for the feasibility and consistency of (4.4) can be found in [3].

4.2 The Log Barrier Functional

A log barrier functional can be added to the cost functional such that the motion planning problem becomes unconstrained, which is preferable because solutions to unconstrained problems generally are obtained with a lower computation time. Specifically, the inequality constraints no longer become constraints and are instead moved to the cost functional by applying a log barrier functional to them.

An optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & \int_0^T l(x(\tau), u(\tau), \tau) d\tau + m(x(T)) \\ \text{subject to} \quad & \dot{x}(t) = f(x(t)), u(t), t), \quad x(0) = x_0 \\ & c_j(x(t)), u(t), t) \leq 0, \qquad t \in [0, T], j \in \{1, \dots, k\} \end{aligned} \tag{4.5}$$

becomes

$$\begin{aligned} \text{minimize} \quad & \int_0^T l(x(\tau), u(\tau), \tau) + \sum_j \beta_\delta(-c_j(x(t), u(t), t)) d\tau + m(x(T)) \\ \text{subject to} \quad & \dot{x}(t) = f(x(t)), u(t), t), \quad x(0) = x_0 \end{aligned} \tag{4.6}$$

Given that the inequality constraint functional must be negative for the solution to be feasible, $\beta_\delta(-c_j(\cdot))$ must be nearly constant for positive values. One possible log barrier functional can be

$$\beta_\delta(z) = \begin{cases} -\log z & z > \delta \\ \frac{k-1}{k}\left[\left(\frac{z-k\delta}{(k-1)\delta}\right)^k - 1\right] - \log \delta & z \leq \delta \end{cases} \tag{4.7}$$

which is continuous and whose derivative around $z = \delta$ is also continuous too. [17].

The ”hockey stick” function, with form given by

$$\delta(z) = \begin{cases} \tanh(z), & z \leq 0 \\ z, & z < 0 \end{cases} \tag{4.8}$$

can be applied to the log barrier functional such that its derivative is smaller for feasible solutions when combined with β. Figure 4.1 shows the log barrier function with and without the hockey functional applied.

4.3 Inter-Vehicular Constraints

There are 2 ways of preventing inter-vehicle collision: *spatial deconfliction* and *temporal deconfliction* [18]. Spatial deconfliction is appropriate for Path Following (PF) and imposes the constraint that the spatial paths of the vehicles under consideration will never intersect and keep a desired safe distance from each other. Temporal deconfliction is appropriate for Trajectory-Tracking (TT) and requires that two

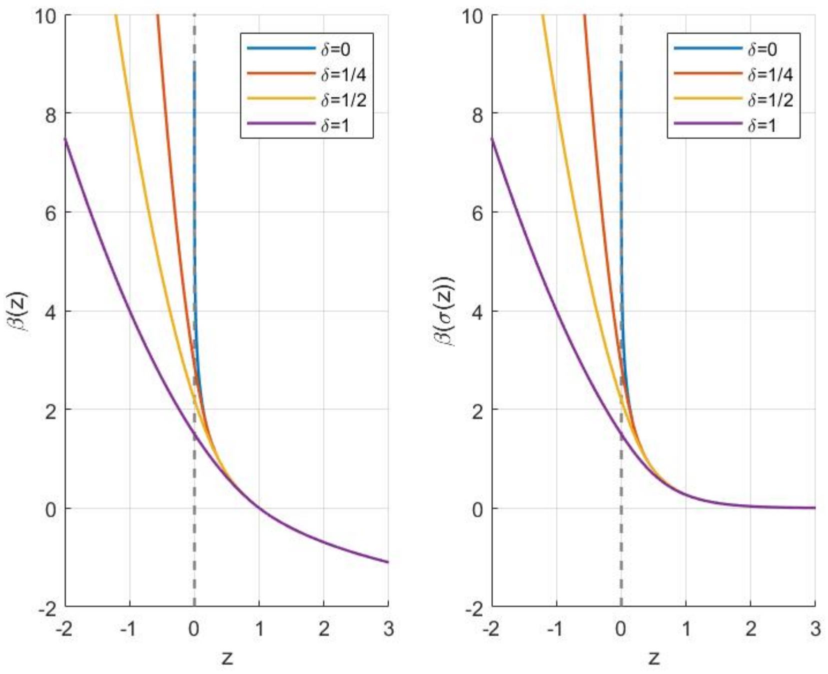
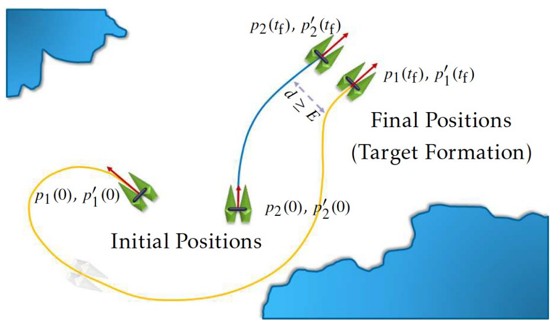


Figure 4.1: log barrier functional without (left) and with (right) the hockey stick function

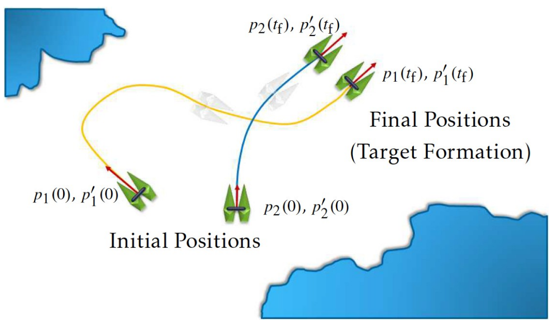
vehicles will never be ”at the same place at the same time”. However, their spatial paths are allowed to intersect. Figure 4.2 illustrates the two types of deconfliction strategies. Temporal deconfliction allows an extra degree of freedom and will intuitively lead to cheaper dynamic costs.

Two methods for temporal deconfliction are presented here, one based on sampling the curves and the other based on directly exploiting the properties of Bernstein polynomials.

For N_v vehicles, any of $\binom{N_v}{2}$ pairs could lead to a collision, which means that all pairs must be tested resulting in quadratic time complexity. This implies that finding fast algorithms to calculate the distance between each pair of trajectories becomes essential.



(a) Spacial Deconfliction



(b) Temporal Deconfliction

Figure 4.2: Inter Vehicle deconfliction Solutions

4.3.1 Sampling Trajectories

The most straightforward way to calculate the minimum distance between two trajectories is to sample each trajectory and calculate the euclidean distance between time equivalent samples; the smallest value being the minimum distance. The smallest yields the shortest distance. This method, however, is not perfect because the finer the samples, the higher the computational cost will be. However, high accuracy in the calculation of the minimum distance between trajectories in not necessary. If, besides knowing the position of the vehicle at a sample, we also know the maximum tangent speed between that sample and the next, then the distance to the other vehicle cannot deviate more than a certain determinable value between those 2 points. It will be smaller than the calculated distance if the vehicles are moving towards each other. The number of samples will determine how large this deviation can be. The optimization algorithm will stop once it finds a minimum distance that is greater than a certain value. This implies that the number of samples must be large enough such that the deviation is relatively small when compared to the desired minimum distance.

For example, consider a minimum distance of $1\,\mathrm{meter}$ between two vehicles which have maximum velocities $1\,\mathrm{m\,s^{-1}}$. At a certain moment between two samples, the vehicles could be closer to each other than they are at the samples. Consider the worst case scenario: during half of the time interval between samples the vehicles move towards each other at maximum velocity and half the time they move away from each other as illustrated in figure 4.3. If the time sample is $10\,\mathrm{ms}$, during $5\,\mathrm{ms}$ the vehicles can travel $1\,\mathrm{cm}$ towards each other, at the relative speed of $2\,\mathrm{m\,s^{-1}}$, which is a $1\,\%$ deviation from the established minimum distance of $1\,\mathrm{m}$. If the optimization problem is reformulated to guarantee $1.01\,\mathrm{m}$ of separation, then, with samples spaced by $10\,\mathrm{ms}$, a successful optimization solution guarantees a minimum distance between vehicles of $1\,\mathrm{m}$. which means a total of 100 samples per second of runtime.

4.3.2 Bezier Curve Distance to a Point

A second method to calculate the minimum distance between two trajectories is based on calculating the minimum distance between Bezier curves [19]. This algorithm is adapted to calculate the minimum distance between a curve and a convex shape. By subtracting the position vectors of one trajectory from the one trajectory from another, which for Bezier curves is explained in section 2.7, finding the minimum distance between trajectories gets translated to finding the closest point of this subtraction curve to the origin which can be interpreted as a 1 point convex shape.

The algorithm consists of recursively breaking down the curve into two halves by obtaining a new set of control points for each half via de De Casteljau algorithm. For each half, two values are calculated: an upper and lower bounds for the minimum distance to the complex shape. The upper bound for the minimum distance will be the closest endpoint of the segment to the convex shape, the lower bound

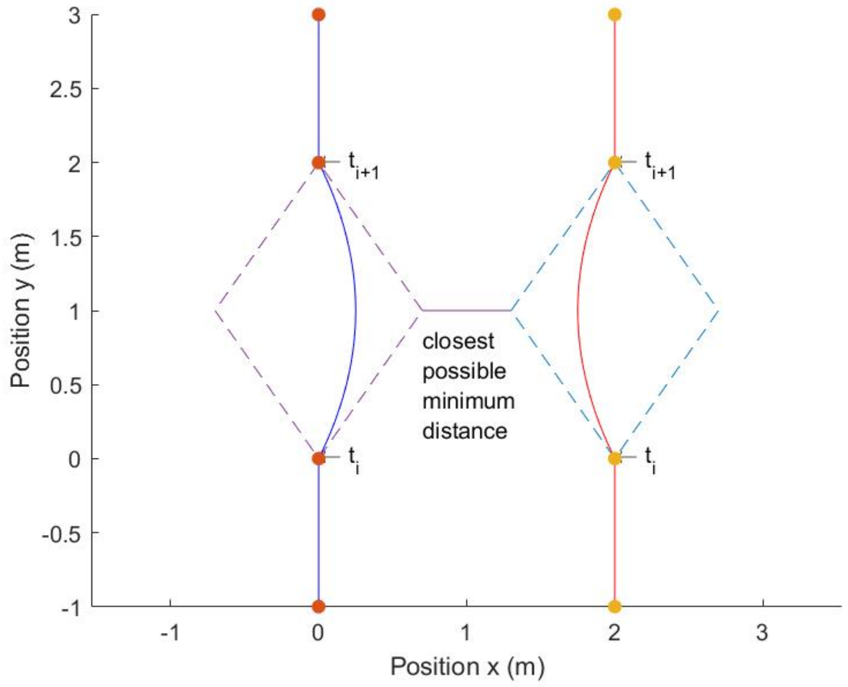


Figure 4.3: Worst case scenario for two trajectories in the time interval between samples

is the closest point of the convex hull of the new control points to the polygon. Both upper and lower bounds are calculated with the GJK algorithm (see section 4.3.3). The exit condition of the recursion is when the lower bound is approximately equal to the upper bound. The lower bound may be zero if the shapes intersect, which has no influence on the execution of the algorithm. If the exit condition is not met, the recursion is repeated and the returned value is the smallest of the upper bound along with the time at which the smallest value was found.

4.3.3 GJK Algorithm

The Gilbert–Johnson–Keerthi (GJK) algorithm [20] is an efficient algorithm to calculate minimum distance between arbitrary convex shapes in any dimension and is a necessary tool for calculating the minimum distance between a Bezier Curve and a point or convex shape.

It relies heavily on a concept called the Minkowski Sum, but, because the difference operator is used for this algorithm instead of the sum, the term Minkowski difference will be used. For two shapes A and B, their Minkowski Difference is given by

$$D = A - B = \{a - b | a \in A, b \in B\} \tag{4.9}$$

where D is a new convex shape given by the subtraction of every point in A by every point in B. Figure

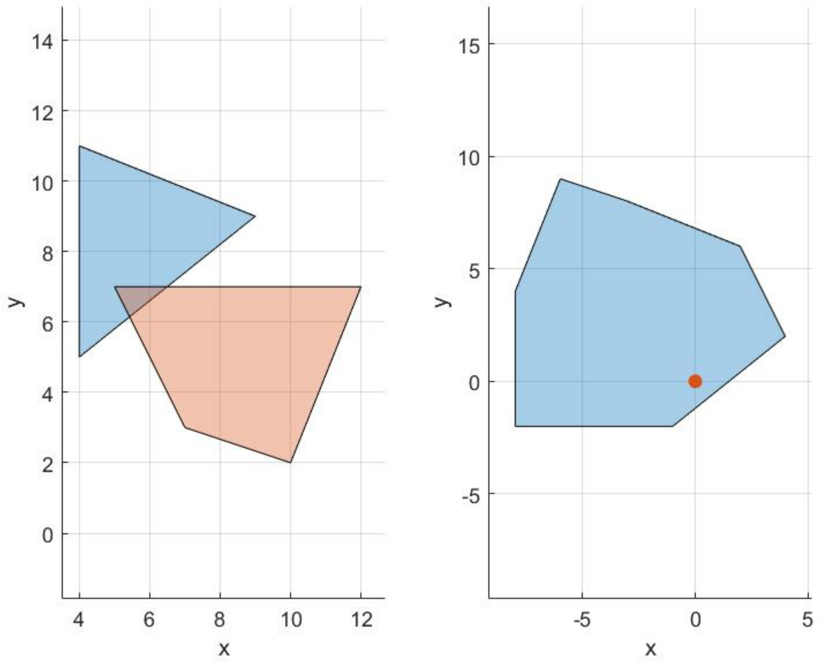


Figure 4.4: Intersecting convex shapes (left) and resulting Minkowsky Difference (right)

4.4 shows 2 shapes on the left hand-side which intersect and the resulting Minkowski Difference on the right hand side. Figure 4.5 shows two shapes that do not intersect and their resulting Minkowski Difference. Notice how the second Minkowski Difference has an identical shape to the first, only differing on its position.

If the Minkowski Difference contains the origin, then the two shapes have common points, i.e., intersect, because the resulting subtraction is zero. It is a two part problem:

1. detect intersection, by testing if the Minkowski Difference contains the origin;
2. if no collision is confirmed, calculate the minimum distance between the shapes.

A convex shape with up to N+1 vertices in an N-dimensional space is known as a simplex. For 2-D Minkowski Differences, the simplices can be a point (1 vertex), a line segment (2 vertices) and a triangle (3 vertices). For 3-D spaces, the simplices can be the same as in 2-D spaces with the addition of a tetrahedron (4 vertices).

The key to GJK's efficiency is to find points in the Minkowski difference that are the best candidates that form a simplex that can contain the origin.

A support function returns the farthest point of a convex shape in some direction. The resulting point is known as the "support" point. Finding a support point in the Minkowski Difference along direction $\overrightarrow{d}$ is the same as subtracting the support point of A along $\overrightarrow{d}$ and B along $\overrightarrow{d}$.

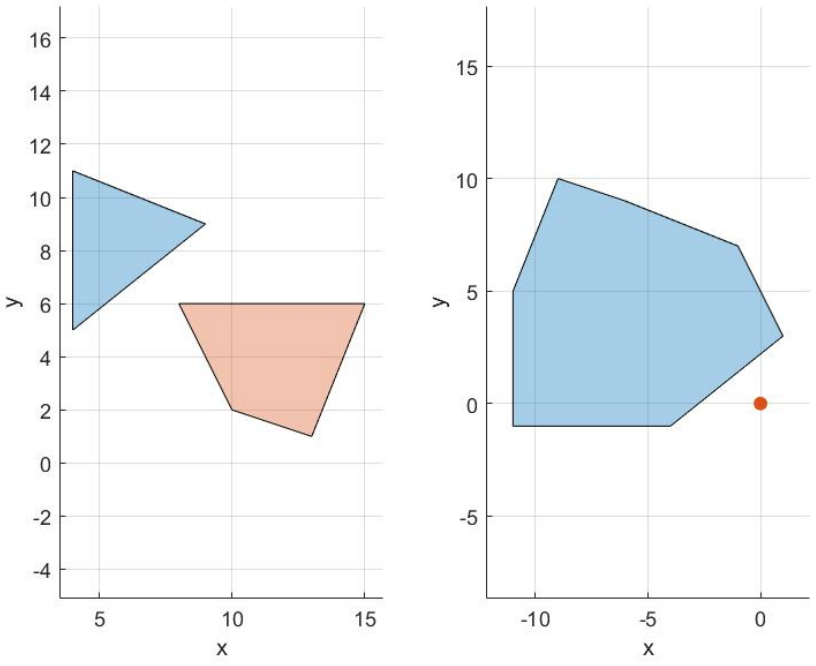


Figure 4.5: Non-intersecting convex shapes (left) and resulting Minkowsky Difference (right)

Choosing the farthest point in a direction has significance because it creates a simplex which contains a maximum area therefore increasing the chance that the algorithm finishes quickly. Let W_k be the set of vertices of the simplex constructed in the k^{th} iteration, and v_k as the point in the simplex closest to the origin. Initially, $W_0 = \varnothing$, and v_0 is an arbitrary point of the Minkowski Difference. Since each v_k is contained in the Minkowski Difference, the length of v_k must be an upper bound for the distance.

GJK generates a sequence of simplices in the following way. In each iteration step, a vertex $w_k = s_{A-B}(-v_k)$ is added to the simplex, with the objective of surrounding the origin. If the simplex contains the origin, then the program interrupts because the shapes intersect. If it is proven that the Minkowski Difference cannot contain the origin because the last added vertex did not move ”beyond” the origin, then program interrupts and moves on to finding the minimum distance to the origin. While intersection is not proven, the new v_{k+1} is perpendicular to the vector given by the last vertex with the one before that, or with the last with the third from the last, if available, depending on which has a dot product greater than 0, and the not used vertex gets removed from the simplex. Alternatively, if no intersection is proven, the new v_{k+1} is the point in the convex hull of $W_k \cup \{w_k\}$ closest to the origin and W_{k+1} becomes the smallest sub-simplex of $W_k \cup \{w_k\}$ that contains v_{k+1}.

4.4 Minimum Distance to Convex Shapes

Earlier, in section 4.3.2, an algorithm to calculate the distance of a trajectory to a convex shape was presented. This algorithm, however, is limited to convex shapes that do not intersect with the trajectory. If the curve intersects with the convex shape, the algorithm returns zero as minimum distance. optimization algorithms, such as Sequential Quadratic Programming[21], require the derivative of the constraints to be non zero, even when the current guess for solution is not feasible because this derivate, in other words, will ”inform” how far the control points must move so that the solution becomes feasible.

A modification to the algorithm that calculates the minimum distance to a convex shape is presented here to calculate the intersection points between the curve and the shape. Afterwards, these intersection points are used to calculate a ”penetration” of the curve in the shape.

First thing to note is, during the recursion of the minimum distance algorithm, some endpoints of the cut segments will land inside the shape. If a segment has an endpoint inside and an endpoint outside the shape and the distance between these two points is approximately zero, then the point inside is added to a stack of intersecting points, otherwise, the recursion continues. If the control points of the recursive segments are partially in the shape while others are not, then the recursion continues, otherwise, if the control points are all outside or all inside the shape, then there is no point in continuing because the segment cannot contain any more intersection points.

Once the intersection points are found, the ”penetration” must be calculated. Figure 4.6 shows a curve intersecting a shape with the intersection points marked. First step is to find the convex hull of the intersection points. Once the convex hull is determined, the Directed Extended Polytope Algorithm (DEPA) explained in section 4.4.1 is performed with the obstacle shape along a predefined direction $\vec{d}$. The bigger the penetration depth, the ”deeper” the trajectory is in the curve.

4.4.1 Directed EPA

If two convex shapes intersect, the GJK algorithm cannot provide collision information like the penetration depth and vector. One algorithm that provides this information is the Extended Polytope Algorithm (EPA) [22]. A slight modification for the EPA algorithm is proposed here. It will be referred as the Directed Extended Polytope Algorithm, whose objective is to find the penetration of one convex shape relative to another along a specific direction $\vec{d}$, while the EPA algorithm finds the shortest vector such that the shapes no longer collide. The penetration along a direction is the length of dislocation that the second shape would have to move so that the two shapes no longer collide.

The shapes intersect when the Minkowski difference contains the origin. This means that the DEPA or the EPA have as its objective the dislocation the Minkowski Difference such that it no longer contains the origin. Penetration along a specific direction can be found by calculating the length of the vector that

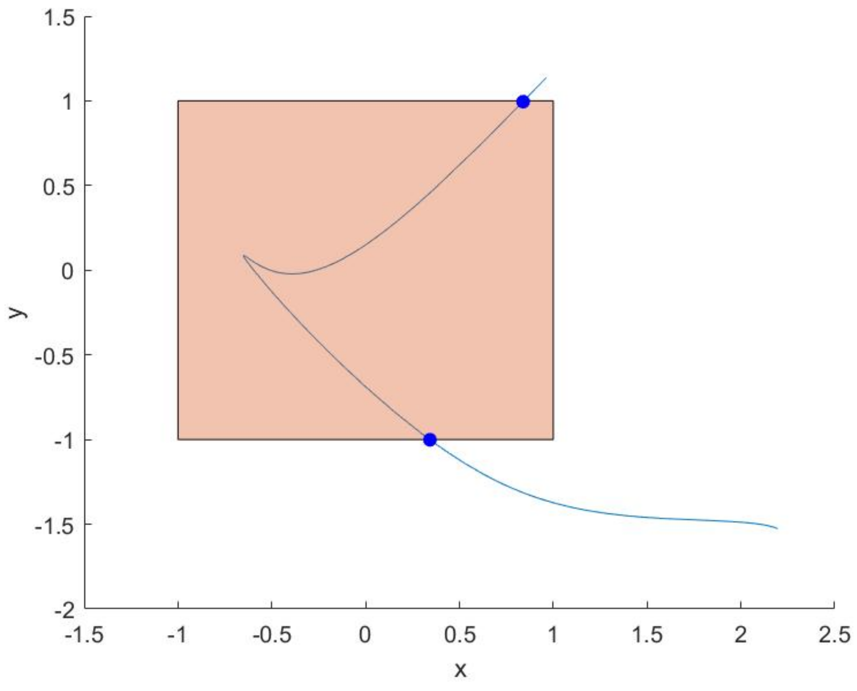


Figure 4.6: A curve intersecting a convex shape

starts at the origin on the Minkowski Difference, has the same direction as $\overrightarrow{d}$, and stops once it finds the edge of the Minkowski Difference. In other words, this is the norm of the intersection point between a ray starting at the origin with direction $\overrightarrow{d}$ and the edge of the Minkowski Difference. Once this length is found, shape B can move by that length along the direction of $\overrightarrow{d}$ such that it is no longer in collision with shape A.

The process of looking for the penetration along a direction $\overrightarrow{d}$ starts with a simplex which contains the origin constructed with points along the edge of the Minkowski difference. The first step is to find the only edge of this polygon which will intersect with the ray that starts in the origin with direction $\overrightarrow{d}$. Once this edge is found, the other edges of the simplex are ignored and an iterative process starts. The first step of the iterative process is to calculate a vector which is normal to the current intersecting segment and points ”outwards” with respect to the origin. Next, the support function is performed with this vector. The resulting point will be closer to the desired final point. There are now three points in play: the two segment ends and the new point that resulted from the support operation. The next step is to define two segments, one from the one of the segment ends to the new support point, the other from the other segment end to the support point. The next step is to find which of these two new segments intersects with the same ray with direction $\overrightarrow{d}$ and then repeat the iteration.

4.5 Soundness of Solution

Once the optimization process is complete, the trajectory is defined along with, depending on the implemented model, the control points for the inputs which can then be re-plugged into the ODE and check what resulting trajectory is obtained, this process is known as solving an IVP. If the approximation is good enough, the resulting trajectory should be nearly identical to the the function for the trajectory.

Another strategy to check soundness is to feed the inputs to the dynamic system just as explained in the previous paragraph but, this time, a correction term is calculated as well for the inputs based on the error with the desired position, such as what is done in Trajectory Tracking.

5

Software Implementation and Simulation Results

Contents

5.1 Description of the Implemented Code

The variables for the motion planning problem are each vehicle's state variables and inputs, as explained in section 4.1. Each of these variables will be referred to as *curves*. optimization algorithms cannot take continuous functions as variables, therefore, some form of parameterization of each curve is necessary, as exemplified in some of the algorithms of chapter 2. Here, each curve will be represented as a Bernstein Polynomial with order N, which will require $N+1$ control points. A distinction is made between state variables and inputs: state variables must have established initial and final conditions, inputs do not. This implies that for state variables, the initial and final control points must be fixed; therefore, they do not need to participate on the optimization problem. The following matrix represents how the control points are stored so that they can be accessed by all functions that perform operations on the curves.

$$\begin{bmatrix} x_0^0 & y_0^0 & \psi_0^0 & u_0^0 & v_0^0 & r_0^0 & \tau_{u_0}^0 & \tau_{r_0}^0 \\ x_1^0 & y_1^0 & \psi_1^0 & u_1^0 & v_1^0 & r_1^0 & \tau_{u_1}^0 & \tau_{r_1}^0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N^0 & y_N^0 & \psi_N^0 & u_N^0 & v_N^0 & r_N^0 & \tau_{u_N}^0 & \tau_{r_N}^0 \end{bmatrix} \tag{5.1}$$

It is believed that `fmincon()` performs better when all variables are stored in a line vector [23]; therefore, a function called `matrify()` is necessary in order to transform the flattened optimization variable to the matrix of equation (5.1).

Elements marked in yellow do not participate in the optimization algorithm. They are concatenated to this matrix in `matrify()`.

High orders are preferable for each curve [3] because the higher the curve, the closer the control points are to the "matching" point in time of the curve, which is achieved once an optimization problem finishes. Chapter 5 shows how the control points approximate to the curve and how it is advantageous to produce the dynamics.

The optimization problem is formulated by constructing a data structure with the fields of table 5.1.

Some notes for each of the fields:

- `xi` has as many lines as state variables (not input variables) and as many lines as number of vehicles. Because these functions are designed for vehicles, x and y must be in the first 2 columns
- `xf` works just as `xi`
- `obstacles_circles` Ncircles by 3, where columns are x, y and radius, respectively
- `recoverxy` takes an aribtrary X matrix and and a `constants` structure and returns a Npoints by 2 matrix

field	**description**	**mandatory**	**example**
`T`	Time horizon	yes	`10`
`xi`	initial conditions	yes	`[0 0 0 1 0]`
`xf`	final conditions	yes	`[5 5 pi/2 1 0]`
`N`	order of the curves	yes	`15`
`obstacles`	polygons	no default: `[]`	
`obstacles_circles`	circles	no default: `[]`	
`min_dist_int_veh`	minimum distance between vehicles for every point in time	no default: `0`	`.8`
`numinputs`	number of input variables (don't have initial conditions)	no default: `0`	
`uselogbar`	make the problem completely unconstrained and use log barrier functionals	no default: `false`	
`usesigma`	a boolean for the use of the sigma function if log barrier functionals are to be used	no default: `false`	
`costfun_single`	a function used to calculate the running cost for each singular vehicle	yes	`@costfun`
`dynamics`	a function that describes how the non linear dynamics of the state variables and inputs are linked	yes	`@dynamics`
`init_guess`	a function that provides an initial guess for the optimization problem which may speed up the process of optimization	no default: `@rand_init_guess`	`@init_guess`
`recoverxy`	a function that returns the x and y variables by solving just the initial value problem of the inputs	yes	`@recoverxy`

Table 5.1: Description of the constants for optimization

- `dynamics` takes in an arbitrary X matrix and constants structure (to provide pre computer information like a derivation matrix) and must return a column vector which is zeros when all of the dynamic constraints are respected

The data structure for the nonlinear optimization problem is then passed to a function called `run_problem` which returns the control points for each of the defined variable along with computation time.

5.1.1 Dynamics

The dynamics are guaranteed with the formulation of 4.3. This means that the code version for the dynamics plus kinematics for the Medusa vehicle, for example, which are 3.15 and 3.16, become

Listing 5.1: `Dynamics Equality Constraint`

```
ceq = [
    DiffMat*x - u.*cos(yaw) + v.*sin(yaw) - Vcx;
    DiffMat*y - u.*sin(yaw) - v.*cos(yaw) - Vcy;
    DiffMat*yaw - r;
    DiffMat*u - 1/m_u*(tau_u + m_v*v.*r - d_u.*u+fu);
    DiffMat*v - 1/m_v*(-m_u*u.*r - d_v.*v+fv);
    DiffMat*r - 1/m_r*(tau_r + m_uv*u.*v - d_r.*r+fr);
];
```

As explained in section 4.1, `Diffmat` preserves the order and the equality is maintained in the control points, not the values of the curve itself so some approximation error is expected, which can be minimized with higher orders.

Given that each variable is defined by a set of control points, and, using the convex hull property of Bernstein polynomials, upper and lower bounds for each variable, state or input, become the biggest or smallest control point, respectively.

5.2 Results

The following results are based on solving the optimization problem (4.3) with the implementation described in chapter 4. Simulations were carried out using Sequential Quadratic Programming [21] on models presented in chapter 3. Simulations were run on a 4 × Intel© Core™i5-7200U CPU @ 2.50GHz processor.

The unicycle model has a total of 5 state variables while the Medusa model has a total of 6 state variables plus 2 inputs. Upper and lower bounds for each variable for each vehicle were implemented, as explained in section 5.1.1, by finding the biggest and smallest control points of each polynomial. For the examples presented in this chapter, the bounds that were used are those presented in table 5.2 which were chosen to represent realistic values of a vehicle.

	Variable	Lower Bounds	Upper Bounds
Dubin's Car	x (m)	$-\infty$	∞
	y (m)	$-\infty$	∞
	ψ (rad)	$-\infty$	∞
	u (m s^{-1})	0	1.1
	r (rad s^{-1})	$-\pi/4$	$\pi/4$
Medusa	x (m)	$-\infty$	∞
	y (m)	$-\infty$	∞
	ψ (rad)	$-\infty$	∞
	u (m s^{-1})	0	1.1
	v (m s^{-1})	$-\infty$	∞
	r (rad s^{-1})	$-.74$	$.74$
	τ_u (N)	0	25.9
	τ_r (N m)	-.113	.113

Table 5.2: Upper and lower bounds for each variable of each vehicle model

A range of experiments were performed with the optimization algorithm in order to study its behaviour with changing parameters. Out of all of the experiments that were performed, the most relevant are presented here.

5.2.1 No Obstacles

The first problem considered here is, for both a single Dubin's car and a single Medusa vehicle, with initial and final states described in table 5.3 and with no obstacles. Figure 5.1 show solutions for order 20 and a time horizon of 60 s. The top two plots minimize the square of the surge of the Dubin's car and Medusa vehicle, while the bottom two minimize the thrust τ_u and torque τ_r, respectively. This figure, and all other figures, flip x and y axis which is standard for marine vehicles.

Variable	Starting Conditions	Final Conditions
x (m)	0	30
y (m)	0	30
ψ (rad)	0	$\pi/2$
u (m s^{-1})	1	1
v (m s^{-1})	0	0
r (rad s^{-1})	0	0

Table 5.3: Initial and final conditions for a basic Motion Planning Problem

The ten plotted arrows in each plot show the position and heading of the vehicles over equidistant time intervals. The closer the arrows are to each other, the slower the vehicle are during the respective time interval. The initial guess that is provided to these examples are a set of random values that

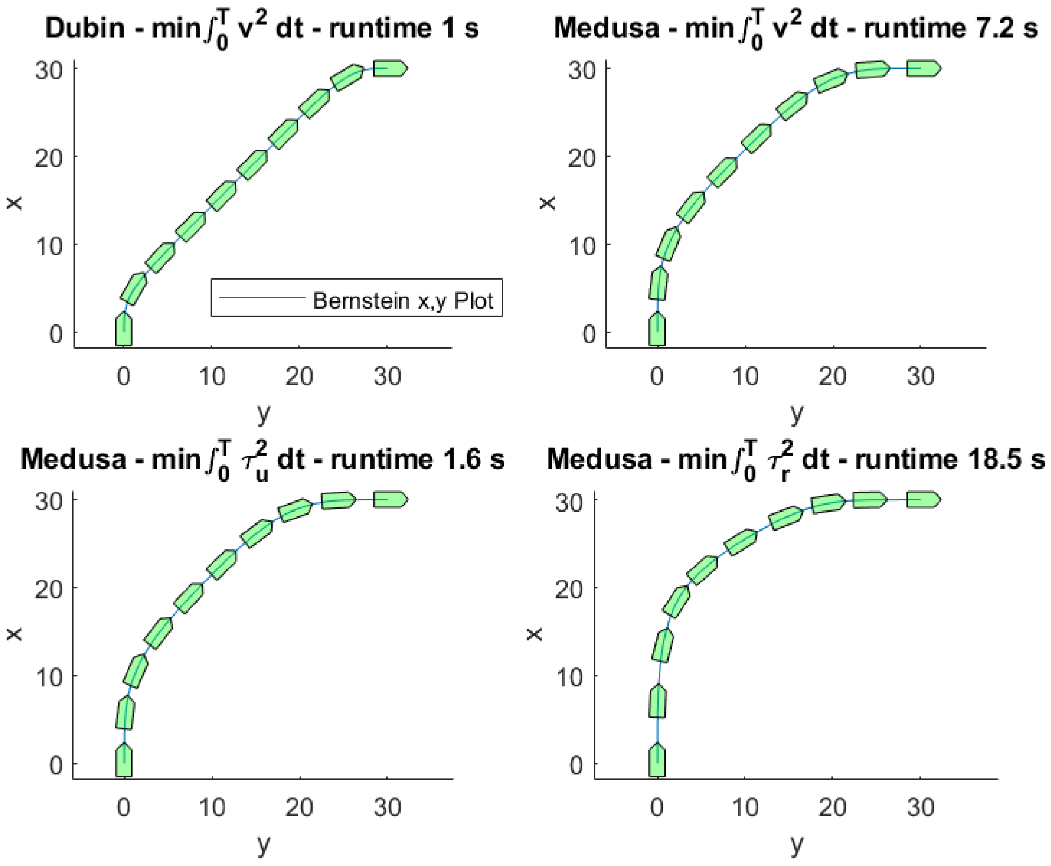


Figure 5.1: Solutions of order $N = 20$ without obstacles and final time $T = 60$

respect the variable bounds established earlier. These examples serve as baselines for comparison once obstacles and multiple vehicles are introduced in the next sections.

5.2.2 Obstacles

Obstacle avoidance is implemented with the algorithms explained in sections 4.3 and 4.4. The top left solution of figure 5.1 serves as the baseline for comparison without obstacles. Specifically, the solutions presented in this section use the same velocity minimisation running cost (min $\int_0^T v^2$), the same initial and final conditions (5.3), and the same order of polynomial $N = 20$ for the Dubin's car. The only difference is the presence of the different obstacles. Figure 5.2 show the solutions of the performed tests. The titles of the plots which say ”Constrained” refer to when the environmental obstacle is implemented as an optimization constraint. The titles which say ”Unconstrained” refer to plots resulting from unconstrained optimization, i.e., the environmental constraint was moved to the cost via the Log Barrier Functional. It is expected that the trajectories of the unconstrained problems are not perfectly tangent to the circle. If the running cost were not added to the total cost, the trajectory would be as far away as possible from the curve while still respecting the dynamic constraints; this is because the derivative of the Log Barrier Functional is always lower than zero even when the solution is feasible. The gap that we see between the curve and the obstacle is the result of balancing the running cost and the Log Barrier component. One implication is that running cost can never be ideal because the vehicle had to travel an extra amount of distance. Using the Log Barrier function, however, has the advantage of potentially reducing the runtime of the optimization process, as can be seen by comparing the computation times of the problems with the polygon obstacles. Furthermore the large difference in computation time between polygon obstacles and circle obstacles can be explained by the fact that the *Minimum Distance to Polygon* algorithm is iterative, which means that all of the duration of the individual runs quickly add up.

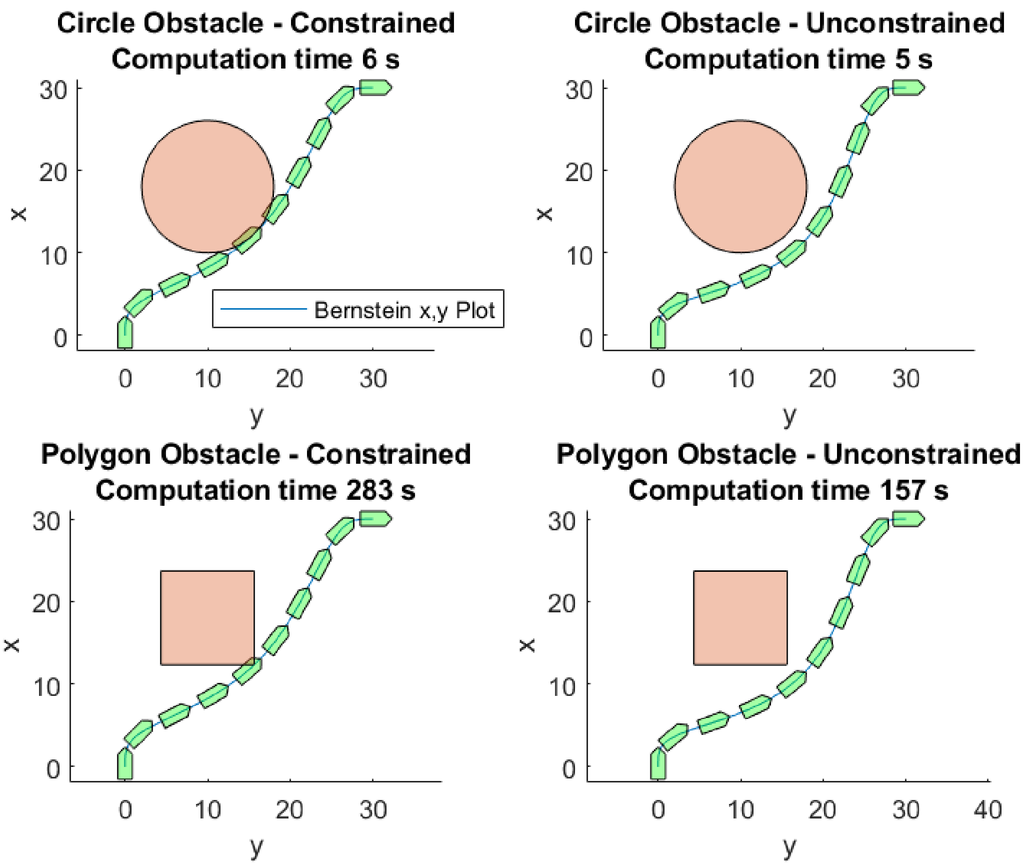


Figure 5.2: Solution of order $N = 20$ with obstacles and final time $T = 60$

5.2.3 Variation of cost with order

A way of calculating solutions for high orders while maintaining low computation time is by running the optimization process on a random guess with low order, then perform degree elevation which increases the order of the solution and re-feed that solution to the same problem as the initial guess. In order to demonstrate this process, a new optimization problem was is presented. It uses the Dubin's car but with initial and final conditions given by table 5.4.

Variable	Starting Conditions	Final Conditions
x (m)	0	30
y (m)	0	30
ψ (rad)	$\pi/4$	$\pi/4$
u (m s^{-1})	1	1
r (rad s^{-1})	0	0

Table 5.4: Initial and final conditions for each step of the *Increasing Order* Process

Figure 5.3 shows the first, last, and two intermediary steps of this process. Figure 5.4 shows the duration and cost of each intermediary run. The execution starts with order $N = 10$, resulting the top left plot of the figure. This solution, then, has its order increased by 10, and is passed as an initial guess of the same problem but now with order 20.This process of increasing order stops when the final cost of the last run varies only by 1% of previous run. For this particular example, the iterative process stops with order 70. The same problem of order 70 but with a random initial guess, took a total of 334 seconds, as opposed to 660 seconds, for the overall duration for all runs from order 10 to 70. Despite the total duration being nearly twice that for the run of order 70 with a random initial guess, this method may still be advantageous because the run of order 70 is faster, which suggests that with a good criterion for increasing order, the overall duration may be smaller. This method has the extra advantage of producing a solution we know is close to optimal.

Figure 5.5 shows, for order 10 on the left and order 70 on the right the two optimal solutions for the same problem, but this time, accompanied by the solution of the initial value problem, which integrates the speed and turn rate as explained in section 4.5. With them, it can be seen how all of the variables returned by the motion planning algorithm are correctly linked with each other and how the higher the order, the more identical these curves are to one another.

5.2.4 Multiple Vehicles

To illustrate the use of multiple vehicles, a series of runs are presented in figure 5.6. The initial and final conditions other than position are given by table 5.5 with order $N = 10$. Figure 5.7 shows how the computation time increases with the added vehicles. These solutions are run with the inter vehicular constraints implemented as constraints on the optimization problem, as opposed to imposing them in the cost via the Log Barrier Functional. Specifically, the imposed constraint is a separation of 3 m between trajectories calculated with the sampling based algorithm. Given that the problem focuses on trajectories, there is no problem in seeing the paths intersect. The same two vehicle problem with order $N = 10$ takes

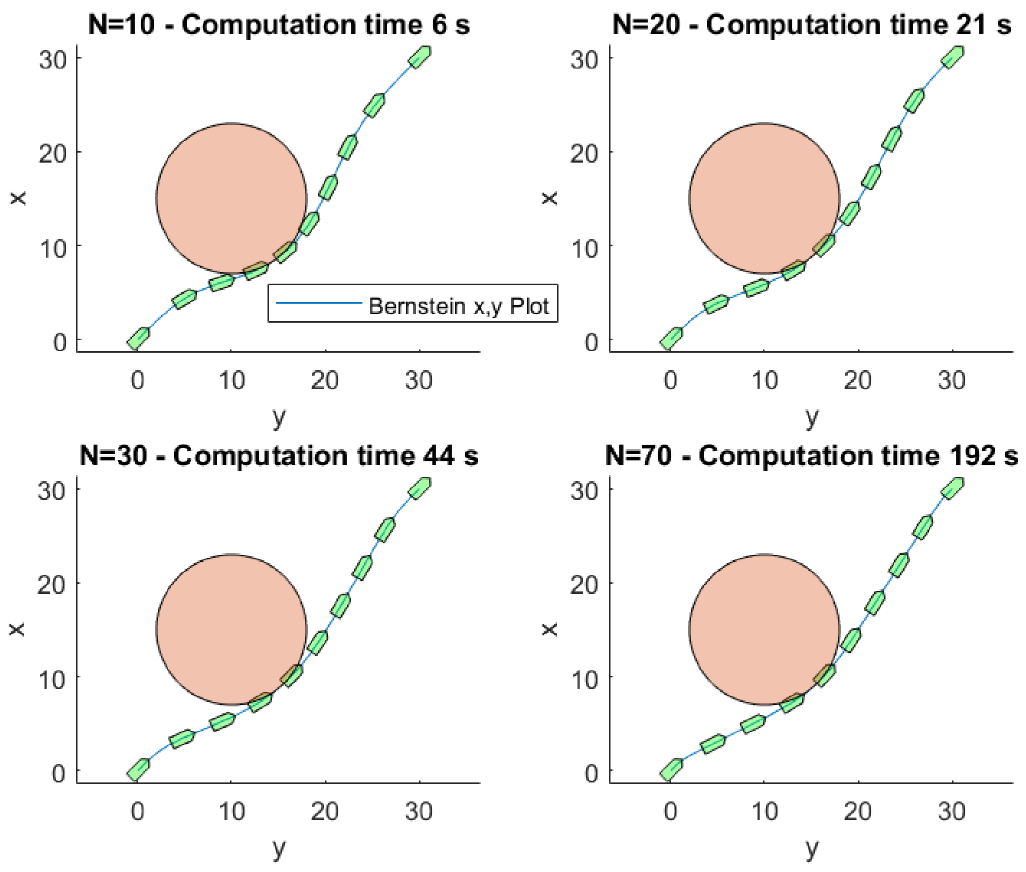
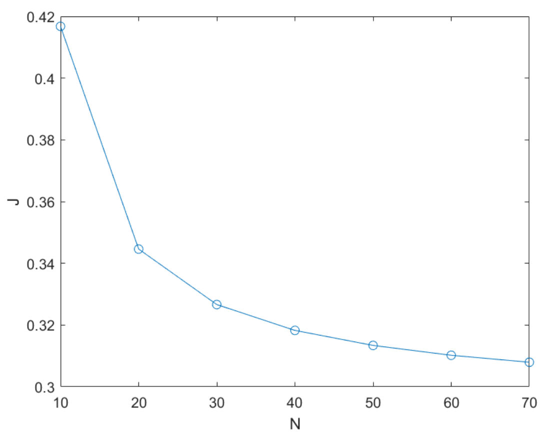
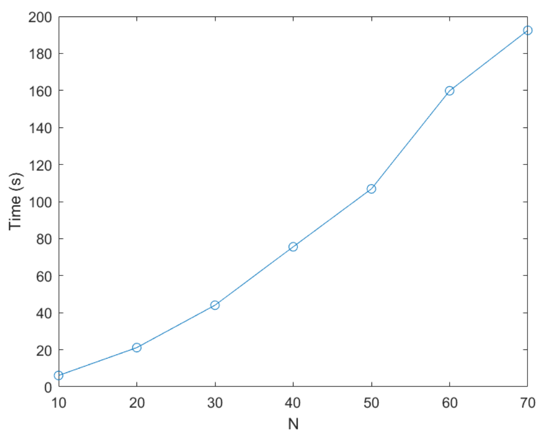


Figure 5.3: Results of Interatively increasing order



(a) Evolution of Cost



(b) Evolution of computation time

Figure 5.4: Evolution of Cost and Time with Order Increase

a total of 56 s with the iterative *Berstein to a point* algorithm which again shows how using an iterative algorithm with an optimization problems adds a lot of computation time.

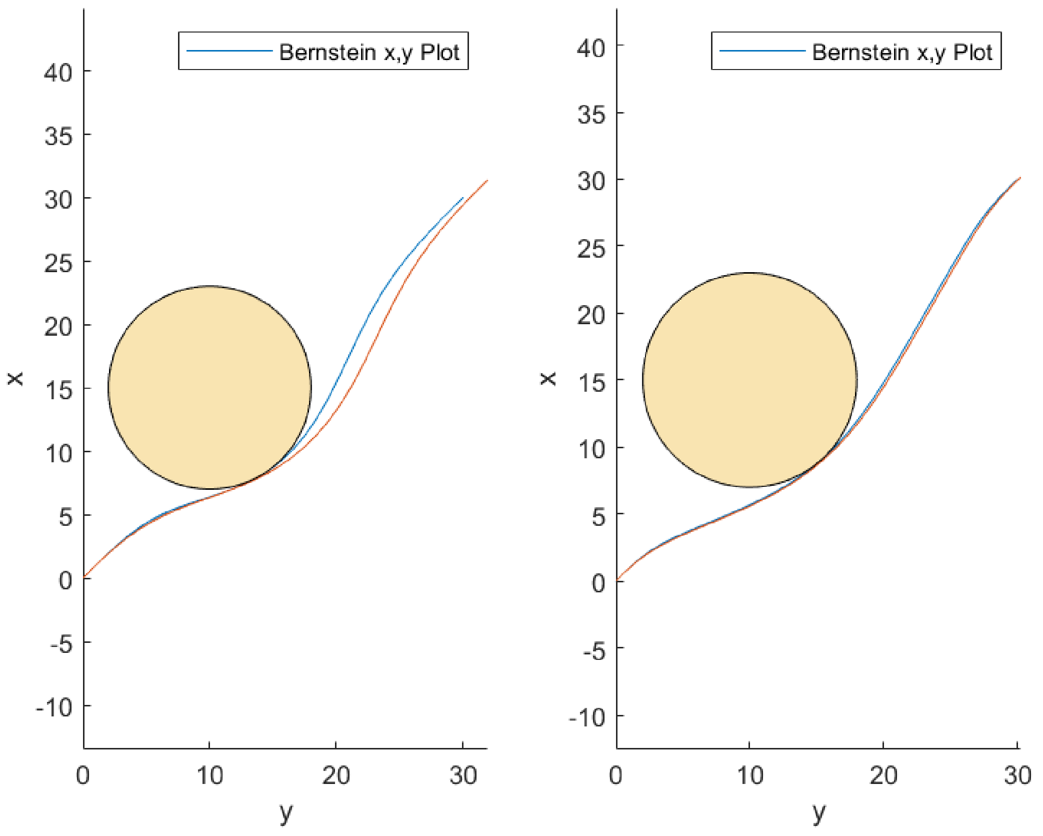


Figure 5.5: optimization solutions accompanied by the Initial Value Solutions

Variable	Starting Conditions	Final Conditions
ψ (rad)	0	0
u (m s^{-1})	1	1
v (m s^{-1})	0	0
r (rad s^{-1})	0	0

Table 5.5: Initial and final conditions for the Multiple Vehicles Problem

5.2.5 Trajectory Tracking Soundness

In section 4.5, it was suggested that soundness of the solution can be verified by feeding the returned inputs into a trajectory tracking algorithm and calculating the resulting size of error correction term. The better the inputs, the lower is the needed correction. A trajectory tracking algorithm that can perform this task is not available here. This task is carried out based on a graphical comparison of the inputs returned by the motion planning algorithm and the inputs calculated by a conventional trajectory tracking algorithm, such as the one found in [21], which calculates the inputs solely on error to the desired position.

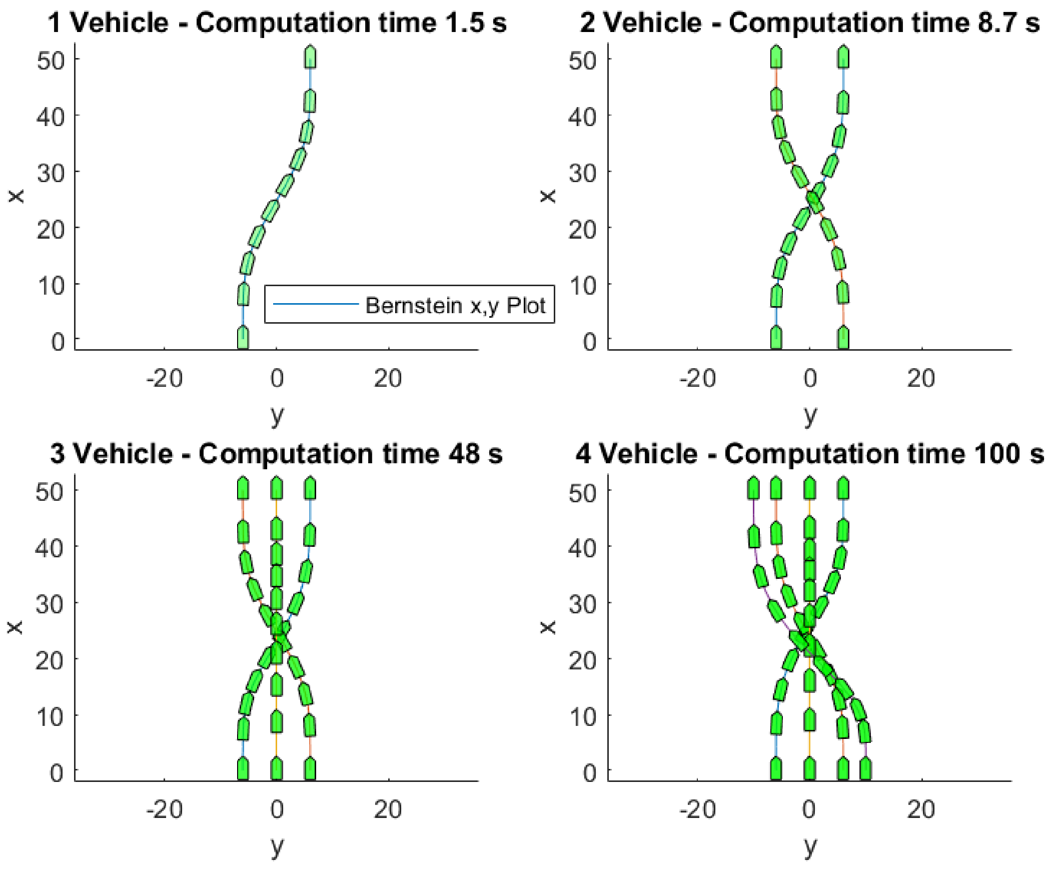


Figure 5.6: Solutions of order $N = 10$ with multiple vehicles

Figure 5.8 shows the solution of a constrained optimization problem for the Medusa Vehicle with order $N = 50$, with initial and final conditions given by table 5.6.

Variable	Starting Conditions	Final Conditions
x (m)	0	0
y (m)	0	40
ψ (rad)	0	$\pi/2$
u (m s^{-1})	1	1
v (m s^{-1})	0	0
r (rad s^{-1})	0	0

Table 5.6: Initial and final conditions for a Constrained Problem

The computation time of the optimization problem is $113\,\mathrm{s}$. It also shows the result of applying the Trajectory Tracking controller in [24]. The inputs, which are surge and torque, are obtained by the optimization problem and the Trajectory Tracking algorithm are shown in figure 5.9. It can be seen that the resulting inputs look similar, which validates the solution of the optimization problem.

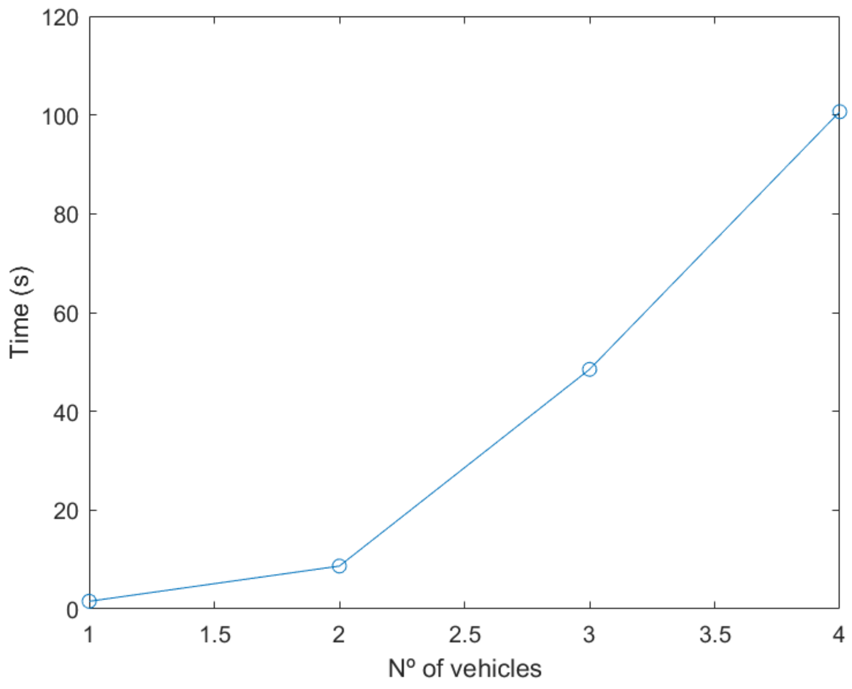


Figure 5.7: Evolution of Computation time with increasing number of vehicles

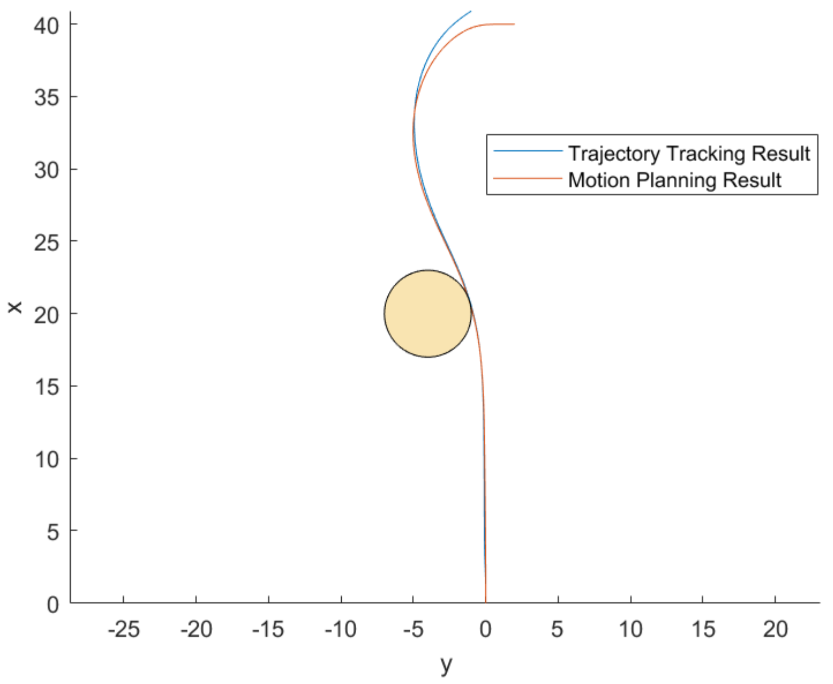


Figure 5.8: optimization Problem Solution and Trajectory Tracked Solution

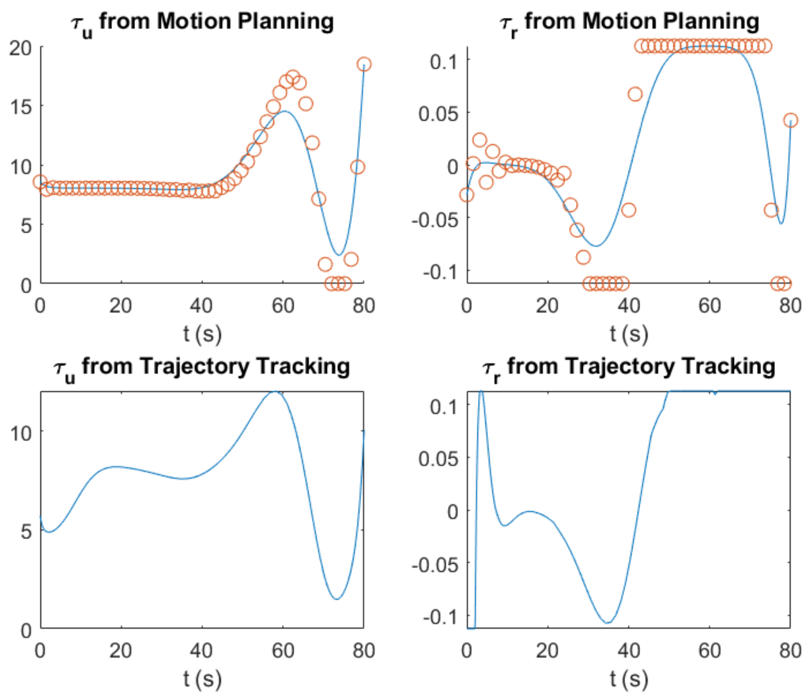


Figure 5.9: Comparison of the optimization Problem's Inputs and the Trajectory Tracking Inputs

5.2.6 Final Example

A final example of the application is shown in figure 5.10. It combines the various algorithms for inter-vehicle collision and obstacle avoidance and the use of the Log Barrier Functional to achieve the fastest possible computation time.

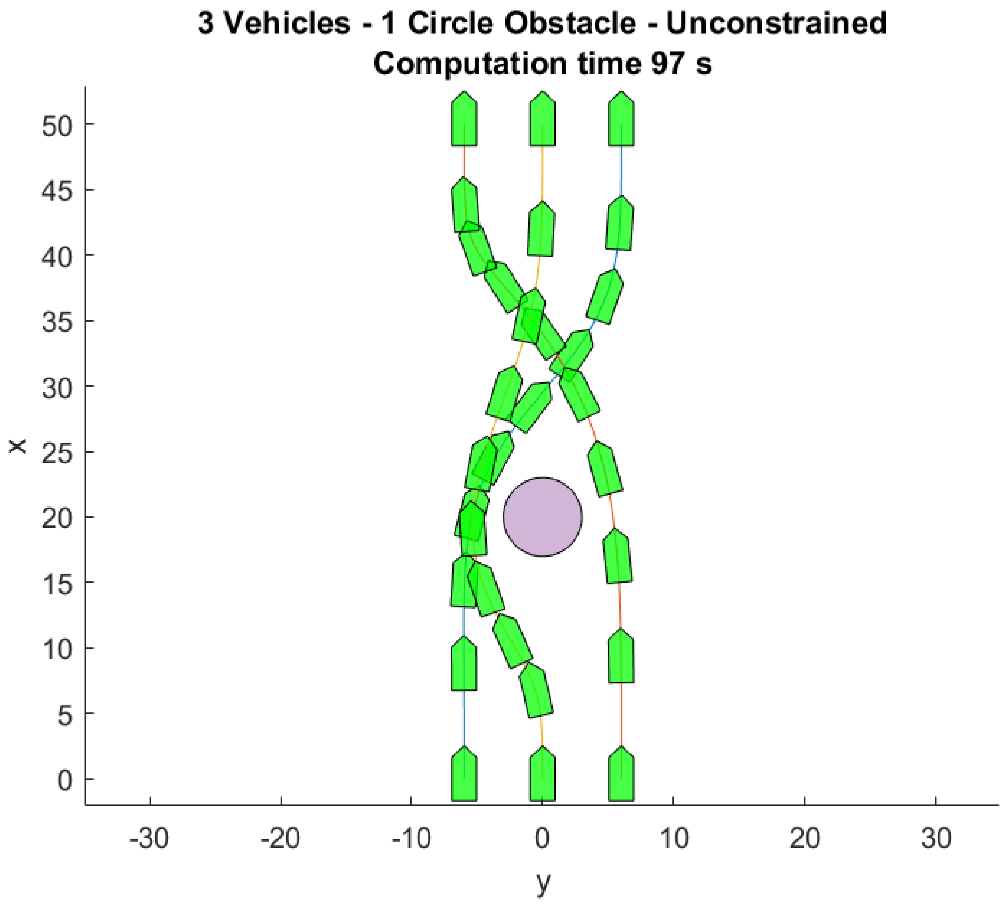


Figure 5.10: Solution of order $N = 10$ with 3 vehicles and 1 circle obstacle

6

Conclusion

The project was carried out within the master's program Robotics and Control under the supervision of Professor Pascoal. It consisted of the development a fast optimal motion planning algorithm for finding feasible and safe trajectories for a group of vehicles such that they reach a number of target points at the same time. It involed solving an optimal control problem by finding its equivalent finite dimensional optimization problem. The final form of the algorithm was chosen based on the comparison of several parameterization methods and their performance in handling dynamics and environmental constraints. Bezier curves were the final choice of parameterization to approximate the optimal trajectory due to their convenient properties that allow efficient computation and enforcement of constraints along the vehicles' trajectories.

The motion planning algorithm was tested with two AMV models: the Dubin's car model and the Medusa model, whose ruling kinematic and dynamics equations were also studied and presented here. It can be concluded that motion planning can now be performed for non-differentially flat systems such as the Medusa model.

Results of the application tests show how, with increasing order, the final cost quickly converges to optimal but at the expense of computation time. As a result, a trade-off must be found between optimality and the computation time when deploying the proposed algorithms in real life scenarios. It can also be concluded that introducing an iterative algorithm in each step of the optimization process - such as the presented *minimum distance to a polygon algorithm* presented here - introduces a disproportionate amount of computation time.

A trade off must also be found between order and number of vehicles, because as it has been seen in the tests, the computation time can quickly grow when adding more vehicles, even when the fastest sample-based minimum distance algorithms are used.

The motion planning algorithm, not only solves the problem for the go-to-formation maneuver but also supports other kinds of missions such as those based on active navigation. Therefore, future research can be based on applying the algorithms presented here for a wide range of complex motion planning problems. Another further investigation that can be derived from this work is testing the motion planning algorithms in close cooperation with trajectory tracking such that they can be applied in real time.

Bibliography

[1] P. Abreu, G. Antonelli, F. Arrichiello, A. Caffaz, A. Caiti, G. Casalino, N. C. Volpi, I. B. De Jong, D. De Palma, H. Duarte, et al., "Widely scalable mobile underwater sonar technology: An overview of the h2020 wimust project," Marine Technology Society Journal, vol. 50, no. 4, pp. 42–53, 2016.

[2] G. G. Lorentz, Bernstein polynomials. American Mathematical Soc., 2013.

[3] V. Cichella, I. Kaminer, C. Walton, N. Hovakimyan, and A. Pascoal, "Bernstein approximation of optimal control problems," arXiv preprint arXiv:1812.06132, 2018.

[4] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in Fast motions in biomechanics and robotics, Springer, 2006, pp. 65–93.

[5] P. C. Abreu, J. Botelho, P. Góis, A. Pascoal, J. Ribeiro, M. Ribeiro, M. Rufino, L. Sebastião, and H. Silva, "The medusa class of autonomous marine vehicles and their role in eu projects," in OCEANS 2016-Shanghai, IEEE, 2016, pp. 1–10.

[6] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," Pacific Journal of Mathematics, vol. 145, pp. 367–393, 1990.

[7] J. Riccati, "Animadversiones in aequationes differentiales secundi gradus (observations regarding differential equations of the second order)," Actorum Eruditorum Supplementa, vol. 8, pp. 66–73, 1724.

[8] A. V. Rao, "A survey of numerical methods for optimal control," Advances in the Astronautical Sciences, vol. 135, no. 1, pp. 497–528, 2009.

[9] M. Frank, P. Wolfe, et al., "An algorithm for quadratic programming," Naval research logistics quarterly, vol. 3, no. 1-2, pp. 95–110, 1956.

[10] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of non-linear systems: Introductory theory and examples," International journal of control, vol. 61, no. 6, pp. 1327–1361, 1995.

[11] M. Hazewinkel, Encyclopaedia of Mathematics: Supplement. 1. Springer Science & Business Media, 1997, p. 119.

[12] M.-S. K. Gerald E. Farin Josef Hoschek, Handbook of Computer Aided Geometric Design. Elsevier, 2002, pp. 4–6.

[13] T. Berry and A. Pascoal, private communication, September, 2020.

[14] C.-K. Shene, Finding a point on a bezier curve: De casteljau's algorithm, University Lectures, 2011.

[15] T. Fossen and A. Ross, "Nonlinear modelling, identification and control of uuvs," in Advances in Unmanned Marine Vehicles, G. Roberts and S. R., Eds. Peter Peregrinus LTD, 2006, ch. 2.

[16] B. Sabetghadam, R. Cunha, and A. Pascoal, "Cooperative motion planning with time, energy and active navigation constraints," in 2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV), IEEE, 2018, pp. 1–6.

[17] J. Hauser and A. Saccon, "A barrier function method for the optimization of trajectory functionals with constraints," in Proceedings of the 45th IEEE Conference on Decision and Control, IEEE, 2006, pp. 864–869.

[18] A. J. Häusler, "Mission planning for multiple cooperative robotic vehicles," Ph.D. dissertation, Department of Electrical and Computer Engineering, Instituto Superior Técnico, 2015.

[19] J.-W. Chang, Y.-K. Choi, M.-S. Kim, and W. Wang, "Computation of the minimum distance between two bézier curves/surfaces," Computers & Graphics, vol. 35, no. 3, pp. 677–684, 2011.

[20] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," IEEE Journal on Robotics and Automation, vol. 4, no. 2, pp. 193–203, 1988.

[21] N. I. M. Gould and P. L. Toint, "Sqp methods for large-scale nonlinear programming," in System Modelling and optimisation, M. J. D. Powell and S. Scholtes, Eds., Boston, MA: Springer US, 1999, pp. 149–178.

[22] G. Van Den Bergen, "Proximity queries and penetration depth computation on 3d game objects," in Game developers conference, vol. 170, 2001.

[23] T. Berry, A. Pascoal, and V. Cichella, private communication, September, 2020.

[24] F. Vanni, "Coordinated motion control of multiple autonomous underwater vehicles," M.S. thesis, 2007.