Motion Planning for Cooperative Autonomous Robots using Optimisation Tools

Thomas Berry
Department of Electrical and Computer Engineering
Instituto Superior Técnico
Lisbon, Portugal
thomasdpberry@tecnico.ulisboa.pt

***Abstract*—There are a wide range of techniques available for cooperative motion planning of autonomous vehicles. This paper focuses on one particular type of methods, known as *Direct Methods*, that solve motion planning expressed as a continuous-time optimal control problem. Out of all of the Direct Methods that were investigated, methods based on Bernstein polynomials were shown to provide the best results, namely, good trajectories that prevent inter-vehicular and environmental collisions while keeping cost approximately optimal.**

***Index Terms*—Motion Planning, Cooperative Autonomous Vehicles, optimization, Bézier curves, Differentially Flat systems**

I. Introduction

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, each with its own advantages and disadvantages, which make them best-suited for certain types of problems. Some of the Direct Methods that were explored included Single Shooting [1], which consists of parameterizing, for a given system, the control input signal as a piece-wise constant function of time. This method produces good results when a sufficiently low step duration is used. However, it can be slow to converge resulting in significant computational time. Multiple Shooting [1] uses the same kind of parameterization as Single Shooting but solves an ordinary differential equation for each time interval and introduces equality constrains which will increase the sparsity search space of the overall problem. Quadratic programming was also considered, but it is limited to linear constraints with a quadratic cost [2]. Monomial based polynomial methods [3] were the next best candidates, because they could work for a wide range of dynamic systems, and are specially suited for *differentially flat* systems [3]. Finally, Bernstein polynomial based methods were the focus of this study because 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.

II. Bézier Curves

A Bézier curve is a parametric curve used in computer graphics and related fields. The curve, is named after Pierre Bézier, who used it in the 1960s for designing curves for the bodywork of Renault cars. [4] Other uses include the design of computer fonts and animation. [4] The mathematical basis for Bézier 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. [5]

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

A Bernstein polynomial is given by

$$p_N(\tau) = \sum_{k=0}^{N} \overline{p}_{k,N} B_{k,N}(\tau), \qquad \tau \in [0,1] \tag{1}$$

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

$$\tau = \frac{t}{T}, \qquad t \in [0,T]. \tag{2}$$

The $p_{k,N}$ are the polynomial coefficients or *control points*, and b_k^N are the *Bernstein Basis*, given by

$$B_k^N(\tau) = \binom{N}{k}(1-\tau)^{N-k}\tau^k \tag{3}$$

As a result, a Bernstein polynomial of order N is a linear combination of $(N+1)$ Bernstein basis with weights given by $\overline{p}_{k,N}$.

For a polynomial of order N, the $i \in 0..N$ control points can be represented in vector form, for example, with $\overline{p}_N$ given by

$$\overline{p}_N = \begin{bmatrix} 0 & 0.5 & 1 & 0.7 & 0.3 & -0.7 & -1 & -0.5 & -0.1 \end{bmatrix}^T, \tag{4}$$

the plot of the polynomial given by these coefficients is shown in figure 1. 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.

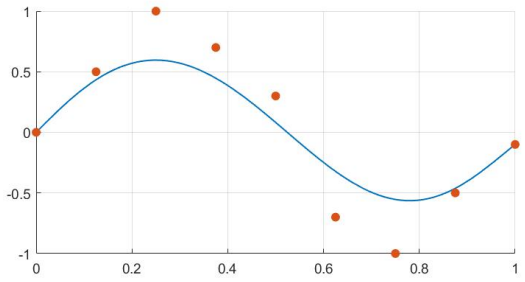

Fig. 1: A Bernstein Polynomial

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 Bézier curves. Figure 1 shows a 2-D Bézier 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 Bézier curve, such as the one in figure 3, despite the control points being the same on the 2-D plot.

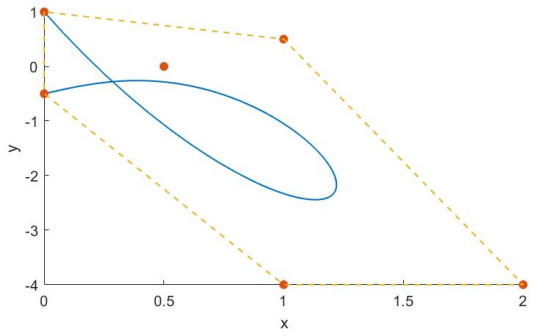

Fig. 2: A 2-D Bernstein Polynomial

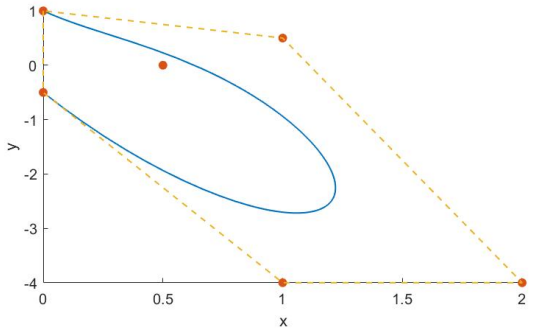

Fig. 3: A 2-D Bernstein Polynomial with Reordered Control Points

One property than can be observed in figure 1 is that, within the domain $t \in [0, T]$, the polynomial is limited by a convex hull [6] formed by the control points.

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

$$\begin{aligned} p_N(0) &= \overline{p}_{0,N} \\ p_N(T) &= \overline{p}_{n,N} \end{aligned} \tag{5}$$

and the derivative and integral are given by

$$\dot{\overline{p}}_{N-1}(t) = \frac{N}{T}\sum_{k=0}^{N-1}(p_{k+1,N} - p_{k,N})B_{k,N-1}(t) \tag{6}$$

$$\int_0^T \overline{p}(t)dt = \frac{T}{N+1}\sum_{k=0}^{N}\overline{p}_{k,N} \tag{7}$$

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

$$\boldsymbol{D}_{N-1} = \begin{bmatrix} -\frac{N}{T} & \frac{N}{T} & 0 & \dots & 0 \\ 0 & -\frac{N}{T} & \frac{N}{T} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & & \dots & -\frac{N}{T} & \frac{N}{T} \end{bmatrix} \in \mathbb{R}^{(N+1)\times N} \tag{8}$$

The control points of the Anti-Derivative/Primitive can be obtained similarly to the derivation by multiplying the control points to a primitive matrix [7], and adding a vector:

$$\overline{p}_N = A \cdot \dot{\overline{p}}_{N-1} + \overline{p}_{0,N} \tag{9}$$

where $\overline{p}_{0,N}$ is the initial values of p_N and the matrix A is given by

$$\boldsymbol{A}_{N+1} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ \frac{T}{N+1} & 0 & \dots & 0 & 0 \\ \frac{T}{N+1} & \frac{T}{N+1} & \dots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \\ \frac{T}{N+1} & \frac{T}{N+1} & \dots & \frac{T}{N+1} & \frac{T}{N+1} \end{bmatrix} \tag{10}$$

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

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

$$\overline{p}_{k,M} = \sum_{j=\max(0,k+M-N)}^{\min(N,k)} \frac{\binom{M-N}{k-j}\binom{N}{j}}{\binom{M}{k}} \overline{p}_{j,N} \in \mathbb{R}^n, \qquad (11)$$

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

$$\boldsymbol{E}_{ij} = \frac{\binom{N}{j}\binom{M-N}{i-j}}{\binom{M}{i}} \qquad (12)$$

Multiplication and addition are given by

$$f_M(t)g_N(t) = \sum_{i=0}^{M+N} \left(\sum_{i=\max(0,i-N)}^{\min(M,i)} \frac{\binom{M}{j}\binom{N}{i-j}}{\binom{M+N}{i}} \overline{f}_{j,M}\overline{g}_{i-j,N} \right) B_{i,M+N}(t) \qquad (13)$$

$$f_M(t) \pm g_N(t) = \sum_{i=0}^{M} \left(\overline{f}_{i,N} \pm \sum_{j=\max(0,i-M+N)}^{\min(N,i)} \frac{\binom{N}{j}\binom{M-N}{i-j}}{\binom{M}{i}} \overline{g}_{j,N} \right) B_{i,M+N}(t) \qquad (14)$$

The De Casteljau's algorithm [8] 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 4 illustrates the breakdown of the control points into polygons and sub polygons.

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

$$\begin{aligned} \overline{p}_{i,N}^{[0]} &= \overline{p}_{i,N}, \quad i = 0, \ldots, N \\ \overline{p}_{i,N}^{[j]} &= \overline{p}_{i,N}^{[j-1]} \frac{t_f - t_{div}}{t_f} + \overline{p}_{i+1,N}^{[j-1]} \frac{t_{div}}{t_f} \end{aligned} \qquad (15)$$

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

$$\overline{p}_N(t_{div}) = \overline{p}_{0,N}^{[N]}. \qquad (16)$$

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

$$\overline{p}_{0,N}^{[0]}, \overline{p}_{0,N}^{[1]}, \ldots, \overline{p}_{0,N}^{[N]}, \quad \text{and} \quad \overline{p}_{0,N}^{[N]}, \overline{p}_{0,N}^{[N-1]}, \ldots, \overline{p}_{0,N}^{[0]}. \qquad (17)$$

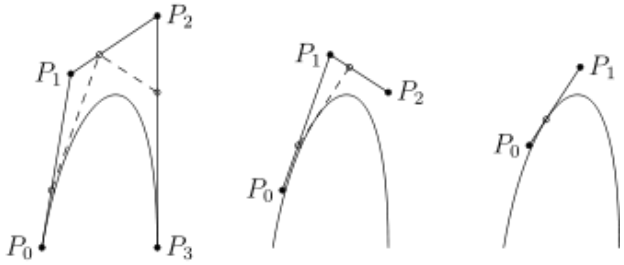

Fig. 4: Visual representation of the De Casteljau algorithm

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.

III. Models

The work carried out here focuses on movement along a 2-D plane, therefore, dynamics and kinematics are simplified such that there are only three degrees of freedom $[x, y, \psi]$.

The kinematics involves only the geometrical aspects of motion, and relates the velocities with position. They will take the form

$$\begin{aligned} \dot{x} &= u\cos\psi - v\sin\psi, \\ \dot{y} &= u\sin\psi + v\cos\psi, \\ \dot{\psi} &= r. \end{aligned} \qquad (18)$$

where x and y and ψ are x-position, y-position and orientation of the body-fixed frame $\{B\}$ relative to the earth-fixed inertial frame $\{U\}$. u (surge) and v (sway) the linear velocities of the origin of $\{B\}$ relative to $\{U\}$, expressed in $\{B\}$, and r is the turn rate of $\{B\}$ relative to $\{U\}$, expressed in $\{B\}$.

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, for the Medusa model, the dynamics equations for the rigid-body can be expressed as

$$\begin{aligned} m_u\dot{u} - m_v vr + d_u u &= \tau_u, \\ m_v\dot{v} + m_u ur + d_v v &= 0, \\ m_r\dot{r} - m_{uv}uv + d_r r &= \tau_r, \end{aligned} \qquad (19)$$

where

$$\begin{aligned} m_u &= m - X_{\dot{u}} \quad d_u = -X_u - X_{|u|u}|u| \\ m_v &= m - Y_{\dot{v}} \quad d_v = -Y_v - Y_{|v|v}|v| \\ m_r &= I_z - N_{\dot{r}} \quad d_r = -N_r - N_{|r|r}|r| \\ m_{uv} &= m_u - m_v. \end{aligned} \qquad (20)$$

$[m_u, m_v, m_r]$ refer to the rigid body intertia masses. In these equations, the external forces are given by $\tau = [\tau_u, \tau_r]^T$, which is is the vector of forces and torques due to thrusters/surfaces which usually can be viewed as the control input and $[-m_v vr + d_u u, m_u ur + d_v v, m_{uv} uv + d_r r]^T$ which represent hydrodynamic forces due to lift, drag, skin friction, etc.

The Medusa model can be even further simplified to a Dubins' car. The Dubins' car is a model for a vehicle which possess no side slip (lateral movement), which means that $v = 0$ always, and it also it also disregards hydrodynamic forces. As a result, only surge u and yaw rate r are provided as inputs. The Dubins' car model then becomes modelled only by the kinematic equations

$$\begin{aligned} \dot{x} &= u\cos\psi \\ \dot{y} &= u\sin\psi \\ \dot{\psi} &= \omega. \end{aligned} \tag{21}$$

IV. Cooperative Motion Planning: Problem Formulation and Adopted Solutions

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

A. The Optimisation Problem

The Motion Planning problem for multiple vehicles that are the focus of this project consists of a "go-to" formation manoeuvre [9]. It consists of the simultaneous arrival of a formation of vehicles to desired locations whilst simultaneously avoiding collisions between each other and the environment.

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

$$\begin{aligned} \underset{\substack{x^{[i]}(.),u^{[i]}(.)\\ i=1,\dots N_v}}{\text{minimize}} \quad & \sum_{i=1}^{N_v}\left(\int_0^T L_i(x^{[i]}(t),u^{[i]}(t))dt + \Psi(x^{[i]}(T))\right) \\ \text{subject to} \quad & x^{[i]}(0) = x_0^{[i]}, \\ & r^{[i]}(x^{[i]}(T)) = 0, \\ & \dot{x}^{[i]}(t) = f^{[i]}(x^{[i]}(t),u^{[i]}(t)), t\in[0,T] \\ & h(x(t),u(t)) \geq 0, \end{aligned} \tag{22}$$

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 $t \in [0; T]$.

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:

$$\begin{aligned} x^{[i]}(t) \approx x_N(t) &= \sum_{j=0}^{N} \overline{x}_{j,N}^{[i]} b_{j,N}(t), \quad t\in[0,T] \\ u^{[i]}(t) \approx u_N(t) &= \sum_{j=0}^{N} \overline{u}_{j,N}^{[i]} b_{j,N}(t), \quad t\in[0,T] \end{aligned} \tag{23}$$

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 $\overline{x}_N = [\overline{x}_{0,N}, \dots, \overline{x}_{N,N}]$, and $\overline{u}_N = [\overline{u}_{0,N}, \dots, \overline{u}_{N,N}]$

The optimisation problem now becomes

$$\begin{aligned} \underset{\substack{\overline{x}_N^{[i]},\overline{u}_N^{[i]}\\ i=1,\dots N_v}}{\text{minimize}} \quad & \sum_{i=1}^{N_v}\left(\int_0^T L^{[i]}(\overline{x}_N^{[i]},\overline{u}_N^{[i]})dt + \Psi(\overline{x}_{N,N})\right) \\ \text{subject to} \quad & \overline{x}_{0,N}^{[i]} = x_0^{[i]}, \\ & \overline{x}_{N,N}^{[i]} = x_f^{[i]}, \\ & \sum_{k=0}^{N} \boldsymbol{D}_{j,k}\overline{x}_{k,N}^{[i]} = f^{[i]}(\overline{x}_{j,N},\overline{u}_{j,N}), \forall j = 0,\dots,N \\ & h(x_N(t),u_N(t)) \geq 0, \end{aligned} \tag{24}$$

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 (8) with the Bernstein degree elevation matrix whose indices are given by (12).

What this new optimisation 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 22. If the the solution if is continuous, then problem 24 admits a feasible solution too. Moreover, let (x^*, u^*) be an optimal solution to 24 and $\{(x_N^*, u_N^*)\}_{N=N_1}^{\infty}$ be a sequence of optimal solution to problem 24, it can be proven that

$$\lim_{N\to\infty}(x_N^*(t), u_N^*(t)) = (x^*(t), u^*(t)), \tag{25}$$

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 (25) can be found in [6].

B. 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 optimisation problem of the form

$$\begin{aligned} \text{minimize} \quad & \int_0^T l(x(\tau), u(\tau), \tau)d\tau + m(x(T)) \\ \text{subject to} \quad & \dot{x}(t) = f(x(t)), u(t), t), x(0) = x_0 \\ & c_j(x(t)), u(t), t) \leq 0, t \in [0, T], \\ & j \in \{1, \ldots, k\} \end{aligned} \tag{26}$$

becomes

$$\begin{aligned} \text{minimize} \quad & \int_0^T l(x(\tau), u(\tau), \tau) + \\ & \sum_j \beta_\delta(-c_j(x(t), u(t), t))d\tau + m(x(T)) \\ \text{subject to} \quad & \dot{x}(t) = f(x(t)), u(t), t), \quad x(0) = x_0 \end{aligned} \tag{27}$$

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

$$\beta_\delta(z) = \begin{cases} -\log z & z > \delta \\ \frac{k-1}{k}\left[\left(\frac{z-k\delta}{(k-1)\delta}\right)^k - 1\right] - \log\delta & z \leq \delta \end{cases} \tag{28}$$

which is continuous and whose derivative around $z = \delta$ is also continuous too. [10].

The "hockey stick" function, with form given by

$$\sigma(z) = \begin{cases} \tanh(z), & z \leq 0 \\ z, & z < 0 \end{cases} \tag{29}$$

can be applied to the log barrier functional such that its derivative is smaller for feasible solutions when combined with β. Figure 5 shows the log barrier function with and without the hockey functional applied.

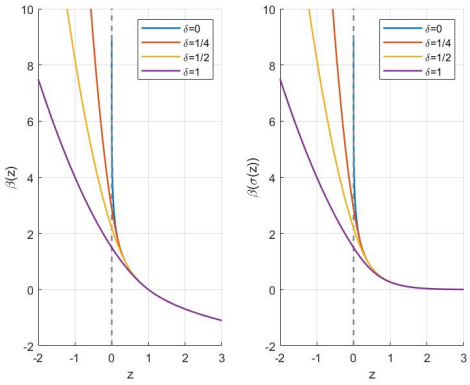

Fig. 5: log barrier functional without (left) and with (right) the hockey stick function

C. Inter-Vehicular Constraints

There are 2 ways of preventing inter-vehicle collision: *spatial deconfliction* and *temporal deconfliction* [11]. 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 vehicles will never be "at the same place at the same time". However, their spatial paths are allowed to intersect. 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.

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 optimisation 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

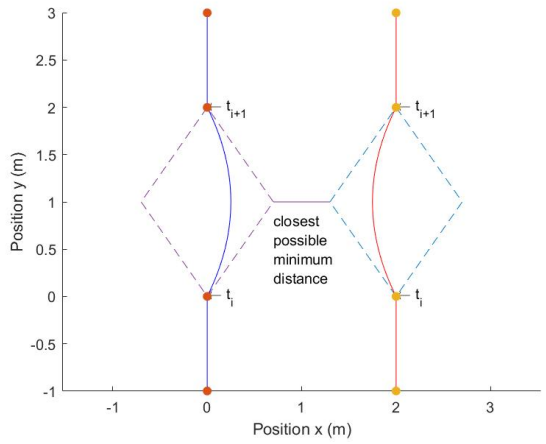

Fig. 6: Worst case scenario for two trajectories in the time interval between samples

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 6. 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 optimisation problem is reformulated to guarantee $1.01\,\mathrm{m}$of separation, then, with samples spaced by $10\,\mathrm{ms}$, a successful optimisation solution guarantees a minimum distance between vehicles of $1\,\mathrm{m}$. which means a total of 100 samples per second of runtime.

2) Bézier Curve Distance to a Point: A second method to calculate the minimum distance between two trajectories is based on calculating the minimum distance between Bézier curves. [12] 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 Bézier curves is explained in section II, 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 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 Gilbert–Johnson–Keerthi (GJK) algorithm [13]. 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.

D. Minimum Distance to Convex Shapes

Earlier, in section IV-C2, 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. Optimisation algorithms, such as Sequential Quadratic Programming[14], 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 7 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 IV-D1 is performed with the obstacle shape along a predefined direction $\overrightarrow{d}$. The bigger the penetration depth, the ”deeper” the trajectory is in the curve.

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) [15]. 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 $\overrightarrow{d}$, while the EPA algorithm finds the shortest vector such that the shapes no longer collide. The penetration along a direction is the length

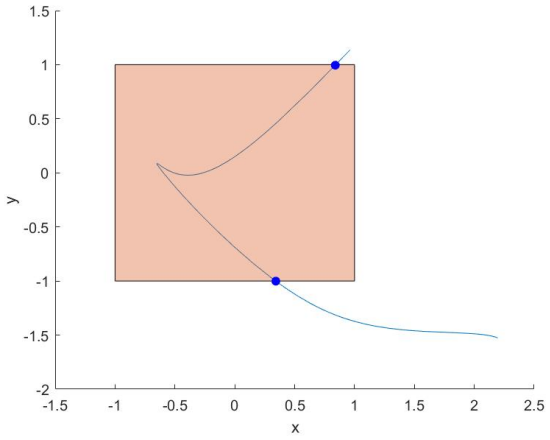

Fig. 7: A curve intersecting a convex shape

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 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 DEPA algorithm works in three basic steps: first, take the current polygon in the Minkowski which is guaranteed to contain the origin and find which edge intersects with the a ray that starts in the origin and has direction $\overrightarrow{d}$, second, once this edge is found, find a vector which is normal to this edge and points "outwards" w.r.t. the origin, and finally, perform the support function on the Difference with that vector. If the distance between the new point and one of the existing edges is below a tolerance, then the point has been found, otherwise, the point is added to the polygon and step one starts again. An initial polygon which is guaranteed to contain the origin can be found with the GJK algorithm.

E. Soundness of Solution

Once the optimisation process gets completed, the trajectory is defined along with, depending on the used model, the control points for the inputs which can then be re-plugged into the Ordinary Differential Equation (ODE) and check what resulting trajectory is obtained, this process is known as solving an Initial Value Problem (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, calculate too a correction term for the inputs based on the error with the desired position, such as what is done in Trajectory Tracking.

V. Results

The following results are based on solving the optimisation problem (24) with the implementation described in section IV. Simulations were carried out using Sequential Quadratic Programming [14] on models presented in section III. 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. Due to the convex hull property, upper and lower bounds for each variable for each vehicle were implemented by finding the biggest and smallest control points of each polynomial. For the examples presented here, the bounds that were used are those presented in table I which were chosen to represent realistic values of a vehicle.

	Variable	Lower Bounds	Upper Bounds
Dubins' Car	$u\ (\mathrm{m\,s^{-1}})$	0	1.1
	$r\ (\mathrm{rad\,s^{-1}})$	$-\pi/4$	$\pi/4$
Medusa	$u\ (\mathrm{m\,s^{-1}})$	0	1.1
	$v\ (\mathrm{m\,s^{-1}})$	$-\infty$	∞
	$r\ (\mathrm{rad\,s^{-1}})$	$-.74$	.74
	$\tau_u\ (\mathrm{N})$	0	25.9
	$\tau_r\ (\mathrm{N\,m})$	-.113	.113

TABLE I: Upper and lower bounds for each variable of each vehicle model

A range of experiments were performed with the optimisation 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. With exception of the position and heading, all of the examples presented in this section had initial and final conditions described by table II. Position and heading can be deduced from the plotted solutions.

Variable	Starting Conditions	Final Conditions
$u\ (\mathrm{m\,s^{-1}})$	1	1
$v\ (\mathrm{m\,s^{-1}})$	0	0
$r\ (\mathrm{rad\,s^{-1}})$	0	0

TABLE II: Initial and final conditions for the dynamic variables for all the vehicles of the examples

A. No obstacles

The first problem considered here is, for both a single Dubins' Car and a single Medusa vehicle, with initial and final states described in table II and with no obstacles. Figure 8 show solutions for order 20 and a time horizon of $60\,\mathrm{s}$. The top two plots minimize the square of the surge of the Dubins' 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.

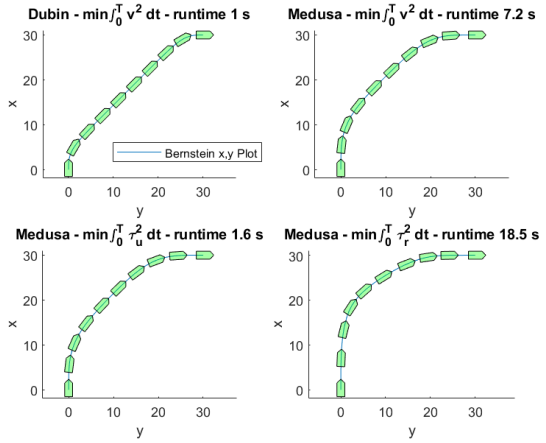

Fig. 8: Solutions of order $N = 20$ without obstacles and final time $T = 60$

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

B. Obstacles

Obstacle avoidance is implemented with the algorithms explained in sections IV-C and IV-D. The top left solution of figure 8 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 (II), and the same order of polynomial $N = 20$ for the Dubins' Car. The only difference is the presence of the different obstacles. Figure 9 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 optimisation constraint. The titles which say "Unconstrained" refer to plots resulting from unconstrained optimisation, 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 optimisation 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.

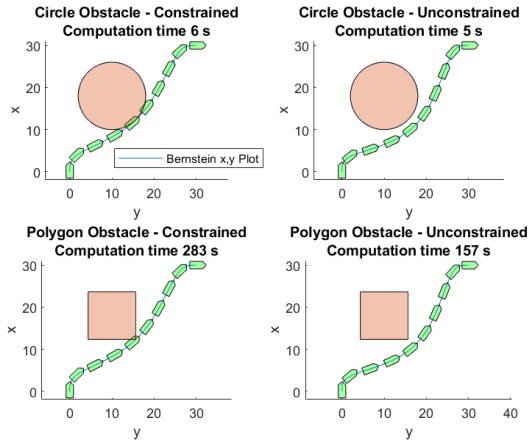

Fig. 9: Solution of order $N = 20$ with obstacles and final time $T = 60$

C. Variation of cost with order

A way of calculating solutions for high orders while maintaining low computation time is by running the optimisation 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 optimisation problem was is presented. It uses the Dubins' Car but with initial and final conditions of the dynamics given by table II.

Figure 10 shows the first, last, and two intermediary steps of this process. Figure 11 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 12 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,

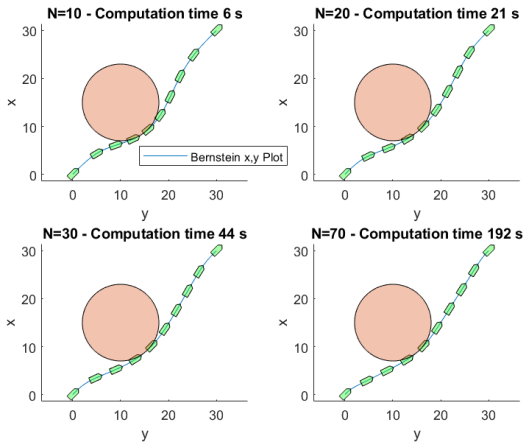

Fig. 10: Results of Interatively increasing order

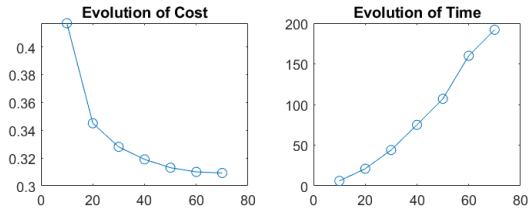

Fig. 11: Evolution of Cost and Time with Order Increase

which integrates the speed and turn rate as explained in section IV-E. 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.

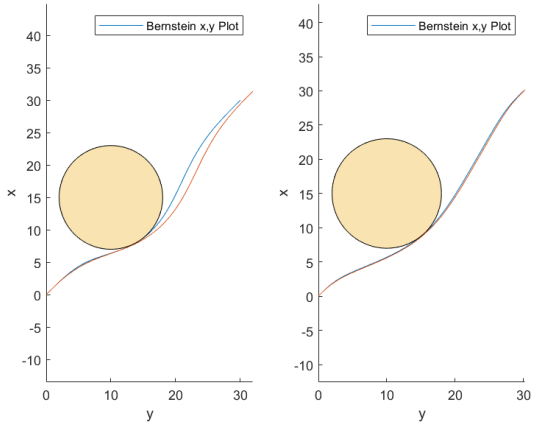

Fig. 12: Optimisation solutions accompanied by the Initial Value Solutions

D. Multiple Vehicles

To illustrate the use of multiple vehicles, a series of runs are presented in figure 13. The initial and final conditions of the dynamics variables for each vehicle are given by table II with order $N = 10$. These solutions are run with the inter vehicular constraints implemented as constraints on the optimisation problem, as opposed to imposing them in the cost via the Log Barrier Functional. Specifically, the imposed constraint is a separation of $3\,\mathrm{m}$ 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 a total of $56\,\mathrm{s}$ with the iterative *Berstein to a point* algorithm which again shows how using an iterative algorithm with an optimisation problems adds a lot of computation time.

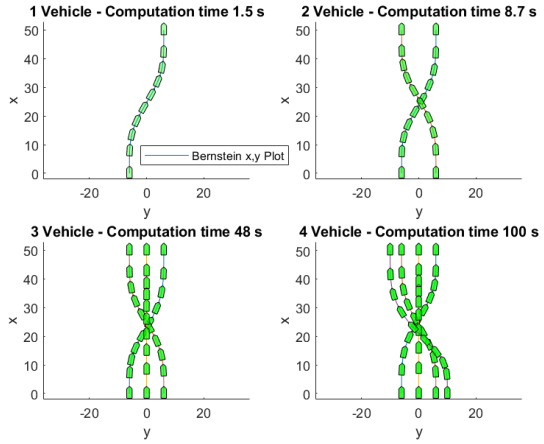

Fig. 13: Solutions of order $N = 10$ with multiple vehicles

E. Trajectory Tracking Soundness

In section IV-E, 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 14 shows the solution of a constrained optimisation problem for the Medusa Vehicle with order $N = 50$, with initial and final conditions for the dynamics variables given by table II.

The computation time of the optimisation problem is $113\,\mathrm{s}$. It also shows the result of applying the Trajectory Tracking controller in [16]. The inputs, which are surge and torque, are obtained by the optimisation problem and the Trajectory Tracking algorithm are shown in figure 15. It can be seen that the resulting inputs look similar, which validates the solution of the optimisation problem.

F. Final Problem

A final example of the application is shown in figure 16. It combines the various algorithms for inter-vehicle collision and

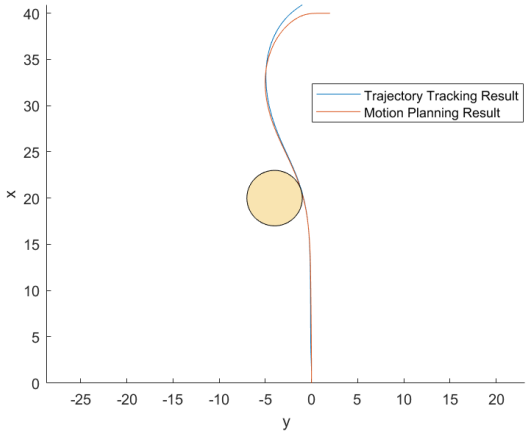

Fig. 14: Optimisation Problem Solution and Trajectory Tracked Solution

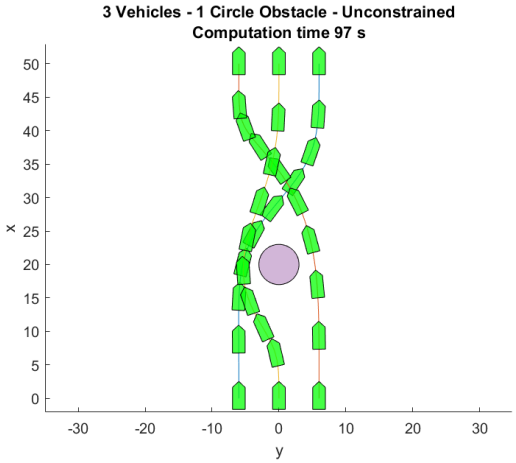

Fig. 16: Solution of order $N = 10$ with 3 vehicles and 1 circle obstacle

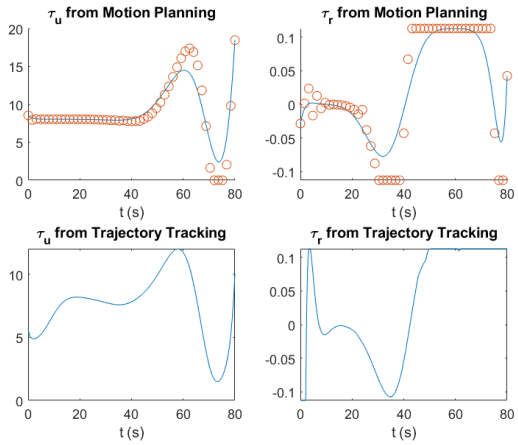

Fig. 15: Comparison of the Optimisation Problem's Inputs and the Trajectory Tracking Inputs

obstacle avoidance and the use of the Log Barrier Functional to achieve the fastest possible computation time.

VI. 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. Bézier 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 Autonomous Marine Vehicle (AMV) models: the Dubins' 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.

References

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

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

[3] 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.

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

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

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

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

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

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

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

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