




## Article

# Bernstein Polynomial-Based Method for Solving Optimal Trajectory Generation Problems

Calvin Kielas-Jensen <sup>1,\*</sup> , Venanzio Cichella <sup>1</sup> , Thomas Berry <sup>2</sup>, Isaac Kaminer <sup>3</sup>, Claire Walton <sup>4</sup> and Antonio Pascoal <sup>2</sup> 

<sup>1</sup> Cooperative Autonomous Systems (CAS) Lab, Department of Mechanical Engineering, University of Iowa, Iowa City, IA 52242, USA; venanzio-cichella@uiowa.edu

<sup>2</sup> Laboratory of Robotics and Engineering Systems (LARSyS), ISR/IST, University of Lisbon, 1049-001 Lisbon, Portugal; thomasdperry@tecnico.ulisboa.pt (T.B.); antonio@isr.tecnico.ulisboa.pt (A.P.)

<sup>3</sup> Department of Mechanical and Aerospace Engineering, Naval Postgraduate School, Monterey, CA 93943, USA; kaminer@nps.edu

<sup>4</sup> Department of Electrical Engineering, University of Texas at San Antonio, San Antonio, TX 78249, USA; claire.walton@utsa.edu

\* Correspondence: calvin-kielas-jensen@uiowa.edu

**Abstract:** This paper presents a method for the generation of trajectories for autonomous system operations. The proposed method is based on the use of Bernstein polynomial approximations to transcribe infinite dimensional optimization problems into nonlinear programming problems. These, in turn, can be solved using off-the-shelf optimization solvers. The main motivation for this approach is that Bernstein polynomials possess favorable geometric properties and yield computationally efficient algorithms that enable a trajectory planner to efficiently evaluate and enforce constraints along the vehicles' trajectories, including maximum speed and angular rates as well as minimum distance between trajectories and between the vehicles and obstacles. By virtue of these properties and algorithms, feasibility and safety constraints typically imposed on autonomous vehicle operations can be enforced and guaranteed independently of the order of the polynomials. To support the use of the proposed method we introduce BeBOT (Bernstein/Bézier Optimal Trajectories), an open-source toolbox that implements the operations and algorithms for Bernstein polynomials. We show that BeBOT can be used to efficiently generate feasible and collision-free trajectories for single and multiple vehicles, and can be deployed for real-time safety critical applications in complex environments.

**Keywords:** optimal trajectory generation; Bernstein polynomials; Bézier curves; optimal control



**Citation:** Kielas-Jensen, C.; Cichella, V.; Berry, T.; Kaminer, I.; Walton, C.; Pascoal, A. Bernstein Polynomial-Based Method for Solving Optimal Trajectory Generation Problems. *Sensors* **2022**, *22*, 1869. <https://doi.org/10.3390/s22051869>

Academic Editor: Roberto Teti

Received: 6 December 2021

Accepted: 2 February 2022

Published: 27 February 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The field of autonomous guidance has exploded in the past decade. Significant progress has been made in self driving vehicles, bringing them one step closer to reality [1]. Precision agriculture utilizes autonomous aerial vehicles to monitor crops and spray pesticides [2], and the development in autonomous weed pulling robots may reduce or eliminate the need for potentially harmful pesticides [3]. Underactuated marine surface vehicles can be controlled using a flatness-based approach [4]. Companies such as Amazon, Starship, and Zipline have already begun making autonomous deliveries [5–7]. In fact the first autonomous aerial vehicle has already been flown on a different planet [8]. This progress has led to high demand for computationally efficient algorithms that may yield safe and optimal trajectories to be planned for groups of autonomous vehicles. Our proposed method aims to accomplish these tasks by formulating the optimal trajectory generation problem as a nonlinear programming problem and exploiting the useful features of Bernstein polynomials.

Most techniques for planning and control of autonomous systems fall into one of two categories: closed-loop methods or open-loop methods. Closed-loop methods, sometimes referred to as feedback or reactive methods, use the current state knowledge to

determine, in real time, what the next control input should be. On the other hand, open-loop methods determine control values or compute motion trajectories out to a specified time horizon with the use of the system's model.

One common closed-loop technique that originally stemmed from maze solving algorithms is the bug algorithm. The bug algorithm, e.g., [9,10], uses local knowledge of the environment and a global goal to either follow a wall or move in a straight line towards the goal. This algorithm can be implemented on very simple devices due to typically requiring only two tactile sensors. However, it does not account for a vehicle's dynamics and constraints. Moreover, bug algorithms are non-optimal methods and cannot be used for the execution of complex missions that require the optimization of some cost. For a review and comparison of bug algorithms, the reader is referred to [11].

Rather than working on an agent's positions, the velocity obstacle (VO) algorithm uses relative velocities between the agent and obstacles to determine trajectories which will avoid collisions. The original term velocity obstacle was presented in [12]. Variations on the VO method include Common Velocity Obstacle [13], Nonlinear Velocity Obstacles [14], and Generalized Velocity Obstacles [15]. Other relevant closed-loop methods use artificial potential fields, which leverage a potential function providing attractive forces towards the goal and repulsive forces away from obstacles [16–18].

Among the advantages of closed-loop methods are fast computation and the ability to react to changing environments and unforeseen events. Furthermore, theoretical tools aimed at deriving safety guarantees of closed-loop methods are fairly well developed, and are mostly rooted in nonlinear systems analysis and robust and adaptive control. Despite these benefits, closed-loop methods are difficult to employ for multiple vehicle teams. They also generally lack the capability of presenting a human operator with a predicted trajectory and act rather like a black box, which can result in a lack of trust between the operator and the autonomous system.

In contrast to closed-loop methods, open-loop methods can generate solutions in one-shot for the whole mission time, and are therefore able to present an operator with an intuitive representation of the future trajectory. This representation is typically shown as a 2D or 3D path and may also include speed, acceleration, and higher derivatives of the vehicle's motion. Randomized algorithms such as probabilistic roadmaps (PRMs) [19] and rapidly exploring random trees (RRT, RRT\*) [20,21] randomly sample the work space to reduce computational complexity. PRMs randomly sample feasible regions within the work space to construct a dense graph. A graph-based solver can then be used to determine the optimal route. RRTs compute trajectories by using directed sampling to build trees. This approach can find feasible solutions in situations involving both a high number of constraints and high dimensional search spaces. Unfortunately, random sampling algorithms may be difficult to use in real-time applications due to computational complexity and may end up exploring regions that will not lead to a solution.

Similar to PRMs, other graph-based approaches aim to efficiently build and then search a graph. Cell decomposition methods, e.g., [22,23], build a graph of their environment by recursively increasing the resolution of areas of interest resulting in a few large nodes of open space and many small nodes near obstacles. Once a graph has been built, a graph solver can be used. A popular graph solver is the A\* algorithm [24], which is an extension of Dijkstra's algorithm that uses a heuristic function to improve the search speed. Many modifications to the A\* algorithm also exist, such as Lifelong Planning A\* [25], which replans a path anytime an obstacle appears on the existing path and utilizes Dijkstra's algorithm to transition the robot from its current pose to the new path. Similarly, Anytime Dynamic A\* (ADA\*) [26] iteratively decreases a suboptimality bound to improve the plan's optimality within a specified maximum computation time limit. Iterative approaches such as ADA\*, sometimes called "anytime" methods, compute a coarse solution and then refine it until a computation timeout is reached. For example, Ref. [27] investigates the addition of committed trajectories and branch-and-bound tree adaptation to the RRT\* algorithm to produce an online anytime method.

In addition to graph-based representations of trajectories, polynomial approximation methods can be used as well. In [28], trajectories are represented as piecewise polynomial functions and are generated in a manner that minimizes their snap. In TrajOpt [29], a sequential quadratic program is solved to generate optimal polynomial trajectories while performing continuous time collision checking.

Other open-loop methods include CHOMP [30,31], STOMP [32], and HOOP [33]. In CHOMP, infeasible trajectories are pulled out of collisions while simultaneously smoothing the trajectories using covariant gradient descent. STOMP adopts a similar cost function to that found in CHOMP, but generalizes to cost functions whose gradients are not available. This is done by stochastically sampling noisy trajectories. HOOP utilizes a problem formulation which computes vehicle trajectories in two steps. In the first step, a path is planned quickly by considering only the vehicle's kinematics. The second step then refines this trajectory into a higher order piecewise polynomial using a quadratic program. Other methods, such as WASPAS-mGqNS [34], balance the optimality of motion plans with respect to the mission objectives against exploring unknown environments.

Open-loop methods provide useful tools for dealing with high dimensional problems such as multiple vehicles and several constraints. They are also capable of producing trajectories that accomplish multiple goals. However, due to the curse of dimensionality, the computational complexity of open-loop methods grows significantly with the number of vehicles, constraints, and goals. For the most part, motion planning methods trade optimality and/or safety for computational speed. Our goal is to introduce a method that mitigates this trade-off, and that provides provably safe solutions for high dimensional problems while retaining the computational efficiency of low-order trajectory planning algorithms. This is achieved by exploiting the useful features of Bernstein polynomials.

The Bernstein basis was originally introduced by Sergei Natanovich Bernstein (1880–1968) in order to provide a constructive proof of Weierstrass's theorem. Bernstein polynomials were not widely used until the advent of digital computers due to their slow convergence as function approximants. Widespread adoption eventually occurred when it was realized that the coefficients of Bernstein polynomials could be intuitively manipulated to change the shape of curves described by these polynomials. In the 1960s, two French automotive engineers became interested in using this idea: Paul de Faget de Casteljau and Pierre Étienne Bézier.

Designing complex shapes for automobile bodies by sculpting clay models proved to be a time consuming and expensive process. To combat this, de Casteljau and Bézier sought to develop mathematical tools that would allow designers to intuitively construct and manipulate complex shapes. Due to de Casteljau publishing most of his research internally at his place of employment, Bézier's name became more widely associated with Bernstein polynomials, frequently referred to as Bézier curves. Building on existing research and modern technology, Bernstein polynomials provide several useful properties for many fields.

The Bernstein basis provides numerical stability [35], as well as useful geometric properties and computationally efficient algorithms that can be used to derive and implement efficient algorithms for the computation of trajectory bounds, trajectory extrema, minimum temporal and spatial separation between two trajectories and between trajectories and obstacles, and collision detection. Bernstein polynomials also allow for the representation of continuous time trajectories using low-order approximations.

Our method for trajectory generation builds upon [36–38], where Bernstein polynomials were introduced as a tool to approximate the solutions of nonlinear optimal control problems with provable convergence guarantees. While the results concerning the convergence of the Bernstein approximation method are out of the scope of the present paper, here we focus on the design of algorithms and functions for Bernstein polynomials. These include evaluating bounds, minimum spatial distance, collision detection, and penetration distance. Additionally, we show how these properties can be used for trajectory generation in realistic mission scenarios such as trajectory generation for swarms, navigating cluttered

environments, and motion planning for vehicles operating in a Traveling Salesman mission. For the interested reader, an open-source implementation is provided. This paper extends the results initially presented in [39]. In particular, in [39] we focused on autonomous vehicle trajectories representation by Bernstein polynomials, and proposed a preliminary implementation of BeBOT for minimum distance computation and collision detection for safe autonomous operation. In the present paper, we extend previous work by exploiting properties and proposing algorithms for both Bernstein polynomials and rational Bernstein polynomials. BeBOT includes an open-source Python implementation of these algorithms, which enables the user to exploit the properties of (rational) Bernstein polynomials for trajectory generation. Furthermore, we address several applications for multiple autonomous systems and show the efficacy of BeBOT in enabling safe autonomous operations. We added a new algorithm, the penetration algorithm, and several additional examples including air traffic control, navigation of a cluttered environment, vehicle overtaking, 1000 vehicle swarming, a marine vehicle example, and two examples of a vehicle routing problem. An implementation of the examples presented in this paper is available at our GitHub website [40] and can be customized to facilitate the toolbox's usability.

The goal of this manuscript is to provide a general framework which can be applied to a plethora of different systems ranging from mobile robots to manipulators. However, we do provide several numerical examples for mobile robots and include the governing motion equations for ease of implementation, e.g., see examples in Sections 5.1 and 5.6.

In brief, the main contributions of this article are:

1. Novel algorithms which exploit the useful properties of (rational) Bernstein polynomials for use in trajectory generation.
2. Several examples implementing the aforementioned algorithms in realistic mission scenarios.

The paper is structured as follows. In Section 2 we introduce Bernstein polynomials and their properties. Section 3 introduces the use of Bernstein polynomials to parameterize 2D and 3D trajectories. In Section 4 we present computationally efficient algorithms for the computation of state and input constraints typical of trajectory generation applications. In Section 5 we demonstrate the efficacy of these algorithms through several numerical examples. The paper ends with Section 6, which draws some conclusions. A Python implementation of the properties and algorithms presented, as well as the scripts used to generate the plots and examples found throughout this paper, can be found on our GitHub webpage [40].

In what follows, vectors are denoted by bold letters, e.g.,  $\mathbf{p} = [p_x, p_y]^T$  and  $\|\cdot\|$  denotes the Euclidean norm (or magnitude), e.g.,  $\|\mathbf{p}\| = \sqrt{p_x^2 + p_y^2}$ .

## 2. Mathematical Preliminaries

The motion planning problems addressed in this work can be in general formulated as optimal control problems. Letting the states and control inputs of the vehicles be denoted by  $\mathbf{x}(t)$  and  $\mathbf{u}(t)$ , respectively, the optimal motion planning problem can formally be stated as follows:

$$\min_{\mathbf{x}(t), \mathbf{u}(t)} I(\mathbf{x}(t), \mathbf{u}(t)) = E(\mathbf{x}(0), \mathbf{x}(t_f)) + \int_0^{t_f} F(\mathbf{x}(t), \mathbf{u}(t)) dt \quad (1)$$

subject to

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \forall t \in [0, t_f], \quad (2)$$

$$\mathbf{e}(\mathbf{x}(0), \mathbf{x}(t_f)) = \mathbf{0}, \quad (3)$$

$$\mathbf{h}(\mathbf{x}(t), \mathbf{u}(t)) \leq \mathbf{0}, \quad \forall t \in [0, t_f], \quad (4)$$



where  $I : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$ ,  $E : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ ,  $F : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$ ,  $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_x}$ ,  $e : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_e}$ , and  $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_h}$ .

Here,  $I$  defined in Equation (1) is a Bolza-type cost functional, with end point cost  $E$  and running cost  $F$ . The constraint in Equation (2) enforces the dynamics of the vehicles considered, Equation (3) enforces the boundary conditions, e.g., initial and final position, speed, heading angles of the vehicles, and Equation (4) describes feasibility and mission specific constraints, e.g., minimum and maximum speed, acceleration, collision avoidance constraints, etc.

In previous work [36–38] we presented a discretization method to approximate state and input by  $n$ th order Bernstein polynomials. This approximation allows us to transcribe the optimal control problem into a non-linear programming problem, which can then be solved by off-the-shelf optimization solvers. In particular, we show that the solution to the non-linear programming problem converges to the solution of the original optimal control problem as  $n$  increases. The present paper focuses on the geometric properties of Bernstein polynomials and their implementation for computationally efficient and safe trajectory generation. In the following, we report the properties of Bernstein polynomials and rational Bernstein polynomials which are relevant to this paper.

An  $n$ th order Bernstein polynomial defined over an arbitrary interval  $[t_0, t_f]$  is given by

$$\mathbf{C}_n(t) = \sum_{i=0}^n \mathbf{P}_{i,n} B_{i,n}(t), \quad t \in [t_0, t_f], \quad (5)$$

where  $\mathbf{P}_{i,n} \in \mathbb{R}^D$  is the  $i$ th Bernstein coefficient,  $D$  is the number of dimensions, and  $B_{i,n}(t)$  is the Bernstein polynomial basis defined as

$$B_{i,n}(t) = \binom{n}{i} \frac{(t - t_0)^i (t_f - t)^{n-i}}{(t_f - t_0)^n}, \quad \binom{n}{i} = \frac{n!}{i!(n-i)!},$$

for all  $i = 0, \dots, n$ . Typically the dimensionality of a Bernstein polynomial,  $D$ , is either 2 or 3 for 2D or 3D spatial curves, respectively. In this case, Bernstein polynomials are often referred to as Bézier curves and their Bernstein coefficients are known as *control points*. While Bézier's original work did not explicitly use the Bernstein basis [41,42], it was later shown that the original formulation is equivalent to the Bernstein form polynomial [43].

An  $n$ th order *rational* Bernstein polynomial,  $R_n(t)$ , is defined as

$$R_n(t) = \frac{\sum_{i=0}^n P_{i,n} w_{i,n} B_{i,n}(t)}{\sum_{i=0}^n w_{i,n} B_{i,n}(t)}, \quad t \in [t_0, t_f], \quad (6)$$

where  $w_{i,n} \in \mathbb{R}$ ,  $i = 0, \dots, n$ , are referred to as weights. A list of relevant properties of Bernstein polynomials used throughout this article can be found in Appendix A.

### 3. Generation of 2D and 3D Trajectories Using (Rational) Bernstein Polynomials

#### 3.1. 2D Trajectories

Here we will examine several illustrative examples of the properties of Bernstein polynomials and rational Bernstein polynomials in 2D. All the plots presented can be generated using the example code available at [40].

Figures 1–9 contain several examples of 2D trajectories in the spatial domain. Two trajectories are plotted in Figure 1 along with an obstacle. The trajectories  $\mathbf{C}^{[1]}(t)$  and  $\mathbf{C}^{[2]}(t)$  are defined as in Equation (5) with  $t_0 = 10$  s and  $t_f = 20$  s where the Bernstein coefficients are temporally equidistant. The vector of Bernstein coefficients of trajectory  $\mathbf{C}^{[1]}(t)$  is

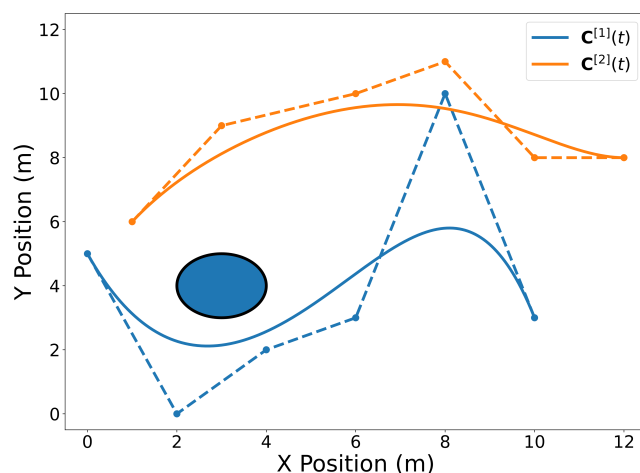
$$\mathbf{P}_5^{[1]} = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 & 10 \\ 5 & 0 & 2 & 3 & 10 & 3 \end{bmatrix}.$$

The vector of Bernstein coefficients of trajectory  $\mathbf{C}^{[2]}(t)$  is

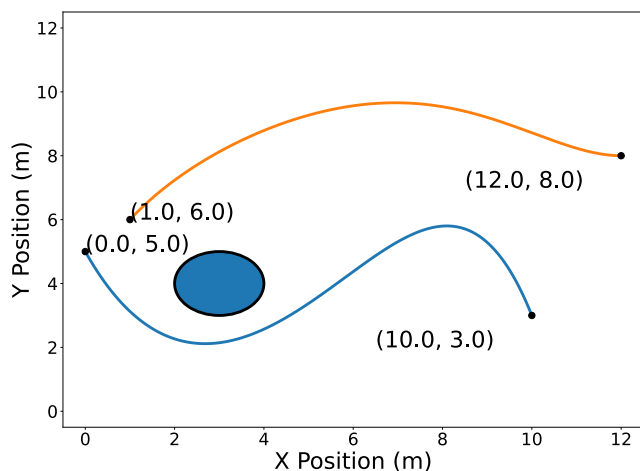
$$\mathbf{P}_5^{[2]} = \begin{bmatrix} 1 & 3 & 6 & 8 & 10 & 12 \\ 6 & 9 & 10 & 11 & 8 & 8 \end{bmatrix}.$$

The circular obstacle has a radius of 1 and is centered at point  $[3, 4]^\top$ . Figure 2 highlights the endpoints property (Property A2). Note that the trajectory  $\mathbf{C}^{[1]}(t)$  passes through its first and last Bernstein coefficients  $\mathbf{P}_{0,5}^{[1]} = [0, 5]^\top$  and  $[10, 3]^\top$ , respectively. Likewise, the trajectory  $\mathbf{C}^{[2]}(t)$  passes through its first and last Bernstein coefficients  $[1, 6]^\top$  and  $[12, 8]^\top$ , respectively. The convex hull property (Property A1) is illustrated in Figure 3.

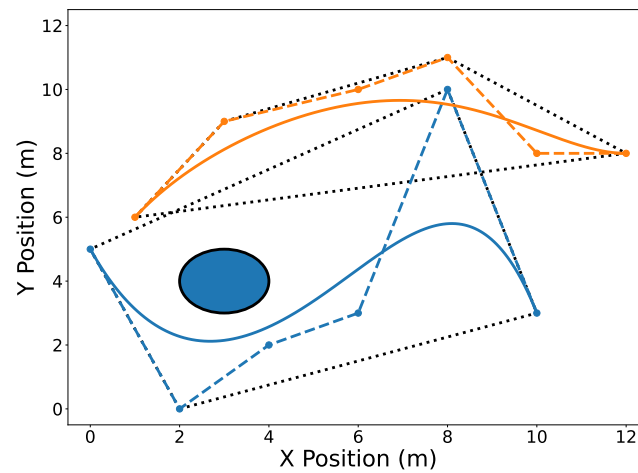
Useful operations can be efficiently performed on Bernstein polynomials by manipulating only their coefficients. The de Casteljau algorithm (Property A5) allows one to split a Bernstein polynomial into two separate polynomials. This is shown in Figure 4 where trajectories  $\mathbf{C}^{[1]}(t)$  and  $\mathbf{C}^{[2]}(t)$  are split at  $t_{div} = 15$  s. Degree elevation (Property A6) is performed on both trajectories in Figure 5. Note that in both Figures 4 and 5, the convex hulls are more accurate than the conservative convex hulls in Figure 3. This idea will be expanded upon in the next section.



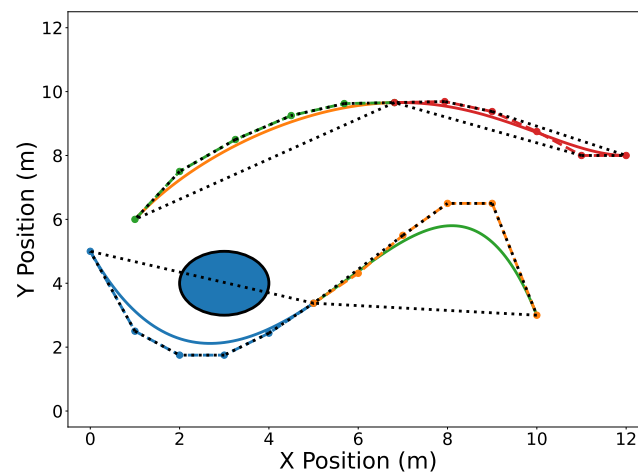
**Figure 1.** Spatial representation of two Bernstein polynomial trajectories in 2D near a circular obstacle. Trajectory  $\mathbf{C}^{[1]}(t)$  is drawn in blue and trajectory  $\mathbf{C}^{[2]}(t)$  is drawn in orange. The solid lines are the polynomials and the dashed lines connect the Bernstein coefficients for convenience. Note that the temporal aspect of the trajectories above has been omitted for clarity of presenting the geometric properties of Bernstein polynomials.



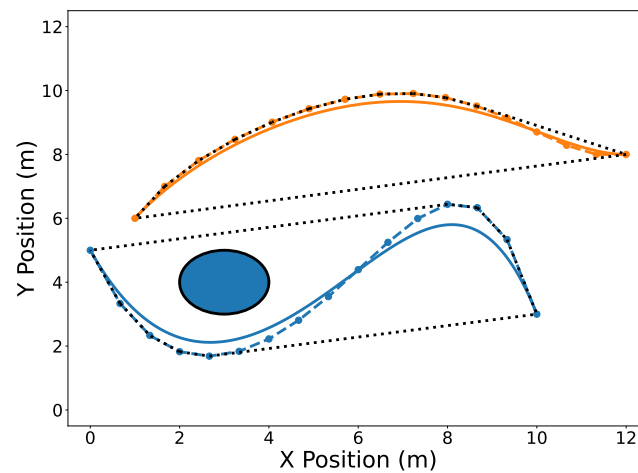
**Figure 2.** Trajectories  $\mathbf{C}^{[1]}(t)$  (blue) and  $\mathbf{C}^{[2]}(t)$  (orange) with their endpoints highlighted in 2D.



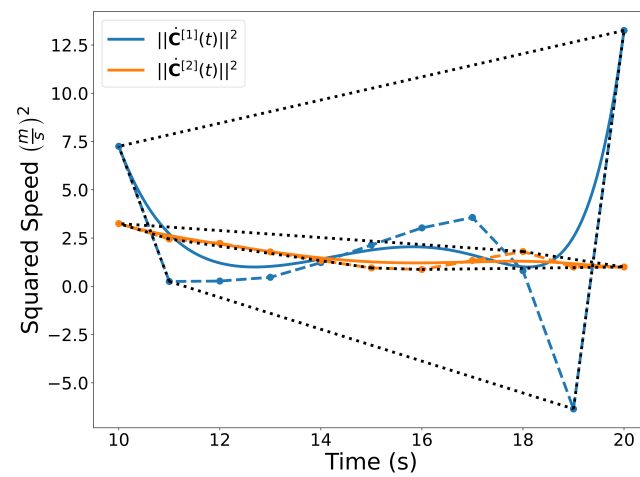
**Figure 3.** Convex hulls drawn as black dotted lines around the Bernstein coefficients of trajectories  $C^{[1]}(t)$  (blue) and  $C^{[2]}(t)$  (orange). The dashed lines connect the control points of their corresponding trajectories.



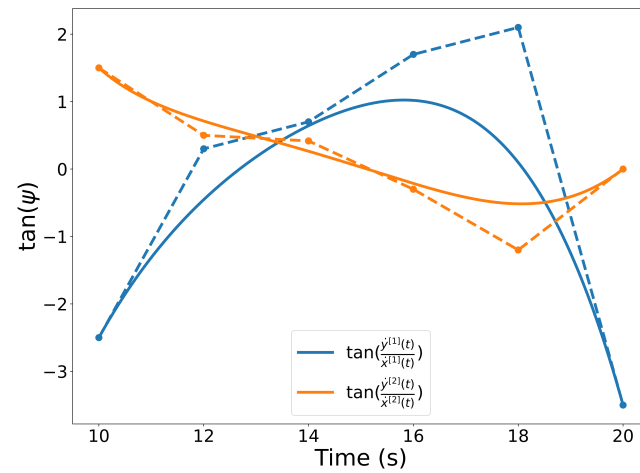
**Figure 4.** Trajectories  $C^{[1]}(t)$  (split into blue and green) and  $C^{[2]}(t)$  (split into orange and red) split at  $t_{div} = 15$  s. Convex hulls are drawn around the Bernstein coefficients of the new split trajectories.



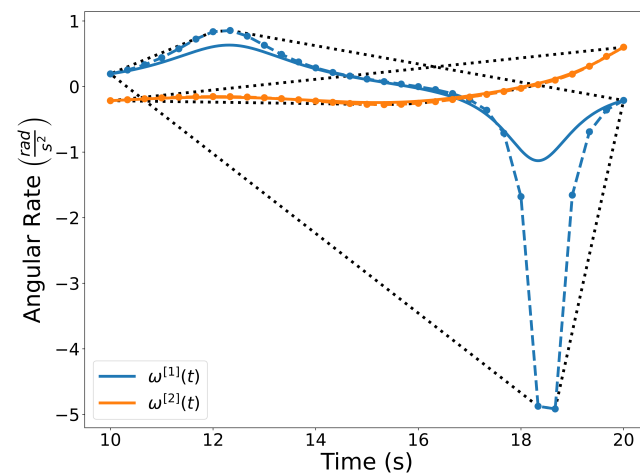
**Figure 5.** Convex hull drawn around the elevated Bernstein coefficients of trajectories  $C^{[1]}(t)$  (blue) and  $C^{[2]}(t)$  (orange).



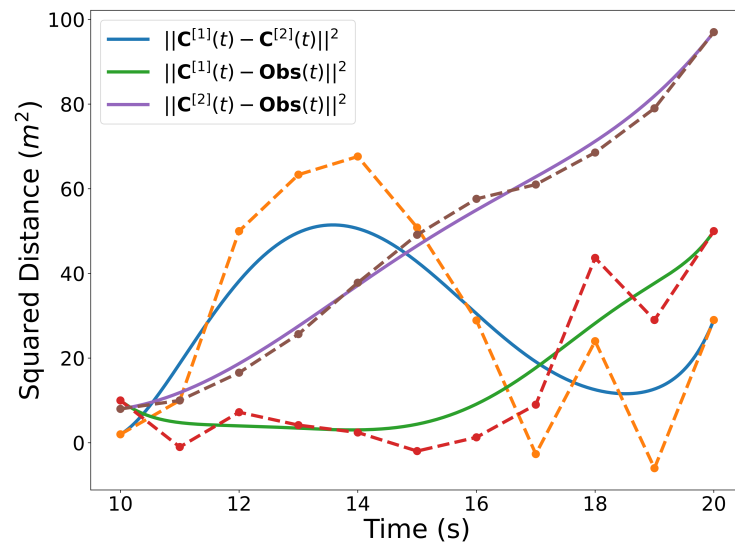
**Figure 6.** Squared speed of the trajectories  $C^{[1]}(t)$  and  $C^{[2]}(t)$ . A convex hull is drawn around the Bernstein coefficients. Note that even though the coefficients may be negative, the actual curve is not.



**Figure 7.** Illustration of the quantity expressed in Equation (8) for trajectories  $C^{[1]}(t)$  and  $C^{[2]}(t)$ .



**Figure 8.** Angular rates of trajectories  $C^{[1]}(t)$  and  $C^{[2]}(t)$ . Note that the angular rates are rational Bernstein polynomials.



**Figure 9.** Squared distance between trajectories and the center of the circular obstacle.

Bernstein polynomials can also be used to extract useful information about the dynamics of the trajectories. In Figure 6, Bernstein polynomials representing the squared speed of trajectories  $\mathbf{C}^{[1]}(t) = [x^{[1]}(t), y^{[1]}(t)]^\top$  and  $\mathbf{C}^{[2]}(t) = [x^{[2]}(t), y^{[2]}(t)]^\top$  are shown along with their corresponding coefficients and convex hulls. The squared speed is computed using the derivative and arithmetic operation properties (Properties A3 and A7) as follows

$$(v^{[1]}(t))^2 = (\dot{x}^{[1]}(t))^2 + (\dot{y}^{[1]}(t))^2$$

Note that the squared speed of a trajectory described by a Bernstein polynomial is also a Bernstein polynomial.

Letting  $\mathbf{C}^{[1]}(t) = [x^{[1]}(t), y^{[1]}(t)]^\top$ , the heading angle  $\psi(t)$  of a trajectory can be computed as

$$\psi^{[1]}(t) = \tan^{-1} \left( \frac{\dot{y}^{[1]}(t)}{\dot{x}^{[1]}(t)} \right). \quad (7)$$

Since the inverse tangent of a Bernstein polynomial is not a Bernstein polynomial, we take the tangent of both sides of the equation, yielding

$$\tan(\psi^{[1]}(t)) = \left( \frac{\dot{y}^{[1]}(t)}{\dot{x}^{[1]}(t)} \right), \quad (8)$$

which is a rational Bernstein polynomial. Figure 7 illustrates the quantity expressed in Equation (8) computed for trajectories  $\mathbf{C}^{[1]}(t)$  and  $\mathbf{C}^{[2]}(t)$ .

To determine the angular rate along the trajectory at any point in  $t \in [t_0, t_f]$ , we can take the derivative of the heading angle, yielding

$$\omega^{[1]}(t) = \dot{\psi}^{[1]}(t) = \frac{\dot{x}^{[1]}(t)\ddot{y}^{[1]}(t) - \ddot{x}^{[1]}(t)\dot{y}^{[1]}(t)}{(\dot{x}^{[1]}(t))^2 + (\dot{y}^{[1]}(t))^2}. \quad (9)$$

Since the angular rate can be determined using Properties A3 and A7, it can be represented as a rational Bernstein polynomial. The angular rates of trajectories  $\mathbf{C}^{[1]}(t)$  and  $\mathbf{C}^{[2]}(t)$  are shown in Figure 8.



Finally, the Bernstein polynomial representing the squared distance between two trajectories at every point in time can be computed from

$$d^2(t) = (x^{[2]}(t) - x^{[1]}(t))^2 + (y^{[2]}(t) - y^{[1]}(t))^2, \quad \forall t \in [t_0, t_f] \quad (10)$$

The center point of a circular, static obstacle  $\mathbf{Obs}(t)$ , can be represented as a Bernstein polynomial whose coefficients are all identical and set to the position of the obstacle, i.e.,

$$\mathbf{P}^{[Obs]} = \begin{bmatrix} x^{[Obs]} & \dots & x^{[Obs]} \\ y^{[Obs]} & \dots & y^{[Obs]} \end{bmatrix}.$$

The degree of the Bernstein polynomial representing the center point of the circular obstacle is equal to that of the order of the Bernstein polynomials representing the trajectories.

### 3.2. 3D Trajectories

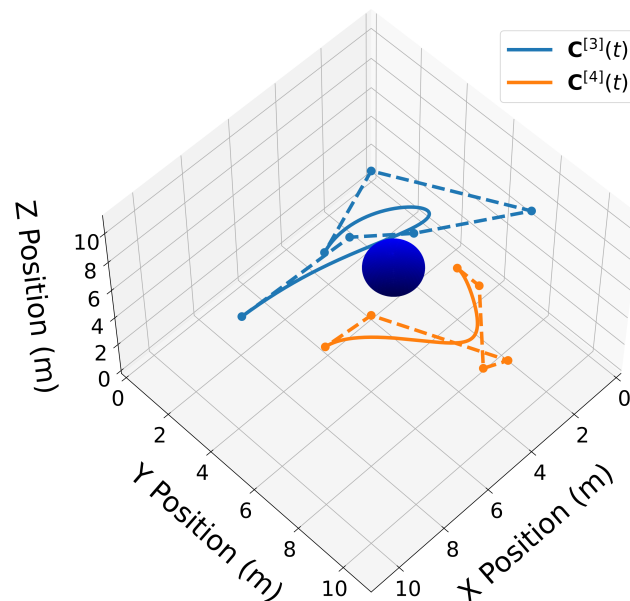
We now introduce two 3D Bernstein polynomials with  $t_0 = 10$  s and  $t_f = 20$  s, where the coefficients are equidistant in time, and illustrate their properties in Figures 10–17. The Bernstein coefficients of trajectory  $\mathbf{C}^{[3]}(t)$  are

$$\mathbf{P}_5^{[3]} = \begin{bmatrix} 7 & 3 & 1 & 1 & 3 & 7 \\ 1 & 2 & 3 & 8 & 3 & 5 \\ 0 & 2 & 1 & 9 & 8 & 10 \end{bmatrix},$$

and the Bernstein coefficients of trajectory  $\mathbf{C}^{[4]}(t)$  are

$$\mathbf{P}_5^{[4]} = \begin{bmatrix} 1 & 1 & 4 & 4 & 8 & 8 \\ 5 & 6 & 9 & 10 & 8 & 6 \\ 1 & 1 & 3 & 5 & 11 & 6 \end{bmatrix}.$$

These polynomials are drawn in Figure 10.



**Figure 10.** Two 3D Bernstein polynomial trajectories near a spherical obstacle. Trajectory  $\mathbf{C}^{[3]}(t)$  is drawn in blue and trajectory  $\mathbf{C}^{[4]}(t)$  is drawn in orange.

Similar to the 2D examples, Figures 11–14 illustrate the end points, convex hull, de Casteljau, and elevation properties, respectively. Figures 15 and 16 show the squared speed and squared acceleration of trajectories  $C^{[3]}(t)$  and  $C^{[4]}(t)$ , respectively. These values were computed using the derivative and arithmetic properties. Finally, Figure 17 shows the squared Euclidean distance between the trajectories and the center of the spherical obstacle at every point in time. The distance was found using Property A7.

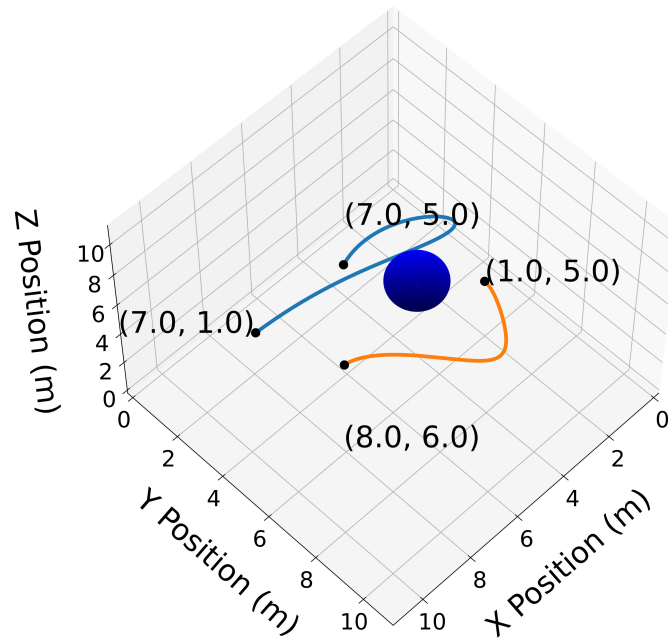


Figure 11. 3D trajectories  $C^{[3]}(t)$  and  $C^{[4]}(t)$  with their endpoints highlighted.

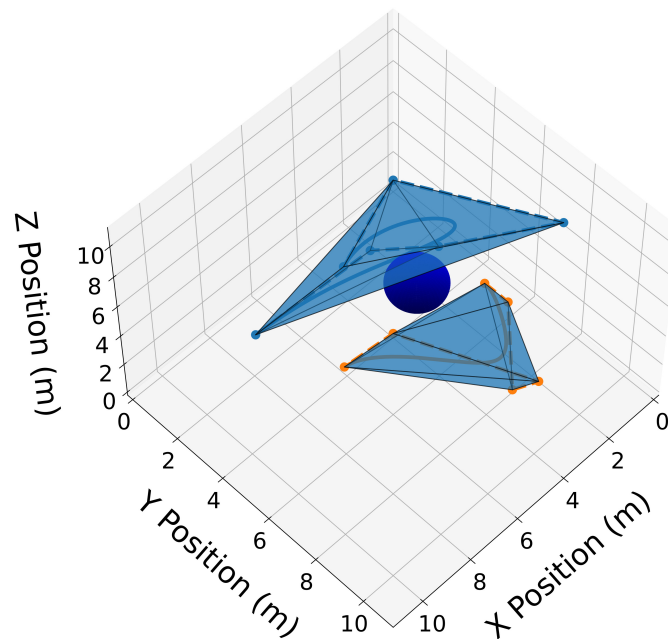
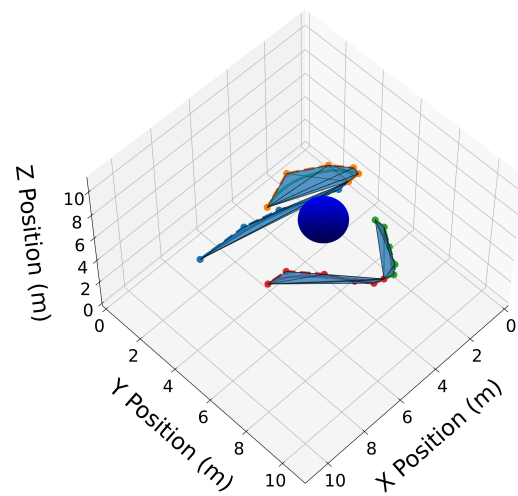
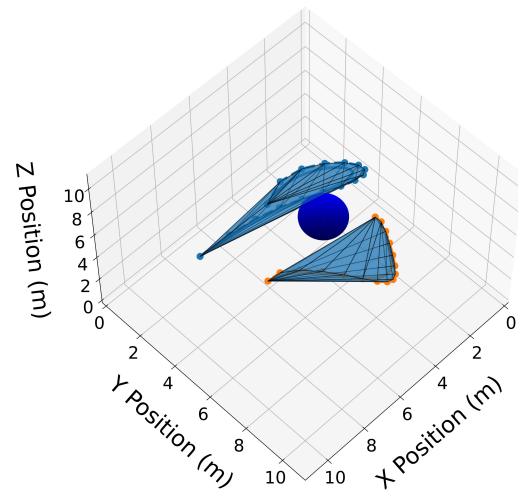


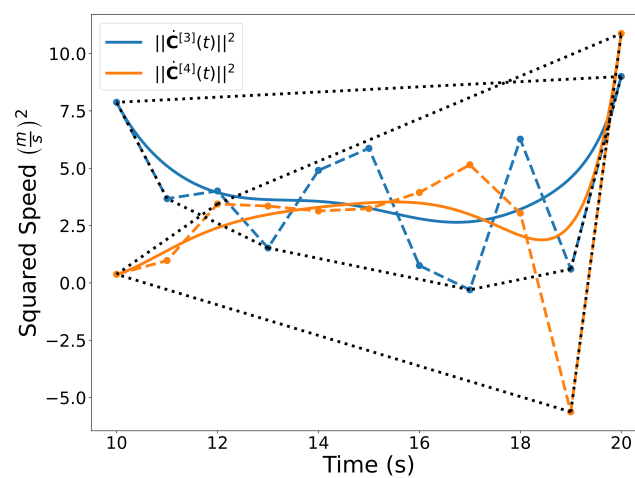
Figure 12. 3D convex hulls drawn as transparent blue surfaces around the Bernstein coefficients of trajectories  $C^{[3]}(t)$  and  $C^{[4]}(t)$ .



**Figure 13.** Trajectories  $C^{[3]}(t)$  and  $C^{[4]}(t)$  split at  $t_{div} = 15$  s. Convex hulls are drawn around the Bernstein coefficients of the new split trajectories.



**Figure 14.** Convex hull drawn around the elevated Bernstein coefficients of trajectories  $C^{[3]}(t)$  and  $C^{[4]}(t)$ .



**Figure 15.** Squared speed of the trajectories  $C^{[3]}(t)$  and  $C^{[4]}(t)$ . A convex hull is drawn around the Bernstein coefficients. Note that even though the coefficients may be negative, the actual curve is not.

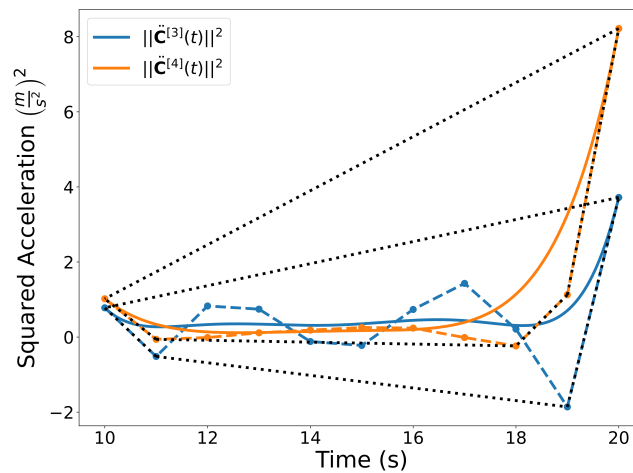


Figure 16. Squared acceleration of the trajectories  $\mathbf{C}^{[3]}(t)$  and  $\mathbf{C}^{[4]}(t)$  with corresponding convex hulls.

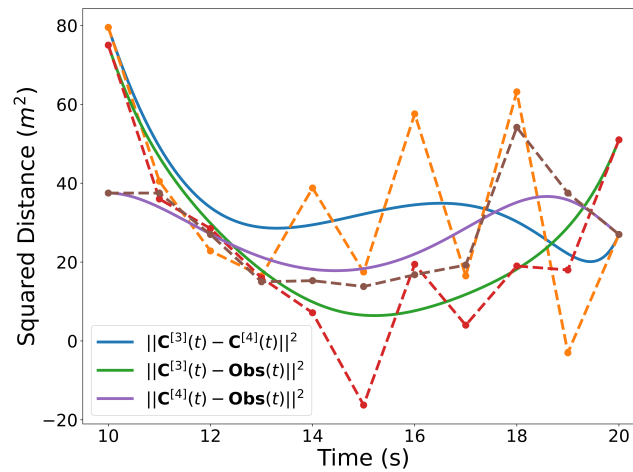


Figure 17. Squared distances between the trajectories and then center of the spherical obstacle.

#### 4. Algorithms for (Rational) Bernstein Polynomials

This section contains algorithms and procedures for Bernstein polynomials that make use of the properties presented in Section 2. These functions include: evaluating bounds, using the convex hull property to quickly find conservative bounds; evaluating extrema, through an iterative procedure that computes a solution within a desired tolerance; minimum spatial distance, applying a similar iterative procedure to find the minimum spatial distance between two Bernstein polynomials; and collision detection, which quickly determines whether a collision may exist or does not exist.

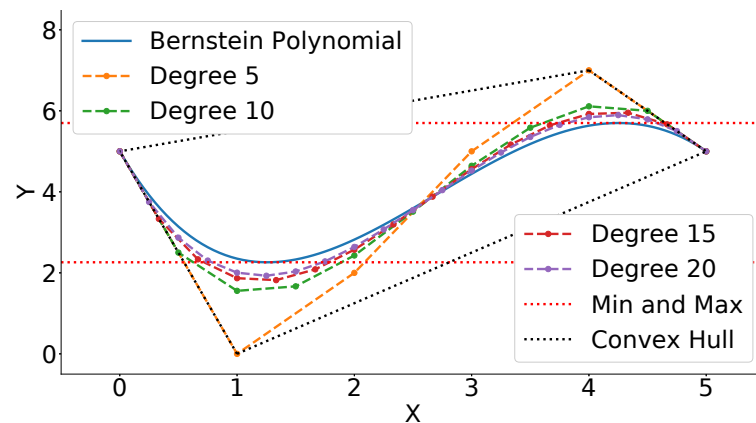
##### 4.1. Evaluating Bounds

Property A1 allows one to quickly establish conservative bounds on the Bernstein polynomial. For example, given the 2D Bernstein polynomial, see Equation (5), with coefficients given by

$$\mathbf{P}_5 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 5 & 0 & 2 & 5 & 7 & 5 \end{bmatrix},$$

lower and upper bounds  $C_{\min}$  and  $C_{\max}$  satisfying  $C_{\min} \leq C(t) \leq C_{\max}$ ,  $\forall t \in [t_0, t_f]$  can be derived using Equation (A1). Figure 18 exhibits the Bernstein polynomial (solid blue line) given the above coefficients (orange dots connected with dashes). The most conservative estimate of the minimum and maximum  $Y$  values of the Bernstein polynomial is given by the coefficients with the lowest and highest  $Y$  values, respectively. The lower bound is 0

and the upper bound is 7. As mentioned in Property A6 and Equation (A8), the Bernstein coefficients converge to the curve as the polynomial is degree elevated. This fact can be used to derive tighter bounds. A degree elevation of 20 results in a lower bound of 1.93 and an upper bound of 5.89. This is a closer estimate of the actual minimum and maximum, 2.26 and 5.70, respectively (see red dotted lines and Section 4.2). Figure 18 also illustrates degree elevations of 5, 10, and 15. Since the degree elevation matrix, see Equation (A7), is independent of the Bernstein coefficients, a database of elevation matrices can be computed ahead of time to produce tight estimates of the bounds at a low computational cost.



**Figure 18.** Bounds for Bernstein polynomials. The solid blue line is the Bernstein polynomial, the dashed lines connect the coefficients of each different order, the black dotted line represents the convex hull of the 5th degree Bernstein polynomial, and the red dotted lines represent the actual extrema.

#### 4.2. Evaluating Extrema

The extrema of a Bernstein polynomial are calculated using an iterative procedure similar to the one proposed in [44]. This is done by recursively splitting the curve and using the Convex Hull (Property A1) to obtain an estimate within some desired tolerance. Algorithm 1 outlines the process for determining the maximum of a Bernstein polynomial and can easily be modified to determine the minimum of a Bernstein polynomial.

The inputs to Algorithm 1 are the Bernstein polynomial's coefficients,  $\mathcal{P} = \{P_n\}$ ,  $P_n = [P_{0,n}, \dots, P_{n,n}]$ , an arbitrarily large *negative* global lower bound,  $\alpha$ , and a desired tolerance,  $\epsilon$ . Note that in order to compute a reliable maximum,  $\alpha \leq \min(P)$ . In practice,  $\alpha$  is set to the lowest possible value that can be reliably represented in the computer.

Line 1 finds the maximum of the two endpoints of the Bernstein polynomial, where  $n$  is the degree of the polynomial. This makes use of the End Points (Property A2). Next, we determine the upper bound by simply finding the maximum of  $\mathcal{P}$ . The *if* statement on line 3 determines whether the global lower bound should be replaced with the current lower bound. The next *if* statement, line 6, will prune the current set of Bernstein coefficients. This is valid because  $\alpha$  always provides a lower bound on the global maximum. If the upper bound of any subset is below  $\alpha$ , then we know that it is impossible for any point on that subset to be the global maximum. The final *if* statement, line 9, determines whether the difference between the upper and lower bounds is within the desired tolerance and returns the global minimum bound  $\alpha$  if the tolerance is met.

The *else* statement, starting on line 11, splits the curve and then recursively calls Algorithm 1 again. The function `split()` utilizes the de Casteljau algorithm (Property A5). One of two different splitting points,  $t_{div}$ , can be employed. The first option simply splits the curve in half, i.e.,  $t_{div} = t_0 + \frac{t_f - t_0}{2}$ . The second option uses the index of the largest valued coefficient,  $i_{ub} = \arg\max(\mathcal{P})$ , to determine the splitting point, i.e.,  $t_{div} = t_0 + (t_f - t_0) \frac{i_{ub}}{n}$ .

Algorithm 1 (and its converse) is employed to find the minimum and maximum of the 5th degree Bernstein polynomial depicted in Figure 18 (red lines). The execution time to compute the minimum is 320  $\mu$ s on a Lenovo ThinkPad laptop using an Intel Core i7-8550U, 1.80 GHz CPU. The implementation can be found in [40].



**Algorithm 1:** Evaluating Maximum Value of a Bernstein Polynomial

---

**Input:**  $\mathcal{P}, \alpha, \epsilon$

```

1  $P_{lb} = \max\{\mathcal{P}[0], \mathcal{P}[n]\}$ 
2  $P_{ub} = \max(\mathcal{P})$ 
3 if  $P_{lb} > \alpha$  then
4   |  $\alpha = P_{lb}$ 
5 end
6 if  $\alpha > P_{ub}$  then
7   | return  $\alpha$ 
8 end
9 if  $P_{ub} - P_{lb} < \epsilon$  then
10  | return  $\alpha$ 
11 else
12  |  $\mathcal{P}^A, \mathcal{P}^B = \text{split}(\mathcal{P})$ 
13  |  $\alpha_A = \text{Algorithm 1}(\mathcal{P}^A, \alpha, \epsilon)$ 
14  |  $\alpha_B = \text{Algorithm 1}(\mathcal{P}^B, \alpha, \epsilon)$ 
15  |  $\alpha = \max(\alpha_A, \alpha_B)$ 
16 end
17 return  $\alpha$ 

```

---

**4.3. Minimum Spatial Distance**

The minimum spatial distance between two Bernstein polynomials can be computed using the method outlined in [44]. This is done by exploiting the Convex Hull property (Property A1), the End Point Values property (Property A2), the de Casteljau Algorithm (Property A5), and the Gilbert-Johnson-Keerthi (GJK) algorithm [45]. The latter is widely used in computer graphics to compute the minimum distance between convex shapes.

The algorithm for the minimum spatial distance between two Bernstein polynomials is shown in Algorithm 2. The first two inputs to the function are the sets of Bernstein coefficients,  $\mathcal{P} = \{\mathbf{P}_m\}$  and  $\mathcal{Q} = \{\mathbf{Q}_n\}$ , which define the two Bernstein polynomials in question. The last two inputs are the global upper bound on the minimum distance,  $\alpha$ , and a desired tolerance  $\epsilon$ .

The upper\_bound() function on line 1 finds the furthest distance between the end points of the two Bernstein polynomials, i.e.,  $\text{lower\_bound}(\mathcal{P}, \mathcal{Q}) = \max\{\mathcal{P}[0] - \mathcal{Q}[0], \mathcal{P}[0] - \mathcal{Q}[n], \mathcal{P}[m] - \mathcal{Q}[0], \mathcal{P}[m] - \mathcal{Q}[n]\}$  where  $m$  and  $n$  are the degrees of the polynomials represented by  $\mathcal{P}$  and  $\mathcal{Q}$ , respectively. This is a valid upper bound on the minimum distance between the two polynomials due to End Point Values (Property A2).

The lower\_bound() function on line 2 finds the lower bound on the distance between the two polynomials by using the GJK algorithm. This is a valid lower bound because of Property A1, Convex Hull. The next three *if* statements on lines 3, 6, and 9 are very similar to those seen in Algorithm 1. Line 3 updates the global upper bound  $\alpha$  if the current upper bound is smaller. Line 6 prunes the current iteration since it is impossible the current lower bound, *lower*, to be the minimum distance if it is larger than the global upper bound. Line 9 returns  $\alpha$  if the desired tolerance is met.

The lines within the *else* statement split the Bernstein polynomials defined by  $\mathcal{P}$  and  $\mathcal{Q}$  and recursively call Algorithm 2. Like in Algorithm 1, the first option for splitting would be to simply split at the halfway point. The second option for splitting the curves is outlined in [44] and uses the location at which the minimum distance occurs to choose the splitting point. Figure 19a illustrates the minimum distance between several different Bernstein polynomials. The code to generate this plot can be found in [40]. The execution time to compute the minimum spatial distance is 3.29 ms on a Lenovo ThinkPad laptop using an Intel Core i7-8550U, 1.80 GHz CPU.

**Algorithm 2:** Minimum Distance Between Two Bernstein Polynomials

---

**Input:**  $\mathcal{P}, \mathcal{Q}, \alpha, \epsilon$

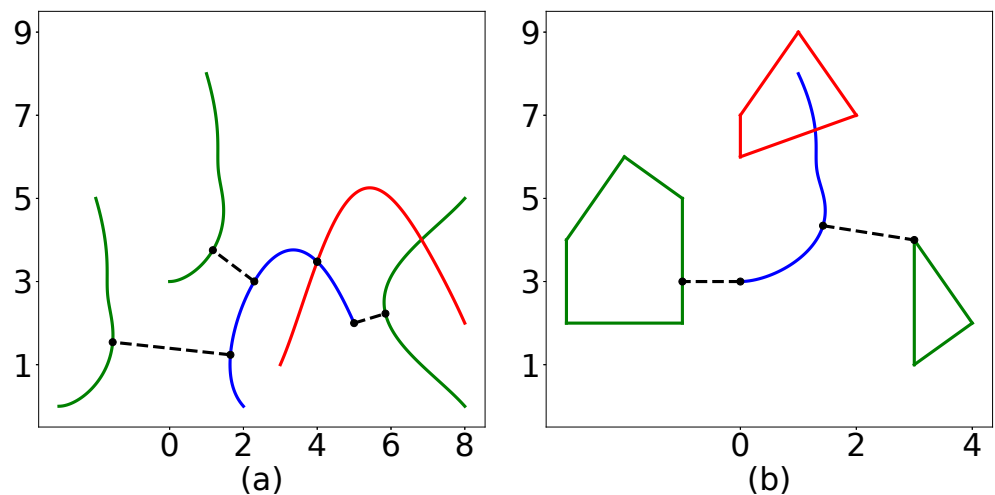
```

1   $upper = upper\_bound(\mathcal{P}, \mathcal{Q})$ 
2   $lower = lower\_bound(\mathcal{P}, \mathcal{Q})$ 
3  if  $upper < \alpha$  then
4     $\alpha = upper$ 
5  end
6  if  $\alpha < lower$  then
7    return  $\alpha$ 
8  end
9  if  $upper - lower < \epsilon$  then
10   return  $\alpha$ 
11 else
12    $\mathcal{P}^A, \mathcal{P}^B = \text{split}(\mathcal{P})$ 
13    $\mathcal{Q}^A, \mathcal{Q}^B = \text{split}(\mathcal{Q})$ 
14    $\alpha = \min(\alpha, \text{Algorithm 2}(\mathcal{P}^A, \mathcal{Q}^A, \alpha))$ 
15    $\alpha = \min(\alpha, \text{Algorithm 2}(\mathcal{P}^A, \mathcal{Q}^B, \alpha))$ 
16    $\alpha = \min(\alpha, \text{Algorithm 2}(\mathcal{P}^B, \mathcal{Q}^A, \alpha))$ 
17    $\alpha = \min(\alpha, \text{Algorithm 2}(\mathcal{P}^B, \mathcal{Q}^B, \alpha))$ 
18 end
19 return  $\alpha$ 

```

---

**Remark 1.** Note that Algorithm 2 can also be employed to compute the minimum distance between a Bernstein polynomial and a point or a convex shape. This is shown in Figure 19b.



**Figure 19.** (a) Minimum distance between curves. (b) Minimum distance between a curve and a polygon. All distances are measured to the blue curve. A red curve or polygon indicates that a collision exists.

#### 4.4. Collision Detection

In some cases it may be desirable to quickly check the feasibility of a trajectory rather than finding a minimum distance. The collision detection algorithm can be used in these cases. The two major differences between the Collision Detection Algorithm and the Minimum Distance Algorithm described previously are a modification of the GJK algorithm and a change in the stopping criteria. Rather than having the GJK algorithm return a minimum distance, it simply returns whether a collision has been detected (i.e., convex hulls intersecting). The stopping criteria is set to return the moment no collisions are found rather than continuing iterations to meet a desired tolerance. For example,

if the original convex hulls of two Bernstein polynomials do not intersect, the collision detection algorithm will return *no collision* after the first iteration while the minimum distance algorithm will continue to iterate until the desired tolerance is met. Therefore, this algorithm is computationally inexpensive compared to the minimum distance algorithm, with the drawback that it only returns a binary value (no collision or collision possible) rather than a minimum distance.

The collision detection algorithm is shown in Algorithm 3. The inputs are the coefficients of the Bernstein polynomials being compared,  $\mathcal{P}$  and  $\mathcal{Q}$ , and the maximum number of iterations  $max\_iter$ . The *while* loop beginning on line 2 runs until it is determined that a collision does not exist or until the maximum number of iterations is met. The `find_collisions()` function on line 3 uses the modified GJK algorithm to determine which convex hulls from the set  $\mathcal{P}$  collide with those from the set  $\mathcal{Q}$ . The *if* statement on line 4 checks to see whether collisions were found. If both  $\mathcal{P}_{col}$  and  $\mathcal{Q}_{col}$  are empty, then no collisions exist. If collisions do exist then the *for* loops starting on lines 7 and 11 split all the convex hulls that were found to collide and add them to the set to be checked. Note that the parent set that is split is removed from the set of convex hulls to check. If the maximum number of iterations is met, then the algorithm returns that a collision is possible. The execution time when a collision is possible is 1.10 ms on a Lenovo ThinkPad laptop using an Intel Core i7-8550U, 1.80 GHz CPU. However, when a collision does not exist, the execution time is only 7.25  $\mu$ s.

---

**Algorithm 3:** Collision Detection
 

---

**Input:**  $\mathcal{P}, \mathcal{Q}, max\_iter$

```

1  $k = 0$ 
2 while  $k < max\_iter$  do
3    $\mathcal{P}_{col}, \mathcal{Q}_{col} = \text{find\_collisions}(\mathcal{P}, \mathcal{Q})$ 
4   if  $\mathcal{P}_{col} \cup \mathcal{Q}_{col} = \{\}$  then
5     return No Collision
6   end
7   for  $\mathcal{P}_i \in \mathcal{P}_{col}$  do
8      $\mathcal{P}^A, \mathcal{P}^B = \text{split}(\mathcal{P}_i)$ 
9      $\mathcal{P} = \mathcal{P} \cup \{\mathcal{P}^A, \mathcal{P}^B\} \setminus \mathcal{P}_i$ 
10  end
11  for  $\mathcal{Q}_i \in \mathcal{Q}_{col}$  do
12     $\mathcal{Q}^A, \mathcal{Q}^B = \text{split}(\mathcal{Q}_i)$ 
13     $\mathcal{Q} = \mathcal{Q} \cup \{\mathcal{Q}^A, \mathcal{Q}^B\} \setminus \mathcal{Q}_i$ 
14  end
15   $k++$ 
16 end
17 return Collision Possible
  
```

---

#### 4.5. Penetration Algorithm

If two convex shapes intersect, in order to derive information such as the penetration depth and vector, the EPA [46] can be used. A slight modification of the EPA algorithm is proposed here, which is referred to as the DEPA, whose objective is to find the penetration of one convex shape relative to another along a specific direction  $\vec{d}$ . The top left plot of Figure 20 shows two shapes intersecting each other, and the remaining plots show examples of penetration vectors, i.e., the vector  $\vec{d}$  needed to move the second shape so that it no longer intersects the first shape. The DEPA algorithm finds the shortest possible penetration vector. The pseudocode is reported below (see Algorithm 4).

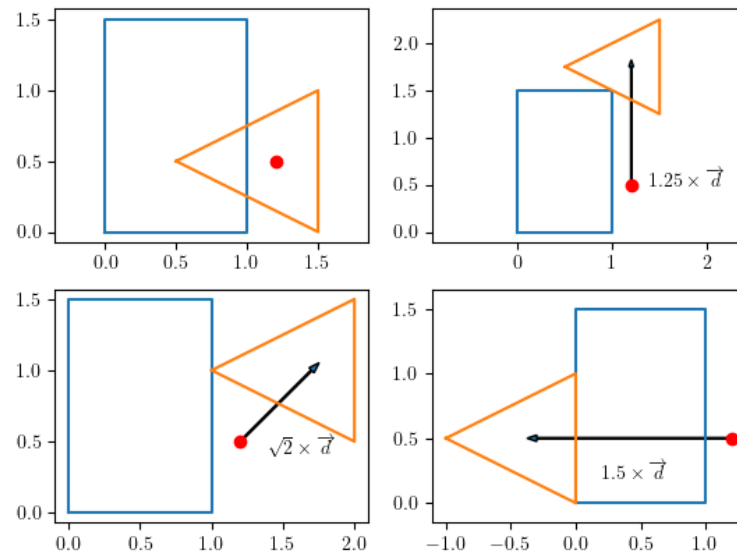


Figure 20. Illustration of penetration.

---

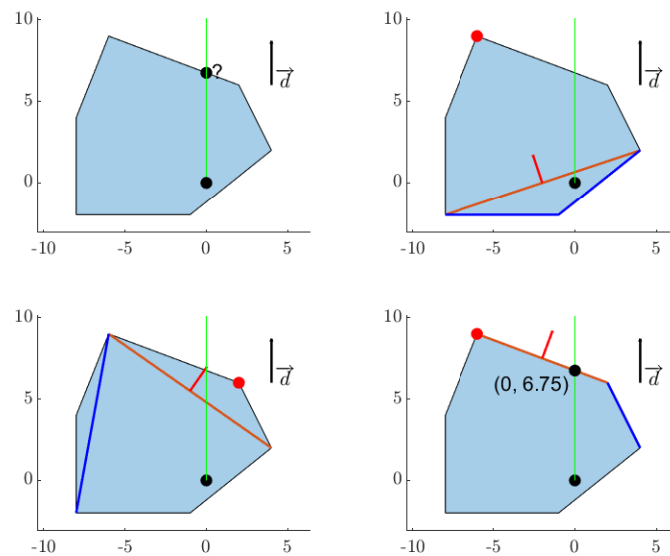
**Algorithm 4:** Directed Extended Polytope Algorithm

---

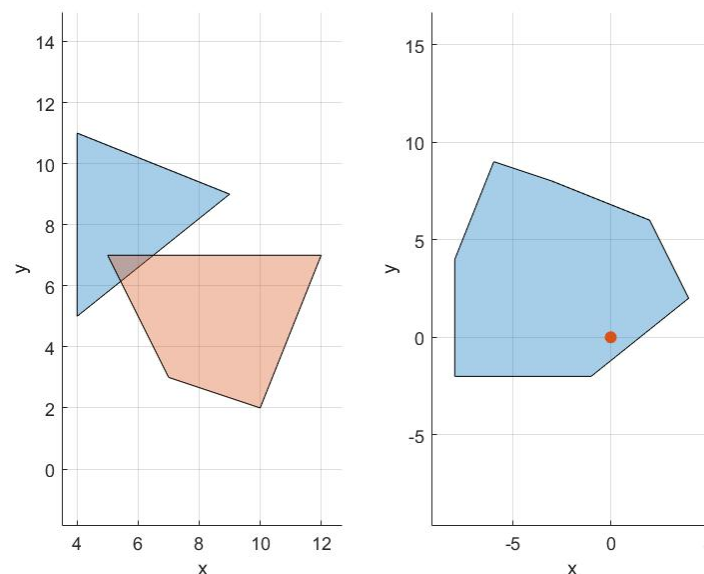
**Data:**  $\vec{d}$   
**Data:** *MinkowskiDifference*  
**Data:** *simplex*  
1 ContainsOrigin(*simplex*);  
2  $A, B \leftarrow \text{IntersectingEdge}(\text{simplex}, \text{Ray}(\vec{0}, \vec{d}));$   
3 **Loop**  
4    $\vec{n} \leftarrow \text{TripleProduct}(\vec{AB}, \vec{A}, \vec{AB});$   
5    $C \leftarrow \text{Support}(\text{MinkowskiDifference}, \vec{n});$   
6   **if** Parallel( $\vec{C}, \vec{d}$ ) **then**  
7      $\vec{vec} \leftarrow \vec{C};$   
8     **return**;  
9   **else if** Equals( $C, B$ ) OR Equals( $C, A$ ) **then**  
10     **break**;  
11   **else if** Intersection( $\text{Ray}(0, \vec{d}), \overline{CA}$ ) **then**  
12      $B \leftarrow C;$   
13   **else if** Intersection( $\text{Ray}(0, \vec{d}), \overline{CB}$ ) **then**  
14      $A \leftarrow C;$   
15    $\vec{vec} \leftarrow \text{Intersection}(\text{Ray}(0, \vec{d}), \overline{AB});$   
16 **EndLoop**

---

Figure 21 demonstrates the algorithm in 4 steps. The first plot shows the Minkowski Difference of the shapes depicted in Figure 22, which contains the origin and with a triangle simplex that contains the origin. This is the desired direction to move a polytope A (blue polytope) of Figure 22 such that it no longer contains polytope B (beige polytope). Once the norm of the point along the edge of the Minkowski Difference parallel to  $\vec{d}$  is found, A can then move in the same direction with the same length to no longer intersect B.



**Figure 21.** Iteration of the DEPA algorithm.



**Figure 22.** Two intersecting polygons and resulting Minkowski Difference.

## 5. Numerical Examples

In this section, numerical examples using the BeBOT toolkit and Python's Scipy Optimization package are examined (flight tests are available at [47]). The implementation of the following examples can be found in [40].

### 5.1. Dubins Car—Time Optimal

In this simple example, several trajectories for a vehicle with Dubins car dynamics are generated to illustrate the properties of Bernstein polynomials. We let the desired trajectory assigned to the vehicle be given by the Bernstein polynomial

$$\begin{bmatrix} C_n^{[x]}(t) \\ C_n^{[y]}(t) \end{bmatrix} = \mathbf{C}_n(t) = \sum_{i=0}^n \mathbf{P}_{i,n} B_{i,n}(t), \quad t \in [t_0, t_f]. \quad (11)$$



The square of the speed of the vehicle is a 1D Bernstein polynomial given by

$$v^2(t) = \|\dot{\mathbf{C}}_n(t)\|^2.$$

The heading angle is

$$\psi(t) = \tan^{-1} \frac{\dot{C}_n^{[y]}(t)}{\dot{C}_n^{[x]}(t)}, \quad (12)$$

and the angular rate is a 1D rational Bernstein polynomial given by

$$\omega(t) = \frac{\ddot{C}_n^{[y]}(t)\dot{C}_n^{[x]}(t) - \dot{C}_n^{[y]}(t)\ddot{C}_n^{[x]}(t)}{\|\dot{\mathbf{C}}_n(t)\|^2}. \quad (13)$$

The objective at hand is to find a trajectory that arrives at a desired destination in the minimum possible time while adhering to feasibility and safety constraints. In particular, the trajectory generation problem is as follows:

$$\min_{\mathbf{P}_n, t_f} t_f$$

subject to

$$\begin{aligned} \mathbf{C}_n(t_0) &= \mathbf{C}_0, \quad \mathbf{C}_n(t_f) = \mathbf{C}_f, \\ \psi(t_0) &= \psi_0, \quad \psi(t_f) = \psi_f, \\ \|\dot{\mathbf{C}}_n(t_0)\| &= v_0, \quad \|\dot{\mathbf{C}}_n(t_f)\| = v_f, \\ \|\dot{\mathbf{C}}_n(t)\|^2 &\leq v_{\max}^2, \quad \forall t \in [t_0, t_f], \\ \|\dot{\psi}(t)\| &\leq \omega_{\max}, \quad \forall t \in [t_0, t_f], \\ \|\mathbf{C}_n(t) - \mathbf{O}_i\|^2 &\geq d_s^2, \quad \forall t \in [t_0, t_f], \quad i = 1, 2. \end{aligned}$$

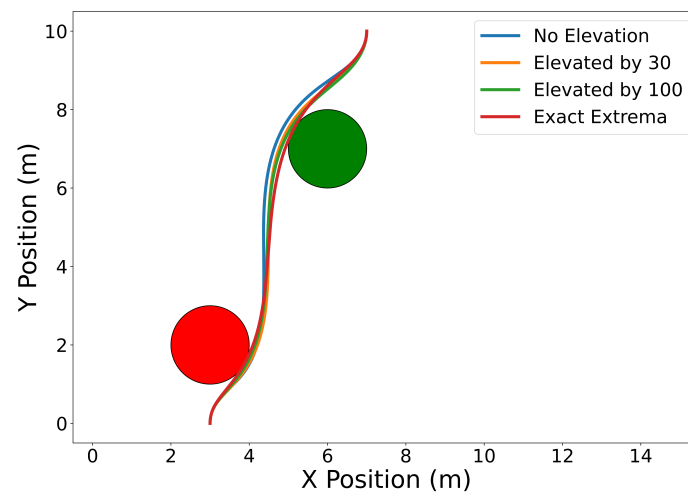
We set the initial and final position, heading, and speed to  $\mathbf{C}_0 = [3, 0]^\top$  m,  $\mathbf{C}_f = [7, 10]^\top$  m,  $\psi_0 = \psi_f = \frac{\pi}{2}$  rad, and  $v_0 = v_f = 1 \frac{\text{m}}{\text{s}}$ . The maximum speed, maximum angular rate, and minimum safe distance constraints are  $v_{\max} = 5 \frac{\text{m}}{\text{s}}$ ,  $\omega_{\max} = 1 \frac{\text{rad}}{\text{s}}$ , and  $d_s^2 = 1$  m, respectively. The positions of the obstacles are  $\mathbf{O}_1 = [3, 2]^\top$  m and  $\mathbf{O}_2 = [6, 7]^\top$  m.

In the problem above, the initial and final constraints for position, heading, and speed are enforced using the End Point Values property (Property A2) together with Equations (A2), (12) and (13). Similarly, the same property is used to enforce the initial and final speeds and headings (see (A2)). Note that the norm squared of the speed and of the distance between the trajectory and the obstacles can be expressed as Bernstein polynomials (the sum, the difference, and the product between Bernstein polynomials are also Bernstein polynomials). A similar argument can be made for the norm square of the angular rate, which can be expressed as a rational Bernstein polynomial (see Property A7). Thus, the maximum speed and angular rate, and collision avoidance constraints can be enforced using the Evaluating Bounds or Evaluating Extrema procedures described in Sections 4.1 and 4.2.

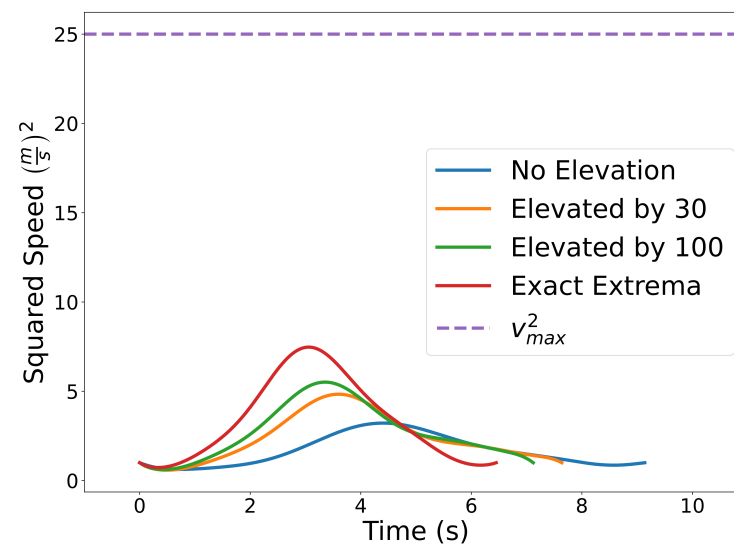
Figure 23 shows the results with  $n = 10$ . The blue curve is obtained by enforcing the constraints using the Evaluating Bounds procedure. The optimal time of arrival at the final destination is  $t_f = 9.14$  s. Next, we solve the same problem by enforcing the constraints using the Evaluating Bound procedure together with Degree Elevation. Recall that by degree elevating a Bernstein polynomial, the Bernstein coefficients converge towards the curve. Thus, degree elevation can be used to enforce constraints with tighter bounds. The orange and green lines show the trajectories obtained using degree elevations of 30 and 100, respectively. Degree elevation to degree 30 results in an optimal final time  $t_f = 7.64$  s. The elevation to degree 100 provides an optimal value  $t_f = 7.12$  s. Finally, the trajectory with smallest optimal final time,  $t_f = 6.45$  s, depicted as the red curve in

Figure 23, is obtained by enforcing the constraints using the Evaluating Extrema algorithm (Section 4.2). While higher degree elevations or evaluating the exact extrema can produce more optimal trajectories, that optimality comes at the cost of additional computation time. Using a Lenovo Thinkpad P52s with an Intel Core i7-8550U CPU with a 1.8 GHz clock and 8 GB of memory, the computation time required for no degree elevation, a degree elevation of 30, a degree elevation of 100, and the exact extrema algorithm was 0.105 s, 0.146 s, 0.201 s, and 0.573 s, respectively.

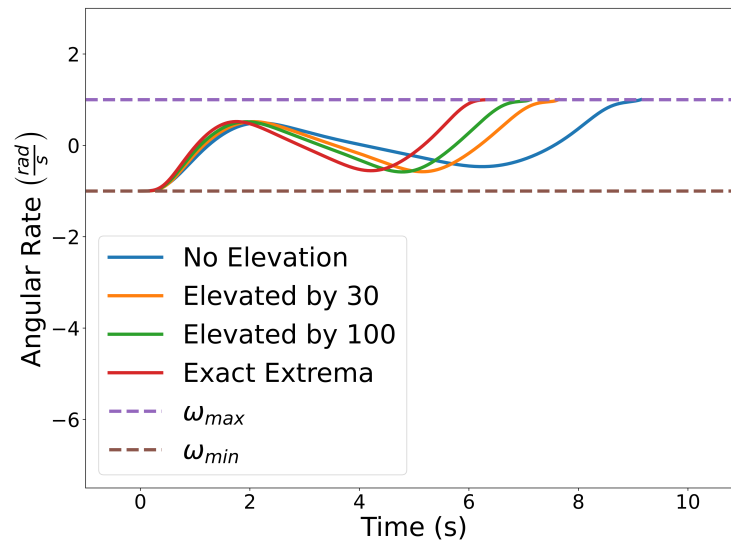
Figure 24 illustrates the squared speed of each example. Figure 25 shows the angular rate of each trial. It can be seen that the vehicle correctly adheres to the speed and angular rate constraints for each trial with the only differences being the final time and proximity to the obstacles.



**Figure 23.** Time optimal trajectory for vehicle with initial and final speeds and headings, maximum speed, maximum angular rate, and maximum safe distance constraints ranging from least to most conservative distance estimates.



**Figure 24.** Plot of the squared speed constraints for each separate trial.



**Figure 25.** Plot of the angular rate constraints for each separate trial.

**Remark 2.** The Exact Extrema function is a complex non-linear and non-smooth function. When it is used to enforce constraints, gradient-based optimization solvers such as the one used in this work can fail to converge to a feasible solution, especially if the initial guess is not feasible. One option is to use an iterative procedure where (i) a feasible sub-optimal solution is obtained by enforcing the collision avoidance constraint using the Evaluating Bounds function, and (ii) this solution is then used as an initial guess to solve the (more accurate) problem with the Exact Extrema constraint.

## 5.2. Air Traffic Control—Time Optimal

In this example, we consider the problem of routing several commercial flights between major US cities in two dimensions (i.e., constant altitude). Assuming that each flight departs at the same time, the goal is to minimize the combined flight time of all the vehicles. Let the position, speed, heading, and angular rate of each vehicle under consideration be parameterized as in Section 5.1. We shall also make the assumption that the trajectories are on a 2D plane rather than on the surface of the Earth.

The goal is to compute cumulatively time optimal trajectories subject to maximum speed and angular velocity bounds, initial and final position, angle, and speeds. The vehicles must also maintain a minimum safe distance between each other. This problem can be formulated as follows:

$$\min_{\mathbf{P}_n, t_f} \sum_{k=1}^m t_f^{[k]}$$

subject to

$$\begin{aligned} \mathbf{C}_n^{[k]}(0) &= \mathbf{C}_0^{[k]}, \quad \mathbf{C}_n^{[k]}(t_f^{[k]}) = \mathbf{C}_f^{[k]}, \\ \psi^{[k]}(0) &= \psi_0^{[k]}, \quad \psi^{[k]}(t_f^{[k]}) = \psi_f^{[k]}, \\ \|\dot{\mathbf{C}}_n^{[k]}(0)\| &= v_0^{[k]}, \quad \|\dot{\mathbf{C}}_n^{[k]}(t_f^{[k]})\| = v_f^{[k]}, \\ v_{\min}^2 &\leq \|\dot{\mathbf{C}}_n^{[k]}(t)\|^2 \leq v_{\max}^2, \quad \forall t \in [0, t_f^{[k]}], \\ |\dot{\psi}^{[k]}(t)| &\leq \omega_{\max}, \quad \forall t \in [0, t_f^{[k]}], \\ \|\mathbf{C}_n^i(t) - \mathbf{C}_n^j(t)\|^2 &\geq d_s^2, \quad \forall i, j \in \{1, \dots, m\}, i \neq j. \end{aligned}$$

where the superscript  $[k]$  corresponds to the  $k$ th vehicle out of  $m$  vehicles,  $\mathbf{C}_0^{[k]}$  and  $\mathbf{C}_f^{[k]}$  are the initial and final positions,  $\psi_0^{[k]}$  and  $\psi_f^{[k]}$  are the initial and final headings,  $v_0^{[k]}$  and  $v_f^{[k]}$  are the initial and final speeds,  $v_{\min}$  and  $v_{\max}$  are the minimum and maximum speeds,  $\omega_{\max}$  is the maximum angular velocity,  $d_s$  is the minimum safe distance, and  $t_f^{[k]}$  is the final time of the  $k$ th vehicle.

The departure cities, in vehicle order, are: San Diego, New York, Minneapolis, and Seattle. The arrival cities, in vehicle order, are: Minneapolis, Seattle, Miami, and Denver. The initial and final speeds are all  $v_0^{[k]} = v_f^{[k]} = 205 \frac{\text{m}}{\text{s}} \forall k \in \{1, \dots, m\}$ , the initial headings are  $\psi_0 = [0, \pi, 0, 0]^\top$  rad, the final headings are  $\psi_f = [0, \pi, -\frac{\pi}{2}, 0]$  rad, the minimum speed is  $v_{\min} = 200 \frac{\text{m}}{\text{s}}$ , the maximum speed is  $v_{\max} = 260 \frac{\text{m}}{\text{s}}$ , the maximum angular velocity is  $\omega_{\max} = 3 \frac{\text{deg}}{\text{s}} = 0.0524 \frac{\text{rad}}{\text{s}}$ , the minimum safe distance is  $d_s = 5$  km, and the degree of the Bernstein polynomials being used is 5.

The initial and final position constraints are enforced using the End Point Values property (Property A2). Similarly, the same property is used to enforce the initial and final speeds and headings (see (A2)). Note that the norm square of the speed and the norm square of the distance between vehicles can be expressed as 1D Bernstein polynomials (the sum, difference, and product between Bernstein polynomials are also Bernstein polynomials). A similar argument can be made for the norm square of the angular rate, which can be expressed as a rational Bernstein polynomial (see Property A7). Thus, the maximum speed and angular rate, and collision avoidance constraints can be enforced using the Evaluating Bounds or Evaluating Extrema procedures described in Sections 4.1 and 4.2.

The optimized flight plans can be seen in Figure 26. The squared speed of each vehicle is shown in Figure 27. Note that each vehicle begins and ends with the same speed. The vehicles never slow down less than their initial speeds which means they never reach the minimum speed constrain, nor do the vehicles go faster than the maximum speed. In Figure 28, the angular velocity of each vehicle is shown. The minimum and maximum angular rate constraints are shown by the dotted lines. The vehicles' angular rates never approach the minimum or maximum angular rate constraints due to the large area being covered. Finally, the squared euclidean distance between vehicles is shown in Figure 29. As expected, the squared Euclidean distance between two vehicles never falls below the minimum safe distance. Note that curves within the constraint plots end at different times. This is expected since each vehicle has a different final time. The furthest time reached in Figure 29 is less than that of the other plots because the other vehicles have already reached their final time before the longest flight reaches its final time.

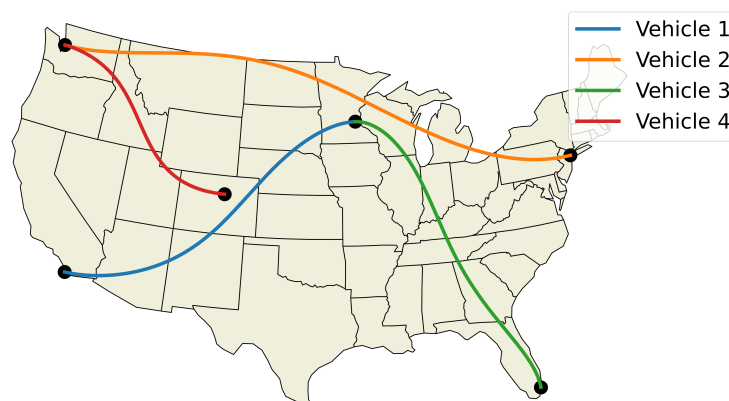
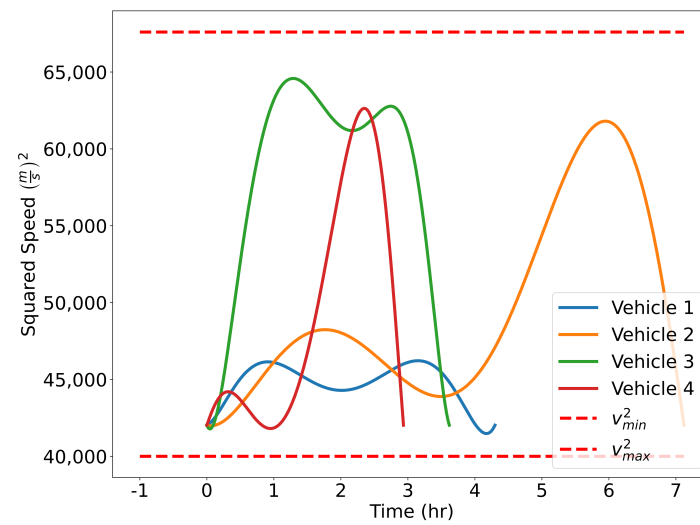
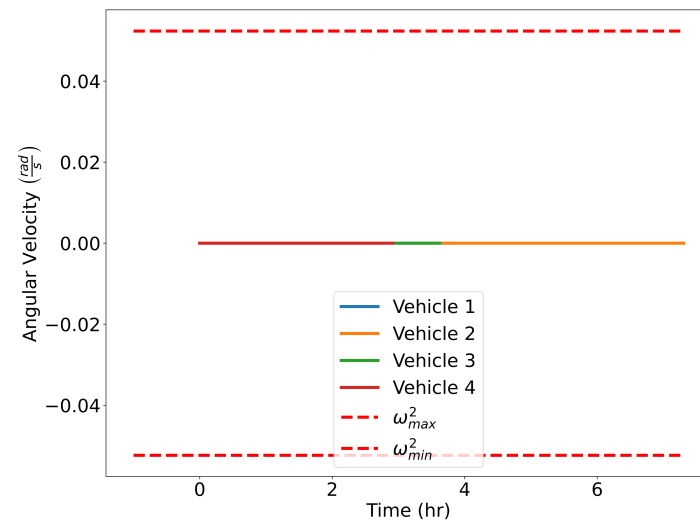


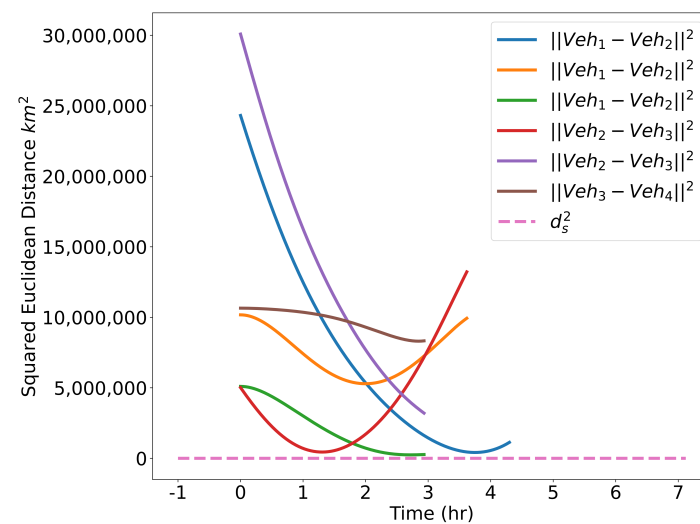
Figure 26. Commercial flight trajectories between major US cities.



**Figure 27.** Verifying speed constraints for the Air Traffic Control example.



**Figure 28.** Verifying angular rate constraints for the Air Traffic Control example.



**Figure 29.** Verifying minimum safe distance constraints for the Air Traffic Control example.



### 5.3. Cluttered Environment

In many real world scenarios, robots must safely traverse cluttered environments. In this example, three aerial vehicles traveling at a constant altitude must navigate around several obstacles while also adhering to dynamic and minimum safe distance constraints. Let the position, speed, heading angle, and angular rate of each vehicle be defined as in Section 5.1. The goal of this example is to compute trajectories whose arc length is minimized subject to maximum speed constraints along with initial and final positions, heading angles, and speeds. The vehicles should also adhere to a minimum safe distance between each other and between obstacles. We formulate the problem as follows:

$$\min_{\mathbf{P}_n} \sum_{i=1}^m \sum_{k=0}^{n-1} \|\mathbf{P}_{k+1,n}^{[i]} - \mathbf{P}_{k,n}^{[i]}\| \quad (14)$$

subject to

$$\begin{aligned} \mathbf{C}_n^{[k]}(0) &= \mathbf{C}_0^{[k]}, \quad \mathbf{C}_n^{[k]}(t_f) = \mathbf{C}_f^{[k]}, \\ \psi^{[k]}(0) &= \psi_0^{[k]}, \quad \psi^{[k]}(t_f) = \psi_f^{[k]}, \\ \|\dot{\mathbf{C}}_n^{[k]}(0)\| &= v_0^{[k]}, \quad \|\dot{\mathbf{C}}_n^{[k]}(t_f)\| = v_f^{[k]}, \\ \|\dot{\mathbf{C}}_n^{[k]}(t)\|^2 &\leq v_{\max}^2, \quad \forall t \in [0, t_f], \\ \|\mathbf{C}_n^{[i]}(t) - \mathbf{C}_n^{[j]}(t)\|^2 &\geq d_s^2, \quad \forall i, j \in \{1, \dots, m\}, i \neq j, \\ \|\mathbf{C}_n^{[i]}(t) - \mathbf{O}_j\|^2 &\geq d_{obs}^2, \quad \forall t \in [0, t_f], \quad i \in \{1, \dots, m\}, \\ &\quad j \in \{1, \dots, b\}, \end{aligned}$$

where  $\mathbf{O}_j$  is the position of the  $j$ th obstacle out of  $b$  obstacles.

The initial positions for each vehicle, in order, are  $[0, 0]^\top$  m,  $[10, 0]^\top$  m, and  $[20, 0]^\top$  m. The initial speeds are all  $1 \frac{\text{m}}{\text{s}}$  and the initial heading angles are all  $\frac{\pi}{2}$  rad. The final positions for each vehicle are, in order,  $[20, 30]^\top$  m,  $[0, 30]^\top$  m, and  $[10, 30]^\top$  m. The final speeds and final heading angles are the same as the initial speeds and heading angles. The order of the Bernstein polynomials being used is 7, the final time is  $t_f = 30$  s, the minimum safe distance between vehicles is  $d_s = 1$  m, the minimum safe distance between vehicles and obstacles is  $d_{obs} = 2$  m, and the maximum speed is  $v_{\max} = 10 \frac{\text{m}}{\text{s}}$ . The vehicles traversing the cluttered environment can be seen in Figure 30. This experiment has been repeated in the Cooperative Autonomous Systems (CAS) lab using three AR Drones 2.0. The flight tests can be viewed at [47].

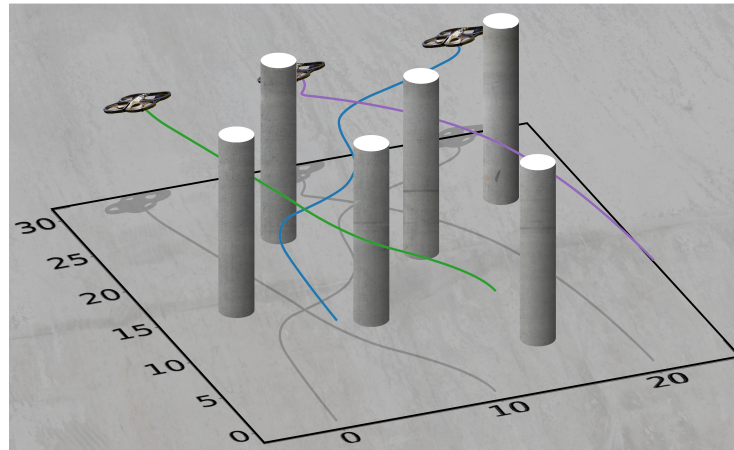


Figure 30. Aerial vehicles navigating a cluttered environment.

#### 5.4. Vehicle Overtake

Here we consider an autonomous driving example in which one vehicle attempts to overtake another vehicle while driving around a  $90^\circ$  corner. The corner is defined by two arcs with a center point located at  $[140, 0]^\top$  m. The inner track has a radius of  $r_{inner} = 125$  m and the outer track has a radius of  $r_{outer} = 140$  m. To clearly distinguish the vehicle being overtaken, it will be referred to as the opponent.

For simplicity, we consider the objective of minimizing the arc-length of the trajectory, which can be done by minimizing the sum of the squared Euclidean norm of consecutive control points, i.e.,

$$E(\mathbf{P}_n) = \sum_{i=1}^n \|\mathbf{P}_{i,n} - \mathbf{P}_{i-1,n}\|^2. \quad (15)$$

The desired endpoint of the vehicle is at the end of the corner. This is computed by measuring the angle between the vehicle's position and the end of the curve,

$$A(\mathbf{P}_n) = \left( \arctan 2 \left( \mathbf{P}_{n,n}^{[y]} - \mathbf{q}^{[y]}, \mathbf{P}_{n,n}^{[x]} - \mathbf{q}^{[x]} \right) - \frac{\pi}{2} \right)^2, \quad (16)$$

where the function  $\arctan 2$  returns an angle in the correct quadrant [48]. Given Equations (15) and (16), we formulate the problem as

$$\min_{\mathbf{P}_n} ((1 - \alpha)E(\mathbf{P}_n) + \alpha A(\mathbf{P}_n))\beta \quad (17)$$

subject to

$$\begin{aligned} \mathbf{C}_n(0) &= \mathbf{C}_0, \quad \psi(0) = \psi_0, \quad \|\dot{\mathbf{C}}_n(0)\| = v_0, \\ \|\dot{\mathbf{C}}_n(t)\|^2 &\leq v_{\max}^2, \quad \forall t \in [0, t_f], \\ |\dot{\psi}(t)| &\leq \omega_{\max}, \quad \forall t \in [0, t_f], \\ r_{inner}^2 &\leq \|\mathbf{C}_n(t) - \mathbf{q}\|^2 \leq r_{outer}^2, \\ \|\mathbf{C}_n(t) - \mathbf{O}_n(t)\|^2 &\geq d_s^2, \end{aligned}$$

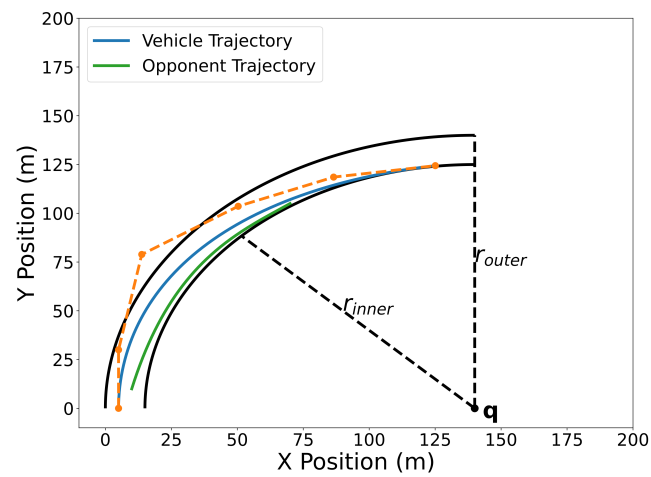
where  $\alpha$  and  $\beta$  are tuning parameters.  $\mathbf{C}_0$ ,  $\psi_0$ , and  $v_0$  are the initial position, heading, and speed of the vehicle, respectively.  $v_{\max}$  and  $\omega_{\max}$  are the maximum speed and angular rate, respectively. The predicted trajectory of the opponent is represented as the Bernstein polynomial  $\mathbf{O}_n(t)$  and the minimum safe distance to the opponent is  $d_s$ . Using a sensor such as a camera or LiDAR, one could measure the state of the opponent and then predict its future position using a method such as the one presented in [49].

At time  $t = t_0$ , when planning occurs, the position of the vehicle is  $[5, 0]^\top$  m, its speed is  $50 \frac{\text{m}}{\text{s}}$ , and its initial heading angle is  $\frac{\pi}{2}$  rad. The control points of the Bernstein polynomial representing the opponent's trajectory are

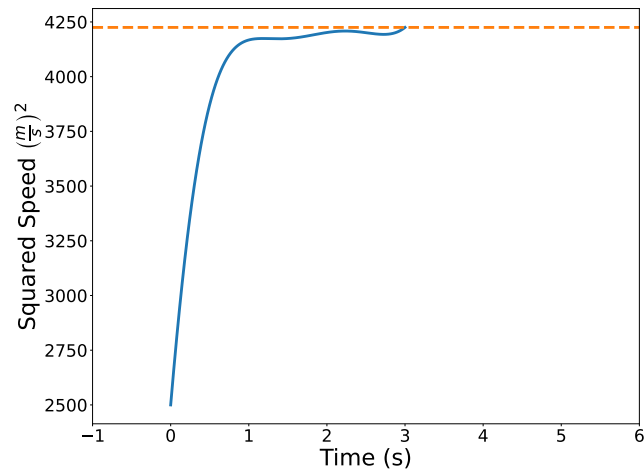
$$\begin{bmatrix} 10 & 25 & 50 & 70 \\ 10 & 75 & 90 & 105 \end{bmatrix} \text{m}.$$

The maximum speed is  $65 \frac{\text{m}}{\text{s}}$ , the maximum angular rate is  $\frac{\pi}{5} \frac{\text{rad}}{\text{s}}$ , and the minimum safe distance is 3 m. The tuning parameters are  $\alpha = 1 - 10^{-6}$  and  $\beta = 100$ .

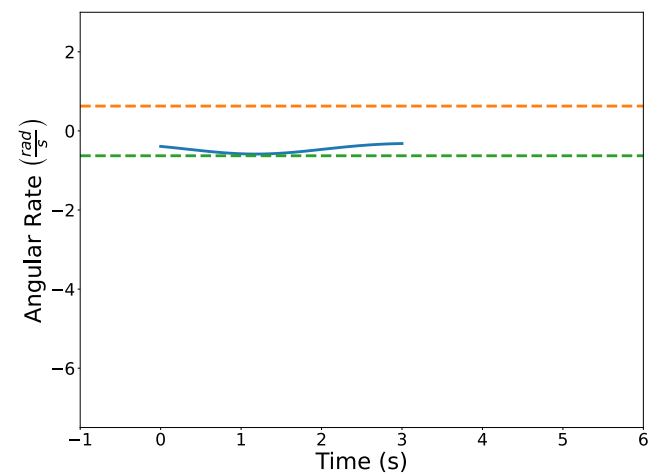
Figure 31 illustrates the optimized vehicle trajectory overtaking the opponent's trajectory. Figure 32 shows the squared speed of the vehicle along with the maximum speed constraint. As expected, the vehicle's speed approaches the maximum speed in order to successfully overtake the opponent. Figure 33 shows the vehicle's angular rate and its upper and lower constraints. It is clear that the vehicle remains within the desired bounds. Figure 34 shows the squared distance between the vehicle and the opponent. While the vehicle does come close to the opponent, it is never closer than the minimum safe distance.



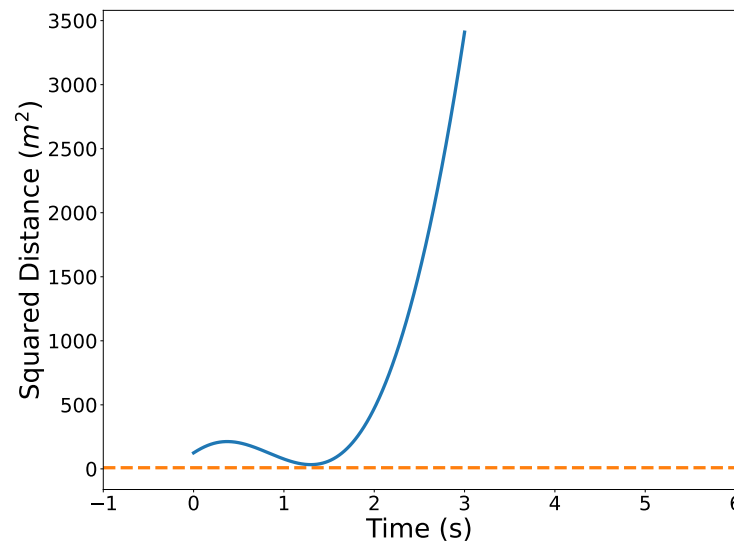
**Figure 31.** Overtaking trajectory. The track follows a quarter circle whose center point is at  $q$ . The inner track line has a radius of  $r_{inner}$  and the outer track line has a radius of  $r_{outer}$ .



**Figure 32.** Squared speed profile of the vehicle.



**Figure 33.** Angular rate profile of the vehicle.



**Figure 34.** Squared distance between vehicle and opponent.

### 5.5. Swarming

This section examines two methods for generating trajectories for large groups of autonomous aerial vehicles. The centralized method optimizes every trajectory at once. On the other hand, the decentralized method generates trajectories one at a time and compares them to previously generated trajectories.

The position of each vehicle in a swarm of  $m$  vehicles for the following examples is parameterized as a 3D Bernstein polynomial, i.e.,

$$\sum_{i=0}^n \mathbf{P}_{i,n}^{[j]} B_{i,n}(t) = \mathbf{C}_n^{[j]}(t), \quad \forall j \in \{1, \dots, m\}, \quad \mathbf{P}_n^{[j]} \in \mathbb{R}^{3 \times n}.$$

#### 5.5.1. 101 Vehicle—Centralized

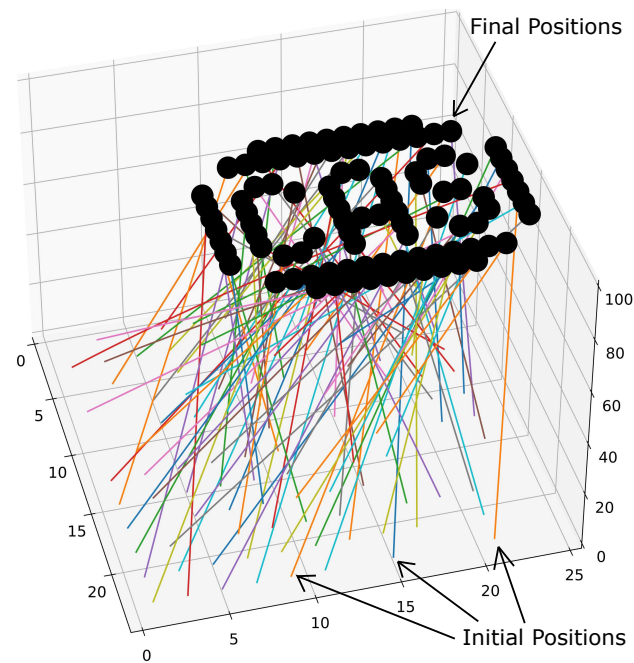
The centralized method optimizes the trajectories for each vehicle simultaneously. The goal is to minimize the arc length of each trajectory. There are  $m$  vehicles with 3rd order Bernstein polynomials representing their trajectories which are constrained to a minimum safe distance between each other and initial and final positions. This is formulated as follows:

$$\min_{\mathbf{P}_n} \sum_{i=1}^m \sum_{k=0}^{n-1} \|\mathbf{P}_{k+1}^{[i]} - \mathbf{P}_k^{[i]}\|,$$

subject to

$$\begin{aligned} \mathbf{C}_n^{[i]}(0) &= \mathbf{C}_0^i, \quad \mathbf{C}_n^{[i]}(t_f) = \mathbf{C}_f^i, \quad \forall i \in \{1, \dots, m\}, \\ \|\mathbf{C}_n^{[i]}(t) - \mathbf{C}_n^{[j]}(t)\|^2 &\geq d_s^2, \quad \forall i, j \in \{1, \dots, m\}, i \neq j. \end{aligned}$$

The initial positions for each vehicle were chosen randomly from a  $25 \text{ m} \times 25 \text{ m}$  grid at an altitude of  $z = 0 \text{ m}$ . The final positions were chosen to spell out “CAS”, as seen in Figure 35, at an altitude of  $z = 100 \text{ m}$ . In the next section we significantly reduce the number of dimensions in the optimization vector by using the decentralized approach.



**Figure 35.** 101 vehicles spelling out CAS using the centralized method.

### 5.5.2. 101 Vehicle—Decentralized

The decentralized method iteratively computes trajectories for the  $i$ th vehicle. Each new iteration is compared to the previously computed trajectories so that the minimum safety distance constraint is met. The problem that is solved at each iteration is written as

$$\min_{\mathbf{P}_n^{[i]}} \sum_{i=1}^m \sum_{k=0}^{n-1} \|\mathbf{P}_{k+1}^{[i]} - \mathbf{P}_k^{[i]}\|$$

subject to

$$\begin{aligned} \mathbf{C}_n^{[i]}(0) &= \mathbf{C}_0^{[i]}, \quad \mathbf{C}_n^{[i]}(t_f) = \mathbf{C}_f^{[i]}, \\ \|\mathbf{C}_n^{[i]}(t) - \mathbf{C}_n^{[j]}(t)\|^2 &\geq D_s^2, \quad \forall j \in \{1, \dots, i-1\}, \quad i > 1. \end{aligned}$$

Note that the first vehicle does not need to satisfy the minimum safe distance constraint since no trajectories have been computed before it.

The parameters used in this example were identical to that of the previous subsection. The resulting figure has been omitted due to its similarity to Figure 35.

### 5.5.3. 1000 Vehicle—Decentralized

The decentralized method can be used to compute 1000 trajectories. In this example, it is employed to generate the paths seen in Figure 36 to display the University of Iowa Hawkeye logo. The initial points are equally dispersed at an altitude of  $z = 0$  m on a  $100 \text{ m} \times 100 \text{ m}$  grid. The final points are the pattern shown at an altitude of  $z = 100$ . The cost function aims to maximize the temporal distance between the current  $i$ th trajectory and the previously generated  $j$ th trajectories by taking the reciprocal of the sum of the Bernstein coefficients of the norm squared difference, i.e.

$$\min_{\mathbf{P}_n^{[i]}} \frac{1}{\sum_{j=1}^{i-1} \mathbf{P}^{[norm,j]}}, \quad i > 1,$$

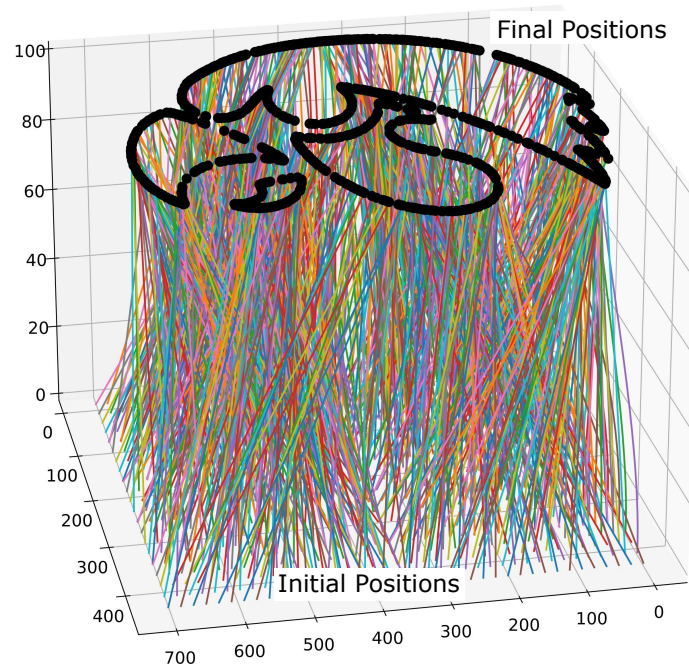
subject to

$$\mathbf{C}_n^{[i]}(0) = \mathbf{C}_0^{[i]}, \quad \mathbf{C}_n^{[i]}(t_f) = \mathbf{C}_f^{[i]},$$

where  $\mathbf{P}^{[norm,j]}$  are the Bernstein coefficients of the Bernstein polynomial representing the squared temporal distance between the  $i$ th and  $j$ th trajectories, i.e.,

$$\|\mathbf{C}^{[i]}(t) - \mathbf{C}^{[j]}(t)\|^2 = \sum_{i=0}^n P^{[norm,j]} B_{i,n}(t).$$

It should be noted that this formulation of cost function and constraints is used as a proof of concept. For other possible cost function and constraint formulations, the reader is referred to [50,51].



**Figure 36.** Trajectories for 1000 aerial vehicles with initial and final position and minimum safety distance constraints.

### 5.6. Marine Vehicle Model

In this example, we consider a marine vehicle model known as the medusa. The equations of motion of the medusa are as follows

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

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

where  $x$  and  $y$  represent the vehicle's position,  $\psi$  is the orientation,  $u$  (surge) and  $v$  (sway) are the linear velocities,  $r$  is the turning rate, and  $\tau = [\tau_u, \tau_r]^T$  is the vector of forces and torques due to thrusters/surfaces (control input).

In this example, we let the state,  $[x, y, \psi, u, v, r]^T$ , and input,  $[\tau_u, \tau_r]^T$ , be approximated by Bernstein polynomials, and impose the vehicle's dynamics directly through Bernstein

polynomial differentiation. Using Property A3, the dynamics constraints given by Equations (18) and (19) reduce to a set of algebraic constraints. Additional constraints imposed on this problem include collision avoidance and input saturation constraints. Figure 37 shows an example of motion planning for a medusa vehicle, which is required to reach a final destination in the minimum time. Ten markers are plotted along the trajectory (shown in blue) to represent the heading of the vehicle at that point in time. It can easily be seen that the vehicle's trajectory avoids the (inflated) unsafe region illustrated by the orange circle.

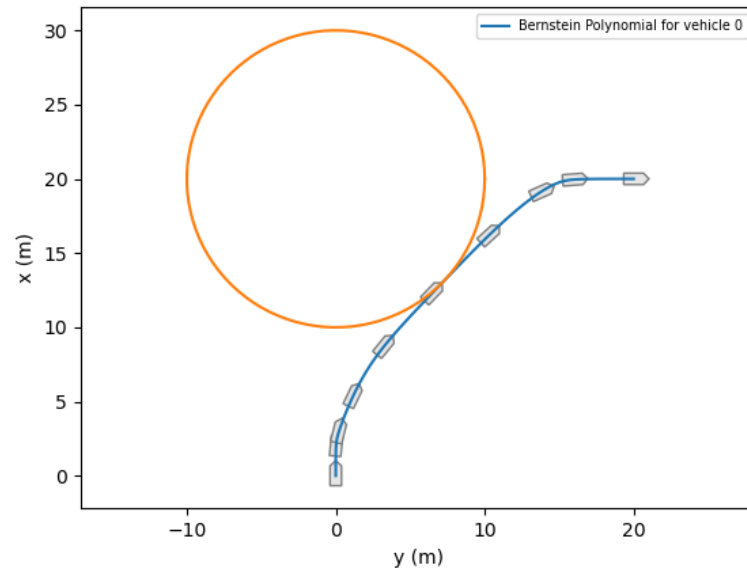


Figure 37. Medusa Example.

### 5.7. Dynamic Routing Problem

#### 5.7.1. Single Vehicle Case

We consider a problem where a single vehicle is supposed to visit  $M$  neighborhoods  $B_i = \{x \in \mathbb{R}^2 : \|x - b_i\| \leq r\}$  in minimum time  $t_f$ . Here  $r > 0$  and the vectors  $b_i \in \mathbb{R}^2$ ,  $i \in [1, M]$  represent a sequence of points of interest that has been generated by a Traveling Salesman Problem (TSP) algorithm. Let  $t_i$  denote a time instance when the vehicle's position satisfies  $C_n(t_i) \in B_i$ . Then, the dynamic routing problem for a single vehicle can be formulated as follows,

**DR<sub>1</sub>**

$$\min_{t_i, i \in [1, M], \mathbf{p}_n} t_f \quad (20)$$

subject to

$$\begin{aligned} C_n(t_i) &\in B_i \\ t_{i-1} &< t_i < t_{i+1}, \forall i = 2, M-1 \\ t_1 &> 0, t_f > t_M \\ \|\dot{C}_n\|_\infty &\leq 1, \\ \|\ddot{C}_n\|_\infty &\leq 1, \\ C_n(0) &= C_n(t_f) = C_0 \end{aligned}$$



### 5.7.2. Numerical Solution: Single Vehicle Case

Let

$$\mathbf{C}_n^i(t) = \begin{cases} \sum_{k=0}^n \mathbf{P}_{i,k} B_{k,n}(t), t \in [t_{i-1}, t_i], t_0 = 0, t_{M+1} = t_f, i \in [1, M+1] \\ 0, \text{ o.w.} \end{cases} \quad (21)$$

Define

$$\mathbf{C}_n(t) = \sum_{i=1}^{M+1} \mathbf{C}_n^i(t) \quad (22)$$

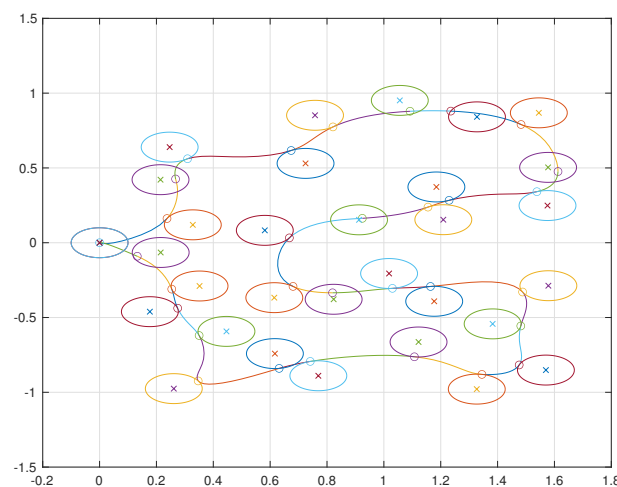
Then the numerical solution of the dynamic routing problem  $\mathbf{DR}_1$  was obtained by solving the optimization problem

$$\min_{t_i, i \in [1, M], \mathbf{P}_n^i} t_f \quad (23)$$

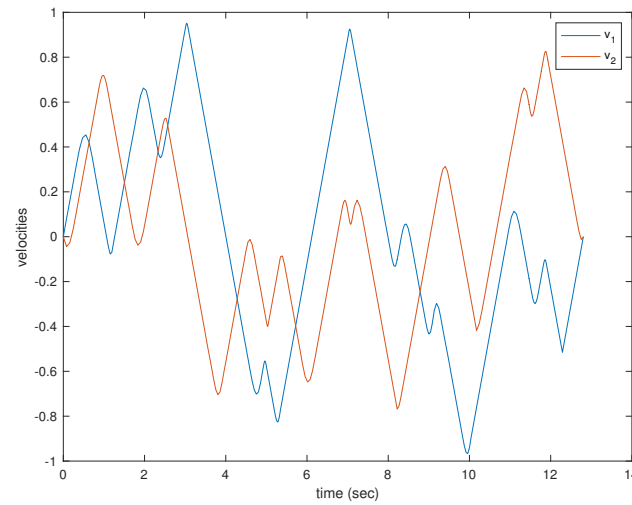
subject to

$$\begin{aligned} \|\mathbf{C}_n^i(t_i) - b_i\| &\leq r \\ t_{i-1} &< t_i < t_{i+1}, \forall i = 2, M-1 \\ t_1 &> 0, t_f > t_M \\ \|\dot{\mathbf{C}}_n\|_\infty &\leq 1, \\ \|\ddot{\mathbf{C}}_n\|_\infty &\leq 1, \\ \mathbf{C}_n(0) &= \mathbf{C}_n(t_f) = \mathbf{C}_0 \\ \mathbf{C}_n^i(t_i) &= \mathbf{C}_n^{i+1}(t_i), i \in [1, M-1] \\ \dot{\mathbf{C}}_n^i(t_i) &= \dot{\mathbf{C}}_n^{i+1}(t_i) i \in [1, M-1] \end{aligned}$$

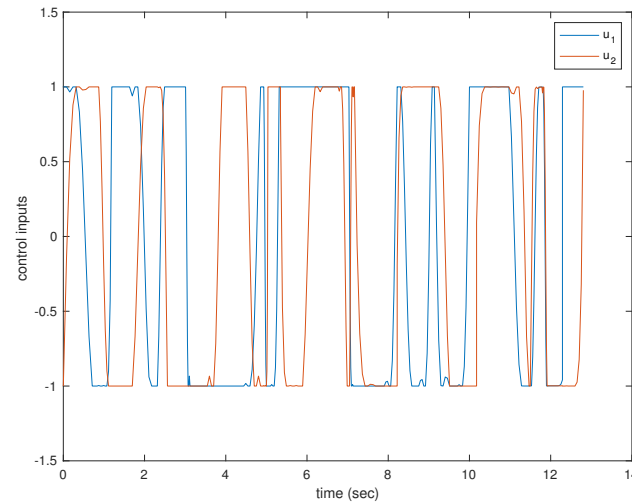
A simulation was performed illustrating a single agent visiting 30 neighborhoods. The resulting trajectory is shown in Figure 38. The agent is limited to velocities of arbitrary units ranging from  $-1$  to  $1$  and is similarly limited to accelerations of arbitrary units also from  $-1$  to  $1$ . The velocities and accelerations of the vehicle can be seen in Figure 39 and Figure 40, respectively.



**Figure 38.** Trajectory of a single agent visiting 30 neighborhoods. The circles with an X in their center represent the neighborhoods and the circles along the vehicle's trajectory represent the points at which the vehicle makes its delivery.



**Figure 39.** Velocity components of a single agent visiting 30 neighborhoods.



**Figure 40.** Acceleration components of a single agent visiting 30 neighborhoods.

### 5.7.3. Multiple Vehicle Case

In this case,  $K$  drones are assigned a total of  $K$  neighborhood sets to visit. Each neighborhood set,  $\mathcal{P}_k$ , consists of an equal number of neighborhoods  $B_{ij}$  which are defined by a set of points of interest  $b_{ik} \in \mathbb{R}^2$ , i.e.,

$$B_{ik} = \{x \in \mathbb{R}^2 : \|x - b_{ik}\| \leq r, i \in [1, M], k \in [1, K]\},$$

and

$$\mathcal{P}_k = \{B_{1k}, \dots, B_{Mk}, k \in [1, K]\}.$$

Let  $t_{fk}, k \in [1, K]$  denote the total time it takes for the  $k$ th vehicle to visit every neighborhood in the set  $\mathcal{P}_k$  once and let  $t_{ik}$  denote a time instance when the  $k$ th vehicle's position satisfies  $\mathbf{C}_n^k(t_{ik}) \in B_{ik}$ . Using this notation, we propose the following definition of the multi-vehicle dynamic routing problem for given positive number  $w_k$  and  $d$

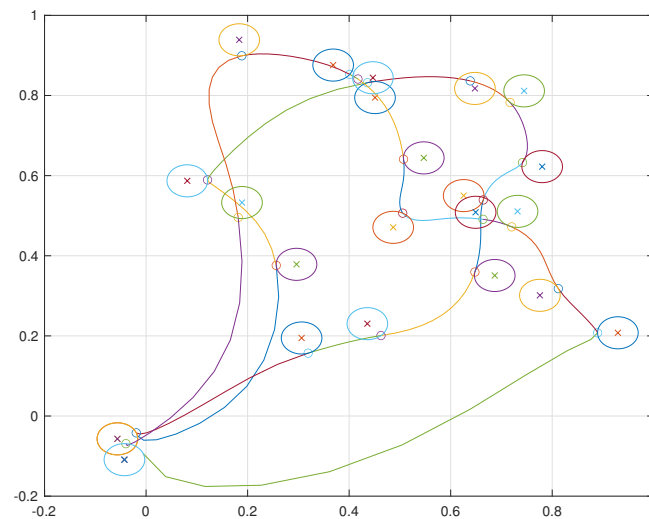
**DR<sub>2</sub>**

$$\min_{t_{ik}, i \in [1, M], k \in [1, K], \mathbf{P}_n^k} \sum_{k=1}^K w_k t_{fk} \quad (24)$$

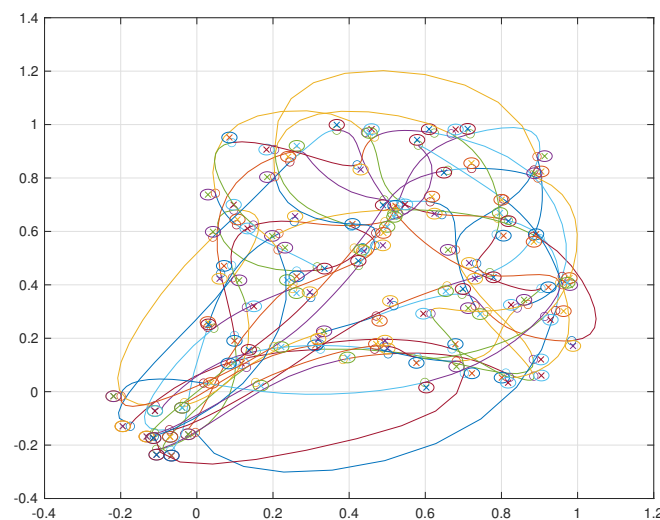
subject to

$$\begin{aligned}
 \mathbf{C}_n^k(t_{ik}) &\in B_{ik} \in \mathcal{P}_k, \forall k \in [1, K] \\
 t_{i-1,k} &< t_{ik} < t_{i+1,k}, \forall i \in [2, M-1], k \in [1, K] \\
 t_{1k} &> 0, t_{fk} > t_{Mk} \\
 |\ddot{\mathbf{C}}_n^k(t)| &\leq u_{max}, \forall t \in [0, t_{fk}] \\
 \mathbf{C}_n^k(0) &= \mathbf{C}_n^k(t_f) = \mathbf{C}_{k0}, \forall k \in [1, K] \\
 |\mathbf{C}_n^k(t) - \mathbf{C}_n^l(t)| &\geq d, k \neq l, \forall t \in [0, \max_{j \in [1, K]} t_{fj}], k \in [1, K], l \in [1, K].
 \end{aligned}$$

Simulations were performed for a two agent and ten agent case each visiting 10 neighborhoods per agent. The resulting trajectories for the two agent case are shown in Figure 41 and for the ten agent case in Figure 42.



**Figure 41.** Two agents visiting ten neighborhoods each.



**Figure 42.** Ten agents visiting ten neighborhoods each.

## 6. Conclusions

We presented a method to generate optimal trajectories by using Bernstein polynomials to transcribe the problem into a nonlinear programming problem. By exploiting the useful properties of Bernstein polynomials, our method provides computationally efficient algorithms that can also guarantee safety in continuous time which are useful in optimization routines. These algorithms include evaluating bounds, evaluating extrema, minimum spatial distance between two Bernstein polynomials, minimum spatial distance between a Bernstein polynomial and a convex polygon, collision detection, and the penetration algorithm. We also developed an open source toolbox which makes these transcription methods readily available in the Python programming language.

Numerical examples were provided to demonstrate the efficacy of the method. Simple cost functions and constraints were implemented to generate trajectories for several realistic mission scenarios including air traffic control, navigating a cluttered environment, overtaking a vehicle, trajectory generation for a large swarm of vehicles, trajectory generation for a marine vehicle, and navigation for vehicles operating in a Traveling Salesman mission. Our formulation offers a powerful tool for users to generate optimal trajectories in real time scenarios for single or multiple robot teams. Future work includes developing new cost functions, exploring different optimization frameworks, and replanning trajectories to react to a changing environment.

**Author Contributions:** Conceptualization, C.K.-J., V.C., I.K., A.P., C.W.; methodology, C.K.-J., V.C., I.K., A.P., C.W.; software, C.K.-J., T.B., I.K., C.W.; validation, C.K.-J., T.B., I.K., C.W.; investigation, C.K.-J., V.C., I.K., A.P., C.W.; resources, V.C., I.K., A.P.; writing—original draft preparation, C.K.-J., V.C., I.K.; writing—review and editing, C.K.-J.; supervision, V.C., I.K., A.P., C.W.; project administration, V.C.; funding acquisition, V.C., I.K., A.P. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the Office of Naval Research, grants N000141912106, N000142112091 and N0001419WX00155. Antonio Pascoal was supported by H2020-EU.1.2.2—FET Proactive RAMONES, under Grant GA 101017808 and LARSyS-FCT under Grant UIDB/50009/2020. Isaac Kaminer was supported by the Office of Naval Research grant N0001421WX01974.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A

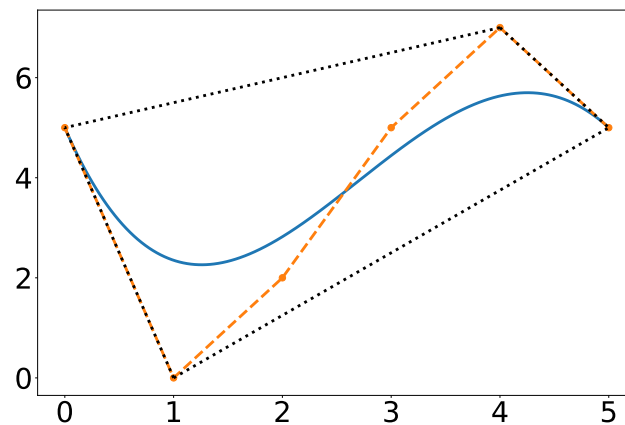
In this section we present and examine relevant properties of Bernstein polynomials and rational Bernstein polynomials, which are used throughout the article.

### Property A1. Convex Hull

Both the Bernstein polynomial, Equation (5), and the rational Bernstein polynomial (6) (provided that  $w_{i,n} > 0, i = 0, \dots, n$ ) satisfy

$$\min_{k \in \{0, \dots, n\}} P_k \leq C_n(t) \leq \max_{k \in \{0, \dots, n\}} P_k, \forall t \in [t_0, t_f]. \quad (\text{A1})$$

It follows that 2D (or 3D) Bernstein polynomials and rational Bernstein polynomials lie within the convex hull defined by their Bernstein coefficients. An example of this property is depicted in Figure A1, which shows a 2D Bernstein polynomial contained within its convex hull.



**Figure A1.** Convex hull of a Bernstein polynomial where the solid blue line is the curve, the black dotted line is the convex hull, and the orange points connected by dashed lines are the Bernstein coefficients.

**Property A2.** End Point Values

The first and last Bernstein coefficients of the Bernstein polynomial introduced in Equation (5), as well as the rational Bernstein polynomial in Equation (6), are their endpoints, i.e.,

$$C_n(t_0) = P_0 \quad \text{and} \quad C_n(t_f) = P_n.$$

Furthermore, the tangents to the curve at its first and last coefficients are the same as the lines connecting the first and second coefficients and the penultimate and ultimate coefficients, respectively. It follows that the first derivative of the Bernstein polynomial at the end points is as follows

$$\begin{aligned} \dot{C}_n(t_0) &= \frac{n}{t_f - t_0} (P_{1,n} - P_{0,n}), \\ \dot{C}_n(t_f) &= \frac{n}{t_f - t_0} (P_{n,n} - P_{n-1,n}). \end{aligned} \quad (\text{A2})$$

Similarly, for a rational Bernstein polynomial we have

$$\begin{aligned} \dot{C}_n(t_0) &= \frac{nw_1}{(t_f - t_0)w_0} (P_{1,n} - P_{0,n}), \\ \dot{C}_n(t_f) &= \frac{nw_{n-1}}{(t_f - t_0)w_n} (P_{n,n} - P_{n-1,n}). \end{aligned} \quad (\text{A3})$$

**Property A3.** Derivatives

The derivative of the Bernstein polynomial introduced in Equation (5) is an  $(n - 1)$ th order Bernstein polynomial given by

$$\dot{C}_{n-1}(t) = \sum_{i=0}^{n-1} P'_{i,n-1} B_{i,n-1}(t), \quad (\text{A4})$$

with the vector of Bernstein coefficients  $P'_{n-1} = [P'_{0,n-1}, \dots, P'_{n-1,n-1}]$  given by

$$P'_{n-1} = P_n \mathbf{D}_n.$$

In the equation above,  $\mathbf{D}_n$  denotes the differentiation matrix given by

$$\mathbf{D}_n = \frac{n}{t_f - t_0} \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 1 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \in \mathbb{R}^{n+1 \times n}. \quad (\text{A5})$$

**Remark A1.** Properties A2 and A3 can be easily extended to compute higher order derivatives of Bernstein polynomials.

**Property A4. Integrals**

The definite integral of a Bernstein polynomial is given by

$$\int_{t_0}^{t_f} C_n(t) dt = \frac{t_f - t_0}{n+1} \sum_{i=0}^n P_{i,n}. \quad (\text{A6})$$

**Property A5. The de Casteljau Algorithm**

The de Casteljau algorithm computes a Bernstein polynomial defined over an interval  $[t_0, t_f]$  at any given  $t_{div} \in [t_0, t_f]$ . Moreover, it can be used to split a single Bernstein polynomial into two independent Bernstein polynomials. Given the Bernstein polynomial introduced in Equation (5) with the vector of coefficients  $P_n = [P_{0,n}, \dots, P_{n,n}]$ , and a scalar  $t_{div} \in [t_0, t_f]$ , the Bernstein polynomial at  $t_{div}$  is computed using the following recursive relationship:

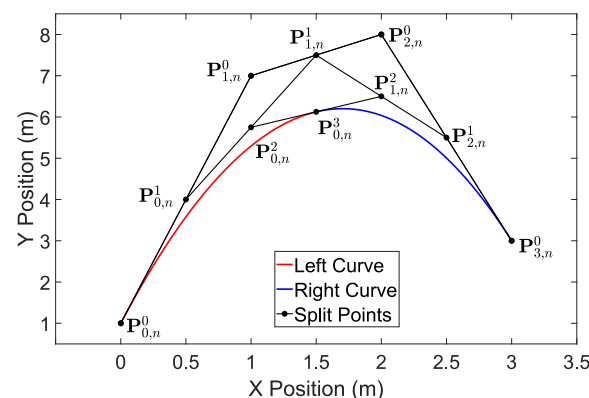
$$P_{i,n}^0 = P_{i,n}, \quad i = 0, \dots, n,$$

$$P_{i,n}^j = \frac{t_f - t_{div}}{t_f - t_0} P_{i,n}^{j-1} + \frac{t_{div} - t_0}{t_f - t_0} P_{i+1,n}^{j-1},$$

with  $i = 0, \dots, n-j$ , and  $j = 1, \dots, n$ . Then, the Bernstein polynomial evaluated at  $t_{div}$  is  $C_n(t_{div}) = P_{0,n}^n$ . Note that the superscript here signifies an index and not an exponent. Moreover, the Bernstein polynomial can be subdivided at  $t_{div}$  into two  $n$ th order Bernstein polynomials with coefficients

$$P_{0,n}^0, P_{0,n}^1, \dots, P_{0,n}^n \quad \text{and} \quad P_{0,n}^n, P_{1,n}^{n-1}, \dots, P_{n,n}^0.$$

A geometric example of a 3rd order Bernstein polynomial being split at  $t_{div} = 0.5$  using the de Casteljau is shown in Figure A2. Note that at the final iteration only a single point remains,  $P_{0,n}^3$ , and lies on the original curve. The points  $\{P_{0,n}^0, P_{0,n}^1, P_{0,n}^2, P_{0,n}^3\}$  become the Bernstein coefficients of the left curve and the points  $\{P_{0,n}^3, P_{1,n}^2, P_{2,n}^1, P_{3,n}^0\}$  become the coefficients of the right curve.



**Figure A2.** Geometric example of the de Casteljau algorithm splitting a 2D Bernstein polynomial at  $t_{div} = 0.5$ . The original curve is defined by the Bernstein coefficients  $\{P_{0,n}^0, \dots, P_{3,n}^0\}$ , the left hand curve is shown in red, the right hand curve is shown in blue, and the superscript corresponds to the current iteration of the algorithm.

**Remark A2.** For high order Bernstein polynomials, the exponential terms in Equation (5) can cause overflows. The de Casteljau algorithm can be used to overcome these issues on a computer when computing the value of a Bernstein polynomial, especially if lower bit floating point values are to be used such as 16 or 32 bits.

**Property A6.** Degree Elevation

The degree of a Bernstein polynomial can be elevated by one using the following recursive relationship

$$P_{i,n+1} = \frac{i}{n+1} P_{i-1,n} + \left(1 - \frac{i}{n+1}\right) P_{i,n},$$

where  $P_{0,n+1} = P_{0,n}$  and  $P_{n+1,n+1} = P_{n,n}$ . Furthermore, any Bernstein polynomial of degree  $n$  can be expressed as a Bernstein polynomial of degree  $m$ ,  $m > n$ . The vector of coefficients of the degree elevated Bernstein polynomial, namely  $\mathbf{P}_m = [P_{0,m}, \dots, P_{m,m}]$ , can be calculated as

$$\mathbf{P}_m = \mathbf{P}_n \mathbf{E},$$

where  $\mathbf{E} = \{e_{j,k}\} \in \mathbb{R}^{(n+1) \times (m+1)}$  is the degree elevation matrix with elements given by

$$e_{i,i+j} = \frac{\binom{m-n}{j} \binom{n}{i}}{\binom{m}{i+j}}, \quad (A7)$$

where  $i = 0, \dots, n$  and  $j = 0, \dots, m - n$ , all other values in the matrix are zero, and  $\mathbf{P}_n = [P_{0,n}, \dots, P_{n,n}]$  is the vector of Bernstein coefficients of the curve being elevated (see [52]). Because  $\mathbf{E}$  is conveniently independent of the coefficients, it can be computed ahead of time and stored for efficient computation of degree elevated Bernstein polynomials. By degree elevating a Bernstein polynomial, the coefficients converge to the polynomial, i.e.

$$\max_i \left| P_{i,m} - C_m \left( \frac{i}{m} (t_f - t_0) + t_0 \right) \right| \leq \frac{k}{m}, \quad (A8)$$

where  $k$  is a positive constant independent of  $m$  (see [53]).

**Property A7.** Arithmetic Operations

Arithmetic operations of Bernstein polynomials require that both curves be defined on the same time interval  $[t_0, t_f]$ . In case they are not, the de Casteljau algorithm can be used to split them at an intersecting time interval.

The sum and difference of two polynomials of the same order can be performed by simply adding and subtracting their Bernstein coefficients, respectively. For Bernstein polynomials of different order, the Degree Elevation (Property A6) may be used to increase the order of the lower order Bernstein polynomial.

Given two Bernstein polynomials,  $F_m(t)$  and  $G_n(t)$ , with degrees  $m$  and  $n$ , respectively, and having Bernstein coefficients  $X_{0,m}, \dots, X_{m,m}$  and  $Y_{0,n}, \dots, Y_{n,n}$ , the product  $C_{m+n}(t) = F_m(t)G_n(t)$  is a Bernstein polynomial of degree  $(m+n)$  with coefficients  $P_{k,m+n}$ ,  $k \in \{0, \dots, m+n\}$  given by

$$P_{k,m+n} = \sum_{j=\max(0,k-n)}^{\min(m,k)} \frac{\binom{m}{j} \binom{n}{k-j}}{\binom{m+n}{k}} X_{j,m} Y_{k-j,n}. \quad (A9)$$

The ratio between two Bernstein polynomials,  $F_n(t)$  and  $G_n(t)$ , with coefficients  $X_{0,n}, \dots, X_{n,n}$  and  $Y_{0,n}, \dots, Y_{n,n}$ , i.e.,  $R_n(t) = F_n(t)/G_n(t)$ , can be expressed as a rational Bernstein polynomial as defined in Equation (6), with coefficients and weights

$$P_{i,n} = \frac{X_{i,n}}{Y_{i,n}}, \quad w_{i,n} = Y_{i,n},$$

respectively.

**Property A8.** *The de Casteljau Algorithm Extended to Rational Bernstein Polynomials*

The de Casteljau algorithm can be extended to rational Bernstein polynomials (see [54]). Letting

$$w_{i,n}^r(t) = \sum_{j=0}^r w_{i+j,n} B_{i,r}(t), \quad (\text{A10})$$

we can determine a point on an  $n$ th order rational Bernstein polynomial using the recursive equation

$$\begin{aligned} P_{i,n}^r(t) = & \left( \frac{t_f - t}{t_f - t_0} \right) \frac{w_{i,n}^{r-1}(t)}{w_{i,n}^r(t)} P_{i,n}^{r-1}(t) \\ & + \left( \frac{t - t_0}{t_f - t_0} \right) \frac{w_{i+1,n}^{r-1}(t)}{w_{i+1,n}^r(t)} P_{i+1,n}^{r-1}(t). \end{aligned} \quad (\text{A11})$$

Moreover, the recursive relationship can be used to split the rational Bernstein polynomial into two  $n$ th order rational Bernstein polynomials with weights

$$w_{0,n}^0, w_{0,n}^1, \dots, w_{0,n}^n \quad \text{and} \quad w_{0,n}^n, w_{1,n}^{n-1}, \dots, w_{n,n}^0,$$

and coefficients

$$P_{0,n}^0, P_{0,n}^1, \dots, P_{0,n}^n \quad \text{and} \quad P_{0,n}^n, P_{1,n}^{n-1}, \dots, P_{n,n}^0.$$

**Property A9.** *Degree Elevation for Rational Bernstein Polynomials*

Degree elevation can be extended to rational Bernstein polynomials (see [54]). The  $n$ th order rational Bernstein polynomial given by Equation (6) can be rewritten as a rational Bernstein polynomial of order  $n + 1$  with weights

$$w_{i,n+1} = \frac{i}{n+1} w_{i-1,n} + \left( 1 - \frac{i}{n+1} \right) w_{i,n}.$$

where  $w_{0,n+1} = w_{0,n}$  and  $w_{n+1,n+1} = w_{n,n}$ , and coefficients

$$P_{i,n+1} = \frac{\frac{i}{n+1} w_{i,n} P_{i,n} + \left( 1 - \frac{i}{n+1} \right) w_{i+1,n} P_{i+1,n}}{\frac{i}{n+1} w_{i,n} + \left( 1 - \frac{i}{n+1} \right) w_{i+1,n}}.$$

**References**

1. Milford, M.; Anthony, S.; Scheirer, W. Self-driving vehicles: Key technical challenges and progress off the road. *IEEE Potentials* **2019**, *39*, 37–45. [CrossRef]
2. Mogili, U.R.; Deepak, B. Review on application of drone systems in precision agriculture. *Procedia Comput. Sci.* **2018**, *133*, 502–509. [CrossRef]
3. Koerhuis, R. Odd.Bot Robot Takes on Herbicide Free Weed Elimination. 2018. Available online: <https://www.futurefarming.com/Tools-data/Articles/2018/12/OddBot-robot-takes-on-herbicide-free-weed-elimination-375027E/> (accessed on 4 May 2020)
4. Lutz, M.; Meurer, T. Optimal Trajectory Planning and Model Predictive Control of Underactuated Marine Surface Vessels using a Flatness-Based Approach. *arXiv* **2021**, arXiv:2101.12730.
5. Ackerman, E. Startup Developing Autonomous Delivery Robots That Travel on Sidewalks. 2015. Available online: <https://spectrum.ieee.org/automaton/robotics/industrial-robots/starship-technologies-autonomous-ground-delivery-robots> (accessed on 4 May 2020)
6. Ackerman, E.; Koziol, M. In the Air with Zipline's Medical Delivery Drones. 2019. Available online: <https://spectrum.ieee.org/robotics/drones/in-the-air-with-ziplines-medical-delivery-drones> (accessed on 4 May 2020)
7. Lardinois, F. A First Look at Amazon's New Delivery Drone. 2019. Available online: <https://techcrunch.com/2019/06/05/a-first-look-at-amazons-new-delivery-drone/> (accessed on 4 May 2020)
8. Johnson, A.; Hautaluoma, G. NASA's Ingenuity Mars Helicopter Succeeds in Historic First Flight. 2021. Available online: <https://www.nasa.gov/press-release/nasa-s-ingenuity-mars-helicopter-succeeds-in-historic-first-flight> (accessed on 18 June 2020).
9. Lumelsky, V.J.; Stepanov, A.A. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica* **1987**, *2*, 403–430. [CrossRef]



10. Kamon, I.; Rivlin, E. Sensory-based motion planning with global proofs. *IEEE Trans. Robot. Autom.* **1997**, *13*, 814–822. [\[CrossRef\]](#)
11. McGuire, K.; de Croon, G.; Tuyls, K. A Comparative Study of Bug Algorithms for Robot Navigation. *Robot. Auton. Syst.* **2019**, *121*, 103261. [\[CrossRef\]](#)
12. Fiorini, P.; Shiller, Z. Motion planning in dynamic environments using the relative velocity paradigm. In Proceedings of the IEEE International Conference on Robotics and Automation, Atlanta, GA, USA, 2–6 May 1993; pp. 560–565.
13. Abe, Y.; Yoshiki, M. Collision avoidance method for multiple autonomous mobile agents by implicit cooperation. In Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180), Maui, HI, USA, 29 October–3 November 2001; Volume 3, pp. 1207–1212.
14. Large, F.; Sckhavat, S.; Shiller, Z.; Laugier, C. Using non-linear velocity obstacles to plan motions in a dynamic environment. In Proceedings of the 7th International Conference on Control, Automation, Robotics and Vision, ICARCV 2002, Singapore, 2–5 December 2002; Volume 2, pp. 734–739.
15. Wilkie, D.; Van Den Berg, J.; Manocha, D. Generalized velocity obstacles. In Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO, USA, 10–15 October 2009; pp. 5573–5578.
16. Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous Robot Vehicles*; Springer: New York, NY, USA, 1986; pp. 396–404.
17. Chen, Y.B.; Luo, G.C.; Mei, Y.S.; Yu, J.Q.; Su, X.L. UAV path planning using artificial potential field method updated by optimal control theory. *Int. J. Syst. Sci.* **2016**, *47*, 1407–1420. [\[CrossRef\]](#)
18. Orozco-Rosas, U.; Montiel, O.; Sepúlveda, R. Mobile robot path planning using membrane evolutionary artificial potential field. *Appl. Soft Comput.* **2019**, *77*, 236–251. [\[CrossRef\]](#)
19. Kavraki, L.E.; Svestka, P.; Latombe, J.C.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580. [\[CrossRef\]](#)
20. LaValle, S.M.; Kuffner, J.J., Jr. Randomized kinodynamic planning. *Int. J. Robot. Res.* **2001**, *20*, 378–400. [\[CrossRef\]](#)
21. Karaman, S.; Frazzoli, E. Incremental sampling-based algorithms for optimal motion planning. *arXiv* **2010**, arXiv:1005.0416.
22. Sleumer, N.; Tschichold-Gürmann, N. *Exact Cell Decomposition of Arrangements Used for Path Planning in Robotics*; Technical Report; ETH Zurich, Department of Computer Science: Zurich, Switzerland, 1999; Volume 329.
23. Cai, C.; Ferrari, S. Information-driven sensor path planning by approximate cell decomposition. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **2009**, *39*, 672–689.
24. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [\[CrossRef\]](#)
25. Vasconcelos, J.V.R.; Brandão, A.S.; Sarcinelli-Filho, M. Real-Time Path Planning for Strategic Missions. *Appl. Sci.* **2020**, *10*, 7773. [\[CrossRef\]](#)
26. Likhachev, M.; Ferguson, D.I.; Gordon, G.J.; Stentz, A.; Thrun, S. Anytime Dynamic A\*: An Anytime, Replanning Algorithm. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), Monterey, CA, USA, 5–10 June 2005; Volume 5, pp. 262–271.
27. Karaman, S.; Walter, M.R.; Perez, A.; Frazzoli, E.; Teller, S. Anytime motion planning using the RRT. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 1478–1483.
28. Mellinger, D.; Kumar, V. Minimum snap trajectory generation and control for quadrotors. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 2520–2525.
29. Schulman, J.; Ho, J.; Lee, A.X.; Awwal, I.; Bradlow, H.; Abbeel, P. Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization. In Proceedings of the Robotics: Science and Systems, Berlin, Germany, 24–28 June 2013; Volume 9, pp. 1–10. [\[CrossRef\]](#)
30. Ratliff, N.; Zucker, M.; Bagnell, J.A.; Srinivasa, S. CHOMP: Gradient optimization techniques for efficient motion planning. In Proceedings of the 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan, 12–17 May 2009; pp. 489–494.
31. Zucker, M.; Ratliff, N.; Dragan, A.D.; Pivtoraiko, M.; Klingensmith, M.; Dellin, C.M.; Bagnell, J.A.; Srinivasa, S.S. Chomp: Covariant hamiltonian optimization for motion planning. *Int. J. Robot. Res.* **2013**, *32*, 1164–1193. [\[CrossRef\]](#)
32. Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; Schaal, S. STOMP: Stochastic trajectory optimization for motion planning. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 4569–4574.
33. Tang, S.; Thomas, J.; Kumar, V. Hold or Take Optimal Plan (HOOP): A quadratic programming approach to multi-robot trajectory generation. *Int. J. Robot. Res.* **2018**, *37*, 1062–1084. [\[CrossRef\]](#)
34. Semenas, R.; Bausys, R.; Zavadskas, E.K. A Novel Environment Exploration Strategy by m-generalised q-neutrosophic WASPAS. *Stud. Inform. Control* **2021**, *30*, 19–28. [\[CrossRef\]](#)
35. Farouki, R.; Goodman, T. On the optimal stability of the Bernstein basis. *Math. Comput. Am. Math. Soc.* **1996**, *65*, 1553–1566. [\[CrossRef\]](#)
36. Cichella, V.; Kaminer, I.; Walton, C.; Hovakimyan, N. Optimal Motion Planning for Differentially Flat Systems Using Bernstein Approximation. *IEEE Control Syst. Lett.* **2018**, *2*, 181–186. [\[CrossRef\]](#)
37. Cichella, V.; Kaminer, I.; Walton, C.; Hovakimyan, N.; Pascoal, A. Bernstein approximation of optimal control problems. *arXiv* **2018**, arXiv:1812.06132.

38. Cichella, V.; Kaminer, I.; Walton, C.; Hovakimyan, N.; Pascoal, A.M. Optimal Multi-Vehicle Motion Planning using Bernstein Approximants. *IEEE Trans. Autom. Control* **2020**, *66*, 1453–1467. [\[CrossRef\]](#)
39. Kielas-Jensen, C.; Cichella, V. BeBOT: Bernstein polynomial toolkit for trajectory generation. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 3–8 November 2019; pp. 3288–3293.
40. Kielas-Jensen, C.; Cichella, V. BeBOT. Available online: <https://github.com/caslabuiowa/BeBOT> (accessed on 21 May 2020).
41. Bézier, P. Definition numerique des courbes et surface. *Automatisme* **1966**, *11*, 625–632.
42. Bézier, P. Définition numérique des courbes et surfaces (ii). *Automatisme* **1967**, *12*, 17–21.
43. Forrest, A.R. Interactive interpolation and approximation by Bézier polynomials. *Comput. J.* **1972**, *15*, 71–79. [\[CrossRef\]](#)
44. Chang, J.W.; Choi, Y.K.; Kim, M.S.; Wang, W. Computation of the minimum distance between two Bézier curves/surfaces. *Comput. Graph.* **2011**, *35*, 677–684. [\[CrossRef\]](#)
45. Gilbert, E.G.; Johnson, D.W.; Keerthi, S.S. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE J. Robot. Autom.* **1988**, *4*, 193–203. [\[CrossRef\]](#)
46. Van Den Bergen, G. Proximity queries and penetration depth computation on 3d game objects. In Proceedings of the Game Developers Conference, San Jose, CA, USA, 20–24 March 2001; Volume 170.
47. Kielas-Jensen, C.; Cichella, V. Trajectory Generation Using Bernstein Polynomials (Bézier Curves). Available online: <https://www.youtube.com/watch?v=e0SPt92W578> (accessed on 31 July 2019).
48. *Programming Language C Standard*; International Organization for Standardization: Geneva, Switzerland, 1999.
49. Patterson, A.; Lakshmanan, A.; Hovakimyan, N. Intent-aware probabilistic trajectory estimation for collision prediction with uncertainty quantification. In Proceedings of the 2019 IEEE 58th Conference on Decision and Control (CDC), Nice, France, 11–13 December 2019; pp. 3827–3832.
50. Hauser, J.; Saccon, A. A barrier function method for the optimization of trajectory functionals with constraints. In Proceedings of the 45th IEEE Conference on Decision and Control, San Diego, CA, USA, 13–15 December 2006; pp. 864–869.
51. Zhang, X.; Liniger, A.; Borrelli, F. Optimization-based collision avoidance. *IEEE Trans. Control. Syst. Technol.* **2020**, *29*, 972–983. [\[CrossRef\]](#)
52. Lee, B.G.; Park, Y. Distance for Bézier curves and degree reduction. *Bull. Aust. Math. Soc.* **1997**, *56*, 507–515. [\[CrossRef\]](#)
53. Prautzsch, H.; Kobbelt, L. Convergence of subdivision and degree elevation. *Adv. Comput. Math.* **1994**, *2*, 143–154. [\[CrossRef\]](#)
54. Farin, G. Algorithms for rational Bézier curves. *Comput.-Aided Des.* **1983**, *15*, 73–77. [\[CrossRef\]](#)