Bernstein polynomial-based transcription method for solving optimal trajectory generation problems

Calvin Kielas-Jensen\textsuperscript{1} and Venanzio Cichella\textsuperscript{1}

Abstract—This paper presents a method and an open-source implementation, Bernstein/Bézier Optimal Trajectories (BeBOT), for the generation of trajectories for autonomous system operations. The proposed method is based on infinite dimensional optimal control formulations of trajectory generation problems. By approximating the trajectories using Bernstein polynomials, these problems can be transcribed as nonlinear programming problems, which can then be solved using off-the-shelf solvers. Bernstein polynomials possess favorable geometric properties that enable the trajectory planner to efficiently evaluate and enforce constraints along the vehicles’ trajectories, including maximum speed and angular rates, minimum distance between trajectories and between the vehicles and obstacles. By virtue of these properties, feasibility and safety constraints typically imposed in autonomous vehicle operations can be enforced and guaranteed independently on the order of the polynomials. Thus, the trajectory generation algorithm can efficiently generate feasible and collision-free trajectories, and can be deployed for real-time safety critical applications in complex environments and for multiple vehicle missions.

I. INTRODUCTION

The field of autonomous navigation 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 development in autonomous weed pulling robots may reduce or eliminate the need for potentially harmful pesticides [3]. Companies such as Amazon, Starship, and Zipline have already begun making autonomous deliveries [4], [5], [6]. There is even a mission underway to fly the first autonomous aerial vehicle on a different planet [7]. This progress has led to high demand for computationally efficient feasible algorithms that allow 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 value should be. On the other hand, open-loop methods determine control values or 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. [8], [9], 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 provide optimal paths and does not generally take the system’s dynamics into account. For a review and comparison of bug algorithms, the reader is referred to [10].

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 [11]. Variations on the VO method include Common Velocity Obstacle [12], Nonlinear Velocity Obstacles [13], and Generalized Velocity Obstacles [14]. 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 [15], [16], [17].

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.

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) [18] and rapidly exploring random trees (RRT,RRT*) [19], [20] 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

\textsuperscript{1}Calvin Kielas-Jensen and Venanzio Cichella are with the Mechanical Engineering department, University of Iowa, Iowa City, IA 52242, USA calvin-kielas-jensen@uiowa.edu, venanzio-cichella@uiowa.edu
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 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. [21], [22], 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 [23] which is an extension of Dijkstra’s algorithm that uses a heuristic function to improve the search speed.

In addition to graph-based representations of trajectories, polynomial approximation methods can be used as well. In [24] trajectories are represented as piecewise polynomial functions and are generated in a manner that minimizes their snap. In TrajOpt [25] a sequential quadratic program is solved to generate optimal polynomial trajectories while performing continuous time collision checking.

Other open-loop methods include CHOMP [26], [27], STOMP [28], and HOOP [29]. 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 robot’s kinematics. The second step then refines this trajectory into a higher order piecewise polynomial using a quadratic program.

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 the Weierstrass theorem. Bernstein polynomials were not widely used until the advent of digital computers due to their slow convergence as a function approximation. 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 [30], 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 the representation of continuous time trajectories using low-order approximations.

Our method for trajectory generation builds upon [31], [32], [33], where Bernstein polynomials were introduced as tools to approximate nonlinear optimal control problems with provable convergence guarantees. This paper also extends the preliminary work shown in [34] by providing an open-source implementation of algorithms for Bernstein polynomials, as well as examples that illustrate their use for trajectory generation applications.

We formulate the motion planning problem as an infinite dimensional optimal control problem. By parameterizing the trajectories as Bernstein polynomials, this problem can then be transcribed into a nonlinear programming problem using, which can then be solved using an off-the-shelf solution such as MATLAB’s fmincon, SciPy Python’s minimize, or SNOPT.

The paper is structured as follows. In Section II we introduce the Bernstein polynomials and their properties. In Section III we present computationally efficient algorithms for the computation of state and input constraints typical of trajectory generation applications. In Section IV we demonstrate the efficacy of these algorithms through several numerical examples. The paper ends with Section V, 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, [35].

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

II. Bernstein Polynomials

This section examines the properties of Bernstein polynomials and rational Bernstein polynomials that are relevant to this paper.
A 1-dimensional, \( n \)th order Bernstein polynomial, \( C_n(t) \), is defined as
\[
C_n(t) = \sum_{i=0}^{n} P_{i,n}B_{i,n}(t), \quad t \in [t_0, t_f],
\tag{1}
\]
where \( P_{i,n} \in \mathbb{R} \) is the \( i \)th Bernstein coefficient and \( B_{i,n}(t) \) is the Bernstein polynomial basis defined as
\[
B_{i,n}(t) = \binom{n}{i} \frac{(t - t_0)^i (t_f - t)^{n-i}}{(t_f - t_0)^n},
\]
for all \( i = 0, \ldots, n \), where
\[
\binom{n}{i} = \frac{n!}{i!(n-i)!}
\]
is the binomial coefficient.

Similarly, an \( N \)-dimensional, \( n \)th order Bernstein polynomial, \( \mathbf{C}_n(t) \), is defined as
\[
\mathbf{C}_n(t) = \sum_{i=0}^{n} \mathbf{P}_{i,n} B_{i,n}(t), \quad t \in [t_0, t_f],
\tag{2}
\]
with \( \mathbf{P}_{i,n} \in \mathbb{R}^N \). Bernstein polynomials can be used to describe 2D (or 3D) spatial curves. In this case, Bernstein polynomials are often referred to as Bézier curves. While Bézier’s original work did not explicitly use the Bernstein basis \([36],[37]\), it was later shown that the original formulation is equivalent to the Bernstein form polynomial \([38]\).

A 1-dimensional, \( n \)th order rational Bernstein polynomial, \( R_n(t) \), is defined as
\[
R_n(t) = \frac{\sum_{i=0}^{n} P_{i,n}w_{i,n}B_{i,n}(t)}{\sum_{i=0}^{n} w_{i,n}B_{i,n}(t)}, \quad t \in [t_0, t_f],
\tag{3}
\]
where \( w_{i,n} \in \mathbb{R}, \ i = 0, \ldots, n, \) are referred to as weights.

We provide a review of relevant properties of (rational) Bernstein polynomials, which will be used in the remainder of this paper. For an extensive review on Bernstein polynomials the reader is referred to \([39]\).

**Property 1: Convex Hull**

The Bernstein polynomial introduced in (2), as well as the rational Bernstein polynomial in (3) (provided that \( w_{i,n} > 0, \ i = 0, \ldots, n, \)) lie within the convex hull defined by their Bernstein coefficients. An illustrative example of this property is depicted in Figure 1 which shows a 2-dimensional Bernstein polynomial within its convex hull.

Given the 1-dimensional Bernstein polynomial introduced in (1), the convex hull property implies
\[
\min_{k \in \{0,\ldots,n\}} P_k \leq C_n(t) \leq \max_{k \in \{0,\ldots,n\}} P_k, \ \forall t \in [t_0, t_f].
\tag{4}
\]

**Property 2: End Point Values**

The first and last Bernstein coefficients of the Bernstein polynomial introduced in (2), as well as the rational Bernstein polynomial in (3), are their endpoints, i.e.,
\[
C_n(t_0) = \mathbf{P}_0 \quad \text{and} \quad C_n(t_f) = \mathbf{P}_n.
\]
Furthermore, the first derivative of the Bernstein polynomial at the end points can be easily computed as follows
\[
\dot{C}_n(t_0) = \frac{n}{t_f - t_0}(\mathbf{P}_{1,n} - \mathbf{P}_{0,n}),
\]
\[
\dot{C}_n(t_f) = \frac{n}{t_f - t_0}(\mathbf{P}_{n,n} - \mathbf{P}_{n-1,n}),(5)
\]
Similarly, for a rational Bernstein polynomial we have
\[
\dot{C}_n(t_0) = \frac{nw_1}{(t_f - t_0)w_0}(\mathbf{P}_{1,n} - \mathbf{P}_{0,n}),
\]
and
\[
\dot{C}_n(t_f) = \frac{nw_{n-1}}{(t_f - t_0)w_n}(\mathbf{P}_{n,n} - \mathbf{P}_{n-1,n}).\tag{6}
\]

**Remark 1: Property 2 and Equation (5) can be easily extended to compute higher order derivatives of Bernstein polynomials at the end points.**

**Property 3: Derivatives**

The derivative of the Bernstein polynomial introduced in (2) is an \( (n-1) \)th order Bernstein polynomial
\[
\mathbf{C}_{n-1}(t) = \sum_{i=0}^{n-1} \mathbf{P}_{i,n-1} B_{i,n-1}(t),
\tag{8}
\]
with vector of Bernstein coefficients \( \mathbf{P}_{n-1} = [\mathbf{P}_{0,n-1}, \ldots, \mathbf{P}_{n-1,n-1}] \) given by
\[
\mathbf{P}_{n-1} = \mathbf{P}'_n D_n,
\]
where
\[
D_n = \frac{n}{t_f - t_0} \begin{bmatrix}
-1 & 0 & \cdots & 0 \\
1 & \ddots & \ddots & \vdots \\
0 & \ddots & \ddots & -1 \\
\vdots & \ddots & \ddots & 0 \\
0 & \cdots & 0 & 1
\end{bmatrix} \in \mathbb{R}^{n+1 \times n}.
\tag{9}
\]

**Property 4: Integrals**

The definite integral of a Bernstein polynomial can be found from its coefficients,
\[
\int_{t_0}^{t_f} C_n(t) dt = \frac{t_f - t_0}{n+1} \sum_{i=0}^{n} P_{i,n}.
\tag{10}
\]
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 \( m > n \). The vector of coefficients of the degree elevated Bernstein polynomial, namely \( P_m = [P_{0,m}, \ldots, P_{m,m}] \), can be calculated as

\[ P_m = P_n E, \]

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

\[ e_{i,i+j} = \binom{m-n}{i} \binom{n}{j+i}, \]

(11)

where \( i = 0, \ldots, n \) and \( j = 0, \ldots, m - n \), all other values in the matrix are zero, and \( P_n = [P_{0,n}, \ldots, P_{n,n}] \) is the vector of Bernstein coefficients of the curve being elevated (see [40]). By degree elevating a Bernstein polynomial, the coefficients converge to the polynomial, i.e.

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

(12)

where \( k \) is a positive constant independent of \( m \) (see [41]).

**Property 5: The de Casteljau Algorithm**

The de Casteljau algorithm computes a Bernstein polynomial defined over an interval \([t_0, t_f]\) at any given \( t_{\text{div}} \in [t_0, t_f] \). Moreover, it can be used to split a single Bernstein polynomial into two independent Bernstein polynomials. Given the Bernstein polynomial introduced in (2) with the vector of coefficients \( P_n = [P_{0,n}, \ldots, P_{n,n}] \), and a scalar \( t_{\text{div}} \in [t_0, t_f] \), the Bernstein polynomial at \( t_{\text{div}} \) is computed using the following recursive relationship:

\[ P_{i,n}^0 = P_{i,n}, \quad i = 0, \ldots, n, \]

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

with \( i = 0, \ldots, n-j \), and \( j = 1, \ldots, n \). Then, the Bernstein polynomial evaluated at \( t_{\text{div}} \) is given by \( C_n(t_{\text{div}}) = P_{0,n}^0 \).

Note that the superscript here signifies an index and not an exponent. Moreover, the Bernstein polynomial can be subdivided at \( t_{\text{div}} \) into two \( n \)th order Bernstein polynomials with coefficients

\[ P_{0,n}^0, P_{0,n}^{1}, \ldots, P_{0,n}^{n} \quad \text{and} \quad P_{n,n}^0, P_{n,n}^{n-1}, \ldots, P_{n,n}^0. \]

A geometric example of a 3rd order Bernstein polynomial being split at \( t_{\text{div}} = 0.5 \) using the de Casteljau is shown in Figure 2. 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.

**Property 6: 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 \( m > n \). The vector of coefficients of the degree elevated Bernstein polynomial, namely \( P_m = [P_{0,m}, \ldots, P_{m,m}] \), can be calculated as

\[ P_m = P_n E, \]

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

\[ e_{i,i+j} = \binom{m-n}{i} \binom{n}{j+i}, \]

(11)

where \( i = 0, \ldots, n \) and \( j = 0, \ldots, m - n \), all other values in the matrix are zero, and \( P_n = [P_{0,n}, \ldots, P_{n,n}] \) is the vector of Bernstein coefficients of the curve being elevated (see [40]). By degree elevating a Bernstein polynomial, the coefficients converge to the polynomial, i.e.

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

(12)

where \( k \) is a positive constant independent of \( m \) (see [41]).

**Property 7: Arithmetic Operations**

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

The sum (difference) of two polynomials of the same order can be performed by simply adding (subtracting) their Bernstein coefficients. If the Bernstein polynomials are not of the same order, Degree Elevation (Property 6) may be used to increase the order of the lower order Bernstein polynomial.

Let \( F_m(t) \) and \( G_n(t) \) be two 1-dimensional Bernstein polynomials of degree \( m \) and \( n \), respectively, with Bernstein coefficients \( X_0, m, X_{m,m} \) and \( Y_0, n, Y_{n,n} \). The product \( C_{m+n}(t) = F_m(t) G_n(t) \) is a Bernstein polynomial of degree \( (m+n) \) with coefficients \( P_{k,m+n} \) given by

\[ P_{k,m+n} = \min(m,k) \sum_{j=max(0,k-n)}^\min(m,k) \binom{m}{j} \binom{n}{k-j} X_j Y_{k-j,n}. \]

(13)

The ratio between two 1-dimensional Bernstein polynomials, \( F_m(t) \) and \( G_n(t) \), with coefficients \( X_0, m, X_{m,m} \) and \( Y_0, n, Y_{n,n} \), i.e., \( R_m(t) = F_m(t) / G_n(t) \), can be expressed as a rational Bernstein polynomial as defined in (3), with coefficients and weights

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

respectively.

**Property 8: Rational Recursive Algorithm**

The de Casteljau algorithm can be extended to rational Bernstein polynomials (see [42]). Letting

\[ w_{i,n}(t) = \sum_{j=0}^r w_{i+j,n} B_{i,r}(t), \]

(14)
we can determine a point on an $n$th order rational Bernstein polynomial using the following recursive equation
\[
P_{i,n}^{r}(t) = \left( \frac{t_f - t}{t_f - t_0} \right) w_{i,n}^{r-1}(t) P_{i,n}^{r-1}(t) + \left( \frac{t - t_0}{t_f - t_0} \right) w_{i+1,n}^{r-1}(t) P_{i+1,n}^{r-1}(t).
\]

(15)

Moreover, the recursive relationship can be used to split the rational Bernstein polynomial into two $n$th order rational Bernstein polynomials with weights
\[
\begin{align*}
w_{0,n}^0, w_{0,n}^1, \ldots, w_{0,n}^n \\
w_{n,0}^0, w_{n,0}^1, \ldots, w_{n,0}^n
\end{align*}
\]
and coefficients
\[
\begin{align*}
P_{0,n}^0, P_{0,n}^1, \ldots, P_{0,n}^n \\
P_{n,0}^0, P_{n,0}^1, \ldots, P_{n,0}^n.
\end{align*}
\]

Property 9: Rational Degree Elevation

Degree elevation can be extended to rational Bernstein polynomials (see [42]). Given an $n$th order rational Bernstein polynomial, the identical rational Bernstein polynomial of order $(n + 1)$ can be determined. The weights are found using
\[
w_{i,n+1} = \frac{i}{n+1} w_{i-1,n} + \left( 1 - \frac{i}{n+1} \right) w_{i,n},
\]
where $w_{0,n+1} = w_{0,n}$ and $w_{n+1,n+1} = w_{n,n}$. The Bernstein coefficients are found using
\[
P_{i,n+1} = \frac{i}{n+1} w_{i,n} P_{i,n} + \left( 1 - \frac{i}{n+1} \right) w_{i+1,n} P_{i+1,n}.
\]

A. Numerical Examples in 2D

Here we will examine several numerical examples using the properties of Bernstein polynomials in 2D. All the plots presented can be generated using the example code available at [35].

Figure 3 contains several examples of 2-dimensional trajectories. Two trajectories are plotted in Figure 3a along with an obstacle. The trajectories $C^{[1]}(t)$ and $C^{[2]}(t)$ are defined as in (2) with $t_0 = 100$ and $t_f = 200$. The Bernstein coefficients of trajectory $C^{[1]}(t)$ are
\[
P_5^{[1]} = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 & 10 \\ 5 & 0 & 2 & 3 & 10 & 3 \end{bmatrix}.
\]
The Bernstein coefficients of trajectory $C^{[2]}(t)$ are
\[
P_5^{[2]} = \begin{bmatrix} 1 & 3 & 6 & 8 & 10 & 12 \\ 6 & 9 & 10 & 11 & 8 & 8 \end{bmatrix}.
\]
The circular obstacle has a radius of 1 and is centered at point [3, 4]T. Figure 3b highlights the endpoints property (Property 2). Note that the trajectory $C^{[1]}(t)$ passes through its first and last Bernstein coefficients which are $P_0^{[1]} = [0, 5]$T and $[10, 3]$T, respectively. Likewise, the trajectory $C^{[2]}(t)$ passes through its first and last Bernstein coefficients which are $[1, 6]$T and $[12, 8]$T, respectively. The convex hull property (Property 1) is illustrated in Figure 3c.

Useful operations can be efficiently performed on Bernstein polynomials by only manipulating the coefficients. The de Casteljau algorithm (Property 5) allows one to split a Bernstein polynomial into two separate polynomials. This is shown in Figure 3d where trajectories $C^{[1]}(t)$ and $C^{[2]}(t)$ are split at $t_{\text{div}} = 150$. Degree elevation (Property 6) is performed on both trajectories in Figure 3e. Note that in both Figure 3d and 3e the convex hulls are more accurate than the conservative convex hulls in Figure 3c. This idea will be expanded upon in the next section.

Bernstein polynomials can also be used to extract useful dynamics and control information. In Figure 3f Bernstein polynomials representing the squared speed of trajectories $C^{[1]}(t)$ and $C^{[2]}(t)$ are shown along with their corresponding coefficients and convex hulls. The squared speed is computed using the derivative and arithmetic operation properties (Properties 3 and 7) as follows
\[
(\dot{x}^{[1]}(t))^2 = (\dot{x}^{[1]}(t))^2 + (\dot{y}^{[1]}(t))^2
\]

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

Figure 3g shows the tangent of the heading angle of trajectories $C^{[3]}(t)$ and $C^{[4]}(t)$. For example, letting $C^{[1]}(t) = [x^{[1]}(t), y^{[1]}(t)]$T, the heading angle $\psi(t)$ of a trajectory can be found by
\[
\psi^{[1]}(t) = \tan^{-1}\left( \frac{\dot{y}^{[1]}(t)}{\dot{x}^{[1]}(t)} \right),
\]

Since the inverse tangent of a Bernstein polynomial is not a Bernstein polynomial, we take the tangent of both sides of the equation and represent the tangent of the heading angle as a rational Bernstein polynomial (see Figure 3g).

To determine the angular rate, we can take the derivative of the heading angle,
\[
\omega^{[1]}(t) = \dot{\psi}^{[1]}(t) = \frac{\dot{x}^{[1]}(t)\dot{y}^{[1]}(t) - \dot{y}^{[1]}(t)\dot{x}^{[1]}(t)}{(\dot{x}^{[1]}(t))^2 + (\dot{y}^{[1]}(t))^2}.
\]

Since the angular rate can be determined using Properties 3 and 7, it can be represented as a rational Bernstein polynomial. The angular rates of trajectories $C^{[1]}(t)$ and $C^{[2]}(t)$ are shown in Figure 3h.

Finally, the Bernstein polynomial representing the squared distance between two trajectories at every point in time can be found by
\[
d^2(t) = (x^{[2]}(t) - x^{[1]}(t))^2 + (y^{[2]}(t) - y^{[1]}(t))^2,
\]

\[
\forall t \in [t_0, t_f]
\]

where $d^2(t)$ is the squared distance between two trajectories whose positions are defined by $[x^{[1]}(t), y^{[1]}(t)]$T and $[x^{[2]}(t), y^{[2]}(t)]$T. The position of a static obstacle $\text{Obs}(t)$ can be represented as a Bernstein polynomial whose coefficients are all identical and set to the position of the obstacle, i.e.
\[
\text{P}[\text{Obs}] = \begin{bmatrix} x^{[\text{Obs}]} & \ldots & x^{[\text{Obs}]} \\ y^{[\text{Obs}]} & \ldots & y^{[\text{Obs}]} \end{bmatrix}.
\]

The degree of the Bernstein polynomial representing the position of the obstacle is equal to that of the order of the
Bernstein polynomials representing the trajectories. In order to represent the speed and distance as Bernstein polynomials, it is necessary to use their squared values because the square root of a Bernstein polynomial is not a Bernstein polynomial.

B. Numerical Examples in 3D

Next, we introduce two 3D Bernstein polynomials with \( t_0 = 10s \) and \( t_f = 20s \) and illustrate their properties in Figure 5. The Bernstein coefficients of trajectory \( C^{[3]}(t) \) are

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

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

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

These polynomials are drawn in Figure 5a.

Similar to the 2D examples, Figures 5b, 5c, 5d, and 5e illustrate the end points, convex hull, de Casteljau, and elevation properties, respectively. Figures 5f and 5g 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 5h 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 the arithmetic properties.

III. ALGORITHMS AND FUNCTIONS

This section presents algorithms and procedures for Bernstein polynomials that use the properties listed above.

A. Evaluating Bounds

Using Property 1, one can quickly determine conservative bounds of the Bernstein polynomial. For example, given the 1-dimensional Bernstein polynomial introduced in (1), with coefficients given by

\[
P_5 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 5 & 0 & 2 & 5 & 7 & 5 \end{bmatrix},
\]

lower and upper bounds can be derived using (4). The Bernstein polynomial (blue line), and the coefficients (orange dots connected with dashes) are illustrated in Figure 6. The lower and upper bounds of the Bernstein polynomial are determined by the values of the minimum and maximum coefficients (0 and 7, respectively), while the actual minimum and maximum of the curve are 2.26 and 5.70, respectively (see horizontal red dotted lines). While the bounds are conservative, the Bernstein polynomial can be degree elevated. As discussed earlier (see Property 6 and (12)) the coefficients of a degree elevated Bernstein polynomial converge towards the curve. Thus, they can be used to derive tighter bounds. Figure 6 shows the same Bernstein polynomial elevated to orders 10, 15, and 20. The new estimated minimum and maximum values are 1.93 and 5.89, respectively, for the polynomial elevated to order 20. Degree elevation is performed by multiplying the coefficients of a Bernstein polynomial with the elevation matrix given by (11). A database of elevation matrices can be pre-computed offline to produce tight bounds at a low computational cost.

B. Evaluating Extrema

The extrema of a 1-dimensional Bernstein polynomial are calculated using an iterative procedure similar to the one proposed in [43]. Algorithm 1, which is used to compute the maximum of a Bernstein polynomial, is described with the understanding that a similar procedure can evaluate the minimum value. The input to Algorithm 1 is a set containing the coefficients of the Bernstein polynomial, \( \mathcal{P} = \{P_n\} \), \( P_n = \{P_{0,n}, \ldots, P_{n,n}\} \), an arbitrarily large negative global maximum value \( \alpha \), and a desired tolerance \( \epsilon \).

The algorithm first determines a lower bound on the maximum value. This is done by picking the larger of the two end points since, by the end point property (Property 2), the curve is guaranteed to pass through the two points. Next, the upper bound \( \mathcal{P}_{ub} \) is computed by simply finding the largest Bernstein coefficient. The corresponding index of the upper bound \( i_{ub} \) is also computed for later use. The first 'if' statement replaces the global maximum with the lower bound if the lower bound is larger. This means that the choice of the global maximum value \( \alpha \) should be lower than all other expected values otherwise the user will risk returning an incorrect value.

The second 'if' statement prunes the recursive tree. If \( \alpha \) is greater than the upper bound of the current iteration, there is no reason to continue splitting the current curve since, by the convex hull property (Property 1), no larger value will be found on that curve.

The third 'if' statement checks to see if the upper and lower bounds are within the desired tolerance \( \epsilon \) and returns the current estimate of the global maximum. Otherwise, the process continues by splitting the Bernstein polynomial, using the de Casteljau algorithm (see Property 5), at an intermediate value \( t_{div} = \frac{i_{ub}}{n}(t_f - t_0) + t_0 \). The resulting sets of Bernstein coefficients are \( \mathcal{P}^A \) and \( \mathcal{P}^B \). The algorithm proceeds by making recursive calls for each pair of split curves, until the maximum value is found and returned.

Remark 2: By returning \( \alpha \) in the third 'if' statement, we acknowledge that the true maximum value can be as much as \( \epsilon \) smaller than the estimate. If a conservative estimate is preferable, one should return \( \mathcal{P}_{ub} \) instead.

Algorithm 1 (and its converse) is employed to find the minimum and maximum of the 5th degree Bernstein polynomial depicted in Figure 6 (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 [35].

C. Minimum Spatial Distance

The minimum spatial distance between two Bernstein polynomials can be computed using the method outlined in [43]. This is done by exploiting the Convex Hull (Property
Fig. 3: Several illustrative examples of the properties of Bernstein polynomials being applied to 2D trajectories.
Algorithm 1: Evaluating Maximum for 1-Dimensional Bernstein Polynomial

\begin{verbatim}
Input: \( \mathcal{P}, \alpha, \epsilon \)
1 \( P_0 = \text{lower_bound}(\mathcal{P}) \)
2 \( P_{ub} = \text{max}(\mathcal{P}) \)
3 \( i_{ub} = \text{argmax}(\mathcal{P}) \)
4 if \( P_0 > \alpha \) then
5 \quad \alpha = P_0
6 end
7 if \( \alpha > P_{ub} \) then
8 \quad return \alpha
9 end
10 if \( P_{ub} - P_0 < \epsilon \) then
11 \quad return \alpha
12 else
13 \quad \{A^A, B^B\} = \text{split}(\mathcal{P}, i_{ub})
14 \quad A_{max} = \text{Algorithm 1}(A^A, P_{max}, \epsilon)
15 \quad B_{max} = \text{Algorithm 1}(B^B, P_{max}, \epsilon)
16 \quad P_{max} = \text{max}(A_{max}, B_{max})
17 end
18 return \( P_{max} \)
\end{verbatim}

Algorithm 2: Minimum Distance Between Two Bernstein Polynomials

\begin{verbatim}
Input: \( \mathcal{P}, \mathcal{Q}, \alpha, \epsilon \)
1 \( \text{upper} = \text{upper_bound}(\mathcal{P}, \mathcal{Q}) \)
2 \( \text{lower} = \text{lower_bound}(\mathcal{P}, \mathcal{Q}) \)
3 if \( \text{upper} < \alpha \) then
4 \quad \alpha = \text{upper}
5 end
6 if \( \text{upper} - \text{lower} < \epsilon \) then
7 \quad return \alpha
8 else
9 \quad \{A^A, B^B\} = \text{split}(\mathcal{P})
10 \quad Q^A, Q^B = \text{split}(\mathcal{Q})
11 \quad \alpha = \min(\alpha, \text{Algorithm 2}(A^A, Q^A, \alpha))
12 \quad \alpha = \min(\alpha, \text{Algorithm 2}(A^A, Q^B, \alpha))
13 \quad \alpha = \min(\alpha, \text{Algorithm 2}(B^B, Q^A, \alpha))
14 \quad \alpha = \min(\alpha, \text{Algorithm 2}(B^B, Q^B, \alpha))
15 end
16 return \alpha
\end{verbatim}

Fig. 4: Squared distance between trajectories and the center of the circular obstacle.

D. Collision Detection

The collision detection algorithm is similar, in principle, to the minimum distance algorithm. The difference between the two is instead of having the GJK algorithm return a minimum distance, it simply returns whether a collision has been detected, i.e., two convex hulls intersect. Thus, this algorithm is computationally inexpensive with respect to the minimum distance algorithm, but it returns a binary value (no collision/collision possible) rather than the actual distance between the polynomials.

The collision detection algorithm is outlined in Algorithm 3. The inputs to Algorithm 3 are the sets containing the Bernstein coefficients of the Bernstein polynomials, \( \mathcal{P} = \{P_n\} \) and \( \mathcal{Q} = \{Q_n\} \), and a maximum iteration value, \( \text{max_iter} \). The \text{find_collisions} function returns two sets, \( \mathcal{P}_{col} \) and \( \mathcal{Q}_{col} \), which represent convex hulls of one set that were detected to collide with at least one of the convex hulls in the other set. If no collisions are found, the algorithm returns no collision found. For all the curves where a collision was found, they are split at \( t = \frac{\alpha - t_0}{Q_0} + t_0 \). The previous curve is removed from the set and the new curves from the split are added to the set. If the maximum number of iterations has been met, the algorithm returns that a collision is possible. The implementation of the collision detection algorithm can be found in [35].
(a) Two 3D Bernstein polynomial trajectories near a spherical obstacle. Trajectory $C^{[3]}(t)$ is drawn in blue and trajectory $C^{[4]}(t)$ is drawn in orange.

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

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

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

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

(f) 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.

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

(h) Squared distances between the trajectories and then center of the spherical obstacle.

Fig. 5: Several illustrative examples of the properties of Bernstein polynomials being applied to 3D trajectories.
IV. Numerical Examples

In this section, numerical examples using the toolkit and Python’s Scipy Optimization package are examined (flight tests are available at [45]). The implementation of the following examples can be found in [35].

A. Dubin’s Car - Time Optimal

In this simple example several trajectories for a vehicle with Dubin’s car dynamics are generated to illustrate the properties of Bernstein polynomials. Let the position of the vehicle under consideration be parameterized as a Bernstein polynomial, namely,

$$\begin{align*}
C_n(t) &= \sum_{i=0}^{n} P_{i,n} B_{i,n}(t), \quad t \in [t_0, t_f]. \quad (19)
\end{align*}$$

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

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

The heading angle is

$$\psi(t) = \tan^{-1} \frac{C_n^{[y]}(t)}{C_n^{[x]}(t)},$$

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

$$\omega(t) = \frac{C_n^{[y]}(t) C_n^{[x]}(t) - C_n^{[y]}(t) C_n^{[x]}(t)}{||C_n(t)||^2}.$$

The goal is to compute a time optimal trajectory subject to maximum speed and angular rate bounds, initial and final position, angle, and speed. The vehicle must also maintain a minimum safe distance from two obstacles. This problem can be formulated as follows:

$$\min_{P_{n,f}} t_f$$

subject to

$$C_n(0) = C_0, \quad C_n(t_f) = C_f,$$

$$\psi(0) = \psi_0, \quad \psi(t_f) = \psi_f,$$

$$||\dot{C}_n(0)|| = v_0, \quad ||\dot{C}_n(t_f)|| = v_f,$$

$$||\dot{C}_n(t)|| \leq v_{\text{max}}, \quad \forall t \in [0, t_f]$$

$$||\dot{\psi}(t)|| \leq \omega_{\text{max}}, \quad \forall t \in [0, t_f]$$

$$||C_n(t) - O_i|| \geq d_i^2, \quad \forall t \in [0, t_f], \quad i = 1, 2,$$

where $O_i$ is the position of the $i$th obstacle.

The initial and final positions are $C_0 = [3, 0]^T m$ and $C_f = [7, 10]^T m$, respectively. The initial and final headings are $\psi_0 = \psi_f = \frac{\pi}{2} rad$ (facing toward the positive $Y$ direction). The initial and final speeds are given by $v_0 = v_f = \frac{5}{2} m/s$. The maximum speed and angular rate constraints are $v_{\text{max}} = 5 m/s$ and $\omega_{\text{max}} = 1 rad/s$, respectively.

In the problem above, the order of the Bernstein polynomial, $C_n(t)$, is set to $n = 10$. The initial and final position constraints are enforced using the End Point Values property (Property 2). Similarly, the same property is used to enforce

Algorithm 3: Collision Detection

| Input: $P, Q, max_iter$ |
|--------------------------|
| $k = 0$                  |
| while $k < max_iter$ do |
| \begin{align*} |
| P_{\text{col}}, Q_{\text{col}} &= \text{find_collisions}(P, Q) \\
| \text{if } P_{\text{col}} \cup Q_{\text{col}} = \{\} \text{ then} |
| \text{return } \text{No Collision} |
| \end{align*} |
| end                      |
| for $P_i \in P_{\text{col}}$ do |
| \begin{align*} |
| P^A, P^B &= \text{split}(P_i) \\
| P &= P \cup \{P^A, P^B\} \setminus P_i |
| \end{align*} |
| end                      |
| for $Q_i \in Q_{\text{col}}$ do |
| \begin{align*} |
| Q^A, Q^B &= \text{split}(Q_i) \\
| Q &= Q \cup \{Q^A, Q^B\} \setminus Q_i |
| \end{align*} |
| end                      |
| $k +=$                  |
| return Collision Possible |

Fig. 6: 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.

Fig. 7: (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.

(a)  
(b)
the initial and final speeds and headings (see (5)). Note that the norm squared of the speed and of the distance between the trajectory and the obstacles can be expressed as 1-dimensional 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 7). 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 III-A and III-B.

In Figure 8, the blue curve is obtained by enforcing the constraints using the Evaluating Bounds procedure without degree elevation. The optimal time for the procedure without degree elevation is \( t_f = 9.14s \). The orange and green curves are obtained by enforcing the constraints using the same property after degree elevations of 30 and 100, respectively. Degree elevation to degree 30 results in an optimal final time \( t_f = 7.64s \). The elevation to degree 100 provides an optimal value \( t_f = 7.12s \). Finally, the trajectory with smallest optimal final time, \( t_f = 6.45s \), depicted as the red curve in Figure 8, is obtained by enforcing the constraints using the Evaluating Extrema algorithm (Section III-B).

Figure 9 illustrates the squared speed of each example. As the trajectory is able to pass closer to the obstacle, the maximum speed is increased and in turn lowers the final time. Figure 10 shows the angular rate of each trial. It can be seen that the vehicle correctly adheres to the angular rate constraints for each trial with the only differences being the final time and proximity to the obstacles.

Remark 4: 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 (1) a feasible sub-optimal solution is obtained by enforcing the collision avoidance constraint using the Evaluating Bounds function, and (2) this solution is then used as an initial guess to solve the (more accurate) problem with the Exact Extrema constraint.

B. 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 IV-A. We shall also make the assumption that the trajectories are on a 2D plane rather than on the surface of a sphere.

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:

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

subject to

- \( \mathbf{C}_i^{[k]}(0) = \mathbf{C}_0^{[k]}, \quad \mathbf{C}_n^{[k]}(t_f) = \mathbf{C}_f^{[k]}, \)
- \( \psi_i^{[k]}(0) = \psi_0^{[k]}, \quad \psi_i^{[k]}(t_f) = \psi_f^{[k]}, \)
- \( \| \dot{\mathbf{C}}_n^{[k]}(0) \| = v_0^{[k]}, \quad \| \dot{\mathbf{C}}_n^{[k]}(t_f) \| = v_f^{[k]}, \)
- \( v_{\min}^2 \leq \| \dot{\mathbf{C}}_i^{[k]}(t) \|^2 \leq v_{\max}^2, \quad \forall t \in [0, t_f^{[k]}], \)
- \( \| \dot{\psi}(t) \| \leq \omega_{\max}, \quad \forall t \in [0, t_f^{[k]}], \)
- \( \| \mathbf{C}_n(t) - \mathbf{C}_n(t) \|^2 \geq d_s^2, \quad \forall i, j \in \{1, \ldots, m\}, i \neq j. \)
where the superscript $[k]$ corresponds to the $k$th vehicle out of $m$ vehicles, $C_{0}^{[k]}$ and $C_{f}^{[k]}$ 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, $v_{\text{min}}$ and $v_{\text{max}}$ are the minimum and maximum speeds, $\omega_{\text{max}}$ 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

$$v_{0}^{[k]} = v_{f}^{[k]} = 205 \text{ m/s} \quad \forall k \in \{1, \ldots, m\},$$

the initial headings are $\psi_{0} = [0, \pi, 0, 0]^{\top} \text{ rad}$, the final headings are $\psi_{f} = [0, \pi, -\frac{\pi}{2}, 0]^{\top} \text{ rad}$, the minimum speed is $v_{\text{min}} = 200 \frac{\text{m}}{\text{s}}$, the maximum speed is $v_{\text{max}} = 260 \frac{\text{m}}{\text{s}}$, the maximum angular velocity is $\omega_{\text{max}} = 3 \deg \text{s}^{-1} = 0.0524 \text{ rad/s}$, the minimum safe distance is $d_{s} = 5\text{ km}$, 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 2). Similarly, the same property is used to enforce the initial and final speeds and headings (see (5)). Note that the norm square of the speed and the norm square of the distance between vehicles can be expressed as 1-dimensional 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 7). 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 III-A and III-B.

The optimized flight plans can be seen in Figure 11. The squared speed of each vehicle is shown in Figure 12. Note that each vehicle begins and ends with the same speed. This is due to the initial and final speed constraints of the problem.

The vehicles never slow down less than their initial speeds which means they never reach the minimum speed constraint. Nor do the vehicles go faster than the maximum speed. In Figure 13, 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 14. 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 14 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.

### C. 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 IV-A. 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:

\[
\begin{align*}
\min_{\mathbf{P}} & \sum_{i=1}^{m} \sum_{k=0}^{n-1} ||\mathbf{P}_{i,k}^{[k]} - \mathbf{P}_{i,k+1}^{[k]}|| \\
\text{subject to} & \quad \mathbf{C}_{i}^{[k]}(0) = \mathbf{C}_{i}^{[k]}(0), \quad \mathbf{C}_{i}^{[k]}(t_f) = \mathbf{C}_{i}^{[k]}(0), \\
& \quad ||\mathbf{C}_{i}^{[k]}(0)|| = ||\mathbf{C}_{i}^{[k]}(t_f)|| = v_{i}^{[k]}, \\
& \quad ||\mathbf{C}_{i}^{[k]}(t)||^2 \leq v_{i}^{[k]}^2, \quad \forall t \in [0, t_f], \\
& \quad ||\mathbf{C}_{i}^{[k]}(t) - \mathbf{C}_{j}^{[k]}(t)||^2 \geq d_s^2, \quad \forall i, j \in \{1, \ldots, m\}, i \neq j, \\
& \quad ||\mathbf{C}_{i}^{[k]}(t) - \mathbf{O}_j||^2 \geq d_{obs}^2, \quad \forall t \in [0, t_f], \quad i \in \{1, \ldots, m\}, \\
& \quad j \in \{1, \ldots, b\}.
\end{align*}
\]

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 \(10 m/s\) 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 \(t_f = 30s\), the minimum safe distance between vehicles is \(d_s = 1m\), the minimum safe distance between vehicles and obstacles is \(d_{obs} = 2m\), and the maximum speed is \(v_{max} = 10 m/s\). The vehicles traversing the cluttered environment can be seen in Figure 15. This experiment has been repeated in the Cooperative Autonomous Systems (CAS) lab using three AR 2.0 Drones. The flight tests can be found at [45].

D. 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 3-dimensional Bernstein polynomial, i.e.,

\[
\sum_{i=0}^{n} P_{i,n}^{[i]}(t) = C_{n}^{[i]}(t), \quad \forall j \in \{1, \ldots, m\}.
\]

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 \( n \)th 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:

\[
\min_{P_{i,n}^{[i]}} \sum_{i=1}^{m} \sum_{k=0}^{n-1} \|P_{k+1}^{[i]} - P_{k}^{[i]}\|, \quad \text{subject to}
\]

\[
C_{n}^{[i]}(0) = C_{0}^{[i]}, \quad C_{n}^{[i]}(t_f) = C_{f}^{[i]}, \quad \forall i \in \{1, \ldots, m\},
\]

\[
\|C_{n}^{[i]}(t) - C_{n}^{[j]}(t)\|^2 \geq D_s^2, \quad \forall i, j \in \{1, \ldots, m\}, i \neq j.
\]

The initial positions for each vehicle were chosen randomly from a \( 25m \times 25m \) grid at an altitude of \( z = 0m \). The final positions were chosen to spell out “CAS”, as seen in Figure 16, at an altitude of \( z = 100m \). In the next section we significantly reduce the number of dimensions in the optimization vector by using the decentralized approach.

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

\[
\min_{P_{i,n}^{[i]}} \sum_{i=1}^{m} \sum_{k=0}^{n-1} \|P_{k+1}^{[i]} - P_{k}^{[i]}\|, \quad \text{subject to}
\]

\[
C_{n}^{[i]}(0) = C_{0}^{[i]}, \quad C_{n}^{[i]}(t_f) = C_{f}^{[i]},
\]

\[
\|C_{n}^{[i]}(t) - C_{n}^{[j]}(t)\|^2 \geq D_s^2, \quad \forall j \in \{1, \ldots, i - 1\}, \quad i > 1.
\]

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 16.

Remark 5: While the decentralized method can provide faster computation times, it is not guaranteed to find a solution. For example, if previous trajectories took up too much space, a future trajectory may not have a feasible solution. In the centralized method, this problem would not occur since the optimizer could move all the trajectories in such a way that they all remain feasible. This is a common issue when using decentralized approaches. However, decentralized approaches can offer faster computation times and are robust to certain types of network drops.

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 17 to display the University of Iowa Hawkeye logo. The initial points are equally dispersed at an altitude of \( z = 0m \) on a \( 100m \times 100m \) 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.

\[
\min_{P_{i,n}^{[i]}} \frac{1}{\sum_{j=1}^{i-1} P_{[\text{norm},j]}}, \quad i > 1,
\]

subject to

\[
C_{n}^{[i]}(0) = C_{0}^{[i]}, \quad C_{n}^{[i]}(t_f) = C_{f}^{[i]},
\]

where \( P_{[\text{norm},j]} \) are the Bernstein coefficients of the Bernstein polynomial representing the squared temporal distance between the \( i \)th and \( j \)th trajectories, i.e.

\[
\|C_{n}^{[i]}(t) - C_{n}^{[j]}(t)\|^2 = \sum_{i=0}^{n} p_{[\text{norm},j]} B_{i,n}(t).
\]

It should be noted that this formulation of cost function and constraints is used as a proof of concept but many other possible functions exist. For other possible cost function and constraint formulations, the reader is referred to [46], [47].

V. 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. 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 which avoided obstacles, safely plan commercial flight paths, navigate a team of drones through a cluttered environment, and even generate 1000 trajectories to display a university logo. 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.

REFERENCES

[1] M. Milford, S. Anthony, and W. Scheiter, “Self-driving vehicles: Key technical challenges and progress off the road,” IEEE Potentials, vol. 39, no. 1, pp. 37–45, 2019.
[2] U. R. Mogili and B. Deepak, “Review on application of drone systems in precision agriculture,” Procedia computer science, vol. 133, pp. 502–509, 2018.
[3] R. Koerhuis, “Oddbot robot takes on herbicide free weed elimination,” Dec 2018. [Online]. Available: https://www.futurefarming.com/tools-data/articles/2018/12/oddbot-robot-takes-on-herbicide-free-weed-elimination
[4] E. Ackerman, “Startup developing autonomous delivery robots that travel on sidewalks,” Nov 2015. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/industrial-robots/starship-technologies-australian-ground-delivery-robots
[5] E. Ackerman and M. Koziol, “In the air with zipline’s medical delivery drones,” Apr 2019. [Online]. Available: https://spectrum.ieee.org/robotics/drones/in-the-air-with-ziplines-medical-delivery-drones
[6] F. Lardinois, “A first look at amazon’s new delivery drone,” Jun 2019. [Online]. Available: https://techcrunch.com/2019/06/05/a-first-look-at-amazons-new-delivery-drone/
[7] A. Johnson and G. Hataikoma, “Nasa, ula launch mars 2020 perseverance rover mission to red planet,” Jul 2020. [Online]. Available: https://www.nasa.gov/press-release/nasa-ula-launch-mars-2020-perseverance-rover-mission-to-red-planet
[8] V. J. Lumelsky and A. A. Stepanov, “Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape,” Algorithmica, vol. 2, no. 1-4, pp. 403–430, 1987.
[9] I. Kamon and E. Rivlin, “Sensory-based motion planning with global proofs,” IEEE transactions on Robotics and Automation, vol. 13, no. 6, pp. 814–822, 1997.
[10] K. McGuire, G. de Croon, and K. Turks, “A comparative study of bug algorithms for robot navigation,” CoRR, vol. abs/1808.05080, 2018. [Online]. Available: http://arxiv.org/abs/1808.05080
[11] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using the relative velocity paradigm,” in [1993] Proceedings. IEEE International Conference on Robotics and Automation. IEEE, 1993, pp. 560–565.
[12] Y. Abe and M. Yoshiki, “Collision avoidance method for multiple autonomous mobile agents by implicit cooperation,” in Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180), vol. 3. IEEE, 2001, pp. 1207–1212.
[13] F. Large, S. Sckhavat, Z. Shiller, and C. Laugier, “Using non-linear velocity obstacles to plan motions in a dynamic environment,” in 7th International Conference on Control, Automation, Robotics and Vision, 2002, ICARCV 2002., vol. 2. IEEE, 2002, pp. 734–739.
[14] D. Wilkie, J. Van Den Berg, and D. Manocha, “Generalized velocity obstacles,” in 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2009, pp. 5573–5578.
[15] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in Autonomous robot vehicles. Springer, 1986, pp. 396–404.
[16] Y.-b. Chen, G.-c. Luo, Y.-s. Mei, F.-q. Yu, and X.-j. Su, “Uav path planning using artificial potential field method updated by optimal control theory,” International Journal of Systems Science, vol. 47, no. 6, pp. 1407–1420, 2016.
[17] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, “Mobile robot path planning using membrane evolutionary artificial potential field,” Applied Soft Computing, vol. 77, pp. 236–251, 2019.
[18] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” IEEE transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996.
[19] S. M. LaValle and J. J. Kuffner Jr, “Randomized kinodynamic planning,” The international journal of robotics research, vol. 20, no. 5, pp. 378–400, 2001.
[20] S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning,” Robotics Science and Systems VI, vol. 104, no. 2, 2010.
[21] N. S. Semler and N. Tschichold-Gürmann, “Exact cell decomposition of arrangements used for path planning in robotics,” Technical report/ETH Zurich, Department of Computer Science, vol. 327, 1999.
[22] C. Cai and S. Ferrari, “Information-driven sensor path planning by approximate cell decomposition,” IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 39, no. 3, pp. 672–689, 2009.
[23] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.
[24] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in 2011 IEEE International Conference on Robotics and Automation. IEEE, 2011, pp. 2520–2525.
[25] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization,” in Robtics, science and systems, vol. 9. Citeseer, 2013, pp. 1–10.
[26] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, “Chomp: Gradient optimization techniques for efficient motion planning,” in 2009 IEEE International Conference on Robotics and Automation. IEEE, 2009, pp. 489–494.
[27] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Diller, J. A. Bagnell, and S. S. Srinivasa, “Chomp: Covariant hamiltonian optimization for motion planning,” The International Journal of Robotics Research, vol. 32, no. 9-10, pp. 1164–1193, 2013.
[28] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, “Stomp: Stochastic trajectory optimization for motion planning,” in 2011 ieee international conference on robotics and automation. IEEE, 2011, pp. 4569–4574.
[29] S. Tang, J. Thomas, and V. Kumar, “Hold or take optimal plan (HOOP): A quadratic programming approach to multi-robot trajectory generation,” The International Journal of Robotics Research, vol. 37, no. 9, pp. 1062–1084, 2018.
[30] R. Farouki and T. Goodman, “On the optimal stability of the Bernstein basis,” Mathematics of Computation of the American Mathematical Society, vol. 65, no. 216, pp. 1553–1566, 1996.

[31] V. Cichella, I. Kaminer, C. Walton, and N. Hovakimyan, “Optimal motion planning for differentially flat systems using Bernstein approximation,” IEEE Control Systems Letters, vol. 2, no. 1, pp. 181–186, Jan 2018.

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

[33] V. Cichella, I. Kaminer, C. Walton, N. Hovakimyan, and A. M. Pascoal, “Optimal multi-vehicle motion planning using bernstein approximants,” IEEE Transactions on Automatic Control, 2020.

[34] V. Cichella, I. Kaminer, C. Walton, N. Hovakimyan, and A. M. Pascoal, “Optimal multi-vehicle motion planning using Bernstein approximants,” arXiv preprint arXiv:1812.06132, 2018.

[35] “Bebot,” 2019. [Online]. Available: https://github.com/caslabuiowa/BeBOT

[36] P. Bézier, “Définition numérique des courbes et surface,” Automatisme, vol. 11, no. 4, pp. 625–632, 1966.

[37] P. Bézier, “Définition numérique des courbes et surfaces (ii),” Automatisme, vol. 12, pp. 17–21, 1967.

[38] A. R. Forrest, “Interactive interpolation and approximation by bézier polynomials,” The Computer Journal, vol. 15, no. 1, pp. 71–79, 1972.

[39] R. T. Farouki, “The Bernstein polynomial basis: A centennial retrospective,” Computer Aided Geometric Design, vol. 29, no. 6, pp. 379–419, 2012.

[40] B.-G. Lee and Y. Park, “Distance for bézier curves and degree reduction,” Bulletin of the Australian Mathematical Society, vol. 56, no. 3, pp. 507–515, 1997.

[41] H. Prautzsch and L. Kobbelt, “Convergence of subdivision and degree elevation,” Advances in Computational Mathematics, vol. 2, no. 1, pp. 143–154, Jan 1994. [Online]. Available: https://doi.org/10.1007/BF02519040

[42] G. Farin, “Algorithms for rational bézier curves,” Computer-aided design, vol. 15, no. 2, pp. 73–77, 1983.

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

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

[45] C. Kielas-Jensen and V. Cichella, “Trajectory generation using Bernstein polynomials (Bézier curves),” 2019, accessed: 2019-07-31. [Online]. Available: https://www.youtube.com/watch?v=e0SPt92W578

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

[47] X. Zhang, A. Liniger, and F. Borrelli, “Optimization-based collision avoidance,” IEEE Transactions on Control Systems Technology, 2020.