AIAA AVIATION Forum
June 25-29, 2018, Atlanta, Georgia
2018 Aviation Technology, Integration, and Operations Conference

10.2514/6.2018-3845



# Silhouette-Informed Trajectory Generation through a Wire Maze for Small UAS

Javier Puig-Navarro*, Naira Hovakimyan†
*University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA*

Natalia M. Alexandrov‡ and B. Danette Allen§
*NASA Langley Research Center, Hampton, VA, 23681, USA*

**Current rapidly-exploring random tree (RRT) algorithms rely on proximity query packages that often include collision checkers, tolerance verification, and distance computation algorithms for the generation of safe paths. In this paper, we broaden the information available to the path-planning algorithm by incorporating silhouette information of nearby obstacles in conflict. A silhouette-informed tree (SIT) is generated through the flight-safe region of a wire maze for a single unmanned aerial system (UAS). The silhouette is used to extract local geometric information of nearby obstacles and provide path alternatives around these obstacles, thus focusing the search for the generation of new tree branches near these obstacles and decreasing the number of samples required to explore the narrow corridors within the wire maze. The SIT is then processed to extract a path that connects the initial location of the UAS with the goal, reduce the number of line segments in this path, if possible, and smooth the resulting path using Pythagorean Hodograph Bézier curves. To ensure that the smoothed path remains in the flight-safe region of the configuration space, a tolerance verification algorithm for Bézier curves and convex polytopes in three dimensions is proposed. Lastly, temporal specifications are imposed on the smoothed path in the shape of an arbitrary speed profile.**



## I. Nomenclature

| | | |
|---|---|---|
| $\mathcal{X}$ | = | configuration space |
| $\mathcal{X}_{obs}$ | = | obstacle region of the configuration space |
| $\mathcal{X}_s$ | = | safe region of the configuration space |
| $\mathcal{X}_{us}$ | = | unsafe region of the configuration space |
| $d_s$ | = | safe separation distance |
| $h_s$ | = | safe altitude |
| $\zeta$ | = | nondimensional curve parameter, $\zeta \in [0,1]$ |
| $\tilde{t}$ | = | normalized time, $\tilde{t} \in [0,1]$ |
| $\boldsymbol{p}(\cdot)$ | = | path if expressed as a function of $\zeta$, or trajectory if expressed as a function of time $t$ |
| $\ell(\cdot)$ | = | arclength |
| $\sigma(\cdot)$ | = | parametric speed |
| $v(\cdot)$ | = | desired speed profile |
| $a(\cdot)$ | = | desired acceleration profile |
| $\boldsymbol{p_v}$ | = | point of view |
| $O$ | = | obstacle |
| $\boldsymbol{w_1}$ | = | first vertex of a wire |
| $\boldsymbol{w_2}$ | = | second vertex of a wire |
| $sil$ | = | silhouette |
| $d_e$ | = | distance for the generation of the expanded silhouette |
| $\boldsymbol{x_{rand}}$ | = | random sample in the configuration space |

*PhD Candidate, Department of Aerospace Engineering, 104 S. Wright St., Urbana, IL, AIAA Student Member.
†Professor, Department of Mechanical Engineering, 1206 W. Green St., Urbana, IL, AIAA Fellow.
‡Senior Research Scientist, Aeronautics Systems Analysis Branch, MS 442, AIAA Associate Fellow.
§Senior Technologist for Intelligent Flight Systems, Crew Systems & Aviation Operations Branch, MS 492, AIAA Senior Member.

## II. Introduction

As the capabilities to sense, avoid and plan in complex scenarios advance, autonomous robotic systems transcend the boundaries of industrial, manufacturing and research environments and become increasingly prevalent in everyday life. The ATTRACTOR effort (Autonomy Teaming and TRAjectories for Complex Trusted Operational Reliability) at NASA, a new Convergent Aeronautics Solution project, is specifically interested in missions led by a human commander with an autonomous fleet comprised of multiple aerial assets. Each member of the fleet is equipped with the necessary payload to perform their assigned task such as LiDARs, infra-red cameras, and transceivers. The work presented in this paper aims to develop novel path-planning mechanisms. Through their integration with state-of-the-art sensing, computing and decision-making technologies these algorithms enable the support of challenging mission applications such as search and rescue, wildfire suppression, and urban mobility.

In this work, we propose a new method to sample near the boundary of the flight-safe configuration space and improve the narrow passage behavior of randomly-exploring random trees. The algorithm extracts silhouette information from the obstacle in conflict when a candidate tree branch fails a tolerance verification check. The silhouette is leveraged to extract local geometric information about the environment and generate path alternatives around the obstacle in conflict. As result, a silhouette-informed tree (SIT) is generated through the flight-safe regions in the configuration space. This tree is post-processed to extract a path that connects the initial point of the UAS with the goal point. Finally, the path is smoothed using Pythagorean Hodgraph Bézier curves that still lie in the flight-safe configuration space. To this end, a tolerance verification algorithm for Bézier curves and convex polytopes is developed and presented.

Previously, rapidly-exploring random trees (RRT) were initially developed as a single-query method that can quickly expand through the configuration space [1]. More recently, a modified version of the algorithm with asymptotic optimality guarantees, RRT$^*$, was proposed [2]. However, these methods tend to require large numbers of samples to identify paths through narrow corridors. Also, convergence to the optimal solution in [2] requires exhaustive sampling of the configuration space. As a result, numerous efforts have pursued the derivation of heuristic techniques that reduce the number of samples without degrading the quality of the path produced.

The work in [3] utilized potential functions to guide the sampling towards the regions of interest, reduce memory utilization, and increase the convergence rate towards the optimal solution. Researchers in [4] combined RRT$^*$ with a linear quadratic regulator (LQR) for the linearized dynamics of an under-actuated system. In this case, the LQR was used to automatically define a domain-specific distance method and the corresponding node extension mechanism. The authors in [5] guided the growth of the tree towards lower-cost regions in the configuration space through biased sampling. The work in [6] introduced two methods: *i*) local biasing to improve convergence to the optimal solution, only activated after a solution is found, and *ii*) node rejection to reduce the number of nodes in the tree and improve efficiency. In this case, the algorithm rejects nodes that cannot improve the cost of the solution. Finally, they combined these heuristics with RRT$^*$ [2] and RRT-connect [7]. In summary, a significant part of the heuristic efforts in rapidly-exploring random trees focuses on improving the rate of convergence to the optimal solution. On the other hand, the work in [8] increased the probability of sampling in a narrow corridor by borrowing tools initially developed for probabilistic road maps (PRM) [9]. The authors in [10] suggested the use of a penetration-depth algorithm to sample on the boundary of the obstacle-free region of the configuration space, while the algorithm in [11] analyzes the topology of the configuration space and creates a graph through the obstacle-free region. Then, this graph is leveraged to sample near the regions of interest from a topological viewpoint.

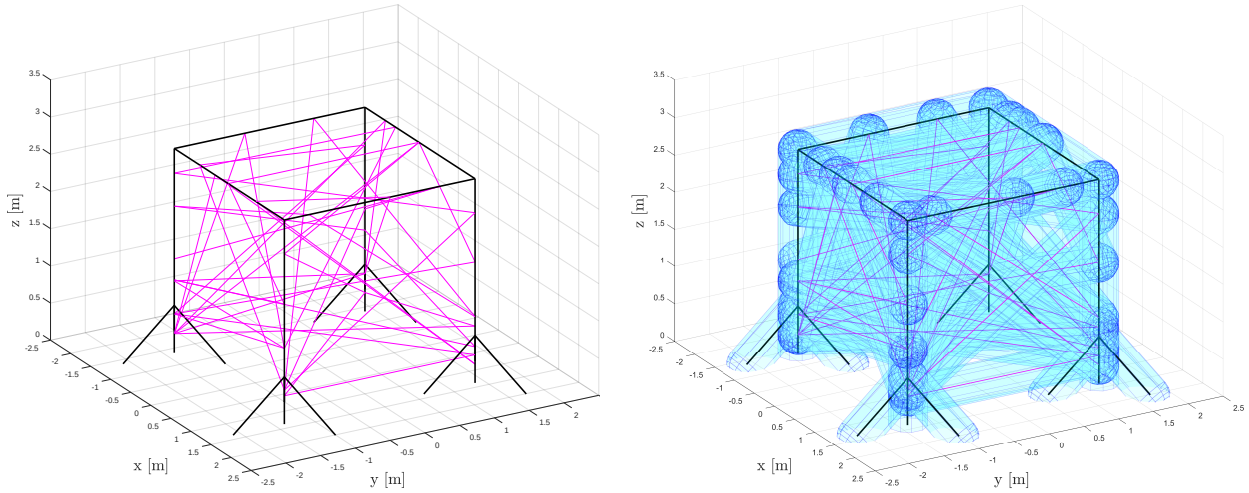
The next section describes the wire maze, the challenges it poses for trajectory generation, and potential benefits of testing and stressing trajectory generation algorithms through cluttered environments. Section IV formally defines the problem addressed in this paper. Section V provides a brief overview of Bézier curves, their properties, and the Pythagorean Hodograph condition. Section VI details how to compute, expand, and sample the silhouette of a wire to inform the path-planning algorithm. Section VII presents a tolerance verification algorithm for Bézier curves and convex polytopes in three dimensions, necessary to check whether the smoothed path lies in the flight-safe region of the configuration space. Section VIII unifies all the tools presented in previous sections to generate a trajectory through the wire maze. Section IX provides simulation results of the trajectory generation algorithm. The paper ends with a summary of the algorithms developed, trends observed from the simulation results, and potential benefits of informing other algorithms with the silhouette of nearby obstacles.

## III. Motivational Scenario

Efficient path-planning through cluttered environments has spurred the development of a diverse pool of algorithms with applications in manipulator robotics [12–14], mobile robotics [15–18], self-driving vehicles [19, 20], flight management, control of the national airspace [21–24], and even autonomous design of ducts and pipes [25], to name a few examples. Autonomous navigation of small UAS through cluttered environments still poses significant challenges in sensing, computing, systems integration, and decision making [26, 27]. Nonetheless, the solution to this problem has the potential to revolutionize disciplines such as, urban mobility, logistics, mail and package delivery systems, surveillance and monitoring, search and rescue missions, and wild fire suppression.

Motivated by the need of a thorough validation and verificaiton tool, we decided to build a test bed to develop and stress path-planning algorithms, both in a simulation and a laboratory environment [28]. As a result, the wire maze depicted in Figure 1a was first modeled in simulation, and then built at the Autonomy Incubator at NASA Langley Research Center. This platform was chosen as a cost effective solution due to its portability, ease of reconfiguration, and the capability to create complex scenarios that exemplify some of the challenges in path planning and sensing technologies. The main structure of the wire maze consists of 8 struts, and 12 legs that hold the vertical struts. The structural elements of the wire maze are depicted in Figure 1a as black rods. In addition, 43 wires were randomly placed on the structure to complete the wire maze, shown in magenta in Figure 1a. This defines a total of 63 obstacles. Moreover, the majority of the wires span the configuration space from one extreme to the opposite, which makes it particularly difficult to partition the flight area in separable regions where only a set of obstacles is present. Consequently, the use of typically efficient data structures – such as octrees [29, 30] – that compartmentalize the configuration space and reduce the computational workload is not significantly beneficial in this scenario. Also, the number of wires and close proximity among them defines a large number of narrow tortuous corridors the vehicle could fly through.

To ensure vehicles fly safely through the wire maze, we define a safe separation distance $d_s$ that the UAS shall maintain with all obstacles. The safe separation distance not only accounts for possible errors that alter the desired relative position between the UAS and the maze – position errors, path-following errors, or even manufacturing tolerances in the wire maze – but also serves as a parameter to tune the challenge of the path-planning problem. Indeed, as the safety distance increases, the size of the narrow passages within the wire maze decreases. Figure 1b shows a set of capped cylinders in cyan that represent the unsafe region around each wire. In this case, the safe separation distance was set to 0.25 $m$. The capped cylinders are included for visualization purposes, so that the reader can estimate the difficulty of the problem. However, none of the algorithms developed in this paper require explicit computation of such objects around the wires. In the remainder of this paper, the fundamental tools and methods used to create a path through the wire maze are discussed.




**(a) 3D view of the wire maze.**

**(b) 3D view of the wire maze with the unsafe region around each wire.**

**Fig. 1 Wire maze.**

## IV. Problem Formulation

Given a configuration set $\mathcal{X} \in \mathbb{R}^3$, a set $\mathcal{X}_{obs} = \bigcup_{i=1}^{n_o} O_i$ defining the obstacle region, and a constant safety distance $d_s$, where $O_i$ represents the $i$th obstacle and $n_o$ is the total number of obstacles, we define the unsafe region of the configuration space as

$$\mathcal{X}_{us} := \{\boldsymbol{x} \in \mathcal{X} \mid \|\boldsymbol{x} - \boldsymbol{y}\| < d_s,\ \forall \boldsymbol{y} \in \mathcal{X}_{obs}\},$$

where $\|\cdot\|$ denotes the Euclidean norm. In addition, the flight-safe region is defined as $\mathcal{X}_s := \mathcal{X} \setminus \mathcal{X}_{us}$. Then, the trajectory generation problem is to define a parametric curve as a function of time $\boldsymbol{p} : t \mapsto \mathcal{X}_s$ that takes the UAS from the initial point $\boldsymbol{x_{init}}$ to the goal point $\boldsymbol{x_{goal}}$, subject to additional spatial and temporal constraints detailed in the following subsections. Inspired by the work in [31–34], we aim to decouple the spatial component of the problem from the temporal assignments. As a result, we divide the trajectory generation problem into two sub-problems: path planning and design of temporal specifications. This approach was first utilized in [35] and lets us adjust the spatial paths and speed profiles independently.



### A. Path planning

The path-planning problem is exclusively defined by the spatial specifications of the trajectory such as, initial and final position, the flight-safe region of the configuration space, and any additional requirements. Some illustrative examples are initial and final directions for the path, curvature, or altitude constraints. For the purpose of this paper, the objective of the path-planning problem is to generate a path $\boldsymbol{p}(\zeta) : [0, 1] \to \mathbb{R}^3$, where $\zeta$ is the nondimensional curve parameter, such that the path:

1) starts and finishes at the designated initial and final points: $\boldsymbol{p}(0) = \boldsymbol{x_{init}}$, $\boldsymbol{p}(1) = \boldsymbol{x_{goal}}$;
2) lies entirely in the flight-safe space: $\boldsymbol{p}(\zeta) \in \mathcal{X}_s$, $\forall \zeta \in [0, 1]$;
3) is always above a certain altitude $h_s$ considered safe: $p_z(\zeta) \geq h_s$, $\forall \zeta \in [0, 1]$, where $p_z(\cdot)$ denotes the altitude of the path.

The objective of the path-planning problem is to generate a single path that satisfies the conditions above. This is typically referred to as a single-query path-planning problem [36, 37]. The main focus of this research is not necessarily finding the optimal solution, but rapidly finding a solution through a complex scenario that takes the UAS from the start point to its assigned goal. In this case, path optimality is often traded for computational efficiency. In this regard, there exist algorithms with asymptotic optimality guarantees [2, 38, 39], where the probability of finding the optimal solution approaches 100% as the algorithm increases the number of samples in the configuration space. The path-planning approach developed in this paper is significantly influenced by some of these algorithms.

### B. Temporal Specifications

The speed design problem defines the temporal evolution of the UAS along its assigned path. Hence, it requires the path definition as well as time-related constraints such as, desired mission time, bounds on the speed and acceleration profiles, or even the rate of change in altitude. As a result, the speed design problem is only tackled after a solution for the path-planning problem has been found. The design of a speed profile is beyond the scope of this paper. However, to determine the temporal evolution of the position of the UAS, we assume a speed profile $v(\tilde{t})$ is given as a function of the normalized time

$$\tilde{t} = \frac{t - t_{init}}{t_{goal} - t_{init}},$$

where $t$ is the mission time, $t_{init}$ is the start time of the mission, and $t_{goal}$ is the desired time of arrival to $\boldsymbol{x_{goal}}$. For the problem at hand, we wish to impose the following temporal specifications:

1) the desired speed of the UAS should be able to follow an arbitrary speed profile $v(\tilde{t})$ with a polynomial structure;
2) the trajectory $\boldsymbol{p}(t)$ shall be twice continuously differentiable with respect to time: $\boldsymbol{p}(t) \in C^2$.

The output of the speed design problem produces two pieces of information of relevance to the execution of the mission. First, the speed profile of the UAS

$$v(t) = \left\| \frac{\mathrm{d}\boldsymbol{p}(t)}{\mathrm{d}t} \right\| = \|\dot{\boldsymbol{p}}(t)\|,$$

which is often used by path-following algorithms [40–42]; and second the temporal evolution of the curve parameter $\zeta(t)$, which is a useful tool to enforce temporal separation among multiple vehicles [43].

The solution we propose generates a RRT through the flight-safe region of the configuration space. Some of the branches in this tree are generated with the aid of the silhouette information extracted from neighboring obstacles. This results in a set of line segments that connect the initial point $\boldsymbol{x_{init}}$ with the goal point $\boldsymbol{x_{goal}}$. However, this sequence of line segments does not satisfy the continuity requirements defined. Therefore, the tree is post-processed to reduce the number of segments and smooth the result. The smoothing technique, that will be described later, must guarantee that the path still lies entirely in the flight-safe region of the configuration space. To this end, a family of smooth curves and associated algorithms, as well as a set of geometric queries – such as silhouette or tolerance verification methods – have been selected. The following sections provide further details on the curves and methods selected, and some of the properties and algorithms that support these choices.

## V. Pythagorean Hodograph Bézier Curves

The curves selected for the final path are Pythagorean Hodographs (PH) expressed in a Bernstein basis, commonly referred to as PH Bézier curves. This type of polynomial curve and the basis selected were chosen for a wide range of considerations such as, optimality, numerical stability, favorable geometric properties, and availability of efficient proximity-query and root-finding algorithms, to mention but a few examples. The Bernstein basis was first introduced by Russian mathematician Sergei N. Bernstein in 1912 to develop a proof of the Weierstrass theorem [44]. Later, French engineer Pierre Bézier started generating curves and surfaces with this basis in the 1960s to design cars. Since then, Bézier curves have become ubiquitous in CAD design, gaming and animation, and even digital calligraphy for their advantageous properties [45]. This section provides a brief description of Bézier curves, the PH condition, and some of the properties that will later be exploited for trajectory generation.

### A. Bézier Curves

A spatial Bézier curve of degree $n$ is a map $\boldsymbol{p}(\zeta) : [0,1] \rightarrow \mathbb{R}^3$ defined by the following expression:

$$\boldsymbol{p}(\zeta) = \sum_{k=0}^{n} \boldsymbol{p_k}\, b_k^n(\zeta),$$

where $\zeta$ is the dimensionless curve parameter, $\boldsymbol{p_k} \in \mathbb{R}^3$ is the $k$th control point, and $b_k^n(\zeta)$ is a Bernstein basis polynomial given by

$$b_k^n(\zeta) = \binom{n}{k}(1-\zeta)^{n-k}\zeta^k\,.$$

Bézier curves are polynomial curves with no additional structure, and are fully defined by their control points. Figure 2 shows an example of a spatial Bézier curve and its control points, where some of the properties that are of interest for trajectory generation can be observed:

i) A Bézier curve starts at its first control point, and finishes at its last control point.

ii) The parametric derivatives $\boldsymbol{p}'(\zeta)$ of the curve evaluated at the start and the end points are vectors with the same direction as the line segments defined by the first and second control points, and the last and second to last control points, respectively:

$$\boldsymbol{p}'(0) = n\,(\boldsymbol{p_1} - \boldsymbol{p_0})\,, \qquad \boldsymbol{p}'(1) = n\,(\boldsymbol{p_n} - \boldsymbol{p_{n-1}})\,.$$

This is depicted in Figure 2 by the black arrows, and provides a convenient mechanism to enforce the initial and final direction of a curve if needed.

iii) A Bézier curve lies within the convex hull of its control points. Therefore, the convex hull naturally defines a bounding region where the curve is contained in its entirety. This property is often leveraged to perform efficient proximity queries with Bézier curves [46]. Figure 2 provides visual confirmation that the curve is fully enclosed in the convex hull of the control points, represented by the magenta polyhedron.

iv) A Bézier curve can be modified intuitively by moving its control points. Therefore, curves expressed in a Bernstein basis are especially amenable for manipulation by a human operator.

v) The Bernstein basis is "*optimally stable*" [47]. In other words, it is not possible to design a non-negative basis* for generic polynomials of degree $n$ that will consistently reduce the condition number† of this basis.

*A basis $\{\phi_0^n(\zeta), \ldots, \phi_n^n(\zeta)\}$ for polynomials of degree $n$ is non-negative on $\zeta \in [a,b]$ if $\phi_k^n(\zeta) \geq 0\,,\ \forall \zeta \in [a,b]$ and $\forall k \in \{0,\ldots,n\}$.

†A basis is said to be ill-conditioned for a particular polynominal if the condition number is large.

Consequently, Bézier curves present excellent numeric stability, and floating-point error propagation behavior.

vi) The variation-diminishing property states that, given a Bézier curve $r(\zeta) : [0,1] \rightarrow \mathbb{R}$ with Bernstein coefficients $r_k$

$$r(\zeta) = \sum_{k=0}^{n} r_k b_k^n(\zeta),$$

the number of real roots $n_r$ of $r(\zeta)$, $\forall \zeta \in (0,1)$, is less than or equal to the number of sign changes $n_c$ in the Bernstein coefficients by an even amount

$$n_r = n_c - 2m,$$

where $m \in \mathbb{Z}^+$. Therefore, if $n_c = 1$ then $n_r = 1$. The combination of this property with de Casteljau's algorithm is a powerful tool for fast isolation and approximation of real roots for $n$-degree polynomials [45, 48]. This property can be leveraged to check whether the altitude of a path remains above a pre-specified safety altitude $h_s$.

vii) The set of Bézier curves – polynomial curves – is closed under the addition, subtraction, differentiation, integration and composition, which makes it easier to operate with them.

In addition, there is an ample repertoire of algorithms for curve subdivision, evaluation, and manipulation [45, 49, 50] that can be used in conjunction with the properties above to generate trajectories that avoid collision with nearby obstacles, satisfy dynamics constraints, and meet desired temporal specifications.

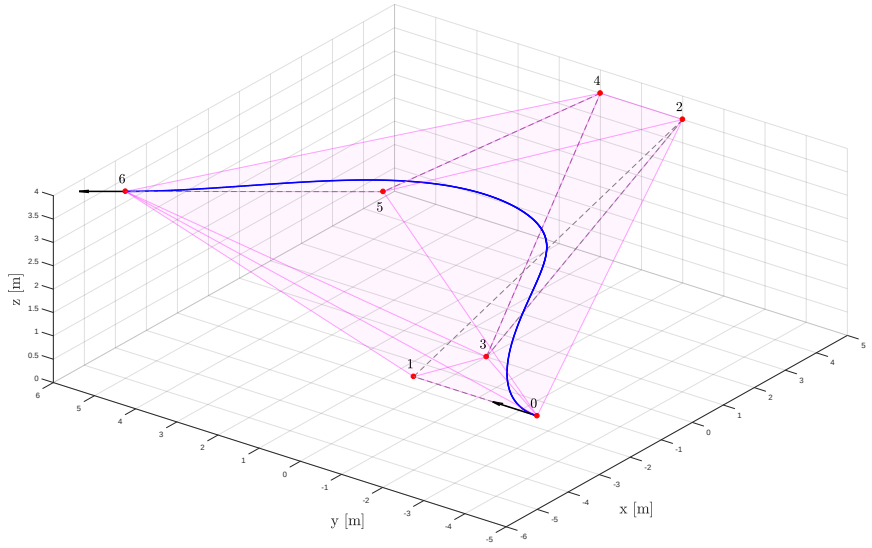



**Fig. 2 Bézier curve depicted in blue, control points shown in red, and convex hull of the control points represented in magenta.**

### B. Pythagorean Hodograph Curves

The hodograph of a curve $\boldsymbol{p}(\zeta)$ is defined as its parametric derivative $\boldsymbol{p}'(\zeta)$. For a spatial curve, this can be written as:

$$\boldsymbol{p}'(\zeta) = \frac{\mathrm{d}\boldsymbol{p}(\zeta)}{\mathrm{d}\zeta} = \left[p_x'(\zeta), p_y'(\zeta), p_z'(\zeta)\right]^\top,$$

where $p_x$, $p_y$, and $p_z$ represent the $x$, $y$, and $z$ coordinates of the path, respectively. Hence, a spatial PH Bézier curve of degree $n$ is a map $\boldsymbol{p}(\zeta) : [0,1] \rightarrow \mathbb{R}^3$ whose hodograph satisfies the Pythagorean condition:

$$(\boldsymbol{p}'(\zeta))^\top \boldsymbol{p}'(\zeta) = \sigma^2(\zeta),$$

where $\sigma(\zeta) : [0,1] \rightarrow \mathbb{R}$ has a polynomial expression, that can be written in a Bernstein basis as

$$\sigma(\zeta) = \sum_{k=0}^{n-1} \sigma_k b_k^{n-1}(\zeta).$$

In the context of a spatial path, $\sigma(\zeta)$ represents the parametric speed of a curve, and $\sigma_k$ are its control points. The integration of the parametric speed yields the arc length

$$\ell(\zeta) = \int_0^\zeta \sigma(\epsilon)\,\mathrm{d}\epsilon = \sum_{k=0}^{n} \ell_k b_k^n(\zeta),$$

where $\ell_0 = 0$ and $\ell_k = \frac{1}{n}\sum_{j=0}^{k-1}\sigma_j$. Hence, the total length of a PH Bézier curve of degree $n$ is

$$L = \frac{1}{n}\sum_{k=0}^{n-1}\sigma_k\,. \tag{1}$$

As opposed to general Bézier curves, the arclength of a PH Bézier has a closed-form expression. Since the arclength is often one of the parameters considered for the evaluation and optimization of a curve, PH Bézier curves were selected to avoid approximate numerical integration of the arclength, thus decreasing the computational load and increasing the accuracy of the arc-length computations. There are numerous bibliographic methods to generate piecewise PH Bézier curves for different continuity requirements. In particular, we highlight [51, 52] for $C^1-$continuity, and [53, 54] for $C^2-$continuity. For the purposes of this paper, we will use the algorithm in [53], which yields a sequence PH Bézier curves of degree 9 with $C^2-$continuity

$$\boldsymbol{p}(\zeta) = \sum_{k=0}^{9} \boldsymbol{p_k}\, b_k^n(\zeta),$$

where the control points $\boldsymbol{p_k} \in \mathbb{R}^3$ are a function of the initial and final position, the initial and final values of the hodograph, the initial and final values of the second derivative of the path, and a set of free parametes that are chosen using the heuristics described in [53]. Having described the fundamental tools that define the path, the generation of these curves is informed with local geometric information about the environment through the silhouette of nearby obstacles in the following section.

## VI. Silhouette

The silhouette of an obstacle $O$ from a point of view $\boldsymbol{p_v} \notin cl(O)$, where $cl(\cdot)$ represents the closure of a set, is defined as the contour of such object as seen from point $\boldsymbol{p_v}$. For a convex polyhedron, the silhouette is a closed sequence of edges that describes the boundary between the facets that are visible from $\boldsymbol{p}_v$ and the ones that are not [55]. Consequently, the silhouette contains local geometric information that can be used for simulation and path-planning purposes. In simulation, the silhouette can be exploited to determine the region of an obstacle that is visible for a UAS at a specific location in the configuration space, as depicted in Figure 3. For path planning, however, the silhouette contains local information on how to avoid a particular obstacle in space. Note that in Figure 3 if a vehicle were to fly in a straight line along a direction contained in the convex hull of all vectors $\boldsymbol{u_{v,i}}$, $i \in \{1, ..., n_v\}$, where $n_v$ is the number of vertices in the silhouette, then the UAS would eventually collide with the obstacle $O$. In this section, we describe a method to compute the silhouette of the unsafe region around a wire using the coordinates of the extreme points of the wire $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$, the point of view $\boldsymbol{p_v}$, and the safety distance $d_s$. In addition, we provide general guidelines on how to leverage this information for path-planning purposes.

### A. Silhouette Computation

The unsafe region around a wire consists of a cylinder of radius $d_s$ and two hemispheres of the same radius attached to either end of the cylinder, see Figure 4. Henceforth, these hemispheres will be referred to as caps. The first step of the silhouette computation is to determine what regions are visible from $\boldsymbol{p_v}$. To this end, we define the length of the wire $d_w$, a unit vector with the direction of the wire $\boldsymbol{u_w}$, and a vector $\boldsymbol{v_{v,1}}$:

$$\boldsymbol{u_w} := \frac{\boldsymbol{w_2} - \boldsymbol{w_1}}{d_w}\,, \qquad \boldsymbol{v_{v,1}} := \boldsymbol{w_1} - \boldsymbol{p_v}\,.$$

There are only two options regarding the visibility of the different components of the capped cylinder, depending on the distance $d_{v,\ell}$ between the line defined by $\boldsymbol{w_1}$ and $\boldsymbol{u_w}$, and the point of view $\boldsymbol{p_v}$ (line 3, Algorithm 1)[‡]:

[‡]The function distLine2Point computes the distance between an infinite line and a point. For the sake of brevity, and due to its simplicity no further details are provided.

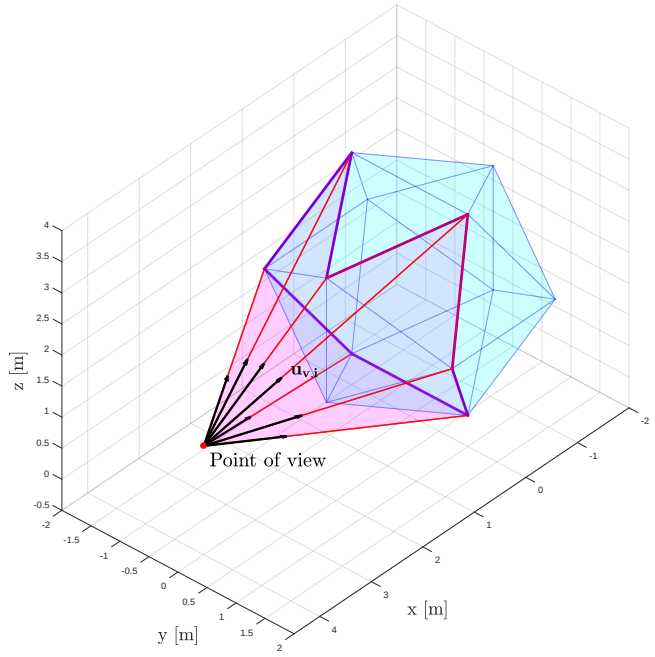



**Fig. 3 Silhouette shown in purple of an obstacle depicted in cyan, as seen from the point of view. Unit vectors with origin at $p_v$ that point towards each of the vertices of the silhouette are shown in black. The object in magenta represents the positive linear combination of all vectors $u_{v,i}$ .**

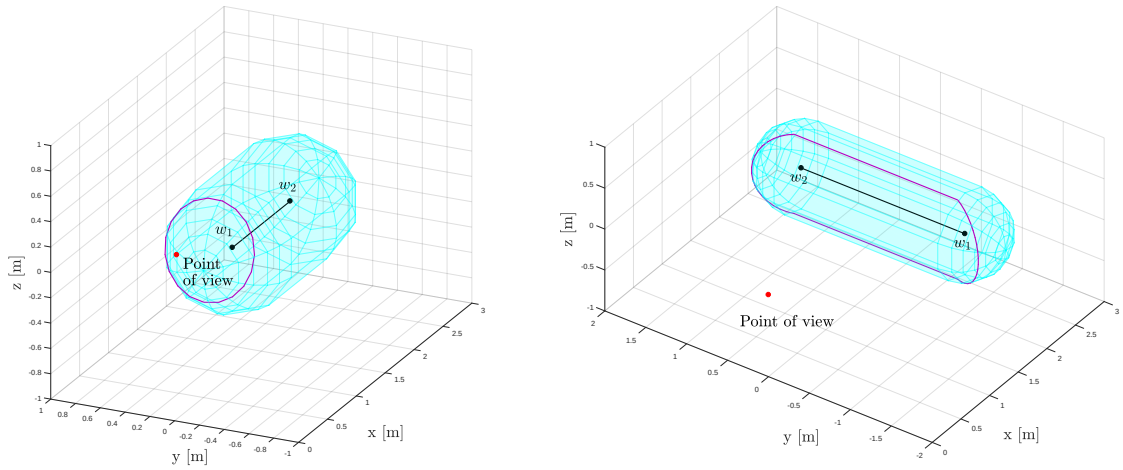
i) If $d_{v,\ell} \leq d_s$, then only part of a cap is visible (line 4, Algorithm 1). The resulting silhouette is a circumference, as shown in Figure 4a. Next, the sign of the projection of $\boldsymbol{v_{v,1}}$ onto $\boldsymbol{u_w}$ determines which cap is visible from $\boldsymbol{p_v}$ (lines 6 and 8). Lastly, to determine the center $\boldsymbol{c_{sil}}$, radius $r_{sil}$, and normal vector of the circumference $\boldsymbol{n_{sil}}$, we resort to basic geometry:

$$d_{v,i} = \|\boldsymbol{p_v} - \boldsymbol{w_i}\|, \quad \boldsymbol{n_{sil}} = \frac{\boldsymbol{p_v} - \boldsymbol{w_i}}{d_{v,i}}, \quad \boldsymbol{c_{sil}} = \boldsymbol{w_i} + \frac{d_s^2}{d_{v,i}}\boldsymbol{n_{sil}}, \quad r_{sil} = \frac{d_s}{d_{v,i}}\sqrt{d_{v,i}^2 - d_s^2}, \tag{2}$$

where $i$ is either 1 or 2, depending on which cap is seen from $\boldsymbol{p_v}$. This fully defines the silhouette for this case, and describes the content of the function onlyCapSilhouette in lines 7 and 9.

ii) If $d_{v,\ell} > d_s$, then a section of the cylinder and regions of the two caps are visible from $\boldsymbol{p_v}$ simultaneously (line 11 in Algorithm 1). In this case, the silhouette consists of two straight segments that correspond to the cylinder, and two circular arcs that result from the caps, all connected to form a closed curve as shown in Figure 4b. The centers, radii, and normal vectors that partially define the circular arcs are calculated as in Equation 2. Furthermore, the intersection of the planes that contain the circular arcs and the end planes of the cylinder define a line. This line intersects the caps at the points where the circular arcs start and end. As a result, these points can be easily computed as the intersection of a line and a sphere of radius $d_s$ and center $\boldsymbol{w_i}$. This fully defines the silhouette for this case, and describes the content of the function capAndCylinderSilhouette in line 12.

Once the silhouette has been computed, the question of how this information can be leveraged for path planning still remains unanswered. To this purpose, we will expand the silhouette into a surface perpendicular to the boundary of the unsafe region. Figure 5 shows in purple the surface that results from extruding the silhouette. This provides a region of interest around each wire that contains local geometric information on how to avoid the obstacle. Next, random samples that do not violate safety distance constraints with other obstacles will be picked from the purple surface, shown in blue in Figure 5. This focuses the path-planning search closer to the possible narrow passages that exist in the surroundings of each wire, and hence increases the probability of finding paths through the small but geometrically relevant Voronoi regions in the narrow corridors. The next section contains the details on how to expand the silhouette and sample the resulting surface.




**(a) The visible region from the point of view is a section of the cap.**

**(b) The visible region from the point of view includes parts of the two caps and a section of the cylinder.**

**Fig. 4 Wire in black, unsafe area around it in cyan, and silhouette of the unsafe region in purple.**

**Algorithm 1:** Silhouette computation for the unsafe region around a wire.

**function** wireSilhouette($\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{p_v}, d_s$);
**Input** : $\boldsymbol{w_1}$ first vertex of the wire
$\boldsymbol{w_2}$ second vertex of the wire
$\boldsymbol{p_v}$ point of view
$d_s$ safe separation distance
**Output** : $sil$ silhouette
$\boldsymbol{u_w} \leftarrow \boldsymbol{w_2} - \boldsymbol{w_1}$; $d_w \leftarrow \sqrt{\boldsymbol{u_w}^\top \boldsymbol{u_w}}$; $\boldsymbol{u_w} \leftarrow \boldsymbol{u_w}/d_w$;
$d_{v,\ell} \leftarrow$ distLine2Point($\boldsymbol{w_1}, \boldsymbol{u_w}, \boldsymbol{p_v}$) ; ▷ Distance between a line defined by $\boldsymbol{w_1}$ and $\boldsymbol{u_w}$, and a point $\boldsymbol{p_v}$.
**if** $d_{v,\ell} \leq d_s$ **then**
  $\boldsymbol{v_{v,1}} \leftarrow \boldsymbol{w_1} - \boldsymbol{p_v}$;
  **if** $\boldsymbol{v_{v,1}}^\top \boldsymbol{u_w} > 0$ **then**
    $sil \leftarrow$ onlyCapSilhouette($\boldsymbol{p_v}, \boldsymbol{w_2}, d_s$); ▷ Cap with center in $\boldsymbol{w_2}$ is visible.
  **else**
    $sil \leftarrow$ onlyCapSilhouette($\boldsymbol{p_v}, \boldsymbol{w_1}, d_s$); ▷ Cap with center in $\boldsymbol{w_1}$ is visible.
  **end**
**else**
  $sil \leftarrow$ capAndCylinderSilhouette($\boldsymbol{p_v}, \boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{u_w}, d_w, d_s$); ▷ Caps and cylinder are visible.
**end**
**return** $sil$

### B. Silhouette Expansion and Sampling

To expand the silhouette, we define the expanding distance $d_e$. This value is a measure of how far around each obstacle in conflict one is willing to explore to provide informed path alternatives that avoid the obstacle. Once again, there are only two options regarding the expansion of the silhouette, depending on the sections visible from $\boldsymbol{p_v}$:

i) If only a portion of a cap is visible from $\boldsymbol{p_v}$, then the expanded silhouette is a conical frustum as shown in Figure 5a. The apex of the corresponding cone is $\boldsymbol{w_i}$, the axis of symmetry of the frustum is $\boldsymbol{n_{sil}}$, and the slanted height of the frustum is $d_e$. Once the silhouette expansion is complete, we sample over the surface of the frustum and discard all the samples that do not meet safe separation constraints with other obstacles. To avoid biasing the solution towards any specific region in space, the lateral surface of the frustum is sampled uniformly. To this end, we follow the method described in [56]. First, we define a Cartesian coordinate system $\{\boldsymbol{o_c}, \boldsymbol{e_x}, \boldsymbol{e_y}, \boldsymbol{e_z}\}$. The center of the coordinate system is $\boldsymbol{o_c} = \boldsymbol{w_i}$, and its $z$ axis coincides with the axis of symmetry of the frustum $\boldsymbol{e_z} = \boldsymbol{n_{sil}}$. The direction of $\boldsymbol{e_x}$ and $\boldsymbol{e_y}$ is irrelevant, as long as they are orthonormal

and define a right-handed coordinate system. Next, we introduce the height of the cones defined by the apex and the silhouette $h_{sil}$, and the apex and the expanded silhouette $h_{esil}$, as well as the radius of the expanded silhouette $r_{esil}$:

$$h_{sil} = \frac{d_s^2}{d_{v,i}}, \qquad h_{esil} = (d_s + d_e)\frac{d_s}{d_{v,i}}, \qquad r_{esil} = \frac{d_s + d_e}{d_{v,i}}\sqrt{d_{v,i}^2 - d_s^2}$$

where $i$ is either 1 or 2, depending on which cap is seen from $\boldsymbol{p_v}$. Then, we define variables $\theta$ and $\chi$, and sample them uniformly over the intervals

$$\theta \sim \mathcal{U}(0, 2\pi), \qquad \chi \sim \mathcal{U}(0, 1),$$

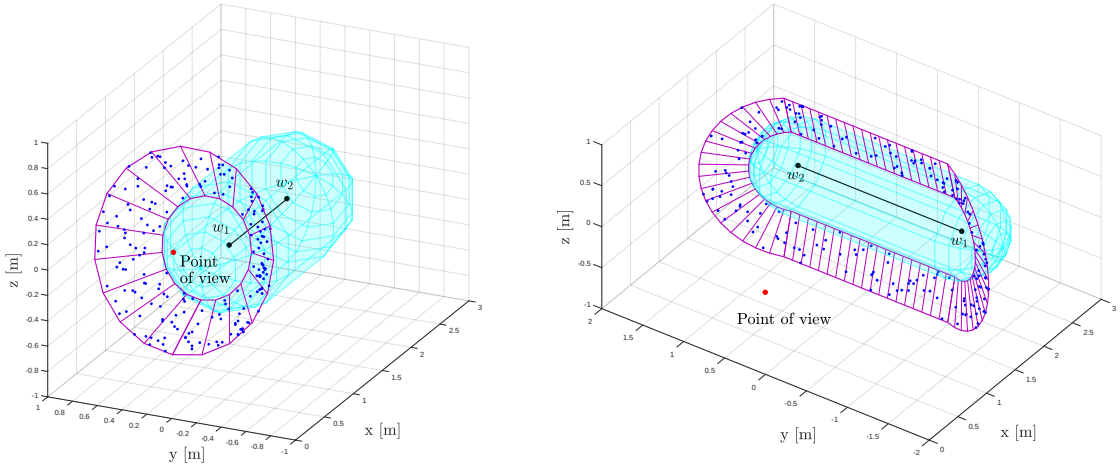
and compute the height and radius of the cylindrical coordinates of a single sampled point in the coordinate system defined above as

$$h = \sqrt{h_{esil}^2 - \chi\,(h_{esil}^2 - h_{sil}^2)}, \qquad r = r_{sil} + (r_{esil} - r_{sil})\frac{h - h_{sil}}{h_{esil} - h_{sil}}.$$

Finally, the point sampled randomly over the surface of the frustum with uniform probability density can be obtained as

$$\boldsymbol{x_{rand}} = \boldsymbol{o_c} + h\,\boldsymbol{e_z} + r\left(\boldsymbol{e_x}\cos(\theta) + \boldsymbol{e_y}\sin(\theta)\right).$$

A batch of 200 points was sampled following this method and is shown in Figure 5a.




**(a) The visible region from the point of view is a section of the cap.**

**(b) The visible region from the point of view includes parts of the two caps and a section of the cylinder.**

**Fig. 5 Wire in black, unsafe area around it in cyan, silhouette and expanded silhouette in purple, and sampled points on the expanded silhouette in blue.**

ii) If parts of the two caps and a portion of the cylinder are visible from $\boldsymbol{p_v}$, then the expanded silhouette is composed of two conical patches and two rectangular surfaces, as depicted in Figure 5b. The apex, axis of symmetry, and slanted height of the conical patches are defined as in the previous section. As for the rectangular surfaces, their length and width is $d_w$ and $d_e$, respectively. To avoid biasing the path-planning algorithm towards any particular region of the space, we sample this surface with uniform probability density. To this end, we ensure that the probability of sampling each patch is proportional to its surface area. This can be easily achieved through weighted sampling of integer numbers. Moreover, two separate methods must be derived to sample from the conical and rectangular patches. For the conical patches, a slight modification of the method for the conical frustum will suffice. Indeed, the only modification required is in the sampling interval for $\theta$ that should be replaced by:

$$\theta \sim \mathcal{U}\left(\underline{\theta}, \bar{\theta}\right),$$

where the upper $\bar{\theta}$ and lower $\underline{\theta}$ sampling bounds depend on $\boldsymbol{e_x}$, $\boldsymbol{e_y}$, and the points where the circular arcs of the silhouette start and finish. As for the rectangular surfaces, uniform sampling is trivial and no further details are provided. Again, a batch of 200 points was sampled following this method and is shown in Figure 5b.

At first glance, the sampled points depicted in Figure 5 may not appear as relevant information for the path-planning algorithm. However, much like a human would when trying to avoid an obstacle, these points focus the attention in the vicinity of the obstacle in conflict, yet not so close as to render all potential paths unsafe. Moreover, unlike existing sample-based path-planning methods that pick arbitrary points in the configuration space [2, 7] or bounded regions of the configuration space [38, 39], these points inherently incorporate additional information obtained from the analysis of the local geometry of the obstacle. In the next section, insight on how these points can be used to generate paths around a single wire is provided.

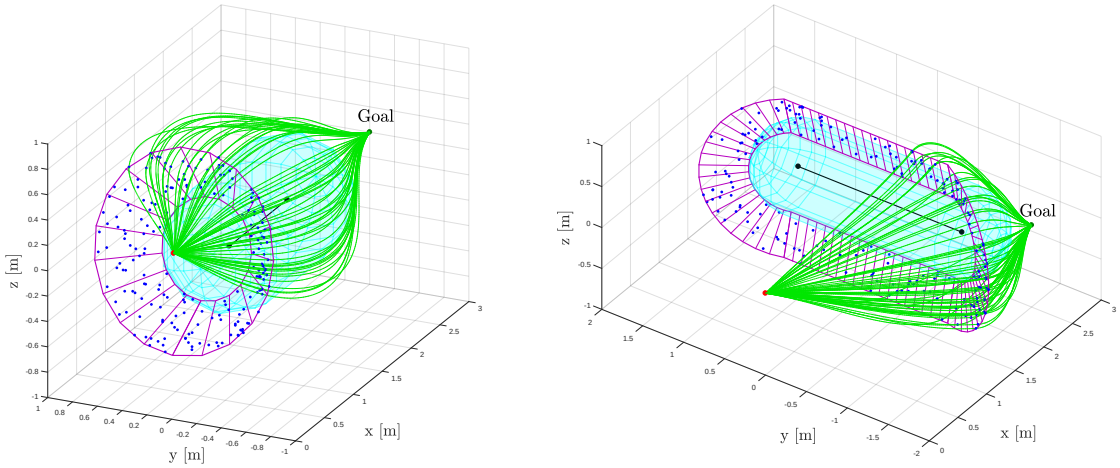
### C. Silhouette-informed Paths

In order to build a set of paths around a single wire, we define a goal point $\boldsymbol{x_{goal}}$, depicted as a green dot in Figure 6. The goal is selected so that if one were to draw a straight segment between $\boldsymbol{p_v}$ and $\boldsymbol{x_{goal}}$, then the resulting path would violate safe distance separation with the wire. The design of a silhouette-informed path around a single wire is divided in two fundamental steps:

i) First, a path from the point of view $\boldsymbol{p_v}$ to each of the sampled points $\boldsymbol{x_{rand}}$ on the expanded silhouette is sketched. Note that all the points on the expanded silhouette are in line-of-sight from $\boldsymbol{p_v}$. Thus, it should be relatively simple to design a path from $\boldsymbol{p_v}$ to each $\boldsymbol{x_{rand}}$ that does not violate safe separation constraints with the wire. In this example, we use the method discussed in [51] to generate $C^1-$continuous curves. Each curve has $\boldsymbol{p_v}$ as the initial point, $\boldsymbol{x_{rand}}$ as the final point, $\boldsymbol{x_{rand}} - \boldsymbol{p_v}$ as the initial value of the hodograph, and a vector perpendicular to the purple surface in Figure 6 as the final value of the hodograph.

ii) To complete the path a second leg from each sampled point $\boldsymbol{x_{rand}}$ to the goal $\boldsymbol{x_{goal}}$ is built. In this case, the initial position is $\boldsymbol{x_{rand}}$, the final position is $\boldsymbol{x_{goal}}$, the initial hodograph is the same as the final hodograph of the previous leg, and the final hodograph is $\boldsymbol{x_{goal}} - \boldsymbol{x_{rand}}$. It should be emphasized that $\boldsymbol{x_{goal}}$ is often not in line-of-sight from each $\boldsymbol{x_{rand}}$ since the unsafe region around the wire is in the way. If the second leg of the path violates the safe distance separation with $\mathcal{X}_{us}$, one could recompute the silhouette using the corresponding $\boldsymbol{x_{rand}}$ as the new point of view, thus adding additional legs as needed until a safe path that ends in $\boldsymbol{x_{goal}}$ is generated. To avoid cluttering Figure 6 only shows the paths that were designed successfully without having to recompute the silhouette.

Note that some of the sampled points of the purple surface in Figure 6 are not used to trace a path. Those points correspond to the paths that violated the safe distance constraints with the wire. To discard these paths, we developed an algorithm that performs tolerance verification queries between a Bézier curve and convex polytopes in three dimensions. The next section describes the tolerance verification algorithm.




**(a) The visible region from the point of view is a section of the cap.**

**(b) The visible region from the point of view includes parts of the two caps and a section of the cylinder.**

**Fig. 6 Safe paths generated around a wire.**

## VII. Tolerance Verification Queries

Given the $V$ representation[§] of a convex polytope $\mathcal{O}$ and a polynomial parametric curve $\boldsymbol{p}(\zeta) : [0,1] \to \mathbb{R}^3$ of degree $n$, the goal was to implement an algorithm that determines whether the distance between the polytope and the curve $d(\mathcal{O}, \boldsymbol{p})$ is greater than a safety distance $d_s$. To this purpose, we define the distance between $\mathcal{O}$ and $\boldsymbol{p}(\zeta)$ as the minimum Euclidean distance between any two points in the aforementioned geometric objects:

$$d(\mathcal{O}, \boldsymbol{p}) := \min \{ \|\boldsymbol{y} - \boldsymbol{x}\| \mid \boldsymbol{y} \in \mathcal{O}, \boldsymbol{x} \in \boldsymbol{p} \} \ .$$

As a result of the tolerance verification check, the algorithm shall return a binary answer

$$v_t = \begin{cases} 1\,, & \text{if } d(\mathcal{O}, \boldsymbol{p}) \geq d_s \\ 0\,, & \text{if } d(\mathcal{O}, \boldsymbol{p}) < d_s \end{cases} \ .$$

This could be done by computing the distance $d(\mathcal{O}, \boldsymbol{p})$ within a specified tolerance, as in [57], to then compare $d(\mathcal{O}, \boldsymbol{p})$ with $d_s$ and produce the binary answer we are looking for. The work in [57] proposes a branch-and-bound algorithm that combines de Casteljau's algorithm, the convex hull property of Bézier curves, and the Gilbert-Johnson-Keerthi (GJK) algorithm for distance computations between convex polytopes [49, 50]. However, the information that this algorithm produces – $d(\mathcal{O}, \boldsymbol{p})$ – is more than we need. This typically leads to higher computational costs than those generating the bare minimum information. Since the algorithms presented in this paper are intended to be implemented onboard a small UAS with limited computation resources, we developed a tolerance verification algorithm that does not require distance computations. The proposed algorithm relies on a modification of the GJK algorithm that performs tolerance verification queries between convex polytopes in three dimensions, and de Casteljau's algorithm for the subdivision of Bézier curves. For the purposes of this paper, the polytope is always a wire, but the methods developed here can be applied to any convex polytope in three dimensions. The next section includes a brief explanation of de Casteljau's algorithm.

### A. de Casteljau's Algorithm

As described in [45], de Casteljau's algorithm is a fundamental tool for subdividing Bézier curves due to its simplicity and numerical stability. The algorithm is based on the following property of the Bernstein basis:

$$b_k^{r+1}(\zeta) = \zeta\, b_{k-1}^r(\zeta) + (1-\zeta)\, b_k^r(\zeta)\,.$$

Given a value of the curve parameter $\zeta^*$ where the curve must be split, de Casteljau's algorithm constructs the following Pascal-like triangle:

$$\begin{array}{ccccccccc}
\boldsymbol{p_0^0} & & \boldsymbol{p_1^0} & & \boldsymbol{p_2^0} & & \cdots & & \boldsymbol{p_n^0} \\
& \boldsymbol{p_1^1} & & \boldsymbol{p_2^1} & & \cdots & & \boldsymbol{p_n^1} & \\
& & \boldsymbol{p_2^2} & & \cdots & & \boldsymbol{p_n^2} & & \\
& & & \ddots & \vdots & \iddots & & & \\
& & & & \boldsymbol{p_n^n} & & & &
\end{array}$$

where the $(j, r)$ element can be expressed as a function of the control points as

$$\boldsymbol{p_j^r} = (1-\zeta^*)\, \boldsymbol{p_{j-1}^{r-1}} + \zeta^*\, \boldsymbol{p_j^{r-1}}\,,$$

for $j \in \{1, \ldots, n\}$ and $r \in \{1, \ldots, n\}$. Then, the control points of the curves that result from this division are the lower left and lower right diagonals of the Pascal-like triangle. Consequently, the expressions for each subdivision are

$$\boldsymbol{q}(\zeta) = \sum_{k=0}^{n} \boldsymbol{p_k^k}\, b_k^n(\zeta)\,, \qquad \boldsymbol{r}(\zeta) = \sum_{k=0}^{n} \boldsymbol{p_n^k}\, b_k^n(\zeta)\,.$$

The next section addresses how the algorithms and the Bézier properties presented were combined to perform tolerance verification queries between a polynomial curve and a convex polytope in three dimensions.

[§]The $V$ representation of a convex polytope is a set of vertices whose convex hull defines the polytope itself. The minimal $V$ representation is the minimum number of vertices whose convex hull defines the polytope itself. In this case we do not require that the $V$ representation be minimal, since the polytopes arising in the implementation of the algorithm developed may have internal vertices.

### B. Tolerance Verification Algorithm for Polytopes and Bézier Curves

The essence of the algorithm lies in subdividing the original Bézier curve into smaller curves using de Casteljau's algorithm. Then, tolerance verification checks are performed between the convex hull of the control points of these subdivisions and the obstacle to determine whether $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ are further apart than the safety distance $d_s$. Recall that the convex hull of the control points naturally defines a bounding region where the curve is contained. Algorithm 2 presents the pseudo-code, while Figures 7 and 8 depict how the algorithm works for a case where $v_t = 1$ and $v_t = 0$, respectively. Captions in Figures 7 and 8 provide a physical interpretation of the different steps within the algorithm:

1) A tolerance verification check between the convex hull of the control points of $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ is performed using a modification of the GJK algorithm for tolerance verification queries, denoted as GJKTolerance in line 2 of Algorithm 2. In this case, there are two options:
   a) If $v_{ch} = 0$, then the convex hull of the control points of $\boldsymbol{p}(\zeta)$, and $\mathcal{O}$ are not further apart than $d_s$ (line 3). However, this does not provide any guarantees regarding the distance between $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$.
   b) If $v_{ch} = 1$, then $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ are further apart than $d_s$. The algorithms stops and returns $v_t = 1$ (line 20).
2) If $v_{ch} = 0$, then the relative position between $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ must be further investigated. The algorithm performs a tolerance verification check between the first control point $\boldsymbol{p_0}$ and $\mathcal{O}$ (line 4). Again, there are two possible options:
   a) If $v_0 = 0$, then $\mathcal{O}$ and $\boldsymbol{p_0}$ are closer than $d_s$. Since $\boldsymbol{p_0} \in \boldsymbol{p}(\zeta)$ – property $i$) of Bézier curves – $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ are not further apart than $d_s$. The algorithm stops and returns $v_t = 0$ (line 6).
   b) If $v_0 = 1$, then $\mathcal{O}$ and $\boldsymbol{p_0}$ are further apart than $d_s$. However, this does not provide any guarantees regarding the distance between $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$.
3) If $v_0 = 1$, then the relative position between $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ must be further investigated. The algorithm performs a tolerance verification check between the last control point $\boldsymbol{p_n}$ and $\mathcal{O}$ (line 8). Again, there are two possible options:

**Algorithm 2:** Tolerance verification algorithm for Bézier curves and polytopes.

```
1  function toleranceVerificationBezierPolytope(A, P, d_s);
   Input  : A matrix with coordinates of vertices in O organized in columns
            P matrix with the coordinates of the control points of p(ζ) organized in columns
            d_s safe separation distance
   Output : v_t tolerance verification result
                v_t = 1 if distance between polytopes O and curve p(ζ) is greater than d_s
                v_t = 0 otherwise
2  v_ch ← GJKTolerance(A, P, d_s);                  ▷ Tolerance verification polytope O - convex hull of P.
3  if ¬ v_ch then
4  |  v_0 ← GJKTolerance(A, p_0, d_s);             ▷ Tolerance verification polytope O - first control point.
5  |  if ¬ v_0 then
6  |  |  v_t ← 0;                                  ▷ Polytope O and p_0 are not at least d_s apart.
7  |  else
8  |  |  v_n ← GJKTolerance(A, p_n, d_s);         ▷ Tolerance verification polytope O - last control point.
9  |  |  if ¬ v_n then
10 |  |  |  v_t ← 0;                               ▷ Polytope O and p_n are not at least d_s apart.
11 |  |  else
12 |  |  |  (Q, R) ← deCasteljau(P, 0.5);          ▷ Curve subdivision at ζ* = 0.5. Defines curves q(ζ) and r(ζ).
13 |  |  |  v_t ← toleranceVerificationBezierPolytope(A, Q, d_s) ;   ▷ Recursive call with subdivision q(ζ).
14 |  |  |  if v_t then
15 |  |  |  |  v_t ← toleranceVerificationBezierPolytope(A, R, d_s) ;   ▷ Recursive call with subdivision r(ζ).
16 |  |  |  end
17 |  |  end
18 |  end
19 else
20 |  v_t ← 1 ;                                    ▷ Polytopes O and convex hull of control points are at least d_s apart.
21 end
22 return v_t
```

a) If $v_n = 0$, then $\mathcal{O}$ and $\boldsymbol{p_n}$ are closer than $d_s$. Since $\boldsymbol{p_n} \in \boldsymbol{p}(\zeta)$, $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ are not further apart than $d_s$. The algorithm stops and returns $v_t = 0$ (line 10).

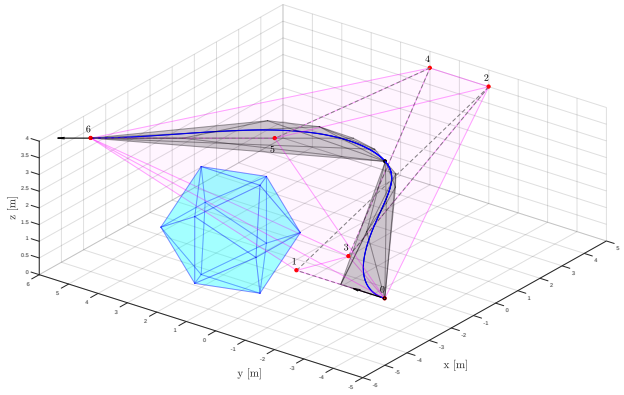
b) If $v_n = 1$, then $\mathcal{O}$ and $\boldsymbol{p_n}$ are further apart than $d_s$. This does not provide any guarantees regarding the distance between $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$. The curve is then subdivided using de Casteljau's algorithm (line 12) to determine whether the previous result – $v_{ch} = 0$ – is a consequence of approximating the curve by its bounding region, or the $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ are indeed closer than $d_s$. This results in two curves $\boldsymbol{q}(\zeta)$ and $\boldsymbol{r}(\zeta)$ with $\zeta \in [0, 1]$, defined by their corresponding control points, represented as $\boldsymbol{Q}$ and $\boldsymbol{R}$ in Algorithm 2.

4) If $v_n = 1$, then we utilize a recursive call to determine whether curve subdivisions $\boldsymbol{q}(\zeta)$ and $\boldsymbol{r}(\zeta)$ are further away than $d_s$ from polytope $\mathcal{O}$. First, we do so with subdivision $\boldsymbol{q}(t)$ (line 13). Again, there are two options:

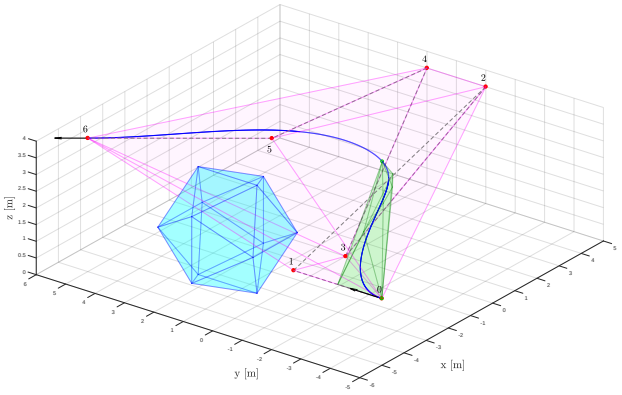
a) If $v_t = 1$, then a tolerance verification check must be performed on $\boldsymbol{r}(t)$ (line 15) to finish analyzing the entire curve.

b) If $v_t = 0$, then $\boldsymbol{r}(t)$ does not need to be checked, since $\boldsymbol{p}(\zeta)$ and $\mathcal{O}$ are not further apart than $d_s$. In this case, the algorithm stops and returns $v_t = 0$.

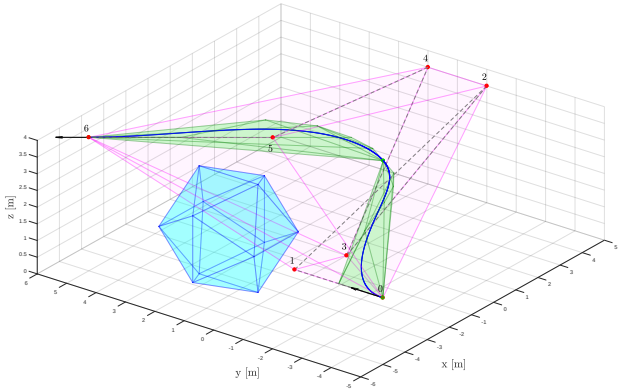
To the best of our knowledge, this is the first published tolerance verification algorithm for convex polytopes in three dimensions and Bézier curves. The next section combines all the tools presented – Bézier curves, silhouette information, and tolerance verification queries – to compute safe trajectories through the wire maze depicted in Figure 1.




**(a) $p(\zeta)$ is divided into two curves. The convex hull of the control points of $q(\zeta)$ and $r(\zeta)$ is depicted in gray.**

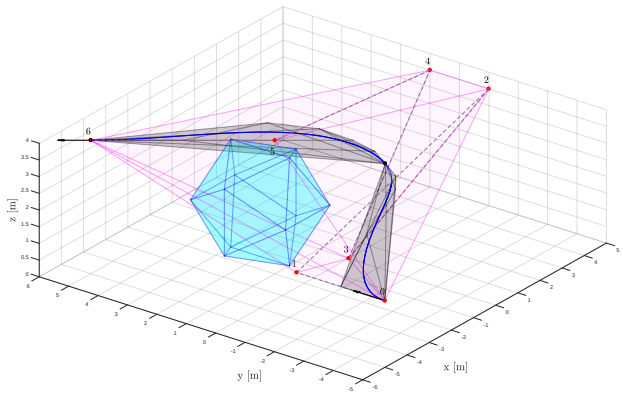



**(b) The convex hull of the control points of $q(\zeta)$ is deemed safe, and colored green, after a tolerance verification check.**

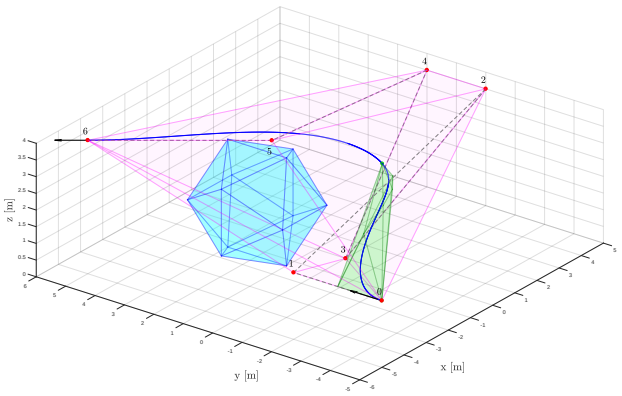



**(c) The convex hull of the control points of $r(\zeta)$ is deemed safe, and colored green, after a tolerance verification check.**

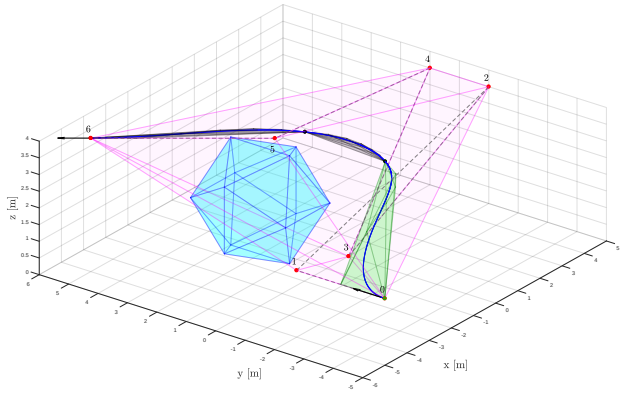
**Fig. 7 Visualization of the tolerance verification algorithm for $v_t = 1$. The convex hull of all subdivisions is deemed safe. Thus, $p(\zeta)$ does not violate safe separation constraints with the polytope.**



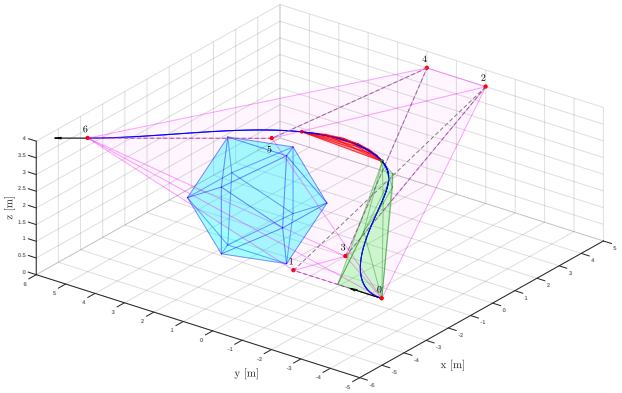

**(a) $p(\zeta)$ is divided into two curves. The convex hull of the control points of $q(\zeta)$ and $r(\zeta)$ is depicted in gray.**




**(b) The convex hull of the control points of $q(\zeta)$ is deemed safe, and colored green, after a tolerance verification check.**




**(c) $r(\zeta)$ is divided into two curves. The convex hull of the control points of $s(\zeta)$ and $t(\zeta)$ is depicted in gray.**




**(d) The curve $s(\zeta)$ is deemed unsafe, and colored red, after a tolerance verification check with its last control point.**

**Fig. 8 Visualization of the tolerance verification algorithm for $v_t = 0$. A single subdivision is deemed unsafe. Thus, $p(\zeta)$ violates safe separation constraints with the polytope.**

## VIII. Trajectory Generation Framework

Sections V through VII described the fundamental tools that are leveraged to generate trajectories through the wire-maze in Figure 1. To this end, we have developed a trajectory generation algorithm that builds upon single-query sampling-based methods in robotics [2, 38, 39] to generate a tree through the safe configuration space $\mathcal{X}_s$. This algorithm enhances the search through the most complicated areas of the wire maze by sampling near the boundary of the unsafe region with the silhouette algorithm presented in Section VI. In this manner, the method presented steers the sampling towards the narrow corridors in the wire maze, rather than just sampling uniformly over the domain, which would prioritize the exploration of larger but less interesting Voronoi regions. Once $\boldsymbol{x_{goal}}$ has been reached, a path is extracted from the silhouette-informed tree and smoothed using the Bézier curves described in Section V. To ensure the smoothed path still maintains the safe distance separation with all the obstacles, we developed an iterative smoothing technique that uses the tolerance verification algorithm for polytopes and Bézier curves defined in Section VII. Finally, temporal specifications are imposed to meet the constraints specified in Section IV. Hence, the trajectory generation algorithm can be broken down into three fundamental steps: *i*) silhouette-informed tree, *ii*) path smoothing, and *iii*) imposing temporal specifications. These steps are described in further detail in the following subsections.

### A. Silhouette Informed Trees (SIT)

Path generation through challenging scenarios has led to the emergence of heuristic approaches that improve the quality of the planned path, reduce the computational cost, or steer the search towards the goal region. Heuristic

approaches are both common in single-query algorithms, such as RRTs, and multiple-query algorithms, such as PRMs. One common approach is biased sampling of the free space, as detailed in [5, 58]. In both cases, the underlying goal is to reduce the overall number of samples by steering the graph toward the regions of interest. In fact, [37] highlights the importance of sampling along the boundary of the unsafe space $\mathcal{X}_{us}$ for PRMs. The work in [9, 59] confirms that sampling on the boundary of $\mathcal{X}_{us}$ improves the quality of the graph, as compared to just sampling $\mathcal{X}_s$. To this end, the algorithms presented in [9, 37, 59] provide mechanisms to identify and sample along narrow corridors for PRMs. However, the methods proposed are rarely used for rapidly expanding trees, due to the necessary initial analysis of the configuration space and associated computational cost.

The algorithm we propose broadens the spectrum of heuristic approaches for rapidly-exploring random trees by using silhouette information. Algorithm 3 contains the pseudo-code of the method. Note that Algorithm 3 is similar to RRT* described in [2]. Indeed, lines 13 through 34 are identical to the corresponding lines in [2]. Much like RRT*, Algorithm 3 builds a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ through the flight-safe region $\mathcal{X}_s$. This tree consists of a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$ that maintain safe distance separation from all obstacles. The algorithm differs from RRT* in that it uses silhouette information to sample around an obstacle in conflict when a candidate branch fails a tolerance verification test. To provide a coherent structure, the algorithm has been divided into four subsections: sampling logic, tolerance verification logic, connections along a minimum-cost path, and tree rewiring.



*1. Sampling logic*

The first difference between Algorithm 3 and RRT* is the definition of a sampling-mode flag $s_m$ in line 3. This flag determines the sampling approach that will be used in each iteration of the SIT algorithm.

i) If $s_m = 1$, then samples are picked from the expanded silhouette of an obstacle that violated safe separation constrains with the previous candidate branch. This obstacle is identified through $id_c$, a number returned by the function toleranceVerification in line 13. The function silhouetteWire in line 6 takes the identification number of the obstacle in conflict $id_c$ and uses $\boldsymbol{x_{nearest}}$ – from the previous failed candidate branch – as the point of view to compute the silhouette as detailed in Section VI.A. Next, sampleSilhouette computes a random sample in the expanded silhouette as shown in Section VI.B.

ii) If $s_m = 0$, then the SIT algorithm samples the configuration space as RRT* would. The function sample in line 9 may also contain some of the sample-biasing methods proposed in [5, 37], as well as other heuristics with the potential to reduce the number of samples required to reach $\boldsymbol{x_{goal}}$. For the purposes of this paper, the sampling was biased towards $\boldsymbol{x_{goal}}$, as suggested in [5, 58], to steer the tree in that direction. In some sense, this creates two opposing forces that drive the evolution of $\mathcal{T}$. One that pulls the tree towards $\boldsymbol{x_{goal}}$, and another – spurred by the silhouette – helps avoid nearby obstacles[¶].

Note that only lines 35 and 37 in Algorithm 3 modify the value of the sampling-mode flag. Line 35 ensures that the regular sampling mode – $s_m = 0$ – is set after a tree edge has been added successfully, whereas line 37 negates the current value of $s_m$. Thus, if $s_m = 0$ in the previous step, then in the next iteration silhouette information will be leveraged for path planning. However, if $s_m = 1$ in the previous step, then the SIT algorithm will return to the regular sampling mode. This was designed to ensure that the algorithm does not focus all attention on a cul-de-sac; and intended to balance the narrow passage search – provided by the silhouette – with the exploration of the configuration space that RRT* performs. Lines 11 and 12 contain the steps that let RRT* build the tree incrementally. The function nearest returns the node in $\mathcal{T}$ that is closest to $\boldsymbol{x_{rand}}$, while steer$(\boldsymbol{x}, \boldsymbol{y})$ returns a point $\boldsymbol{x_{new}}$:

$$\boldsymbol{x_{new}} = \begin{cases} \boldsymbol{y}, & \text{if } \|\boldsymbol{x} - \boldsymbol{y}\| \leq \eta \\ \operatorname{argmin}_{z \in \mathcal{B}_{\boldsymbol{x},\eta}} \|z - \boldsymbol{y}\|, & \text{otherwise} \end{cases},$$

where $\mathcal{B}_{\boldsymbol{x},\eta}$ is a ball centered at $\boldsymbol{x}$ of radius $\eta$, and $\eta$ is the steering distance.

*2. Tolerance verification logic*

Once a candidate branch $\{\boldsymbol{x_{nearest}}, \boldsymbol{x_{new}}\}$ has been generated, the function toleranceVerification evaluates whether it lies in $\mathcal{X}_s$. In our case, it loops through different obstacles and utilizes the modification of the GJK algorithm for tolerance verification, denoted by GJKTolerance in Algorithm 2. If the function detects a conflict, it returns $v_t = 0$ and the identification number of the corresponding obstacle $id_c$. If $\{\boldsymbol{x_{nearest}}, \boldsymbol{x_{new}}\} \in \mathcal{X}_s$, then toleranceVerification

[¶]The idea of these "forces" was inspired by zombie apocalypse movies where zombies head towards their pray relentlessly while avoiding obstacles and slipping through crevices.

returns $v_t = 1$ and an empty identification number $id_c = \emptyset$. The remaining subsections are commented briefly, since they were developed for RRT* [2].

*3. Connections along a minimum-cost path*

As described in [2], first the algorithm considers possible connections with neighboring vertices in a ball of radius $r$ centered around $\boldsymbol{x_{new}}$. The radius is computed at each iteration by the function radius that implements the expression defined in [2]

$$r = \min\left\{\gamma_{RRT^*}\left(\frac{\log\left(\text{card}(\mathcal{V})\right)}{\text{card}(\mathcal{V})}\right)^{\frac{1}{d}}, \eta\right\},$$

where $\gamma_{RRT^*}$ is a tuning parameter, and $d$ is the dimension of the configuration space. The vertex with the smallest cost $\boldsymbol{x_{min}}$ is initialized with the nearest vertex $\boldsymbol{x_{nearest}}$ in line 18, where the function cost returns the cost associated with the input vertex, and the function costLine returns the cost associated with the line segments that connects the input vertices. Notice that only the edge in $\mathcal{X}_s$ with the minimum cost is added to the tree in lines 19 through 25.

*4. Tree rewiring*

Finally, if the path created through $\boldsymbol{x_{new}}$ to some $\boldsymbol{x_{near}} \in \mathcal{X}_{near}$ has lower cost than the original path to $\boldsymbol{x_{near}}$, then the initial edge is deleted and replaced by $\{\boldsymbol{x_{new}}, \boldsymbol{x_{near}}\}$. To this end, the function parent in line 31 extracts the original parent node of $\boldsymbol{x_{near}}$.

## B. Path Smoothing

The details of this algorithm are beyond the scope of this paper, and hence only a brief description is provided. Given a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with a set of edges $\tilde{\mathcal{E}}$ that connects $\boldsymbol{x_{init}}$ with $\boldsymbol{x_{goal}}$, the path-smoothing algorithm extracts $\tilde{\mathcal{E}}$, applies an iterative edge reduction technique, and smooths the path.

The objective of the edge reduction technique is to decrease the number of edges in $\tilde{\mathcal{E}}$, to provide a set $\tilde{\mathcal{E}}_r$. The set $\tilde{\mathcal{E}}_r$ contains the same number or fewer edges than $\tilde{\mathcal{E}}$ but still connects $\boldsymbol{x_{init}}$ with $\boldsymbol{x_{goal}}$ safely. The method used loops through all the vertices in $\tilde{\mathcal{E}}$ and attempts to connect them to non-contiguous vertices. To ensure that the result lies in $\mathcal{X}_s$, the algorithm leverages the function GJKTolerance used in Algorithm 2. If the tolerance verification check fails, then the candidate segment is never added to $\tilde{\mathcal{E}}_r$.

Finally, $\tilde{\mathcal{E}}_r$ is fed to a path-smoothing algorithm. The method implemented uses the PH Bézier curves of degree 9 described in [53]. These curves allow the specification of the value and direction of the first and second parametric derivatives at the initial and final points of the curve. The algorithm ensures that the first and second parametric derivatives of the curve at the common point between contiguous curves coincide in direction, but not necessarily in magnitude. Later, the speed design algorithm will guarantee that both the magnitude and direction of the first and second temporal derivatives of the curves coincide. This is achieved through a careful design of the map from time $t$ to the curve parameter $\zeta$. Initially, the algorithm attempts to smooth every pair of contiguous edges with curves that have larger curvature values. These curves are tested for safe distance separation constraints using Algorithm 2. If the curves fail the tolerance verification check, then the algorithm attempts to smooth the pair of edges with smaller curvature values. As this process is repeated, the smoothed curves approach the edges in $\tilde{\mathcal{E}}_r$. In the event that the path cannot be smoothed safely – due to the existence of a very narrow passage at a particular location – the algorithm returns those straight edges after a number of pre-specified iterations. The result $\boldsymbol{p}(\zeta) \in \mathcal{X}_s$ is a smoothed path, to the extent possible, expressed as a sequence of $n_c$ curves

$$\boldsymbol{p}(\zeta) = {}^{\boldsymbol{i}}\boldsymbol{p}\left(\frac{\zeta - \zeta_{i-1}}{\zeta_i - \zeta_{i-1}}\right), \qquad \forall \zeta \in [\zeta_{i-1}, \zeta_i], \qquad \forall i \in \{1, \ldots, n_c\}, \tag{3}$$

where ${}^{\boldsymbol{i}}\boldsymbol{p}(\cdot)$ represents the $i$th curve, and

$$\zeta_0 = 0, \qquad \zeta_i = \frac{\sum_{j=1}^{i} L_i}{L_T}, \qquad \text{with} \qquad L_T = \sum_{i=1}^{n_c} L_i,$$

and $L_i$ is the length of the $i$th curve, which can be computed with Equation (1).

**Algorithm 3:** Silhouette Informed Trees.

**function** silhouetteInformedTree($\boldsymbol{x_{init}}, \boldsymbol{x_{goal}}, n_{sil}, d_s, d_e, \mathcal{X}_{obs}$);
**Input** : $\boldsymbol{x_{init}}$ coordinates of the initial point
$\boldsymbol{x_{goal}}$ coordinates of the goal point
$n$ number of samples
$d_s$ safe separation distance
$d_e$ expanding distance
**Output** : $\mathcal{T}$ tree
$\mathcal{V} \leftarrow \{\boldsymbol{x_{init}}\}$; $\mathcal{E} \leftarrow \emptyset$; $\mathcal{T} = (\mathcal{V}, \mathcal{E})$; ▷ Initialize tree.
$s_m \leftarrow 0$; ▷ Initialize sampling-mode flag.
**for** $i = 1, \ldots, n$ **do**

// Sampling logic
**if** $s_m$ **then**
$sil \leftarrow$ silhouetteWire($\boldsymbol{x_{nearest}}, id_c, d_s, d_e$) ; ▷ Compute silhouette.
$\boldsymbol{x_{rand}} \leftarrow$ sampleSilhouette($sil, n_b$) ; ▷ Sample from the extended silhouette.
**else**
$\boldsymbol{x_{rand}} \leftarrow$ sample ; ▷ Sample from the configuration space.
**end**
$\boldsymbol{x_{nearest}} \leftarrow$ nearest($\mathcal{T}, \boldsymbol{x_{rand}}$); ▷ Find vertex on the tree that is closest to $\boldsymbol{x_{rand}}$.
$\boldsymbol{x_{new}} \leftarrow$ steer($\boldsymbol{x_{nearest}}, \boldsymbol{x_{rand}}$); ▷ Candidate to new vertex.

// Tolerance verification logic
$(v_t, id_c) \leftarrow$ toleranceVerification($\boldsymbol{x_{nearest}}, \boldsymbol{x_{new}}$); ▷ Check safety distance separation.
**if** $v_t$ **then**

// Connections along a minimum-cost path
$r \leftarrow$ radius($\mathcal{V}$); ▷ Compute maximum radius to consider other edge connections.
$\mathcal{X}_{near} \leftarrow$ near($\mathcal{T}, \boldsymbol{x_{new}}, r$) ; ▷ Compute set of near vertices.
$\mathcal{V} \leftarrow \mathcal{V} \cup \boldsymbol{x_{new}}$ ; ▷ Add vertex.
$\boldsymbol{x_{min}} \leftarrow \boldsymbol{x_{nearest}}$; $c_{min} \leftarrow$ cost($\boldsymbol{x_{nearest}}$) + costLine($\boldsymbol{x_{nearest}}, \boldsymbol{x_{new}}$);
**foreach** $\boldsymbol{x_{near}} \in \mathcal{X}_{near}$ **do**
$(v_{t_n}, \sim) \leftarrow$ toleranceVerification($\boldsymbol{x_{near}}, \boldsymbol{x_{new}}$); ▷ Check safety distance separation.
$c_n \leftarrow$ cost($\boldsymbol{x_{near}}$) + costLine($\boldsymbol{x_{near}}, \boldsymbol{x_{new}}$); ▷ Compute cost.
**if** $v_{t_n} \wedge c_n < c_{min}$ **then**
$\boldsymbol{x_{min}} \leftarrow \boldsymbol{x_{near}}$; $c_{min} \leftarrow c_n$ ; ▷ Update vertex with minimum cost and minimum cost.
**end**
$\mathcal{E} \leftarrow \mathcal{E} \cup \{\boldsymbol{x_{min}}, \boldsymbol{x_{new}}\}$ ; ▷ Add edge with minimum cost.
**end**

// Tree rewiring
**foreach** $\boldsymbol{x_{near}} \in \mathcal{X}_{near}$ **do**
$(v_{t_n}, \sim) \leftarrow$ toleranceVerification($\boldsymbol{x_{new}}, \boldsymbol{x_{near}}$); ▷ Check safety distance separation.
$c_n \leftarrow$ cost($\boldsymbol{x_{new}}$) + costLine($\boldsymbol{x_{new}}, \boldsymbol{x_{near}}$); ▷ Compute cost.
**if** $v_{t_n} \wedge c_n <$ cost($\boldsymbol{x_{near}}$) **then**
$\boldsymbol{x_{parent}} \leftarrow$ parent($\boldsymbol{x_{near}}$) ; ▷ Find parent node.
**end**
$\mathcal{E} \leftarrow (\mathcal{E} \backslash \{\boldsymbol{x_{parent}}, \boldsymbol{x_{near}}\}) \cup \{\boldsymbol{x_{new}}, \boldsymbol{x_{near}}\}$ ; ▷ Remove initial edge, and add new edge.
**end**
$s_m \leftarrow 0$
**else**
$s_m \leftarrow \neg\, s_m$
**end**
**end**

### C. Temporal Specifications

The last step of the trajectory generation framework is to design the speed profile. Since this problem is beyond the scope of this paper, Section IV assumed that the speed profile $v(\tilde{t})$ was provided as a polynomial function of a normalized time $\tilde{t}$. Then, the only problem left is to find the evolution of the position of the UAS as a function of time. To this end, we express $v(\tilde{t})$ in a Bernstein basis

$$v(\tilde{t}) = \sum_{i=0}^{n} v_k \, b_k^n(\tilde{t}), \tag{4}$$

and find the parametric speed of each curve as:

$$\sigma(\zeta) = {}^i\sigma\left(\frac{\zeta - \zeta_{i-1}}{\zeta_i - \zeta_{i-1}}\right), \qquad \forall \zeta \in [\zeta_{i-1}, \zeta_i], \qquad \forall i \in \{1, \ldots, n_c\}\ . \tag{5}$$

Finally, the integral of Equation (4) over the normalized time and the integral of Equation (5) over the curve parameter are both equal to the arclength of the path. Hence, equating both expressions, one obtains the relationship between the normalized time $\tilde{t}$ and the curve parameter $\zeta$:

$$\int_0^{\zeta} \sigma(\epsilon)\, \mathrm{d}\epsilon = \int_0^{\tilde{t}} v(\tau)\, \mathrm{d}\tau$$

Since the curves in Equation (3) are PH, each ${}^i\sigma(\cdot)$ has a polynomial expression. In addition, $\sigma(\zeta)$ is a positive definite function and, thus, its integral is a monotonically increasing function. Hence, the expression above is a polynomial equation with a single real root for every $\tilde{t} \in [0, 1]$, which can be efficiently solved using property $vi$) in Section V.A as described in [48].

## IX. Simulation Results

The trajectory generation framework detailed in Section VIII was used to plan a trajectory through the wire maze in Figure 9. To this purpose, the $x$ coordinates of $\boldsymbol{x_{init}}$ and $\boldsymbol{x_{goal}}$ were set to ensure that these points were on either side of the maze, while the $y$ and $z$ coordinates were set randomly to the following values:

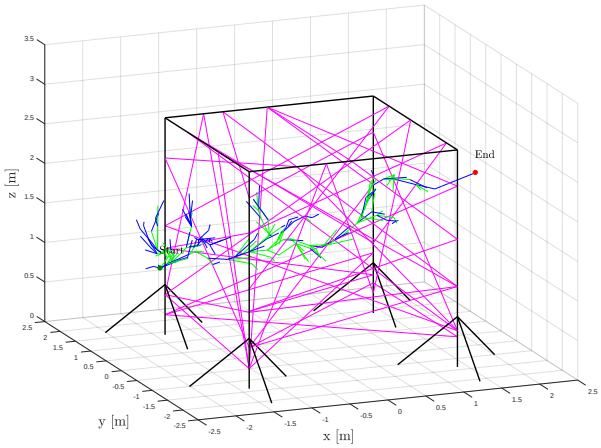
$$\boldsymbol{x_{init}} = [-2.00\,,\ -0.02\,,\ 1.25]^\top, \qquad \boldsymbol{x_{goal}} = [2.00\,,\ -0.41\,,\ 2.15]^\top.$$

In addition, the safety distance and tuning parameters of the SIT algorithms were set to

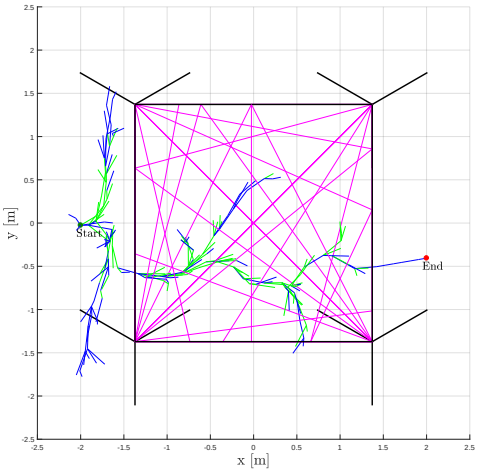
$$d_s = 0.25\ m, \qquad h_s = 1.00\ m, \qquad \eta = 0.10\ m, \qquad \gamma_{RRT^*} = 1.5\,.$$

The sampling of the configuration space in line 9 of Algorithm 3 was biased towards the goal so that 40% of the samples were $\boldsymbol{x_{goal}}$, whereas the remaining 60% were distributed uniformly over the configuration space. As for the silhouette computation and sampling in lines 6 and 7, the expanding distance was set to $d_e = 0.25\ m$. The cost functions in lines 18 and 29 of Algorithm 3 implemented the arclength of the path.

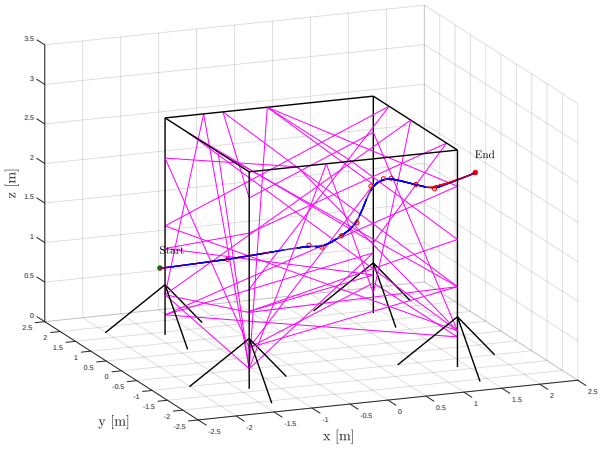
Figures 9e and 9b depict the tree that was generated by the SIT algorithm. The edges of the tree are color coded to highlight the sampling mechanisms that informed each of the branches. Green edges correspond to silhouette-informed branches, while blue edges correspond to the sampling of the configuration space biased towards $\boldsymbol{x_{goal}}$. From a qualitative point of view, silhouette-informed branches tend to concentrate around obstacles, whereas blue branches tend to grow towards the largest neighboring Voronoi regions. This is particularly noticeable in Figure 9b. Note also that the goal-biased sampling pulls the tree towards $\boldsymbol{x_{goal}}$. However, as this tree finds different wires on its way, the silhouette-informed sampling is creating alternate paths to surpass those obstacles. This can be easily deduced by observing how the green branches tend to concentrate around the regions where the tree diverges in different directions. It is also noticeable that the algorithm was able to find a relatively good solution without having to explore the entire configuration space in depth. Nonetheless, if the algorithm is run with larger numbers of samples, the tree eventually populates the entire configuration space, as RRT* would. If one wishes to reduce computational cost, then the trade-off solution is to reduce the number of samples in the configuration space. However, the heuristic approach utilized by SIT guided the tree so that the solution found had a relatively low cost, while maintaining a relatively low number of samples of the configuration space. In exchange for this informed guidance of the tree through the narrow corridors, the algorithm has a larger number of tuning variables, and an additional but relatively small computational cost to generate the silhouette information.



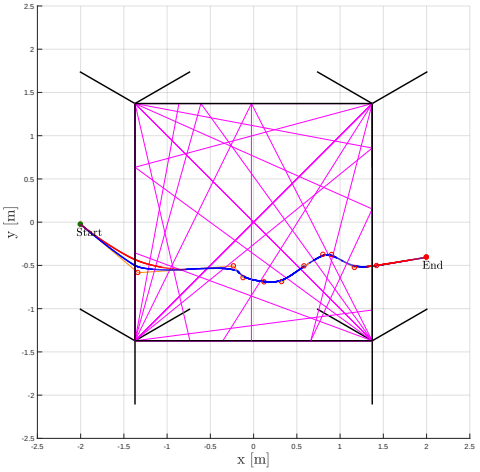

(a) 3D view of the silhouette-informed tree. Green edges are silhouette informed, whereas blue edges are not.



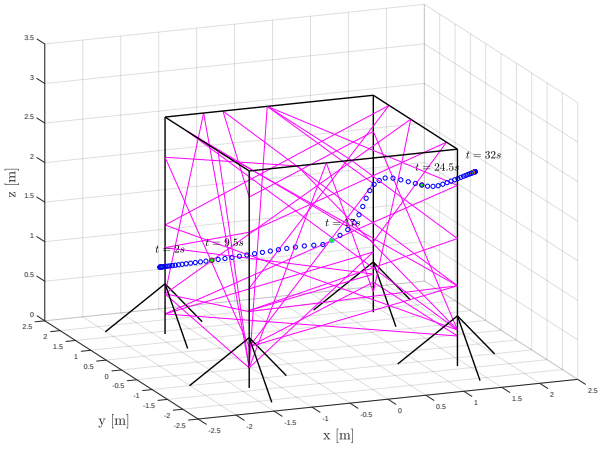

(b) Top view of the silhouette-informed tree. Green edges are silhouette informed, whereas blue edges are not.



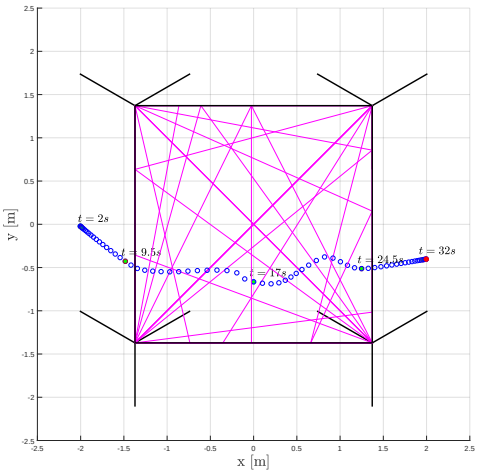

(c) 3D view of the smoothed path in blue, and the path after the segment reduction in orange.




(d) Top view of the smoothed path in blue, and the path after the segment reduction in orange.




(e) 3D view of the trajectory, points are sampled along the path every 0.5 seconds.

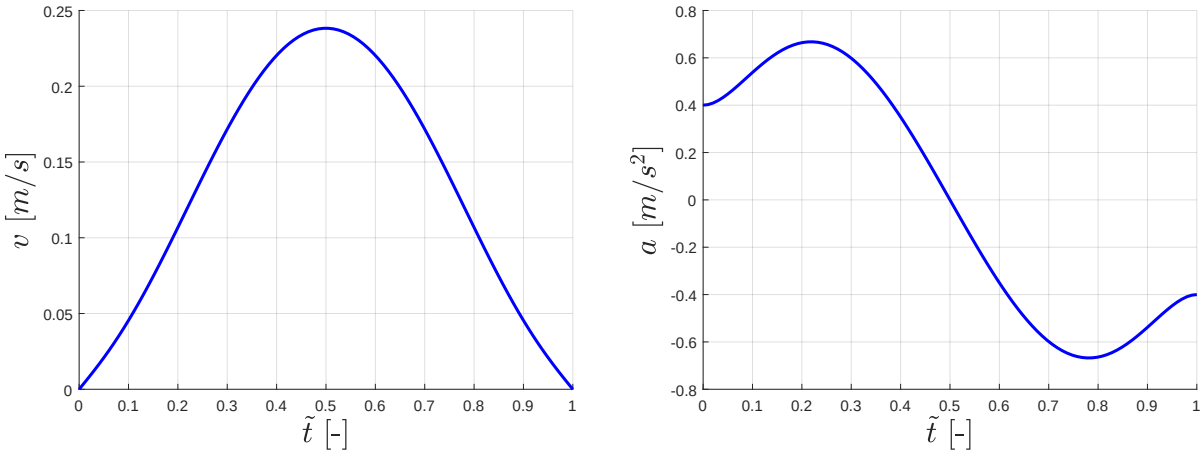



(f) Top view of the trajectory, points are sampled along the path every 0.5 seconds.

**Fig. 9 Trajectory generation results of the SIT algorithm for the wire maze.**

Figures 9c and 9d represent the results of the path-smoothing algorithm. The line segments shown in orange with circles on their vertices depict the set $\tilde{\mathcal{E}}_r$ that was returned by the edge reduction algorithm. The blue curve describes the final path, and the red curve – seen close to $\boldsymbol{x_{init}}$ and $\boldsymbol{x_{goal}}$ – represents a curve that was initially generated to smooth the corresponding pair of line segments but failed the tolerance verification checks. Notice how the red curves have larger curvature values than the corresponding blue curves.

Finally, Figures 9e and 9f show the temporal evolution of the UAS along the path. The points in this figure are sampled every 0.5 $s$. Consequently, the points are closer together for smaller speed values. Temporal tags are added to some of these points for reference. The planned flight time was 34 $s$, the speed profile is shown in Figure 10a, and acceleration is shown in Figure 10b. Note that the speed profile at $\boldsymbol{x_{init}}$ and $\boldsymbol{x_{goal}}$ was set to 0 $m/s$ to ensure a smooth departure and arrival. In addition, the acceleration profile consistently maintains relatively low values.




**(a) Speed profile as a function of the normalized time $\tilde{t}$.** **(b) Acceleration as a function of the normalized time $\tilde{t}$.**

**Fig. 10 Temporal specifications for the flight.**

## X. Conclusion

The SIT algorithm broadens the heuristic methods available for rapidly-exploring random trees. Moreover, the silhouette-informed method proposed here has the potential to be applied to other path-planning algorithms such as PRM or even potential fields. The algorithm successfully informs the generation of a tree through the narrow corridors of interest in the wire maze, even for a relatively low number of samples. In addition, to support the SIT algorithm silhouette computation, expansion, and sampling methods were developed. A brief description of a path-smoothing mechanism and the necessary tools to ensure the smoothed path remains in the flight-safe configuration space were presented. A tolerance verification algorithm for Bézier curves and convex polytopes in three dimensions was developed. Unlike other path-smoothing algorithms that resort to uniform sampling over the path to ensure safe separation with obstacles, the tolerance verification algorithm avoids sampling of the path. Finally, we were able to enforce an arbitrary speed profile with a polynomial structure by exploiting the PH condition of the curves returned by the path-smoothing algorithm.

## Acknowledgments

This research was supported by the Autonomy Incubator at NASA Langley Research Center. The authors acknowledge the use of services and facilities at NASA Langley Research Center, and the valuable help of the team at the Autonomy Incubator at NASA Langley Research Center and the Advanced Controls Research Laboratory at the University of Illinois at Urbana-Champaign.

# References


[1] Lavalle, S. M., "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Tech. rep., 1998.

[2] Karaman, S., and Frazzoli, E., "Sampling-based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research*, Vol. 30, No. 7, 2011, pp. 846–894.

[3] Qureshi, A. H., and Ayaz, Y., "Potential Functions Based Sampling Heuristic for Optimal Path Planning," *Autonomous Robots*, Vol. 40, No. 6, 2016, pp. 1079–1093.

[4] Perez, A., Platt, R., Konidaris, G., Kaelbling, L. P., and Lozano-Pérez, T., "LQR-RRT*: Optimal Sampling-based Motion Planning with Automatically Derived Extension Heuristics," Saint Paul, MN, USA, 2012, pp. 2537–2542.

[5] Urmson, C., and Simmons, R., "Approaches for Heuristically Biasing RRT Growth," *International Conference on Intelligent Robots and Systems*, Vol. 2, Las Vegas, NV, USA, 2003, pp. 1178–1183.

[6] Akgun, B., and Stilman, M., "Sampling Heuristics for Optimal Motion Planning in High Dimensions," *International Conference on Intelligent Robots and Systems*, San Francisco, CA, USA, 2011, pp. 2640–2645.

[7] Kuffner, J. J., and LaValle, S. M., "RRT-connect: An Efficient Approach to Single-query Path Planning," *International Conference on Robotics and Automation*, Vol. 2, San Francisco, CA, USA, 2000, pp. 995–1001.

[8] Du, M., Chen, J., Zhao, P., Liang, H., Xin, Y., and Mei, T., "An Improved RRT-based Motion Planner for Autonomous Vehicle in Cluttered Environments," *International Conference on Robotics and Automation*, Hong Kong, China, 2014, pp. 4674–4679.

[9] Amato, N. M., Bayazit, O. B., Dale, L. K., Jones, C., and Vallejo, D., "OBPRM: An Obstacle-based PRM for 3D Workspaces," *Algorithmic Foundations of Robotics*, Natick, MA, USA, 1998.

[10] Zhang, L., and Manocha, D., "An Efficient Retraction-based RRT Planner," *International Conference on Robotics and Automation*, Pasadena, CA, USA, 2008, pp. 3743–3750.

[11] Denny, J., Sandström, R., Bregger, A., and Amato, N. M., "Dynamic Region-biased Rapidly-exploring Random Trees," 2016.

[12] Cohen, B., Subramanian, G., Chitta, S., and Likhachev, M., "Planning for Manipulation with Adaptive Motion Primitives," *International Conference on Robotics and Automation*, Shanghai, China, 2011, pp. 1–8.

[13] Hourtash, A., and Tarokh, M., "Manipulator Path Planning by Decomposition: Algorithm and Analysis," *IEEE Transactions on Robotics and Automation*, Vol. 17, No. 6, 2001, pp. 842–856.

[14] Weghe, M. V., Ferguson, D., and Srinivasa, S. S., "Randomized Path Planning for Redundant Manipulators without Inverse Kinematics," *IEEE-RAS International Conference on Humanoid Robots*, 2007, pp. 477–482.

[15] Martínez-Alfaro, H., and Gómez-García, S., "Mobile Robot Path Planning and Tracking Using Simulated Annealing and Fuzzy Logic Control," *Expert Systems with Applications*, Vol. 15, No. 3, 1998, pp. 421– 29.

[16] Hu, Y., Yang, S. X., Xu, L.-Z., and Meng, M. Q. H., "A Knowledge Based Genetic Algorithm for Path Planning in Unstructured Mobile Robot Environments," *IEEE International Conference on Robotics and Biomimetics*, Shenyang, China, 2004, pp. 767–772.

[17] Jan, G. E., Chang, K. Y., and Parberry, I., "Optimal Path Planning for Mobile Robot Navigation," *IEEE/ASME Transactions on Mechatronics*, Vol. 13, No. 4, 2008, pp. 451–460.

[18] García, M. P., Montiel, O., Castillo, O., Sepúlveda, R., and Melin, P., "Path Planning for Autonomous Mobile Robot Navigation with Ant Colony Optimization and Fuzzy Cost Function Evaluation," *Applied Soft Computing*, Vol. 9, No. 3, 2009, pp. 1102–1110.

[19] Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., and How, J. P., "Real-Time Motion Planning With Applications to Autonomous Urban Driving," *IEEE Transactions on Control Systems Technology*, Vol. 17, No. 5, 2009, pp. 1105–1118.

[20] Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J., "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments," *The International Journal of Robotics Research*, Vol. 29, No. 5, 2010, pp. 485–501.

[21] Shim, D. H., Kim, H. J., and Sastry, S., "Decentralized Nonlinear Model Predictive Control of Multiple Flying Robots," *IEEE International Conference on Decision and Control*, Vol. 4, Maui, HI, USA, 2003, pp. 3621–3626.

[22] Idris, H., Shen, N., and Wing, D. J., "Complexity Management Using Metrics for Trajectory Flexibility Preservation and Constraint Minimization," *AIAA Aviation Technology, Integration, and Operations Conference*, Virginia Beach, VA, USA, 2011, pp. 1–18.

[23] Karr, D., Vivona, R., Roscoe, D., Depascale, S., and Wing, D., "Autonomous Operations Planner: A Flexible Platform for Research in Flight-Deck Support for Airborne Self-Separation," *AIAA Aviation Technology, Integration, and Operations Conference*, Indianapolis, IN, USA, 2012, pp. 1–21.

[24] Wing, D. J., Ballin, M. G., Koczo, S., and Vivona, R. A., "Developing an On-Board Traffic-Aware Flight Optimization Capability for Near-Term Low-Cost Implementation," *AIAA Aviation Technology, Integration, and Operations Conference*, Los Angeles, CA, USA, 2013, pp. 1–13.

[25] Van der Velden, C., Bil, C., Yu, X., and Smith, A., "An Intelligent System for Automatic Layout Routing in Aerospace Design," *Innovations in Systems and Software Engineering*, Vol. 3, No. 2, 2007, pp. 117–128.

[26] Canny, J., and Reif, J., "New Lower Bound Techniques for Robot Motion Planning Problems," 1987, pp. 49–60.

[27] Valavanis, K. P., "Unmanned Aircraft Systems Challenges in Design for Autonomy," *International Workshop on Robot Motion and Control*, Wasowo, Poland, 2017, pp. 73–86.

[28] Lee, R., Kochenderfer, M., Mengshoel, O., Brat, G., and Owen, M., "Adaptive Stress Testing of Airborne Collision Avoidance Systems," *IEEE/AIAA Digital Avionics Systems Conference*, Prague, Czech Republic, 2015, pp. 1–24.

[29] Ruijters, D., and Vilanova, A., "Optimizing GPU Volume Rendering," *Winter School of Computer Graphics*, Vol. 14, Bory, Czech Republic, 2006, pp. 9–16.

[30] Jessup, J., Givigi, S. N., and Beaulieu, A., "Merging of Octree Based 3D Occupancy Grid Maps," *IEEE International Systems Conference Proceedings*, IEEE, Washington, DC, USA, 2014, pp. 371–377.

[31] Lapierre, L., Soetanto, D., and Pascoal, A., "Coordinated Motion Control of Marine Robots," *IFAC Proceedings Volumes*, Vol. 36, No. 21, 2003, pp. 217–222.

[32] Soetanto, D., Lapierre, L., and Pascoal, A., "Adaptive, Non-singular Path-following Control of Dynamic Wheeled Robots," *IEEE International Conference on Decision and Control*, Vol. 2, Maui, HI, USA, 2003, pp. 1765–1770.

[33] Xargay, E., "Time-critical Cooperative Path-following Control of Multiple Unmanned Aerial Vehicles," Ph.D. thesis, University of Illinois at Urbana–Champaign, 2013.

[34] Choe, R., Puig-Navarro, J., Cichella, V., Xargay, E., and Hovakimyan, N., "Cooperative Trajectory Generation Using Pythagorean Hodograph Bézier Curves," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 8, 2016, pp. 1744–1763.

[35] Taranenko, V. T., "Experience of Employment of Ritz's, Poincare's, and Lyapunov's Methods for Solving Flight Dynamics Problems," *Air Force Engineering Academy Press*, Moscow, Russia, 1968. (in Russian).

[36] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, 2005.

[37] LaValle, S. M., *Planning Algorithms*, Cambridge, 2006.

[38] Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D., "Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic," *International Conference on Intelligent Robots and Systems*, Chicago, IL, USA, 2014, pp. 2997–3004.

[39] Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D., "Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs," *International Conference on Robotics and Automation*, Seattle, WA, USA, 2015, pp. 3067–3074.

[40] Al-Hiddabi, S. A., and McClamroch, N. H., "Tracking and Maneuver Regulation Control for Nonlinear Nonminimum Phase Systems: Application to Flight Control," *IEEE Transactions on Control Systems Technology*, Vol. 10, No. 6, 2002, pp. 780–792.

[41] Aguiar, A. P., and Hespanha, J. P., "Position Tracking of Underactuated Vehicles," *American Control Conference*, Vol. 3, Denver, CO, USA, 2003, pp. 1988–1993.

[42] Kaminer, I., Pascoal, A., Xargay, E., Hovakimyan, N., Cao, C., and Dobrokhodov, V., "Path Following for Small Unmanned Aerial Vehicles Using $\mathcal{L}_1$ Adaptive Augmentation of Commercial Autopilots," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 2, 2010, pp. 550–564.

[43] Choe, R., "Distributed Cooperative Trajectory Generation for Multiple Autonomous Vehicles Using Pythagorean Hodograph Bézier Curves," Ph.D. thesis, University of Illinois at Urbana–Champaign, 2017.

[44] Bernstein, S., "Demonstration of a Theorem of Weierstrass Based on the Calculus of Probabilities," *Communications of the Kharkov Mathematical Society*, Vol. 13, 1912, pp. 1–2. (in Russian).

[45] Farouki, R. T., *Pythagorean-Hodograph Curves*, Springer, 2008.

[46] Chang, J.-W., Choi, Y.-K., Kim, M.-S., and Wang, W., "Computation of the Minimum Distance between Two Bézier Curves/Surfaces," *Computers & Graphics*, Vol. 35, No. 3, 2011, pp. 677–684.

[47] Farouki, R. T., and Goodman, T. N. T., "On the Optimal Stability of the Bernstein Basis," *Mathematics of Computation*, Vol. 65, No. 216, 1996, pp. 1553–1566.

[48] Nishita, T., Sederberg, T. W., and Kakimoto, M., "Ray Tracing Trimmed Rational Surface Patches," *Computer Graphics*, Vol. 24, No. 4, 1990, pp. 337–345.

[49] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S., "A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space," *IEEE Journal on Robotics and Automation*, Vol. 4, No. 2, 1988, pp. 193–203.

[50] Cameron, S., "Enhancing GJK: Computing Minimum and Penetration Distances Between Convex Polyhedra," *International Conference on Robotics and Automation*, Vol. 4, Albuquerque, NM, USA, 1997, pp. 3112–3117.

[51] Farouki, R. T., Giannelli, C., Manni, C., and Sestini, A., "Identification of spatial PH quintic Hermite Interpolants with Near-optimal Shape Measures," *Computer Aided Geometric Design*, Vol. 25, No. 4, 2008, pp. 274–297.

[52] Bastl, B., Bizzarri, M., Krajnc, M., Lávička, M., Slabá, K., Šír, Z., Vitrih, V., and Agar, E., "$C^1$ Hermite Interpolation with Spatial Pythagorean-hodograph Cubic Biarcs," *Journal of Computational and Applied Mathematics*, Vol. 257, 2014, pp. 65–78.

[53] Šír, Z., and Jüttler, B., "$C^2$ Hermite Interpolation by Pythagorean Hodograph Space Curves," *Mathematics of Computation*, Vol. 76, No. 259, 2007, pp. 1373–1391.

[54] Bastl, B., Bizzarri, M., Ferjančič, K., Kovač, B., Krajnc, M., Lávička, M., Michálková, K., Šír, Z., and Žagar, E., "$C^2$ Hermite Interpolation by Pythagorean-hodograph Quintic Triarcs," *Computer Aided Geometric Design*, Vol. 31, No. 7, 2014, pp. 412–426.

[55] van den Bergen, G., *Collision Detection in Interactive 3D Environments*, Morgan Kaufmann, 2003.

[56] Wang, C.-M., Chung Hsien, C., Hwang, N.-C., and Tsai, Y.-Y., "A Novel Algorithm for Sampling Uniformly in the Directional Space of a Cone," *IEICE Transactions Fundamentals*, Vol. 89-A, No. 9, 2006, pp. 2351–2355.

[57] Chang, J.-W., Choi, Y.-K., Kim, M.-S., and Wang, W., "Computation of the Minimum Distance Between Two Bézier Curves/Surfaces," *Computers & Graphics*, Vol. 35, No. 3, 2011, pp. 677–684.

[58] LaValle, S. M., and James J. Kuffner, J., "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, Vol. 20, No. 5, 2001, pp. 378–400.

[59] Amato, N. M., and Wu, Y., "A Randomized Roadmap Method for Path and Manipulation Planning," *International Conference on Robotics and Automation*, Vol. 1, Minneapolis, MN, USA, 1996, pp. 113–120.