sensors

Article

A Distributed Algorithm for Real-Time Multi-Drone Collision-Free Trajectory Replanning

Bahareh Sabetghadam *, Rita Cunha and António Pascoal

Laboratory of Robotics and Engineering Systems (LARSyS), ISR/IST, University of Lisbon, 1649-004 Lisbon, Portugal; rita@isr.ist.utl.pt (R.C.); antonio@isr.ist.utl.pt (A.P.)
* Correspondence: bsabetghadam@isr.ist.utl.pt

Abstract: In this paper, we present a distributed algorithm to generate collision-free trajectories for a group of quadrotors flying through a common workspace. In the setup adopted, each vehicle replans its trajectory, in a receding horizon manner, by solving a small-scale optimization problem that only involves its own individual variables. We adopt the Voronoi partitioning of space to derive local constraints that guarantee collision avoidance with all neighbors for a certain time horizon. The obtained set of collision avoidance constraints explicitly takes into account the vehicle's orientation to avoid infeasiblity issues caused by ignoring the quadrotor's rotational motion. Moreover, the resulting constraints can be expressed as Bézier curves, and thus can be evaluated efficiently, without discretization, to ensure that collision avoidance requirements are satisfied at any time instant, even for an extended planning horizon. The proposed approach is validated through extensive simulations with up to 100 drones. The results show that the proposed method has a higher success rate at finding collision-free trajectories for large groups of drones compared to other Voronoi diagram-based methods.

Keywords: distributed trajectory generation; Voronoi diagram; multi-drone applications; real-time replanning

Citation: Sabetghadam, B.; Cunha, R.; Pascoal, A. A Distributed Algorithm for Real-Time Multi-Drone Collision-Free Trajectory Replanning. *Sensors* **2022**, *22*, 1855. https://doi.org/10.3390/s22051855

Academic Editor: Sašo Blažič

Received: 26 January 2022
Accepted: 24 February 2022
Published: 26 February 2022

1. Introduction

Trajectory generation is a key element for the execution of complex autonomous vehicle missions. It can be defined as the computational problem of finding a valid trajectory that guides a vehicle from an initial state to a given final state in an environment with static and/or moving obstacles. In most applications, the main concern, rather than just finding a feasible trajectory between the initial and final states, is to obtain the optimal trajectory with respect to a certain objective function. In such a setting, trajectory generation is formulated as an optimization problem with a cost function that quantifies the accomplishment of mission goals and objectives, and different types of constraints to ensure safety and feasibility of resulting trajectories.

With rapid advances in communication and computational technology, autonomous vehicles continue to take part in more complex missions, and even engage in teams of collaborating vehicles to take on increasingly demanding tasks. This necessitates incorporating intervehicle collision avoidance constraints in the optimization problem to guarantee that generated trajectories for a group of vehicles sharing a common workspace are collision-free. Therefore, for a large group of vehicles, the optimization problem would involve a large number of constraints and decision variables, and the computational cost of solving it centrally can be prohibitively high. To reduce the computational complexity, a multitude of distributed schemes, reviewed below, have been proposed for decomposing the optimization problem into smaller sub-problems that can be solved locally by each vehicle. The major challenge is to ensure that local decisions do also satisfy the coupling collision avoidance constraints. This is mainly addressed by exchanging information among the vehicles on their current states, future input sequences, etc. Depending on the communication strategy, the sub-problems might be solved sequentially or concurrently, with possibly several iterations of optimization and communication to achieve the required performance.

In [1], the collision avoidance constraint, usually expressed in terms of the two-norm of a relative position vector, is approximated by a set of linear constraints. The sub-problem for each vehicle is then formulated as a mixed-integer linear programming (MILP) that includes the vehicle's individual variables as well as variables of a subset of neighbors. This enables cooperation among vehicles by allowing a vehicle to make feasible perturbations to neighboring vehicles' decisions. The sub-problems are solved sequentially by each vehicle, and the algorithm iterates over the group of vehicles until convergence, during each cycle of a model predictive control (MPC) scheme.

Sequential convex programming (SCP)-based methods have also been used for solving distributed multiple vehicle trajectory generation problems [2,3]. The work in [4] addresses the infeasiblity of intermediate problems in decoupled-SCP methods, arising from convex approximation of collision avoidance constraints, i.e., linearizing them, and proposes incremental SCP (iSCP), which tightens collision constraints incrementally. Compared to sequential approaches in [5–7], that cast the trajectory of anterior vehicles as dynamic obstacles for a posterior vehicle, the methods proposed in [1,4] result in less constrained intermediate problem and faster convergence rate, yet, similar to most MPC-SCP-based methods, they would require the vehicles to exchange a full representation of their decisions to neighboring vehicles over a communication network.

The synchronous approach in [8] extends the distributed MPC (dMPC) scheme in [9] for formation control, based on alternating direction method of multipliers (ADMM), to problems with intervehicle collision avoidance constraints. These constraints are decoupled using separating hyperplanes, which enforces each vehicle to stay within one half-space of a time-varying plane over a certain time horizon. The resulting sub-problems are solved simultaneously by vehicles, while the normal vector and offset shared between a vehicle and a neighbor, for characterizing their separating hyperplane, are updated at each cycle of the dMPC, using the interchanged information about generated trajectories at the previous cycle.

In the decentralized trajectory planner proposed in [10], vehicles replan their trajectories asynchronously, independent of the planning status of other vehicles. At each iteration, a vehicle considers trajectories assigned to neighboring vehicles as constraints, and solves an optimization problem including as decision variables the normal and offset of planes that separate the outer polyhedral representation of its trajectory and those of its neighbors. A check–recheck scheme is then performed to ensure that the generated trajectory does not collide with trajectories other vehicles have committed to during the optimization time. Therefore, to guarantee deconfliction between vehicles, the planner requires a vehicle to broadcast its computed trajectory to its neighboring vehicles at the end of each replanning iteration.

The on-demand approach to local collision avoidance, proposed in [11], imposes constraints only at specific time instances when collisions between a vehicle and its neighbors are predicted. Predicting collisions along a time horizon, however, relies on an accurate knowledge of the neighbors' future actions which must be communicated at every sampling time. The dMPC scheme in [12] for distributed trajectory generation is based on this predict–avoid paradigm and an event-triggered replanning strategy, and has been shown to result in less conservative trajectories, but at the cost of voiding the collision avoidance guarantees for all time instances over the horizon. To capture the downwash effect of quadrotor's propellers, the collision avoidance constraint in [12] is modified with a diagonal scaling matrix, which approximates the quadrotor body with a translating ellipsoid elongated along the vertical axis, yet it ignores the quadrotor's rotational motion.

Reciprocal velocity obstacle (RVO) and its variants have been widely used in distributed collision avoidance [13–17]. At each time step, RVO [13] builds the set of all relative velocities, leading to a collision between a vehicle and its neighbors, and chooses a new constant velocity outside this set, and closest to the desired value, to avoid collisions. Therefore, RVO requires the position and velocity information to be communicated, or sensed, between nearby neighbors. Other variants, such as acceleration velocity obstacle (AVO), which addresses the instantaneous change of velocity in RVO by taking into account acceleration constraints, need further information such as acceleration to be interchanged. Reciprocally-rotating velocity obstacle (RRVO) [18] uses rotation information to mitigate deadlocks caused by symmetries of representing vehicles with translating discs in RVO. It relies on the assumption that neighbors may rotate equally (or equally opposite), bounded by a maximum value, to compute an approximation of swept areas for rotating polygon-shaped vehicles, and uses them for constructing velocity obstacles. A new velocity and rotation is then selected at each time step to avoid collisions.

Another approach to distributed collision avoidance is to construct the Voronoi diagram of the group of vehicles and generate the trajectory for each vehicle so that it is entirely within the vehicle's Voronoi cell [19–22]. Since Voronoi cells do not overlap, it can be guaranteed that the generated trajectories are collision-free. To consider the physical size of a vehicle, the modified Voronoi cell used in [23,24] retracts boundary hyperplanes of the general Voronoi cell by a safety radius for disc-shaped vehicles. At each sampling time, upon receiving the relative position information, trajectories are replanned to conform to the updated Voronoi diagram. The resulting sub-problems can be solved simultaneously, in a receding horizon manner, until the vehicles reach their final positions. The Voronoi-based approaches only require the vehicles to know relative positions to neighboring vehicles, and therefore are well suited to applications where vehicles only have relative position sensing and no communication network [25].

In this paper we develop a distributed trajectory generation framework, with low computation and communication demands, for multiple quadrotors flying in (relatively) close proximity to each other. We specifically address the shortcomings of approximating the drone body with a disc (or sphere) for generating feasible collision-free trajectories for large groups of drones. A sphere model, used in most existing distributed collision avoidance schemes, may be overly conservative in confined spaces since it invalidates trajectories whose feasibility depends on the consideration of the flight attitude. Instead, we model the drone body with an ellipsoid, and employ the Voronoi partitioning of space to derive local collision avoidance constraints that take into account the drone's real size and orientation. The same approach can be integrated into other distributed schemes that utilize separating hyperplanes for decoupling collision avoidance constraints. Yet the main reason for adopting Voronoi diagram is that using time-invariant boundary hyperplanes determined prior to solving a sub-problem, despite being more conservative, can significantly reduce communication and computational load, allowing for higher replanning rates. Incorporating the resulting set of constraints into sub-problems, solved by each vehicle, allows finding collision-free trajectories for guiding a group of drones through confined spaces by proper adjustment of attitude angles. In addition, the obtained set of constraints can be expressed as Bézier curves, and hence can be efficiently evaluated to guarantee that intervehicle collision avoidance requirements are met at any instant of time even over a long planning horizon.

In the proposed synchronous distributed scheme, each vehicle uses the position information of its neighbors, updated at each sampling time, and solves a sub-problem to generate its trajectory inside (a subset of) its Voronoi cell towards the closest point (in the cell) to its goal position. We present an efficient method to compute this point, which is needed to appropriately define the terminal constraint and cost in the sub-problem. A sequence of sub-problems are then solved in a receding-horizon manner until the vehicles reach their goal positions. The simulation results show that the proposed method has a higher success rate at finding collision-free trajectories for larger groups of quadrotors compared to other Voronoi diagram-based methods. In addition, it can effectively reduce the total flight time required to perform point-to-point maneuvers. Furthermore, the computation time of generating those trajectories satisfies timing constraints imposed by real-time applications.

The rest of this paper is organized as follows: In Section 2 we formulate the optimization sub-problem solved by each vehicle. In Section 2.1 we study the differentially flat system describing the drone equations of motion, and parameterize trajectories with Bézier curves. We derive the set of local collision avoidance constraints in Section 2.2, and present an efficient algorithm for finding the closest point in a Voronoi polytope to a goal position in Section 2.3. In Section 3 we obtain the continuity conditions between two adjacent Bézier curve segments, and present a method for evaluating inequalities in Bézier form without discretization. Finally, we provide simulation results in Section 4.

2. Problem Formulation

The multiple vehicle trajectory generation problem addressed in this paper can be defined as finding optimal trajectories that act as references to guide a group of vehicles from their initial positions to some desired final positions. The generated trajectories should jointly minimize a cost function, corresponding to the accomplishment of mission goals and objectives, and satisfy a set of local and coupling constraints, so that they are dynamically-feasible and collision-free. For N_v vehicles, this problem can be formulated as the following optimal control problem.

$$\min_{\substack{\mathbf{u}_i(.)\\ i\in[N_v]}} \sum_{i\in[N_v]} J(\mathbf{x}_i(.),\mathbf{u}_i(.)) \tag{1}$$

$$\begin{aligned}
s.t.\quad & \dot{\mathbf{x}}_i(t) = f(\mathbf{x}_i(t),\mathbf{u}_i(t)) && \text{(Dynamics)}\\
& \mathbf{x}_i(0) = \mathbf{x}_{i,0} && \text{(Initial state)}\\
& \mathbf{x}_i(t_f) = \mathbf{x}_{i,f} && \text{(Final state)}\\
& c(\mathbf{x}_i(t),\mathbf{x}_j(t)) \le 0 \quad j\in[N_v]\backslash\{i\} && \text{(collision avoidance)}\\
& \mathbf{x}_i(t) \in \mathcal{X}_i && \text{(State Constraints)}\\
& \mathbf{u}_i(t) \in \mathcal{U}_i && \text{(Input constraints)}
\end{aligned}$$

where $[N_v] = \{1,\dots,N_v\}$. The cost to be minimized is the sum of the vehicles' individual costs, J, given by the functional,

$$J[\mathbf{u}_i(.)] = \int_0^{t_f} L(\mathbf{x}_i,\mathbf{u}_i)dt \tag{2}$$

where $\mathbf{x}_i(t) \in \mathbb{R}^{n_x}$ and $\mathbf{u}_i(t) \in \mathbb{R}^{n_u}$ are the state and the input vectors of the vehicle's model described by an ODE, and $\mathbf{x}_{i,0}$ and $\mathbf{x}_{i,f}$ are the initial and final values of the state of the i-th vehicle, respectively. $\mathcal{X}_i$ and $\mathcal{U}_i$ denote the set of admissible states and inputs for the i-th vehicle derived from limits imposed by vehicle dynamics and the surrounding environment.

In order to reduce the computational complexity of solving (1) for larger N_v with increased numbers of constraints and variables, one can divide the problem into a set of small-scale sub-problems. Here, the sub-problems are formulated such that each involves only a vehicle's individual costs and constraints, and hence can be solved independently by the vehicle. The sub-problems must include constraints to ensure that the trajectory generated locally by a vehicle does satisfy the coupling collision avoidance constraints.

The key idea to ensure intervehicle collision avoidance is to decompose the space into non-overlapping regions, provided by a Voronoi diagram, and generate the trajectory for each vehicle such that it is entirely within its partition. The Voronoi diagram is updated at each sampling time according to the relative positions of vehicles, and a sequence of sub-problems is solved in a receding horizon manner until the vehicles reach their final positions. For the i-th vehicle, the problem that has to be solved at the time instant t_k can be formulated as

$$\begin{aligned}
\min_{\mathbf{x}_{i,k}(.),\mathbf{u}_{i,k}(.)} \quad & J(\mathbf{x}_{i,k}(.),\mathbf{u}_{i,k}(.)) && (3)\\
s.t. \quad & \dot{\mathbf{x}}_{i,k}(t) = f(\mathbf{x}_{i,k}(t),\mathbf{u}_{i,k}(t)) && \text{(Dynamics)}\\
& \mathbf{x}_{i,k}(t_k) = \hat{\mathbf{x}}_{i,k} && \text{(Initial state)}\\
& \mathbf{x}_{i,k}(t) \in \mathcal{C}_{i,k}(\bar{\mathbf{x}}_k) && \text{(collision avoidance)}\\
& \mathbf{x}_{i,k}(t) \in \mathcal{X}_{i,k} && \text{(State Constraints)}\\
& \mathbf{u}_{i,k}(t) \in \mathcal{U}_{i,k} && \text{(Input constraints)}
\end{aligned}$$

where $\mathbf{x}_{i,k}(t)$ and $\mathbf{u}_{i,k}(t)$ are the state and the input profiles of the vehicle over the time interval $[t_k, t_k + t_h]$, with t_h being the planning horizon, and $\hat{\mathbf{x}}_{i,k}$ denotes its state at the time instant t_k. The cost function in the above sub-problem is modified as

$$J[\mathbf{u}_{i,k}(.)] = \int_{t_k}^{t_k+t_h} L(\mathbf{x}_{i,k},\mathbf{u}_{i,k})dt + \phi(\mathbf{x}_{i,k}(t_k+t_h)) \tag{4}$$

where the second term is added to penalize the distance, at $t_k + t_h$, to the point in the Voronoi partition that is closest to the goal position.

In the optimization problem (3), $\mathcal{C}_{i,k}$ may denote the Voronoi partition assigned to the i-th vehicle. The Voronoi diagram is updated for each sub-problem according to the vehicles' configuration at each time instant t_k, i.e., $\bar{\mathbf{x}}_k = \{\hat{\mathbf{x}}_{j,k}\}_{j\in[N_v]}$. Since Voronoi partitions are disjoint and the assigned trajectory to each vehicle for the time horizon t_h is contained within its partition, it can be guaranteed that there is no collision between the trajectories over the time interval $[t_k, t_k + t_h]$.

The distributed trajectory generation framework is summarized in Algorithm 1. In Section 2.2, we study the Voronoi diagram for a group of vehicles and modify $\mathcal{C}_{i,k}$ to explicitly take into account the orientation while generating collision-free trajectories for multiple drones.

Algorithm 1 Distributed Trajectory Generation Framework

```
1:  $k = 0$
2:  $\hat{\mathbf{x}}_{i,0} \leftarrow$ Initial position of the $i$-th vehicle
3:  **repeat**
4:      Receive position information from neighbors
5:      Broadcast own position to neighbors
6:      Update Voronoi partition
7:      Compute the closest point in the Voronoi partition to
        the goal position      ▷ Section 2.3
8:      Set the cost function (4)
9:      Set the constraints    ▷ Sections 2.2 and 3.1
10:     Solve the optimization sub-problem
11: **until** $\hat{\mathbf{x}}_{i,k} = \mathbf{x}_{i,f}$.
```

2.1. Quadrotor Model

In this paper, the simplified quadrotor equations of motion are described by

$$m\ddot{\mathbf{p}} = mg\mathbf{e}_3 + f, \tag{5}$$

where $\mathbf{p} \in \mathbb{R}^3$ is the position and m is the mass of the quadrotor. In addition, $g = 9.8\,\frac{\text{m}}{\text{s}^2}$ is the gravitational acceleration, and $\mathbf{e}_3 = [0\;0\;1]^T$. The first term on the right-hand side of (5) is gravity in the $\mathbf{z}_I$ direction, and the second term, $f \in \mathbb{R}^3$, is the thrust force aligned with the body's $\mathbf{z}$-axis.

$$f = -T^I\mathbf{z}_B \tag{6}$$

where $T \in \mathbb{R}$ is the net thrust, ${}^I\mathbf{z}_B = R^B\mathbf{z}_B = R\mathbf{e}_3$ is the body frame $\mathbf{z}$-axis expressed in $\{I\}$, and $R \equiv {}^I_BR \in SO(3)$ is the rotation matrix from the body frame $\{B\}$, centered at the quadrotor's center of gravity, to the fixed inertial frame $\{I\}$. For simplicity, we drop the superscript ^{I} and consider $\mathbf{z}_B = {}^I\mathbf{z}_B$. Figure 1 is a graphical representation of the quadrotor and the associated reference frames.

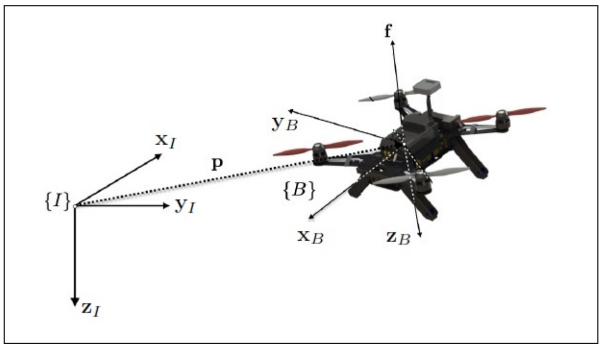


Figure 1. The quadrotor reference frames.

Trajectory Parametrization

The quadrotor dynamics (5) with the four inputs is differentially flat [26], and therefore the state and the input of the system can be expressed as functions of the flat outputs and a finite number of its derivatives. The position vector together with the yaw angle can be selected as flat outputs of the system. Here, $\mathbf{p} \in \mathbb{R}^3$ is parameterized as a Bézier curve, given by

$$\mathbf{p}(\tau) = \sum_{l=0}^{n} \bar{p}_l B_{l,n}(\tau), \tag{7}$$

where $\bar{p}_l \in \mathbb{R}^3$ are the control points, $B_{l,n}$ are Bernstein basis polynomials of degree n, and $\tau \in [0,1]$ is defined as

$$\tau = \frac{t}{t_f} \tag{8}$$

The linear velocity, $\mathbf{v} = \dot{\mathbf{p}}$, and linear acceleration, $\mathbf{a} = \ddot{\mathbf{p}}$, can be expressed as parametric Bézier curves through the first and second derivative of $\mathbf{p}$ with respect to time, yielding

$$\begin{aligned} \mathbf{v}(\tau) &= \sum_{l=0}^{n-1} \bar{v}_l B_{l,n-1}(\tau) \\ \mathbf{a}(\tau) &= \sum_{l=0}^{n-2} \bar{a}_l B_{l,n-2}(\tau) \end{aligned} \tag{9}$$

where the control points $\bar{v}_l$ and $\bar{a}_l$ are obtained as

$$\begin{aligned} \bar{v}_l &= \frac{n}{t_f}(\bar{p}_{l+1} - \bar{p}_l) & l &= 0, \dots, n-1 \\ \bar{a}_l &= \frac{n(n-1)}{t_f^2}(\bar{p}_{l+2} - 2\bar{p}_{l+1} + \bar{p}_l) & l &= 0, \dots, n-2 \end{aligned} \tag{10}$$

The thrust T and rotation matrix R can also be expressed as functions of the flat output and its derivatives. The net thrust T can be written as

$$T = m\|\ddot{\mathbf{p}} - g\mathbf{e}_3\|. \tag{11}$$

Assuming that the rotation matrix $R = [\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B]$ is parameterized by the *Z*-*Y*-*X* Euler angles $\boldsymbol{\lambda} = [\phi, \theta, \psi]^T$ as

$$R = R_z(\psi)R_y(\theta)R_x(\phi), \tag{12}$$

then the columns of the rotation matrix are extracted from

$$\mathbf{z}_B = \frac{\mathbf{g}\mathbf{e}_3 - \ddot{\mathbf{p}}}{\|\mathbf{g}\mathbf{e}_3 - \ddot{\mathbf{p}}\|} \quad \mathbf{x}_B = \frac{\mathbf{r} \times \mathbf{z}_B}{\|\mathbf{r} \times \mathbf{z}_B\|}, \quad \mathbf{y}_B = \mathbf{z}_B \times \mathbf{x}_B \tag{13}$$

where the unit vector $\mathbf{r}$ is defined as

$$\mathbf{r} = [-\sin\psi, \cos\psi, 0]^T \tag{14}$$

The above equations declare that the vehicle's orientation can be fully determined from the second derivative of the trajectory and the yaw angle. As mentioned before, the yaw angle ψ is a component of the flat output, and therefore it can be controlled independently without affecting the trajectory generation. Using the differential flatness property of the system, trajectories consistent with dynamics can be planned in the space of flat outputs, where (5) is trivially satisfied and the original input and state constraints are transformed into constraints on the flat output and its derivatives.

2.2. Collision Avoidance

In this section, we present a Voronoi diagram-based approach to decoupling inter-vehicle collision avoidance constraints. Although the presence of obstacles, interpreted as non-decision-making agents, is not explicitly considered here, incorporating vehicle–obstacle collision avoidance constraints into the problem simply amounts to taking into account the obstacles' position when updating the Voronoi partition (step 6 of Algorithm 1).

The widely used approach in the literature to avoiding collisions with obstacles in the environment is to model the drone body as a sphere with radius $r_{\mathbf{D}}$, and then simply building the collision-free space, $\mathcal{C}_{free}$, by inflating the obstacles with a factor $r_{\mathbf{D}}$. As a result, collision-free trajectories can be obtained by enforcing the vehicle, which is now treated as a point in the space, to be inside $\mathcal{C}_{free}$ [27]. Considering now the collision avoidance between the i-th and j-th drones, the corresponding constraint can be derived similarly by

$$\|\mathbf{p}_i - \mathbf{p}_j\| \geq 2r_{\mathbf{D}} \tag{15}$$

where $\|.\|$ denotes the Euclidean distance. Ignoring the real shape and orientation of the drone, and approximating its body with a sphere, invalidates trajectories that are feasible upon considering the flight attitude. For this reason, the above approach might be too conservative for trajectory generation in confined spaces and can even fail to find feasible collision-free trajectories when multiple drones are involved.

Approximating the drone body with an ellipsoid, whose principal axes are aligned with the body frame axes, allows considering the drone orientation while inspecting for collisions against other vehicles. For the i-th drone, the ellipsoid, E_i, centered at the drone position $\mathbf{p}_i$, is given by

$$E_i \equiv \{\mathbf{p} \in \mathbb{R}^3 | \mathbf{p} = \mathbf{p}_i + R_i\Lambda\mathbf{w}, \|\mathbf{w}\| \leq 1\} \tag{16}$$

where Λ is a 3×3 diagonal matrix of the form

$$\Lambda = \begin{bmatrix} r_{\mathbf{D}} & 0 & 0 \\ 0 & r_{\mathbf{D}} & 0 \\ 0 & 0 & h_{\mathbf{D}} \end{bmatrix} \tag{17}$$

with $r_{\mathbf{D}}$ and $h_{\mathbf{D}}$ being the lengths of the principle semi-major and semi-minor axes, respectively. (See Figure 2).

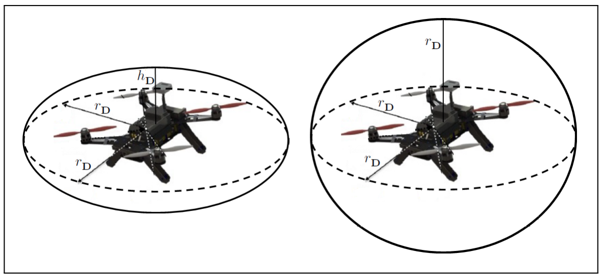


Figure 2. The quadrotor body can be represented as a sphere with radius $r_{\mathbf{D}}$ (**right**), or an ellipsoid aligned with the axes of the body frame (**left**). Approximating the drone body with an ellipsoid allows considering the quadrotor's rotational motion.

As proposed in [28], collision avoidance constraints for two ellipsoid-shaped drones can be derived using separating hyperplanes. Denoting by $\mathbf{a} \in \mathbb{R}^3$ and $b \in \mathbb{R}$ the normal vector and offset of a hyperplane, respectively, the separating hyperplane for E_i and E_j, defined as $H \equiv \{\mathbf{p}|\mathbf{a}^T\mathbf{p} - b = 0\}$, must satisfy

$$\begin{aligned} \mathbf{a}^T\mathbf{p} - b \leq 0 \quad \forall \mathbf{p} \in E_i \\ \mathbf{a}^T\mathbf{p} - b > 0 \quad \forall \mathbf{p} \in E_j \end{aligned} \tag{18}$$

Since

$$-\|\Lambda R^T\mathbf{a}\| \leq \mathbf{a}^T R\Lambda\mathbf{w} \leq \|\Lambda R^T\mathbf{a}\| \tag{19}$$

The set of inequalities (18) holds if, and only if,

$$\begin{aligned} \mathbf{a}^T\mathbf{p}_i - b \geq & \qquad \|\Lambda R_i^T\mathbf{a}\| \\ \mathbf{a}^T\mathbf{p}_j - b \leq & \quad -\|\Lambda R_j^T\mathbf{a}\| \end{aligned} \tag{20}$$

Satisfying the set of constraints (20) will guarantee that there is no collision between the two ellipsoids E_i and E_j associated with the i-th and j-th drones, respectively.

For multiple vehicle trajectory generation, collision avoidance constraints, either in the form of the inequality constraint (15), for spheres, or the set of constraints (20), for ellipsoids, must be incorporated in the optimization problem for each pair of vehicles. As the number of vehicles involved in a mission grows, the resulting increase in the number of constraints would inevitably exacerbate the computational issues of finding collision-free trajectories in a centralized manner.

Distributed Collision Avoidance

Here, we propose a distributed approach to collision avoidance which takes into account the shape and orientation of a drone. The presented approach uses Voronoi partitioning of space and generates the trajectory of each vehicle such that it is entirely within (a subset of) the vehicle's Voronoi cell for a time interval t_h. Since Voronoi cells are pairwise disjoint, and each vehicle only moves inside its Voronoi cell, intervehicle collision avoidance is guaranteed for all future time before t_h.

Each Voronoi cell in an n-dimensional space is a convex polytope bounded by a number of $(n-1)$-dimensional convex polytopes. For a group of vehicles in three-dimensional space, the general Voronoi cell of the i-th vehicle is defined as

$$\mathcal{V}_i = \{\mathbf{p} \in \mathbb{R}^3 | \mathbf{p}_{ij}{}^T(\mathbf{p} - \frac{1}{2}(\mathbf{p}_i + \mathbf{p}_j)) \leq 0, \forall j \in [N_v] \backslash \{i\}\} \tag{21}$$

where $\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i$, and $\mathbf{p}_i$ and $\mathbf{p}_j$ are the position of the i-th and j-th vehicles at the current time instant. Note that $\mathcal{V}_i$ is the intersection of half-spaces corresponding to hyperplanes with $a = \mathbf{p}_{ij}$ and $b = \frac{1}{2}\mathbf{p}_{ij}^T(\mathbf{p}_i + \mathbf{p}_j)$. An arbitrary point in $\mathcal{V}_i$ is closer to the i-th vehicle than any other vehicle [22], i.e.,

$$\|\mathbf{p} - \mathbf{p}_i\| \leq \|\mathbf{p} - \mathbf{p}_j\|, \quad \forall \mathbf{p} \in \mathcal{V}_i \quad \& \quad j \neq i \tag{22}$$

The boundary of the Voronoi cell, $\partial\mathcal{V}_i$, is the union of multiple faces, each of which include points in the space that are equidistant to the i-th vehicle and a neighboring vehicle.

In order to account for the size of a vehicle, the buffered Voronoi cell (BVC) proposed in [24] retracts the boundary of the general Voronoi cell by a safety radius, so that if the vehicle's center is inside the BVC, its body, approximated by a sphere of radius $r_\mathbf{D}$, will be entirely within its Voronoi cell. The BVC of the i-th vehicle, denoted by $\bar{\mathcal{V}}_i$, is defined as

$$\bar{\mathcal{V}}_i = \{\mathbf{p} \in \mathbb{R}^3 | \quad \mathbf{p}_{ij}^T(\mathbf{p} - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) + r_\mathbf{D}\|\mathbf{p}_{ij}\| \leq 0, \forall j \in [N_v]\backslash\{i\}\} \tag{23}$$

It can be easily shown that BVC is a subset of the general Voronoi cell, i.e., $\bar{\mathcal{V}}_i \subset \mathcal{V}_i$. In addition, for any two points $\mathbf{p}' \in \bar{\mathcal{V}}_i$ and $\mathbf{q}' \in \bar{\mathcal{V}}_j$, $\|\mathbf{p}' - \mathbf{q}'\| \geq 2r_\mathbf{D}$ holds. Therefore, the vehicles are guaranteed to avoid collisions due to the buffer region of $r_\mathbf{D}$ along $\partial\mathcal{V}_i$. Figure 3 shows the Voronoi diagram for 10 drones in a collision-free configuration and the BVC for two adjacent drones.

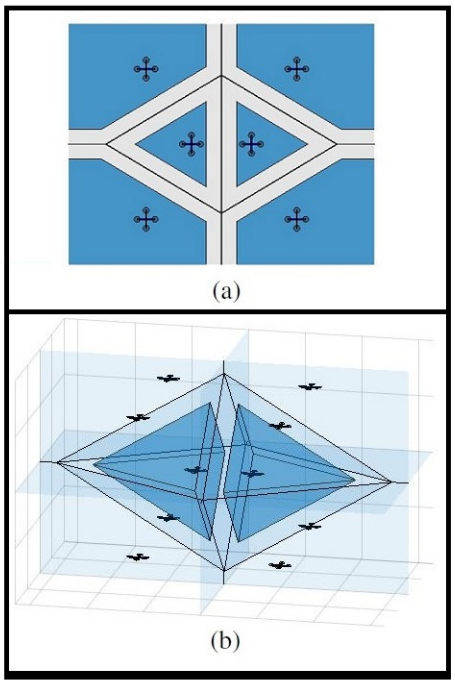


Figure 3. (**a**) The Voronoi diagram for six drones in 2D space. The Voronoi boundary edges are shown with solid black lines, and the buffered Voronoi cells are shaded in dark blue. (**b**) The Voronoi diagram for 10 drones in a collision-free configuration in 3D space. The Voronoi boundary $\partial\mathcal{V}$ is shaded in light blue, and the buffered Voronoi cells $\bar{\mathcal{V}}$ for two neighboring drones in the center are shown in dark blue.

The BVC is defined based on a symmetrical approximation of the vehicle's body with a translating disc. In order to reduce the conservatism and avoid infeasibility issues due to ignoring the real shape and orientation of the vehicle, we approximate the drone with an ellipsoid (16), and bearing in mind that

$$R\Lambda^2R^T = r_\mathbf{D}^2 I + (h_\mathbf{D}^2 - r_\mathbf{D}^2)\mathbf{z}_B\mathbf{z}_B^T, \tag{24}$$

we propose $\mathcal{C}_i$ in problem (3) to be defined as

$$\mathcal{C}_i = \{(\mathbf{p}, \ddot{\mathbf{p}}) \in \mathbb{R}^6 | \mathbf{p}_{ij}^T(\mathbf{p} - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) + \|\Lambda R^T \mathbf{p}_{ij}\| \leq 0, \forall j \in [N_v] \backslash \{i\}\} \tag{25}$$

If the trajectory of the i-th drone $\mathbf{p}_i(t)$ satisfies the above set of local collision avoidance constraints for all $t \in [t_k, t_k + t_h]$, then the ellipsoid representing the drone body is within the Voronoi cell for the entire time horizon; that is, the ellipsoid centered at $\mathbf{p}_i$ and aligned with the columns of R_i does not intersect the Voronoi boundary, stated mathematically

$$\|\partial E_i - \partial \mathcal{V}_i\| \geq 0 \tag{26}$$

Noting that

$$h_{\mathbf{D}} \|\mathbf{p}_{ij}\| \leq \|\Lambda R^T \mathbf{p}_{ij}\| \leq r_{\mathbf{D}} \|\mathbf{p}_{ij}\|, \tag{27}$$

it can be induced that

$$\bar{\mathcal{V}}_i(r_{\mathbf{D}}) \subset \text{proj}_{XYZ}\mathcal{C}_i \subset \bar{\mathcal{V}}_i(h_{\mathbf{D}}) \subset \mathcal{V}_i \tag{28}$$

where $\text{proj}_{XYZ}\mathcal{C}_i$ is the projection of $\mathcal{C}_i$ onto the three-dimensional subspace spanned by $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$.

Therefore, incorporating (25) into the optimization problem (3) will ensure that the generated trajectories are collision-free while alleviating infeasibility problems by taking orientations into account. Also, since $\mathbf{z}_B$ is fully obtained from $\ddot{\mathbf{p}}$ (13), the above set of local collision avoidance constraints can be expressed as constraints imposed on Bézier curves. Later, we present an efficient method for evaluating inequalities in Bézier form.

2.3. Finding the Closest Point to the Goal Position

As explained above, at each time instant the Voronoi cell $\mathcal{V}_i$ is updated according to the relative position of the i-th vehicle to other vehicles. The optimization problem (3) is then solved to generate a trajectory, for a time horizon t_h, that guides the vehicle towards the point in the cell closest to the goal position. This process is repeated until the vehicle reaches its final position. At each sampling time, the closest point must be found prior to solving the trajectory generation problem. Therefore, having an efficient scheme for finding the closest point is critically important to avoid long computational delays between updating the Voronoi cell and replanning the trajectory.

The point in a convex polytope that is closest to a query point $\mathbf{q}$ is either $\mathbf{q}$ itself or a point on the boundary of the polytope. A naive way to find the closest point in a convex polytope in a three-dimensional space, represented by $P = (\mathbb{F}, \mathbb{E}, \mathbb{V})$, where $\mathbb{F}$ is the set of faces, $\mathbb{E}$ is the set of edges, and $\mathbb{V}$ is the set of vertices, is to check the distance between $\mathbf{q} \in \mathbb{R}^3$ to all faces, edges, and vertices for finding the minimum. However, for complex polytopes, the computation time is not negligible.

The geometric approach proposed in [24] for a polygon in a two-dimensional space calculates the barycentric coordinates and an angle from the query point to the two vertices of each edge to find the closest point. Since this approach iterates over all edges, its computational complexity can significantly increase as the number of Voronoi neighbors of a vehicle increases. Here, we make use of the Gilbert–Johnson–Keerthi (GJK) distance algorithm and devise an approach that can efficiently determine whether the query point is inside the polytope, i.e., $\mathbf{q} \in P$, in which case the closest point is $\mathbf{q}$ itself. Otherwise, the presented algorithm returns the closest feature of P to $\mathbf{q}$, and the closest point can be obtained by projecting $\mathbf{q}$ onto it. The proposed approach is not limited to distance queries between a point and a polytope, and can also be used when the final constraint in (3) is relaxed to a small box or sphere around the goal position, and used in conjunction with a terminal cost term.

The GJK distance algorithm or simply GJK algorithm is an iterative algorithm that relies on a support mapping function to incrementally build simplices that are closer to the query point [29]. The algorithm has been extensively used for collision detection between generic convex shapes [30,31]. The original algorithm, however, can be used to compute the minimum distance, and also the respective pair of (closest) points, between two convex shapes [32].

In order to obtain the minimum distance between two general convex sets A and B, GJK approximates the closest point in the Minkowski difference of the two sets, $C = A - B$ to the origin, denoted by $\nu(C)$, with an iterative search. At each iteration, a simplex in C is constructed such that it is closer to the origin $\mathcal{O}$ than the simplex in the previous iteration. In $\mathbb{R}^3$, a simplex can be a point, a line, a triangle, or a tetrahedron with 1, 2, 3, and 4 affinely independent vertices.

GJK relies on the so-called support mappings to construct a new simplex. A support mapping function $s_C(\mathbf{d})$ of the convex set C maps the vector $\mathbf{d}$ to a point in the set, called the support point, according to

$$\mathbf{d}^T s_C(\mathbf{d}) = \max\{\mathbf{d}^T\mathbf{p}; \mathbf{p} \in C\} \tag{29}$$

At each iteration, a support point $\mathbf{w}_k = s_C(-\boldsymbol{\nu}_k)$ is added as a vertex to the current simplex, indicated with V_k, where $\boldsymbol{\nu}_k$ is the closest point of V_k to the origin, i.e., $\boldsymbol{\nu}_k = \nu(\text{conv}(V_k))$. V_{k+1} is then updated such that it only contains the smallest set of vertices that supports $\boldsymbol{\nu}_{k+1} = \nu(\text{conv}(V_k \cup \mathbf{w}_k))$, and earlier vertices that do not back $\boldsymbol{\nu}_{k+1}$ are discarded [30].

It is proved in [30] that in each iteration the new $\boldsymbol{\nu}$ is closer to the origin than the previous one, and thus the sequence of $\{\boldsymbol{\nu}_k\}$ converges to the closest point of C to the origin. In addition, it is shown that

$$\|\boldsymbol{\nu}_k - \nu(C)\|^2 \leq \|\boldsymbol{\nu}_k\|^2 - \boldsymbol{\nu}_k^T\mathbf{w}_k \tag{30}$$

which is used to construct the terminating condition of the GJK algorithm for general convex shapes.

The GJK algorithm, as explained above, depends heavily on the computation of $\boldsymbol{\nu}_k$ to test the termination condition and to determine the search direction for finding the support point. In each iteration of the algorithm, $\boldsymbol{\nu}_k$ must be computed with the *Johnson Distance Subalgorithm* [29] or more robust alternatives such as the signed volume method in [33]. Here, we exploit unique features of polytopes and propose a faster way to evolve simplices in the GJK algorithm without computing $\boldsymbol{\nu}_k$ in each iteration.

For polytopes, GJK arrives at the actual $\nu(C)$ in a finite number of iterations [30]. The pseudocode in Algorithm 2 describes the GJK distance algorithm for a polygon P. In order to find the support point $\mathbf{w}_k$, we employ a search direction $\mathbf{d}_k \uparrow\downarrow \boldsymbol{\nu}_k$, which is updated in each iteration of the algorithm with S1D, S2D, or S3D subroutines. To update $\mathbf{d}$ and V, these subroutines, summarized in Algorithms 3–5, inspect the Voronoi regions of the simplex for the one that contains the origin. Figure 4 shows the Voronoi regions of an m-simplex ($m = 1, 2, 3$) where the origin possibly lies. Once the Voronoi region containing the origin is found, $\mathbf{d}$ is determined as a vector from the associated vertex, edge, or face (of the simplex) pointing towards the origin.

The stop criterion for the conditional loop in Algorithm 1 is also constructed using the search direction, offered as

$$\mathbf{d}_k^T(\mathbf{w}_k - \mathbf{v}_1) \leq 0 \tag{31}$$

where $\mathbf{v}_1 \in V_k$. Considering that $\mathbf{d}_k^T(\boldsymbol{\nu}_k - \mathbf{v}_1) = 0$, we can conclude that the above criterion, for deciding whether V_k represents the closest feature of P to the origin, is equivalent to the stop criterion in [30] for determining whether $\boldsymbol{\nu}_k$ s the closest point, that is

$$\|\boldsymbol{\nu}_k\|^2 - \boldsymbol{\nu}_k^T\mathbf{w}_k \leq 0 \longleftrightarrow \mathbf{d}_k^T(\mathbf{w}_k - \mathbf{v}_1) \leq 0 \tag{32}$$

If the inequality (31) holds, then the closest feature of P to the origin can be determined from V_k. Figure 5 shows all possible outputs of Algorithm 2, which can be a vertex, an edge, or a face of P or a tetrahedron inside it, and $\nu(P)$ for each of the cases.

Algorithm 2 Compute the closest point of P to the origin

```
1:  $\mathbf{v}$ = "Arbitrary point in vert($P$)"
2:  $\mathbf{d} = -\mathbf{v}$
3:  $V = \{\mathbf{v}\}$
4:  repeat
5:      $\mathbf{w} = s_P(\mathbf{d})$;
6:      if $\mathbf{d}^T(\mathbf{w} - \mathbf{v}_1) \leq 0$ then
7:          $V$ represents the closest feature of $P$ to $\mathcal{O}$
8:          return $\nu(V)$
9:      end if
10:     $V \leftarrow V \cup \mathbf{w}$;
11:     $[V, \mathbf{d}] \leftarrow$ CallSmD($V$);[1]
12: until $|V| = 4$;
13: $P$ contains $\mathcal{O}$
14: return $\nu = \mathcal{O}$
```

[1] One of the three subroutines S1D, S2D or S3D is called in accordance with $|V|$.

A support point of a convex polytope can also be computed efficiently. For a polytope P, the support point is a vertex of P, i.e., $s_P(\mathbf{d}) \in \text{vert}(P)$, and we can take $\mathbf{w} = s_P(\mathbf{d})$ $= s_{\text{vert}(P)}(\mathbf{d})$, that is,

$$\mathbf{d}^T s_P(\mathbf{d}) = \max\{\mathbf{d}^T\mathbf{v}; \mathbf{v} \in \text{vert}(P)\} \tag{33}$$

Therefore, for polytopes, the support point can be uniquely determined by simply scanning through the list of vertices for the vertex that is the most extreme in the search direction $\mathbf{d}$. Therefore, the computation time is linear in the number of vertices of P. For complex polytopes, the vertices adjacency information and the coherence between consecutive calls to support mapping functions can be exploited to find the support point with almost constant time complexity [30].

Algorithm 3 Sub-routine for $|V| = 2$

```
1: function S1D($\{\mathbf{v}_2, \mathbf{v}_1\}$) [1]
2:     if $\mathbf{v}_1^T\mathbf{v}_{12} \geq 0$ then
3:         $V \leftarrow \{\mathbf{v}_1\}$
4:         $\mathbf{d} \leftarrow -\mathbf{v}_1$
5:     else
6:         $V \leftarrow \{\mathbf{v}_2, \mathbf{v}_1\}$
7:         $\mathbf{d} \leftarrow -\mathbf{v}_{12} \times \mathbf{v}_1 \times \mathbf{v}_{12}$
8:     end if
9: end function
```

[1] The input is the ordered list of vertices, with $\mathbf{v}_1$ being the last added element to V_k.

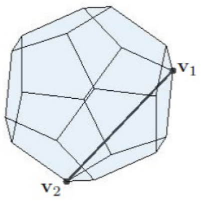
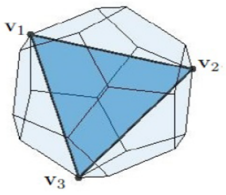
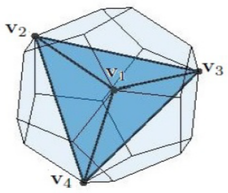
1-simplex 	$\mathcal{V}_{(1)}$ $\mathcal{V}_{(1,2)}$
2-simplex 	$\mathcal{V}_{(1)}$ $\mathcal{V}_{(1,2)}, \mathcal{V}_{(1,3)}$ $\mathcal{V}_{(1,2,3)}$
3-simplex  $\mathbf{v}_2$ $\mathbf{v}_1$ $\mathbf{v}_3$ $\mathbf{v}_4$	$\mathcal{V}_{(1)}$ $\mathcal{V}_{(1,2)}, \mathcal{V}_{(1,3)}, \mathcal{V}_{(1,4)}$ $\mathcal{V}_{(1,2,3)}, \mathcal{V}_{(1,2,4)}, \mathcal{V}_{(1,3,4)}$ $\mathcal{V}_{(1,2,3,4)}$

Figure 4. An m-simplex is linked to $2^{m+1} - 1$ Voronoi regions associated with its vertices, edges, faces, and volume. The list of 2^m Voronoi regions that can possibly contain the origin is given in this table. It should be noted that $\mathbf{v}_1$ is the latest vertex added to V.

Algorithm 4 Sub-routine for $|V| = 3$

```
 1: function S2D ($\{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$)
 2:     $\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} = \mathbf{v}_{12} \times \mathbf{v}_{13}$
 3:     if $\mathbf{v}_1^T(\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} \times \mathbf{v}_{12}) \geq 0$ then
 4:         $[V, \mathbf{d}] \leftarrow \text{S1D}(\{\mathbf{v}_2, \mathbf{v}_1\})$
 5:     else
 6:         if $\mathbf{v_1}^T(\mathbf{v}_{13} \times \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}) \geq 0$ then
 7:             $[V, \mathbf{d}] \leftarrow \text{S1D}(\{\mathbf{v}_3, \mathbf{v}_1\})$
 8:         else
 9:             if $\mathbf{v}_1^T\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} \geq 0$ then
10:                 $V \leftarrow \{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$
11:                 $\mathbf{d} \leftarrow -\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}$
12:             else
13:                 $V \leftarrow \{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$
14:                 $\mathbf{d} \leftarrow \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}$
15:             end if
16:         end if
17:     end if
18: end function
```

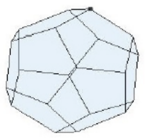
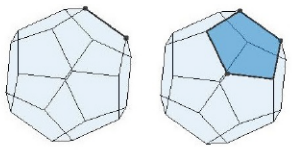
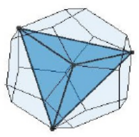
$\lvert V \rvert = 1$ 	$\nu(P) = \mathbf{v}_1$
$\lvert V \rvert = 2, 3$ 	$\nu(P) = \dfrac{\mathbf{d}^T \mathbf{v}_1}{\lVert \mathbf{d} \rVert^2} \mathbf{d}$
$\lvert V \rvert = 4$ 	$\nu(P) = \mathcal{O}$

Figure 5. Examples of the closest feature of a polyhedron to a query point are shown above. Once the closest feature is obtained from Algorithm 1, the closest point, i.e., $\nu(P)$, can be determined, as shown above.

Algorithm 5 Sub-routine for $|V| = 4$

```
1:  function S3D($\{\mathbf{v}_4, \mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$)
2:      $\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} = \mathbf{v}_{12} \times \mathbf{v}_{13}$
3:      if $(\mathbf{v}_1^T \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1})(\mathbf{v}_{14}^T \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}) \geq 0$ then
4:          $[V, \mathbf{d}] \leftarrow$ S2D($\{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$)
5:      else
6:          $\mathbf{n}_{\mathbf{v}_4\mathbf{v}_3\mathbf{v}_1} = \mathbf{v}_{13} \times \mathbf{v}_{14}$
7:          if $(\mathbf{v}_1^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_3\mathbf{v}_1})(\mathbf{v}_{12}^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_3\mathbf{v}_1}) \geq 0$ then
8:              $[V, \mathbf{d}] \leftarrow$ S2D($\{\mathbf{v}_4, \mathbf{v}_3, \mathbf{v}_1\}$)
9:          else
10:             $\mathbf{n}_{\mathbf{v}_4\mathbf{v}_2\mathbf{v}_1} = \mathbf{v}_{12} \times \mathbf{v}_{14}$
11:             if $(\mathbf{v}_1^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_2\mathbf{v}_1})(\mathbf{v}_{13}^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_2\mathbf{v}_1}) \geq 0$ then
12:                 $[V, \mathbf{d}] \leftarrow$ S2D($\{\mathbf{v}_4, \mathbf{v}_2, \mathbf{v}_1\}$)
13:             else
14:                 $V \leftarrow \{\mathbf{v}_4, \mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$
15:             end if
16:         end if
17:     end if
18: end function
```

3. Bézier Curves

3.1. Continuity Conditions

As explained before, at each sampling time, a trajectory, expressed as a parametric Bézier curve, is generated for the time horizon $[t_k, t_k + t_h]$, and the trajectory for the entire flight time $[0, T]$ is formed by joining segments of these Bézier curves end-to-end. The smoothness of the resulting composite trajectory must be guaranteed by enforcing continuity at the joining points of two consecutive segments up to a certain derivative. In the following, in order to derive conditions that address parameter continuity between consecutive curves, we assume that the time horizon is equal to $\Delta t_k = t_{k+1} - t_k$, which is not necessarily the same for all sub-problems. In practice, however, the time horizon is greater than Δt_k, in which case a Bézier curve describing the segment over the time interval $[t_k, t_{k+1}]$ can be obtained by subdividing $\mathbf{p}_k(.)$ at t_{k+1} with the de Casteljau's algorithm. For simplicity we drop the subscript $i \in [N_v]$.

The Bézier curve describing the trajectory over the time interval $[t_k, t_{k+1}]$ is defined as

$$\mathbf{p}_k(\tau_k) = \sum_{l=0}^{n_k} \bar{p}_{l,k} B_{l,n_k}(\tau_k), \tag{34}$$

Assuming that the global parameter t runs over the interval $[t_k, t_{k+1}]$, the local parameter τ_k is related to t by

$$0 \leq \tau_k = \frac{t - t_k}{t_{k+1} - t_k} \leq 1 \tag{35}$$

The parametric continuity condition, C^{r}, requires the r-th derivative and all lower derivatives of two consecutive segments to be equal at the joining point. In other words,

$$\frac{d^r \mathbf{p}_k(1)}{dt^r} = \frac{d^r \mathbf{p}_{k+1}(0)}{dt^r} \quad r \in \{0, \dots, \mathrm{r}\} \tag{36}$$

Zero-order parametric continuity, C^0, requires the endpoints of two consecutive curves, $\mathbf{p}_k(.)$ and $\mathbf{p}_{k+1}(.)$, to intersect at one endpoint, that is,

$$\mathbf{p}_k(1) = \mathbf{p}_{k+1}(0) \tag{37}$$

Since a Bézier curve is coincident with its control points at the two ends, i.e.,

$$\mathbf{p}_k(0) = \bar{p}_{0,k} \quad \mathbf{p}_k(1) = \bar{p}_{n_k,k}, \tag{38}$$

the position continuity condition (37) translates into

$$\bar{p}_{n_k,k} = \bar{p}_{0,k+1} \tag{39}$$

The first-order parametric continuity condition, C^1, for the k-th and $k+1$-th Bézier curves, can be obtained as

$$\begin{aligned} \Delta t_{k+1} n_k (\bar{p}_{n_k,k} - \bar{p}_{n_k-1,k}) = \\ \Delta t_k n_{k+1} (\bar{p}_{1,k+1} - \bar{p}_{0,k+1}) \end{aligned} \tag{40}$$

Finally, the k-th and $k+1$-th Bézier curves are C^2-continuous if

$$\begin{aligned} &\frac{\Delta t_{k+1}}{\Delta t_k} (\bar{p}_{n_k-1,k} - \bar{p}_{n_k-2,k}) \\ &\qquad + n_k(n_{k+1} - 1)\bar{p}_{n_k-1,k} + n_k \bar{p}_{n_k,k} = \\ &\frac{\Delta t_k}{\Delta t_{k+1}} (\bar{p}_{1,k+1} - \bar{p}_{2,k+1}) \\ &\qquad + n_{k+1}(n_k - 1)\bar{p}_{1,k+1} + n_{k+1} \bar{p}_{0,k+1} \end{aligned} \tag{41}$$

Higher-order parametric continuity conditions can be obtained likewise.

3.2. Evaluating Inequalities in Bézier Form

Parameterizing the trajectory with a Bézier curve converts the original infinite dimensional problem (3) into a semi-infinite optimization problem with a finite number of decision variables and an infinite number of constraints. The commonly used approach to obtaining a standard optimization problem is time gridding, which inspects satisfaction of constraints on a finite number of points only. Although this method is straightforward, it cannot guarantee that constraints are satisfied over the entire time interval. Using fine discretization can remedy this issue, but, it will increase the number of constraints as well as the computation time. Since all constraints involved in the trajectory generation problem addressed above can be expressed as Bézier curves, we can employ the method proposed in [33] to recast the semi-infinite optimization problem into one that is computationally tractable. As explained below this method exploits unique features of Bézier curves to efficiently evaluate constraints while avoiding problems associated with time gridding.

If $h(\tau)$ can be expressed as a Bézier curve, then any inequality constraint of the form $h(\tau) \leq 0, \tau \in [0,1]$ can be replaced by a finite set of constraints on its control points. More specifically, if $h(\tau)$ is defined as

$$h(\tau) = \sum_{l=0}^{n_h} \bar{h}_l B_{l,n_h}(\tau), \tag{42}$$

then from the *convex hull* property of Bézier curves we know that

$$h(\tau) \in CH(\bar{H}) \quad \tau \in [0,1] \tag{43}$$

where $CH(\bar{H}) = \{\alpha_0 \bar{h}_0 + \cdots + \alpha_{n_h} \bar{h}_{n_h} | \alpha_0 + \ldots, \alpha_{n_h} = 1, \alpha_l \geq 0\}$ is the convex hull defined by the set of control points [34]. Thus, the inequality constraint $h(\tau) \leq 0$ holds if

$$\bar{h}_l \leq 0 \quad for \quad l = 0, \ldots, n_h. \tag{44}$$

This finite set of inequality constraints can ensure that the original inequality constraint is satisfied over the entire interval $[0,1]$. However, Ineqs. (44) might be conservative due to the existing gap between the control points $\bar{h}_l$ and the actual curve $h(\tau)$. This problem can be alleviated by refining the control polygon and finding closer control points to the curve using recursive subdivision of $h(\tau)$ with the de Casteljau's algorithm. The sequence of control polygons generated with repeated subdivision converges to the underlying Bézier curve [35]. Figure 6 shows a threefold subdivision of a cubic Bézier curve. Furthermore, the de Casteljau's algortithm allows refining the control polygon locally. Using recursive subdivsion of $h(\tau)$ to reduce the conservatism in the finite set of constraints results in an increase in the number of constraints; hence, a trade-off has to be made between the computational effort and the conservatism. Nevertheless, the optimization variables remain the same [36].

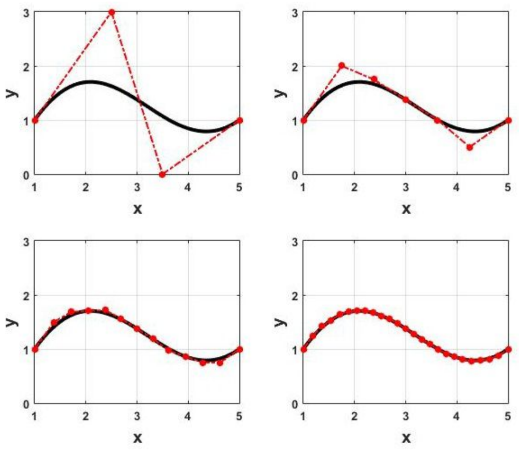


Figure 6. A cubic Bézier curve (**top left**) is subdivided into two Bézier curves of the same degree (**top right**) using the de Casteljau's algorithm. The control polygon generated by recursive subdivision converges to the original Bézier curve ref. [28]. Copyright 2021 IEEE. Successive refinement of the original control polygon after 2 (**bottom left**) and 3 (**bottom right**) subdivisions.

4. Simulation Results

In this section, the efficacy of the proposed method for generating feasible and collision-free trajectories in (vehicle-) dense environments are assessed through different simulation examples. We compare the resulting trajectories to those generated with the well-studied BVC approach. We specifically test the capability of the two methods to generate trajectories that ensure all drones involved in a simulation example reach their final positions, and compare the flight time, obtained with each of them, to complete point-to-point transition missions. We also present the recorded computation time for executing the proposed algorithm in this paper to emphasize its suitability for real-time applications.

In the simulations presented below, we assume all drones have the same size, and their BVC (23) is defined with the safety radius $r_{\mathbf{D}} = 0.30$ m. To specify the set (25), we approximate the drone body with an oblate spheroid with $\Lambda = \text{diag}([0.30 \text{ m}, 0.30 \text{ m}, 0.11 \text{ m}])$. In both methods, trajectories are parameterized with Bézier curves. Upper and lower bounds on the speed and acceleration are assumed to be $\pm 2.3 \frac{\text{m}}{\text{s}}$ and $\pm 7.1 \frac{\text{m}}{\text{s}^2}$ respectively. At each replanning step, the planner finds the closest point in the updated Voronoi cell to the goal position using the algorithm in Section 2.3. The computed point is then used to define the terminal cost term. The first term of the cost function in all subproblems is defined as $\int_0^1 \|\mathbf{p}_{i,k}^{(4)}(\tau)\|^2 d\tau$. The time horizon and the replanning step are also considered to be the same for both methods. The obtained solution at the previous replanning step is used to set the initial guess for the current sub-problem. We use FORCES Pro [37] to generate solvers for the resulting sub-problems. The sub-problems, involving the set of control points $\bar{p}_{i,k}$ as decision variables, can be reformulated to match the supported classes of problem in FORCES Pro. Here, all computations are executed on a single desktop computer, with 2.60 GHz i7-4510U CPU and 6.00 GB RAM; however, in practice, the resulting independent sub-problems can be solved in parallel.

As mentioned before in the paper, in Voronoi-based methods, a vehicle only requires the position information from its neighboring vehicles to generate its trajectory. Therefore, they are more suitable for implementation when vehicles have limited communication capability, and have to rely solely on onboard sensing. In reality, the position sensor noise can impact the planner performance, yet this is more pronounced when estimating other information, such as velocity, from noise-corrupted measurements is needed. Therefore, Voronoi-based planners are more robust when there is no communication network. Nevertheless, in the following simulations, we assume that accurate position information is available with no delay at the replanning time.

In the first example, we consider five drones flying from their initial positions to given final positions. This example is similar to one in [24] where a random offset is added to break the symmetry in the drones' initial and final configurations. Figure 7 (right) shows collision-free trajectories generated with the distributed scheme described above, with a replanning rate of 20 Hz. For this particular example, the resulting trajectories match those generated with BVC with a flight time of 11.6782 s. Figure 7 (left) shows collision-free trajectories obtained from the centralized solution, which delivers a total flight time of 9.4347 s, yet, while the central solution is obtained in 601 ms, the average computation time for solving the sub-problems in the decentralized scheme is only 49 ms.

In the next example, we consider 18 drones switching positions in a 3 m $\times$ 5 m $\times$ 2 m space, with a maximum speed and acceleration of $\pm 4.7 \frac{\text{m}}{\text{s}}$ and $\pm 9.8 \frac{\text{m}}{\text{s}^2}$, respectively. Figure 8a shows the initial and final configurations, and Figure 8b displays collision-free trajectories generated with the proposed distributed scheme in the paper implemented at 10 Hz. While both methods could find collision-free trajectories for guiding the team of drones from their initial positions to their goal positions, the flight time achieved with the proposed method is markedly shorter than the time obtained with BVC. We also performed a trial simulation with 34 drones in a similar configuration. Table 1 compares the success rate and the flight time to complete the transition using BVC and the proposed method.

In the third example, we consider 100 drones flying in an 8 m $\times$ 8 m $\times$ 3.5 m space. The initial and final positions for the drones are displayed with dot and square markers in Figure 9. We test both methods in 30 different trials. In each trial, final positions are randomly assigned to drones. A trial is considered successful if all drones could reach their final positions within the stipulated time. The proposed method with 23 successful trials and an average total flight of 1s outperforms the BVC with only 16 completed trials. It should be noted that using well-devised deadlock prevention strategies or loosening time constraints can improve the success rate of both methods. Figure 9 shows collision-free trajectories generated with the proposed method for one of the trials at different time steps. The average computation time for solving sub-problems in this example was around 115 milliseconds. In addition, compared to the geometric algorithm in [24], the closest point in a Voronoi cell to the goal position was obtained at least 10 times faster with the proposed algorithm in Section 2.3. The computation time for finding the closest point, and solving the optimization problem, depends on the number of neighboring drones (See Table 2 for recorded computation times in simulation examples with 18, 34, and 100 drones). In most applications, with typical Voronoi diagrams, the number of boundary planes, i.e., the number of Voronoi neighbors, is small. Thus, the proposed distributed algorithm is scalable to arbitrary numbers of vehicles.

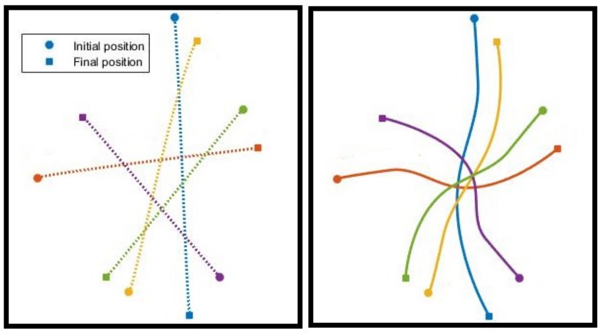
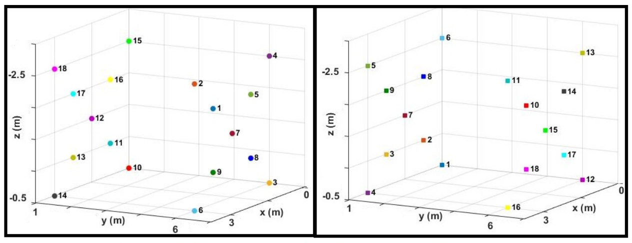
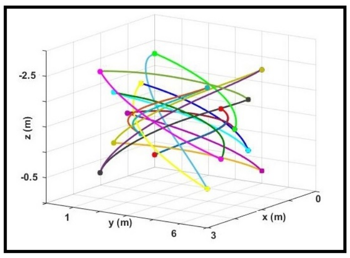


Figure 7. Comparing collision-free trajectories generated with the centralized solution (**left**) and the proposed decentralized approach (**right**) for five drones flying from their initial positions to given final positions. While the central solution yields a shorter flight time, its computation time is significantly longer than average time required to solve the sub-problems in the distributed method.



(a) Initial (left) and final (right) configurations



(b) Collision-free trajectories

Figure 8. (**a**) Initial (left) and final (right) position configurations for 18 drones. Each drone is assigned a unique color and a number next to it. (**b**) Collision−free trajectories for 18 drones switching their positions in a 3 m × 5 m × 2 m space. The total flight time for the drones to reach their final positions is 5.1 s using the proposed method, which is shorter than the 6.3 s flight time obtained with the BVC.

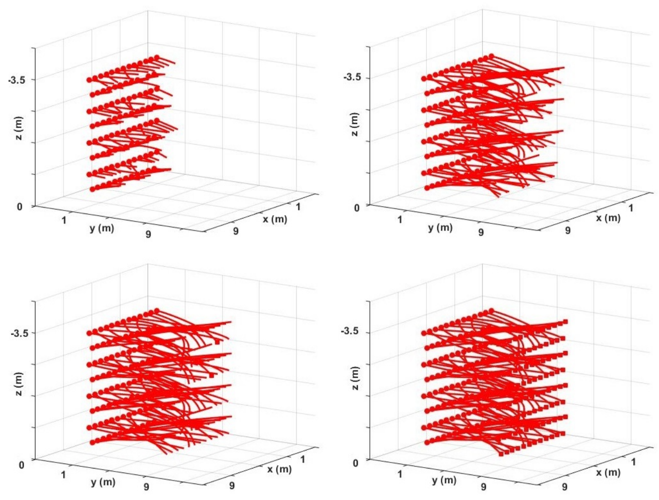


Figure 9. Collision−free trajectories for 100 drones flying from their initial positions (dots) to randomly specified final positions (squares) at different replanning steps.

Table 1. Comparing the number of successful trials and the average flight time achieved with the BVC and the proposed method in the paper.

Number of Drones	BVC		Proposed Method	
	Flight Time	**Completed Trials**	**Flight Time**	**Completed Trials**
18	6.812 s	5/5	5.327 s	5/5
34	8.105 s	7/10	6.625 s	8/10
100	14.573 s	16/30	11.462 s	23/30

Table 2. Recorded computation times for finding the closest point in a Voronoi cell to the goal position and solving the optimization problem in simulation examples with 18, 34, and 100 drones.

Number of Drones	Computation Time (ms)	
	Finding the Closest Point	**Solving the Sub-Problem**
18	<0.1	77.562
34	<0.1	98.330
100	0.171	121.633

5. Conclusions

In this paper, we introduce an efficient distributed algorithm for generating collision-free trajectories for multiple drones, taking into account their orientation. In order to avoid substantial communication between drones, we adopt Voronoi-based space partitioning and derive local constraints that guarantee collision avoidance with neighboring vehicles for an entire time horizon. We leverage Bézier curve properties to ensure that the set of collision avoidance constraints are satisfied at any time instant of the planning horizon. The same approach can be employed to obtain local collision avoidance constraints for the cases where the normal vector and offset of separating planes are time-varying parameters or decision variables of sub-problems. Yet, adopting Voronoi diagram with fixed planes for an entire planning horizon, though being conservative, results in simple, small sub-problems allowing for the trajectories to be replanned at a higher rate. We present different simulation results to highlight the scalability of the algorithm to large numbers of drones, and also its capability to generate less conservative trajectories with notably shorter flight times, compared to other Voronoi-based methods.

Our future work includes implementation and experimental validation of the algorithm for teams of drones. As we explain in the paper, at each time sample, upon receiving (or sensing) the new position information, a vehicle must find the closest point in its Voronoi cell to the goal position, and solve an optimization problem that uses the current state of the vehicle as the initial condition, to generate its trajectory for a certain time horizon. Although the time to compute the closest point is mainly negligible, the computation time to find the optimal solution can lead to a (significant) delay between updating the position information and executing the trajectory. Therefore, in practice, the computational delay must be explicitly considered to avoid performance degradation (or even failure) of the planner.

Author Contributions: Conceptualization and writing—original draft preparation, B.S.; Supervision and writing—review and editing, R.C. and A.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by Fundação para a Ciência e a Tecnologia and FEDER funds through the projects UIDB/50009/2020 and LISBOA-01-0145-FEDER-031411. B. Sabetghadam acknowledges the support of Instituto Superior Técnico through scholarship BL216/2018/IST-ID.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Kuwata, Y.; How, J.P. Cooperative distributed robust trajectory optimization using receding horizon MILP. *IEEE Trans. Control Syst. Technol.* **2010**, *19*, 423–431. [CrossRef]
2. Augugliaro, F.; Schoellig, A.P.; D'Andrea, R. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012; pp. 1917–1922.
3. Morgan, D.; Chung, S.J.; Hadaegh, F.Y. Model predictive control of swarms of spacecraft using sequential convex programming. *J. Guid. Control Dyn.* **2014**, *37*, 1725–1740. [CrossRef]
4. Chen, Y.; Cutler, M.; How, J.P. Decoupled multiagent path planning via incremental sequential convex programming. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 5954–5961.
5. Kuwata, Y.; Richards, A.; Schouwenaars, T.; How, J.P. Distributed robust receding horizon control for multivehicle guidance. *IEEE Trans. Control Syst. Technol.* **2007**, *15*, 627–641. [CrossRef]
6. Chaloulos, G.; Hokayem, P.; Lygeros, J. Distributed hierarchical MPC for conflict resolution in air traffic control. In Proceedings of the American Control Conference, Baltimore, MD, USA, 30 June–2 July 2010; pp. 3945–3950.
7. Tedesco, F.; Raimondo, D.M.; Casavola, A.; Lygeros, J. Distributed collision avoidance for interacting vehicles: A command governor approach. *IFAC Proc. Vol.* **2010**, *43*, 293–298. [CrossRef]
8. Van Parys, R.; Pipeleers, G. Distributed model predictive formation control with inter-vehicle collision avoidance. In Proceedings of the 2017 11th Asian Control Conference (ASCC), Gold Coast, QLD, Australia, 17–20 December 2017; pp. 2399–2404.
9. Van Parys, R.; Pipeleers, G. Distributed MPC for multi-vehicle systems moving in formation. *Robot. Auton. Syst.* **2017**, *97*, 144–152. [CrossRef]
10. Tordesillas, J.; How, J.P. MADER: Trajectory planner in multiagent and dynamic environments. *IEEE Trans. Robot.* **2021**, *38*, 463–476. [CrossRef]
11. Luis, C.E.; Schoellig, A.P. Trajectory generation for multiagent point-to-point transitions via distributed model predictive control. *IEEE Robot. Autom. Lett.* **2019**, *4*, 375–382. [CrossRef]
12. Luis, C.E.; Vukosavljev, M.; Schoellig, A.P. Online trajectory generation with distributed model predictive control for multi-robot motion planning. *IEEE Robot. Autom. Lett.* **2020**, *5*, 604–611. [CrossRef]
13. Van den Berg, J.; Lin, M.; Manocha, D. Reciprocal velocity obstacles for real-time multi-agent navigation. In Proceedings of the IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 19–23 May 2008; pp. 1928–1935.
14. Van Den Berg, J.; Guy, S.J.; Lin, M.; Manocha, D. Reciprocal n-body collision avoidance. In *Robotics Research*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 3–19.
15. van den Berg, J.; Guy, S.J.; Snape, J.; Lin, M.C.; Manocha, D. rvo2 Library: Reciprocal Collision Avoidance for Real-Time Multi-Agent Simulation. 2011. Available online: https://gamma.cs.unc.edu/RVO2/ (accessed on 24 January 2022).

16. Alonso-Mora, J.; Breitenmoser, A.; Beardsley, P.; Siegwart, R. Reciprocal collision avoidance for multiple car-like robots. In Proceedings of the IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA, 14–18 May 2012; pp. 360–366.
17. Snape, J.; Van Den Berg, J.; Guy, S.J.; Manocha, D. The hybrid reciprocal velocity obstacle. *IEEE Trans. Robot.* **2011**, *27*, 696–706. [CrossRef]
18. Giese, A.; Latypov, D.; Amato, N.M. Reciprocally-rotating velocity obstacles. In Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 31 May–7 June 2014; pp. 3234–3241.
19. Bortoff, S.A. Path planning for UAVs. In Proceedings of the 2000 American Control Conference (ACC), Chicago, IL, USA, 28–30 June 2000; pp. 364–368.
20. Garrido, S.; Moreno, L.; Abderrahim, M.; Martin, F. Path planning for mobile robot navigation using voronoi diagram and fast marching. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China, 9–15 October 2006; pp. 2376–2381.
21. Bhattacharya, P.; Gavrilova, M.L. Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path. *IEEE Robot. Autom. Mag.* **2008**, *15*, 58–66. [CrossRef]
22. Cortes, J.; Martinez, S.; Karatas, T.; Bullo, F. Coverage control for mobile sensing networks. *IEEE Trans. Robot. Autom.* **2004**, *20*, 243–255. [CrossRef]
23. Bandyopadhyay, S.; Chung, S.J.; Hadaegh, F.Y. Probabilistic swarm guidance using optimal transport. In Proceedings of the 2014 IEEE Conference on Control Applications (CCA), Juan Les Antibes, France, 8–10 October 2014; pp. 498–505.
24. Zhou, D.; Wang, Z.; Bandyopadhyay, S.; Schwager, M. Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells. *IEEE Robot. Autom. Lett.* **2017**, *2*, 1047–1054. [CrossRef]
25. Senbaslar, B.; Hönig, W.; Ayanian, N. Robust trajectory execution for multi-robot teams using distributed real-time replanning. In *Distributed Autonomous Robotic Systems*; Springer: Cham, Switzerland, 2019; pp. 167–181.
26. Mellinger, D.; Kumar, V. Minimum snap trajectory generation and control for quadrotors. In Proceedings of the IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 2520–2525.
27. LaValle, S.M. *Planning Algorithms*; Cambridge University Press: Cambridge, UK, 2006.
28. Sabetghadam, B.; Cunha, R.; Pascoal, A. Trajectory Generation for Drones in Confined Spaces using an Ellipsoid Model of the Body. *IEEE Control Syst. Lett.* **2021**, *6*, 1022–1027. [CrossRef]
29. Gilbert, E.G.; Johnson, D.W.; Keerthi, S.S. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE J. Robot. Autom.* **1988**, *4*, 193–203. [CrossRef]
30. Van Den Bergen, G. *Collision Detection in Interactive 3D Environments*; CRC Press: Boca Raton, FL, USA, 2003.
31. Ericson, C. *Real-Time Collision Detection*; CRC Press: Boca Raton, FL, USA, 2004.
32. Tereshchenko, V.; Chevokin, S.; Fisunenko, A. Algorithm for finding the domain intersection of a set of polytopes. *Procedia Comput. Sci.* **2013**, *18*, 459–464. [CrossRef]
33. Montanari, M.; Petrinic, N.; Barbieri, E. Improving the GJK algorithm for faster and more reliable distance queries between convex objects. *ACM Trans. Graph. (Tog)* **2017**, *36*, 1–7. [CrossRef]
34. Farouki, R.T. The Bernstein polynomial basis: A centennial retrospective. *Comput. Aided Geom. Des.* **2012**, *29*, 379–419. [CrossRef]
35. Cichella, V.; Kaminer, I.; Walton, C.; Hovakimyan, N.; Pascoal, A.M. Optimal Multivehicle Motion Planning Using Bernstein Approximants. *IEEE Trans. Autom. Control* **2020**, *66*, 1453–1467. [CrossRef]
36. Sabetghadam, B.; Cunha, R.; Pascoal, A. Real-time Trajectory Generation for Multiple Drones using Bézier Curves. *IFAC-PapersOnLine* **2020**, *53*, 9276–9281. [CrossRef]
37. Domahidi, A.; Jerez, J. *Forces Professional*; Embotech AG: Zürich, Switzerland, 2014–2019. Available online: https://embotech.com/FORCES-Pro (accessed on 22 June 2021).