drones

MDPI

Article

Distributed Control for Multi-Robot Interactive Swarming Using Voronoi Partioning †

Alexandre Eudes, Sylvain Bertrand, Julien Marzat * and Ioannis Sarras

DTIS, ONERA, Université Paris-Saclay, F-91123 Palaiseau, France; alexandre.eudes@onera.fr (A.E.); sylvain.bertrand@onera.fr (S.B.); ioannis.sarras@onera.fr (I.S.)

* Correspondence: julien.marzat@onera.fr

† This paper is an extended version of our paper: Bertrand, S.; Sarras, I.; Eudes, A.; Marzat, J. Voronoi-Based Geometric Distributed Fleet Control of a Multi-Robot System. In Proceedings of the 16th International Conference on Control, Automation, Robotics and Vision (ICARCV), Shenzhen, China, 13–15 December 2020; pp. 85–91.

Abstract: The problem of safe navigation of a human-multi-robot system is addressed in this paper. More precisely, we propose a novel distributed algorithm to control a swarm of unmanned ground robots interacting with human operators in presence of obstacles. Contrary to many existing algorithms that consider formation control, the proposed approach results in non-rigid motion for the swarm, which more easily enables interactions with human operators and navigation in cluttered environments. Each vehicle calculates distributively and dynamically its own safety zone in which it generates a reference point to be tracked. The algorithm relies on purely geometric reasoning through the use of Voronoi partitioning and collision cones, which allows to naturally account for inter-robot, human-robot and robot-obstacle interactions. Different interaction modes have been defined from this common basis to address the following practical problems: autonomous waypoint navigation, velocity-guided motion, and follow a localized operator. The effectiveness of the algorithm is illustrated by outdoor and indoor field experiments.

Keywords: multi-robot swarms; human-robot interactions; distributed control; voronoi partioning

Citation: Eudes, A.; Bertrand, S.; Marzat, J.; Sarras, I. Distributed Control for Multi-Robot Interactive Swarming Using Voronoi Partioning. *Drones* **2023**, *7*, 598. https://doi.org/10.3390/drones7100598

Academic Editor: Oleg Yakimenko

Received: 8 September 2023
Accepted: 18 September 2023
Published: 23 September 2023

1. Introduction

The deployment of swarms of unmanned vehicles for both civil and defense missions has radically increased in the last years. Recent progresses in vision or laser-based localization and mapping, along with the increase in embedded computational power, have led to the development of mobile and aerial robots of reduced dimensions allowing larger swarms of robotic vehicles to effectively undertake such missions under realistic environmental and communication conditions. Nevertheless, interaction with humans and obstacles or the practical limitations of inter-vehicle communication data links still pose serious challenges that need to be consistently addressed for on-field deployment of teams of autonomous robots [1,2]. This requires the synthesis of distributed control algorithms with increased capabilities in terms of autonomy, safety and resilience.

Several paradigms have been proposed for distributed multi-vehicle control [3–5] such as: leader-following, behavioral rules, virtual structure, artificial potential function, graph-rigidity. As indicated by its name, the leader-following approaches require to define one robot as the leader. In this setting, the leader has access to information such as the final destination or visibility to a target, which is unavailable to the other vehicles. However, the role selection of a particular robot as a leader is strongly related to the time-dependent mission and environment scenario as well as the swarm status. As such, if a leader change is necessary, particular additional rules need to be established in order to define the hierarchy alternations [6]. The second category of methods for cooperative control is inspired by initial works studying team behaviors in nature. These are based on behavioral rules

that each agent should follow according to its local task and its interactions with the environment [7,8]. Such approaches are nevertheless usually very problem-dependent and are not easy to modify whenever an unexpected event occurs. The third category of methods hinges upon virtual geometrical structures in which the swarm of agents should remain globally. The control law must first ensure that the agents are located within the structure and then define a suitable structure evolution depending on the mission requirements [9,10]. A common approach is to design potential fields and navigation functions that are sometimes difficult to construct with good properties (differentiable, without multiple critical points).

Most of these methods require that the geometric formation of the swarm is quasi-explicitly defined through fixed, desired relative positions or distances to be attained [11–13], and they cannot incorporate naturally the interactions with external agents such as a human, be it a pedestrian that has to be avoided or an operator that has to be followed at a distance. With respect to a fleet formation, fixing inter-agent distances or positions restricts the relative motions of the agents, as well as of the global formation, and does not allow much flexibility and adaptability when dealing with uncertain, dynamic and cluttered environments. Such type of formations are usually called rigid [4]. An alternative approach has been derived in [14] where it is ensured that all the vehicles stay inside a region, with a minimum distance between neighbors, whose shape can be assigned by modifying the associated potential function. However, the gain selection for practical deployment of the approach is not easy.

A less rigid behavior, more suitable to interactions with a cluttered environment or human operators, can be achieved by partitioning the space for motion coordination. Voronoi diagrams have been widely considered as a natural way to define space partitioning. In the context of multi-agent swarming, it also a convenient way to define the topology of the interaction network between the agents [15], by considering the graph associated to the corresponding Delaunay triangulation [16]. For motion coordination of multi-agent systems, Voronoi diagrams have been mostly used for allocation and coverage tasks [17–19] and more recently for cooperative pursuit of a single target [20] or multiple targets [21] by multi-agent systems, as well as cooperative exploration using dynamic centroid-based area partition [22]. A simplified version of the swarm navigation problem compared to the one addressed in this paper has been addressed in [23–25]. These algorithms rely on user-defined navigation functions to compute the centroid of the Voronoi cell of each agent, which is used as the reference position to be tracked by the robots, and collision avoidance is handled directly by the space partioning. An improved version relying on geometric constraints has been proposed by the authors in [26], with a more intuitive and direct management of swarm navigation and collision avoidance behaviors between the vehicles. The work presented in this paper is an extension of the latter method, with the inclusion of obstacle avoidance and interaction with a human operator. Contrary to the majority of the literature on Voronoi tesselations for multi-vehicle applications that treat obstacle avoidance through the partitioning, the proposed solution prefers to exploit the appealing concept of collision cones that has been very successful, especially in mono-vehicle applications [27,28]. An alternative approach relying also on Voronoi partitions but with artificial potential fields for collision avoidance has been proposed in [29] and applied to a fleet of quadrotors in simulation and experiments.

We thus propose a new Voronoi-partioning swarm control algorithm which allows to define three different modes of interactions from a common basis. They allow operators to be included in a swarm of autonomous vehicles and guide the robots with their own velocity and/or position in a safe coordinated motion while evolving in unknown cluttered environments. Indeed, extending the earlier concept of swarm teleoperation by a human operator [30], more advanced interactions can be integrated in control algorithms for human–multi-robot swarming. Different types of interactions can be considered depending on information flows available between the robots and human operators (one-way/two-ways)

and the nature of the interactions themselves (physical/non-physical), see e.g., [31] for a large overview.

The contributions of the paper can be summarized as follows:

- Proposition of a distributed control algorithm enabling non-rigid motion for human-multi-robot swarming in cluttered environments.
- Design of a purely geometric approach applied by each robot to define distributively a reference point to be tracked inside its Voronoi cell, accounting for other robots and obstacles (collision avoidance), as well as human operators (collision avoidance and other possible interactions, see below).
- Decoupling between the mechanisms of obstacle avoidance and collision avoidance. This allows to reduce the design complexity when accounting for obstacles, as opposed to navigation functions for example, and to render the gain tuning more straightforward.
- Possibility to handle different modes of interactions between human operator and robots of the swarm. These modes of interactions correspond to practical problems of interest which are autonomous waypoint navigation, velocity-guided motion and follow a localized operator.
- Implementation and real-world field experiments in indoor and outdoor environments with self-localized ground mobile robots and human operator, in presence of various obstacles.

The problem definition and the proposed swarm control method are described in Section 2, a corresponding system architecture is defined in Section 3 and field experiments with up to three mobile robots and a localized human operator are reported in Section 4. A Video of the experimental setup is available at https://tinyurl.com/OneraHumanRobotSwarm (accessed on 19 September 2023).

2. Swarm Control Method

2.1. Problem Definition

The problem studied is the guidance of a swarm of N Unmanned Ground Vehicles (UGVs) to a waypoint, denoted by $P^* \in \mathbb{R}^2$, and by extension to successive waypoints either on a given path or defined dynamically (see Section 2.3), in a cluttered environment with no prior map available. A typical applicative context is search-and-rescue or tactical missions, where human operators are assisted by a swarm of autonomous robots for transportation of critical resources, wounded persons or communication link maintenance. A fully autonomous behavior is expected from the swarm under safety constraints with respect to the presence of obstacles and humans, and allowing an automatic reconfiguration in case of vehicle loss(es). It is assumed that each vehicle is able to estimate its own position with respect to a common global fixed reference frame, where P^* is also defined, and to broadcast it to all other vehicles within a given range. The position of the ith agent (referred as Robot i thereafter) will be denoted by $p_i \in \mathbb{R}^2$ and the set of its neighbor robots is indexed by $\mathcal{N}^i = \{j \,|\, j = 1, \ldots, N, j \neq i, \|p_i - p_j\| \leq r_{com}\}$ where $r_{com} > 0$ is the communication range assumed to be constant. The number of neighbors of Robot i will be referred to as $N^i = Card\{\mathcal{N}^i\}$. Each robot is assumed to be able to localize a set of surrounding obstacles, which are modeled as disks with a radius incorporating the desired safety distance. Human operators are assumed to be equipped with equivalent localization devices, and are considered as additional vehicles with no control input computed. Three levels of interaction between the swarm and a human operator have been studied:

1. An *Autonomous* mode, in which the swarm has to follow autonomously a predefined path at a given nominal speed. Examples of tasks that can be performed with this mode are transfers of equipment or injured people between two locations. Other tasks could be the persistent surveillance of zones in order to detect abnormal events, by making the UGVs autonomously and repeatedly move along a surveillance path composed of predefined waypoints.
2. A *Velocity-Guided* mode, where the swarm follows a predefined path at the same speed as a human operator. In other words, the desired positions and orientations of

the robots with respect to the waypoints are the same as in the Autonomous mode, but the velocity to reach them is defined by the motion of a human operator.

3. A *Follow* mode, where the current waypoint to be tracked is defined with respect to the localized operator and also takes into account the positions of all the robots. Note that the human operator could be replaced by a tele-operated robot or a virtual point to obtain a platooning behavior, using the same underlying control algorithm.

All the modes share the common constraint that each robot should remain at a desired safety distance from any human operator, other robot and obstacle.

2.2. Algorithm Description

The main idea of the proposed distributed algorithm is that each vehicle computes online a Voronoi partition of the space involving other physical agents (other vehicles, human operators), and virtual (mirror) agents that are added to maintain the coherence of the swarm (see Figure 1). A reference position to be tracked by a lower-level controller is then computed by each robot inside its own Voronoi cell. A geometric approach has been preferred for this calculation, which is done by considering lines of sight between the vehicle, the waypoint (for attraction), the boundaries of possible obstacles (for avoidance), and other UGVs or human operators (for collision avoidance). The algorithm allows to obtain different behaviors and patterns (e.g., side-by-side, group, convoy-like) by only modifying the initial relative placement of the vehicles. Finally, the distributed nature of the algorithm also grants robustness to online modification (removing or adding) of the number of entities (robots, humans) in the swarm, while also addressing the mono-robot and 2-robot scenarios. If a robot suffers from a failure (e.g., loss of communication or mobility capabilities), the same distributed algorithm is applied without this robot and the swarm can carry on with the given mission. The main steps of the algorithm are the following.

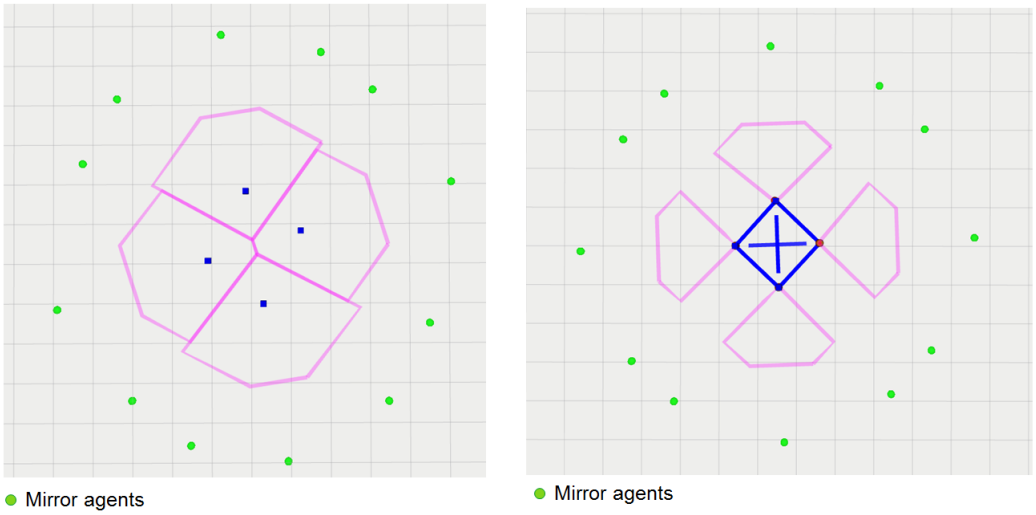

Figure 1. Voronoi diagram for a swarm with four agents: without spacer segments (**left**) or with spacer segments (**right**) to enforce anti-collision between agents. Green dots are mirror agents, blue ones are robots' positions. The cell boundaries of each robot is in pink and spacers are blue segments.

Step 1: Voronoi partitioning

Each robot i computes a Voronoi partition accounting for the other real agents (robots, human operators) in the swarm and virtual mirror agents. The mirror agents are introduced as a means to guarantee the feasibility of the computation of the Voronoi partition, especially with one and two robots, and to adjust the size and bound of each robot's Voronoi cell.

The mechanism proposed in [23] generates mirror neighbors to ensure that the Voronoi cell of Robot i is bounded and help control the expansion of the swarm. As introduced by the authors in [26], let us first define the placement operator $\Psi : \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$ by

$$\Psi(p_i, p_j, d) = \begin{cases} p_i - d\dfrac{p_j - p_i}{\|p_j - p_i\|} & \text{if} \quad p_i \neq p_j \\ p_i \quad \text{otherwise} \end{cases} \tag{1}$$

If Robot i is not in the convex hull generated by the position of the $\mathcal{N}^i$ agents, then N^i mirror neighbors are defined with positions computed as

$$m_j^i = \Psi(p_i, p_j, d_{mir}), j \in \mathcal{N}^i \tag{2}$$

where $m_j^i \in \mathbb{R}^2$ and $d_{mir} > 0$ respectively denote the position of the mirror of neighbor j and the distance of placement of the mirror neighbor with respect to Robot i (see Figure 2). The set of all mirror neighbors for Robot i will be denoted by $\mathcal{M}^i = \left\{ m_j^i, j \in \mathcal{N}^i \right\}$.

Now, Robot i computes its own Voronoi partition using the set of points $\{p_i\} \cup \{p_j, j \in \mathcal{N}^i\} \cup \mathcal{M}^i$. From this Voronoi partition, only the edges and vertices that correspond to the partition where Robot i belongs are kept. This Voronoi *cell* of Robot i is denoted by $\mathcal{C}^i = (\mathcal{V}^i, \mathcal{E}^i)$, where $\mathcal{V}^i$ and $\mathcal{E}^i$ are the sets of vertices and edges of the cell. This cell defines the space in which the reference position P_*^i to be tracked by Robot i is placed, as defined by the next steps of the approach.

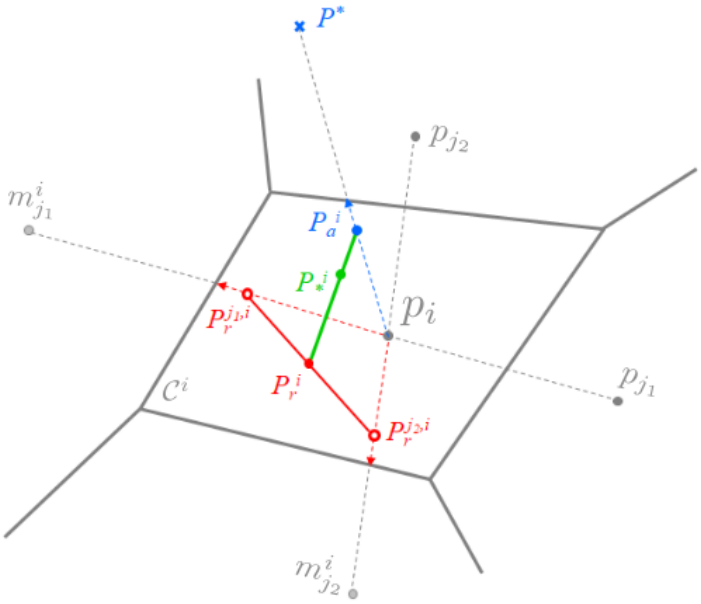

Figure 2. Illustration of computation by robot i of its reference position P_*^i to be tracked. Case with three robots (i, j_1, j_2) and two mirror agents $m_{j_1}^i$ and $m_{j_2}^i$ added to bound the Voronoi cell. Gray points are positions, symbols in blue are related to attraction to waypoint, in red to collision avoidance with other agents, and in green to reference position to be tracked by the robot (Updated from [26]).

An example of such a construction of the Voronoi cells for a swarm of $N = 4$ agents is illustrated on the left part of Figure 1, where real and mirror agents are respectively represented by blue and green dots. The Voronoi cell computed by each of the four agents is represented by magenta lines. In case of collision risk(s) with agent(s), the Voronoi cell is adapted as follows.

Compared to [26], a new feature is introduced in the construction of the cells, for each real agent located at a distance lower than a predefined threshold. In that case, a spacer

segment is inserted between the robot and this agent to modify the construction of the Voronoi cell (see right part of Figure 1, with spacer segments in blue). More details on how spacers are built are given in Step 4 on collision avoidance with other agents.

In our implementation, the Voronoi partitions were computed using the Boost polygon library [32], which has been used in other multi-robot planning algorithms [33,34]. This library is based on the sweepline algorithm for Voronoi diagrams initially proposed in [35].

Step 2: Attraction to waypoint

This step is similar to the one proposed by the authors in [26]. An attraction point P_a^i is defined inside the Voronoi cell C^i of the vehicle i, on the segment directed along the line of sight between the vehicle and the waypoint P^*, and limited inside the Voronoi cell C^i (see Figure 2).

If the waypoint P^* to be reached is located inside the Voronoi cell $\mathcal{C}^i$ of Robot i, then the attraction point is simply defined as $P_a^i = P^*$. If not, P_a^i will be placed inside $\mathcal{C}^i$ along the line of sight between Robot i and the waypoint, as defined by the following procedure.

Let us denote by I_a^i the intersection point of the geometric segment $\overline{p_i\, P^*}$ with edges of $\mathcal{C}^i$ (see Figure 2).The attraction point for Robot i is finally defined as

$$P_a^i = \Psi(p_i, I_a^i, -d_a^i) \tag{3}$$

with the distance

$$d_a^i = \min(d_a^{max}, \lambda_a \left\| I_a^i - p_i \right\|) \tag{4}$$

and where $0 < \lambda_a < 1$ and $d_a^{max} > 0$ are two tuning parameters used to set the position of the attraction point P_a^i on the segment $\overline{p_i\, I_a^i}$ and to limit its distance to Robot i. Parameter λ_a enables to define a margin for the placement of the attraction point inside the Voronoi cell. A value of $\lambda_a = 1$ would correspond to an attraction point located on the edge of the Voronoi cell. A value $\lambda_a < 1$ is therefore preferred to possibly account for uncertainty (e.g., due localization) in the definition of the Voronoi cells and practically ensure with more robustness the belonging of the attraction point to the Voronoi cell. Parameter d_a^{max} is used to limit the distance of the attraction point with respect to the current robot position. This can be useful in cases of large Voronoi cells, e.g., when robots are moving far from each others, to avoid attraction points that would result for the low-level controller in large control input values for the robot.

Special cases $N^i = 0$ or $N^i = 1$: During the mission, the number of neighbors of a robot may change, temporarily or definitively, e.g., due to loss of communication links, loss of robots, etc. If at a given instant, Robot i has zero or one neighbor, one additional step is performed before the standard algorithm. This step is described in Appendix A.

Step 3: Obstacle avoidance

The distances between the vehicle i and the detected obstacles are evaluated to identify obstacles in proximity which need to be checked for collision risk. A map of obstacles is built online thanks to on-board sensors of the UGV (see Section 3.2 for more details). For collision risk evaluation, a disk model of obstacles (including a safety distance) is considered.

Denote by $\{O_l^i, ro_l^i\}, l = 1, \ldots, N_o^i$ the set of N_o^i obstacles detected by Robot i and modeled by disks of centers O_l^i and radius ro_l^i. A first step then consists in computing a cone of "unsafe directions" $\mathcal{U}_l^i$ englobing and tangent to each obstacle l, with the robot's position p_i as vertex (see Figure 3).

A test is then realized to check whether the line of sight (p_i, P^*) between the robot and the waypoint belongs to at least one of these cones $\mathcal{U}_l^i$:

- If not, there is no collision risk with any of the obstacles, and direct straight motion to the waypoint is safe for the robot (as described in Figure 3). The attraction point P_a^i computed at Step 2 is still valid and the algorithm proceeds to the next step.

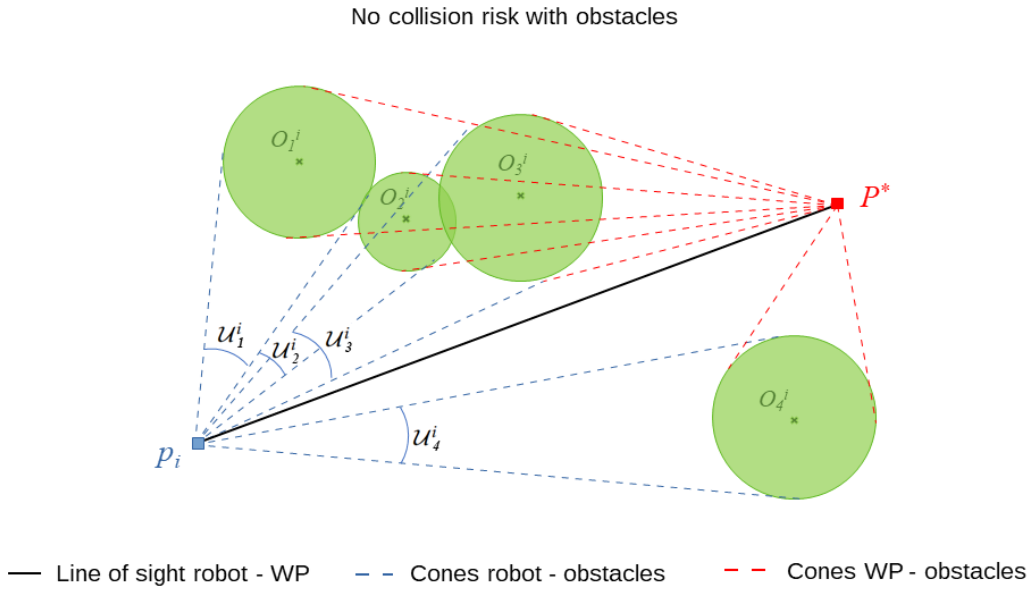

Figure 3. Collision cones and direct safe path towards waypoint in case of no collision risk.

- If there is at least one obstacle with collision risk, the cone of this obstacle is considered. It is enlarged step by step by considering adjacent and intersecting cones related to other obstacles, so as to obtain a larger cone $\bar{\mathcal{U}}^i$ containing a cluster of the obstacles with collision risk for robot i. An example is provided on the left part of Figure 4: the collision cone of obstacle O_2^i is merged with the intersecting collision cone of obstacle O_3^i, which is further merged with collision cone of obstacle O_1^i. This iterative procedure is stopped as there are no other intersection collision cones. The resulting cone is depicted by dashed blue lines. The same procedure is repeated to build another cone $\bar{\mathcal{U}}^*$, but this time by considering the waypoint P^* as vertex. The two intersection points T_1^{i*} and T_2^{i*} between these bounding cones are then computed. They correspond to two intermediate target points for the robot, each of them defining a possible obstacle-free path towards the waypoint. Some heuristics are used at this stage to select the shortest among the two available paths. The target point corresponding to the selected path is considered, instead of the waypoint, to compute a new attraction point P_a^i , in the same way as in Step 2. This new attraction point replaces the one computed in Step 2 and is used instead for the rest of the algorithm.

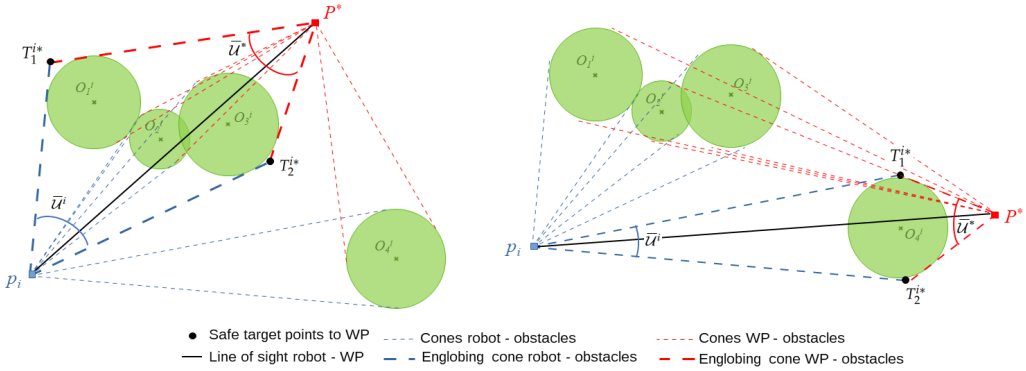

Figure 4. Two examples of collision cones and computation of a safe path towards a given waypoint in case of collision risk with obstacles.

Step 4: Collision avoidance with other agents

This step details the mechanism used to avoid collision between agents. For that purpose, two tools are used to ensure that agents stay at a safety distance from each other. For each other real agent j (either UGV or human operator) at collision risk (distance

criterion), a spacer segment is introduced to reduce the Voronoi cell in the direction of the potential encounter. This spacer segment is used in a range of distance between agents of $\sigma_{col}d_{col}$ and d_{col}, where $d_{col} > 0$ defines a distance threshold representing a collision risk and where $\sigma_{col} \geq 1$ is a smoothing factor to account for some margin in the collision test. Figure 5 illustrates this mechanism. If for any reason, the distance between agents drops under the targeted collision distance $\sigma_{rep}d_{col}$, with $\sigma_{col} > \sigma_{rep} \geq 1$, an additional avoidance mechanism is used that builds a repulsion point. This repulsion point P_r^i is computed as the mean of all the agents individual repulsion point $P_r^{j,i}$ that are defined inside the Voronoi cell of the vehicle i, on the segment directed along the line of sight between agent j and agent i, and limited to the Voronoi cell C^i (see Figure 2). This second mechanism is usually not triggered, due to the use of spacer segments first, but could happen if a robot overshoots its Voronoi cell or for the special case of a localized object like a pedestrian where the Voronoi cell is not enforced. The segment and repulsion points could be used together to exhibit different repulsion behavior. The repulsion point has only a radial influence on the collision avoidance whereas the spacer segment will enforce more parallel trajectories by more aggressively limiting the cell (see Figures 1 and 5). Those mechanisms are compatible with each other as the segment will reduce the Voronoi cell and the repulsion point is defined in the cell itself.

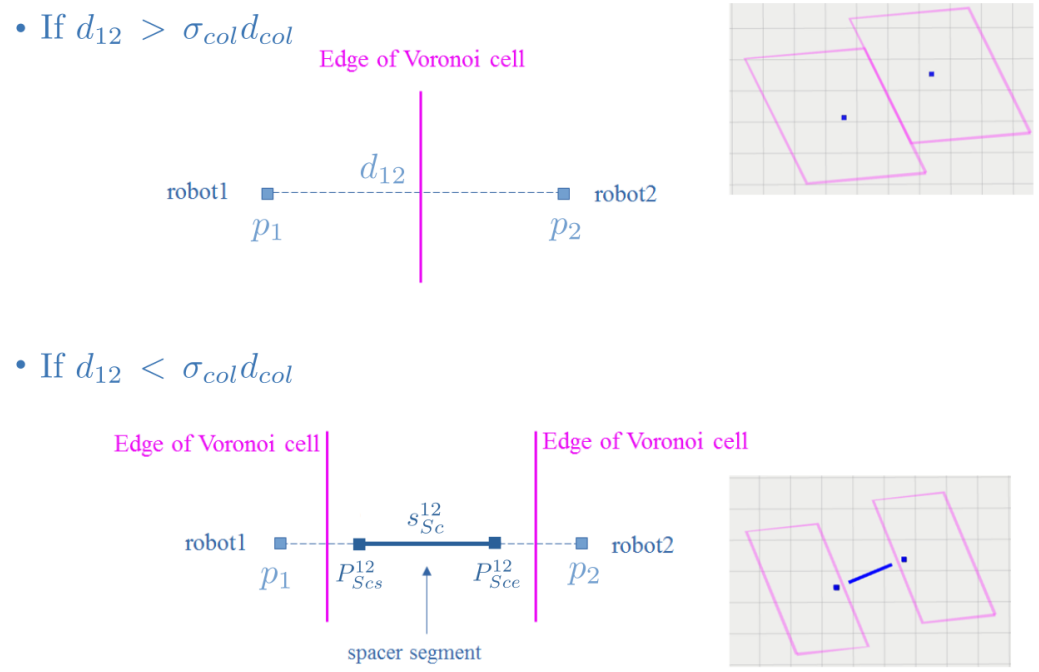

Figure 5. Construction of Voronoi cells: without spacer segments (**top part**) or with spacer segment (**bottom part**) in case of agent too close proximity.

More formally, let the set of robots with collision risk be $\mathcal{N}_{col}$ and the set of neighbor of Robots i that need to be considered in repulsion $\mathcal{N}_{rep}^i$ such that

$$\mathcal{N}_{col} = \left\{(i,j) \in (1,\ldots,N)^2 \,|\, j < i\,, d_{ij} \leq \sigma_{col}d_{col}\right\} \quad (5)$$

$$\mathcal{N}_{rep}^i = \left\{j \in \mathcal{N}^i \,|\, d_{ij} \leq \sigma_{rep}d_{col}\right\} \quad (6)$$

with $d_{ij} = ||p_j - p_i||$ and where d_{col} s.t. $d_{mir} > d_{col} > 0$ is used to define a distance threshold representing a collision risk and where $\sigma_{col} > \sigma_{rep} \geq 1$ is used as a smoothing factor to account for some margin in the collision test. Let $N_{col} = Card\{\mathcal{N}_{col}\}$ be the number of agents with collision risk. If $N_{col} = 0$, the remaining part of Step 4 is skipped. The set

of segments $\mathcal{S}_c = \left\{ S_c^{ij}, \forall (i,j) \in \mathcal{N}_{col} \right\}$ is added during the Voronoi partitioning at Step 1. Each segment S_c^{ij} starts at P_{Scs}^{ij}, ends at P_{Sce}^{ij} and is centered between p_i and p_j (see Figure 5):

$$P_{Scs}^{ij} = \Psi\left(p_i, p_j, \frac{1}{2}(d_{ij} - s_{Sc}^{ij})\right) \tag{7}$$

$$P_{Sce}^{ij} = \Psi\left(p_i, p_j, \frac{1}{2}(d_{ij} + s_{Sc}^{ij})\right) \tag{8}$$

with s_{Sc}^{ij} the size of the segment:

$$s_{Sc}^{ij} = \begin{cases} \dfrac{\sigma_{col} d_{col} - d_{ij}}{\sigma_{col} - 1} & \text{if } d_{ij} > d_{col} \\ d_{ij} - 2d_{vres} & \text{if } d_{ij} \leq d_{col} \end{cases} \tag{9}$$

where d_{vres} is the smallest distance such that $p_i \neq \Psi(p_i, p_j, d_{vres})$ in the Voronoi partitioning step. This is done for all robots pairs in collision and not only for neighbors of robot i, to ensure that the same cell edge is obtained by each robot in the distributed process.

When the repulsion is triggered for a neighbor $j \in \mathcal{N}_{col}^i$ of the robot i, a repulsion point $P_r^{j,i}$ is defined as follows. Let us consider the two intersection points of the geometrical line $(p_i p_j)$ with edges of the Voronoi cell $\mathcal{C}^i$ of Robot i. We denote by I_r^{ji} the intersection point such that the dot product $\overrightarrow{p_i p_j}.\overrightarrow{p_i I_{ji}^r}$ is negative, i.e., I_r^{ji} is located on the edge of $\mathcal{C}^i$ opposite to p_j with respect to p_i . The repulsion point for Robot i to avoid collision with Robot j is then defined by

$$P_r^{j,i} = \Psi\left(p_i, I_r^{ji}, -d_r^{ji}\right) \tag{10}$$

with the distance

$$d_r^{ji} = \min\left(d_r^{max}, \lambda_r \left\| I_r^{ji} - p_i \right\|\right). \tag{11}$$

The parameter λ_r, such that $0 < \lambda_r < 1$, is used to set the position of the repulsion point $P_r^{j,i}$ on the segment $\overline{p_i\, I_r^{ji}}$, and $d_r^{max} > 0$ to limit its distance to Robot i.

Following this procedure, one repulsion point is computed by Robot i for each robot with collision risk. A global repulsion point is then deduced for Robot i by

$$P_r^i = \frac{1}{N_{rep}^i} \sum_{j=1}^{N_{rep}^i} P_r^{j,i} \tag{12}$$

with $N_{rep}^i = Card\left\{\mathcal{N}_{rep}^i\right\}$ the number of robots to be considered for repulsion. Since all the $P_r^{j,i}$ are located inside the Voronoi cell $\mathcal{C}^i$, so does the global repulsion point P_r^i. Note that, by relation (12), P_i^r is computed as a mean of the $P_r^{j,i}$. A weighted mean could also be used for example to give more influence to repulsion points corresponding to the closest robots.

Step 5: Computation of reference

Similarly to [26], the reference position P_*^i that will be tracked by robot i is computed as a weighted mean of the attraction point P_a^i and the repulsion point P_r^i as

$$P_*^i = (1 - \beta) P_r^i + \beta P_a^i \tag{13}$$

where the weighting coefficient $0 \leq \beta \leq 1$ is adapted online depending on the minimum distance to other agents with collision risk. It enables to give more weight on repulsion if some UGVs are very close or more weight on attraction to the waypoint otherwise. If there are no collision risks between the agents ($\beta = 1$), this algorithm results in $P_*^i = P_a^i$, leading to pure attraction to the waypoint.

2.3. Waypoint and Velocity Management

Navigation to successive waypoints has been managed in the following way. For the *Autonomous* and *Velocity-Guided* modes, all vehicles dispose of the full list of waypoints assigned to the swarm from the mission path definition. In *Follow* mode, the waypoint is not predefined from a list but generated dynamically from the position of a target (which could be a localized human operator, a tele-operated robot, or a virtual point). An isosceles triangle of spacers is built with its height defined as the segment from the target towards the centroid of the swarm with a length of d_{follow} and a base of length equal to d_{follow_base}. The waypoint is set at the intersection of the height and the base. An additional *little thumb* sub-mode has been specified, in which waypoints are recorded in a list to be tracked by the robots of the swarm, whenever the target is located at a distance greater than d_{little_thumb} from the robots (note that the distance used here is built in coherence with the mode of validation defined in the following paragraph). An illustration of these definitions is presented in Figure 6.

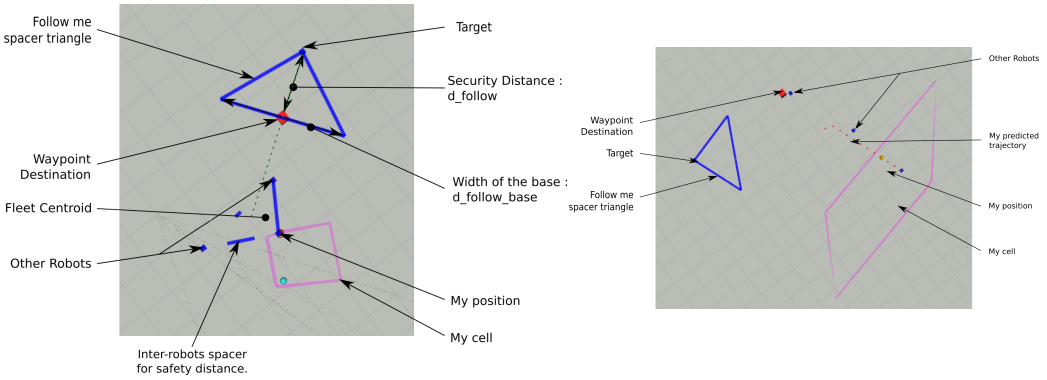

Figure 6. Left: *Follow* mode waypoint and triangle of spacers. **Right**: *little thumb* sub-mode.

Different validation strategies can be defined for the UGVs to determine that the current waypoint has been reached and that they should continue to the next waypoint. We have studied the following strategies:

1. *Selfish*: each robot has to validate its current waypoint (given a parameterized validation distance d_{val}), then it moves to the next waypoint in the list. This way, all the robots will cross each waypoint and stay close to the path. On the other hand, this does not impose any waiting behavior between the robots.
2. *First*: when a robot is the first to validate the current waypoint, all robots head to the next waypoint in the list by sharing its index. This strategy can be applied in large environments where deviation from the path can be allowed. There is also no waiting behavior around each waypoint in this case, however the UGVs always agree on and head towards the same current waypoint.
3. *WaitForAll*: in this strategy, the waypoint is validated only if each robot either gets closer to the waypoint than the validation distance d_{val} or if it is near the avoidance distance of another robot ($\sigma_{chain} d_{col}$) which validates one of these conditions. This is a more collective behavior, where all robots should wait for the others before heading to the next waypoint. This also creates a kind of validation chain between the UGVs, which is a useful feature for large swarms where all the UGVs cannot get closer to the waypoint than the validation distance because of the collision avoidance constraints.

Since the positions of the robots are shared within the swarm at all time instants, each robot is able to compute the validation conditions for itself and its neighbors in a distributed way. In the experiments described in this paper, the *WaitForAll* mode has been preferred to demonstrate group motion around obstacles.

In addition to the computation of the reference point to be tracked by the robot, a speed ratio is also produced by the algorithm. This speed ratio multiplies the speed value that

has been chosen initially by the user, so as to provide the current reference speed to the low-level controller. In the *Autonomous* mode, the speed ratio is set to 0.5 in case of collision risk (slow motion in presence of obstacles) and to 1 otherwise (full-speed motion). In the *Velocity-Guided* mode, the speed ratio is set to copy the speed of the human operator, considered as command for the swarm, with a saturation at 1, which means the operator can go faster than the robot's maximal reference speed. In case of collision risk, the robot's speed ratio is saturated at 0.5, while still copying the operator's speed below this value. In the *Follow* mode, the speed ratio is set to 0 if the robot enters the triangle, otherwise it is set to 1. This speed modulation makes it possible for the target to turn around and come back towards the robots, which are forced to stop during the crossing.

2.4. Main Properties of the Algorithm

2.4.1. Safety Regions

Voronoi cells can be viewed as safety regions in the sense that if each robot performs a trajectory within its cell to reach its reference point to be tracked, collisions between the vehicles can be avoided. This safety consideration is enforced by the additional spacers introduced, which isolate the robot cells at a desired distance from each other.

Note that since the approach is distributed, each robot will compute its own Voronoi partition and cell. Non-overlapping of the cells can only be guaranteed in case of fully connected communication graphs (i.e., $N^i = N - 1, \forall i$) and if this computation is done in a synchronized way, with all the robots disposing of information corresponding to the same situation of the swarm. In practice, as we do not want to enforce a synchronization mechanism, non-overlapping of the cells can be obtained if the vehicle dynamics are slower than the computation period of the Voronoi partition, as mentioned also in [23,24]. In addition, parameters λ_a and λ_r can be chosen to define margins with respect to the edges of the Voronoi cell in the placement of P_a^i, P_r^i and hence P_*^i, and to keep each robot and its reference to be tracked in a segregated partition of the space. In case of non fully connected communication graphs, other mechanisms must be looked at to provide non-overlapping guarantees for the Voronoi cells.

2.4.2. Flexibility and Pattern of the Swarm

Flexibility of the swarm can be adjusted by the parameters d_{mir} and d_{col} which set a compromise between attraction and repulsion between the robots. Choosing $d_{mir} \gg d_{col}$ adds more flexibility to the swarm. A swarm behavior close to a more rigid-formation can be obtained on the contrary for $d_{mir} \approx d_{col}$.

The pattern obtained for the swarm is not pre-specified but can be influenced by the initial positioning of the robots, making this feature an interesting one for practical applications. For instance a pattern close to a "platooning-like" formation can be obtained for an initial positioning of the robots close to a single line. This can be of interest for motion in narrow corridors. More regular patterns (triangle, square, etc.) can also be obtained during the motion by the same consideration. The formation shape will although be distorted in presence of obstacles between or near the waypoints. This trade-off makes it possible to carry out the autonomous navigation of a swarm of UGVs in large-scale environments with various levels of obstacle density, which cannot be achieved with control methods based on rigid formations or virtual structures.

2.4.3. Decentralized Algorithm for Robustness to Robot Failure and Communication Loss

In practice, the number of robots in the swarm and/or in the neighborhood of each robot may vary during the mission: loss or addition of robots, communication links temporarily/definitively unavailable, etc. Robustness with respect to these issues is ensured in practice by the proposed algorithm, being fully distributed and handling the limit cases with one or two robots. The experimental testing of this behavior was reported with a swarm of four mobile robots in [26] using a previous version of the algorithm which did not take obstacles into account.

3. System Architecture

3.1. Architecture

A global architecture to deploy the proposed algorithm in a swarm of autonomous robots is summarized in Figure 7. A global mission supervision module is available to the operator on a portable ground station to select a swarm mode, its parameters, and a reference path (for the *Autonomous* or *Velocity-Guided* mode) which are sent to all the UGVs. Once the mission has started, the algorithm ensures the self-organization of the swarm and the supervision layer is only used for monitoring progress. Each robot runs its own localization, mapping and control algorithms in a distributed scheme with limited exchanges of information.

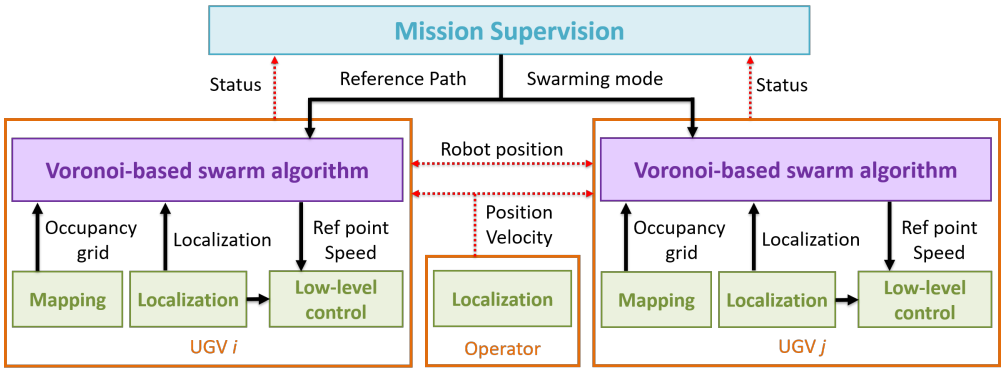

Figure 7. Architecture of the proposed multi-robot system.

The robots and the operator are equipped with localization sensors which provide their own global position, orientation and velocity with respect to a common reference frame (e.g., a WGS-84 local frame or a reference landmark). The mission path is also defined or converted with respect to this reference frame, such that all UGVs share the same list of waypoints. The position of each agent is the only information shared with the others during the mission to be able to execute the distributed swarming algorithm. The robots individually run a mapping process using their own embedded depth measurements to provide an occupancy grid centered on their current position, which is then processed by the proposed algorithm to estimate and keep track of the position of the closest obstacles while distinguishing them from the other robots (see next section for more details).

Based on each robot's own localization and mapping, and the shared localization of the other active agents of the swarm (UGVs and operator), the Voronoi-based swarm algorithm generates a local reference point and a speed modulation ratio on each robot which takes into account all the mission objectives and constraints. A low-level controller is then used to track this reference point and modulate the speed as requested. In our architecture, a proportional controller for a unicycle model derived from [36] was used to compute the steering inputs. The controller has been implemented so as to prefer trajectories close to straight lines in the direction of waypoints by correcting first large orientation errors at a lower speed. This is of a particular importance especially when dealing with nonholonomic vehicles, to obtain trajectories remaining inside the Voronoi cells.

The exchange of information between elements of the system has been kept as simple as possible, both in terms of the nature of the information and of the associated data flow to facilitate interoperability capacities as much as possible. This way, they do not depend on the choice of the technology composing the system, and the required communication bandwith is also quite limited. In a more prospective view, interoperability with aerial vehicles or manned wheeled vehicles could be facilitated and envisaged as long as the same type of communication interfaces can be handled. At the robot level, the modularity of the swarm module would also enable to modify or mix the types of vehicles, by only adapting the low-level control layer and distance parameters, without changing the complete architecture of the system.

3.2. Local Mapping from Embedded Depth Sensors

As mentioned in Step 3 of the algorithm description, obstacle avoidance is based on a local map of the environment. This local map is built from an occupancy grid in robot frame provided by a pre-processing mapping algorithm from raw depth data (either from a LiDAR or stereo cameras). It aims to remove suspicious obstacles by temporal filtering and extend obstacle memory when obstacles go out of sight. The local map is updated when a new occupancy grid is received. The track of each obstacle is kept in memory as: the position of the obstacle (in world frame), the number of times it has been observed in successive occupancy grids and at which time it occurred last. Four main steps are carried out in the mapping process, as described in Figure 8. The predicted trajectory mentioned in the following is obtained by repeating the main algorithm steps on a given time horizon.

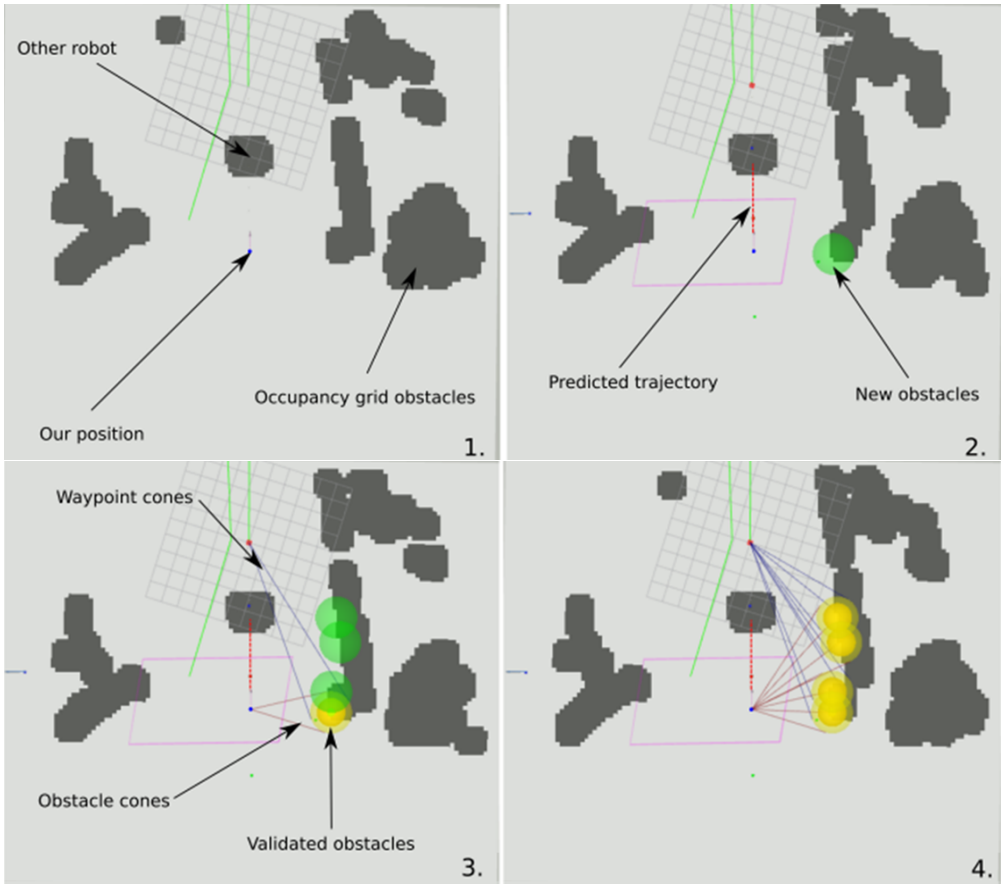

Figure 8. Local map building process. 1—the raw occupancy grid is in the background and consists of dark gray spots. 2—the closest obstacle to the robot is initialized in green. 3—Obstacles closest to the predicted trajectory are initialized and the first obstacle is validated (yellow, obstacle cones are visible). 4—All obstacles validated are in yellow, we can observe that another robot in front of the one building this map is not added to the obstacle list.

1. *Obstacle initialization*
   The local map does not keep track of all obstacles detected by the sensor but focuses only on the most threatening ones. Those are obstacles which are closest to the current position or to the future position along the predicted trajectory. When a possible new obstacle is found, we add it to the local map if it is not already tracked. The created obstacle is a disk of radius d_{des} (parameterized safety distance) centered on the grid point of a threatening obstacle, displayed in green in Figure 8 (sub-figures 2 and 3). All the new obstacles are created and tracked but not taken into account for avoidance immediately. This requires that the corresponding location has been seen as occupied

a certain number of times in successive occupancy grids to filter out measurement artefacts (vegetation, dust, ...).

2. *Obstacle life cycle*
   When a new occupancy grid becomes available, each tracked obstacle is projected in the occupancy grid to check if it is still present. If this is the case, we increase the number of observations of this obstacle and reset the last time it has been seen. When an obstacle reaches a certain number of observations (set to 3 in our experiments), it is validated and considered for obstacle avoidance (in yellow in the map). In this algorithm, the radius of the obstacle is enlarged by an additional distance d_{safe} to take into account the size of the robot or the drift of localization (in light yellow on the map pictures). This number of observations is saturated at a given threshold (set to 100 in our experiments) to be able to remove obstacles that have been seen during a long period at a given location but have moved away afterwards.
3. *Obstacle removal*
   After new obstacles are initialized, tracked and updated, obstacle removal is carried out. This process is based on the number of views and the last time an obstacle has been seen. When an obstacle is not seen anymore: after a certain amount of time passed from the last time it has been seen (set to 1 s in our experiments), the number of observations is decremented. When the number of observations reaches the minimum-view threshold, the obstacle is removed from the map. This process allows to keep track of an obstacle that has been seen for an extended period of time while allowing for a fast removal of an obstacle that has been seen just a short number of times. This process will also remove the obstacles that have been initialized but never seen again (validated).
4. *Occupancy grid pre-filtering*
   Due to the multi-agent context, the input occupancy grid is pre-processed to remove views of the other agents (UGV or operator) such that they are not considered as obstacles, since they are moving and taken into account directly at Step 4 of the main algorithm. The global positions of the other robots and of the operator are projected in the local grid. All obstacle cells present in a given radius (chosen to be consistent with the robot dimensions and localization uncertainty) around these positions are then removed. This can be seen in Figure 8, where the spot in front of the robot currently building the map is at the location of another robot but is not considered as an obstacle.

4. Experimental Results

Outdoor and indoor experiments were carried out for the different modes of the swarming algorithm using the following robotic platforms (see Figures 9 and 10):

- 3 Robotnik Summit XL UGVs of mass 65 kg and base dimension 72 × 61 cm, equipped with a calibrated stereo-rig of IDS UI-3041LE cameras (baseline of 35 cm, running at 20 Hz) and an embedded Intel-NUC CPU.
- 1 operator Portable Localization Kit comprising an Intel RealSense d455 depth camera and a Intel-NUC CPU.
- A standard WiFi network connecting all the embedded computers and a ground station for mission supervision and visualization. Transmission of the positions between the robots was carried out using Ultra Wideband (UWB) DW1000 radio modules in a similar way as in [37].

Figure 9. *Velocity-Guided* outdoor experiment: two UGVs and localized operator.

Each agent (UGV or operator) computed on-board its localization from the stereo-vision data using the eVO visual odometry algorithm [38]. An additional initialization procedure was used to align the local frame of each individual visual odometry with the same global frame. In our experiments, the global frame was defined by a cube of AprilTags [39] and each robot computed its initial global position automatically by estimating its relative position and orientation with respect to this cube (Figure 10). This procedure allows to estimate a global position as long as the visual odometry presents a limited drift. In the presented experiments, the drift at the end of the trajectories was observed to be less than 0.5 m for trajectories of about 100 m. This can be considered accurate enough despite possible perturbations on visual odometry due to the presence of the other moving robots in the field of view, and this did not disturb the demonstration of control performance and was compatible with the desired inter-robot safety distances. For a larger-scale scenario, other visual localization methods should be considered like collaborative localization and distributed SLAM [40]. The UGVs used on-board the ELAS algorithm [41] to estimate the depth from the stereo images, which was then converted into a point cloud and projected in an occupancy grid of obstacles with a resolution of 25 cm as an input for the mapping process. The update rate of the swarming algorithm was set to 4 Hz, which was also the availability rate of the obstacle grid.

Algorithm tuning

The following parameters were used for all the experiments:

- Collisions: $d_{col} = 2.0$ m, $\sigma_{col} = 2$, $\sigma_{rep} = 1$
- Attraction: $d_a^{max} = 2.0$ m, $\lambda_a = 0.75$
- Repulsion: $d_r^{max} = 2.0$ m, $\lambda_r = 0.9$
- Mirrors: $d_{mir} = 3.0$ m
- Validation: $d_{val} = 2.0$ m, $\sigma_{chain} = 1.8$
- Obstacles: $d_{des} = 1.1$ m, $d_{safe} = 0.5$ m

Figure 10. Indoor experimental environment: UGVs and portable localization kit with reference cube.

4.1. Autonomous Mode

Figure 11 presents the results of a repeated experiment on the same reference path in *Autonomous* mode, where the number and starting positions of the robots varied. The experiment comprises 5 runs with a single robot, 6 runs with two robots, and 3 runs with three robots. The path is a loop of length 68.3 m, with two main obstacles along the way: a box (blue in Figure 10) placed in collision between two waypoints, and a large pillar (top left in Figure 10) which is not in direct collision but where there is not enough space for two robots to pass side by side (thus forcing them to change the shape of the formation or wait for the path to clear). It could be seen that with one robot, the achieved path is close to be the same for each run: some variability only occurs on the choice to go left or right around the obstacle box which is placed nearly symmetrically. With two robots, the first one keeps a trajectory close to the one-robot case but it could seen that the leading robot can change during the mission, around the pillar the second one needs to move closer to the obstacle to keep its safety distance with the first one. With three robots, the trajectories exhibit the same kind of pattern but with more diversity. The lengths of the traveled paths obtained during all the experiments are reported in Table 1, with associated statistics. It is interesting to note that experiments with more robots did not increase the path length as late robots are able to find shortcuts under the *WaitForAll* validation mode, since they are not required to reach exactly the waypoint but should only be at a validation distance from it. Another validation mode which would combine the *First* and *WaitForAll* behaviors could be designed to enforce the crossing of the waypoints by each robot if this is of interest for a given mission.

Table 1. *Autonomous* mode experiments: lengths of robot trajectories.

	Nb of Runs	**Mean (m)**	**Std (m)**	**Max (m)**	**Min (m)**
Waypoints		68.3			
1 Robot	5	67.29	2.35	71.33	65.16
2 Robots	6	69.19	2.86	73.28	64.84
3 Robots	3	65.05	4.37	74.10	58.29

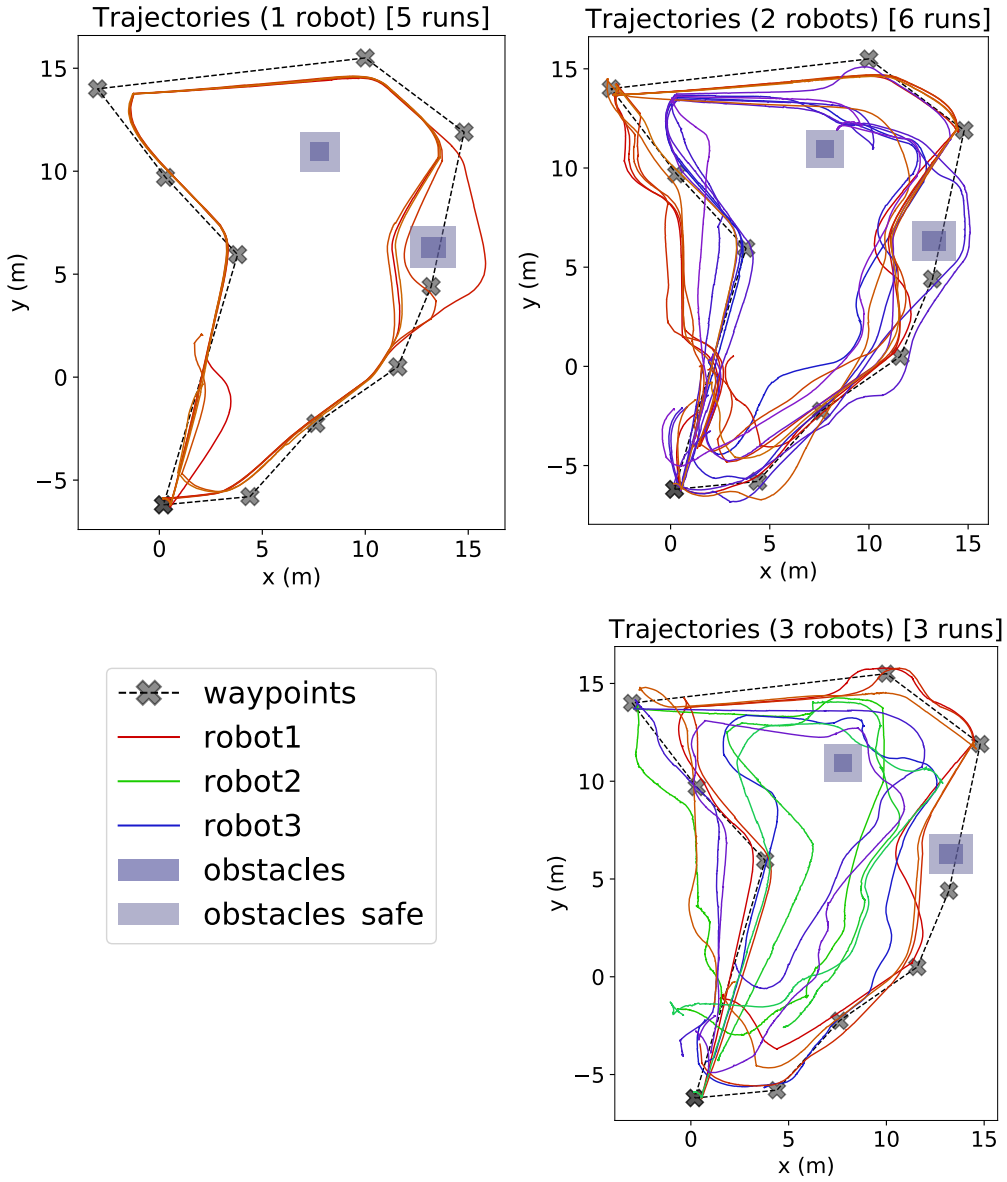

Figure 11. *Autonomous* mode: repeated experiment with 1, 2 and 3 robots on the same reference path.

If a human operator is present in this *Autonomous* mode, it is not considered as an obstacle but as an additional agent of the swarm in the Voronoi partition, while the UGVs follow the path autonomously. In this case, the repulsion mechanism described in Step 4 of the swarming algorithm applies. A dedicated test presented in Figure 12 illustrates the way the localized operator can influence the other robots, where the operator walks between two robots and forces them to make room: the red robot stops and the green one is forced to deviate from its trajectory.

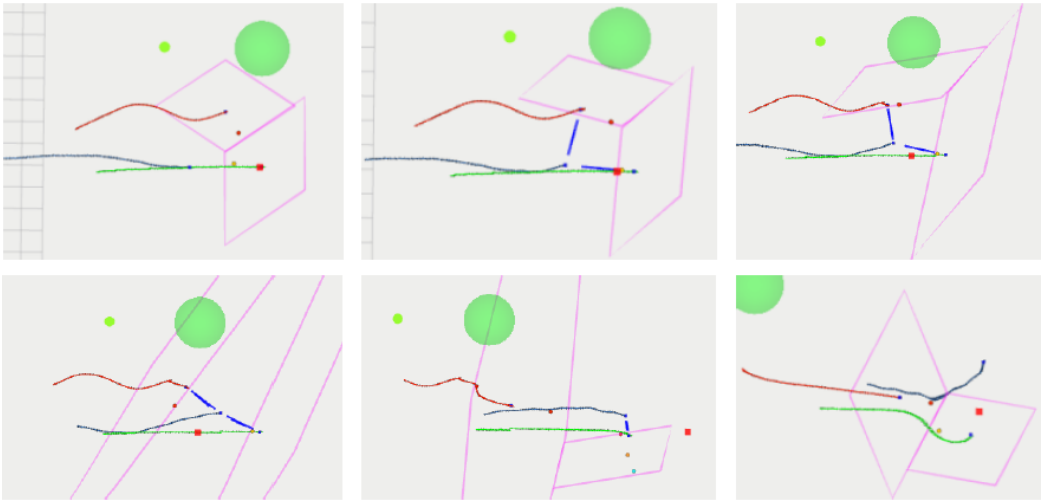

Figure 12. *Autonomous* mode with Human Operator interaction. Each vignette shows a situation with time flowing from left to right on the top then the bottom rows. The Voronoi cell of each robot is represented in magenta, the trajectory of each robot is in red and green, the operator is in blue.

4.2. Velocity-Guided Mode

In this test, two UGVs follow a 8-shaped trajectory (Figure 13). It can be checked that both robots adapt their velocity to the one of the localized human operator, and that safety distances are respected at all times. In particular, it can be seen that the velocities of the robots and the operator are correctly superposed, except when the operator goes faster than the UGV nominal velocity (e.g., a little after time 80 s) or when an obstacle is along the way (at the end of the trajectory). These behaviors are fully consistent with the distributed algorithm and the imposed requirements.

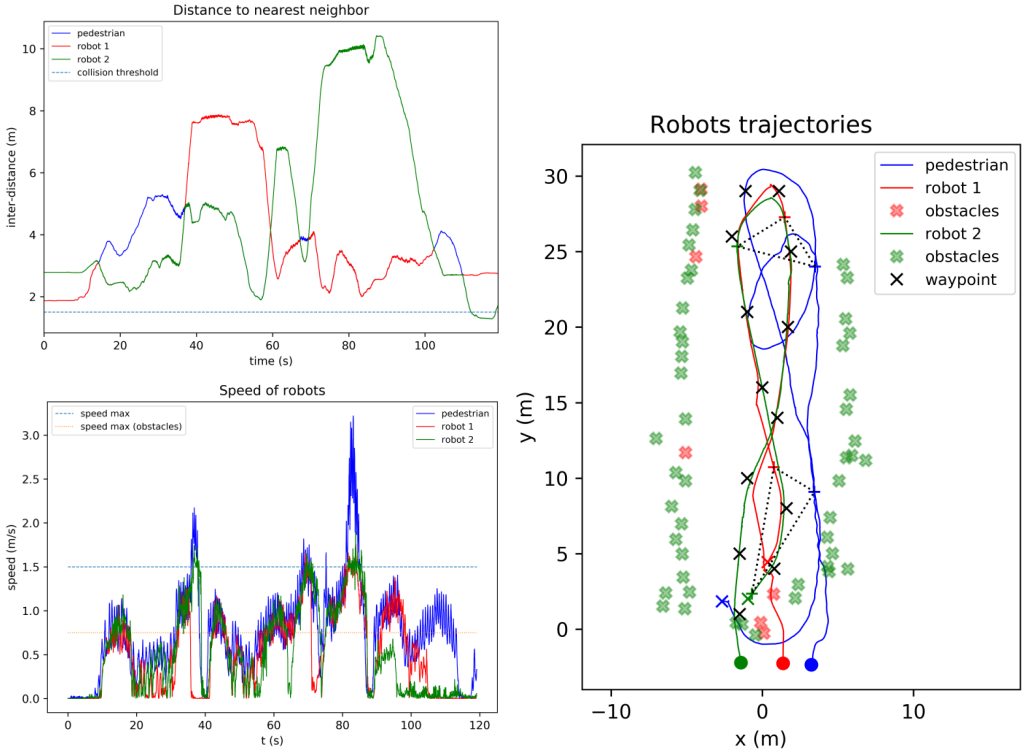

Figure 13. *Velocity-Guided* mode for two UGVs interacting with a localized human operator. **Right**: Trajectories followed by the agents. **Up left**: inter-distances between robots and with operator (safety is always ensured). **Bottom left**: superposition of robots and operator velocities.

4.3. Follow Mode

As described in Section 2.3, the behavior of the *Follow* mode is parameterized by:

- The minimal distance d_{follow} to which a robot could approach the target.
- The d_{follow_base} parameter, which has an influence on the shape of the robots formation behind the target. If the base length is short, the robots will be more in line, while if it is larger a triangular shape will emerge. A standard tuning was chosen equal to $2d_{col}$.

In the following experiments, $d_{follow} = 1.0$ m and $d_{follow\_base} = 2.0$ m. The *little thumb* sub-mode threshold was set to 8 m but never reached during the presented experiments, where the UGVs always followed closer the localized operator.

Figure 14 presents the trajectories of a localized pedestrian and one robot in *Follow* mode, while evolving in the same indoor setup from Figure 10 as the *Autonomous* mode tests, with a total path length of 237 m. In this experiment, the pedestrian forces the robot to make multiple loops around a pillar located around $x = 10$ m, $y = -10$ m and a box located at $x = 15$ m, $y = 5$ m to successfully demonstrate the obstacle avoidance capability in this mode. It can also be seen that the respective velocities are correctly synchronized in time, even if the human operator walks faster than the robot reference speed.

In Figure 15, the human operator traveled a path of 190 m with three robot followers, still in the same indoor environment. The graph on the left shows the relative distances between each agent and its nearest neighbor, which allows to verify that the UGVs and the human operator respect their safety distance from each other during the experiment (the distance values before time 80 s are related to a manual initialization phase before the start of the mission). Note that some of the obstacles detected by robot 2 are in fact the pedestrian, which is an artefact showing that the localization of this robot presented a drift (thus, of the order of magnitude of d_{col}). This is however a "fail-safe" case, since an obstacle is created at the perceived location and not erased in the mapping process, thus the avoidance is still achieved even if this might degrade the actual tracking of the operator. The localization of the robots could be improved by using inter-robot measurements (e.g., from UWB radio signals or vision processing), but this was out of the scope of this paper which is centered on the swarming control and operator interaction performances. The proposed swarm algorithm will remain applicable with relative positioning of all the agents, although the mapping process should be adapted accordingly.

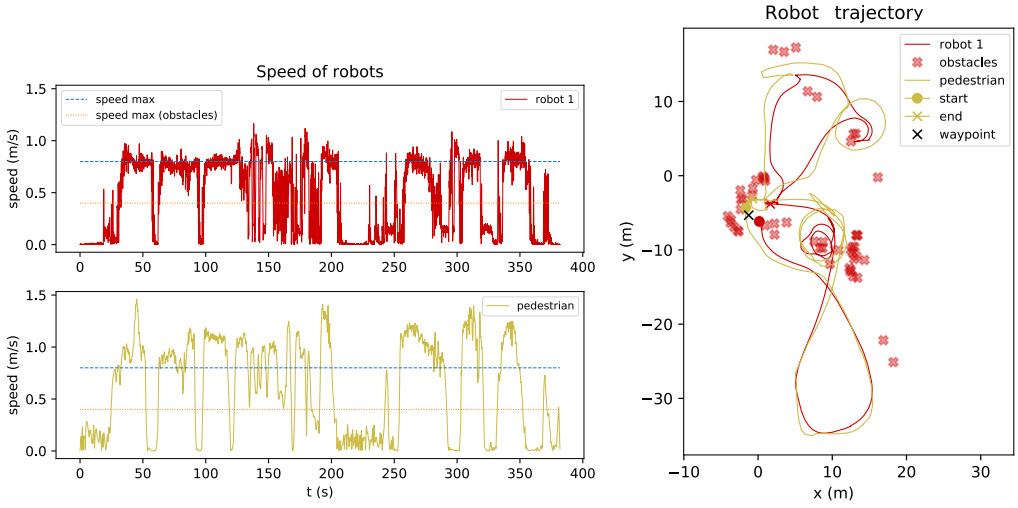

Figure 14. *Follow* mode with 1 robot: robot and pedestrian velocities (**left**) and trajectories (**right**).

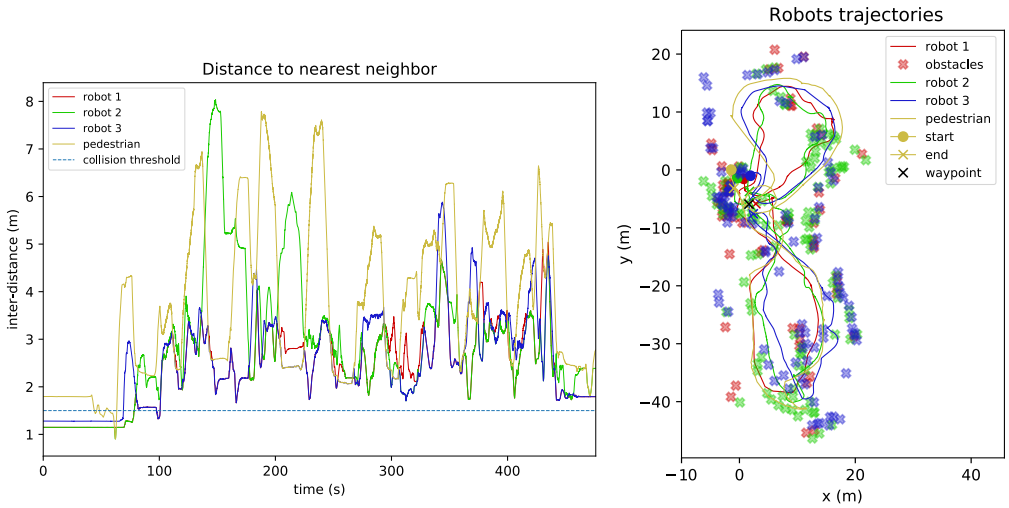

Figure 15. *Follow* mode with 3 robots: relative distances (**left**) and trajectories (**right**).

5. Conclusions and Perspectives

A new distributed control algorithm based on Voronoi partitioning and collision cones has been proposed to coordinate the navigation of a swarm of unmanned ground vehicles interacting with a localized human operator in unknown cluttered environments. Results from field experiments with mobile ground robots have been presented, illustrating a non-rigid swarm motion capability for several navigation modes, all including collision and obstacle avoidance. The behavior of the entire swarm can be easily reshaped by only modifying how the waypoint objectives and the reference speeds are defined, resulting in a range of possible interactions with the swarm for the operator. More elaborate interaction modes are foreseen for future work (e.g., formation split-and-merge or adaptation to multiple operators), as well as larger-scale experiments.

Author Contributions: All authors contributed equally. All authors have read and agreed to the published version of the manuscript.

Funding: This project has received funding from the European Commission through the European Union's Preparatory Action for Defence Research—PADR programme under grant agreement No. 883465—ARTUS.

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

Appendix A

This Appendix addresses the special cases corresponding to $N^i = 0$ or $N^i = 1$. Is has been proposed by the authors in [26] and is recalled here for completeness.

If at a given instant, Robot i has zero or one neighbor, the following additional step is performed before the standard algorithm.

- If $N^i = 0$, Robot i has no neighbors. In this case, the robot only executes Steps 2 & 3 for obstacle avoidance and does not compute the Voronoi partitioning and collision avoidance with neighbors. Finally, the robot will track the reference points $P^i_* = P^i_a$.
- If $N^i = 1$, Robot i has only one neighbor. In this case, the Voronoi cell obtained in Step 1 will be degenerated. To avoid this problem, if $N^i = 1$, two virtual robots are defined as follows and their indices v_1, v_2 are added to $\mathcal{N}^i$.

Let $(u_{ij}, u_{ij}^{\perp})$ denote the local reference frame attached to Robot i such that u_{ij} is the unit vector directed from Robot i to its neighbor Robot j, i.e., $u_{ij} = (p_j - p_i)/d_{ij}$ with $d_{ij} = \|p_j - p_i\|$ and $j \in \mathcal{N}^i$. The second unit vector $u_{ij}^{\perp}$ completes the orthonormal basis (see Figure A1). The positions of the two virtual robots are defined by:

$$V_{ij}^1 = p_i + \frac{d_{ij}}{2} u_{ij} + d_{mir} u_{ij}^{\perp} \tag{A1}$$

$$V_{ij}^2 = p_i + \frac{d_{ij}}{2} u_{ij} - d_{mir} u_{ij}^{\perp} \tag{A2}$$

They are artificially added to the neighborhood of Robot i, i.e., $\mathcal{N}^i \cup \{v_1, v_2\} \rightarrow \mathcal{N}^i$, and with $p_{v_1} = V_{ij}^1, p_{v_2} = V_{ij}^2$. Steps 1 to 5 of the algorithm are then executed by Robot i. By construction, these two virtual robots are located at a distance greater than d_{col} and will not be involved in collision risk.

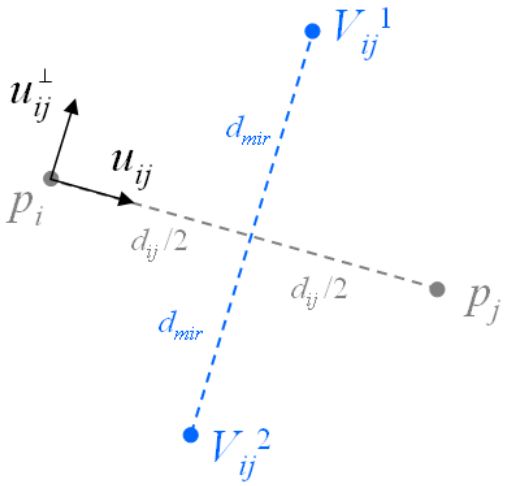

Figure A1. Positioning of virtual robots in case of $N_i = 1$. Reproduced from [26].

References

1. Moussa, M.; Beltrame, G. On the robustness of consensus-based behaviors for robot swarms. *Swarm Intell.* **2020**, *14*, 205–231. [CrossRef]
2. Adoni, W.Y.H.; Lorenz, S.; Fareedh, J.S.; Gloaguen, R.; Bussmann, M. Investigation of Autonomous Multi-UAV Systems for Target Detection in Distributed Environment: Current Developments and Open Challenges. *Drones* **2023**, *7*, 263. [CrossRef]
3. Murray, R.M. Recent research in cooperative control of multivehicle systems. *J. Dyn. Syst. Meas. Control* **2007**, *129*, 571–583. [CrossRef]
4. Mesbahi, M.; Egerstedt, M. *Graph Theoretic Methods in Multi-Agent Networks*; Princeton University Press: Princeton, NJ, USA, 2010.
5. Ren, W.; Beard, R.W. *Distributed Consensus in Multi-Vehicle Cooperative Control*; Springer: Berlin/Heidelberg, Germany, 2010. [CrossRef]
6. Canepa, D.; Potop-Butucaru, M.G. Stabilizing Flocking via Leader Election in Robot Networks. In *Stabilization, Safety, and Security of Distributed Systems*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 52–66. [CrossRef]
7. Balch, T.; Arkin, R.C. Behavior-based formation control for multirobot teams. *IEEE Trans. Robot. Autom.* **1998**, *14*, 926–939. [CrossRef]
8. Lee, G.; Chwa, D. Decentralized behavior-based formation control of multiple robots considering obstacle avoidance. *Intell. Serv. Robot.* **2018**, *11*, 127–138. [CrossRef]
9. Zhou, D.; Wang, Z.; Schwager, M. Agile coordination and assistive collision avoidance for quadrotor swarms using virtual structures. *IEEE Trans. Robot.* **2018**, *34*, 916–923. [CrossRef]
10. Kahn, A.; Marzat, J.; Piet-Lahanier, H. Formation flying control via elliptical virtual structure. In Proceedings of the IEEE International Conference on Networking, Sensing and Control, Paris-Evry, France, 10–12 April 2013; pp. 158–163. [CrossRef]
11. Lafferriere, G.; Williams, A.; Caughman, J.; Veerman, J.J.P. Decentralized control of vehicle formations. *Syst. Control Lett.* **2005**, *54*, 899–910. [CrossRef]
12. Oh, K.K.; Park, M.C.; Ahn, H.S. A survey of multi-agent formation control. *Automatica* **2015**, *53*, 424–440. [CrossRef]

13. Fathian, K.; Rachinskii, D.I.; Spong, M.W.; Summers, T.H.; Gans, N.R. Distributed formation control via mixed barycentric coordinate and distance-based approach. In Proceedings of the American Control Conference, Philadelphia, PA, USA, 10–12 July 2019; pp. 51–58. [CrossRef]
14. Cheah, C.C.; Hou, S.P.; Slotine, J.J.E. Region-based shape control for a swarm of robots. *Automatica* **2009**, *45*, 2406–2411. [CrossRef]
15. Strandburg-Peshkin, A.; Twomey, C.R.; Bode, N.W.; Kao, A.B.; Katz, Y.; Ioannou, C.C.; Rosenthal, S.B.; Torney, C.J.; Wu, H.S.; Levin, S.A.; et al. Visual sensory networks and effective information transfer in animal groups. *Curr. Biol.* **2013**, *23*, R709–R711. [CrossRef]
16. Kolpas, A.; Busch, M.; Li, H.; Couzin, I.D.; Petzold, L.; Moehlis, J. How the Spatial Position of Individuals Affects Their Influence on Swarms: A Numerical Comparison of Two Popular Swarm Dynamics Models. *PLoS ONE* **2013**, *8*, e58525. [CrossRef] [PubMed]
17. Cortes, J.; Martinez, S.; Karatas, T.; Bullo, F. Coverage control for mobile sensing networks. *IEEE Trans. Robot. Autom.* **2004**, *20*, 243–255. [CrossRef]
18. Guruprasad, K.R.; Dasgupta, P. Distributed Voronoi partitioning for multi-robot systems with limited range sensors. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, Portugal, 7–12 October 2012; pp. 3546–3552. [CrossRef]
19. Hatleskog, J.; Olaru, S.; Hovd, M. Voronoi-based deployment of multi-agent systems. In Proceedings of the IEEE Conference on Decision and Control, Miami, FL, USA, 17–19 December 2018; pp. 5403–5408. [CrossRef]
20. Zhou, Z.; Zhang, W.; Ding, J.; Huang, H.; Stipanović, D.M.; Tomlin, C.J. Cooperative pursuit with Voronoi partitions. *Automatica* **2016**, *72*, 64–72. [CrossRef]
21. Kouzeghar, M.; Song, Y.; Meghjani, M.; Bouffanais, R. Multi-Target Pursuit by a Decentralized Heterogeneous UAV Swarm using Deep Multi-Agent Reinforcement Learning. In Proceedings of the IEEE International Conference on Robotics and Automation, London, UK, 29 May–2 June 2023.
22. Gui, J.; Yu, T.; Deng, B.; Zhu, X.; Yao, W. Decentralized Multi-UAV Cooperative Exploration Using Dynamic Centroid-Based Area Partition. *Drones* **2023**, *7*, 337. [CrossRef]
23. Lindhé, M.; Ogren, P.; Johansson, K.H. Flocking with obstacle avoidance: A new distributed coordination algorithm based on Voronoi partitions. In Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, Spain, 18–22 April 2005; pp. 1785–1790. [CrossRef]
24. Lindhé, M.; Johansson, K.H. A Formation Control Algorithm using Voronoi Regions. In *Taming Heterogeneity and Complexity of Embedded Control*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2013; Chapter 24; pp. 419–434. [CrossRef]
25. Jiang, Q. An improved algorithm for coordination control of multi-agent system based on r-limited Voronoi partitions. In Proceedings of the IEEE International Conference on Automation Science and Engineering, Shanghai, China, 7–10 October 2006; pp. 667–671. [CrossRef]
26. Bertrand, S.; Sarras, I.; Eudes, A.; Marzat, J. Voronoi-based Geometric Distributed Fleet Control of a Multi-Robot System. In Proceedings of the 16th International Conference on Control, Automation, Robotics and Vision (ICARCV), Shenzhen, China, 13–15 December 2020; pp. 85–91. [CrossRef]
27. Chakravarthy, A.; Ghose, D. Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Trans. Syst. Man, Cybern. Part A Syst. Hmans* **1998**, *28*, 562–574. [CrossRef]
28. Sunkara, V.; Chakravarthy, A.; Ghose, D. Collision Avoidance of Arbitrarily Shaped Deforming Objects Using Collision Cones. *IEEE Robot. Autom. Lett.* **2019**, *4*, 2156–2163. [CrossRef]
29. Hu, J.; Wang, M.; Zhao, C.; Pan, Q.; Du, C. Formation control and collision avoidance for multi-UAV systems based on Voronoi partition. *Sci. China Technol. Sci.* **2020**, *63*, 65–72. [CrossRef]
30. Moniruzzaman, M.D.; Rassau, A.; Chai, D.; Islam, S.M.S. Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey. *Robot. Auton. Syst.* **2022**, *150*, 103973. [CrossRef]
31. Aggravi, M.; Sirignano, G.; Giordano, P.R.; Pacchierotti, C. Decentralized Control of a Heterogeneous Human–Robot Team for Exploration and Patrolling. *IEEE Trans. Autom. Sci. Eng.* **2021**, *19*, 3109–3125. [CrossRef]
32. Sydorchuk, A. The Boost Polygon Voronoi Extensions. 2013. Available online: https://www.boost.org/doc/libs/1_60_0/libs/polygon/doc/voronoi_main.htm (accessed on 3 July 2023).
33. Pereyra, E.; Araguás, G.; Kulich, M. Path planning for a formation of mobile robots with split and merge. In Proceedings of the 4th International Conference on Modelling and Simulation for Autonomous Systems, Rome, Italy, 24–26 October 2017; Springer: Berlin/Heidelberg, Germany, 2018; pp. 59–71. [CrossRef]
34. Salvado, J.; Mansouri, M.; Pecora, F. Combining multi-robot motion planning and goal allocation using roadmaps. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; pp. 10016–10022. [CrossRef]
35. Fortune, S. A sweepline algorithm for Voronoi diagrams. In Proceedings of the Second Annual Symposium on Computational Geometry, Yorktown Heights, NY, USA, 2–4 June 1986; pp. 313–322.
36. Bak, M.; Poulsen, N.K.; Ravn, O. *Path Following Mobile Robot in the Presence of Velocity Constraints*; Technical Report, Informatics and Mathematical Modelling; Technical University of Denmark: Lyngby, Denmark, 2001; Available online: http://www2.compute.dtu.dk/pubdb/pubs/189-full.html (accessed on 19 September 2023).
37. Guo, K.; Li, X.; Xie, L. Ultra-wideband and Odometry-Based Cooperative Relative Localization with Application to Multi-UAV Formation Control. *IEEE Trans. Cybern.* **2020**, *50*, 2590–2603. [CrossRef]

38. Sanfourche, M.; Vittori, V.; Le Besnerais, G. eVO: A realtime embedded stereo odometry for MAV applications. In Proceedings of the IEEE/RSJ IROS, Tokyo, Japan, 3–7 November 2013; pp. 2107–2114. [CrossRef]
39. Olson, E. AprilTag: A robust and flexible visual fiducial system. In Proceedings of the IEEE international Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 3400–3407. [CrossRef]
40. Tong, P.; Yang, X.; Yang, Y.; Liu, W.; Wu, P. Multi-UAV Collaborative Absolute Vision Positioning and Navigation: A Survey and Discussion. *Drones* **2023**, *7*, 261. [CrossRef]
41. Geiger, A.; Roser, M.; Urtasun, R. Efficient large-scale stereo matching. In Proceedings of the 10th Asian Conference on Computer Vision (ACCV), Queenstown, New Zealand, 8–12 November 2010; pp. 25–38. [CrossRef]