Target Locating in Unknown Environments Using Distributed Autonomous Coordination of Aerial Vehicles

Hannah D. Mohr

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Electrical Engineering

April 10, 2019
Blacksburg, Virginia

Keywords: target locating, obstacle avoidance, multi-agent, swarming, UAV

Target Locating in Unknown Environments Using Distributed Autonomous Coordination of Aerial Vehicles

Hannah D. Mohr

ABSTRACT

The use of autonomous aerial vehicles (UAVs) to explore unknown environments is a growing field of research; of particular interest is locating a target that emits a signal within an unknown environment. Several physical processes produce scalar signals that attenuate with distance from their source, such as chemical, biological, electromagnetic, thermal, and radar signals. The natural decay of the signal with increasing distance enables a gradient ascent method to be used to navigate toward the target. The UAVs navigate around obstacles whose positions are initially unknown; a hybrid controller comprised of overlapping control modes enables robust obstacle avoidance in the presence of exogenous inputs by precluding topological obstructions. Limitations of a distributed gradient augmentation approach to obstacle avoidance are discussed, and an alternative algorithm is presented which retains the robustness of the hybrid control while leveraging local obstacle position information to improve non-collision reliability.

A heterogeneous swarm of multirotors demonstrates the target locating problem, sharing information over a multicast wireless private network in a fully distributed manner to form an estimate of the signal's gradient, informing the direction of travel toward the target. The UAVs navigate around obstacles, showcasing both algorithms developed for obstacle avoidance. Each UAV performs its own target seeking and obstacle avoidance calculations in a distributed architecture, receiving position data from an OptiTrack motion capture system, illustrating the applicability of the control law to real world challenges (e.g., unsynchronized clocks among different UAVs, limited computational power, and communication latency). Experimental and theoretical results are compared.

Target Locating in Unknown Environments Using Distributed Autonomous Coordination of Aerial Vehicles

Hannah D. Mohr

GENERAL AUDIENCE ABSTRACT

In this project, a new method for locating a target using a swarm of unmanned drones in an unknown environment is developed and demonstrated. The drones measure a signal such as a beacon that is being emitted by the target of interest, sharing their measurement information with the other drones in the swarm. The magnitude of the signal increases as the drones move toward the target, allowing the drones to estimate the direction to the target by comparing their measurements with the measurements collected by other drones. While seeking the target in this manner, the drones detect obstacles that they need to avoid. An issue that arises in obstacle avoidance is that drones can get stuck in front of an obstacle if they are unable to decide which direction to travel; in this work, the decision process is managed by combining two control modes that correspond to the two direction options available, using a robust switching algorithm to select which mode to use for each obstacle. This work extends the approach used in literature to include multiple obstacles and allow obstacles to be detected dynamically, enabling the drones to navigate through an unknown environment as they locate the target. The algorithms are demonstrated on unmanned drones in the VT SpaceDrones test facility, illustrating the capabilities and effectiveness of the methods presented in a series of scenarios.

Dedication

To my mother, for her unwavering love, support, and encouragement, and for teaching me the power of hard work.

Acknowledgements

I would like to thank my advisor, Dr. Jonathan Black, for his support and council on this project.

Special thanks to Gustavo Gargioni for his dedication and guidance, for the long hours spent debugging hardware and software, and for encouraging me through the difficult times.

I would like to thank Los Alamos National Laboratory and the National Physical Science Consortium with the Future Female Leaders in Engineering for affording me the opportunity to study at Virginia Tech. I would also like to thank the Bradley Department of Electrical and Computer Engineering for supporting my research through the Webber Fellowship.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Description and Motivation

Autonomous exploration of unknown environments is an active field of research; in particular, it is often of interest to locate a target in a cluttered environment, in which the positions of the target and obstacles are unknown. Target locating applications are numerous, including targets that emit chemical, biological, electromagnetic, acoustic, thermal, and radar signals that attenuate with increasing distance from their source. For example, target locating in rugged terrain has applications to search and rescue efforts. Consider the scenario of a plane crash in the wilderness, in which the lost vehicle is equipped with a beacon to assist rescue crews in finding its exact location. Radar history can be used to determine an estimate of the location, and a team of autonomous vehicles can then be dispatched to navigate potentially dangerous terrain, avoiding trees in a dense forest, while tracking the beacon to locate the crash site without requiring a manned team to perform the potentially hazardous exploration.

Similarly, a target locating problem could be to send emergency supplies to a stranded skier via aerial vehicles when a ground path has not yet been cleared; it is typical for skiers to carry beacons when in dangerous or remote areas, enabling the beacon signal to be measured in the target locating scheme. Obstacles such as trees and other terrain features need to be avoided to ensure the agents arrive with the needed supplies.

Another type of mission involving beacons is navigating to a meet point for supplies exchange, where a beacon has been placed at the landing site by a third party and the agents need to locate the drop site to complete the exchange. In an urban environment, road signs, mailboxes, and parked vehicles are examples of obstacles that need to be avoided, while an indoor space might feature obstacles such as furniture.

An alternative application is scientific monitoring, which presents another case in which autonomous target locating can be advantageous. An artifact of interest might emit a heat signature which can be detected as the signal, enabling the swarm to navigate to the position and take measurements. Monitoring can be a cumbersome task when measurements in remote locations need to be accessed frequently; this is a prime opportunity to use an autonomous swarm to free up human resources for more complex tasks.

Hazard response is another application in which autonomous agents are useful, since they reduce risk to the hazard response team. Consider a scenario where a chemical spill needs to be stopped at its source, requiring a piece of specialized equipment to navigate to the damaged element. A swarm of UAVs can measure the chemical plume to guide the specialized ground vehicle to the spill source, navigating around obstacles such as other equipment and furniture, enabling the hazard to be contained.

In this study, targets are assumed to be point sources that set up a stationary, time-invariant scalar field in the two-dimensional space radially surrounding the target. The scalar field is unaffected by obstacles and agents, and agents are able to collect *in situ* measurements of the signal. These assumptions preclude a few classes of targets, including those that emit propagating or modulated waves, such as a sound field in which the successive maxima and minima can be reflected or refracted by obstacles; those with time-varying amplitudes, such as windblown smoke; magnetic targets whose B-intensity could be modified by obstacles that have non-unity value of relative permeability; and targets that move within the region of interest, such as a during search-and-rescue of a person (infrared emitter) wandering in a forest.

Agents are considered to be single-integrator point objects and are assumed to be equipped with reliable sensors that can unambiguously detect obstacles; the method of detection is not addressed in this work. The obstacles are treated as point objects, which in practice can be irregularly shaped objects roughly approximating a point; within the obstacle avoidance algorithm, the obstacle shape is standardized to account for irregularities. Unallowable obstacles include objects that do not approximate a point or are large compared to the overall environment, including fences, rectangular buildings, etc. Another limitation on the type of obstacle considered stems from the assumption that obstacles do not distort the signal emitted by the target; for example, mirrored obstacles in the presence of a light-emitting target are excluded from this work.

A few considerations arise when the algorithm is applied to an actual problem where the physical properties of the hardware interacting with a realistic environment limit the solution space. In particular, sensing capabilities impose constraints on the agents' velocities when reduced resolution increases the computation time to construct a valid view of the environment, requiring slower movements in order to avoid collisions. In this work, the potential for sensor limitations was addressed by implementing a maximum velocity magnitude, reducing the impacts of unmodeled agent dynamics by keeping the agent within its linear operating range while allowing sufficient time to detect obstacles and measure the target's scalar field. To further address this issue, the barrier value function that will be discussed in Section 2.3.3 should be selected with the sensor sensitivity in mind to address the case when poor sensitivity would require slower movements on the part of the agents to avoid collisions. This consideration should be applied to both the sensor to detect the target's signal to ensure the agents have sufficient temporal response to track the gradient, as well as to the obstacle detection method to ensure the agents maintain an accurate view of the environment to enable obstacle avoidance. These topics are not specifically addressed in this study because the sensors used to measure the target’s signal and detect obstacles are not considered. During a realistic application, however, it would be important to consider these issues within the context of the applicable hardware.

The time required to locate the target in a given environment depends on several factors, including the amount of information available, the computation required to select the next movement, and limitations on the hardware itself. The information collected is related to the number of agents in the swarm, as well as the communication within the swarm: as more agents are available to measure the signal and more agents communicate their information to their neighbors, a more detailed picture of the environment can be constructed. This improved knowledge of the environment enables the agents to make more informed decisions about how to properly navigate to the target, allowing faster convergence to the goal; however, this improvement comes at the cost of computational complexity, which can limit the travel speed of the agents as they combine the information provided by their neighbors to calculate their next move. The trade-off between increased information and resulting computational complexity introduces an optimization problem to minimize the time required to locate the target. This optimization is outside the scope of this study; the solution to this problem would be mission-specific, depending on the complexity of the environment and signal mapping to determine the required level of information, while relying on the available computational power to determine the cost of additional agents.

The purpose of this project is to address the problem of locating a target in an unknown environment, using a swarm of UAVs to autonomously navigate to the target’s position while avoiding obstacles as they are detected. Different strategies have been developed for target locating under these conditions and have been considered in literature. The advantages and limitations of different approaches are discussed within the context of the algorithm development found in Chapter 2. This work addresses the problem in two components: target locating through gradient estimation, and real-time obstacle avoidance. Target locating is addressed in Section 2.2, in which different approaches to the problem found in literature are discussed. Gradient estimation has been considered from a single-agent perspective in [1]–[4], in which the agent estimates the gradient along its trajectory over time to control angular velocity. These methods were limited with regard to properly identifying when the target had been located. Methods that improved signal characterization through the use of sinusoidal or stochastic perturbations to the trajectory, such as in [5], improved the spatial diversity of the gradient but introduced increased actuator wear.

Multi-agent approaches that more closely align with the application presented in this work are discussed in [6]–[10]. These algorithms are designed for different swarm architectures, including leader-follower where a single agent computes the gradient in a centralized location, and distributed in which the agents share information to take full advantage of the swarm capabilities. In this work a distributed least-squares approach is implemented, combining the position and measurement information of each agent and its neighbors to compute an estimate of the gradient as presented in [11] and extended in [12]. Two limitations associated with this method are discussed and overcome. First, the estimation requires sufficient spatial diversity to ensure that the inversion in the computation exists, placing restrictions on the swarm formation and introducing modes in which the collinearity of agents can render the gradient estimate invalid. This problem can occur even when there are enough agents in the swarm to estimate the gradient because the estimation computation relies on relative positions; the solution to this limitation, described in Section 2.2.1, is the use of measurement memory to improve the spatial diversity of the measurements each agent accesses, mitigating the effects of the swarm collinearity. The second concern relates to the sensitivities of the original algorithm discussed in [11] to small changes in position when the swarm is near the target, which can make it difficult for the swarm to know that it has located the target. The solution to this issue is addressed in Section 2.2.2 in which a new normalization strategy is proposed and implemented to overcome the sensitivities of the original algorithm.

Obstacle avoidance is presented in Section 2.3, in which the methods discussed in literature are extended to real-time multi-obstacle scenarios. One approach to obstacle avoidance is to bend contour lines around an obstacle to direct an agent's path to preclude collisions, as described in [13]. This method gives rise to robustness issues, as shown in [14], when certain initial conditions, disturbances, or arbitrarily small exogenous inputs cause the agent to fail to locate the target. A hybrid controller, presented in [15] for single-agent systems and [6] for multi-agent systems, overcomes the topological obstructions found in purely continuous or purely discrete controllers; however, the robust obstacle avoidance schemes do not consider real-time obstacle detection for cases when the environment is initially unknown, and a distributed architecture for swarming capabilities is not developed. These topics are addressed in this work, extending the current state of robust hybrid obstacle avoidance to be applicable to unknown environment exploration using swarming capabilities.

1.2 Preliminaries

1.2.1 Graph Theory

The swarming techniques used in this project, in particular throughout Chapter 2, are built off the graph theory approach to multi-agent systems, in which drones are represented by graph vertices and the communication and interaction among drones is represented by the graph edges. In this work, drones will be referred to as agents, consistent with the terminology used in graph theory. Details on graph theory can be found in [16], [17]; the key results used in this work are briefly described below. The swarms discussed in this project are allowed to be heterogenous, permitting the use of different types of drones within a single application (e.g., a hexarotor and a quadrotor are heterogenous agents).

In the communication graph $\mathcal{G}$, two vertices are defined to be neighbors if they share an edge, and the neighbors of the *i*th agent make up the set $\mathcal{N}_i$; this set includes the drones with which agent i can directly interact through information exchange. The degree matrix, $\Delta(\mathcal{G})$, is defined as the diagonal matrix of the number of neighbors for each agent. The adjacency matrix, $\mathcal{A}(\mathcal{G})$, is a symmetric matrix that indicates whether a vertex is a neighbor, such that $[\mathcal{A}(\mathcal{G})]_{i,j} = 1$ if vertices i and j are neighbors and is zero otherwise. From these definitions, the graph Laplacian is expressed

$$\mathcal{L}(\mathcal{G}) = \Delta(\mathcal{G}) - \mathcal{A}(\mathcal{G}) \tag{1}$$

which has been shown to be a symmetric positive semi-definite matrix for strongly connected graphs that includes a zero eigenvalue corresponding to the eigenvector of all ones (see [18]). The second smallest eigenvector of $\mathcal{L}(\mathcal{G})$ is denoted as λ_2, and it contains information about the effectiveness of swarm communication, often indicating the speed of convergence for distributed motion and estimation.

1.2.2 Hybrid Dynamical Systems

An interesting problem in unknown environment exploration arises when obstacle avoidance is introduced. As agents detect obstacles, they must decide how to avoid colliding with the obstacle while still maintaining a path toward the target. A purely discrete or purely continuous control law for the decision process can lead to undesirable topological obstructions that cause the agents to prematurely believe they have converged to the target. To prevent this issue and ensure the robustness of the obstacle avoidance strategy, a hybrid controller is considered as will be detailed in Section 2.3.1. The advantage of a hybrid approach is the ability of hybrid dynamics to provide a framework that models systems which include both continuous and discrete dynamics. A formal treatment of hybrid dynamical systems can be found in [14]. Further reading can be found in [19]–[23].

Continuous components of the hybrid system are characterized by flow dynamics, mathematically described by the differential inclusion

$$\dot{x} \in H(x) \quad x \in C \tag{2}$$

where H is the flow map that describes the continuous dynamics over the flow set C, and $x \in \mathbb{R}^d$ is the continuous state of dimension d. In this work, the flow dynamics represent the physical dynamics of the drones, with the state describing the position and velocity of the drone.

Discrete components are represented by the jump dynamics, given as the difference inclusion

$$x_k^+ \in G(x_k) \quad x_k \in D \tag{3}$$

where G is the jump map that describes how the discrete dynamics evolve over the jump set D. x_k is the discrete state which does not necessarily have the same dimension as the continuous state. The discrete dynamics are used to implement changing control modes that leverage prior information to select the appropriate mode in the jump set.

Assumptions are placed on the system to ensure it behaves in a manner consistent with reality and enable key results from [6], [14], [15]: it is assumed that the sets C and D are closed, the mappings H and G are outer semicontinuous, locally bounded and nonempty, and H is convex in C. In this application, the flow set C is the two-dimensional real space, which is a closed set under the operations of this project (e.g., addition, multiplication). The flow mapping H describes the motion of the agents, which are modeled as single-integrator point objects and therefore satisfy the assumptions on the continuous mapping. The jump dynamics comprised of D and G are developed within the control law mode switching regime and can therefore be designed to meet the above assumptions.

1.2.3 Signal Properties

In this work, a target emitting a time-invariant concave signal $J(r)$ is considered. Since the target is the source of the signal, the terms “target” and “source” will be used synonymously. The signal is assumed to take on scalar values of the environment space, where $r \in \mathbb{R}^d$ is the position coordinates. The signal takes on a unique bounded maximum value at the location of the target. The signal is assumed to be smooth, have compact level sets, and have a non-zero gradient at all positions excluding the target’s location. These assumptions ensure the signal satisfies the mild conditions to guarantee Caratheodory solutions for the gradient, bounding the gradient magnitude. This type of signal can take many forms; for clarity, within the context of this work the signal will take a quadratic form as an example, as shown in (4).

$$J(r) = -(r - r^*)^T P_r (r - r^*) \tag{4}$$

where r^* is the target’s position, r is the position space, and $P_r \in \mathbb{R}^{d \times d}$ is a positive-definite matrix of weighting coefficients. These properties will be considered in Section 2.2.

1.3 Problem Statement

Consider a target with unknown position that emits a measurable scalar signal $J(r) \in \mathbb{R}$

that decreases with increasing distance such that the signal is concave over the measurement space $r \in \mathbb{R}^d$. The problem addressed in this work is to develop a distributed controller that enables a heterogeneous swarm of N agents whose communication is characterized by the strongly connected graph $\mathcal{G}$ to autonomously navigate through an unknown environment to the location of the target, making use of local signal measurements to select the appropriate direction of travel. The agents must avoid obstacles that may exist along their path to the target, applying avoidance strategies in a manner robust to exogenous inputs as new obstacles are detected within the unknown environment.

2 Proposed Solution

2.1 Overview

In this study, the unknown environment is taken to be a space in $\mathbb{R}^2$ in which elements exist, including agents, targets, and obstacles. The space can be infinite or bounded depending on the application; in simulation the infinite case is considered, in which agents can move anywhere in the real plane, while in implementation the space is restricted to the size of the testing facility. Agents are point objects representing drones modeled with single integrator dynamics, and at any given time are aware of their own position, the measured magnitude of the signal $J(r)$ emitted by the target, and the position and signal measurement of neighboring agents (i.e. agents within communication range). Sensor noise is not considered in this work, but could be addressed in future work through the implementation of a Kalman filter or similar technique. The target is a stationary point object with a radially decreasing emission signal $J(r)$ that is used to guide agents toward the source; this signal passes freely through obstacles and agents. Obstacles are point objects that obstruct the path taken by the agents and must be avoided.

The proposed solution is to use a swarm of autonomous UAVs to perform a distributed target locating algorithm augmented with robust obstacle avoidance to navigate the unknown environment and find the position of the target. Target locating is performed through a distributed least-squares approach to gradient estimation, leveraging the spatially diverse signal measurements collected by agents in the swarm. Two obstacle avoidance algorithms are presented. The first considers a fully distributed approach that relies on augmenting the measured signal with a barrier term as a function of distance to the obstacle to direct the agent around an obstacle, as presented in [24]. The second algorithm modifies the original algorithm to improve robustness to varying formation size, motivated by limitations of the first algorithm. Different types of obstacles have varying shapes, so for consistency a virtual box is placed around each obstacle to enable obstacle avoidance to be applied uniformly without having to identify what type of obstacle has been detected. This virtual box is used in generating the barrier that the agents avoid in order to preclude collisions with the obstacle. Algorithm details and simulation results are presented in the following sections.

2.2 Gradient Estimation Through Least-Squares

The main problem under investigation is how to use the local scalar measurements made by each agent to navigate the swarm to the target. The natural decay of the signal with increasing distance from the target enables the agents to follow an ascending gradient, ultimately ending up at the location of the target. The exact expression for how the signal varies in space is unknown, so the gradient cannot be directly computed; however, the swarm does have spatially diverse measurements available since the agents are distributed in their formation, each measuring the local value of the target's signal, and a single drone can save measurements at different points along its trajectory to increase the number of measurements available. The gradient can then be estimated using local measurements to develop a consensus for the direction in which the swarm should travel to navigate toward the target.

The problem of gradient estimation has been studied in literature, resulting in several different approaches to continuous-time algorithms. Methods involving a single drone were developed in [1]–[4], which focused on control of angular velocity in cases where position measurements were limited. These methods maintained a fixed linear velocity and did not propose solutions for bringing the agent to a stop when the target had been located, instead allowing the agents to orbit the target or even overshoot the desired final location. Other methods, e.g. [5], employ perturbations in the form of sinusoidal or stochastic inputs to search the space and provide sufficient spatial diversity to estimate the gradient, enabling signal characterization at the cost of additional wear on actuators.

More similarly to the approach investigated in this work, other methods of gradient estimation employed a swarm of communicating agents that share their local signal measurements. There are two main swarm architectures employed, including leader-follower (see [6], [7]) where a single agent computes the gradient and the remaining agents follow the estimate provided by the central node, and distributed (see [8]–[10]) where the agents use a consensus-based approach to combine information and compute the gradient estimate as a collective.

A least-squares approach was used to compute the gradient estimate for this project to enable fast computation from limited information. The method used was adapted from [11] and was described in [24], in which a system of N agents interacting according to a strongly connected communication graph $\mathcal{G}$ is considered. Let r_i and $J(r_i)$ denote the position and signal measurement of the ith agent, respectively. The gradient is estimated using a Taylor expansion approximation, shown below:

$$J(r_j) = J(r_i) + (r_j - r_i)^T \nabla J(r_i) + \frac{1}{2}(r_j - r_i)^T \nabla^2 J(r_i)(r_j - r_i) + H.O.T. \tag{5}$$

where *H.O.T.* are the higher order terms. This expression yields an implicit form for the true gradient $\nabla J(r_i)$, given as

$$b_i = R_i \, \nabla J(r_i) \tag{6}$$

where

$$R_i = \begin{bmatrix} (r_1 - r_i)^T \\ \vdots \\ (r_{|\mathcal{N}_i|} - r_i)^T \end{bmatrix} \qquad b_i = \begin{bmatrix} J(r_1) - J(r_i) \\ \vdots \\ J(r_{|\mathcal{N}_i|}) - J(r_i) \end{bmatrix} \tag{7}$$

The estimated gradient $\hat{g}_i$ is solved as a least squares problem to minimize $\|R_i\hat{g}_i - b_i\|^2$ using information from the agent's $|\mathcal{N}_i|$ neighbors. The resulting expression is:

$$\hat{g}_i = (R_i^T R_i)^{-1} R_i^T b_i \tag{8}$$

Note that the gradient estimation computation requires the matrix $R_i^T R_i$ to be full rank, placing constraints on the minimum spatial diversity of the measurements. To guarantee invertibility, each agent must have access to at least $d + 1$ measurements featuring independent positions, where d is the dimension of the signal.

Agents navigate toward the target by moving along the estimated gradient, following their best estimate of the direction to the target. Assuming agents behave as single-integrator point objects, the distributed control law for each of the *i* agents can be written as

$$u_i = \beta \sum_{j\in\mathcal{N}_i} [(r_{f,i}(t) - r_i(t)) - (r_{f,j}(t) - r_j(t))] + \alpha\hat{g}_i(r_i) \tag{9}$$

where $\alpha > 0$ and $\beta > 0$ are scalar gains. It was shown in [11] that this controller guarantees the centroid of the swarm formation will converge to a region with a bounded radius centered at the target's position.

2.2.1 Agent Memory for Improved Estimation

There are a few key points to note about the least-squares approach for gradient estimation. In particular, this method places requirements on the swarm size since it requires at least one more independent measurement than the dimensionality of the signal being estimated. For the distributed gradient estimation, this means each agent must have at least d neighbors, where d is the dimension of the signal, as discussed in [11]. In cases where the number of agents available is limited, the architecture becomes susceptible to gradient estimation loss in the presence of communication failures or agent position collinearity. This issue was observed in early implementations of target seeking, in which the three drones were unable to precisely locate the target in two dimensions because they became collinear. Consider a case in which three agents are used to estimate the gradient to move to the target. The discontinuities in position and gradient estimation are shown in Figure 1 to align with the determinant of the gradient estimation matrix dropping to zero; the agents are able to recover from this condition because the formation control is not affected by the loss of rank, but the target seeking portion of the algorithm was momentarily unavailable.

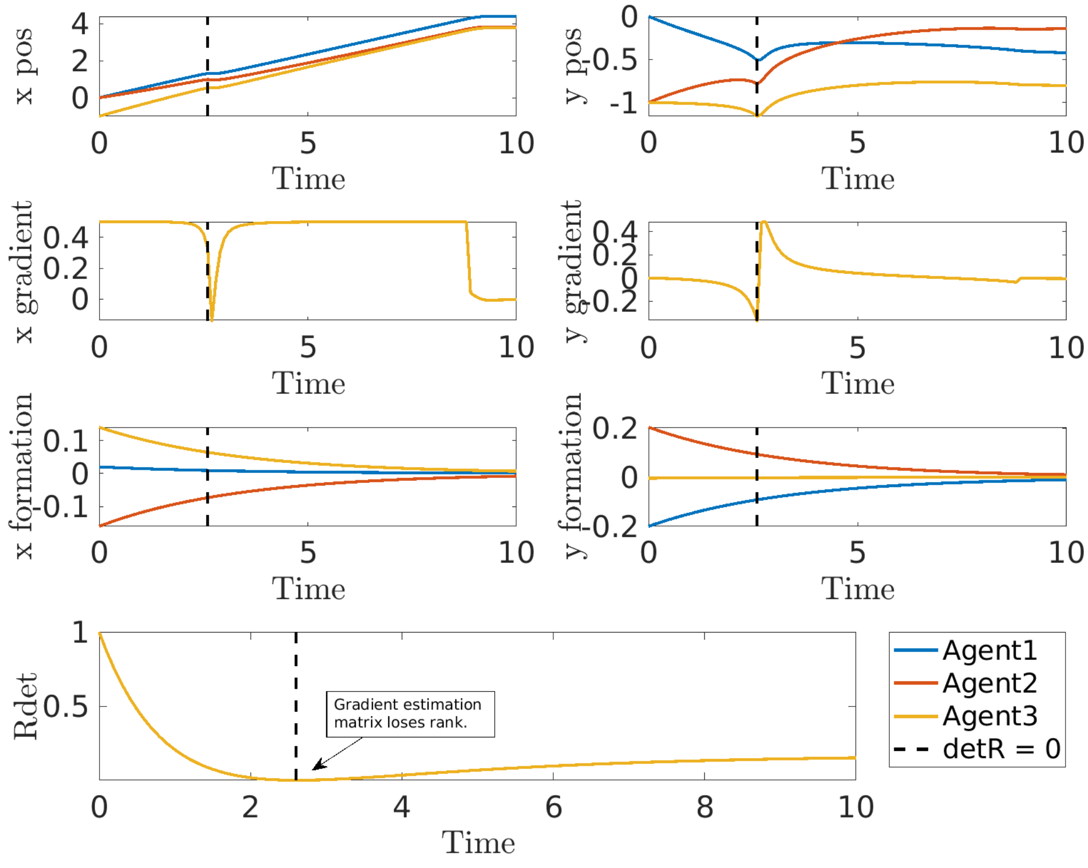


Figure 1: Gradient estimation challenges with uninformative swarm formation. The discontinuity in the trajectory occurs when the estimation matrix loses rank ($\det(R) = 0$). Observe that the gradient estimate drops to zero and rapidly changes when the estimation matrix loses rank. The formation is not affected, enabling the agents to recover from this situation.

To help counteract this potential for gradient estimation loss as well as increase the resolution of the estimate, the agents can save their previous measurements and include them in the least-squares estimation. Increasing the number of saved values improves the resolution of the estimate and increases the probability that there will be a sufficient number of independent measurement points, with the trade-off that additional points require additional memory usage since both the position and signal measurement must be recorded for each point of interest, and that the matrix inversion used to compute the gradient will have higher dimensions and therefore increase in computational complexity. The smoothing effects of the data saving scheme are illustrated in Figure 2, showing how the trajectory is affected for increasing number of locally saved measurements.

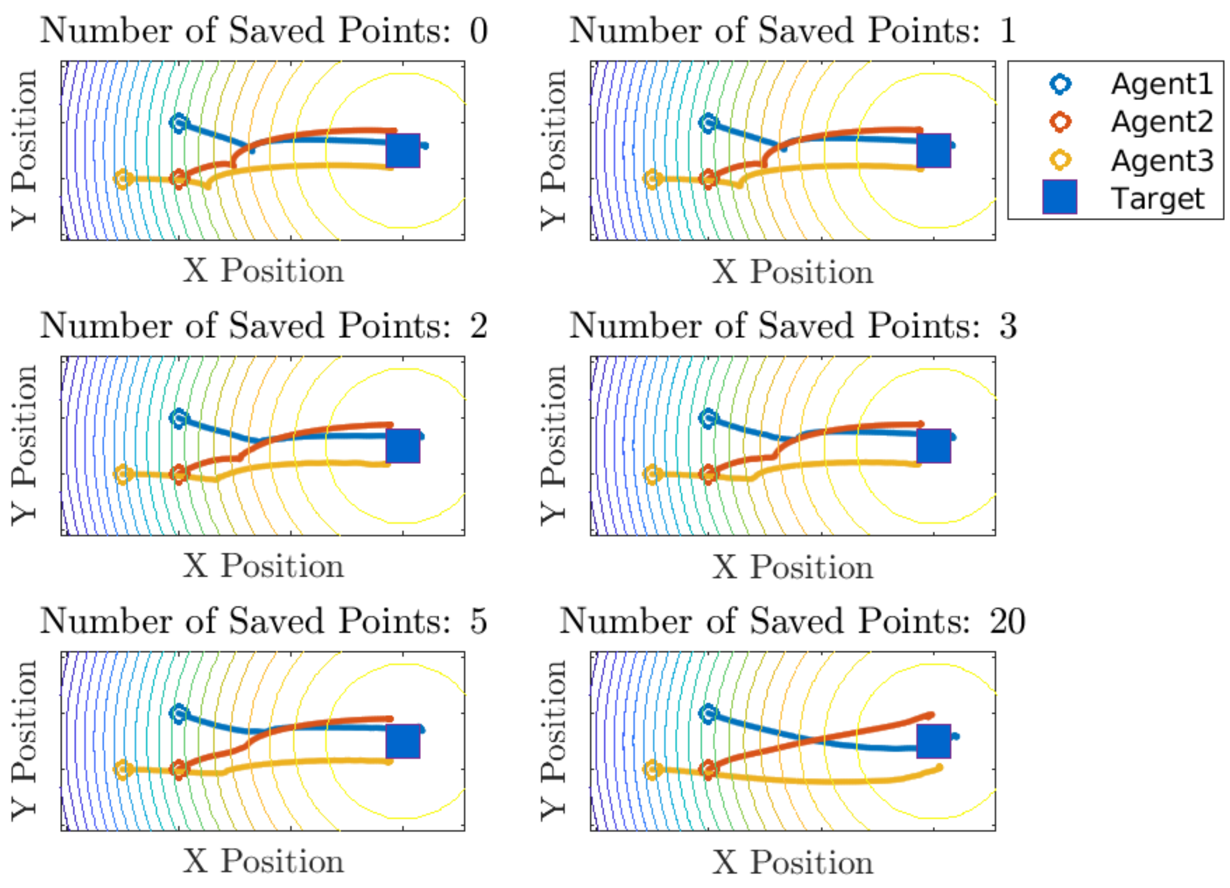


Figure 2: Effect of saving measurements on gradient estimation performance. When the transient response of the formation controller causes the agents to form a line, they no longer have sufficient information to compute the gradient. This is seen as the discontinuity in the trajectory as the agents lose and then regain sufficient spatial diversity to compute the gradient estimate. The six subplots show a comparison of trajectories for the same initial condition and formation with varying number of previous measurements saved by each agent. The discontinuity of the trajectory smooths out as the number of saved points increases the available information, reducing the impact of the collinearity during the formation transient.

A limitation of using previously stored values is that it depends on the trajectory not following a linear path, otherwise the saved points will provide redundant information. Since the path taken by the agents is not designed to ensure the trajectory is sufficiently informative, it remains important to keep the size of the swarm sufficiently large to estimate the signal.

2.2.2 Normalization of the Gradient Estimate

In the method described thus far, the gradient is normalized in order to keep the velocity constant, avoiding cases where very steep gradients substantially increase the speed of the agent potentially beyond the hardware limitations. However, normalization of the gradient gives rise to an issue when the magnitude of the gradient is very small, in particular near the target where the gradient flattens out as the signal achieves its maximum value. This normalization strategy causes the gradient estimate to be divided by a very small number, resulting in dramatic changes from small perturbations to the position. This latter method is discussed in some of the literature, but it does not consider the implementation difficulties that arise when the commanded velocity changes dramatically, causing the hardware to respond outside its linear region in a less predictable manner. To mitigate this problem, this study considers an alternative solution to the gradient normalization problem. In this work, a condition is applied to the normalization feature such that if the magnitude is small (< 0.1), then the gradient estimate is returned without normalization; this enables the changes due to position perturbations to have a minimal effect on the velocity of the agent. The motivation for this approach can be illustrated through a simple example.

Consider the three-agent case described in the previous section where the formation transient causes the swarm to become spatially collinear. Figure 3 shows the estimate of the gradient in the x-direction over time and the agents’ trajectories for the three normalization cases: normalization never applied, normalization always applied, and normalization applied when the gradient magnitude is greater than 0.1. Without normalization, the magnitude of the gradient and therefore the commanded velocity is large and causes the agents to overshoot the target, reaching upwards of 15 m/s, which is unrealistic for the UAVs considered for this project. The next case depicted shows this issue corrected by normalizing the gradient; however, the agents continue to move in a chaotic manner after the target is located. The final panel in the figure shows the normalized gradient being applied until the norm is sufficiently small, at which point the velocity is allowed to taper off to zero.

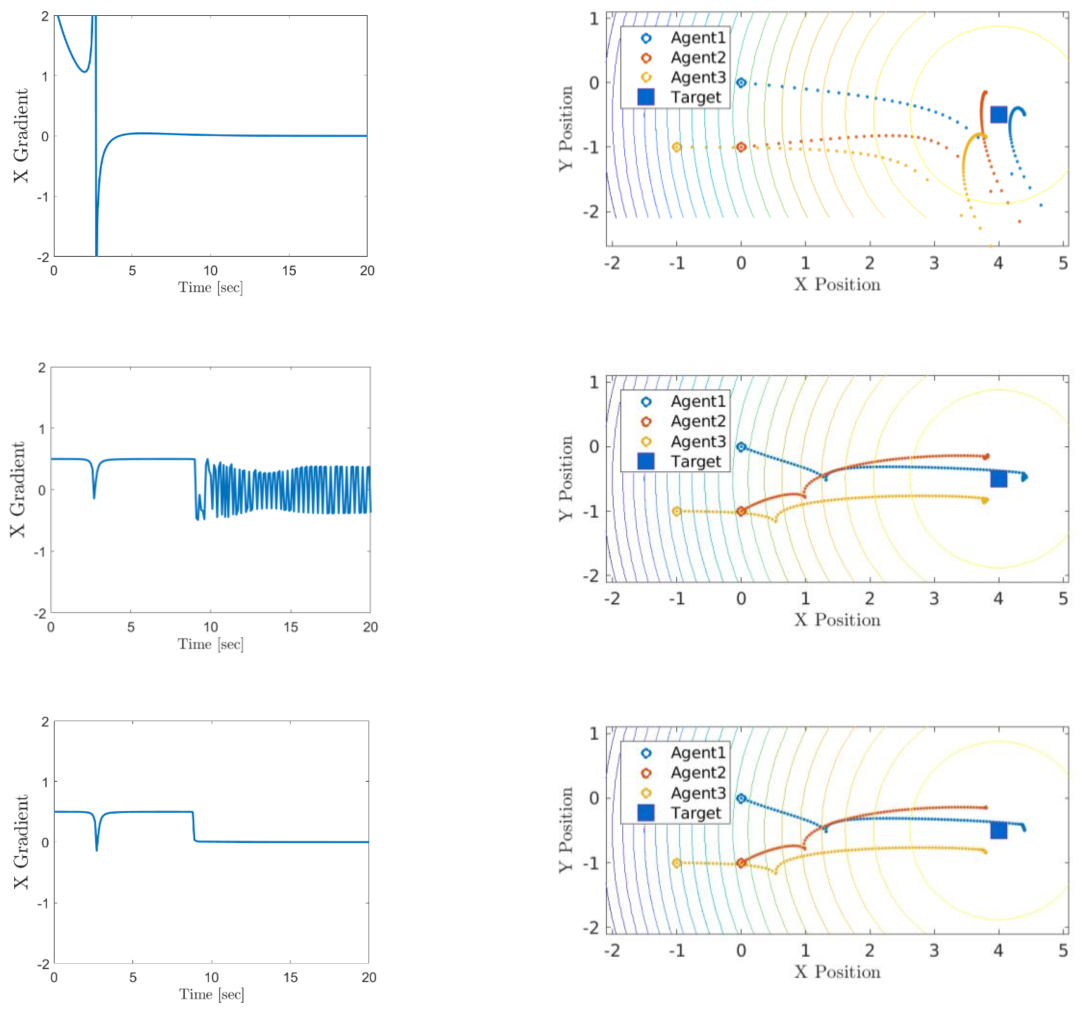


Figure 3: Comparison of gradient estimate normalization strategies. Three methods of normalization are shown, including normalization never applied (*top*), normalization always applied (*middle*), and normalization applied selectively (*bottom*). Without normalization, the velocity becames very large, resulting in overshoot. While the normalized gradient did not overshoot the target, it resulted in toggling velocity near the target because the signal gradient was nearly flat, so the agents continued to move after the target was located. When normalization was selectively applied, the trajectory prior to reaching the target was consistent with the normalized case, and the agents were able to come to a full stop when the target was located.

2.3 Obstacle Avoidance Algorithm 1

Gradient estimation only covers a portion of the target locating problem, since it does not directly address the problem of obstacles distributed in the environment. A typical approach to obstacle avoidance is to augment the measured signal to bend the contour lines around the obstacles, causing the gradient estimation to direct the agent around the obstacle on its path to the target. This is accomplished through a barrier value that adjusts the signal measurement to be much smaller when the agent is within the repulsion region near the obstacle, altering the local perception of the gradient field. In this implementation, a virtual box of height h is placed around the obstacle, as shown in Figure 4, enabling obstacles of varying shapes to be treated in a consistent manner. The equation to accomplish this takes the form given in (10).

$$J_q = J + B(z) \tag{10}$$

where J_{q} is the augmented signal used in the gradient estimation algorithm to determine the direction of travel, J is the measured signal, and $B(z)$ is the barrier function that takes in the square of the Euclidean distance to the boundary (given as the red box in Figure 4). A typical function for $B(z)$ is

$$B(z) = -(\rho - z)^2 \log(1/z) \tag{11}$$

Alternative functions can be used to meet the requirements of the barrier value if they tend to negative infinity as the distance to the barrier goes to zero to ensure obstacle avoidance dominates over target locating, and tend to zero as the agent moves outside the repulsion region to enable the augmented signal to maintain continuity. Each agent shares its augmented signal with its neighbors (i.e. the agents with which it can interact according to the communication graph), indirectly distributing information about the obstacle throughout the swarm.

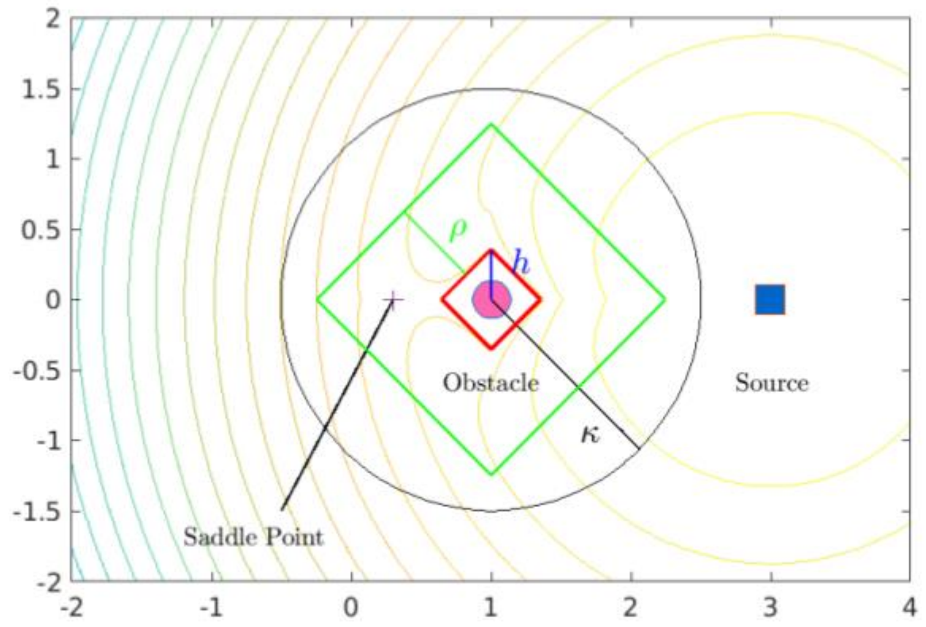


Figure 4: Naive approach to obstacle avoidance. The avoidance algorithm is applied to an obstacle that is measured within the sensor detection radius κ (black). Near the obstacle barrier (red box of height h), there is a repulsion region of depth ρ (green) in which the agent applies obstacle avoidance. The barrier value that augments the measured signal causes the contour lines of the perceived signal to bend around the obstacle, directing the agent around the obstacle as it follows the gradient toward the target. This approach yields a saddle point where the agent decides which direction to take.

The disadvantage of the above approach to obstacle avoidance is the appearance of an unwanted topological obstruction in the form of a saddle point created by the combination of the measured concave signal and the calculated barrier value, representing the decision point where the agent can move either to the left or the right of the obstacle. This topological obstruction has been studied in literature (see [14]), and has been shown to allow certain initial conditions or small exogenous inputs to preclude robust convergence to the source; essentially, arbitrarily small inputs in the form of noise, disturbances, or adversarial inputs can cause the swarm to perceive the saddle point as the source of the signal, resulting in the agents misidentifying the location of the target. To overcome this issue, a hybrid controller is implemented as discussed in [6], [14], [15], in which multiple control modes are used to enable the agent to robustly select a direction in which to travel around the obstacle.

The hybrid control approach implements two active obstacle avoidance modes, illustrated in Figure 5, which direct the agent around the obstacle in a particular direction. This work considers an additional mode for the case when no obstacle has been detected and therefore obstacle avoidance is unnecessary, thereby extending the hybrid approach found in literature to environments where obstacles are detected dynamically and the number and positions of obstacles in the environment is initially unknown. The modes represent sets of potential agent positions whose union includes the allowable space and excludes the keep-out area within the virtual box placed around the obstacle; this enables the agents to move to any position that will not result in a collision with the obstacle.

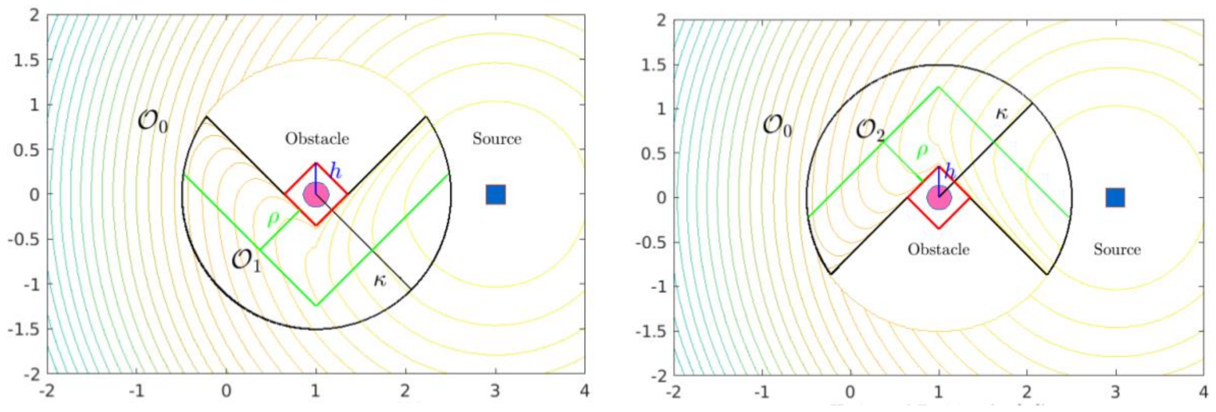


Figure 5: Partitioned space for the discrete modes for obstacle avoidance. The regions O_1 and O_2 are constructed as described in [6], with the adaptation of radial boundedness through the inclusion of region O_0. The obstacle barrier is formed by extending the virtual box (red) edges to intersect the edge of the detection region (black). Mode 1 (*left*) directs the agent to move toward the right of the obstacle, while mode 2 (*right*) directs the agent to the left of the obstacle. The two modes overlap so that neither includes a saddle point.

The hybrid controller works as follows: each mode considers the barrier as the extension of the edge of the box, pushing the agent either to the right of the obstacle (mode 1) or to the left of the obstacle (mode 2). Since each mode has a predetermined direction, the agent does not have a decision point and therefore does not generate a saddle point, allowing for robust convergence to the target. When an agent detects an obstacle, meaning that it is within the detection radius κ, it selects an initial boundary avoidance mode, defaulting to mode 1 unless it is outside the valid region for mode 1, in which case it will instead select mode 2. The jump dynamics define a region in which an agent can transition between modes, enabling the agent to adjust to new information and select the direction of travel that enables convergence to the target location. To prevent rapid toggling between modes, the jump region is designed so the transition regions for the two modes do not overlap. When the agent moves beyond the sensor range, the mode is set to 0 since obstacle avoidance is no longer required.

The method described above is derived from literature, in particular [6], with the key extension of radially bounding the obstacle avoidance modes 1 and 2 through the inclusion of mode 0 to allow for cases when the obstacle is outside the detectable range. This adaptation enables the hybrid control method to be applied to unknown environments since an obstacle only needs to be observed when the agent is in danger of collision, which can be achieved with an onboard sensor.

The jump dynamics that describe the mode transitions differ from those presented in [24] and earlier works, though the same principals are followed. There are five cases to consider when defining the jump dynamics that characterize the transition from the current mode $q \in \{0,1,2\}$ to the next mode q^+. When a new obstacle is detected, which occurs when the Euclidean separation between the agent and the obstacle $\sqrt{z_i}$ is less than the sensor measurement radius κ, and initial mode is selected. This defines two possibilities: if the agent is within the valid region for mode 1, meaning its position is included in the set O_1, then mode 1 is selected; outside that set mode 1 is invalid, so mode 2 is selected. If instead the obstacle has previously been detected, the agent has the option to transition to a different mode as it approaches the boundary of its current mode; this is where the method used in this work differs from previous work conducted in [6], [14], [15], [24], in that the shape of the transition region is defined such that it spans across the barrier into the next mode. The advantage of this approach is that while an agent would theoretically never cross over the boundary without transition modes, practical implementation experiences actuator delays or unexpected disturbances that can cause overshoot. Enabling transitions on both sides of the mode barrier allows the controller to be more robust to physical limitations and recover from otherwise invalid states. A graphical representation of the jump region is shown in Figure 6. Finally, if the agent leaves the range in which it can detect the obstacle, the mode is set back to 0 since obstacle avoidance is no longer necessary. If none of the above conditions apply, the agent does not update its mode and retains the previous value. The mode update law is mathematically described below.

$$q_i^+ = \begin{cases} 1 & \text{if } \sqrt{z_i} \leq \kappa\, r_i \in O_1 \\ 2 & \text{if } \sqrt{z_i} \leq \kappa\, r_i \notin O_1 \\ 3 - q_i & \text{if } q_i \neq 0, |J_{q_i}| > \mu\, |J_{3-q_i}|,\ r_i \in O_{3-\mathrm{q}} \\ 0 & \text{if } q_i \neq 0, \sqrt{z_i} > \kappa \\ q_i & \text{otherwise} \end{cases} \tag{12}$$

where $\mu > 0$ is a design parameter that determines the width of the repulsion region.

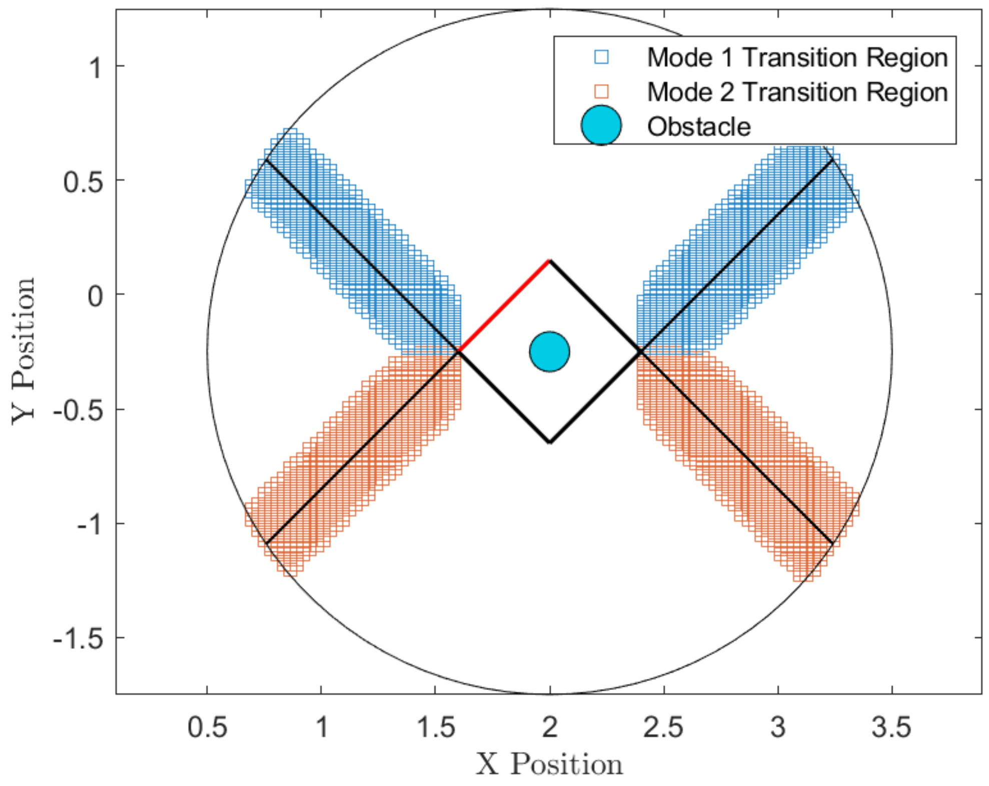


Figure 6: Illustration of mode transition region. The jump region in which an agent can transition from mode 1 to mode 2 is shown in blue, while the transition region from mode 2 to mode 1 is shown in red. The barrier for mode 2 includes a brighter red line segment for orientation reference.

To accommodate multiple obstacles, each agent keeps an array of M modes to account for the M obstacles in its field of view. Detection of new obstacles adds additional modes to the array, while setting the mode back to 0 for an obstacle that is no longer detected causes its mode to be deleted to reduce memory requirements.

An advantage of this control scheme is that not all agents need to be within sensor range of an obstacle to respond to the collision threat since information is passed implicitly in the augmented signal measurements transmitted by neighboring agents who have detected the obstacle. The potentially conflicting objectives of obstacle avoidance, formation flying, and target locating are resolved through proper weighting of the control inputs, as discussed in the following section.

2.3.1 Distributed Hybrid Controller

The final control law is constructed by combining the formation controller and augmented gradient estimator, as shown in equation (13).

$$u = -\beta\mathcal{L}[r - r_f] - \alpha\mathcal{I}\hat{g}(J_q) \tag{13}$$

where α and β are tunable control gains to select the relative impact of formation control and gradient following, respectively; $\mathcal{L}$ is the Laplacian, describing the standard formation controller found in [16] that multiplies the positions of each agent and its neighbors in the vector r along with the desired relative positions r_f; J_q is the augmented measured signal used to estimate the gradient $\hat{g}$; and $\mathcal{I}$ is an indicator function that applies the gradient-tracking controller when the agents have enough information for the estimate to be valid. In implementation, this is approached by setting the gradient estimate to zero if there is not enough information to perform the least-squares calculation, whether due to too few neighbors or collinearity, as described in Section 2.2.1. The hybrid component of the controller is found within the gradient estimate in which the obstacle avoidance modes are used to augment the signal measurement. For a discussion on agent interactions, see [25].

The target locating portion of the continuous control dynamics was shown in [11] to guarantee the centroid of the agents' formation will converge to a region around the target's position defined by a radius ϵ, given as follows:

$$\epsilon = \frac{2\alpha\sqrt{N}\bar{g}}{\beta\lambda_2} \tag{14}$$

where α and β are the control gains from (9), N is the number of agents in the formation, $\lambda_2$is the second smallest eigenvalue of the communication graph Laplacian, and $\bar{g}$ is the maximum value the gradient takes.

2.3.2 Formation Selection

The swarm formation was selected to aid in distributed gradient estimation. For large swarms, such as those including six agents used in the simulation results, it was determined that a circular formation would be advantageous since it ensures the agents will not collapse into a line and lose rank. This formation is designed for cases when the signal emitted by the target has two dimensions, but it can easily be extended to the three-dimensional case. The resulting formation for the *i*th agent takes the form

$$F_i = [A_0 \cos\left(\frac{2i\pi}{N}\right), A_0 \sin\left(\frac{2i\pi}{N}\right)] \tag{15}$$

where A_0 is a tunable formation size gain and N is the number of agents in the formation. This spaces the agents equally on the circumference of a circle of radius A_0, a geometry that ensures any three agents will remain noncollinear, satisfying one of the assumptions for distributed gradient estimation.

It was observed that swarms with few agents were more effective at locating the target if their formation was nonsymmetric. For example, the formation selected for a four-drone swarm during implementation was:

$$F = \begin{bmatrix} A_0 & \frac{2}{3}A_0 \\ \frac{1}{2}A_0 & 1.1A_0 \\ 0 & 0 \\ -\frac{1}{2}A_0 & \frac{1}{2}A_0 \end{bmatrix} \tag{16}$$

Alternative formations would also be effective if designed to ensure the gradient estimation has sufficient information about the signal under investigation.

The formation control law presented above is for a rigid formation that will keep the agents spaced at their defined relative positions as they navigate toward the target and avoid obstacles.

However, the swarm can have increased capabilities for navigating cluttered environments if the swarm can be adjusted to enable agents to break formation to move between closely spaced obstacles. To this end, a condition is placed on the formation control to enable a switching controller that deactivates the rigid formation when obstacle avoidance is necessary; it operates by setting the gain to zero if the barrier value is nonzero, thereby permitting an agent to leave the formation if it is avoiding an obstacle. The remaining agents keep the formation to maintain the gradient estimation. The switching formation control also prevents the formation from overpowering obstacle avoidance and pushing an agent into the obstacle. If all agents are actively avoiding obstacles, completely disabling formation control for the swarm, there is the potential for the gradient estimate to lose rank due to collinearity of measurements; this is acceptable because the agents will resume formation control once the obstacles no longer pose a threat. The result is a more robust algorithm for obstacle avoidance.

2.3.3 Barrier Value Selection

It was observed that the function (11) to compute the barrier value implemented in literature was not effective for the distributed version of the control algorithm since it did not provide sufficiently strong repulsion from the obstacle. The equations considered are included in Table 1 below. For each equation, ρ is the depth of the repulsion region and z is the square of the Euclidean distance to the obstacle barrier.

Table 1: Potential Barrier Value Functions

Label	**Equation**
Scaled Log	$-(z-\rho)^2\log\left(\frac{1}{z}\right)$
Weakly Scaled Log	$-\lvert z-\rho\rvert\log\left(\frac{1}{z}\right)$
Unscaled Log	$-\log\left(\frac{1}{z}\right)$
Scaled Inverse	$-\frac{(z-\rho)^2}{z}$
Weakly Scaled Inverse	$-\frac{\lvert z-\rho\rvert}{z}$
Unscaled Inverse	$-\frac{1}{z}$
Strongly Scaled Inverse	$-\frac{(\rho-z)^3}{z}$
Strongly Scaled Inverse Squared	$-\frac{(\rho-z)^4}{z^2}$

The primary difference in the functions listed above is how aggressively each tends toward infinity as the distance to the barrier decreass to zero. For direct comparison, the equations are plotted in Figure 7 for their effective range of $[0,\rho]$.

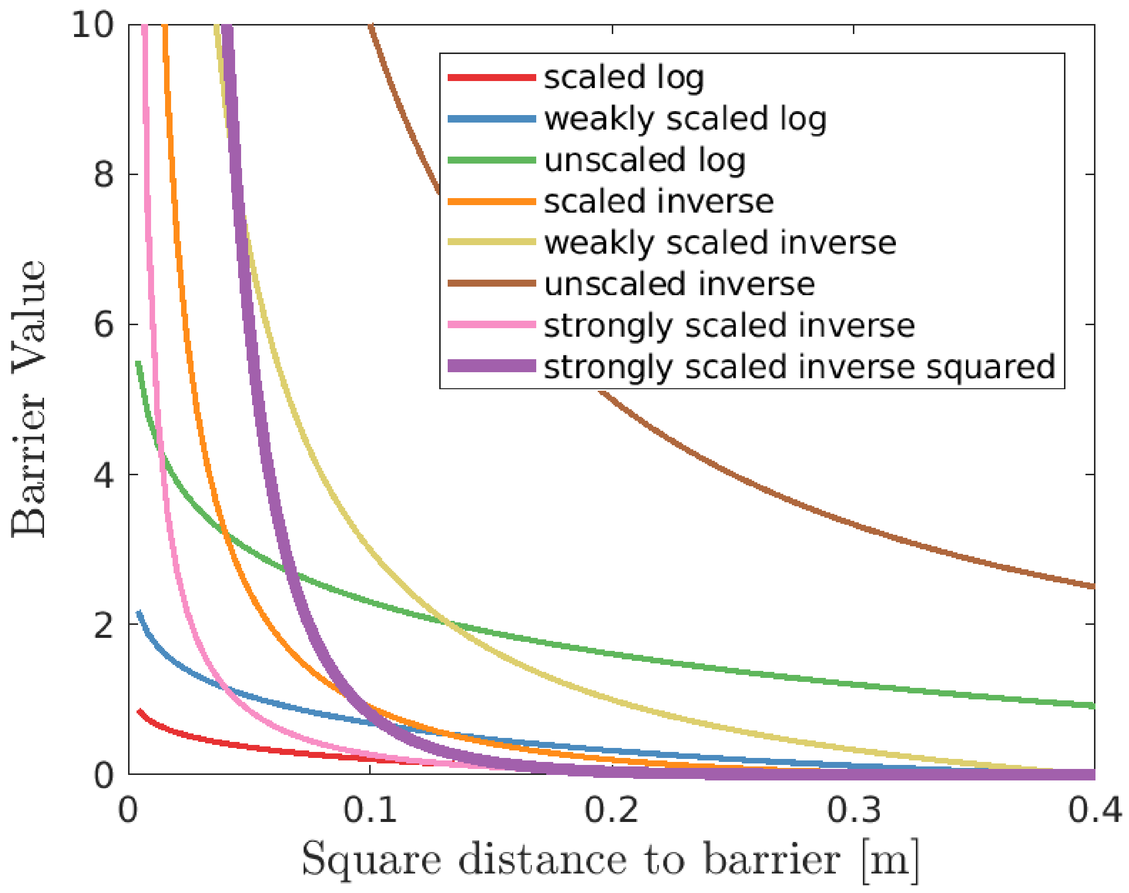


Figure 7: Comparison of potential barrier value functions. The barrier value functions considered for obstacle avoidance are plotted against distance to the barrier for the range $[0, \rho]$ with $\rho = 0.4$ in which obstacle avoidance is active.

During simulation it was observed that the inverse functions were more aggressive than desired, preventing the agents from entering the repulsion region; this aggression improved obstacle avoidance at the cost of path efficiency since the agents traveled farther around the obstacles than necessary, and also limited the cases in which agents could travel between obstacles. In general, the distance between obstacles must be at least $2h$ plus the diameter of the drone in order for an agent to pass between them; if the barrier value prevents the agent from entering the repulsion region, the minimum separation between obstacles increases to $2(h + d)$ plus the drone diameter to allow safe passage. The scaled log function used in literature was found to be too weak, and often did not dominate over the measured signal value and therefore was ineffective at bending the gradient around the obstacle; this issue appeared to be a gain tuning problem, indicating that the methods discussed in literature may be sensitive to gain selection. The selected barrier value function was the strongly scaled inverse squared, $B(z) = -\frac{(\rho-z)^4}{z^2}$, for its ability to allow agents to enter the repulsion region while still dominating the gradient direction.

A concern with directly applying any barrier value function from the list in Table 1 was ensuring it dominated over the measured signal at all times. This was accomplished by including a scaling term in the barrier value function dependent on the measured signal to force the magnitude of the barrier value to always be larger than the magnitude of the measured signal. The equation is shown in (17).

$$B(z,J) = \frac{-(|J|+1)^2(\rho-z)^4}{z^2} \tag{17}$$

2.3.4 Obstacle Rotation

A limitation found in literature was a lack of generality when applying the obstacle avoidance strategy to different environment setups, in that the orientation of the box placed around an obstacle was selected such that the hybrid control modes were oriented in the x direction (see [6]). This appeared to be based around the assumption that the obstacle is always placed directly between the agents' initial positions and the target's location so that the x-axis can be appropriately defined, which relates to the previously discussed limitations imposed by assuming the obstacle position(s) are known. Further, it indicates an assumption that the relative position of the obstacles to the target is known, which is inconsistent with the objective of locating target in an unknown environment. Consideration of cases where the agent encounters obstacles at unknown angles relative to the target location motivates the need for barrier orientation selection. In object identification, a more adaptable approach is used, such as in [13]; this type of dynamic orientation assignment was leveraged in this work to provide robustness to obstacle avoidance in a wider range of scenarios. The addition of obstacle orientation consideration is an extension to the work presented in [24] in order to better generalize the approach.

Instead of orienting the virtual box to be aligned with the x-axis, when an obstacle is detected the orientation is defined to align with the estimated direction of the target, that is, the direction of the gradient estimate. This adaptive orientation assignment enables the agents to apply the same obstacle avoidance strategy without reference to a specific configuration orientation. The orientation adjustment is implemented by applying a rotation to the agent's position, converting the coordinate system to the one assumed in previous work at which point the standard obstacle avoidance previously described can be implemented directly. Figure 8 illustrates an example of the orientation adjustment, showing a case where the original method of aligning the obstacle barrier lines does not achieve the desired placement of the mode overlap to allow the agent to decide which direction to take. The rotation is then performed on the same setup using the calculated gradient as an estimate to the direction of the target to align the mode decision region with the agent.

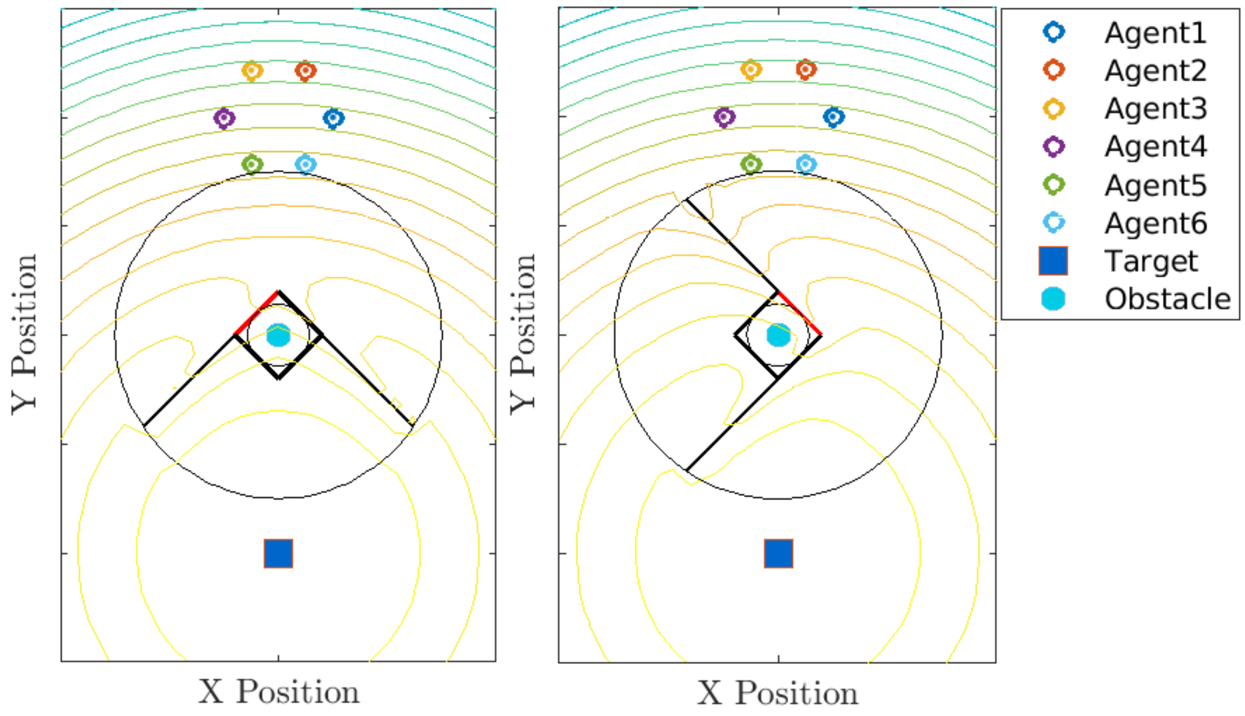


Figure 8: Motivation for obstacle barrier rotation. Obstacle configuration where unrotated modes do not place the decision region in the appropriate location (*left*), resulting in the same potential failure as in the purely continuous obstacle avoidance strategy, and the rotated version that enables robust obstacle avoidance to be applied properly (*right*). The red virtual box edge provides a reference to better view the rotation. The contour lines illustrate the augmented signal and how the obstacle barrier directs the agents as they follow their perceived gradient. The rotation in this figure was computed to align with the target, while in practice the estimate of the target direction is used.

2.3.5 Test Cases

To demonstrate the operation of the proposed solution, a series of obstacle and target configurations was simulated in Matlab to illustrate how the proposed solution addressed different scenarios. The cases of interest are shown below in Figure 9, illustrating the location of the agents' initial positions along with the obstacle and target configuration.

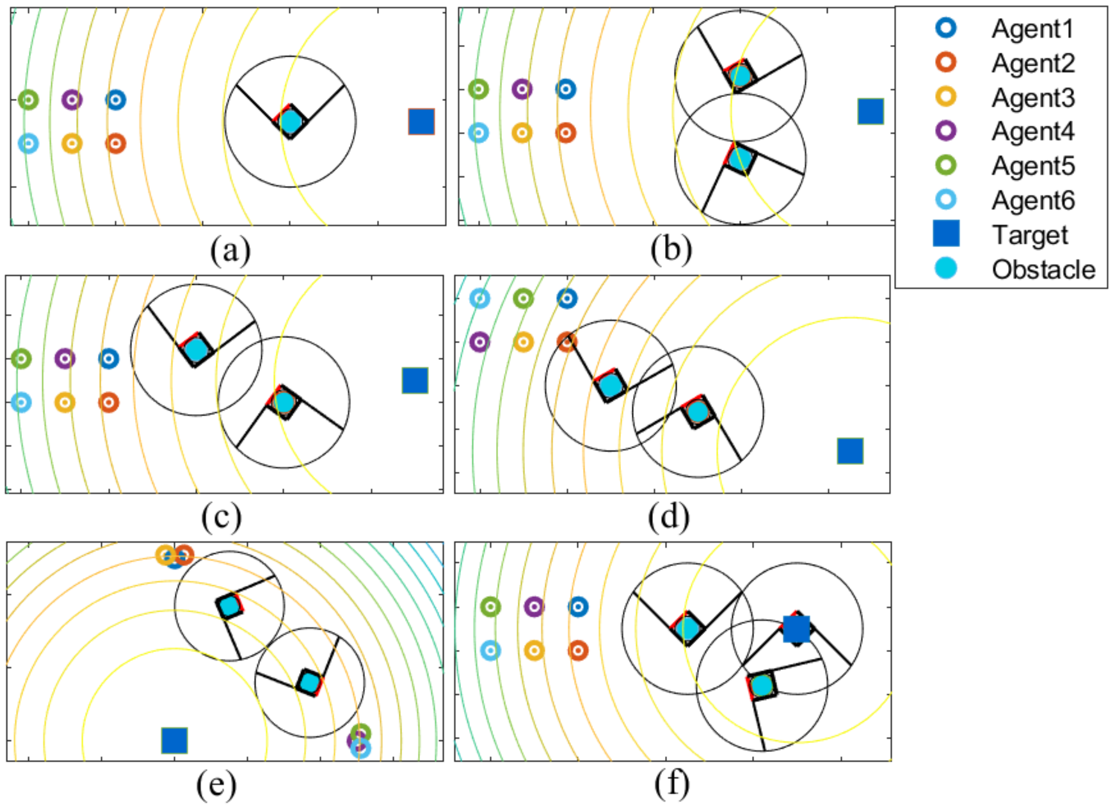


Figure 9: Selection of simulation test configurations. The six different cases are used to demonstrate capability of the proposed algorithms to operate in varying unknown environments.

The first test considered, located in Figure 9.a, demonstrates the six agents in the canonical example configuration, in which a single obstacle is positioned directly between the agents' initial positions and the target. This test was included since it showcases the scenario with a clear decision point, and it enables more direct comparison to literature since the configuration is commonly used (see [6], [14], [15]). The second test, located in Figure 9.b, shows how the swarm responds to a bottleneck situation where multiple obstacles must be avoided; this highlights the effects of formation adaption to the unknown environments. In the third test, Figure 9.c, the obstacles are offset in the x direction to show the changing modes in the hybrid controller as the agents respond to first a single obstacle and then two obstacles. Test 4, in Figure 9.d, illustrates how the control method is impacted by changes in the angle of approach; this is to highlight the generalized nature of the algorithms, as discussed in Section 2.3.4. For test 5, shown in Figure 9.e, two sub-swarms attempt to rendezvous while locating the target, avoiding obstacles that prevent them from achieving their formation initially. The final test, Figure 9.f, shows a different type of target locating problem in which the target itself is viewed as an obstacle; this represents a case where the agents must not collide with the target, essentially changing the problem to be determining which obstacle in an unknown environment is emitting the signal. Such a problem is more complicated to solve since obstacle avoidance and target locating come more closely into conflict, but it represents a more realistic scenario in which the target can be a physical object.

The details of the developed algorithms and the meaning of the regions around the obstacles will be highlighted in the following sections. The parameters for each test case are shown in Table 2.

Table 2: Simulation Control Parameters

Parameter	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**
num saved	5	5	5	5	5	5
α	1.8	1.8	1.8	0.5	1.0	1.8
β	5	5	5	5	5	5
κ	1.5	1.5	1.5	1.5	1.5	1.5
h	0.4	0.4	0.4	0.4	0.4	0.4
ρ	0.4	0.2	0.4	0.4	0.4	0.2
μ	0.8	0.8	0.8	0.8	0.8	0.8
maxVel	1.5	1.5	1.5	1.5	1.5	1.5
A_0	0.5	0.5	0.5	0.5	0.5	1.0

2.3.6 Algorithm 1 Simulations

The algorithm was tested in simulation as described in Section 2.3.5 to show its effectiveness in a variety of environment configurations. The trajectories are shown in Figure 10 below, illustrating how the agents select a path around the obstacles while they navigate toward the target.

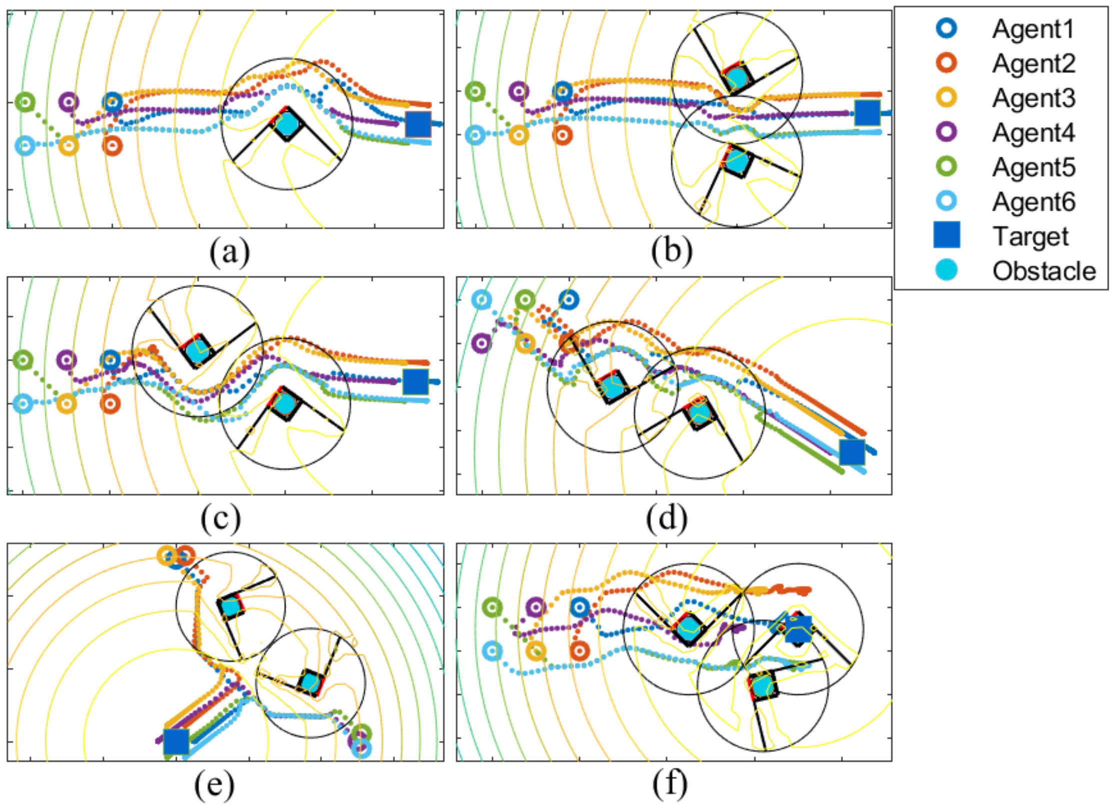


Figure 10: Obstacle avoidance algorithm 1 simulation results. Varying obstacle configurations illustrating the agents' trajectories as they navigate the unknown environment to locate the target. Axes are the xy plane, plotted x vs y.

2.3.7 Algorithm 1 Limitation

A limitation of the fully distributed gradient estimation method is the resolution provided by the formation: the agents struggle to accurately identify the position of obstacles whose virtual box height is smaller than the radius of the formation. Consider the case presented in Figure 11, in which four agents are used to locate the target in the presence of obstacles that have approximately the same size as the agents. The formation is therefore much larger than the obstacles to be avoided, and the reduced resolution can result in collision.

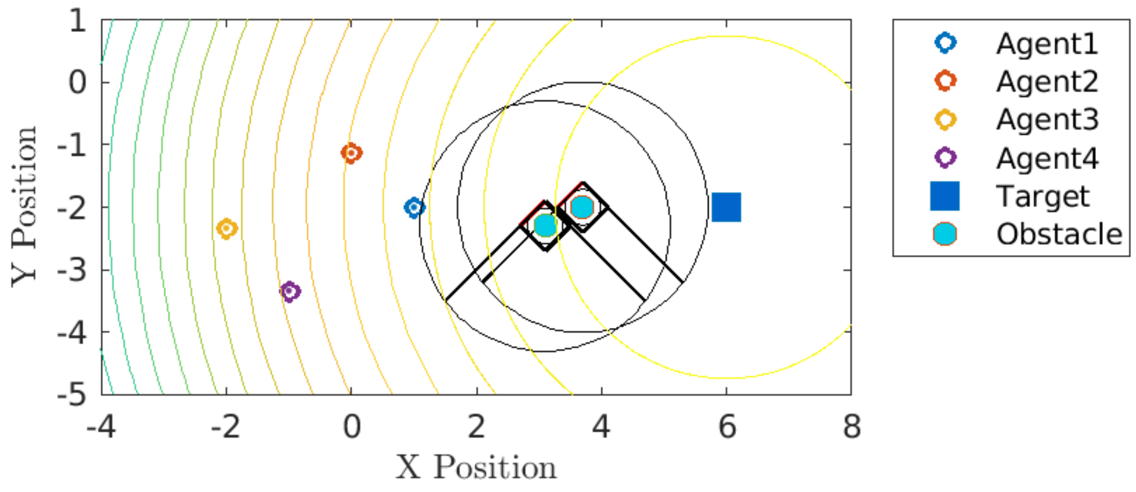


Figure 11: Example configuration of obstacle avoidance algorithm 1 limitation. Unknown environment configuration where the obstacles' virtual box cross-section is much smaller than the radius of the formation. The reduced resolution afforded by the large formation results in a collision with the obstacle. While the nature of swarming enables the mission objective to be met even when an individual agent is lost, the failure mode was investigated to provide insights into the development of an alternative obstacle avoidance algorithm that reduced loss of hardware.

In this case, agent 3 consistently collided with an obstacle. The collision stems from the distributed nature of the obstacle avoidance, in which aggregate knowledge is used to solve a local problem. In this architecture, an agent attempts to avoid the obstacle by augmenting the measured signal directly, comparing to the values recorded by its neighbors, and estimating the gradient accordingly. The agent uses information that may not account for the location of the obstacle since the neighbors are not necessarily aware of the obstacle, which causes cases to arise when the agent misidentifies the appropriate gradient direction. Such a case can be seen in the gradient field shown in Figure 12 where agent 3's gradient estimate passes directly through the obstacle instead of bending around. The issue was that near the obstacle barrier, agent 3 recorded a very small value for the signal since it was applying the barrier value to create a well near the obstacle, while its neighbors were sufficiently far from the obstacle that they were not applying the avoidance term and therefore had relatively large values for the signal. In most cases this would result in agent 3 pushing away from the obstacle; however, because all its neighbors were on the far side of the obstacle compared to agent 3, it was unable to determine the relative direction of the obstacle. To agent 3, it appeared that the signal increased in the forward direction, indicating that the obstacle was behind it. This was not the case, so the agent collided with the obstacle. A depiction of the gradient vector field for each agent given the implemented formation is shown in Figure 12.

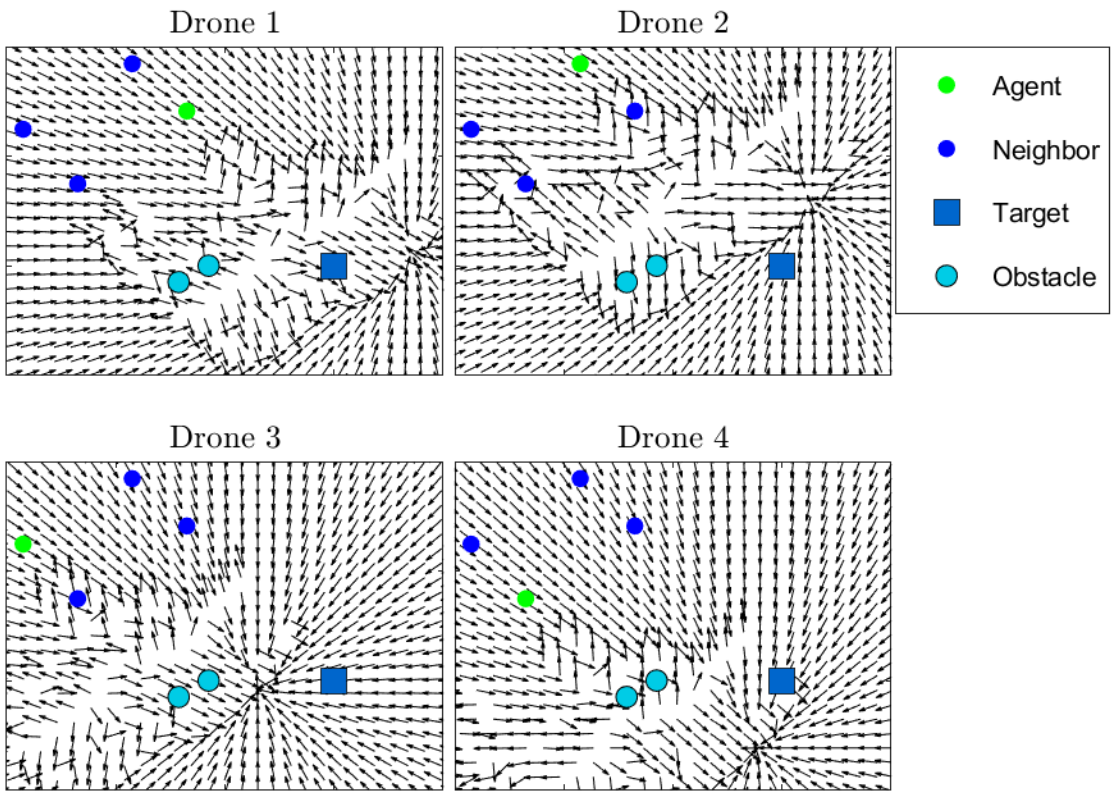


Figure 12: Estimated gradient field with large swarm formation. Vector field of the gradient estimated by each agent given the rigid formation in (16) over the trajectory space. The true locations of the obstacles and target are shown. The ghost obstacles appear where the distributed nature of the augmented gradient estimate causes each agent to perceive the obstacle to be located. For this formation, a radius of $A_0 = 2.0$ was selected.

The information provided by the distributed formation on how the gradient should be augmented creates ghost obstacles that are offset from the true obstacle location. The equilibrium point shown in Figure 12 is also offset from the target for each agent, which is expected since the centroid of the formation should ultimately be collocated with the target; this offset underscores how the gradient can become distorted by the formation size. When a smaller formation radius is used, such as $A_0 = 0.5$ instead of $A_0 = 2.0$, the ghost obstacles begin to converge to the true obstacle locations, as shown in Figure 13.

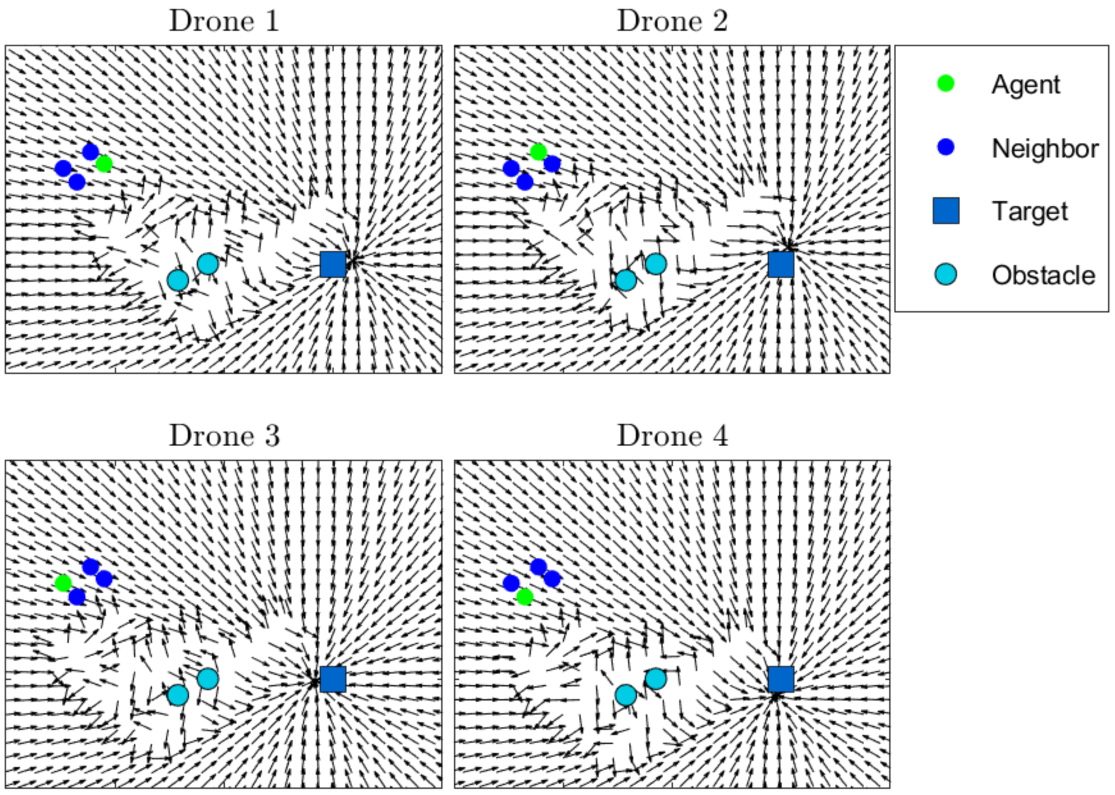


Figure 13: Estimated gradient field with small swarm formation. The smaller formation radius of $A_0 = 0.5$ enables gradient estimate of the individual agents to converge to a consensus.

The results indicate that the size of the box placed around the obstacle (defined by height h) should be selected in part based on the radius of the formation. The formation size determines the resolution of the gradient field, so large formations are not always able to resolve the position features that are smaller than the formation radius. While decreasing the formation radius mitigates the severity of the ghost obstacles and would likely enable the agents to avoid the obstacles under standard conditions, the result is not robust. This inherent limitation motivates the development of an alternative obstacle avoidance strategy.

2.4 Obstacle Avoidance Algorithm 2

To address the limitations of algorithm 1, an update to the obstacle avoidance architecture was implemented in the form of algorithm 2. The new architecture was developed to use local information to avoid the obstacle instead of aggregate information. Previously, obstacle avoidance was embedded directly into the gradient estimate, forcing the contour lines to curve around the obstacle. With the new architecture, the gradient is estimated with the unaugmented measurements to determine the direction toward the target, and obstacle avoidance is applied on top of the main controller.

The barrier value is computed in the same manner as developed in algorithm 1 using the form $B(z) = -\frac{(\rho-z)^4}{z^2}$ but is not applied to augment the measured signal. Instead, it is only used to update the mode so that the appropriate barrier is used, allowing algorithm 2 to retain the robustness advantages of the hybrid controller. The measured signal scaling term applied in algorithm 1 is not used to compute the barrier value in algorithm 2 because there is no need to have the barrier value dominate the measured signal since it is not applied to the gradient estimate.

The non-collision term η for obstacle avoidance for M obstacles is computed as

$$\eta = \sum_{k=1}^{M} B_k(z)\hat{\mathrm{n}}_k \tag{18}$$

where $B_k(z)$ is the barrier value in the appropriate mode and $\hat{\mathrm{n}}_k$ is the normalized vector pointing away from the barrier for the *k*th obstacle. The non-collision term is applied to the target locating velocity (u_T) to get the desired agent velocity (v). Note that the same switching formation controller used in algorithm 1 is applied here, so if obstacle avoidance is active the formation control gain is set to zero. The commanded velocity can be found as

$$v = u_T + \eta \tag{19}$$

which results in the portion of the velocity in the direction of the obstacle barrier being removed from the target locating velocity, forcing the agent to take a path around the obstacle.

Obstacle rotation remained the same as in algorithm 1; algorithm 2 was found to be more suitable for obstacle rotation since the gradient estimate is not augmented and therefore more consistently offers a reliable estimate of the direction to the target.

2.4.1 Algorithm 2 Simulations

Algorithm 2 was tested in the same simulation environments as algorithm 1, described in Section 2.3.5, to show its effectiveness in a variety of environment configurations and provide a performance comparison. The trajectories are shown in Figure 14, illustrating how the agents select a path around the obstacles while they navigate toward the target.

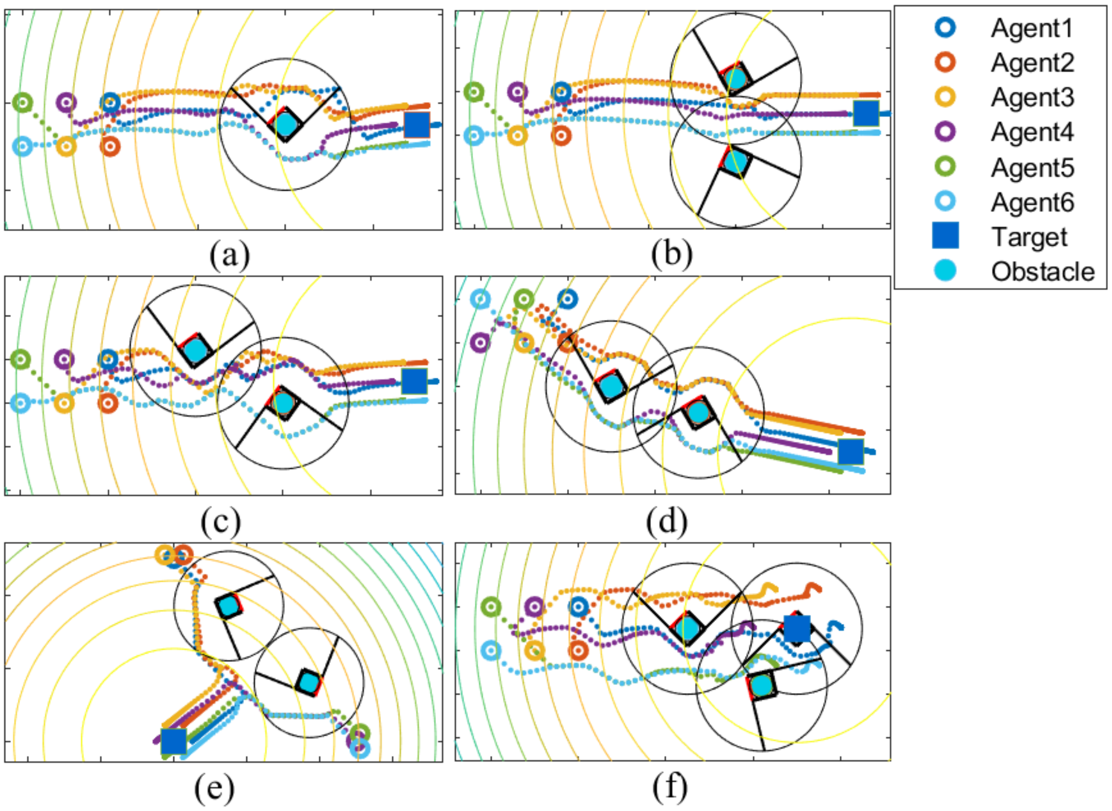


Figure 14: Simulation results of obstacle avoidance algorithm 2. Varying obstacle configurations illustrate the agents’ trajectories as they navigate the unknown environment to locate the target.

2.5 Comparison of Algorithms

To verify that algorithm 2 successfully addressed the formation resolution concern presented in Section 2.3.7, a simulation was performed to compare the trajectories taken for a case in which algorithm 1 was known to fail. The initial positions of the agents, target, and obstacles were perturbed over a series of tests in a low-order Monte Carlo simulation to illustrate the paths the agents follow under different circumstances. A boundary avoidance strategy that will be detailed in Section 3.2.2 was also implemented to demonstrate a constrained environment. Algorithm 1 simulation results are shown in Figure 15.

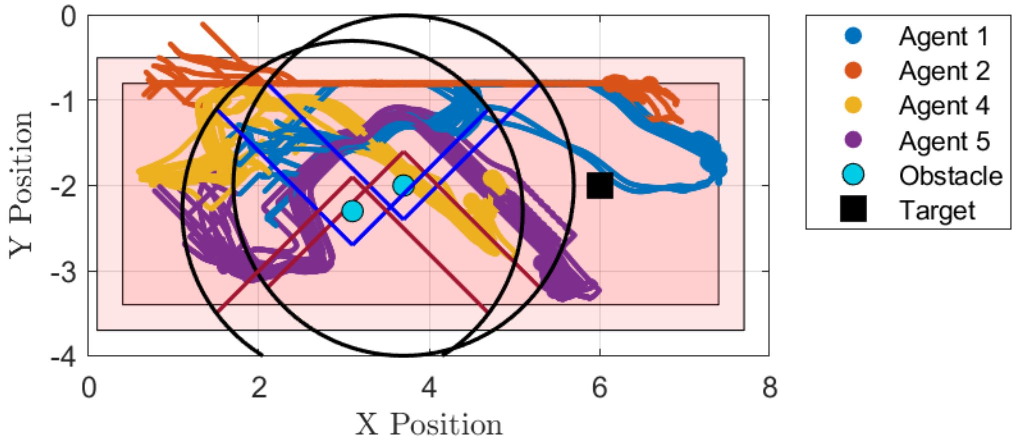


Figure 15: Simulation of algorithm 1 limitations due to insufficient formation resolution. Agent 4 (yellow) does not successfully avoid the obstacle since its perceived barrier location is offset from the true barrier position (maroon). Agents 1 and 2 (blue and red, respectively) are constrained by the boundary (light red box) of the safe operating region. The results show the trajectories for a perturbation test including 20 trials using different initial conditions.

Algorithm 2 was tested under the same simulation parameters, with the results shown in Figure 16.

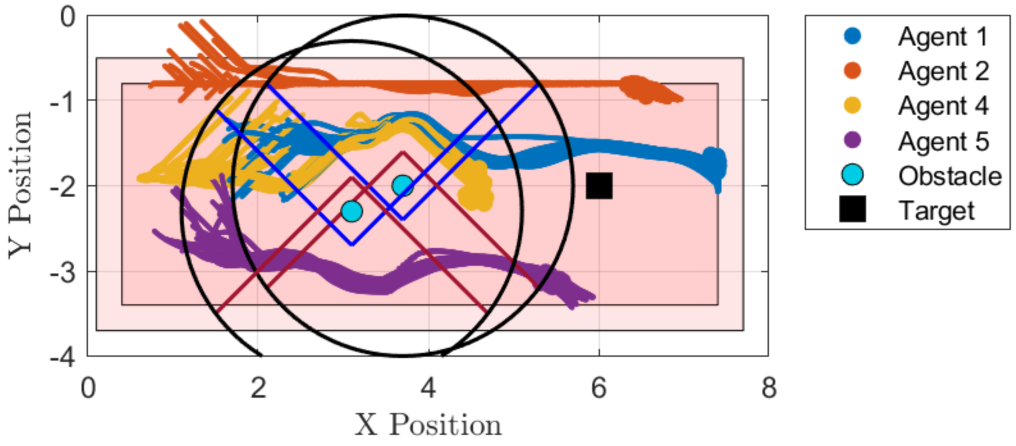


Figure 16: Simulation of algorithm 2 overcoming algorithm 1's formation resolution limitations. All agents successfully avoid the obstacles since the perceived barrier position is no longer variable with formation size. Agents 1 and 2 (blue and red, respectively) are constrained by the boundary (light red box) of the safe operating region. The results show the trajectories for a perturbation test including 20 trials using different initial conditions.

It can be seen that algorithm 2 successfully avoided the obstacles in the case where the formation radius was significantly larger than the obstacle size, as expected. The updated algorithm maintained the capabilities of the distributed version, providing improved results without requiring additional data. It was determined that algorithm 2 was more effective at locating the target in unknown environments because of the reduced danger of loss of agents from collisions.

2.6 Summary

This chapter detailed the theoretical approach used to address the problem of target locating in an unknown environment, featuring several improvements on and additions to methods described in literature. Within the target locating phase, the distributed least-squares method of gradient estimation was adapted to be more robust to loss of spatial diversity through the implementation of measurement memory, allowing the agents to overcome scenarios in which the formation did not have full position rank; in addition, a new normalization strategy was proposed to enable the algorithm to be more robust to small perturbations to the agents' positions as the gradient approached zero, enabling more definitive target locating.

The key contributions made to obstacle avoidance were extending robust hybrid control techniques to real-time obstacle detection, enabling the algorithm to be used in unmapped environments with multiple obstacles in unknown configurations. The problem was posed as a distributed algorithm, removing the single-point failure present in [6]. Table 3 summarizes the comparison of the two algorithms developed in this work to the methods used in literature, highlighting the advancements and contributions.

Table 3: Comparison of Algorithm Capabilities

	Rosero [11]	**Khansari-Zadeh [13]**	**Mayhew [15]**	**Poveda [6]**	**Alg.1**	**Alg.2**
Distributed Algorithm	X				X	X
Target Locating	X		X	X	X	X
Multi-Agent	X			X	X	X
Hybrid Robustness			X	X	X	X
Multi-Obstacle		X			X	X
Obstacle Detection		X			X	X
Obstacle Orientation		X			X	X
Non-collision robustness		X	X	X		X

3 SpaceDrones Implementation

Distributed swarming capabilities were implemented on the VT SpaceDrones hardware. In addition to distributed algorithms, in which individual agents used their own information and information provided by their neighbors to determine their desired velocity without relying on a centralized node, the implementation of the algorithms was designed to be fully distributed so that computation was performed onboard for each agent, rather than computing velocities locally and broadcasting commands to the swarm. This approach more closely resembles a realistic environment where the agents are exploring an unknown environment, and implementing the algorithms in a distributed manner showcases the information exchange among agents. The setup used is described in Section 3.1.

The safety features implemented for swarming tests are presented in Section 3.2. These include boundary avoidance and drone non-collision algorithms demonstrated within the context of cooperative swarming. Limitations caused by practical implementation including communication latency and vehicle dynamics are discussed, and the methods used to mitigate these issues are demonstrated.

Initial work on distributed autonomous coordination included formation flying of three heterogenous drones. The results from these tests are include in Section 3.3, demonstrating the fundamentals of the swarming capabilities developed in this work.

To overcome limitations on the swarm size imposed by available hardware and flight space, the concept of augmented reality was explored and virtual drones implemented. The approach used and advantages gained are detailed in Section 3.4.

The proposed target seeking algorithm is demonstrated in a series of parts. The distributed gradient estimation algorithm is implemented in Section 3.5, illustrating the convergence of the swarm to the target without interference from obstacles. Section 3.6 shows the results of two heterogenous physical drones cooperating with two virtual drones to avoid different configurations of up to three obstacles, while following the estimated gradient to locate the target. The tests showcase the core functionality of the algorithm in a typical use case, but do not explore all the capabilities detailed in the simulation results. Limitations on the original distributed approach are demonstrated and discussed, and algorithm 2 is presented as an alternative obstacle avoidance scheme. The results from the algorithms are compared.

3.1 SpaceDrones Hardware Setup

Testing and verification of the proposed solution was performed in the SpaceDrones test facility located in Randolph 26 at Virginia Tech. The capabilities of the lab are described in [26]–[28] and are summarized below for completeness.

The drones used are a combination of quadrotors and hexarotors featuring a Pixhawk for lower level flight control and a Raspberry Pi companion computer for implementation of the algorithm under test. Agents from the RP and M lines of SpaceDrones hardware were used in the demonstrations. Take-off and landing procedures are handled by heritage code developed in-house that enables a plug-and-play experience for testing new algorithms.

The agents, obstacles, and target positions were tracked using the Optitrack Motion Capture System featuring 25 cameras including Prime 13 and Prime 13W models. For demonstration purposes, the obstacle and target positions were then hard coded into the drones' flight code; however, with few code modifications, the implementation could be adapted to receive those positions directly from OptiTrack. The drone positions were transmitted from OptiTrack to each drone through the Robot Operating System (ROS) using multicast over a private wireless network, and the communication graph was handled by each agent on board to simulate communication limitations. The tests performed assumed all-to-all communication among the drones due to the limited number of agents in the swarm.

SpaceDrones experimental software integration enables either position or velocity to be commanded; for this implementation, velocity was selected as the controlled variable. In particular, linear x, y, and z velocities were controlled; angular z velocity was not necessary for this project.

The SpaceDrones platform differs from similar testbeds (e.g., Georgia Tech's Robotarium [29]–[31], JPL's swarm platform [32]) in that it provides a fully distributed implementation, meaning the computation is performed onboard the drones in real time rather than being broadcast by a centralized node. This enables testing of distributed swarming algorithms in a setting that more closely resembles the final deployment to the field, in which the agents would not have access to a central computer.

The Python scripts used in this project, forming the Experiment Scripts in SpaceDrones terminology, are included in the GitLab repository *Target Locating in Unknown Environments Using Distributed Autonomous Coordination of Aerial Vehicles* [33]. These scripts include the target locating, obstacle avoidance, and formation control algorithms, in addition to the custom data saving functions that enable insight to the different components of the velocity generation, including elements such as detected obstacle mode and position, formation control, and estimated gradient. The parameters that describe a specific test, such as formation radius, safety features included, number of agents, and test purpose are saved in a separate file.

The setup used for unknown environment exploration is pictured in Figure 17. Appendix A describes the methods used to generate the test data.

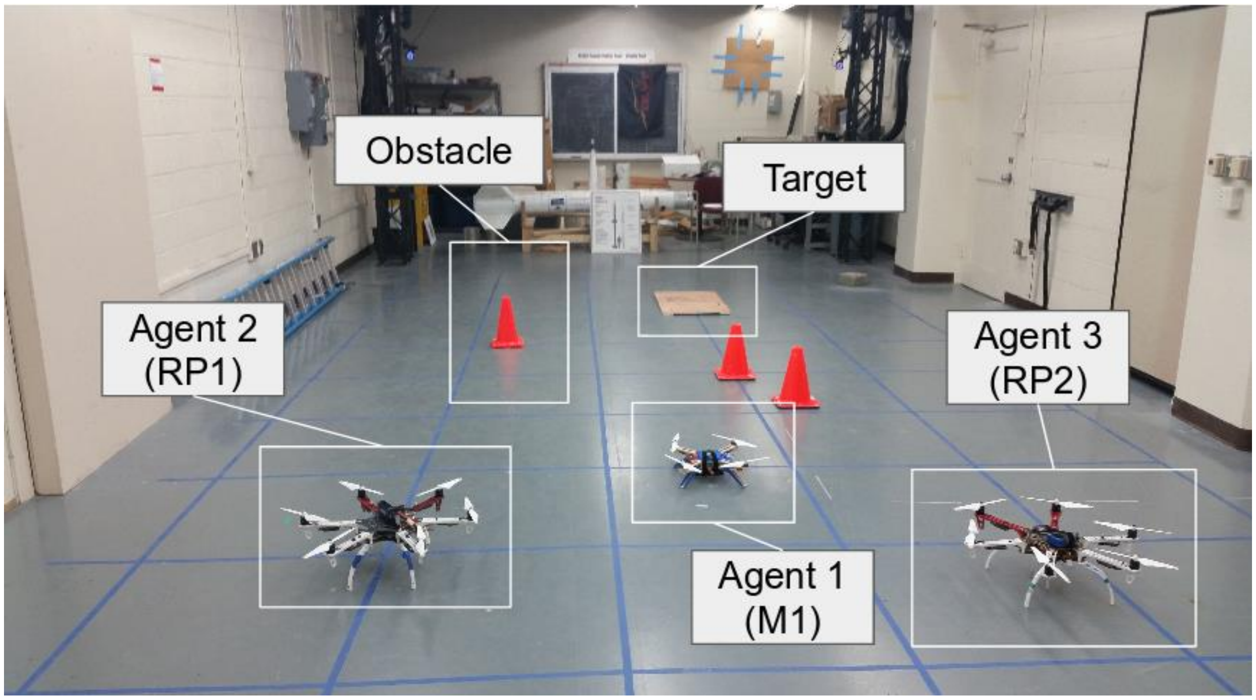


Figure 17: SpaceDrones lab setup for target locating in an unknown environment. Note that Agent 3 (RP2) was not used during unknown environment exploration tests.

3.2 Safety Features - Boundary Avoidance and Non-collision

When flying the algorithm on hardware, it is critical to implement certain safety features to reduce the risk to the drones and supporting equipment. The safety features added in this project include boundary avoidance, which prevents the drones from leaving a designated space to avoid collisions with the walls, and non-collision, which prevents the drones from colliding with one another.

Due to its importance in general implementation, non-collision has been discussed in swarming algorithms in literature. A typical approach is to include the safety features in the path planning algorithm itself, where the computed path is designed to preclude collisions by applying a severe cost in the reward function to cases that result in collision ([34], [35], [36]). Alternatively, collision avoidance can be handled from a graph theory approach via edge-weighting, as presented in [37].

Disadvantages can arise when the planning is computed offline, such as proposed in [38], in which unanticipated disturbances could result in an agent deviating from its intended path and colliding with a neighbor. Real-time collision avoidance as a constraint in path planning overcomes this issue, but it requires the non-collision to be weighted sufficiently strongly to ensure that the perceived benefit does not outweigh the cost of collision. However, this type of approach is not as applicable to the problem investigated in this paper since the path planning portion of the algorithm is based on gradient descent and does not employ a cost function as other popular algorithms such as POMDP and MILP use; therefore, a more generalized approach is needed.

Another approach discussed in literature is the use of Voronoi diagrams (see [39]) to limit the allowable travel space for each agent in a way that mathematically guarantees the allowable regions for different agents do not overlap, as discussed in [32]. This type of algorithm is useful for more chaotic formations where agents' paths are likely to cross, such as in an uncoordinated exploration pattern, but it introduces restrictions on the agents' paths that may conflict with a rigid formation such as used in this paper. In future iterations, if a more chaotic swarming formation is desired, a Voronoi diagram may be considered for collision avoidance.

Barrier Certificates, described in [40], is the method used in the swarming testbed Robotarium [31] in which collision avoidance is implemented in real time and can be applied to any type of control law. The commanded control is overridden if the agents are in danger of collision, causing a safer path to be taken, while maintaining a similar trajectory in a least-squares sense by using Quadratic Programming (QP) to compute the updated path. The key disadvantage of this approach is that it is implemented in a centralized manner and therefore does not scale well with large swarms, though there is a proposed alternative that can be implemented in a distributed fashion ([40]).

The approach used in this work is a layered collision avoidance strategy, similar to the Barrier Certificates, but with a simplified implementation that does not require the computation expense of QP optimization. The implemented methodologies and experimental results are presented in the following sections.

3.2.1 Projection Function and Naive Avoidance Version 1

The initial approach was to use a projection function to slow the agent as it nears the element of danger. The method scales the velocity to zero in the direction of the boundary when the agent is moving toward the boundary, preventing a collision since the agent will gradually slow to a stop. Non-collision works in a similar manner, using a scale factor to drop the velocity to zero when within a certain radius of a neighboring agent; direction of travel is not accounted for in non-collision to make the algorithm more robust to mobile noncooperative agents, such as when a neighbor is not responding properly to controls. This latter case is not fully addressed in this algorithm in that the agent only slows to a stop and is not repulsed from its neighbor, so the dynamic aspect of an uncontrolled agent could result in collision.

3.2.1.1 Boundary Avoidance

In the context of this work, boundary avoidance is the mechanism used to keep the drones within a desired volume defined by minimum and maximum allowable values for each Cartesian dimension (x, y, and z). The purpose of boundary avoidance is to prevent the agent from leaving the region in which its location can be detected by OptiTrack, and to prevent collision with the walls, floor, and ceiling. The main parameters describing boundary avoidance are depicted in Figure 18.

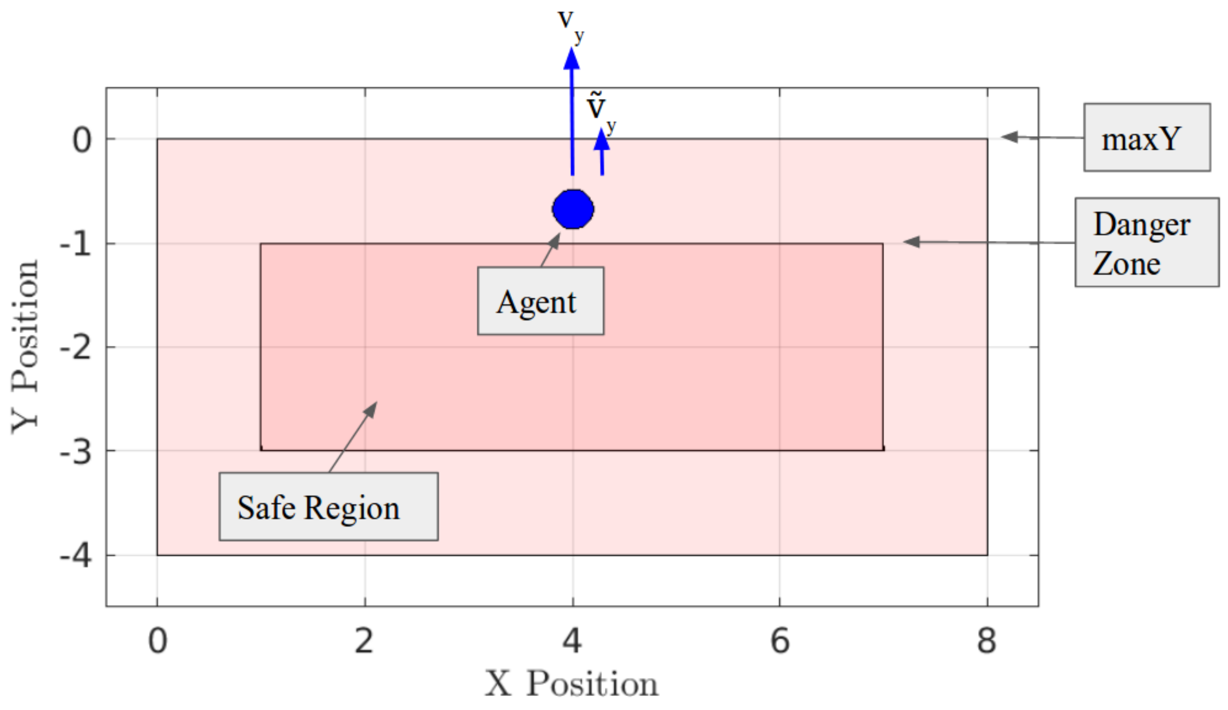


Figure 18: Depiction of an agent applying boundary avoidance. The agent (blue) is within the danger zone, approaching the upper bound on y, denoted *maxY*, with its velocity v_y. The scaled velocity $\tilde{v}_y$ has a smaller magnitude than the original velocity to avoid collision with the boundary.

The pseudocode illustrating how to compute the scale factor of the boundary avoidance algorithm at a given position x is shown below.

```
BOUNDARY AVOIDANCE VERSION 1
1   if $x > x_{max} - d$ and $\dot{x} > 0$
2       ▻ $x$ is in danger zone and moving toward maximum boundary
3       $scale \leftarrow (x_{max} - x)/d$
4   else if $x < x_{min} + d$ and $\dot{x} < 0$
5       ▻ $x$ is in danger zone and moving toward minimum boundary
6       $scale \leftarrow (x - x_{min})/d$
7   else
8       ▻ $x$ is in safety region
9       $scale \leftarrow 1$
```

The projection function generates the scale factor to be applied to the velocity by determining if the agent is within distance d of the boundary. If the agent is moving toward the boundary, the velocity will be linearly scaled to zero; if the agent is moving away from the boundary, the velocity is not scaled because a collision with the boundary is not imminent. Note that if the agent has moved beyond the boundary and is continuing to move further from the boundary, the scale factor becomes negative and the agent is directed back to the interior of the defined region. Practically, it is useful to include a buffer region between the algorithm boundary and the physical wall for cases when a drone might overshoot; ideally, this buffer would be minimally sized to enable more of the space to be used.

3.2.1.2 Non-collision

For non-collision, the scale factor linearly decreases from 1 to 0 as the agent moves deeper into the danger zone, the distance at which the agent begins to apply collision avoidance, toward the safety radius, the minimum distance allowed between agents. This causes the agent to slow to a stop when it is near another agent. This non-collision algorithm does not take velocity direction into account, instead applying the scale factor whenever the neighboring agent is within the danger zone range. An example of non-collision between two agents is provided in Figure 19.

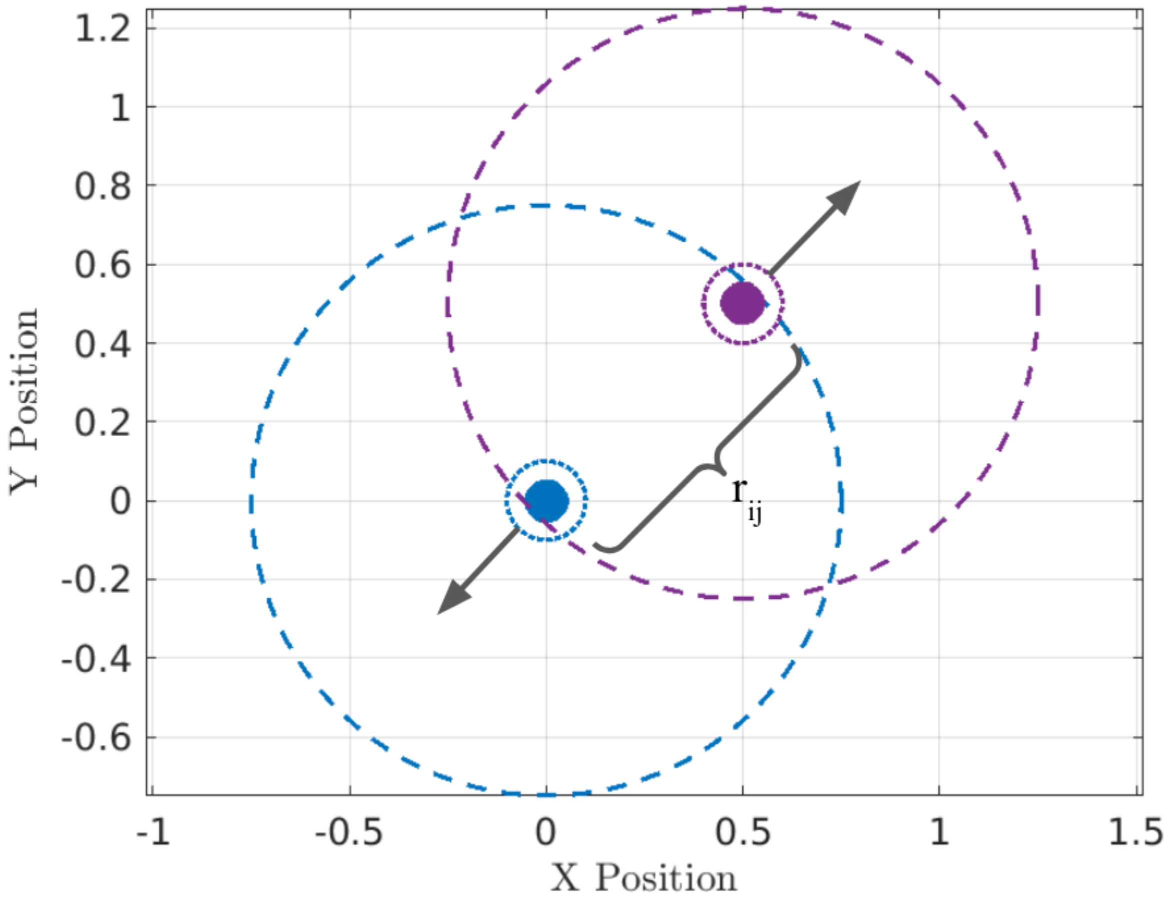


Figure 19: Image depicting non-collision. The danger zone (dashed line) defines the radius within which collision avoidance is applied. The safety radius (dotted line) represents the barrier which a neighboring agent does not cross. The arrows represent the repulsive effect, scaling the velocity to zero as the distance r_{ij} decreases to zero.

The scale factor to prevent inter-agent collisions for a given agent separation of r_{ij} can be implemented as follows:

Non-Collision Version 1

```
1  **if** $|r_{ij}| < danger\_zone$
2      ⊳ apply noncollision within danger zone radius
3      $scale \leftarrow (|r_{ij}| - safety\_radius)/(danger\_zone - safety\_radius)$
4  **else**
5      ⊳ $x$ is in safety region
6      $scale \leftarrow 1$
```

3.2.1.3 Implementation

To apply the safety features to the main algorithm, the code described in Implementation

OF SAFETY SUITE VERSION 1 can be implemented. After calculating the desired velocity for the algorithm under test (i.e., formation flying, target locating), the boundary avoidance and non-collision scale factors are applied. Scaling the velocity after the main algorithm has been addressed causes the safety features to overcome the main algorithm objectives to ensure the safety of the hardware; similarly, non-collision is applied after boundary avoidance since a collision between drones poses a more severe threat than a collision with a boundary.

IMPLEMENTATION OF SAFETY SUITE VERSION 1

```
1  ⊳ apply boundary avoidance in each dimension
2  $velocity_x \leftarrow boundScale_x \times velocity_x$
3  $velocity_y \leftarrow boundScale_y \times velocity_y$
4  $velocity_z \leftarrow boundScale_z \times velocity_z$
5  for $ii \leftarrow 1$ to $|N_j|$
6      ⊳ scale velocity to avoid each neighbor
7      $velocity \leftarrow noncol[ii] \times velocity$
```

A series of tests was performed to determine the performance of the safety features on the drones during actual flight.

The boundary avoidance control law was tested with a single agent by following an ellipse that used the available room space, first without boundary avoidance and second with a virtual room smaller than the actual room to artificially restrict the agent's movement. For safety purposes and to limit damage to the agents in case the boundary was crossed, the implemented boundary restricted the agent to the space defined by $\{x \mid -1.3 \leq x \leq 1.3\}$, $\{y \mid -0.5 \leq y \leq 0.5\}$, and $\{z \mid -0.5 \leq z \leq 1.9\}$, with boundary avoidance disabled for the subspace $\{x \mid -0.8 \leq x \leq 0.8\}$, $\{y \mid -0.2 \leq y \leq 0.2\}$, and $\{z \mid -0.75 \leq z \leq 1.65\}$. Note that these values are centered about the origin of [0,0,0] being the floor at the center of the room; however, the remaining tests were conducted with the center of the room defined to be the point [4.0, -1.54, 0], providing a reference frame that better corresponded to the grid outlined on the floor of the test facility.

The drone tests were conducted using drones RP3 with the standard SpaceDrones PID controller, with control gains provided in Table 4.

Table 4: PID Gains for Boundary Avoidance Initial Testing

PID Gain	Value
Proportional Gain	0.05
Derivative Gain	0.015
Integral Gain	0.05

The test results showed that the boundary avoidance did not successfully restrict the agent to the desired box. Analysis indicated that the dynamics of the drone not perfectly resembling single-integrator dynamics, coupled with communication latency resulted in the drone not updating its velocity quickly enough to meet the boundary avoidance requirements; the agent overshot the boundary before being forced back into the desired region. The simulated results illustrating both the purely theoretical and more realistic responses are shown in Figure 20 and Figure 21.

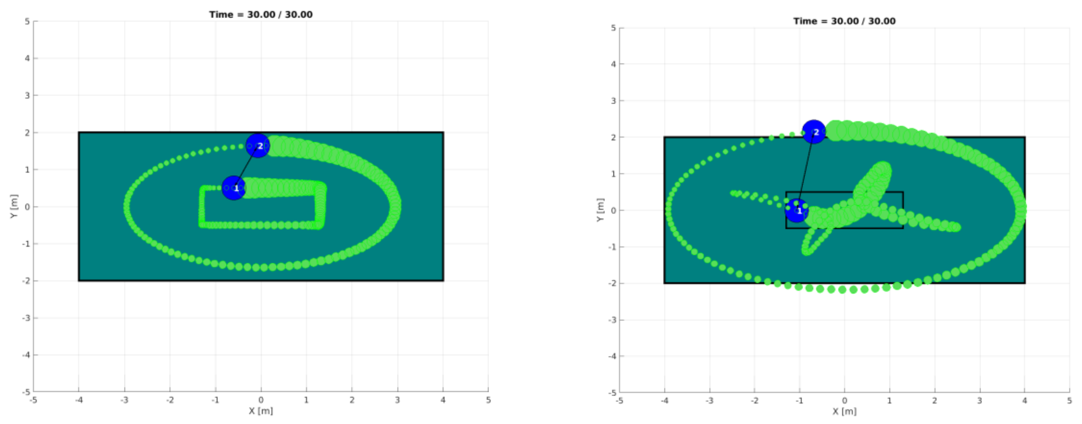


Figure 20: Simulation results for boundary avoidance with a time delay. The trajectories are plotted as position x versus position y. Agent 1, the inner trajectory, has boundary avoidance implemented while agent 2, the outer trajectory, does not. The left figure shows with no time delay, and the right figure shows with a 700 ms time delay.

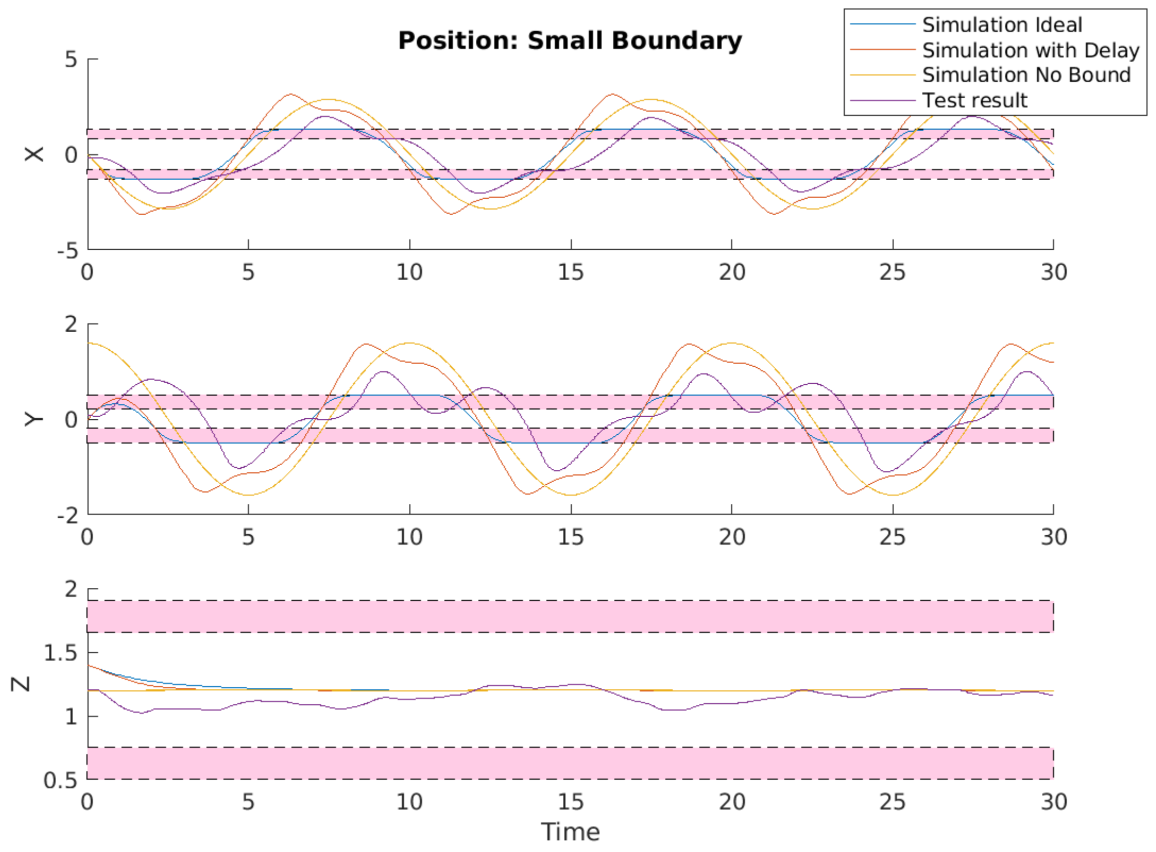


Figure 21: Comparison of simulated and tested boundary avoidance. The simulated data shows a similar trend to the test data, which indicates that the trajectory deformation is likely due to the latency. The drone dynamics were not included in the simulation, which is why the simulation with latency does not reflect all the changes in the boundary avoidance response. The shaded regions indicate the effective boundary regions for each dimension.

Figure 22 shows the xy trajectory of the agent for the tests with and without boundary avoidance, illustrating the distortions to the boundary avoidance response.

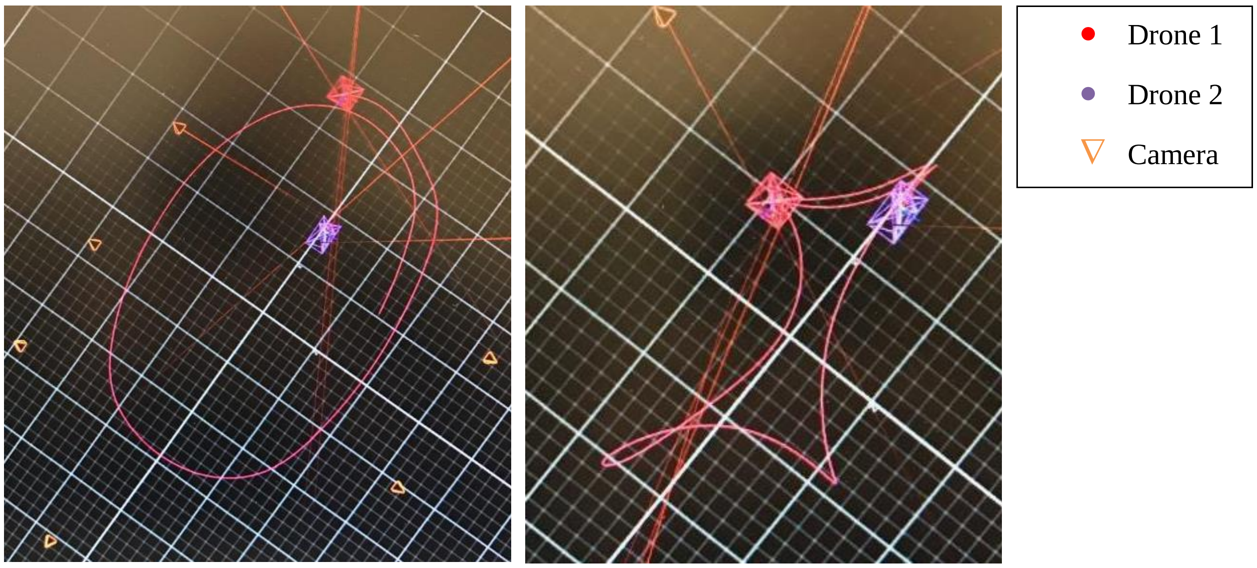


Figure 22: Image of agent trajectory captured by OptiTrack in real time during testing. *Left*: No boundary avoidance implemented. *Right*: boundary avoidance active. Not that drone 2 was not active in this test, despite its position being recorded through OptiTrack.

To overcome the difficulties associated with boundary overshoot, the boundary size was iteratively increased over a series of tests to determine appropriate dimensions to keep the agents within the safe area of the test facility. The result provided boundary avoidance that protected the agent from collisions with the walls but distorted the main signal under test in a larger region than desired. The boundary found to be effective is recorded in Table 5.

Table 5: Version 1 Boundary Values Selected for Randolph 26

Axis	**Minimum [m]**	**Maximum [m]**	**Radius of Effect [m]**
X	0.2	7.0	0.5
Y	-3.0	-0.5	0.5
Z	0.2	1.9	0.25

Non-collision testing was performed within the context of formation flying and was not tested directly. It was observed that the actuator latency and drone dynamics prevented the non-collision algorithm from adequately scaling the agents' velocities to zero when a collision was imminent, resulting in drone crashes despite the implemented safety features. Although the same approach of increasing the effective region of the algorithm as was done with boundary avoidance could have been implemented for non-collision as well, that would have caused the main algorithm to be distorted over a larger area, decreasing the ability of drone testing to illustrate how the algorithm works on a physical system. This motivated the need for a new safety feature algorithm design that was more robust to drone dynamics. The selected algorithm that met this robustness requirement is presented in the following sections.

3.2.2 Improved Projection Function with Repulsive Algorithm Version 2

For the updated algorithm, boundary avoidance and non-collision were considered to be the same objective, enabling the same algorithm to be implemented for both with minor adaptations. Specifically, boundary avoidance was simplified to be in only one dimension (orthogonal to the boundary of interest), whereas the non-collision required three-axis avoidance. In addition, boundary avoidance presented the case where the agent might be pushed outside its restricted area by disturbances, so the algorithm needed to enable the agent to return to its permitted region. Non-collision does not experience this case since the repulsion from another agent is in all directions, precluding the possibility of reaching a negative distance region.

The new algorithm operates by using a repulsive force against the barrier rather than scaling the velocity to zero. The repulsion is implemented as a term that is added to the computed velocity; the term is a scaled version of the original velocity with a magnitude taking values between zero and twice the magnitude of the original velocity. This repulsion strategy causes the velocity to be initially unmodified when the magnitude is zero and decrease to eventually move in the opposite direction as the scale factor increases, hence the agent is repulsed from the element it would otherwise collide with. The repulsion takes place along the vector between the agent and the repulsing element (i.e., the boundary or neighboring agent), reducing the impact on the original velocity since it is only modified in the direction of danger.

This algorithm is more robust to the drone dynamics and delays that hindered the previous version since it actively repels the agent from the barrier rather than simply slowing it down. This method provides the agent with stronger collision avoidance, reducing the chance of overshoot, so the boundary and radius of effect can more closely match the desired values (that is, the boundary avoidance can accommodate the actual size of the room, and the non-collision function can accommodate the desired separation distance), which in turn reduces the area in which the safety features overrule the main algorithm output. The details of the algorithm and implementation are described in the following sections.

3.2.2.1 Boundary Avoidance

The facility used in these tests provided a cuboid region in which the boundaries are orthogonal to each other, enabling computational simplification in that each axis can be considered separately to reduce to a one-dimensional problem. Cases where this assumption is not upheld such as curved boundaries are not considered in this work, but could be addressed through estimating the normal vector to the boundary and scaling each spatial dimension according to the magnitude of the projection onto the curve.

The function was designed to handle both the minimum and maximum boundaries, which required different conditions on how the velocity needed to be changed in order to ensure a repulsion and not attraction.

The user specifies whether the bound to be avoided is a maximum or minimum constraint by updating a flag to be +1 for maximum or -1 for minimum. This manages the signs of the conditions that indicate which type of avoidance should be implemented, and also controls the sign of the velocity repulsion term; otherwise the maximum and minimum cases are equivalent.

Consider the case of avoiding the minimum boundary. When the agent's position in the dimension of interest is within the variable *danger_zone* of the boundary, the repulsion function begins to take affect and becomes nonzero. The magnitude of the scaling term k_n is computed as

$$k_n = \frac{av_n}{d-s} \tag{20}$$

where a is a tunable gain, v_n is the magnitude of the velocity in the dimension of interest, d is the distance between the agent and the boundary, and s is the safety radius around the boundary. As the distance to the boundary decreases, k_n increases to infinity resulting in undesirable spikes in the velocity; to avoid this issue, the scaling term is capped at twice the magnitude of the velocity, which causes the agent to move away from the boundary at up to the same speed as it was moving toward the boundary. In this study, boundary avoidance is posed as a one-dimensional problem, so the direction of the repulsion force, k_v, is given as the direction from the boundary

$$k_v = \frac{p-b}{d} \tag{21}$$

where p is the position of the agent in the dimension of interest, b is the boundary location, and d is the distance between the agent and the boundary.

The final value to be applied to the original velocity is given as

$$k = k_n k_v \tag{22}$$

This formulation gives rise to a singularity if the distance to the boundary is equal to the safety radius s, which can occur if the agent's initial position was outside the boundary or the agent's dynamics caused it to overshoot. To avoid dividing by zero, k_n is selected to be its maximum value, v_n, and the direction is selected to be toward the interior of the boundary (computed using the *maxmin* flag) for such cases. The same approach is used if the agent is beyond the safety radius, directing the agent toward the interior of the allowable region at the maximum speed allowed. The algorithm can be implemented to compute the boundary avoidance term for a given vector to the boundary given as r as shown below.

BOUNDARY AVOIDANCE VERSION 2

```
1   **if** $|r| == 0$ or $|r| == safety\_radius$
2       ▻ at bounary, apply maximum value away from neighboring agent
3       $scale \leftarrow -2 \times |velocity| \times minmax$
4   **else if** $-maxmin \times r < safety\_zone$
5       ▻ inside safety radius, apply maximum value away
6       $scale \leftarrow -2 \times |velocity| \times minmax$
7   **else if** $|r| < danger\_zone$
7       ▻ in repulsion region, compute boundary avoidance term
8       $|scale| \leftarrow a \times |velocity| / (|r| - safety\_radius)$
9       $|scale| \leftarrow \min(|scale|, 2 \times velocity)$
10      $scale \leftarrow |scale| \times \operatorname{sign}(r)$
11  **else**
12      ▻ $x$ is in safety region
13      $scale \leftarrow 0$
```

3.2.2.2 Non-Collision

Non-collision differs from boundary avoidance in that it is posed in three dimensions instead of one, though the methodology remained the same.

When the distance between two agents becomes less than *danger_zone*, a user-defined radius that defines when non-collision is active, the magnitude of the velocity repulsion factor k_n is computed

$$k_n = min\left(\frac{av_n}{d(d-s)}, 2v_n\right) \tag{23}$$

where a is a tunable control gain, v_n is the magnitude of the velocity, d is the Euclidean distance between the two agents, and s is the safety radius around each agent which its neighbor should not cross.

The direction of the repulsion is designed to push the agents apart with reduced impact on their overall trajectories, so that only the direction of collision danger is adjusted. The reduced impact is accomplished by computing the direction k_v as the normalized vector pointing from the agent's neighbor to itself.

Similar to boundary avoidance, if the agents move closer than the safety radius s the maximum magnitude is selected to push the agents apart. The cases that cause singularity in the magnitude computation, $d = 0$ or $d = s$, also default to the maximum magnitude case. The pseudocode to accomplish the above algorithm is provided below.

```
NON-COLLISION VERSION 2
if $|r_{ij}| < danger\_zone$
    ⊳ apply noncollision within danger zone radius
    if $|r_{ij}| == 0$ or $|r_{ij}| == safety\_radius$
        ⊳ at bounary, apply maximum value away from neighboring agent
        $scale \leftarrow -2 \times velocity$
    else if $|r_{ij}| < safety\_radius$
        ⊳ inside safety radius, apply maximum value away
        $scale \leftarrow -2 \times velocity$
    else
        ⊳ in repulsion region, compute noncollision term
        $|scale| \leftarrow a \times |velocity| / (|r_{ij}| - safety\_radius) / |r_{ij}|$
        $|scale| \leftarrow \min(|scale|, 2 \times velocity)$
        $\widehat{scale} \leftarrow \min(|scale|, 2 \times velocity)$
        $scale \leftarrow |scale| \times r_{ij}$
else
    ⊳ $x$ is in safety region
    $scale \leftarrow 0$
```

3.2.2.3 Implementation

To apply the safety features to the main algorithm, the non-collision terms are computed for each neighbor in a loop, followed by the boundary avoidance terms. The safety feature terms are then summed and applied to the appropriate velocity term. An example implementation is shown in the code below.

For the purposes of this work, non-collision was not implemented in the z direction due to controllability issues with the RP line of drones that resulted in occasional unexpected vertical behavior. However, the algorithm can easily be extended to three dimensions, the implementation for which is described below:

Implementation of Safety Suite Version 2
```
1  **for** $ii \leftarrow 1$ to $|N_j|$
2          ▻ compute noncollision term for each neighbor
3          $noncolScale_x \leftarrow noncolScale_x + noncolScale_x(r_{ij}[ii])$
4          $noncolScale_y \leftarrow noncolScale_y + noncolScale_y(r_{ij}[ii])$
5          $noncolScale_z \leftarrow noncolScale_z + noncolScale_z(r_{ij}[ii])$
6  ▻ compute boundary avoidance term in each dimension
7  $boundScale_x \leftarrow boundScale_{x_{min}} + boundScale_{x_{max}}$
8  $boundScale_y \leftarrow boundScale_{y_{min}} + boundScale_{y_{max}}$
9  $boundScale_z \leftarrow boundScale_{z_{min}} + boundScale_{z_{max}}$
10 ▻ update velocity terms
11 $velocity_x \leftarrow velocity_x + boundScale_x + noncolScale_x$
12 $velocity_y \leftarrow velocity_y + boundScale_y + noncolScale_y$
13 $velocity_z \leftarrow velocity_z + boundScale_z + noncolScale_z$
```

Boundary avoidance was tested for a case when virtual drones were placed outside the test facility and the physical agents were commanded to move into a formation that would require they also leave the safe region. The results, shown below, illustrate a worst-case scenario, since most implementations do not directly command the agents to perform unsafe maneuvers. While the agent's dynamics were observed to push the agent outside the boundary region, the effect was reduced from the previous algorithm even with the defined boundary being closer to the true room dimensions. Figure 23 shows the boundary avoidance keeping the agents within the OptiTrack field of view.

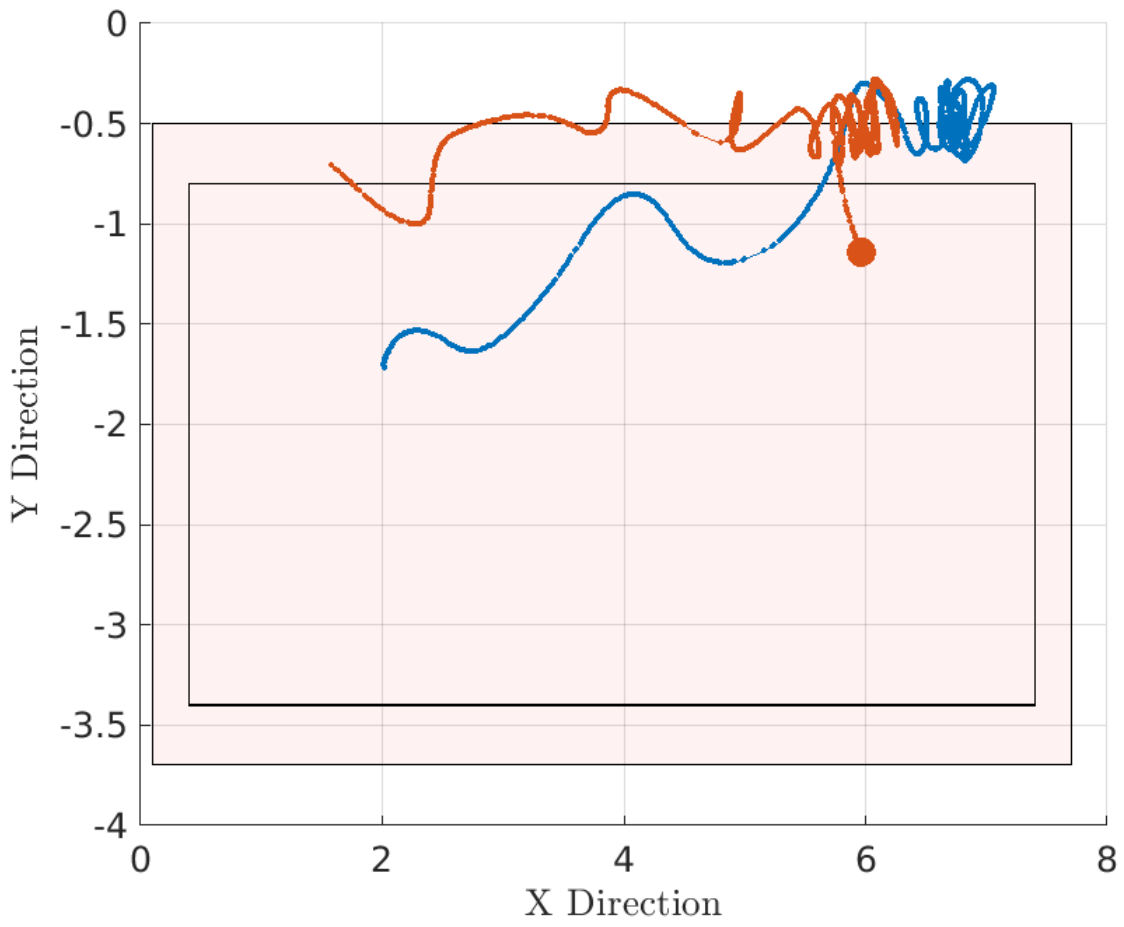


Figure 23: Boundary avoidance version 2 testing demonstration. Agent 1 (blue) and agent 2 (red) are prevented from leaving the allowable region through boundary avoidance, despite the commanded trajectory continuing in the positive y direction. This experiment showcases a worst-case example, in which agents are commanded to leave the safe region; some overshoot is observed, which is addressed by maintaining a buffer region between the room edge ($y = 0$) and the boundary edge ($y = -0.5$).

The selected boundary size for implementation is described in Table 6. The safety feature control parameters are included in Table 7.

Table 6: Version 2 Boundary Values Selected for Randolph 26

Axis	**Minimum [m]**	**Maximum [m]**	**Radius of Effect [m]**
X	0.1	7.7	0.3
Y	-3.7	-0.5	0.3
Z	0.0	1.9	0.01

Table 7: Safety Feature Parameters

Parameter	**Value**
Non-collision *a*	0.6
Non-collision danger zone	0.75
Non-collision safety radius	0.0
Boundary avoidance *a*	0.5
Boundary avoidance safety radius	0.0

3.3 Formation Flying

The initial work performed toward the objective of autonomous coordination capabilities was implementation and demonstration of formation flying. The algorithms used for distributed formation control are well studied in literature (see [16]) and provide a useful starting point for developing swarming capabilities.

Implementation of formation control among a heterogeneous swarm of agents poses a few challenges, including appropriate gain selection and information transmission among agents. The goal of this section is to demonstrate the successful operation of the distributed swarming architecture developed during this project for the SpaceDrones lab, illustrating the additional capabilities afforded by this project.

Three agents were flown in a time-varying formation that spaced them equally on the circumference of a circle which rotated at a constant angular rate. This formation was selected because it showcases the coordination of the agents as they adapt to the changing relative position requirements, maintaining their separation while simultaneously seeking to keep the centroid of the formation at the center of the room. The formation was posed as

$$F_i = [A_0 \cos\left(\frac{2\mathrm{i}\pi}{N} + \omega t\right), A_0 \sin\left(\frac{2\mathrm{i}\pi}{N} + \omega t\right), 0] \tag{24}$$

where agent i's relative position in the formation is given by the ith row of F, A_0 is the radius of the circle formed by the agents' positions, N is the total number of agents, ω is the angular rate of rotation, and t is the simulation time. To maintain the formation centroid position, a term is added to the formation controller commanding each agent to go to the center of the room; this causes the formation centroid to be located at the center of the room instead of at the centroid of the agents' initial positions.

Each agent computes its velocity onboard in real time to maintain and update its position in the formation. The results are shown in Figure 24 below, picturing four timesteps of the trajectory to demonstrate the formation's evolution over time. The formation is distorted by boundary avoidance preventing the agents from moving outside OptiTrack's field of view; for this test, the original boundary avoidance algorithm was implemented, so considerable overshoot was observed. The heterogenous nature of the swarm, using two hexarotors and one quadrotor, causes the smaller agent to adjust more readily and therefore allows the larger agents to trace larger circles; this is a challenge in implementation, since it distorts formations in ways not observed in simulation of homogenous swarms. Furthermore, the radius of the path traced by the agents is smaller than the radius of the formation because of the competing objective to move toward the center of the room. A video demonstrating the flight can be found at [41].

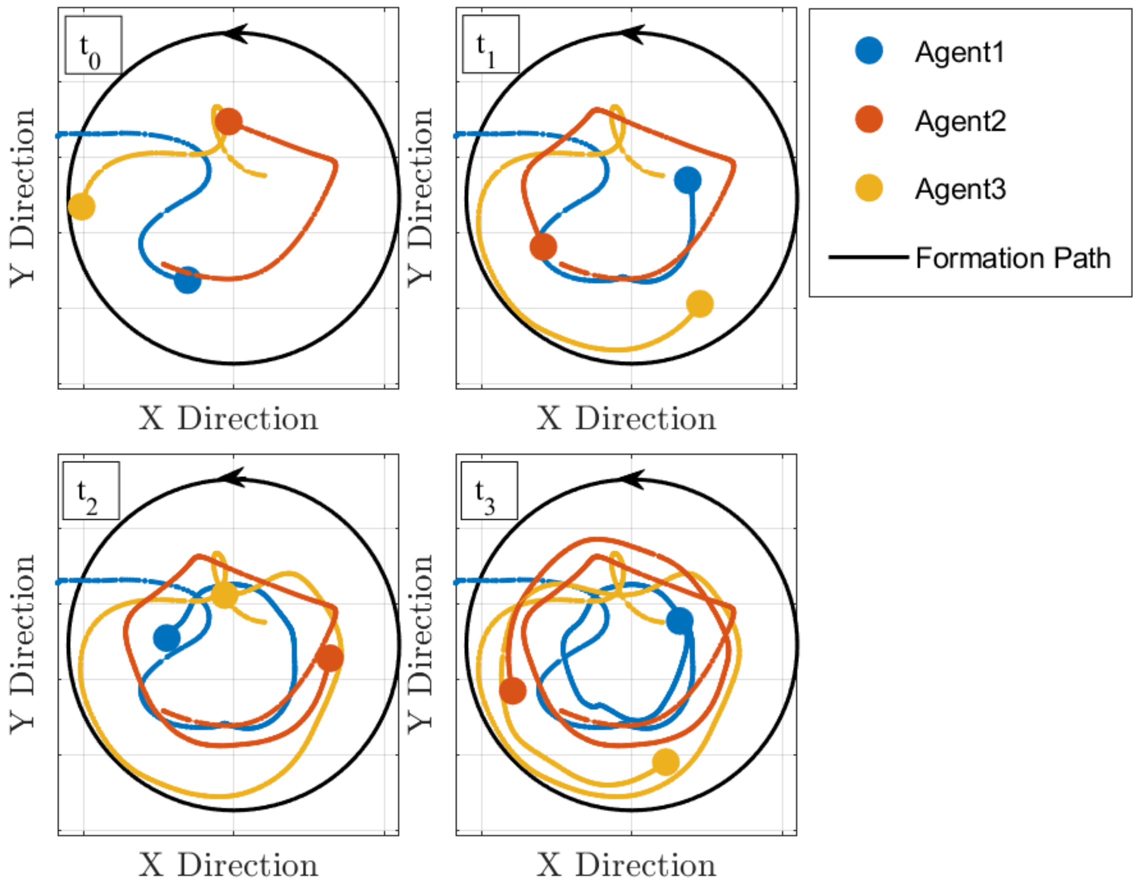


Figure 24: Demonstration of dynamic 3-drone formation flight. Trajectories at four time steps showing the formation flight of three agents using a distributed algorithm implemented in a distributed architecture at four timesteps during the test. The black circle represents the commanded formation path, which is not attained due to the competing go-to-point command and boundary avoidance.

3.4 Virtual Drones

An issue that arises with implementation of swarming algorithms is limited budget for hardware to build a large number of drones, as well as limited space in which to fly. Increasing the number of drones in the swarm increases the potential for collisions, either with a wall, ceiling, or another drone. However, it remains important to test as much functionality in as realistic environment as possible. To address the issue of realistic demonstration under hardware constraints, the physical space can be augmented with virtual elements that interact as if they were present. Specifically, virtual drones were developed that run the same software as the physical drones, publishing and subscribing to ROS topics to transmit and receive data as if they were physical drones. The positions of the virtual drones are updated using single-integrator dynamics based on the commanded velocity from the control law. Issues in scaling formations associated with communication latency and bandwidth limitations would be visible with the use of virtual drones, since their data is transmitted in the same manner as the physical drones.

A few differences in the code are required for the virtual drones to operate. First, their position is updated based on an internal equation instead of through OptiTrack since there is not a physical object being tracked. The virtual agent is therefore required to publish its own position, whereas a physical drone would rely on OptiTrack to publish that information; in future work where the agents determine their positions independent of OptiTrack, such as in the proposed LIDAR solution in the SpaceDrones lab, the physical drones would also publish their positions as the virtual drones do. Therefore, this difference is considered minor and is consistent with anticipated future adaptations.

The virtual drones do not have the pre-flight check requirements, such as verification that arming was successful, so they are able to directly move into their flight code. As such, a portion of the code used by the physical drones is commented out or removed, replaced by hard coded Booleans.

The virtual agents run on the Raspberry Pi and can operate in parallel with the flight code for the physical drones. Of key importance is that virtual drones run on the same equipment as physical drones in order to make them as close to their physical counterparts as possible.

A limitation with the virtual drones is that their current configuration causes them to move directly to the “experiment” portion of the code and does not include take-off and landing. These features were not required for the research presented in this thesis; however, adding the capability to simulate all phases of the flight would have the potential to benefit future projects by more closely resembling physical drones.

In addition to enabling larger formations, the virtual drones have the potential to be used as simulation tools prior to testing on actual hardware. Their primary advantage in this regard is their complete interfacing with ROS, enabling potential issues to be detected at an early stage. See Appendix B for a description of how to convert physical drone code to act as a virtual drone.

3.5 Target Location through Gradient Estimation

The first portion of the unknown environment exploration algorithm to be implemented on the physical hardware was target location through gradient estimation when no obstacles were present. The purpose of this test was to demonstrate the successful operation of the distributed target location procedure, and to illustrate that the agents behaved in a similar manner to the simulated response. Measurement memory was not incorporated in the hardware testing phase. The target seeking algorithm was also simulated with the same configuration, and the results are shown in Figure 25.

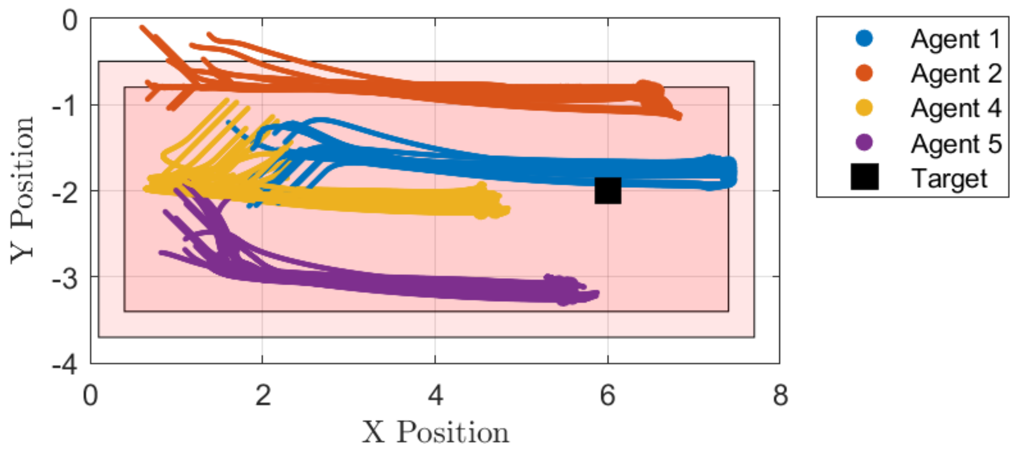


Figure 25: Simulation results for gradient estimation without obstacles. The agents consistently located the target under perturbations to the initial conditions, while remaining within the safe operating area even when initial conditions were outside the safety region.

Two physical drones were used, M1 and RP1 as agents 1 and 2, respectively, alongside two virtual agents, labeled 4 and 5. It was observed that this swarm configuration was able to successfully locate the target for each trial run, as shown in Figure 26. The formation centroid is shown to converge to the target location in Figure 27.

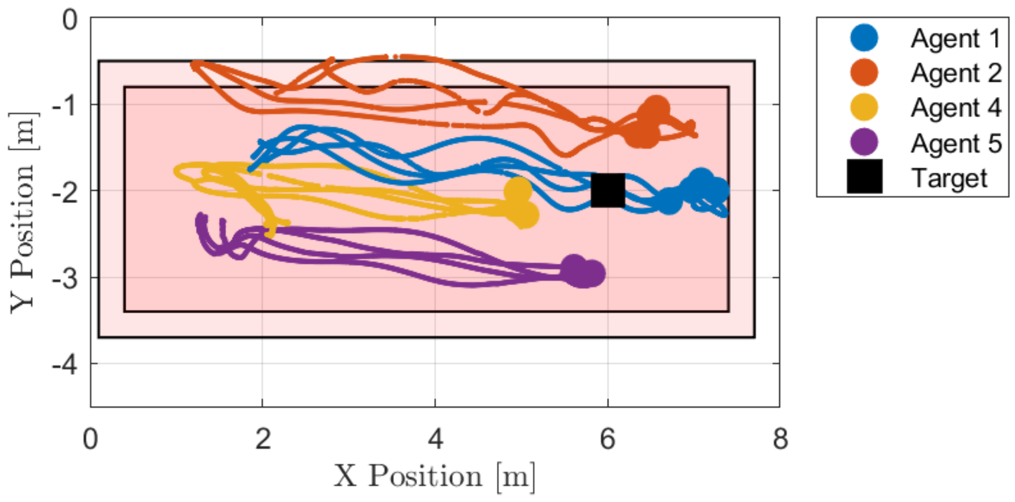


Figure 26: Testing results for target location without obstacles. The formation is maintained throughout target locating. The agents follow similar paths over different trials, demonstrating the repeatability of the algorithm.

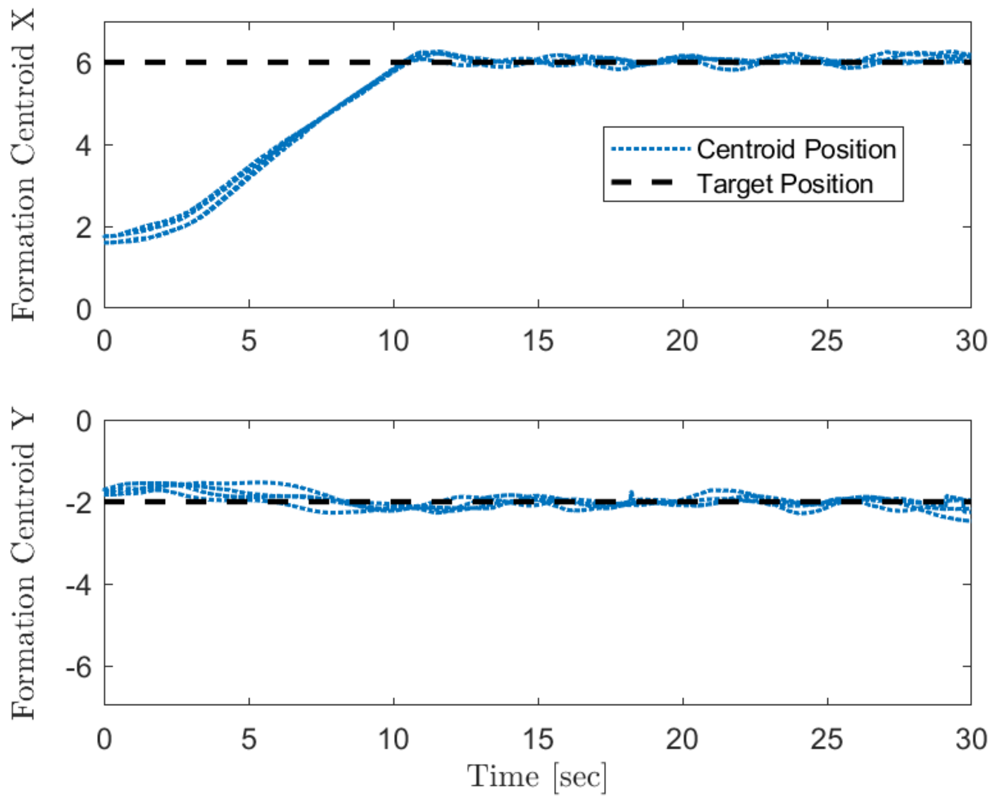


Figure 27: Formation centroid for target locating test with no obstacles. The centroid of the formation converges to the target location (the dashed black line) as the agents follow their collective estimate of the gradient for all tests performed for the case in which no obstacles were present.

3.6 Obstacle avoidance

The two obstacle avoidance algorithms were implemented using the same setup as was used for target seeking alone in Section 3.5. Three configurations of obstacles, shown in Figure 28, were used to validate the algorithm and demonstrate that the simulated results could be reproduced using the SpaceDrones hardware. These included a single obstacle test to demonstrate the canonical setup most frequently discussed in literature, in which the obstacle lies between the swarm's initial position and the target; a two obstacle test in which the obstacles are close to each other to demonstrate how the algorithms respond to multiple obstacles simultaneously in an agent's field of view; and a three obstacle test to demonstrate the response to a bottleneck configuration in which the agents can move between obstacles to reach the target. The purpose of these tests was to demonstrate the core functionality of the algorithms and illustrate their operation. To simplify the implementation, obstacle barrier rotation was not addressed in the hardware testing phase. Note that the physical drones, agents 1 and 2, were restricted to the safe operating region using boundary avoidance, while the virtual drones, agents 3 and 4, were not restricted.

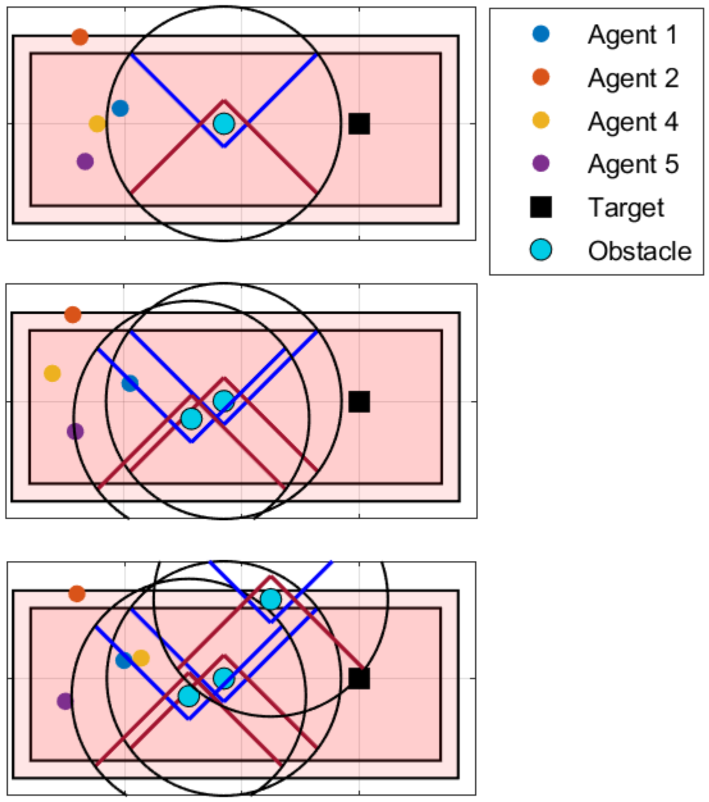


Figure 28: Testing configurations for demonstration on SpaceDrones hardware, featuring one, two, and three obstacles. The drones were initially clustered at the opposite end of the test facility as the target, with the obstacles located in between.

For comparison between theory and implementation, a low order Monte Carlo perturbation study was performed in simulation, in which the initial positions of agents, target, and obstacles were uniformly randomly selected from a small region about their expected location. This simulation approach illustrated how the agents' trajectories may vary with small changes to the initial setup, revealing the stable paths the agents were expected to take in practice.

Testing revealed the limitations of the algorithms, which are outlined in their respective sections below. Cases demonstrating the limitations of algorithm 1 are discussed in Appendix C. Videos of the demonstrations can be found at [42] and [43].

3.6.1 Algorithm 1

3.6.1.1 Test Configuration 1

The first test of algorithm 1 was the single obstacle test, designed to demonstrate the core functionality of the algorithm by placing the obstacle such that the agents were forced to decide which direction to select to navigate around it. The simulated trajectories are shown in Figure 29.

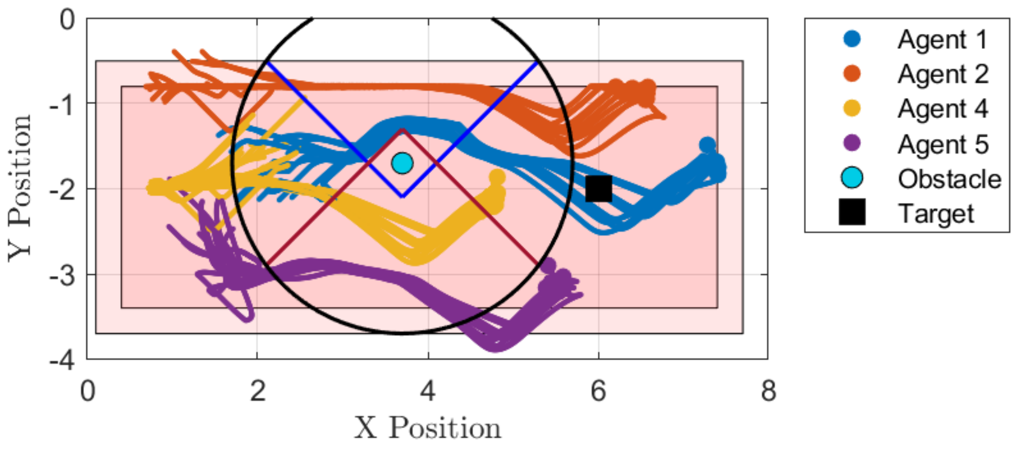


Figure 29: Simulation results for algorithm 1 with one obstacle. The agents follow a stable trajectory under perturbations to the initial conditions. The final agent positions center the formation on the target. The safe operating region, shown as the inner red box, constrains agents 1 and 2 but not 3 and 4.

The experimental trajectories are shown in Figure 30. It was observed that the agents followed similar paths to those predicted by simulation, though the physical drones experienced more erratic behavior when navigating past the obstacle due to the delay between the commanded velocity and the actuated velocity. In one test, agent 5 collided with the obstacle due to insufficient formation resolution as described in Section 2.3.7; this case is further addressed in Appendix C.

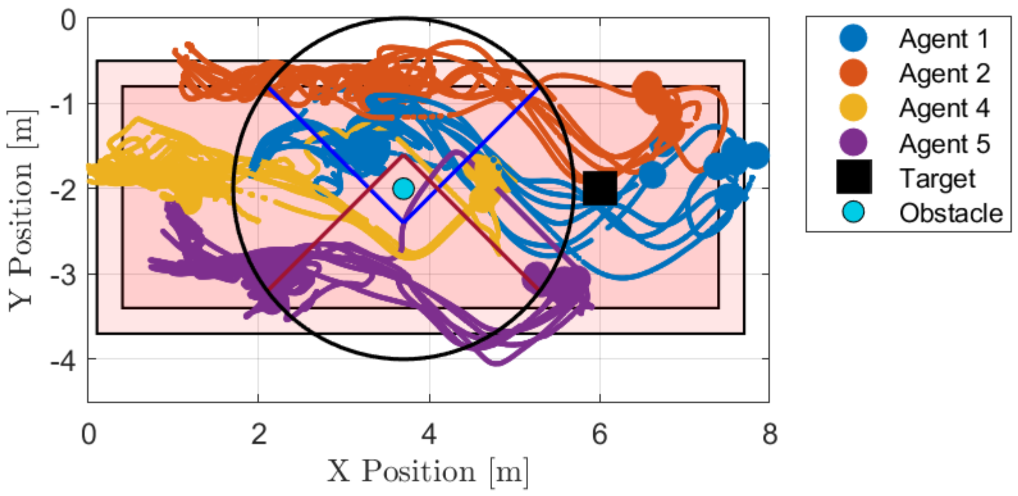


Figure 30: Algorithm 1, test configuration 1 - single obstacle between swarm and source. The agents navigate around the obstacle to achieve a formation centered on the target. One case shows a collision with the obstacle when the augmented gradient did not direct the agent in the correct direction. The safe operating region, shown as the inner red box, constrains agents 1 and 2 but not 3 and 4.

In implementation, the swarm was able to navigate to the target consistently, with one case where a bottleneck prevented the agents from moving past the obstacle. The centroid of the formation over time is shown in Figure 31, illustrating how the agents converge to the target. Prior to properly tuning the collision avoidance parameters, the agents were observed to struggle to locate the target. This mode appears as the case in Figure 31 where the x dimension of the centroid does not converge to the target's position and is discussed in the following section.

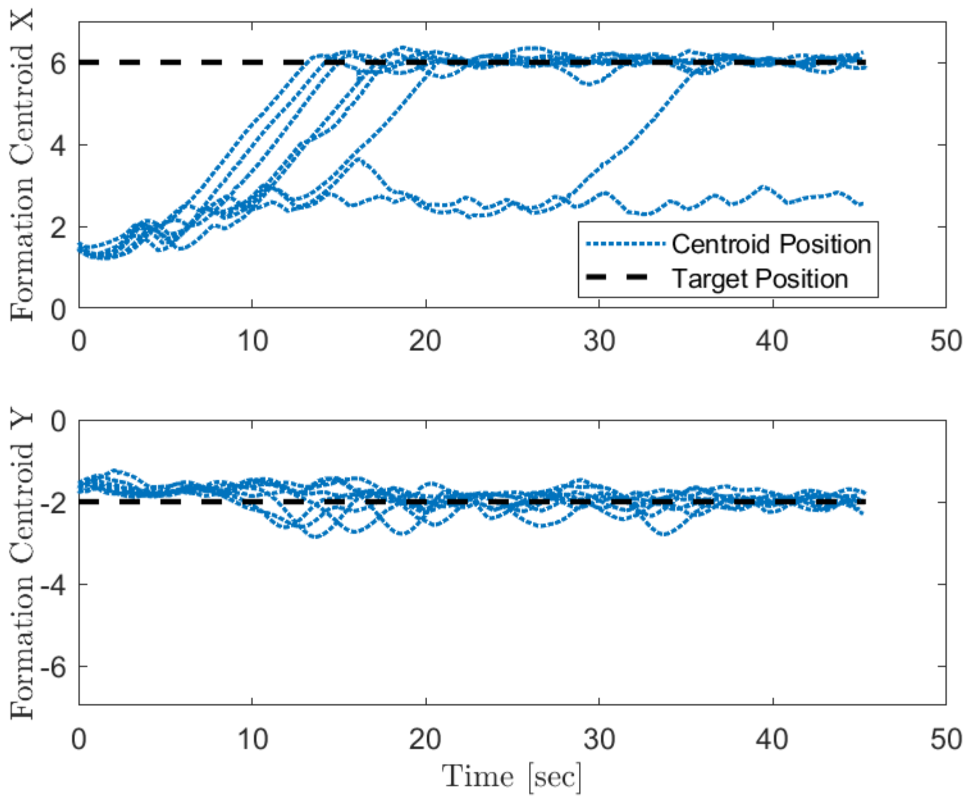


Figure 31: Formation centroid for target locating with algorithm 1, configuration 1. The formation centroid location in X and Y is plotted over time, converging to the target (dashed black line) as the target is located. Note that in one case the target was not located.

Implementation and simulation both experienced cases where the agents were not able to navigate past the obstacle to locate the source; the experimental trajectory of such a case is shown in Figure 32. The issue arises when agents 1 and 2 (blue and red, respectively) become trapped between the wall and obstacle. In order to prioritize the safety of the physical agents, non-collision and boundary avoidance were designed to overrule the target seeking and obstacle avoidance algorithms. This decision of algorithm hierarchy results in competing objectives that can end in a stalemate where the agents oscillate between goals.

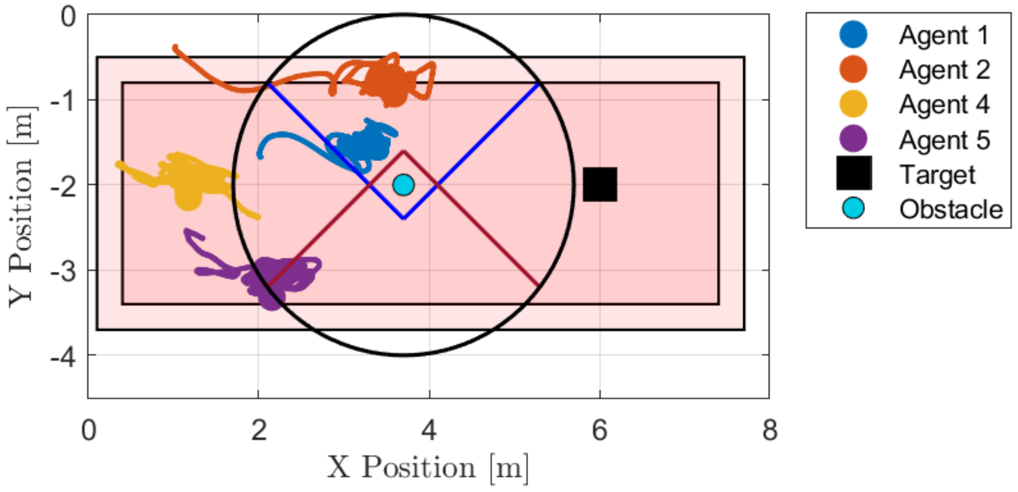


Figure 32: Algorithm 1, configuration 1 trajectory of a case where the agents did not successfully locate the target. Agents 1 and 2 (blue and red, respectively) encountered a bottleneck imposed by the obstacle and room boundary that prevented convergence to the target.

In the above case, agent 1 comes into contact with the obstacle and is repulsed in order to avoid a collision. However, when it moves away from the obstacle, it draws too close to the neighboring agent 2, pushing it back in the direction from which it came. Agent 2 is constrained by the boundary of the room, so it also assumes this oscillatory movement. Additionally, while agent 1 is free of the formation control, agent 2 is not within the obstacle repulsion region and therefore is subject to maintaining the formation, which is why agent 2 does not move forward beyond the position of agent 1. The bottleneck effect causes the agents to continuing oscillating as they are forced away from their respective boundaries and toward each other and are then repelled from each other to prevent collision. This error was corrected by reducing the effective range of the non-collision algorithm to allow the drones to pass more closely to one another, and by increasing the formation size to reduce the chance that non-collision will be necessary, thereby enabling the drones to pass through the bottleneck in a sequential fashion. Figure 33 illustrates the conflict resulting in non-convergence to the target.

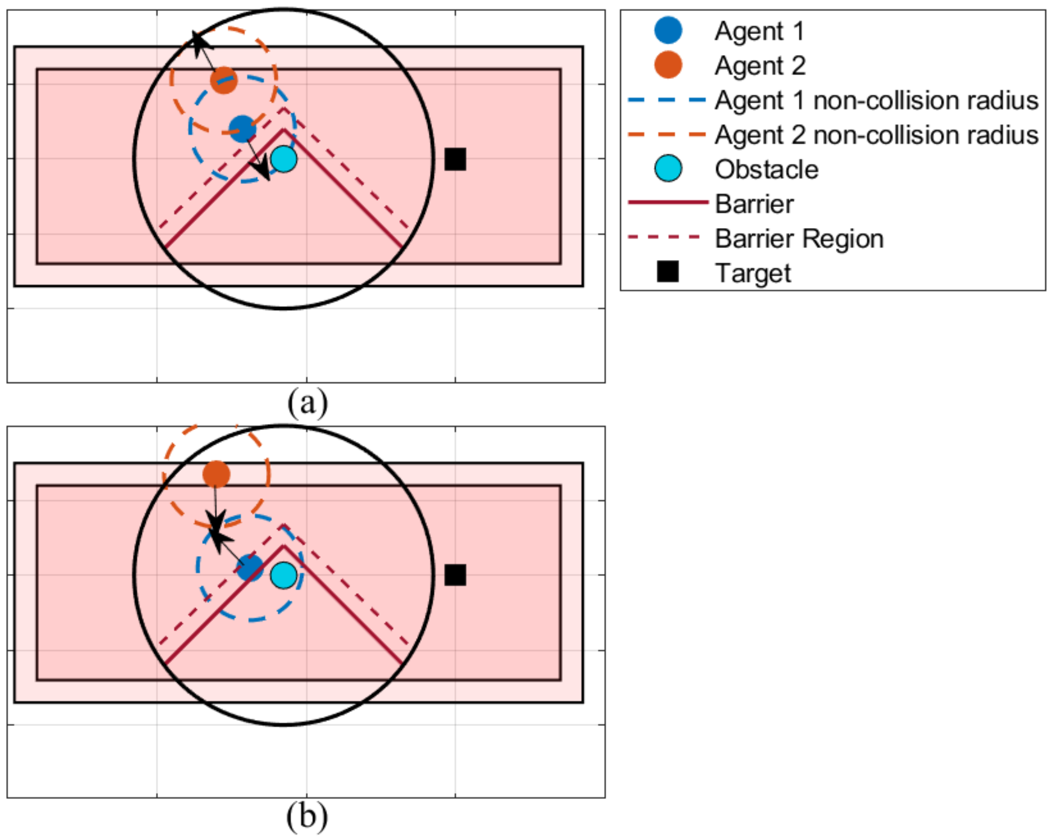


Figure 33: Diagram depicting the conflict between non-collision and barrier avoidance. Non-collision, shown in (a), forces the agents away from each other, while boundary/obstacle avoidance, shown in (b), force the agents back toward each other. The resulting bottleneck prevents the agents from navigating past the obstacle.

3.6.1.2 Test Configuration 2

The second test configuration included another obstacle located close to the first, ensuring that the agents would encounter both and apply obstacle avoidance concurrently. This case also demonstrates the algorithmic error that motivated the development of obstacle avoidance algorithm 2, in which the low resolution of the large formation caused an agent to collide with an obstacle. Simulation results are presented in Figure 34, with the experimental trajectories shown in Figure 35.

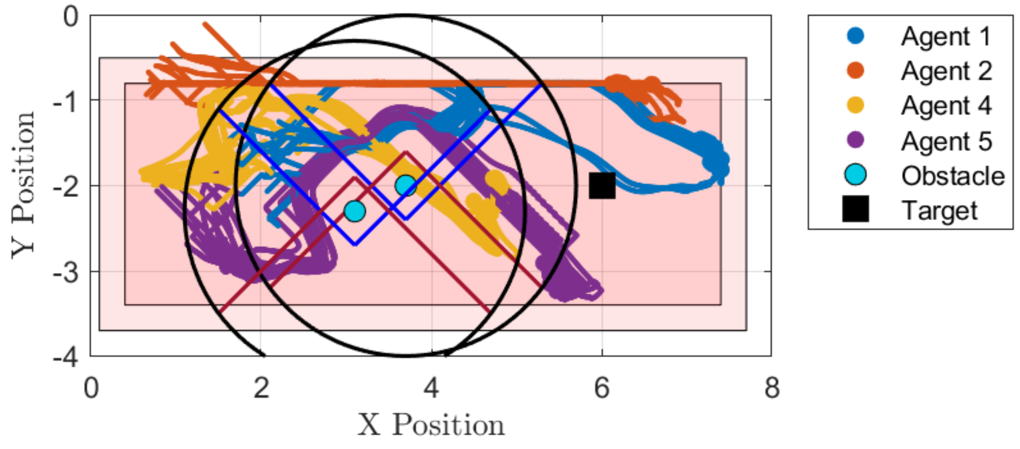


Figure 34: Simulation results for algorithm 1 with two obstacles. The stable path taken by agents under perturbations highlights the algorithmic limitations of this obstacle avoidance approach.

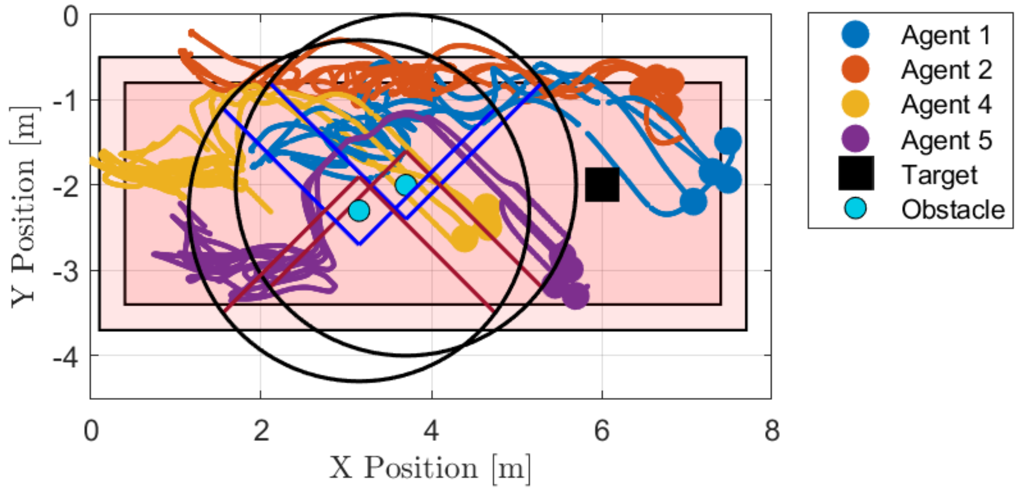


Figure 35: Algorithm 1, test configuration 2 - two obstacles between swarm and source. The experimental results show the paths taken by each agent, resulting in agent 4 consistently colliding with an obstacle due to the formation resolution limitations.

While the agents were not able to successfully avoid the obstacles in this configuration, they did converge to the target as desired. Figure 36 shows the centroid converging to the target location for each test.

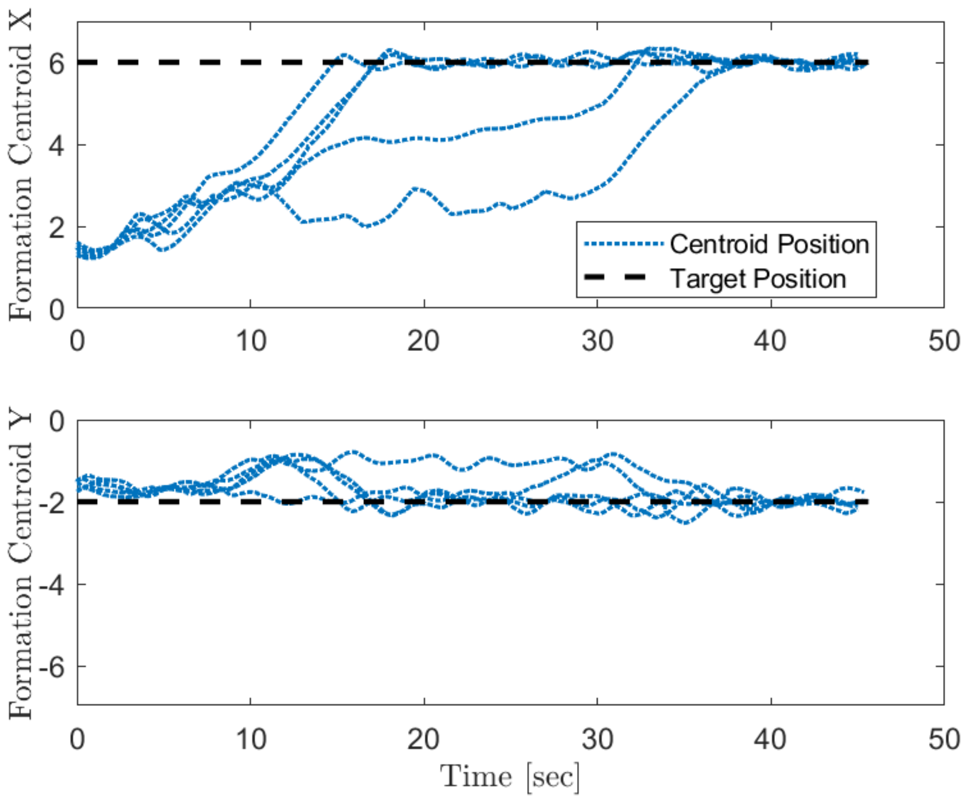


Figure 36: Formation centroid for target locating with algorithm 1, configuration 2. The formation centroid location for each test in x and y is plotted over time, converging to the target (dashed black line) as the target is located when two obstacles were present.

The algorithmic limitation discussed in Section 2.3.7 is clearly observable in this test data. In this case, agent 4 consistently collided with an obstacle in both simulation and testing. The failure stems from the formation size being larger than the obstacle, causing the swarm to have insufficient resolution to properly estimate the location of the barrier from the augmented gradient for obstacle avoidance. Test results also highlighted the importance of formation size and shape, since the perceived gradient vector field for each agent changed over the course of the test as the agents moved in and out of formation. A video of the time-varying gradient estimate using the formation data from a physical test case is recorded in [44]. The effect of the formation shape on the gradient estimate is shown in Figure 37 for agent 4, depicting four times during the test where different formations were formed; the gradient field was computed for the formation shown in each subplot, representing the gradient estimate of agent 4 (shown in green). The initial positions of the agents gave each agent a perspective on the gradient field that changed when the formation was first obtained. The relaxation of formation control during obstacle avoidance resulted in a distorted formation that again changed the perceived gradient. The final panel illustrates the successful convergence of the formation to the target.

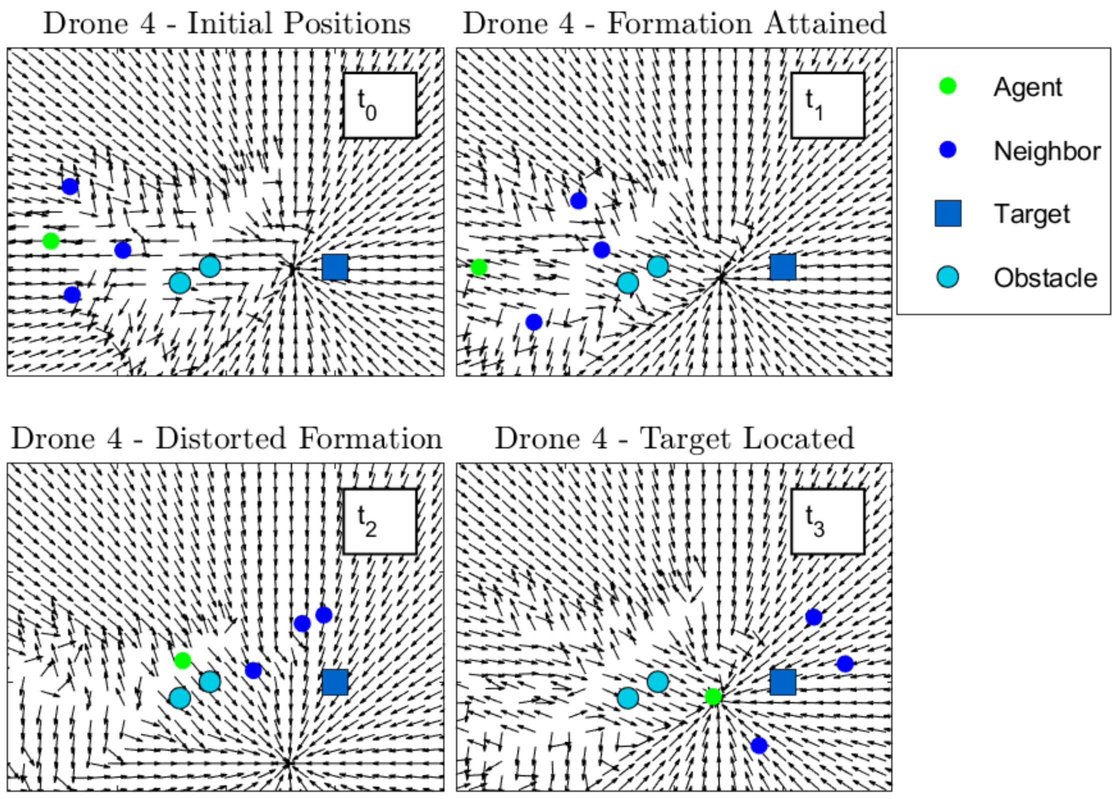


Figure 37: Effect of formation shape on gradient estimation. The time varying gradient field estimate for agent 4 (green) for different relative locations of the remain agents (blue) is shown in four panels. The four formations were selected from test data for the two-obstacle configuration, illustrating how variance in the formation affects the perceived augmented signal gradient. At times t_1 and t_3 the same formation is achieved, resulting in the same ghost obstacle positions; the perceived obstacle locations are shifted by the different formation configurations at times t_0 and t_2. The movement of the ghost obstacles shows the limited resolution of the large formation, resulting in agent 4 colliding with the obstacle.

3.6.1.3 Test Configuration 3

The third test configuration introduced an obstacle in the path of agent 2, creating a bottleneck where the agents had to pass between two obstacles. This configuration demonstrates the use of multiple obstacle avoidance modes. The simulation is shown in Figure 38, illustrating how this more complex environment causes multiple trajectories to occur from small perturbations to initial conditions.

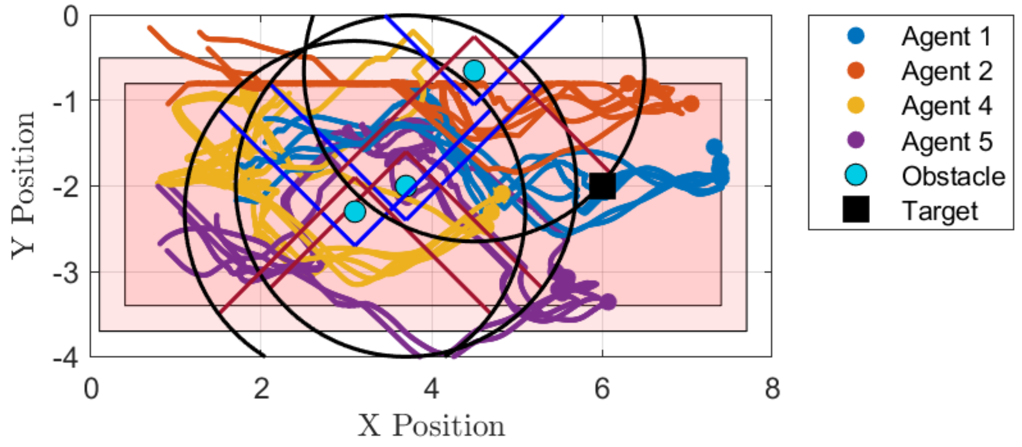


Figure 38: Simulation results for algorithm 1 with three obstacles. Agents 4 and 5 are able to move outside the safe operating region because they are virtual drones and therefore have boundary avoidance disabled.

The testing trajectories are shown in Figure 39. While the simulation data indicated multiple stable trajectories for this configuration, the physical drones tended to take the same path for the different test iterations. The physical drones required longer to move through the bottleneck than in simulation, so the virtual drones were biased toward the path below the obstacles to better maintain the formation; this delay in the physical system resulted in only one of the stable trajectories being observed during testing.

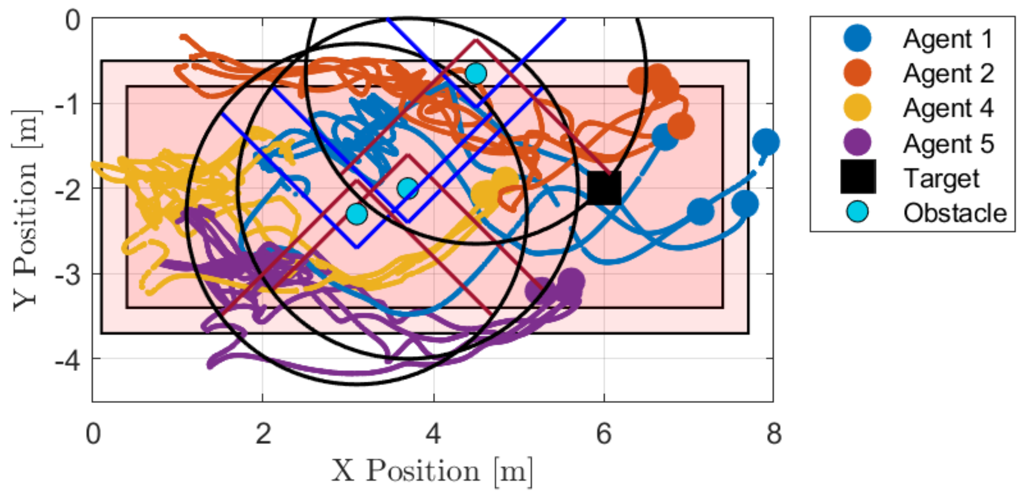


Figure 39: Algorithm 1, test configuration 3 - three obstacles between swarm and source. Agents converging to the target in a complex environment.

All tests successfully located the target, as illustrated in Figure 40. Competing objectives between obstacle avoidance and non-collision caused some test cases to take longer to converge, but all were ultimately successful in locating the target.

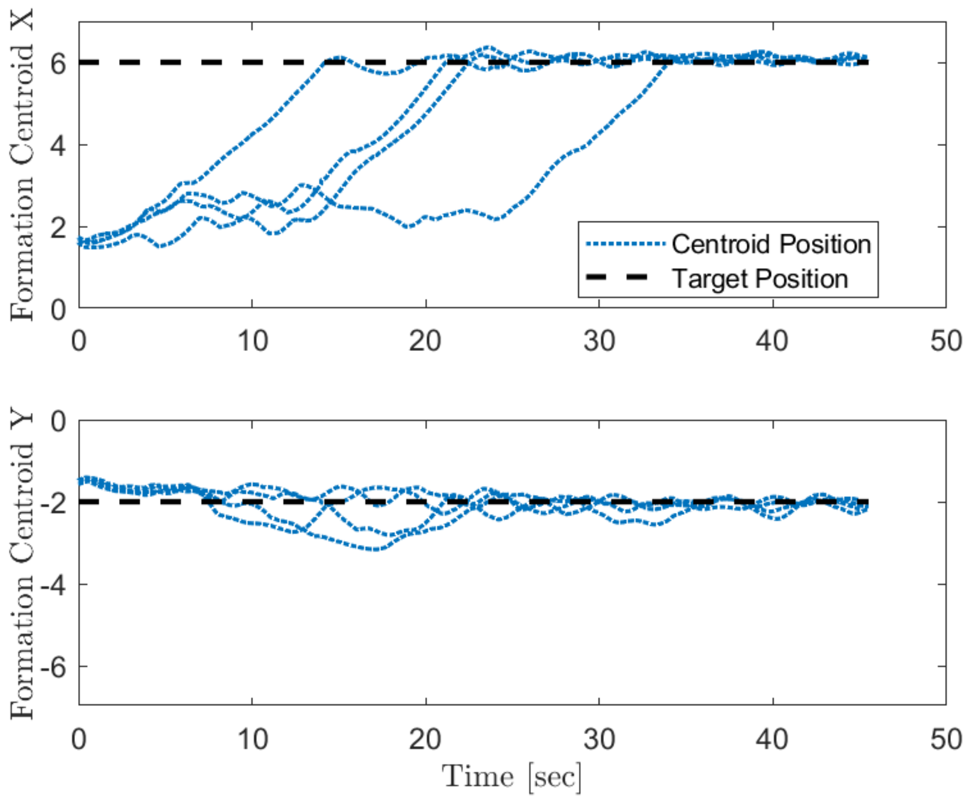


Figure 40: Formation centroid for target locating with algorithm 1, configuration 3. The formation centroid location in x and y are plotted over time, converging to the target (dashed black line) as the target is located when three obstacles were present.

3.6.2 Algorithm 2

The second obstacle avoidance algorithm was tested in the SpaceDrones lab to demonstrate the improved performance for obstacle avoidance when the formation resolution is insufficient to accurately locate the obstacles in the distributed fashion of algorithm 1. While the tests conducted were not identical to those performed on algorithm 1 in that the target was moved from [6,-2] to [6,-1.35], the similarity between simulation and experimental results indicate that the cases not tested would also behave in their simulated manners.

3.6.2.1 Test Configuration 2

This configuration is similar to the case that illustrated the limitations of algorithm 1, and thereby showcases the advantage of algorithm 2 in that the obstacles were successfully avoided by all agents both in simulation (see Figure 41) and experimentation (see Figure 42). The use of local information for obstacle avoidance removed the resolution issues found in the aggregate method of algorithm 1, enabling the agents to robustly avoid both obstacles.

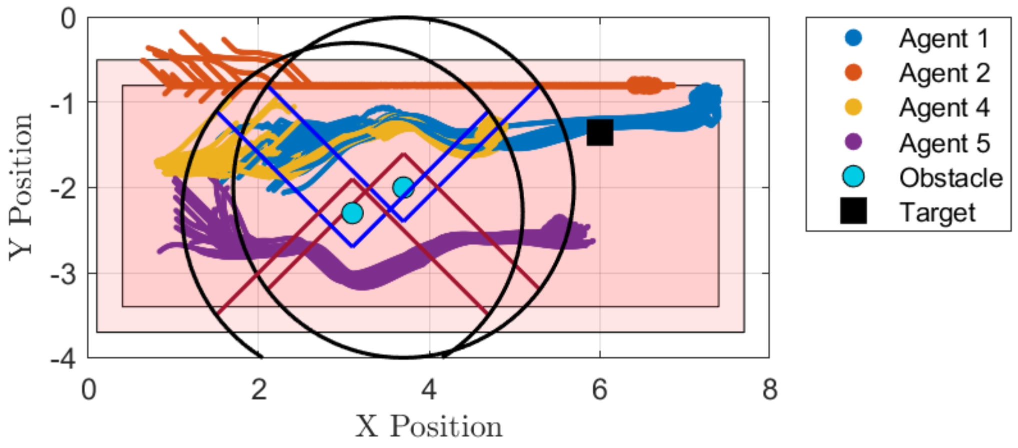


Figure 41: Simulation results for algorithm 2 with two obstacles. The stable path taken by agents under perturbations to the initial conditions do not have the formation resolution problem exhibited in algorithm 1 with a similar configuration. Agents 1 and 2 are limited in converging to their desired formation positions due to boundary avoidance as they near the edge of the safe operating area.

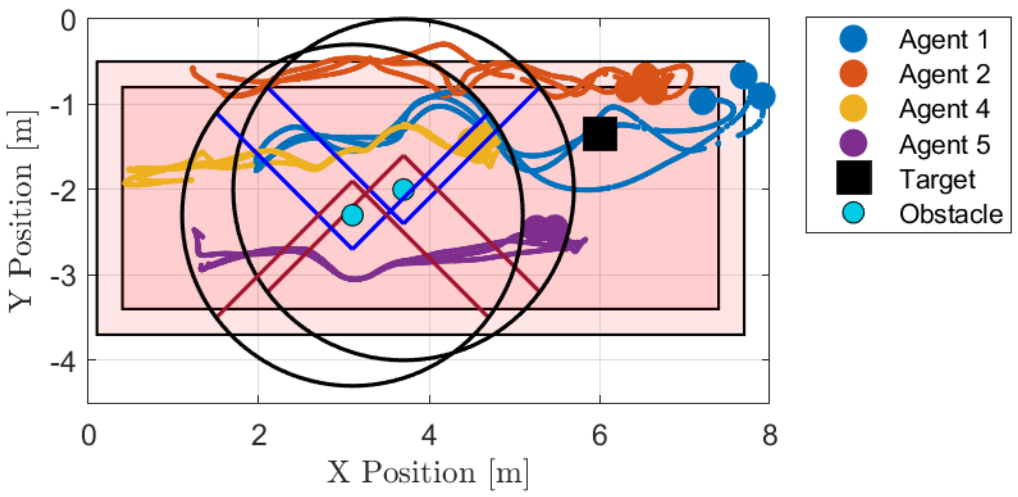


Figure 42: Algorithm 2, test configuration 2 – two obstacles between target and agents. The trajectories of the agents as they avoid obstacles while locating the target for the implementation of algorithm 2 are illustrated.

Each test case was observed to successfully locate the target, as shown in Figure 43.

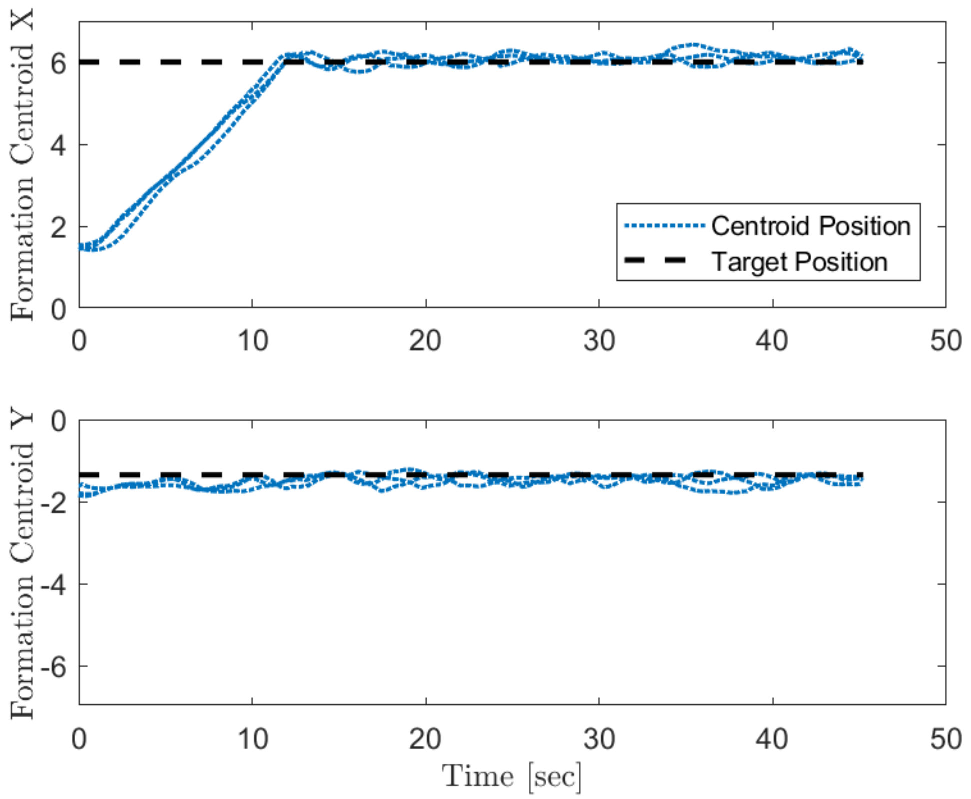


Figure 43: Formation centroid for target locating with algorithm 2, configuration 2. Convergence of centroid to target location.

3.6.2.2 Test Configuration 3

The three-obstacle configuration was simulated (see Figure 44) and tested (see Figure 45) to show how algorithm 2 responded to the path limitations and increased environmental complexity introduced by the third obstacle. It was observed that RP1 (agent 2) was less responsive to commands than it had been in previous tests, possibly due to a change in hardware without recalibration, such as the replacement of rotor blades. The result was that although the algorithm correctly computed the appropriate direction of travel, the agent was less likely to comply with the commanded velocity. Recommendations on how to mitigate issues caused by less stable flight are discussed below.

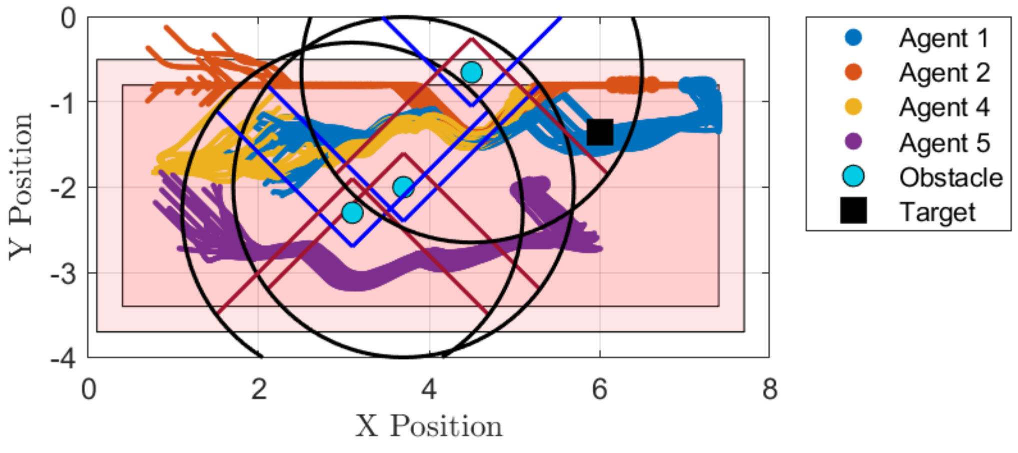


Figure 44: Simulation results for algorithm 2 with three obstacles. It can be seen that there is a stable path that the agents are most likely to follow under small perturbations to the initial conditions. Agents 1 and 2 are limited in converging to their desired formation positions due to boundary avoidance as they near the edge of the safe operating area.

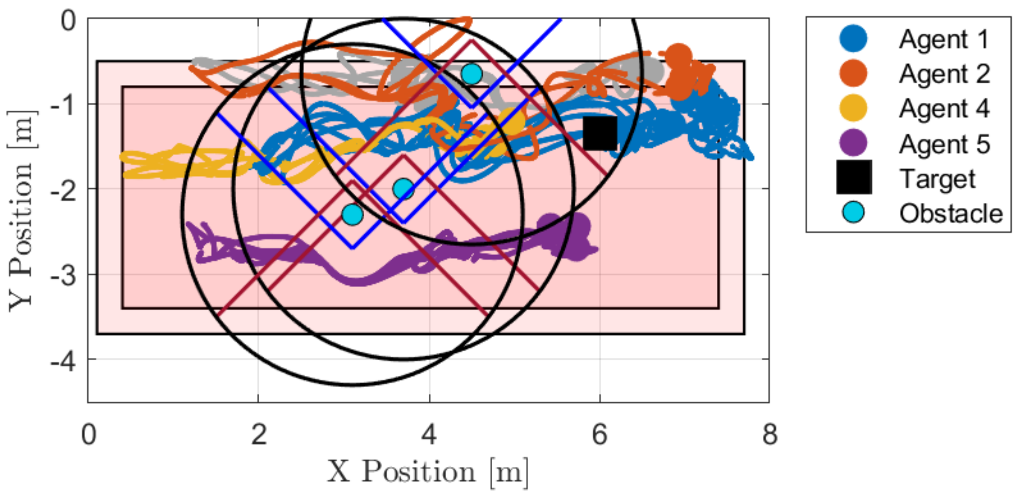


Figure 45: SpaceDrones testing results for algorithm 2 with three obstacles. The successful tests are shown here, in which the physical drones behaved in a manner consistent with their commanded velocities. Gray lines illustrate tests in which the instability of agent 2 caused it to not respond in the commanded manner, instead colliding with an obstacle.

It was observed in Figure 46 that the agents successfully located the target; the delay in one test was caused by agent 2 selecting to go to the left of the obstacle before boundary avoidance forced it to retrace its movement and instead navigate around the obstacle by going right.

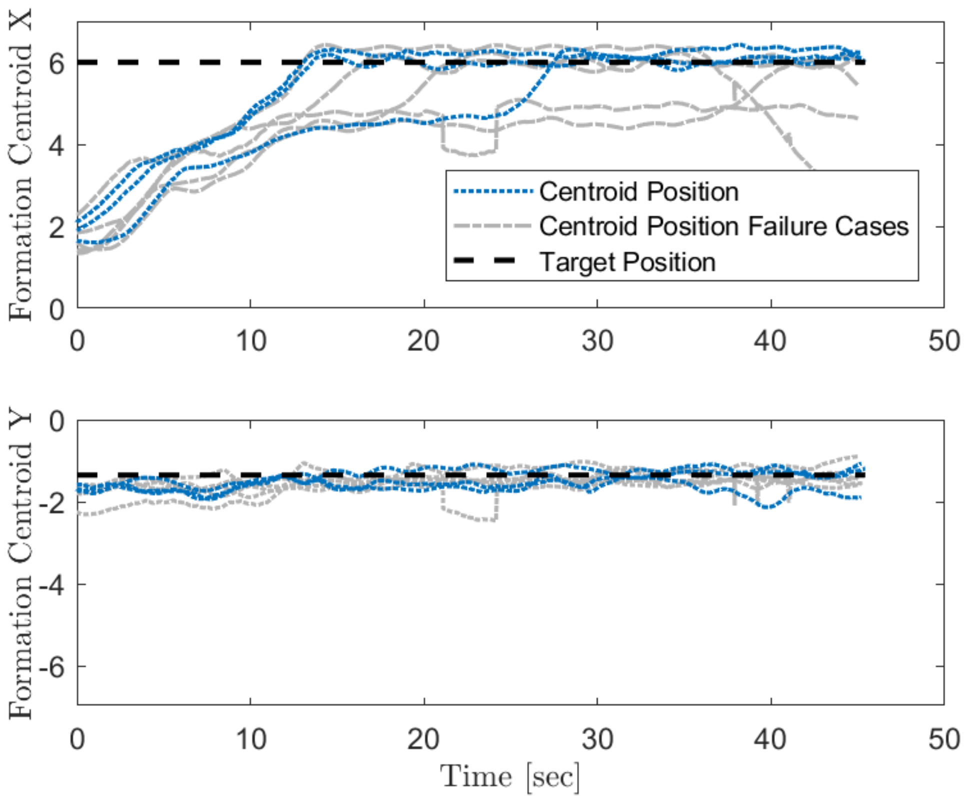


Figure 46: Formation centroid for target locating with algorithm 1, configuration 1. Convergence of centroid to target location is shown for cases in which agent 2 responded well (blue) and nonconvergence of the centroid is shown for cases in which the agent did not follow the commanded trajectory (gray).

There were a few cases in which the new algorithm did not successfully navigate around the three obstacles. This scenario occurred when the agents spent a lot of time deciding how to go through the bottleneck, and it appears that agent 2 was slightly pushed into the obstacle by its inertial dynamics since the discontinuity of the algorithm caused the drone to respond in a delayed manner. Figure 47 shows the trajectories of the agents for cases where agent 2 collided with an obstacle.

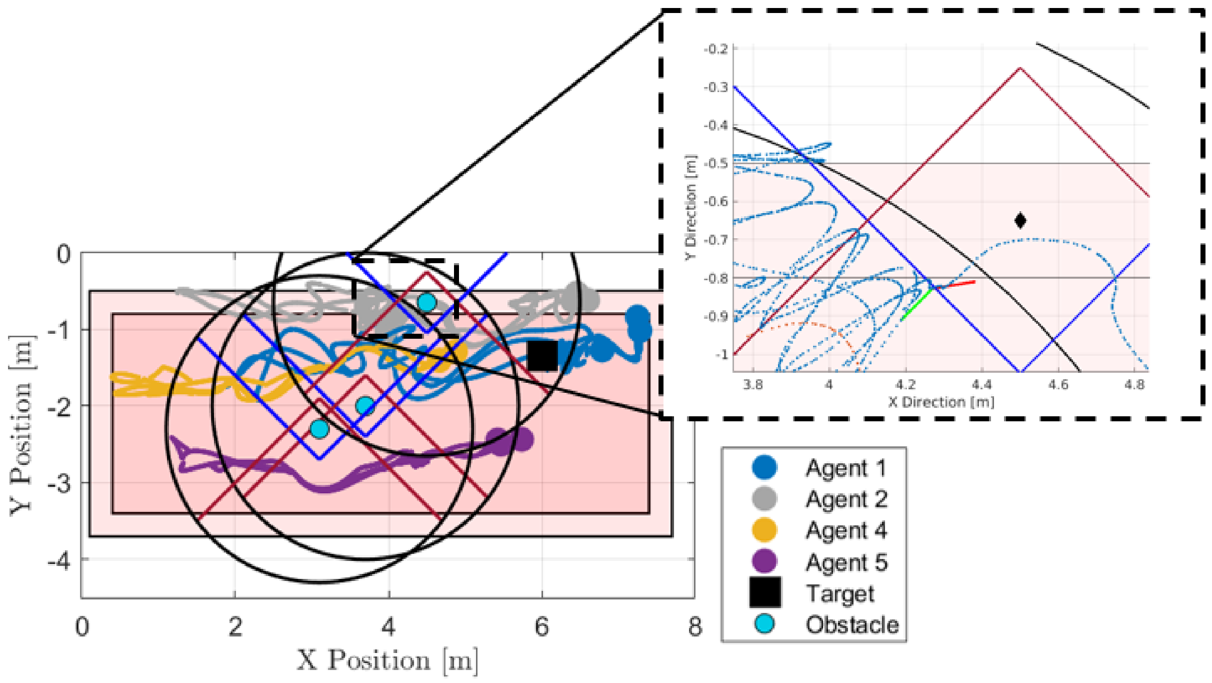


Figure 47: Obstacle collisions due to dynamic inertia. *Left*: Trajectories for cases where the dynamics of the drone did not follow the commanded velocity, causing a collision with the obstacle. *Right*: Agent 2, shown as the blue trajectory, crosses the barrier into the obstacle keep-out zone. The green arrow illustrates the commanded velocity at that time, showing that the algorithm directs the agent away from the obstacle. The red arrow represents the measured velocity, which underscores how the inertial dynamics of the drone play a role in the implementation which is not seen in simulation. This is the cause of the failure to avoid the obstacle in these cases.

The algorithm was found to compute the correct direction of travel to avoid the obstacle; however, the method used introduced discontinuities in the controller that produced an actuation delay since the true response of the drone is continuous. The comparison of the commanded and measured velocity for agent 2 is shown in Figure 48; it is recommended that future iterations of the code address the ranges of velocity values to reduce the discrete nature of the output velocity in order to make the algorithm more robust to drone dynamics and disturbances. Alternatively, more precise hardware could be used for the drones, such as implemented on M1 (agent 1).

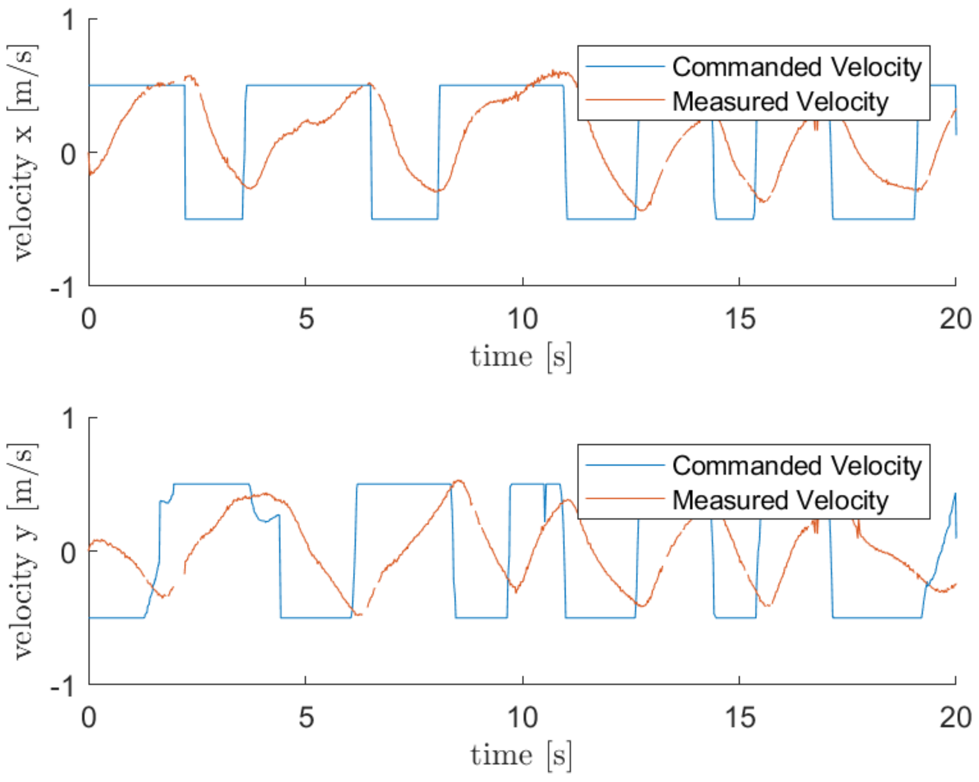


Figure 48: Comparison of commanded and measured velocities. The algorithm is acting as more of a bang-bang controller; the discontinuous controller causes the dynamics to lag the commanded velocity, allowing the agent to drift into the obstacle.

3.6.3 Comparison of Results

The experimental results obtained through implementation of both algorithms confirmed the effectiveness of obstacle avoidance algorithm 2 to successfully navigate through unknown environments where algorithm 1 experienced limitations. Both algorithms responded as expected based on simulation, consistently locating the target and meeting simulated obstacle avoidance performance expectations. It was observed that both algorithms exhibited some sensitivity to unmodeled disturbances such as ground effect and wind generated by the other agent; however, the tests demonstrated that the algorithms can be used in a realistic setup on a physical system to complete the desired target localization mission.

4 Conclusions and Future Work

This project addressed the problem of locating a target in a cluttered unknown environment using measurements of a concave signal emitted by the target coupled with robust obstacle avoidance. It was shown how a swarm of UAVs could leverage their local measurements of the signal to estimate the gradient in order to select the appropriate direction in which to travel to navigate toward the target; a distributed least-squares approach was used to compute the gradient estimate, augmenting the approach found in literature to more effectively address cases where communication loss or topological collinearity resulted in insufficient information to perform the computations. Spatial diversity robustness was accomplished by storing previous values of the signal measurement to diversify the spatial gradient information. In addition, an alternative method of gradient normalization was proposed that enabled agents to maintain constant speed when searching for the target, yet slow to a stop when the target is located.

Obstacle avoidance was approached from two algorithms: initially, a fully distributed method was implemented to augment the signal measurement so the gradient curved around obstacles; to overcome limitations resulting from the formation resolution of the distributed approach, a second algorithm was developed that repelled the agent from the obstacle, relying primarily on local information. Both methods incorporated hybrid control modes to enable the obstacle avoidance strategy to be robust to exogenous inputs that would otherwise have the potential to cause the agents to misidentify the location of the target.

This work introduced dynamic obstacle detection to the non-collision scheme discussed in literature through the use of radially bounded modes. When an obstacle is outside the sensor range of the agent, a collision is not imminent and therefore active avoidance is unnecessary; this motivates the use of a base mode that turns obstacle avoidance off by default and transitions to active mode when an obstacle has been detected. This approach allows for dynamic obstacle detection since the controller can add obstacle avoidance terms in real time as new obstacles are detected.

In order to demonstrate that the algorithms operated effectively on realistic hardware, the algorithms were written in Python and implemented on the VT SpaceDrones setup described in [28]. The first step for implementation on physical drones was to enable formation flight for multiple drones. A time-varying formation where the drones maintained equidistance on a rotating circle about a common goal point was demonstrated using 3 drones, showcasing the scalability of the implementation. In this formation control work, the x and y positions of the drones were controlled in a distributed manner; the z position was not included in the formation since altitude deconfliction was used as a safety precaution. The formations themselves were designed for N drones, but the flying area in the lab space and the available hardware limited testing to a maximum of 3 drones.

Increased number of drones creates potential for more accidents, so safety features were developed accordingly. Basic versions of boundary and collision avoidance were implemented, in which the velocity of the drone is reduced as it approaches an obstacle until it stops at a safe distance; however, actuator delays caused this approach to collision avoidance to be less reliable, motivating the need for further work in this area. A second algorithm was developed to actively repel agents from potential collisions, which was found to be more robust to latency issues. These safety features will continue to be of use for future projects beyond the scope of this work.

The algorithms developed in this work for target locating in an unknown environment were demonstrated on the SpaceDrones hardware, including results from obstacle configurations using 0, 1, 2, and 3 obstacles. The limitations of the first obstacle avoidance algorithm were investigated, and the ability of the second obstacle avoidance algorithm to overcome those limitations was demonstrated.

The work completed here has components that will be applicable to future projects, including testing a suite of safety features for drone flying, building a code base for multi-drone formations and collaboration, enabling augmented reality through virtual drones, and designing a simulation environment compatible with the SpaceDrones architecture.

Several areas for future work to extend the capabilities and validation of the proposed algorithms are presented below, addressing the scope limitations presented in Section 1.1.

The implementation used to demonstrate the algorithms showcased the core functionality of obstacle avoidance with target locating but did not include all the features developed in this work. To continue this work, it is recommended that the full algorithm be implemented on hardware, including obstacle orientation and signal measurement memory. In addition, environments with varying obstacle sizes could be tested, using an adaptive obstacle virtual box size that adjusts based on the cross-section size of the detected obstacle. Adjustable virtual box size would be a straightforward way to extend the method described in this work to more diverse environments that more closely resemble what a swarm would encounter in an actual use case.

The obstacle avoidance algorithm could be extended to more obstacle shapes and configurations by enabling closely spaced obstacles to be combined into a single element, forming a wall of obstacles that the agents would travel around. This increased capability would be particularly advantageous in cases where obstacles form a concave wall, in which the current algorithm could become trapped in the resulting local minimum.

Additional work could be conducted to improve the modeling of the signal emitted by the target. In particular, physical processes would likely result in time-varying signals as disturbances such as wind play a role. Obstacles also can affect the contour of the signal, either blocking as a wall might block a Wi-Fi signal or distorting as a pillar might alter a chemical plume. These types of disturbances were not modeled in the simulation or implementation test cases, so additional study in this area would be useful as a sensitivity test to determine the robustness of the algorithms to changes to the underlying problem assumptions.

For implementation, more realistic environments would be useful for testing. The tests used to validate the algorithms used a simulated target and signal mapping; equipping the drones with sensors to measure a signal directly would be beneficial since it would reveal potential limitations based on measurement noise and latency. The sensor would be application dependent, but a beacon or Wi-Fi approach would likely be accessible for preliminary work.

To further improve the testing environment, transitioning away from OptiTrack and moving toward an onboard position measurement system would be advantageous. The algorithms presented in this work rely on relative positions, so the absolute position of each agent and detected obstacle is unnecessary. Instead, a LIDAR system could be implemented to enable direct measurements in environments where OptiTrack is not available such as an outdoor space, thereby preparing the swarm for use in practical settings.

Appendix A: Guide to Multi-Agent SpaceDrones Implementation

To perform the tests described in this work, conduct the setup described in [27] for two drones. For each drone, open two terminals and SSH into the Raspberry Pi interface; one will be used to run the physical drone code, enabling flight of the corresponding drone, while the other will be for the virtual drone implementation, enabling both versions to be run on the same hardware.

Each of the two drones has a copy of the code onboard that is shared between the physical drone and virtual drone. To select the test, open the parameter file *Mohr_params_vX.py* and edit the “test_select” variable. Test 1 is the boundary avoidance demonstration, Test 2 is the multi-agent dynamic formation test, and Test 3 is the target seeking with obstacle avoidance experiment. The variable “agent_select” should be edited to include all the agents that will be flown in a particular test, typically [1,2,4,5] for the demonstrations presented in this work. Target and obstacle placement are selected from a list of configurations found in *my_pid_Mohr_vX.py*. For algorithm 1, use *my_pid_Mohr_V10.py*; for algorithm 2, use *my_pid_Mohr_V11.py*.

To begin the experiment, create a *SpaceDrone* object in each terminal to initialize two virtual drones and two physical drones by entering

```
rosrun mavros <filename>  <target address>
```

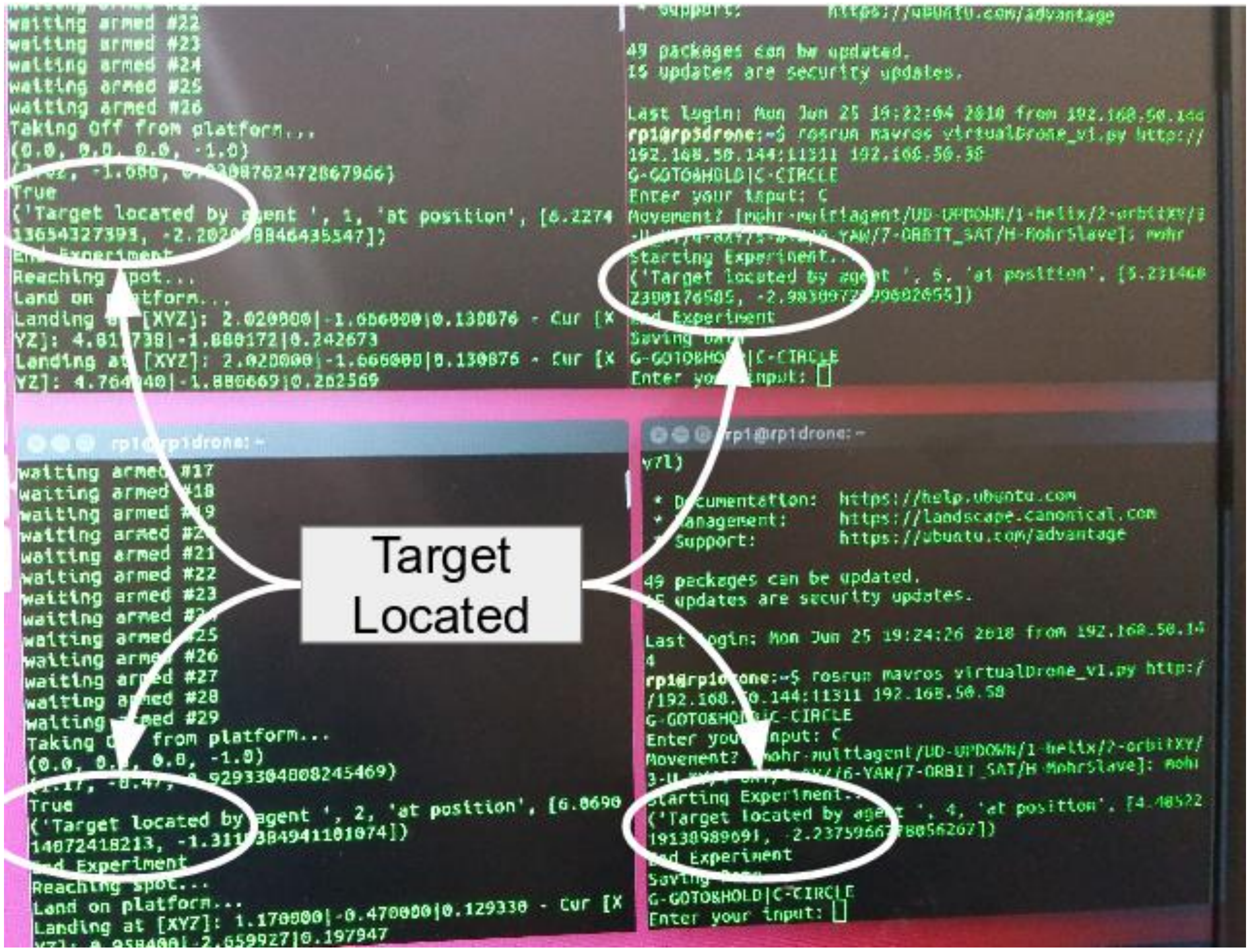
The filename is *mohr_vX.py* for the physical drones and *virtualDrone_vX.py* for the virtual drones. The target address is the address for the drone on which the code is to be executed.

Each terminal will display a prompt, allowing the user to select between GOTOHOLD and

CIRCLE. Type ‘C’ to select the circle function, in which the target locating and formation flying tests are located. The next prompt will request the movement type, for which the user should enter ‘mohr’ to access the swarming experiments. It is important to begin the experiment in each terminal at the same time (or as close together as possible) since the internal experiment clock begins when the movement type is selected.

To ensure the flight data and test parameter data are saved, place the two physical drones back on their starting positions before powering off. In most cases, the drones should return to their proper landing positions without user intervention.

Both algorithms were equipped with a function to print to the console when an obstacle was hit by an agent, including the agent’s position at which the error occurred, and when the agent had successfully located the target. An example of the output of a successful test is shown in Figure 49.



```
waiting armed #22
waiting armed #23
waiting armed #24
waiting armed #25
waiting armed #26
Taking Off from platform...
(0.0, 0.0, 0.0, -1.0)
([illegible], -1.666, [illegible]008762472867966)
True
('Target located by agent ', 1, 'at position', [6.2274
13654327393, -2.202[illegible]8846435547])
[illegible] Experiment
Reaching spot...
Land on platform...
Landing at [XYZ]: 2.020000|-1.666000|0.130876 - Cur [X
YZ]: 4.81[illegible]738|-1.880172|0.242673
Landing at [XYZ]: 2.020000|-1.666000|0.130876 - Cur [X
YZ]: 4.764[illegible]40|-1.880669|0.262569
```

```
* Support:          https://ubuntu.com/advantage

49 packages can be updated.
15 updates are security updates.

Last login: Mon Jun 25 19:22:04 2018 from 192.168.50.144
rp1@rp3drone:~$ rosrun mavros virtualDrone_v1.py http://
192.168.50.144:11311 192.168.50.58
G-GOTO&HOLD|C-CIRCLE
Enter your input: C
Movement? [mohr-multiagent/UD-UPDOWN/1-helix/2-orbitXY/3
-U[illegible]-YAW/7-ORBIT_SAT/H-MohrSlave]: mohr
Starting Experiment...
('Target located by agent ', 5, 'at position', [5.231468
2380176585, -2.983097[illegible]99602655])
End Experiment
Saving [illegible]
G-GOTO&HO[illegible]|C-CIRCLE
Enter yo[illegible]input: 
```

```
rp1@rp1drone: ~
waiting armed #17
waiting armed #18
waiting armed #19
waiting armed #20
waiting armed #21
waiting armed #22
waiting armed #23
waiting armed #24
waiting armed #25
waiting armed #26
waiting armed #27
waiting armed #28
waiting armed #29
Taking Off from platform...
(0.0, 0.[illegible], 0.0, -1.0)
(1.17, -0.47, [illegible]0.9293304808245469)
True
('Target located by agent ', 2, 'at position', [6.0690
14072418213, -1.311[illegible]384941101074])
End Experiment
Reaching spot...
Land on platform...
Landing at [XYZ]: 1.170000|-0.470000|0.129330 - Cur [X
YZ]: 0.958400|-2.659927|0.197947
```

```
rp1@rp1drone: ~
v7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

49 packages can be updated.
15 updates are security updates.

Last login: Mon Jun 25 19:24:26 2018 from 192.168.50.14
4
rp1@rp1drone:~$ rosrun mavros virtualDrone_v1.py http:/
/192.168.50.144:11311 192.168.50.58
G-GOTO&HOLD|C-CIRCLE
Enter your input: C
Movement? [mohr-multiagent/UD-UPDOWN/1-helix/2-orbitXY/
3-U[illegible]/6-YAW/7-ORBIT_SAT/H-MohrSlave]: mohr
Starting Experiment...
('Target located by agent ', 4, 'at position', [4.48522
19138989691, -2.2375966[illegible]8056267])
End Experiment
Saving [illegible]
G-GOTO&HOLD|C-CIRCLE
Enter your input: 
```

Figure 49: Output to terminal displaying the successful target location by each agent. Each agent runs on its own terminal.

Appendix B: Virtual Drone Creation

Virtual drones are constructed to run the code used by physical drones, with a few adaptations. The code is designed to have minimal changes between virtual and physical drones to enable the virtual drones to better approximate the behavior of physical drones.

In this project, the code was structured such that the control laws and test parameters were called by the main function in *mohr_vX.py*, and all adaptations to the code were confined to this script. In the general SpaceDrones experiment setup, this function is often referred to as *arm.py* and defines the *SpaceDrone* class, and the filename is typically updated with the project or experimenter name. For the virtual drone, the file is instead saved as *virtualDrone_vX.py*.

The position of the virtual drone is not broadcast to its neighbors by OptiTrack since it does not have a physical position to be tracked; instead, the virtual drone broadcasts its position itself, publishing as a ROS topic. This is the same approach that would be used if a physical drone was detecting its own position through LIDAR, GPS, or computer vision, then transmitting that location to neighboring drones. The required code is discussed below.

The position variable must be initialized as the correct pose type in Python. This is done inside the class initialization, using the imported type *PoseStamped()* to set up the position class as shown below.

```
# initialize in def __init__(self)
self.ps       = PoseStamped()
self.pos     = self.ps.pose.position
```

The publisher needs to be initialized, which is done in the custom initialization function *init()*.

```
# in init(self,nodeName)
# Publish the virtual position
self.set_pos_pub = rospy.Publisher(self.folderID + 'mocap/pose', PoseStamped, queue_size=10)
```

The data is transmitted after the position has been updated in the function *moveCircle()*, implemented as follows:

```
#in moveCircle(self)
#Transmit position
self.ps.pose.position = self.pos
self.ps.header.stamp = rospy.Time.now()
self.mocap_clock = self.ps.header.stamp
self.set_pos_pub.publish(self.ps)
```

Other subscribe/publish items are disabled for the virtual drone since they are not relevant. These items to be removed or commented out include

```
# other subscribe/publish items
rospy.wait_for_service(self.folderID + 'set_mode')  #Wait until the service is available
self.set_mode_srv = rospy.ServiceProxy(self.folderID + 'set_mode', SetMode)  #Making a service
self.set_savt_pidTwist = rospy.Publisher(self.folderID + 'spaceATVT/pidTwist', TwistStamped, queue_size=10)
self.set_pose_pub = rospy.Publisher(self.folderID + 'setpoint_position/local', PoseStamped, queue_size=10)
self.set_vel_pub = rospy.Publisher(self.folderID + 'setpoint_velocity/cmd_vel', TwistStamped, queue_size=10)
self.set_thrust_pub = rospy.Publisher(self.folderID + 'setpoint_attitude/thrust', Thrust, queue_size=10)
self.set_rate_pub = rospy.Publisher(self.folderID + 'setpoint_attitude/cmd_vel', TwistStamped, queue_size=10)
rospy.Subscriber(self.folderID + "setpoint_raw/local", PositionTarget , self.pose_sp_callback) # View setpoint
rospy.Subscriber(self.folderID + "setpoint_raw/attitude", AttitudeTarget , self.att_sp_callback)
```

The drone dynamics are modeled as a single integrator point object, since the PixHawk PID controller is not included in the virtual drone architecture. The position is propagated using commanded velocity applied for the duration of the time step (*pid.ts*). The code used to update the agent position is provided below, and its position in the code is immediately after the velocity is computed from the controller function and before the position is transmitted via ROS topic. The commanded velocity is not transmitted to the PixHawk since that is not an applicable action for virtual agents.

```
###Virtual Drone Dynamics
self.pos.x = self.pos.x+velX*pid.ts
self.pos.y = self.pos.y+velY*pid.ts
self.pos.z = self.pos.z+velZ*pid.ts
```

Takeoff is not simulated by the virtual drone, so the variable ‘takeoff’ in the *moveCircle* function is hardcoded to **True**. The flag to conduct the experiment is also hardcoded to **True** since the virtual agent does not have to wait for takeoff to complete. Instead, the initial position is set to the appropriate value provided by the user, and the experiment is initiated. The flag indicating that the agent is in offboard mode is set to **True** since the virtual agent does not have a manual control option. The timestep size is set in the *initialTest()* function, which only includes the rate setting portion since the checks for whether the drone is armed do not apply to virtual drones.

Appendix C: Algorithm 1 Failure Mode Implementation Details

The limitations of Algorithm 1 were illustrated in several test cases during the implementation phase. The list of failures observed during testing are included here for completeness. For each case, a figure of the failure trajectory and an explanation of the cause are presented.

Ghost Obstacles

The majority of the failures were the result of the algorithmic error discussed in Sections 2.3.7 and 3.6.1.2. The gradient estimate failed to properly locate the obstacle barrier since the distributed formation did not have high enough resolution compared to the obstacle size, allowing agents to move into the keep-out zone around the obstacle. The observed failure trajectories are shown in Figure 50.

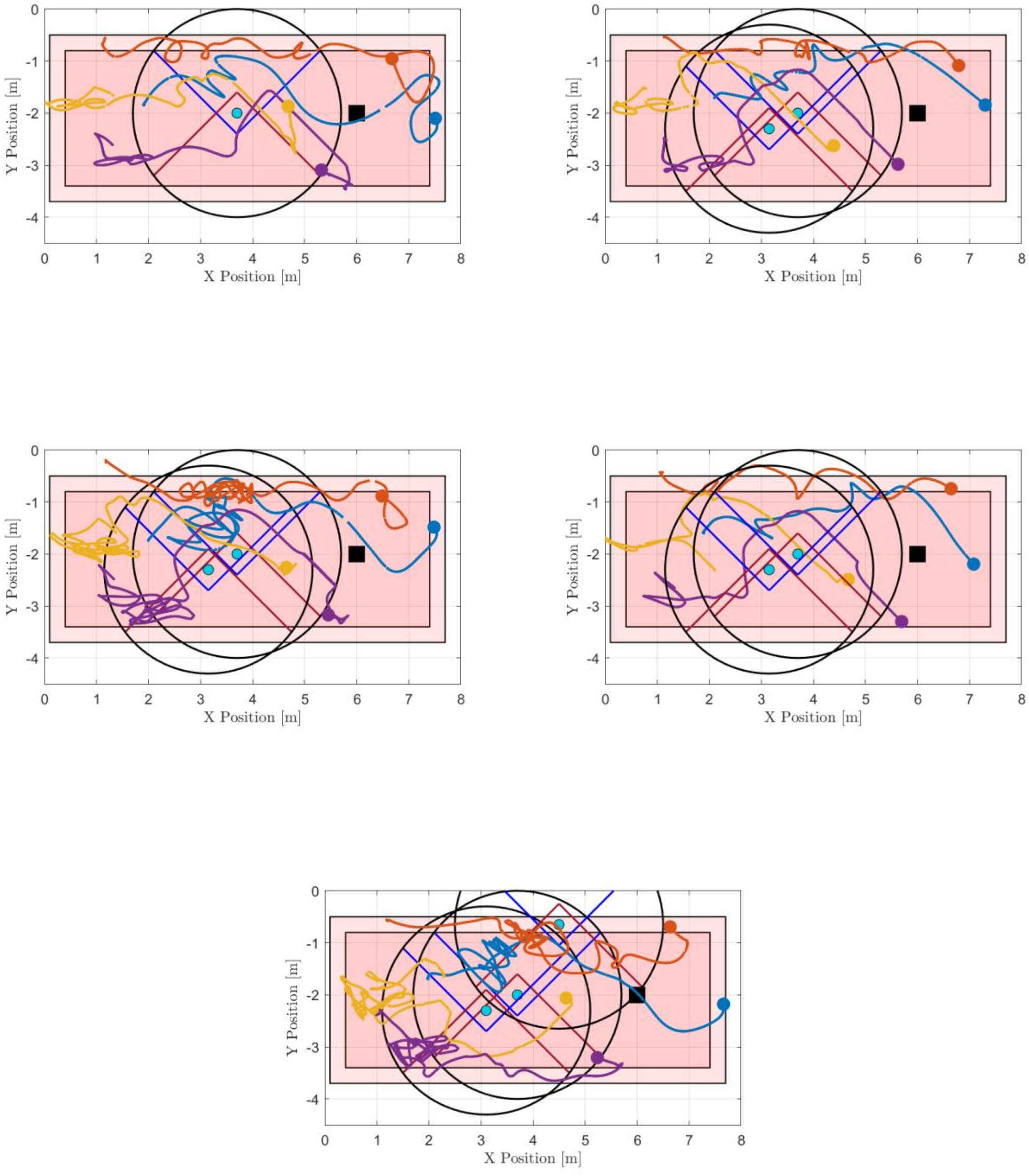


Figure 50: Algorithm 1 implementation formation resolution failure cases. Compilation of test cases in which the formation resolution was insufficient to effectively locate the obstacle barrier, resulting in a collision.

Formation Control

A test was conducted to investigate the effect of the switching formation control design, in which formation control was always enabled. The result was formation control overpowering boundary avoidance, causing the agent to collide with the obstacle as shown in Figure 51. It was determined that the switching architecture employed in the remaining tests did have a positive effect on enabling the agents to complete their objectives.

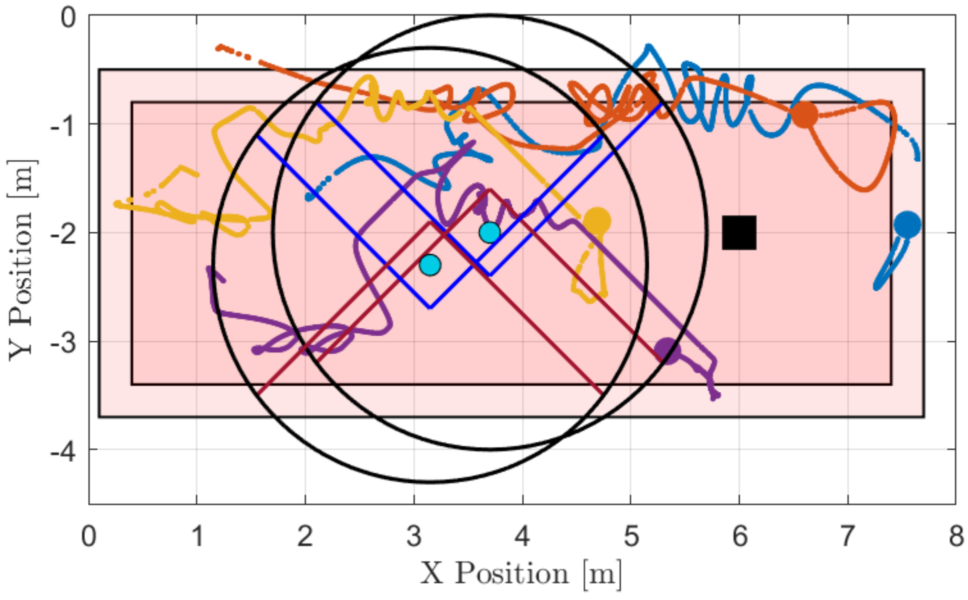


Figure 51: Algorithm 1 implementation of alternative formation control strategy. Test case when formation control was always active, including during obstacle avoidance. Agent 5 (shown in purple) is pushed into the obstacle as it attempts to maintain the formation, illustrating the advantage of a switching formation control scheme.

Drone Dynamics

The final failure case observed during algorithm 1 testing was due to the dynamics of the drone responding differently than commanded by the control law. For this case, agent 1 (shown in blue in Figure 52) moved slightly in to the keep-out zone but was able to recover when the drone began to respond to the commanded velocity. The gradient field is shown along with the commanded and measured velocities for the point at which the agent entered the keep-out zone.

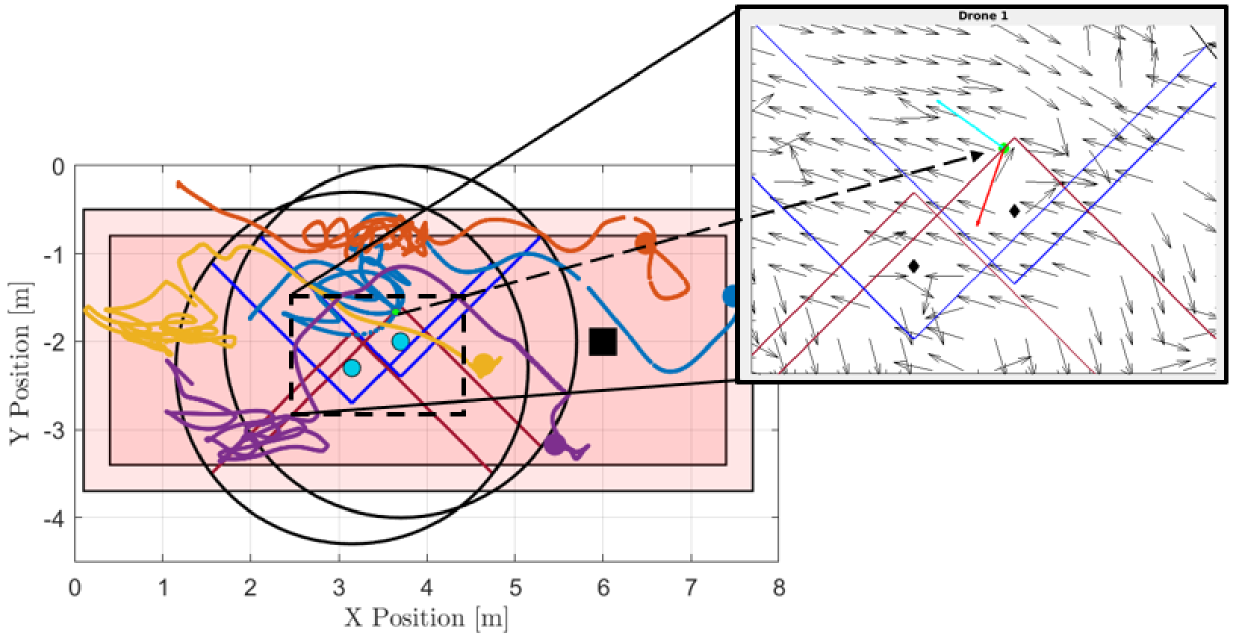


Figure 52: Algorithm 1 implementation failure due to actuator latency. Agent 1 (blue) moves into the keep-out zone of the obstacle (*left*) when the commanded velocity directed the agent away from the barrier, but the dynamics caused the drone to move in a different direction. The perceived gradient field estimate (*right*) shows how the algorithm directs the agent away from the obstacle. The cyan arrow represents the commanded velocity, pointing in a direction consistent with the black gradient estimate, while the red arrow is the measured velocity driving the agent into the obstacle.

References

[1] J. Cochran and M. Krstic, “Nonholonomic Source Seeking With Tuning of Angular Velocity,” *IEEE Trans. Automat. Contr.*, vol. 54, no. 4, p. 717, 2009.

[2] N. Ghods, M. Krstic, J. Cochran, and A. Siranosian, “3-D Source Seeking for Underactuated Vehicles Without Position Measurement,” *IEEE Trans. Robot.*, vol. 25, no. 1, 2009.

[3] S.-J. Liu and M. Krstic, “Stochastic source seeking for nonholonomic unicycle,” *Automatica*, vol. 46, pp. 1443–1453, 2010.

[4] A. S. Matveev, H. Teimoori, and A. V Savkin, “Navigation of a unicycle-like mobile robot for environmental extremum seeking,” *Automatica*, vol. 47, pp. 85–91, 2011.

[5] M. S. Stanković and D. M. Stipanović, “Extremum seeking under stochastic noise and applications to mobile sensors,” *Automatica*, vol. 46, pp. 1243–1251, 2010.

[6] J. ; Poveda *et al.*, “A Hybrid Adaptive Feedback Law for Robust Obstacle Avoidance and Coordination in Multiple Vehicle Systems,” in *2018 Annual American Control Conference (ACC)*, 2018, pp. 116–621.

[7] E. Bıyık and M. Arcak, “Gradient Climbing in Formation via Eextremum Seeking and Passivity-Based Coordination Rules,” in *Decision and Control, 2007 46th IEEE Conference on*, pp. 3133--3138.

[8] F. Zhang and N. E. Leonard, “Cooperative Filters and Control for Cooperative Exploration Battery Modeling and Control View project Robust Curve Tracking Control View project Cooperative Filters and Control for Cooperative Exploration,” *IEEE Trans. Automat. Contr.*, vol. 55, no. 3, 2010.

[9] L. Briñón-Arranz, “Collaborative Estimation of Gradient Direction by a Formation of AUVs under Communication Constraints,” in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, 2011, pp. 5583--5588.

[10] M. Gronemeyer, M. Bartels, H. Werner, and J. Horn, “Using Particle Swarm Optimization for Source Seeking in Multi-Agent Systems,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11427–11433, Jul. 2017.

[11] E. Rosero and H. Werner, “Cooperative source seeking via gradient estimation and formation control (Part 1),” in *2014 UKACC International Conference on Control (CONTROL)*, 2014, pp. 628–633.

[12] E. Rosero and H. Werner, “Cooperative source seeking via gradient estimation and formation control (Part 2),” in *2014 UKACC International Conference on Control (CONTROL)*, 2014, pp. 634–639.

[13] S. M. Khansari-Zadeh and A. Billard, “A Dynamical System Approach to Realtime Obstacle Avoidance,” *Auton. Robots*, vol. 32, no. 4, pp. 433–454, 2012.

[14] R. G. Sanfelice, M. J. Messina, S. Emre Tuna, and A. R. Teel, “Robust hybrid controllers for continuous-time systems with applications to obstacle avoidance and regulation to disconnected set of points,” in *American Control Conference, 2006*, 2006, p. 6 pp.

[15] C. G. Mayhew, R. G. Sanfelice, and A. R. Teel, “Robust source-seeking hybrid controllers for nonholonomic vehicles,” in *American Control Conference, 2008*, 2008, pp. 2722--2727.

[16] M. Mesbahi, M. Egerstedt, P. And, and O. Bookfinal, *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010.

[17] C. Godsil and G. Royle, *Algebraic graph theory*. Springer Science & Business Media, 2013.

[18] B. Mohar, Y. Alavi, G. Chartrand, and Oellermann O.R., "The Laplacian spectrum of graphs," Wiley, 1991.

[19] F. Santoso, M. A. Garratt, S. G. Anavatti, and I. Petersen, "Robust Hybrid Nonlinear Control Systems for the Dynamics of a Quadcopter Drone," *IEEE Trans. Syst. Man, Cybern. Syst.*, pp. 1–13, 2018.

[20] J. Chai and R. G. Sanfelice, "Forward Invariance of Sets for Hybrid Dynamical Systems (Part I)," *IEEE Trans. Automat. Contr.*, pp. 1–1, 2018.

[21] E. Frazzoli, M. A. Dahleh, and E. Feron, "Robust Hybrid Control for Autonomous Vehicle Motion Planning," in *roceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187),* 2000.

[22] R. Alur, T. Henzinger, and E. Sontag, *Hybrid Systems III: Verification and Control*, Vol. 3. Springer Science & Business Media, 1996.

[23] M. Branicky, "Studies in Hybrid Systems: Modeling, Analysis, and Control," Massachusetts Institute of Technology, 1995.

[24] H. Mohr, K. Schroeder, and J. Black, "Distributed Source Seeking and Robust Obstacle Avoidance through Hybrid Gradient Descent," in *IEEE Aerospace Conference*, 2019.

[25] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules," *IEEE Trans. Automat. Contr.*, vol. 48, no. 6, pp. 988–1001, 2003.

[26] G. Gargioni and M. Peterson, "A Fully Distributed Multirotor Autonomous Flight System Using 3D Position tracking with onboard PID," pp. 1–10.

[27] G. Gargioni, "Appendix A - PIXHAWK 1 . 0 AUTONOMOUS UAV SETUP FOR RASPBERRY PI 2 AND 2b," no. February, 2019.

[28] G. Gargioni, M. Peterson, J. B. Persons, K. Schroeder, and J. B. Black, "A Distributed Multipurpose Autonomous Flight System Using 3D Position Tracking and ROS."

[29] R. Olfati-Saber and R. Murray, "Distributed Structural Stabilization and Tracking for Formations of Dynamic Multi-Agents," in *41st IEEE Conference on Decision and Control,* 2002.

[30] X. Xu *et al.*, "Realizing Simultaneous Lane Keeping and Adaptive Speed Regulation on Accessible Mobile Robot Testbeds," in *IEEE Conference on Control Technology and Applications (CCTA),* 2017.

[31] D. Pickem *et al.*, "Safe, Remote-Access Swarm Robotics Research on the Robotarium," *arXiv Prepr. arXiv1604.00640*, 2016.

[32] D. Srivastava, R. Pakkar, A. Langrehr, and C. Yamane, "Adaptable UAV Swarm Autonomy and Formation Platform 347E: Robotics and Mobility Systems Section NASA Jet Propulsion Laboratory," in *IEEE Aerospace Conference,* 2019.

[33] H. Mohr, "Target Locating in Unknown Environments using Distributed Autonomous Coordination of Aerial Vehicles - Code Repository," 2019. [Online]. Available: https://code.vt.edu/hdmohr/target-locating-in-unknown-environments-using-distributed-autonomous-coordination-of-aerial-vehicles.

[34] S. Gebreyohannes and A. Karimoddini, "An Area-decomposition Based Approach for Cooperative Tasking and Coordination of UAVs in a Search and Coverage Mission," in *IEEE Aerospace,* 2019.

[35] O. Walker, F. Vanegas, F. Gonzalez, and S. Koenig, "A Deep Reinforcement Learning

Framework for UAV Navigation in Indoor Environments," in *IEEE Aerospace Conference*, 2019.

[36] F. Vanegas, K. J. Gaston, J. Roberts, and F. Gonzalez, "A Framework for UAV Navigation and Exploration in GPS-denied Environments," in *IEEE Aerospace Conference*, 2019.

[37] R. Falconi, L. Sabattini, C. Secchi, C. Fantuzzi, and C. Melchiorri, "Edge-Weighted Consensus Based Formation Control Strategy With Collision Avoidance," *Robotica*, vol. 33, no. 2, pp. 332–347, 2013.

[38] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, 2002.

[39] F. Aurenhammer, "Voronoi Diagrams-A Survey of a Fundamental Geometric Data Structure," *ACM Comput. Surv.* , vol. 23, no. 3, pp. 345–405, 1991.

[40] U. Borrmann, L. Wang, A. D. Ames, and M. Egerstedt, "Control Barrier Certificates for Safe Swarm Behavior," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 68–73, 2015.

[41] H. Mohr, "Formation Flying Demo - YouTube," 2019. [Online]. Available: https://youtu.be/dPrhw-KPzAs. [Accessed: 27-Mar-2019].

[42] H. Mohr, "Target Locating in an Unknown Environment - Algorithm 1 Demo - YouTube," 2019. [Online]. Available: https://www.youtube.com/watch?v=PduvFyXK1QE. [Accessed: 27-Mar-2019].

[43] H. Mohr, "Target Locating in an Unknown Environment - Algorithm 2 Demo - YouTube," 2019. [Online]. Available: https://youtu.be/6hOL4D3Nnr4. [Accessed: 27-Mar-2019].

[44] H. Mohr, "Estimated Gradient Field Over Time - YouTube," 2019. [Online]. Available: https://youtu.be/C1Wvp_9CZcE. [Accessed: 27-Mar-2019].