BICLARE: A Bio-Inspired Collaborative and Lightweight Algorithm for Robust Exploration

Hugo van Dijk

TU Delft

BICLARE: A Bio-Inspired Collaborative and Lightweight Algorithm for Robust Exploration

Thesis report

by

Hugo van Dijk

to obtain the degree of Master of Science in Computer Science
at the Delft University of Technology
to be defended publicly on May 14, 2025 at 12:30

Thesis committee:

Chair:	Dr. Ranga Rao Venkatesha Prasad
Supervisors:	Ir. Suryansh Sharma
External examiner:	Dr. Georgi Gaydadjiev
Place:	Faculty Electrical Engineering, Mathematics and Computer Science, Delft
Project Duration:	May 2024 - May 2025
Student number:	4864921

Abstract

Having the floorplan of an incident site upon arrival allows first responders to operate quickly and efficiently. This thesis explores the potential of using robotic swarms for environment mapping, intending to deploy these swarms to create maps before emergency personnel arrive on the scene. We present BICLARE, a lightweight collaborative algorithm for robust exploration. Inspired by ant colony behaviour, specifically how they use pheromones for communication and navigation, BICLARE implements a confidence model to determine the occupancy of cells within a map. Target selection, considering the travel time and estimated battery power, ensures efficiency. Including computation-saving parameters ensures its lightweight execution, enabling it to work on inexpensive hardware. The algorithm's performance was evaluated through a series of simulated experiments in various environments, proving it can generate accurate maps with adequate coverage in noisy, volatile environments. A real-world experiment demonstrated the feasibility of the proposed system, successfully running on low-cost, commercially accessible hardware.

Preface

I would like to thank Dr Ranga Rao Venkatesha Prasad and Ir. Suryansh Sharma for their guidance throughout my project. Their advice and feedback greatly helped me to achieve the final product.

I would also like to thank Dr Qing Wang for supplying me with the robotic platforms for my real-life experiment, and the amazing people from the TU Delft swarming lab for supplying me with the CrazyFlie drones and Loco Positioning system, as well as allowing me to perform my initial hardware tests in their lab.

Finally, I would like to thank the secretary of the Faculty Electrical Engineering, Mathematics and Computer Science for allowing me to perform my experiments in one of their meeting rooms.

Contents

Nomenclature

α_φ Pheromone evaporation factor

$\mathcal{B}_R(p)$ Bounding box with center p with side length $2R$

ω^a Agent alignment weight

ω^c Agent cohesion weight

ω^s Agent separation weight

ω^t Target weight

Ω_D Frontier distance weight

Ω_S Frontier size weight

ϕ_Δ Max pheromone value difference for merging children nodes

ρ Map resolution

θ_{turn} Turn threshold

ζ Cell information discount factor

f_e Sensor error factor

k Factor from the frontier selection probability formula

l Factor from the frontier selection probability formula

l_{free} Log-odds increase of occupancy confidence when a cell is observed as free.

l_{max} Log-odds upper clamping value for occupancy confidence.

l_{min} Log-odds lower clamping value for occupancy confidence.

$l_{occupied}$ Log-odds decrease of occupancy confidence when a cell is observed as occupied.

m Factor from the frontier selection probability formula

n Factor from the frontier selection probability formula

N_A Number of agents in the swarm

N_f Maximum number of frontier regions considered in frontier selection

N_s Maximum number of route sections

N_{cc} Number of covered cells by an agent

N_{cc} Number of covered cells

$O_i(j)$ Occupancy of cell j according to agent A_i

P_o Occupied probability threshold

R_a Agent alignment radius

R_c Agent cohesion radius

R_d Frontier reach radius

R_f Frontier search radius

R_o Object avoidance radius

R_s Agent separation radius

R_x Frontier separation radius

R_{comm} Communication range

R_{random} Random walk radius

R_{sensor} Distance sensor threshold

S_i State of agent A_i

T_φ Pheromone evaporation time

T_f	Minimum time between frontier checks
T_{map}	Map exchange interval
T_{spawn}	Average inter-arrival time for dynamic obstacles (Poisson)
T_{sync}	Time synchronization interval

List of Figures

List of Tables

1 Introduction

In emergencies, every second counts. It is vital that first responders can operate quickly and effectively. When searching through buildings or fighting fires, information about a scene is crucial for firefighters to operate safely. A map of the environment showing, among others, the accessibility of rooms is crucial[45]. In 2010, 72 firefighters died in the line of duty. Eight of these fatalities resulted from being trapped within a structure. In 2011, deaths due to becoming trapped increased by 4%, and a civilian dies in a residential fire every 208 minutes[44]. It is therefore essential for first responders to know their surroundings and locate victims quickly. Here, robotic swarms can be of assistance.

Discussing the possible role of swarm robotics in fire rescue, [9] states that dealing with the fire is not the problem. Finding the location of the fire and the casualties is important. They indicate that robots are not yet trusted with complex tasks. However, robots are potentially valuable for straightforward tasks such as building mapping, locating casualties, large-area search, and exit route guidance. Furthermore, they should be easy to handle, deploy, and maintain. In various countries, like the Netherlands and the United States, fire departments are continuously facing budget cuts [29] [14], leaving little room for expensive equipment. We believe robotic swarms could be used to map the potentially quickly changing indoor environment before the first responders arrive. Most fire departments deploy so-called Quick Response Vehicles (QRV) for medical calls [18]. On average, these QRVs arrive 1 minute and 59 seconds earlier at a scene than big fire engines. Here, we believe a QRV could also be used to deploy a robotic swarm at the scene, allowing them to start mapping well before the fire crew arrives. It will be beneficial if the potential loss of robots due to fire or collapsing structures results in no significant financial loss. This introduces the need for a swarming algorithm which:

- **Creates a map of the environment:** The algorithm should create a human-readable map for first responders.
- **Works in dynamic environments:** Obstacles produced throughout the mission should be present in the final map.
- **Is robust against sensor and pose estimation errors:** Indoor environments are prone to inaccurate sensor readings, but an accurate map should still be created.
- **Can run on inexpensive hardware:** Limited budget and potential hardware loss require low-cost hardware.
- **Is efficient:** The algorithm should maximize mapping coverage and accuracy while minimizing battery usage, saving cost.

This algorithm can be extended to support any additional sensors required for fire or victim detection.

We present BICLARE: A Bio-Inspired Collaborative and Lightweight Algorithm for Robust Exploration. Our algorithm is a multi-robot frontier-based exploration algorithm aimed at working in noisy, volatile environments where communication limitations can be present. It is designed to create accurate maps of any potential dynamic environment while being computationally inexpensive. BICLARE uses a confidence model to determine the occupancy of cells in our map. Each cell will have an occupancy value of 0, 1, or 2 if it is determined to be free, occupied, or ambiguous, respectively. In our model, we categorize any cell for which we are unsure about its occupancy (free or occupied) as ambiguous. These cells can be fully unexplored, very little explored, or have their confidence decay over time. The pheromone value $\varphi_i(j,t)$ represents the confidence that cell j is free at time t according to agent A_i. When $\varphi_i(j,t) = 0.5$, agent A_i is fully ambiguous about cell j's occupancy. From this point, whenever we discuss 'higher pheromones', we are indicating that the value is further from 0.5, meaning we are more certain about the cell being either free or occupied, i.e., 0.3 and 0.7 have the same pheromone strength. Furthermore, pheromone decay indicates that the certainty decreases. Unlike existing studies, we include confidence in our definition of a frontier. Formally, we define F as a list of frontier regions. A frontier region $F_k \in F$ is a region of adjacent non-occupied cells, separating explored and certain cells from uncertain or unexplored cells. Setting frontier regions as navigational targets enables agents to increase confidence in uncertain areas. Frontier regions consist of adjacent frontier cells. A cell j in an agent A_i's map is recognized as a frontier cell if and only if the following properties are met: (1) $O_i(j) = 0 \vee 2$ and (2) $\exists j' \in N_8(j) : O_i(j') = 2$. Here $O_i(j)$ denotes the occupancy of cell j according to agent A_i.

Our main contributions are:

1. **A bio-inspired novel lightweight algorithm for accurate indoor multi-robot exploration**
2. **An ant-inspired frontier selection function**
3. **A lightweight path planning algorithm based on wall-following behaviour**

This thesis proceeds as follows. Chapter 2 discusses the relevant literature, followed by the implementation of our algorithm in chapter 3. The setup of our experiments is discussed in chapter 4, and results are presented in chapter 5. We discuss our interpretation of the results in chapter 6. Finally, chapter 7 concludes our findings and suggests directions for future research.

2

Related Work

2.1. Swarming in Fire Rescue

Research has been done on swarming for fire fighting, but they are concerned with extinguishing the fire [7] [**innocente219self**] or assistive swarming [30] [31].

2.2. Models for navigation in Animals

In the paper "Flocks, Herds, and Schools: A Distributed Behavioral Model" [32], Reynolds introduces a powerful model for simulating the collective motion of flocks of birds or schools of fish. The model is based on three simple local rules: separation, avoiding collisions; alignment, steering towards the average heading of neighbouring flockmates; and cohesion, moving towards the average position of neighbouring flockmates. Each agent decentrally adheres to these rules, given the location and heading of its flockmates. Reynolds' study demonstrated that despite its simplicity, the flock displayed a fluid and lifelike interactive motion.

Ants deploy pheromones to recruit other ants. Scout ants leave Chemical trails behind when food is found. No further aid from the scout ant is required for other ants to follow this route [46]. Ants also incorporate negative feedback when locating food. When ants leave pheromone trails behind, each trail's strength indicates the source's quality. This allows for the exploitation of the most profitable resources. However, over-exploitation occurs when too many ants go to a single location. [48] shows that ants often choose sources with fewer ants over sources with many other ants. Also, fewer pheromones are deposited on routes leading to occupied sources. Some ants even deposit negative pheromones [33]. This prevents crowding when other food sources are available and colonies trapping themselves in local optima [48]. Deposited pheromones do not exist forever. They decay over time, which also directly depletes their effect on ants [21]. Attractive and repellent pheromones decay with similar gradients [33]. However, repellent pheromones have a higher initial effect, resulting in them also having a longer-lasting effect. Ant pheromones seem to follow exponential decay:

$$\varphi(z,t) = \varphi(z,t_0) \cdot e^{-\lambda(t-t_0)} \tag{2.1}$$

where λ is a constant [33][21][13], and can be effectively modelled with this formulation [34]. Ants can have knowledge about their environment separate from pheromone information. They can completely ignore pheromone trails when their memory conflicts with the route [48]. Ant behaviour based on pheromone levels can be modelled mathematically as well. When given two route options, right and left, formula 2.2 shows how the probability P_{ant} of an ant choosing

the right route is calculated.

$$P_{ant} = \frac{(k+R)^n}{(k+R)^n + (k+L)^n} \tag{2.2}$$

In this equation, R and L are the qualities of the right and left routes, respectively, determined by the number of ants that have visited that route previously. k and n are chosen constants. n influences the degree of nonlinearity. A high value means that a slight quality difference has a high influence on the probability. k influences the attraction to other routes, meaning a high k shifts the probability towards random [3][12]. Bianchi et al. [4] introduced a variation of this formula, adding a state transition heuristic. For a set of candidate locations for $J_k(i)$ for ant a_i, any two connected states (i.e., an ant's current location i, and a candidate location $j \in J_k(i)$), the probability at time t can be calculated with

$$P_{ant}(i,j,t) = \frac{\varphi(ij,t) \cdot \eta_{ij}^m}{\sum_{r \in J_k(i)} \varphi(ir,t) \cdot \eta_{ir}^m} \tag{2.3}$$

where $\varphi(ij,t)$ is the pheromone intensity along route ij at time t, and $\eta_{ij}^m = 1/d_{ij}$, the inverse distance between i and j. Here, m is a parameter influencing the effect of the distance on the probability.

2.3. Swarming

2.3.1. Stigmergy

The mechanism through which inter-animal communication occurs is called stigmergy. It explains how insects operate individually yet seem coordinated as a whole. Ant pheromones are an example of stigmergy. There are two main types: qualitative stigmergy and quantitative stigmergy. In the former, stimuli of different types will result in different responses from swarm members. The latter entails that locations with many pheromones are more attractive than locations with fewer [41]. De Nicola et al. studied stigmergy in a multi-agent system. The stigmergy was implemented as a decentralized data structure that contained their local knowledge. This knowledge is exchanged through messages with neighbours [11]. Salman et al. [36] simulated stigmergy by projecting UV light. The light simulated a pheromone trail. The colour of the floor, which was covered in a special coating, changed from white to magenta. After about 50 seconds, it changed back to white—this simulated pheromone decay.

2.3.2. Exploration

Much research has been done on environment exploration with robotic swarms. Most works are concerned with static environments. Smith et al. [38] perform fast exploration of environments by inferring unobserved map portions through a library. Simulated sensor readings from the library are matched with real sensor readings, and the enclosed unobserved space is inferred. This uses the assumption that most environmental structures are similar to existing structures. SLAM was used for robot localization. In [49], Yanguas-Rojas et al. concern themselves with victim search, identification, and evacuation in disaster environments. Part of their swarm's task set is an exploration of the mission environment. They use a frontier-based algorithm that tracks the weighted centroid of an area. They consider the time it takes each robot to reach a point. 8-neighbour breadth-first search is used to calculate the route distance to each point, as opposed to using Euclidean distance. Zhou et al. [51] proposed a Capacitated Vehicle Routing

Problem (CVRP). This minimizes the lengths of each agent's coverage path and balances the workload. They also prevent repeated exploration. The map is saved in a 3D hierarchical grid (octree) to decrease memory usage. The CVRP involves creating a $(N_h + 3) \times (N_h + 3)$ cost matrix where N_h is the number of cells in the grid. This describes the cost of moving to adjacent cells and reduces computation for calculating connection costs. A minimal navigation solution is introduced by McGuire et al. [26]. Their algorithm, called the Swarm Gradient Bug Algorithm (SGBA), gives each agent its own preferred direction in which it will travel. When an obstacle is encountered, an agent will follow the wall until its preferred direction is free again. They use RSSI for inter-agent collision avoidance, navigating back to the base station, and odometry for position estimation. McGuire et al. stated that the agents revisited rooms without proper organization. Exploration by setting waypoints is achieved by Kamalova et al. [22]. Global waypoints are set on frontiers, which agents will navigate to. When an agent encounters an obstacle, five equally distributed local waypoints are generated in front of it. It then determines which waypoints will lead to avoiding the obstacle, and one of them is randomly selected. Division of work is important in swarming. Wang et al. [47] achieve this by assigning each agent their sub-area. This avoids collisions and decreases revisitation. The algorithm uses a frontier-based approach for exploration, prioritising frontiers close by and in front of the agent, as moving and turning require energy. A walking state is entered when the sub-area is fully explored, which uses Particle Swarm Optimization (PSO) to decide the movement. Five variants of a random walk model were explored by Kegeleirs et al. [23]. They ported single-robot models to swarming scenarios and state that it is not the best approach to achieve swarm mapping. An agent chooses a random direction in each variant and moves towards it until some condition is met. These conditions, e.g., some set time or until an obstacle was encountered, were varied over the variants. Couceiro et al. [10] utilized a heterogeneous swarm with different roles throughout. They implemented deployment robots, which handle the initial deployment of their exploration robots. They ensure the latter stays communicationally inter-connected by staying spatially close. In [17], Hengstebeck et al. extend Boids for the purpose of search and rescue. In their Boids_Extended algorithm, they add goal-seeking and obstacle avoidance without modifying the three Boids rules. They achieve this by adding ghost boids. These are invisible boids that do not move on their own and are not physically there. They only function as virtual influences. They use solely repelling ghost boids on area boundaries, and ghost boids that only affect alignment to steer other boids towards a goal. They extended even further with their BOIDS_CBF algorithm, which adds a control barrier function. This defines the safe and unsafe states for all possible agent states (position and velocity). Agents are prevented from leaving the safe state set. Both algorithms have the same coverage, but the latter results in fewer collisions. Tran et al. [43] also built on the Boid approach, and introduced some new forces to determine the heading of each agent: previous heading and attraction to the closest frontier cell. Obstacle avoidance is achieved with a dynamic scale factor for the vector weights. They verified that their work still performed adequately with different communication ranges. They extended their algorithm to work with dynamic environments in [42]. In this study, they present their FSP algorithm. Their agents use virtual pheromones to ensure repeated coverage. They each keep two matrices containing these pheromone values: a coverage matrix containing explored and obstacle-free cells, and an obstacle matrix containing occupied cells. They extend Boids with a virtual wall avoidance force to keep agents within specified bounds, and a frontier attraction force. The

latter is a vector towards the navigational goal point defined as:

$$G_{F_j} = \frac{\sum_{F_{j,k} \in F_j} F_{j,k}}{|F_j|} \tag{2.4}$$

with $|F_j|$ the number of cells in frontier region F_j. A frontier region is a group of adjacent frontier cells. The region used in this vector is selected through

$$F^* = \underset{\forall F_j \in F}{\arg\min} (\Omega_D \left\| p_i - G_{F_j} \right\|_2 - \Omega_S |F_j|) \tag{2.5}$$

where ω_D and ω_S are weights and p_i is the agent A_i position.

$$v_i^{frontier} = \begin{Bmatrix} G_{F^*} - p_i, & \text{if } v_i^s = 0 \\ 0, & \text{else} \end{Bmatrix} \tag{2.6}$$

To calculate the cohesion, separation, and alignment vectors we use information from neighbouring agents. For a team of N agents, $A_1, A_2, ..., A_N$, at time t neighbours are defined as follows:

$$N_i^c = \{A_k | k \neq i \wedge \|p_i(t) - p_k(t)\| < R_c\} \tag{2.7}$$

$$N_i^s = \{A_k | k \neq i \wedge \|p_i(t) - p_k(t)\| < R_s\} \tag{2.8}$$

$$N_i^a = \{A_k | k \neq i \wedge \|p_i(t) - p_k(t)\| < R_a\} \tag{2.9}$$

where R_c, R_s, and R_a are defined radii. These are then used to determine the average position of neighbours:

$$C_i^v(t) = \frac{\sum_k p_k(t)}{|A|}, \forall k \in N_i^v \tag{2.10}$$

The cohesion vector is defined as a vector towards the average position of neighbours:

$$v_i^c(t) = C_i(t)^c - p_i(t) \tag{2.11}$$

Separation is calculated inversely:

$$v_i^s(t) = p_i(t) - C_i^s(t) \tag{2.12}$$

Differently, the alignment vector used the average velocity of neighbours:

$$v_i^a(t) = \frac{\sum_k v_k^h(t)}{|A|}, \forall k \in N_i^v \tag{2.13}$$

where v_k^h is agent k's velocity vector.

They also used a virtual wall avoidance vector to keep the agents within specified bounds, v_i^w

All combined, the resulting swarm vector for agent A_i is calculated with

$$v_i^{swarm}(t+1) = \hat{v}_i^{swarm}(t) + \omega^c \hat{v}_i^c(t) + \omega^s \hat{v}_i^s(t) + \omega^a \hat{v}_i^a(t) + \omega^w \hat{v}_i^w(t) + \omega^{frontier} \hat{v}_i^{frontier} \tag{2.14}$$

where ω^c, ω^s, ω^a, ω^w, and $\omega^{frontier}$ are weights[1].

Obstacles are detected using LIDAR and entered into the obstacle matrix. Once a cell has been determined to contain an obstacle, no new observation can change this. When an obstacle cell is within the obstacle avoidance radius R_o from the agent, the agent determines which angles around it are blocked for movement. Given all cells within R_o from agent A_i: C, the object-agent safety radius R_p, and each cell's center C_j, the blocked angles $[-\beta_j;\beta_j]$ for that cell are calculated with:

$$[-\beta_j;\beta_j] = [-\arcsin\frac{R_p}{A_iC_j};\arcsin\frac{R_p}{A_iC_j}] \tag{2.15}$$

Fig. 2.1 illustrates how blocked angles are determined in practice.

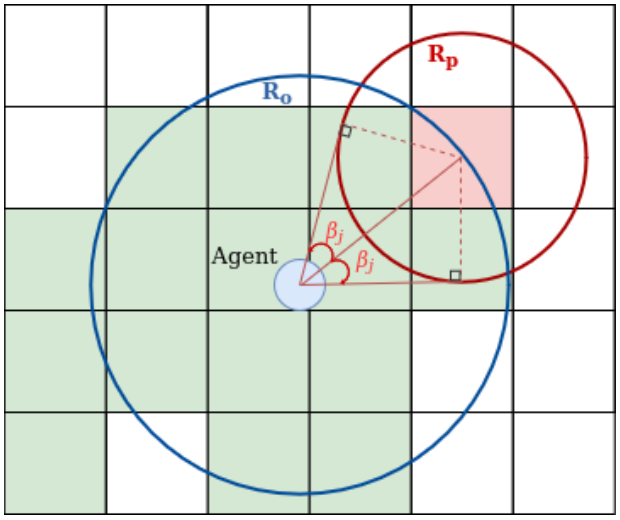


Figure 2.1: Illustration of the determination of blocked angles β_j by an obstacle cell.

The total set of blocked angles is determined by

$$D_j : \{\theta_i :\leq \theta_\iota \leq \theta_{A_i} + \beta_j, \theta_\iota \in \Theta\} \tag{2.16}$$

where $\Theta = \{\theta|\theta = k\theta_s, k \in \mathbb{Z}, \theta \leq \theta < 360\circ\}$. This is the set of all angles $[0, 360]$ given step size θ_s the set of all angles From the set of free-to-move angles $\chi = \Theta \setminus D_j$, the direction α_r is the smallest difference between the target vector (combined force vector v_{target}) direction α_t and all angles in L. It determines the acting force vector with[2]

$$v^i = \begin{bmatrix} v^i_x \\ v^i_y \end{bmatrix}^T = \begin{bmatrix} v^i_{swarm_x} \cdot cos(\alpha_r) - \\ v^i_{swarm_y} \cdot sin(\alpha_r) \\ v^i_{swarm_x} \cdot sin(\alpha_r) + \\ v^i_{swarm_y} \cdot cos(\alpha_r) \end{bmatrix}^T \tag{2.17}$$

[1]Normalization of vectors was not explicitly stated in the original publication [42]. However, the author confirmed via personal correspondence that this is intended.

[2]Personal correspondence with the author confirmed an error in the original publication of this formula. The form presented here is correct.

Work	Decentralized	Unknown environments	Dynamic environments	Mapping	Designed for sensor errors	Verified with limited comm.	Ran on low-cost hardware	Rea -life proof of concept
[10]	✓	✓	×	×	×	range	✓	✓
[17]	×	✓	×	×	×	N.A.	N.A.	×
[20]	✓	✓	×	×	×	×	✓	✓
[22]	×	✓	×	✓	×	×	×	✓
[23]	✓	✓	×	✓	×	N.A.	✓	✓
[26]	✓	✓	×	×	×	×	✓	✓
[27]	✓	✓	✓	✓	×	×	N.A.	×
[38]	✓	✓	×	✓	×	range	×	✓
[40]	✓	✓	✓	×	✓	range	✓	✓
[42]	✓	✓	✓	✓	×	range	×	✓
[43]	✓	✓	×	✓	✓	range	×	✓
[47]	✓	✓	×	✓	×	×	N.S.	✓
[49]	✓	✓	×	✓	×	range	N.A.	×
[51]	✓	✓	×	✓	×	✓	✓	✓

Table 2.1: Comparison of algorithms from existing works.

Pheromones linearly evaporate over time according to a set evaporation time. When a cell's pheromone value reaches zero, it is deemed undiscovered again. Given a rectangular area of X by Y meters and matrix resolution ρ, the memory requirement of each matrix is $O(X/\rho \cdot Y/\rho)$

Mendonça et al. [27] also uses ant-inspired pheromones as inter-agent markers. They use Ant Colony Optimization (ACO) to explore dynamic environments, creating fuzzy maps. It is aimed towards disaster site exploration so rescue teams can increase response times. The agents in their experiments were fitted with sensors to detect the pheromones, making it difficult to implement in real life.

We give an overview of the features of algorithms from existing works in Table 2.1. Here, we indicate which algorithms are decentralized, work in entirely unknown and/or dynamic environments, and if they create a map of their environment. Their robustness is marked by indicating whether they verified their works with limited communication (range and loss, as well as potential sensor and pose estimation errors. We also note whether they ran their algorithm on low-cost hardware. We deem the used hardware low-cost when the microcontroller and obstacle detection sensors are commercially available for under 40 euros a piece.

2.3.3. Localization

Indoor localization is an ongoing challenge. Itani et al. [20] tackle the challenge by using acoustic 'chirps' to calculate the Euclidean distance between robots. Using a base station and related reference points allows their agents to determine their global position. They showed that their method achieved a median localization error of 15cm. Sun et al. [39] achieved indoor localization by combining UWB (anchor), IMU, and odometry data. Placing anchors around the environment, they consider line-of-sight (LOS) and non-line-of-sight situations (NLOS) for the UWB. They show that even in NLOS situations, their proposed method achieves a maximum

localization error of 1.033m.

2.3.4. Communication in dynamic environments

Talamami et al. [40] claim to show that less inter-robot communication allows the robots to better react to environmental changes. They task a swarm with reaching a consensus about the highest quality site in the environment, with each agent exchanging their own opinion. In their experiments, they proved that reducing the range of communication made the swarm react better to changing site quality as it averted local optima.

2.3.5. Mapping

Memory-efficient mapping is crucial for robots with limited resources. Octrees divide a 3D space into a hierarchical voxel grid [19] [15]. The same concept can be applied to 2D environments, reducing the trees to Quadtrees [51]. These trees ensure that only explored cells need to be stored. OctoMap uses a confidence-based model to determine the occupancy of a cell. Cell observations are combined using the log-odds notation, allowing simple addition. Limiting the lower and upper bounds of the L-confidence limits the number of updates required to change the state of a voxel, ensuring a quicker reaction to changing environments.

3

A Bio-Inspired Collaborative and Lightweight Approach for Robust Exploration

In this thesis, the exploration algorithm called BICLARE is inspired by Dynamic Frontier-Led Swarming[42]. Dynamic Frontier-Led swarming achieved repeated coverage while running on robots fitted with a Raspberry Pi 4 [35]. In this study, their work is extended with a confidence model to account for sensor and pose estimation inaccuracies (section 3.2.1), a novel target selection algorithm including probability-based frontier selection (section 3.1.2), and a quadtree for memory-optimized map storing (section 3.2). We show two versions of the algorithm: BICLARE-WF and BICLARE-PP. The former uses wall-following behaviour to navigate around obstacles (section 3.1.4), and the latter uses a simple path-planning algorithm based on wall-following behaviour (section 3.1.5). The codebase will be made available at a later date.

3.1. Exploration Algorithm

Agents have five possible states. Described by S_i, an agent's state can be either NO_MISSION, EXPLORING, RETURNING, FINISHED_EXPLORING, and MAP_RELAYED, denoted as 0, 1, 2, 3, and 4, respectively.

Agents start in NO_MISSION, in which they do nothing but communicate. When an agent receives the mission start signal, it transitions to EXPLORING. This is the main state in which exploration and mapping occur. Agents can only drive straight forward or turn in place, either way. When the mission timer ends, agents transition to RETURNING, where they attempt to return to their deployment site, and conditionally transition to FINISHED_EXPLORING and MAP_RELAYED (see section 3.3). Every timestep, an agent starts by broadcasting all relevant information, like its estimated position and its current target, to agents within communication range R_{comm}. Maps are only exchanged every T_{map} seconds, with some random variations to ensure not all exchanges occur at the same time. Only cells that were updated since the earliest last exchange of all agents within range are sent to reduce unnecessary communications. The pseudocode for the main agent algorithm is given in Alg. 1.

Algorithm 1 Main Agent Loop

```
1:  procedure MainAgentLoop
2:      if S_i = 0 then DoNothing return
3:      BroadcastMessage(p, F*)
4:      BroadcastMapIfCloseAgents
5:      BroadcastMessage(heading)
6:      TimeSyncWithCloseAgents
7:      CheckAndParseMessages
8:      if S_i = 3 ∨ 4 then
9:          StopDriving
10:         if S_i = 3 then
11:             if recently exchanged maps with another agent then
12:                 S_i ← 4
13:     else
14:         CheckForObstacles
15:         CalculateNextPosition
16:         Step                                          ▷ Drive or Turn
17:     CheckMissionEnd
18:     t ← t + t_tick
```

BICLARE assumes an agent is fitted with a number of distance sensors. In CheckForObstacles, the value of each distance sensor is read. Suppose a sensor ray is intersected within R_{sensor}, then the corresponding map cell is updated as observed occupied (see section 3.2.1) unless the detected obstacle is estimated to be another agent. All cells between the agent and the collision are updated as observed free. If no obstacles are detected by a sensor, cells covered by the sensor within R_{sensor} are updated as observed free.

In CalculateNextPosition, the agent cohesion, avoidance, and alignment vectors are calculated as described in section 3.1.1. If the agent is in state exploring, it evaluates if it has to determine a new target. This depends on any of these conditions:

- If the agent is random-walking but has walked far enough (see section 3.1.2).
- If the agent is its current target.
- If it has been at least T_f since the last check and the distance from the agent to the current selected frontier $\leq R_d$ and the difference between our target heading and the vector angle from the agent to the frontier $\leq \theta_{turn}$.
- Distance from agent to current selected frontier $\leq R_o$.

Obstacle avoidance is done according to equation 2.17. However, additionally, as we account for pose estimation inaccuracy, the forward-facing sensor, denoted by y, is read. If this detects a ray intersection within $0.5R_o$, D_j is extended with $[\theta_y - 45°, \theta_y + 45°]$.

3.1.1. Vector Forces

Agent cohesion, separation, and alignment vectors are calculated according to equations 2.11, 2.12, and 2.13, respectively. In our implementation, we assume that the map boundaries are solid walls, so the agent cannot escape the boundaries. Thus, v^w is not included in our implementation.

Similar to [42], our swarm vector is calculated according to:

$$v_i^{swarm}(t+1) = \hat{v}_i^{swarm}(t) + \omega^c \hat{v}_i^c(t) + \omega^s \hat{v}_i^s(t) + \omega^a \hat{v}_i^a(t) + \omega^{target} \hat{v}_i^{target} \tag{3.1}$$

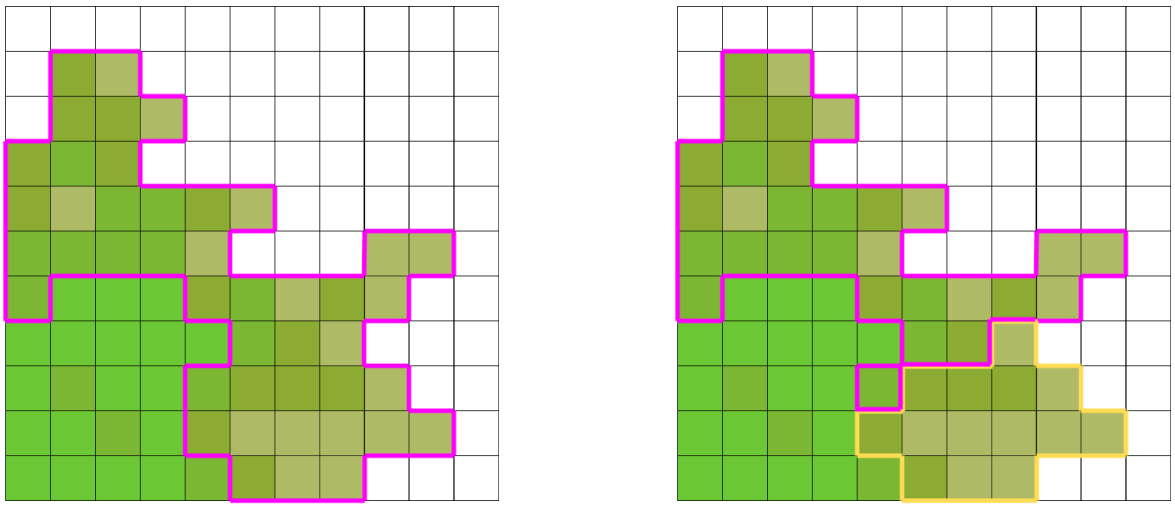
with

$$v_i^{target} = \begin{Bmatrix} G_i - p_i, & \text{if } v_i^s = 0 \\ 0, & \text{if } v_i^s \neq 0 \end{Bmatrix} \tag{3.2}$$

The method for determining the target location G_i is described in section 3.1.3.

3.1.2. Frontier-Led Swarming

New frontiers to discover are determined by creating frontier regions. Frontier regions are created by merging adjacent frontier cells. We use a non-optimal merging algorithm, which is presented in Alg. 2. In our algorithm, potentially separately created but adjacent groups are not merged later. This ensures frontier regions are limited in size. With complete merging, one agent could navigate to a region's centre, where multiple agents can navigate to sub-regions of the same frontier. Our merging algorithm and a smaller choice of maximum considered frontier regions N_f decrease the computation required to select a target. Fig. 3.1 shows the difference between optimal merging and our algorithm.



(a) Full merging **(b)** BICLARE merging

Figure 3.1: Optimal frontier region creation vs. BICLARE frontier region creation.

Algorithm 2 Frontier Cell Merging

$C_q \leftarrow \{c \in C \mid c \in \mathcal{B}_{R_f}(p_{A_i}) \text{ and } O(c) \in \{0, 2\}\}$
$F_q \leftarrow \{c_q \in C_q \mid O_i(c_q) = 0 \vee 2 \text{ and } \exists c'_q \in N_8(c_q) : O_i(c'_q) = 2\}$
$F \leftarrow \emptyset$
for $F_{q,k} \in F_q$ **do**
    $added \leftarrow$ false
    for $F_i \in F$ **do**
        for $F_{i,j} \in F_i$ **do**
            if $F_{q,k} \in N_8(F_{i,j})$ **then**
                $F_{i,j} \leftarrow F_{i,j} \cup \{F_{q,k}\}$
                $added \leftarrow$ true
                break
        if $added$ **then**
            break
    if $\neg added$ **then**
        $F \leftarrow F \cup \{\{F_{q,k}\}\}$
if $|F| > N_f$ **then**
    $F \leftarrow \text{sort}(F, \text{key} = \text{len}, \text{reverse} = \text{true})$
    $|F| \leftarrow n_q$

Not all frontier regions obtained from Alg. 2 are considered by the agent. To stimulate agent dispersion, frontier regions whose centre is close to another agent's position or selected frontier are discarded, that is:

$$F^u = \{F_j \in F \mid d_{F_j,k} > R_x \wedge d_{F_j,k} > R_d \wedge F_j \notin \Gamma^*, \forall k \in A, k \neq i\} \tag{3.3}$$

where $d_{F_j,k} = \left\| G_{F_j} - p_k \right\|$.

R_x is the frontier separation radius, and R_d is the distance at which we deem a frontier reached, and new targets are considered.

When we have established F^u for the given R_f, we determine the probability of choosing every region $F_j \in F^u$. We use a variation on equations 2.2 and 2.3:

$$P_{agent}(p, F_j, t) = \frac{(k + \Phi(F_j, t))^{-n} \cdot |F_j|^l \cdot H(p, F_j)^{-m}}{\sum_{F_r \in F_u} (k + \Phi(F_r, t))^{-n} \cdot |F_r|^l \cdot H(p, F_r)^{-m}} \tag{3.4}$$

where

$$H(p, F_j) = \omega_t \cdot T(p, F_j) + \omega_b \cdot B(p, F_j) \tag{3.5}$$

$T(p, F_j)$ is the estimated time it takes an agent to go from position p to the centre of the region F_j. $B(p, F_j)$ is the estimated battery power it takes the agent to go from position p to the centre of the region F_j. For these estimations, BICLARE-PP uses the route generated from PeriPlanner (section 3.1.5), and BICLARE-WF uses Euclidean distance. Battery estimations are made according to the specifications of two TT DC gearbox motors [1]. Only motion-related estimations are used here, as only that is directly affected by the route.

For our purposes, the pheromones are seen as repellent pheromones. This entails higher pheromone values in an area, making it less favourable. The multiplication with $|F_j|^l$ ensures bigger frontier regions are more favourable, resulting in a trade-off between ambiguity and region size. $H(p, F_j)$ ensures a trade-off between region quality and time and battery considerations. Time and battery usage are combined in the term as they are closely related. Φ represents the pheromone value of a region and is calculated according to Alg. 3.

Algorithm 3 Φ

Input: F_j, t
$\varphi_i(F_j, t) \leftarrow 0$
for each $F_{j,k} \in F_j$ **do**
  for each $F_{n,l} \in N_8(F_{j,k})$ **do**
    if $\varphi_i(F_{n,l}, t) = 0.5$ **then**
      $\varphi_i(F_{j,k}, t) \leftarrow 0.5$
      BREAK
    else
      $\varphi_i(F_{j,k}, t) \leftarrow \underset{u \in \{\varphi_i(F_{j,k},t), \varphi_i(F_{nl},t)\}}{\arg\min} (|u - 0.5|)$
  $\varphi_i(F_j, t) \leftarrow \varphi_i(F_j) + \varphi_i(F_{j,k}, t)$
return $\frac{\varphi_i(F_j,t)}{|F_j|}$

Different from [42], agents use a randomly generated number in combination with a Cumulative Distribution Function to determine their frontier objective F^*. The corresponding target point is calculated using equation 2.4.

It can occur that even though a route to a target is established, the route is not actually viable, for example, with narrow passages or mapping inaccuracies. When an agent is unable to proceed towards its target for an extended period of time, its FrontierEvaluator will make it temporarily avoid that target and the closest corresponding subtargets. A set Γ is established with Alg. 4. Then a set of all blocked locations Γ^* is generated with equation 3.6.

$$\Gamma^* = \{l \in L | \, \|l - \gamma\| < R_x, \forall \gamma \in \Gamma\} \tag{3.6}$$

Algorithm 4 $FrontierEvaluator$

Global $count_G,\ \theta_e, \Gamma$
if $\|G_{\text{previous}} - G_i\| < \frac{1}{2}R_x$ **then**
    if $|\theta_\alpha| > \theta_e$ **then**
        $count_G = count_G + 1$
        $\theta_e \leftarrow \min(\theta_e + \frac{1}{3}f_{\text{tick}}, 89)$
        if $count_G > count_{threshold}$ **then**
            $\Gamma \leftarrow \Gamma \cup \{G_i\} \cup \{G_{i,\text{sub}}\}$
            $count_G \leftarrow 0$
        else
            $\theta_e \leftarrow \max(\theta_e - 1, 0)$
    else
        $count_G \leftarrow 0$
        $\theta_e \leftarrow 89$
$G_{\text{previous}} \leftarrow G_i$

Whenever no frontier within range can be considered ($F^u = \emptyset$), agents perform random-walk behaviour. A random point along the perimeter of the root cell of an agent's quadtree is selected and set as $G_{i,random}$ such that $G_{i,random} \notin \Gamma^*$. Whenever an agent has moved outside R_{random} from the random walk deployment site, a new random target is selected.

3.1.3. Target Selection

An agent A_i's target G_i is set according to

$$G_i = \begin{cases} G_{i,sub}, & \text{if } S_i < 2 \wedge G_{i,sub} \neq 0 \\ G_{i,random}, & \text{if } S_i < 2 \wedge G_{i,sub} = 0 \\ G_{i,t_0,sub}, & \text{if } S_i \geq 2 \end{cases} \tag{3.7}$$

The determination of $G_{i,sub}$ is described in section 3.1.5, and $G_{i,t_0,sub}$ is the subtarget towards the agent's deployment position, i.e. the agent's position at $t = 0$ (see section 3.3).

3.1.4. Wall Following

When an agent notices there is no free angle within 90 degrees towards the target, it initiates wall-following. It chooses a random direction (left or right), in which it will follow the wall. When the direction towards the target becomes free again, it will stop wall-following. When the initial chosen direction leads it back to the position at which it started wall-following, it will detect that it is in a loop, and progress in the other direction. Fig. 3.2 illustrates this.

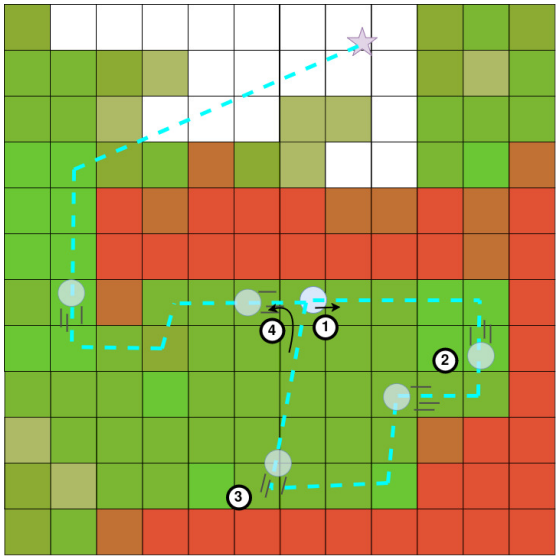


Figure 3.2: Wall-following implementation. ① The agent randomly chooses to follow the wall to the right. ② The agent keeps following the wall. ③ The agent detects a loop, and continues in the other direction.

3.1.5. Path Planning

We present our novel simple path planner: PeriPlanner, which plans routes along the perimeter of obstacles and is thus based on wall-following behaviour. The algorithm raytraces a straight line from a start coordinate (agent position) to a given target coordinate and follows the outline of the first obstacle it encounters in the given wall-following direction. It utilizes the fact that in our quadtree, adjacent highly likely occupied cells ($\varphi(z,t) < P_o$) are 4-connected. For each route section, we perform ray tracing towards the target. If the nearest intersection with an occupied cell is at least half the resolution away, we classify that direction as free. If for any route section, an unobstructed straight line can be traced towards the agent, all sections before that are removed from the route, optimizing the route. Lines are raytraced using the Amanatides-Woo voxel traversal algorithm [2]. Then the intersected edge is determined using the Liang-Barsky line clipping algorithm [24]. Alg. 5, 6, and 7 show the pseudocode for the main functions of PeriPlanner.

As we perform raytracing for each route section, we set a maximum number of route sections N_s for which we deem a route viable. This saves computation time by discontinuing the establishment of a route when it contains too many sections, preventing many raytracing operations. Note that this does not represent the actual distance traversed by the route, as edge sections will most often be much shorter than ray-traced sections.

BICLARE initiates PeriPlanner to establish routes with both wall-following directions, and the shortest one is returned (Fig. 3.3). If a potential new target is close to the agent's current target, the same wall-following direction used for the route towards the current target is used to find a new route. This prevents an agent from frequently switching directions.

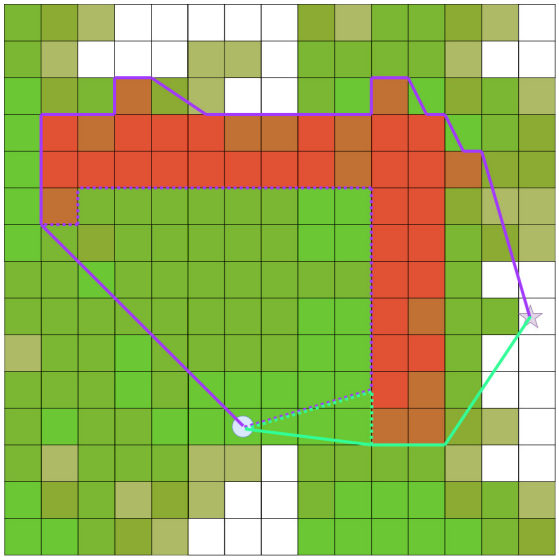


Figure 3.3: Illustration of how PeriPlanner determines a route. The dotted lines display the full generated route. The solid lines show the route after optimization. In this case, the cyan route is the shortest and would be returned.

Occasionally, PeriPlanner may fail to establish a route to a target. This occurs when it overlaps its own route, at which point it terminates. An illustration of this phenomenon can be seen in Fig. 3.4. When this happens in both directions, the target is deemed unreachable at this time.

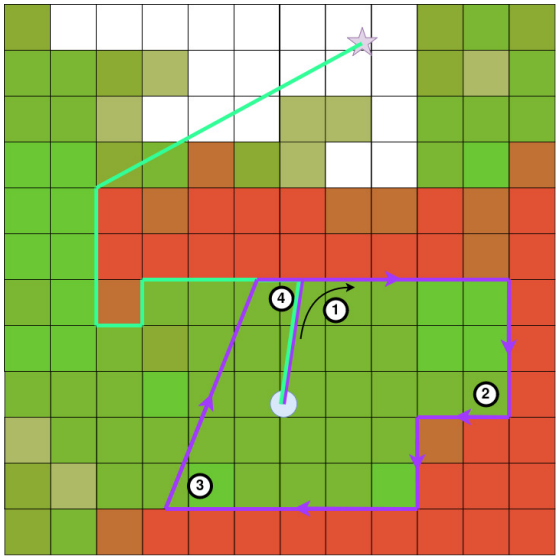


Figure 3.4: Illustration of a case where PeriPlanner fails (purple line). ① PeriPlanner follows the wall to the right. ② It keeps following the contours of the obstacle. ③ Direction to the target free. ④ It overlaps its route, resulting in a fail. PeriPlanner still generates a route in the other direction, which will be chosen.

Algorithm 5 PeriPlanner Function

function PeriPlanner
  Input: p_{start}, p_{target}, wfd
  Output:ψ ▷ Route
  $(c_{isec}, e_{isec}) \leftarrow$ U(p_{start}, p_{target})
  if $\neg c_{isec}$ **then**
    return {(p_{start}, p_{target})}
  $(p_L, p_{mid}, p_R) \leftarrow e_{isec}$
  $\psi = ((p_{start}, p_{mid}))$
  if wfd = 1 **then** ▷ Left
    $\psi \leftarrow \psi \frown ((p_{mid}, p_L))$
  else ▷ Right
    $\psi \leftarrow \psi \frown ((p_{mid}, p_R))$
  FollowWallToTarget(c_{isec}, e_{isec}, p_{target}, wfd, ψ)
  if $\neg\psi$ **then**
    return ()
  $a = 0$
  for $i = 0$ **to** $|\psi|$) **do**
    $s = \psi[i]$
    if $s[0] = p_{A_i}$ **then**
      $a = i$
  TruncateFront(ψ, a)
  $\psi \leftarrow \psi \frown ((\psi[-1][1], p_{target}))$
  return ψ

Algorithm 6 followWallToTarget Function

1: **function** FOLLOWWALLTOTARGET(p_{A_i}, c, e, p_{target}, wfd, ψ)
2: **if** $|\psi| > N_s$ **then**
3: $\psi \leftarrow()$
4: **return**
5: d = DIRTOTARGETFREE(c, e, p_{target}, wfd, ψ)
6: **if** d > 0 **then**
7: **if** d = 1 **then**
8: c_{isec}, $e_{isec} \leftarrow$U(c, p_{target})
9: FOLLOWWALLTOTARGET(p_{A_i}, c_{isec}, e_{isec}, p_{target}, wfd, ψ)
10: **return**
11: $(e, c) \leftarrow$ GETCONNECTEDEDGEANDCELL($\psi[-1]$)
12: **if** ¬c **then**
13: $\psi \leftarrow$
14: **else**
15: ($p_{e,start}$, $p_{e,end}$) $\leftarrow e$
16: c_{isec,A_i}, $e_{isec,A_i} \leftarrow$U(p_{A_i}, $p_{e,end}$)
17: **if** $\neq c_{isec,A_i}$ **then**
18: $\psi \leftarrow \psi \frown ((p_{A_i}, p_{e,start}))$
19: $\psi \leftarrow \psi \frown ((p_{e,start}, p_{e,end}))$
20: FOLLOWWALLTOTARGET(p_{A_i}, c, e, p_{target}, wfd, ψ)

Algorithm 7 DirToTargetFree Function

```
1:  function DIRTOTARGETFREE(c, e, p_target, wfd, ψ)
2:      if p_target = c then
3:          return 2
4:      c_isec, e_isec ←U(start, target)
5:      if c_isec then
6:          if c_isec = c then
7:              return 0          ▷ We are intersecting the same cell, we are on the edge of
8:          if e_isec ∈ψ then
9:              return 0                          ▷ We cannot go over the same edge twice
10:         if ‖e − e_isec‖ > SIZE(c_isec)/2 then
11:             e_last ←ψ[−1]
12:             p_start,e_last, p_mid,e_last, p_end,e_last ←e_last
13:             REMOVELAST(ψ)
14:             ψ ←ψ ⌢((p_start,e_last, p_mid,e_last))
15:             p_start,e_isec, p_mid,e_isec, p_end,e_isec ←e_isec
16:             ψ ←ψ ⌢((p_mid,e_last, p_mid,e_isec))
17:             if p_end,e_last = p_start,e_isec then
18:                 ψ ←ψ ⌢((p_mid,e_isec, p_end,e_isec))
19:             else if p_end,e_last = p_end,e_isec then
20:                 ψ ←ψ ⌢((p_mid,e_isec, p_start,e_isec))
21:             else
22:                 if wfd = 1 then
23:                     ψ ←ψ ⌢((p_mid,e_isec, p_start,e_isec))
24:                 else
25:                     ψ ←ψ ⌢((p_mid,e_isec, p_end,e_isec))
26:             return 1                          ▷ Line to target is partly free
27:         return 0
28:     e_last ←ψ[−1]
29:     p_start,e_last, p_mid,e_last, p_end,e_last ←e_last
30:     REMOVELAST(ψ)
31:     ψ ←ψ ⌢((p_start,e_last, p_mid,e_last))
32:     return 2                                  ▷ Line to target is fully free
```

$U(start, target)$ gives the first intersection and the corresponding cell with an obstacle on the line from start to target. Raytracing with the Amanatides-Woo Voxel algorithm is employed here.

When a route ψ is established, an agent's sub-target $G_{i,sub}$ will be set according to Alg. 8. This algorithm chooses the next route section required to progress towards the target.

Algorithm 8 FollowPath

Input: ψ_i
Global variables: $\psi_{\text{current}}, k_{\text{current}}$
if $\psi_{\text{current}} \neq \psi_j$ **then**
    $\psi_{\text{current}} \leftarrow \psi_j$
    $k_{\text{current}} \leftarrow 0$
$start, end \leftarrow \psi_{\text{current}}[k_{\text{current}}]$
$G_{i,sub} \leftarrow end$
if $\|G_{i,sub} - p_i\| < 3r_0$ **then**
    if $\neg\text{U}(p_i, G_{i,sub})$ **then**
        $k_{\text{current}} \leftarrow k_{\text{current}} + 1$
        $start_{next}, end_{next} \leftarrow \psi_{\text{current}}[k_{\text{current}}]$
        $G_{i,sub} \leftarrow end_{next}$
return $G_{i,sub}$

3.1.6. Time Synchronization

In a simulation, all agents' steps are started in sequence. This means that potential CPU overload or time oscillator drift will not cause misalignment of perceived mission time between agents. In real life, however, this can happen. Mission time alignment is crucial for pheromone evaporation calculations. So to counter misalignment, we implemented a TimeSynchronizer to balance mission time between agents. Based on [37], it calculates the difference between message propagation from i to j and vice versa for every pair of neighbouring agents ij. The differences are averaged, and each agent's mission time is compensated accordingly. This ensures that, for any drift between agents, a consensus in mission time is reached over all agents in a matter of cycles. A time synchronization cycle is initiated every T_{sync} seconds, with some random variations to ensure that not all message exchanges occur at the same time.

3.2. Mapping

As a local map, agents use a single quadtree. A quadtree is a hierarchical tree where each node that is not a leaf contains four children (see Fig. 3.5). The benefit of using a quadtree is that groups of unexplored cells are not unpacked to leaves, and can be contained in some higher-level node. Also, cells with similar confidence can be merged and also be contained by some parent nodes. The quadtree is updated through agent observations and through maps exchanged with agents. The maximum depth of the quadtree is determined by setting the desired resolution ρ. Then depth $= \log_2(\frac{L}{\rho})$ where L is the diameter of the square containing the environment, i.e., the size of the quadtree root.

3.2.1. Confidence

In BICLARE, the occupancy confidence level [0,1] of a cell describes the confidence an agent has that that cell is free. When a cell has a value of 0.5, the agent is entirely ambiguous about its occupancy. A confidence value closer to 0 or 1 means the agent is more certain that the cell is occupied or free, respectively. Our implementation is inspired by [19].

When an agent observes the map, it updates the corresponding cells in its map. When agent

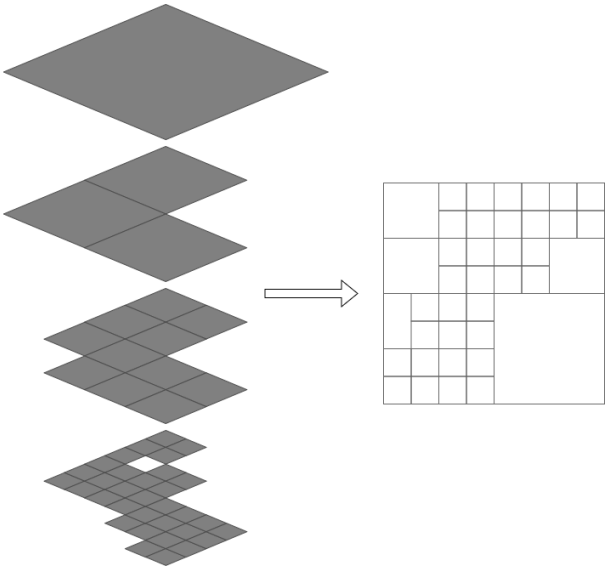


Figure 3.5: Illustration of the workings of a quadtree.

A_i gets information $X(z,t)$ about node z at time t, either being its own observation $X_{own}(z,t)$ or information received from another agent $X_{other}(z,t)$), its confidence (and thus pheromone level) is updated according to equation 3.8.

$$L(\varphi_i(z,t)) = max(min(L(\varphi_i(z,t-1)) + L(P(z \mid X(z,t))), l_{max}), l_{min}) \tag{3.8}$$

with

$$L_i(P) = \log\left[\frac{P}{1-P}\right] \tag{3.9}$$

$L(0.5) = 0$, so fully ambiguous information about a cell will not change its confidence level.

l_{min} and l_{max} denote the lower and upper bounds on the log-odds value. These clamping values limit the number of updates to change the occupancy of a cell and prevent potential overconfidence in noisy environments. As we are assuming obstacles are ever existing, but new obstacles can spawn, it is sensible to choose $l_{max} < -l_{min}$. This ensures free space decays towards uncertainty faster than occupied space.

$X_{own}(z,t)$ will be an observation about the cell, i.e., the cell is observed free or occupied. $L(P(z \mid X_{own}(z,t)) = l_{free} > 0$ or $L(P(z \mid X_{own}(z,t)) = l_{occupied} < 0$ respectively. What this entails is that if $l_{free} = -l_{occupied}$, a cell that is observed as occupied N times and observed as free N times will have a confidence value of 0.5. $t_{0,i}(z)$ is the time at which cell z was last observed according to agent A_i. After updating with $X_{own}(z,t)$, the agents sets $t_{0,i}(z) \leftarrow t$.

A discount factor ζ is introduced when info is received from another agent A_j. Because pheromone values of cells received from other agents will probably already have been influenced by observations from the receiving agent A_i, ζ prevents overconfidence due to their own observations. Using the discount factor and received pheromone value $\varphi_j(z,t)$ from agent A_j,

$$P_i(z \mid X_{other}(z,t)) = \begin{cases} \zeta \cdot (\varphi_j(z,t) - 0.5) + 0.5, & \text{if } t_{0,j}(z) > t_{0,i}(z) \\ 0, & \text{else} \end{cases} \tag{3.10}$$

Hereafter, agent A_i sets $t_{0,i}(z) \leftarrow \max(t_{0,i}(z), t_{0,j}(z))$. The condition ensures that if agent A_i is the last to observe cell z according to agent A_j, $X_{other,j}(z,t)$ does not affect $\varphi_i(z,t)$.

As we design for dynamic environments, certainty, $|\varphi_i(z,t) - 0.5|$ decays over time. Certainty decay occurs according to equation 2.1, adapted to our confidence implementation:

$$\varphi(z,t) = (\varphi(z,t_0) - P_x) * e^{-\lambda(t-t_0)} + P_x \tag{3.11}$$

with

$$\lambda = -\log\left[\frac{\alpha_\varphi}{T_\varphi}\right] \tag{3.12}$$

T_φ is the time after which our pheromone value will have dissipated to $\alpha_\varphi \cdot \varphi(z,t_0)$.

$$P_{i,x}(z) = \begin{cases} P_o, & \text{if } O_i(z) = 1 \\ 0.5, & \text{else} \end{cases} \tag{3.13}$$

This ensures cells deemed occupied will never be considered otherwise, solely due to time decay. Note that observations about the cell being free can change its occupancy.

When all children of a node have similar pheromone values, that is, the minimum and maximum pheromone values are within ϕ_Δ, they are 'merged', by deleting them and having their parent obtain their average pheromone value. This saves significant memory. Cells with $\varphi(z,t) < P_o$ are not merged, as lowest-level cells and neighbours are required for PeriPlanner to function properly.

3.3. Return strategy and mission end

When an agent is in the RETURNING (2) state, it attempts to return to the position it was at when the mission started, p_{i,t_0}. It uses PeriPlanner to determine the best route ψ_{return}, and switches to a random walk when no viable route is found. $G_{i,sub}$ is obtained with FollowPath(ψ_{return}), which is then used to calculate v_i^{target} according to equations 3.7 and 3.2.

When an agent estimates that it is within R_{return} from its deployment location, it transitions to the FINISHED_EXPLORING state. Since agents occasionally get stuck, R_{return} is dynamic, and increases when an agent does not progress towards the deployment location (Alg. 9). In FINISHED_EXPLORING, an agent makes sure it has recently exchanged maps with another agent so that other not yet finished agents have relayed the latest information. Depending on R_{comm}, this exchange will contain most, if not all, information from the remaining agents. Hereafter, it switches to MAP_RELAYED, as shown in Alg. 1, and its mission is finished. It still exchanges messages in this final state, for inter-agent collision avoidance, and remaining map updates.

Algorithm 9 Return strategy

```
if ||p_i − p_{i,t0}|| ≤ R_return then
    S_i ← 3
else
    if agent A_i got closer to p_{i,t0} then
        R_return ← max(R_deployment − 0.01, 0.1)
    else
        R_return ← R_return + 0.002
```

3.4. Complexity Analysis

3.4.1. Time Complexity

Frontier selection requires the most computation time. Frontier merging is $O(N_{fc}^3)$, where N_{fc} is the maximum number of cells within a box with sides of $2 \cdot R_f$. This gives

$$N_{fc} = \max(\left\lceil \frac{2R_f}{\rho} + 1 \right\rceil^2, N_l) \tag{3.14}$$

since we are including any cell intersected by the bounding box. N_l is the maximum number of leaves in the quadtree, i.e., its map is completely filled with cells of ρ size ($N_l = (L/\rho)^2$, where L is the size of the square sides containing the environment, i.e., the size of the quadtree root). However, N_{fc}^3 is a very loose upper bound. Since in Alg. 2 whenever F gets bigger, F_i will get smaller. PeriPlanner has a worst case time complexity if $O(\max(N_l, N_s) \cdot \sqrt{N_l})$, as the Amanatides-Woo Voxel algorithm has time complexity $O(\sqrt{N_l})$. For each frontier region in F, we run PeriPlanner twice. Updating leaves in the quadtree takes $O(\log \sqrt{N_l})$ time, and exchanging maps takes $O(N_l)$. Note that our cell 'merging' of non-occupied cells will greatly decrease the occupation time by reducing the number of leaves. The fact that agents only exchange cells with newly observed changes ensures that the actual number of exchanged cells will often be much less than N_l.

So BICLARE's worst-case time complexity is $O(N_l^3)$ if $\left\lceil \frac{2R_f}{\rho} + 1 \right\rceil^2$, N_f, and N_s are all bigger than N_l. However, when we ensure they are smaller, we get $O\left(\left\lceil \frac{2R_f}{\rho} + 1 \right\rceil^2\right) + O\left(N_f \cdot N_s \cdot \sqrt{N_l}\right) + O(\log \sqrt{N_l}) + O(N_l)$.

3.4.2. Memory Complexity

BICLARE requires each robot to keep a quadtree of the map. The number of nodes is determined by the depth of the tree depth $= \log_2(\frac{L}{\rho})$. N_l will be to 4^L. However, due to merging, the actual number of leaves n_l will be smaller in practice. The total amount of nodes in the quadtree will be $n_n = \frac{4n_l - 1}{3}$. So the worst-case memory complexity in a full non-merged quadtree is $O(N_l)$,

4

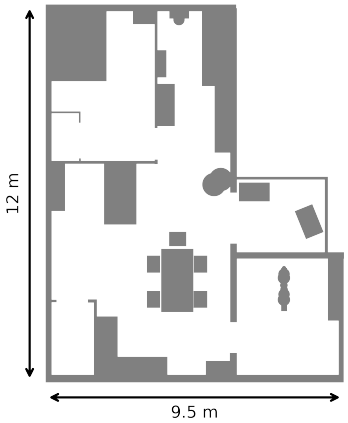
Experiments

4.1. Environments

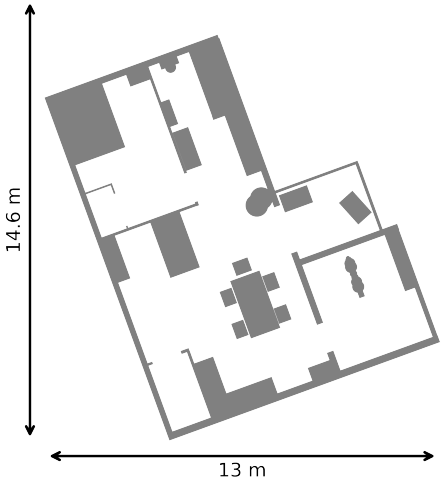
4.1.1. Simulated Environments

The first set of experiments was conducted in simulation. The simulated experiments were executed on PCs running Ubuntu and in the ARGoS3 robot simulator [8]. ARGoS3 is suitable for simulating large-scale robotic swarms in a multi-physics simulator. Its open-source nature makes it easy to implement adaptations to existing robots. In our simulations, an agent is fitted with four ultrasonic distance sensors pointed forwards, right, left, and backwards. Performance was evaluated on three maps, which were designed to represent typical structures that firefighters might encounter: a house, an office building, and a museum. Each of these maps was also rotated by $20°$ to evaluate BICLARE's performance when the agent's map grid is not aligned with walls and other structures. This gives us a total of six maps with which we run our simulated experiments: HOUSE, HOUSE_TILTED, OFFICE, OFFICE_TILTED, MUSEUM, and MUSEUM_TILTED. Fig. 4.1 illustrates the maps and notes their accessible (AA) and inaccessible area (IA). A cell is considered inaccessible if it is entirely occupied by static obstacles or walls. For both accessible and inaccessible cells we only look at cells at least partially within the arena's bounds. If this holds, an agent cannot observe any part of the cell. Here, we only account for static obstacles. Cells at any point reachable during the experiment are counted towards AA. Experiments in HOUSE and HOUSE_TILTED have a mission timer T_{end} of 400 seconds, in OFFICE and OFFICE_TILTED $T_{end} = 600$ seconds, and in MUSEUM and MUSEUM_TILTED $T_{end} = 1000$ seconds. All agents are deployed close together, at the side of each map (Fig. 4.5), which likely reflects how they would be deployed in a real-life disaster site exploration. For more realism, message transmission time was simulated with a transmission speed of 10 Mbps with 10ms jitter. Map exchange happened in blocks of 50 nodes. We also introduce a new parameter P_{loss}, which is the communication message loss probability. This portrays the random chance of each message being lost in transmission. For BICLARE, the root node's width and height, and thus FSP's coverage and obstacle matrix width and height, were set to $\max(\text{map width} + 1, \text{map height} + 1)$, to account for possible inferred cells outside the area bounds due to pose estimation and sensor inaccuracy, which is described below.

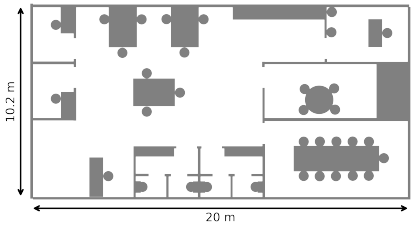
As real-life sensor readings and indoor pose estimations are not perfect, we tested the performance of our algorithm with simulated sensor inaccuracy and noise. As our algorithm expects four distance sensors, we chose to simulate the HC-SR04 ultrasonic sensor [28]. We base our implementation on a study by Gandha et al. [16]. Our implementation uses their accuracy data combined with simulated noise to compose an equation simulating a sensor's reading given



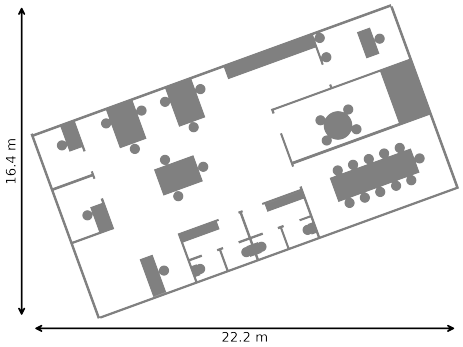
(a) HOUSE. $\rho = 0.203125$, $AA = 75.299m^2$, $IA = 15.472m^2$



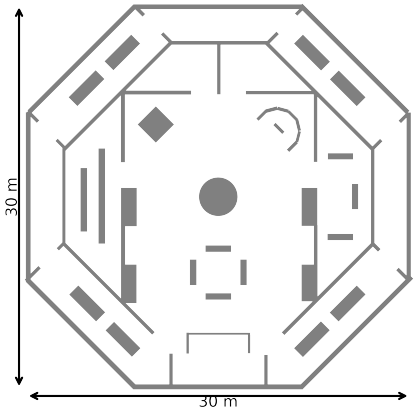
(b) HOUSE_TILTED. $\rho = 0.24375$, $AA = 78.605m^2$, $IA = 12.715m^2$



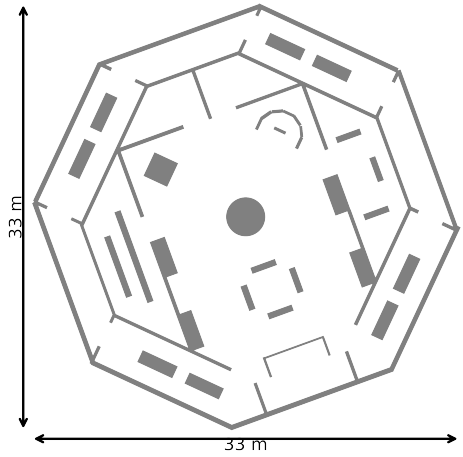
(c) OFFICE. $\rho = 0.1640625$, $AA = 173.154m^2$, $IA = 30.443m^2$



(d) OFFICE_TILTED. $\rho = 0.18125$, $AA = 176.347m^2$, $IA = 28.055m^2$



(e) MUSEUM. $\rho = 0.2421875$, $AA = 689.311m^2$, $IA = 44.108m^2$



(f) MUSEUM_TILTED. $\rho = 0.265625$, $AA = 701.827m^2$, $IA = 34.503m^2$

Figure 4.1: Six virtual maps used for performance evaluation.

the actual distance in meters:

$$d_{measured} = d_{actual} - (0.027 d_{actual} + 0.00543) f_e + d_{noise} \tag{4.1}$$

where

$$d_{i,noise} = J_d() + n_{d,i}, \quad J_d() \sim \mathcal{N}(0, \sigma_d^2) \tag{4.2}$$

with $\sigma_d = 5 f_e$. $n_{d,i}$ is noise individual to agent A_i, based on its position and id. This simulates inter-robot differences. $J_d()$ is regenerated every assignment of $d_{i,noise}$.

$$n_i = \mathrm{K}(Z_i + \mathrm{round}((p_{i,actual,x} + p_{i,actual,y}) \cdot 100), 2\sigma) - \sigma \tag{4.3}$$

Z_i is an individual value for each agent.

$$K(a,b) = \begin{cases} \lfloor \frac{a}{b} \rfloor \mod b, & \text{if } \lfloor \frac{a}{b} \rfloor = 0 \\ b - (\lfloor \frac{a}{b} \rfloor \mod b), & \text{else} \end{cases} \tag{4.4}$$

To simulate indoor pose estimation errors, heatmaps were generated for each map, which determine the offset and direction of the pose estimation errors. These heatmaps were based on a study by Sun et al. [39]. Their localization solution depends on anchors placed around the environment. This could be done in firefighting scenarios by, for example, placing beacons at entrances or windows of the structure. They achieved the following performance:

Spec	Value (m)
min	0.004
max	1.033
μ	0.227
median	0.193
σ	0.154

Table 4.1: Specifications of localization error simulation.

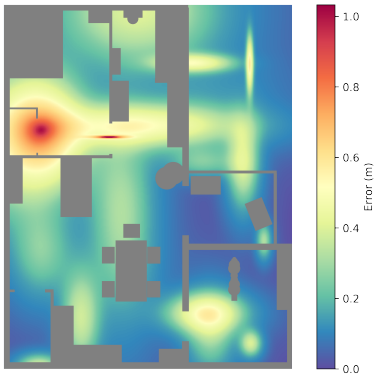
We ensured our generated heatmaps for position error magnitude had similar specifications and simulated (non-) line-of-sight areas. They are depicted in Fig. 4.2. The direction of the position estimation error was also determined using heatmaps. These can be seen in Fig. 4.3. The _TILTED variants were simulated with the same heatmaps.

Using the position estimation magnitude and direction heatmaps, M_{mag} and M_{dir} respectively, are used to determine the simulated estimated position p_i of each agent A_i:

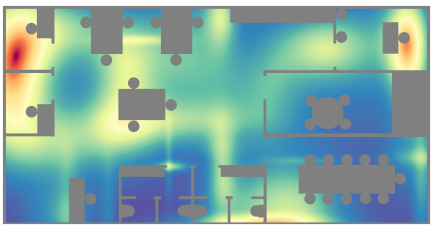
$$p_i = p_{i,actual} + v_{i,offset} \tag{4.5}$$

$$v_{i,offset} = \begin{bmatrix} \varepsilon_p \cos\theta_p \\ \varepsilon_p \sin\theta_p \end{bmatrix} \tag{4.6}$$

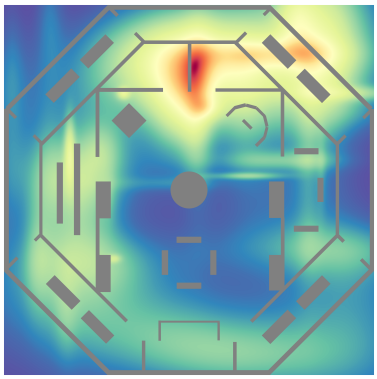
$$\varepsilon_p = M_{mag}(p_{i,actual,x}, p_{i,actual,x}) f_e + p_{i,noise} \tag{4.7}$$



(a) HOUSE

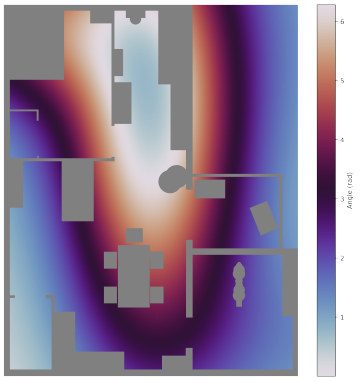


(b) OFFICE

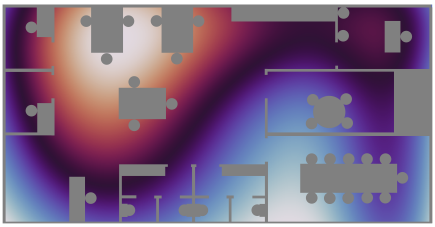


(c) MUSEUM

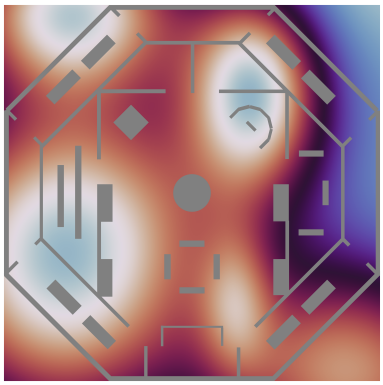
Figure 4.2: Localization error magnitude heatmaps M_{mag}. For all maps: $\mu : 0.226, \text{min} : 0.004, \text{max} : 1.033, \sigma : 0.152, \text{median} : 0.193$



(a) HOUSE



(b) OFFICE



(c) MUSEUM

Figure 4.3: Localization error direction heatmaps. M_{dir}

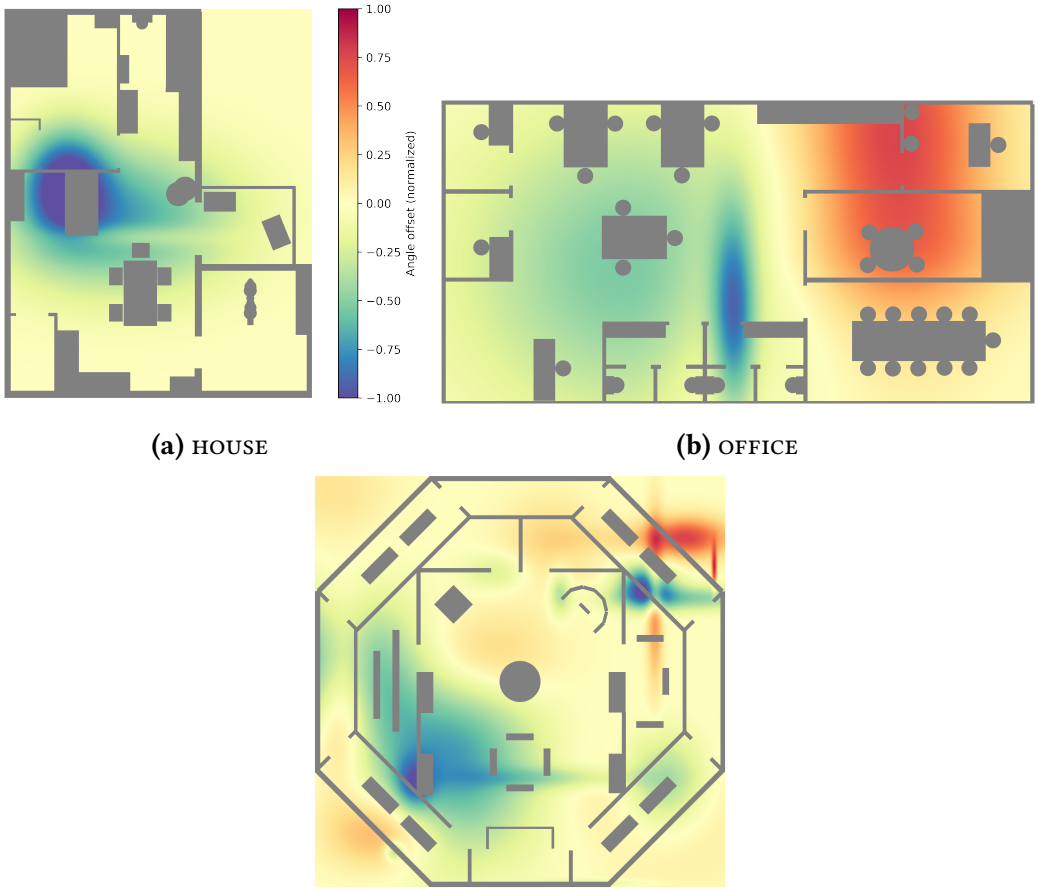
$$p_{i,noise} = J_p() + n_{p,i} \tag{4.8}$$

$$\theta_p = M_{dir}(p_{i,actual,x}, p_{i,actual,x}) f_e + \theta_{i,noise} \tag{4.9}$$

$$\theta_{i,noise} = J_\theta() + n_{\theta,i} \tag{4.10}$$

$\sigma_p = 0.05 f_e$ $\sigma_\theta = 0.0698 f_e$ in radians

Orientation estimation errors were also simulated in M_θ, shown in Fig. 4.4.



(a) HOUSE **(b)** OFFICE

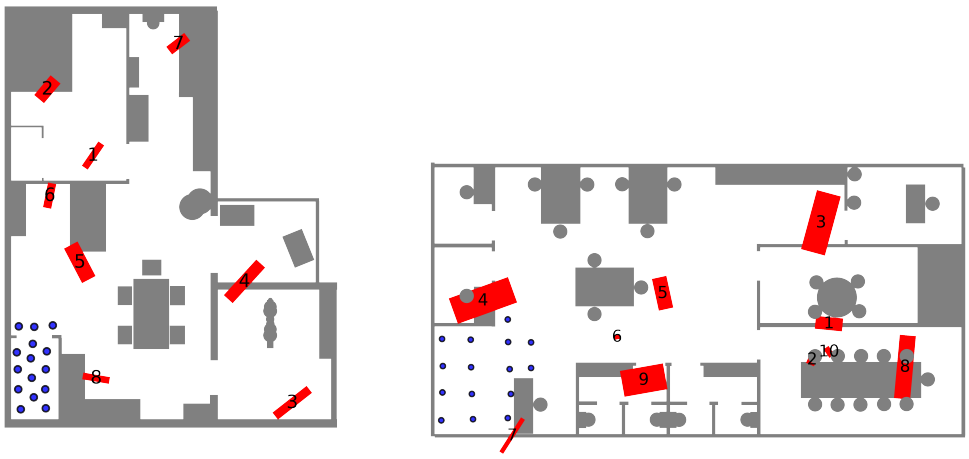
(c) MUSEUM

Figure 4.4: Orientation error heatmaps M_θ.

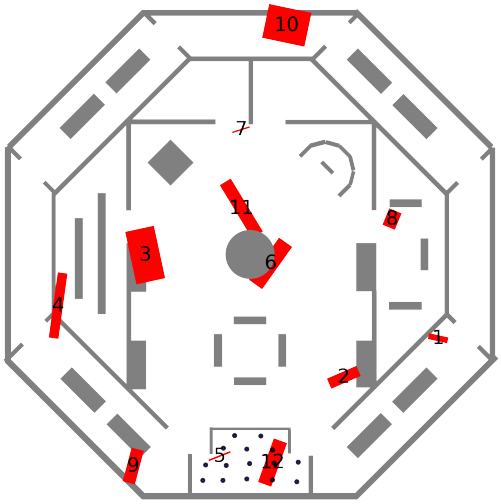
$$\theta_i = \theta_{i,actual} + M_\theta(p_{i,actual,x}, p_{i,actual,x}) f_e + \theta_{i,noise} \tag{4.11}$$

4.1.2. Dynamic Spawning Obstacles

We also verified the adaptability of the swarm when new obstacles emerged in free spaces. For each map, dynamic obstacles were created. T_{spawn} determines the average inter-arrival time between two obstacles, this is randomized between repeated experiments according to a Poisson process. When we state that $T_{spawn} = \infty$ no dynamic obstacles are spawned.



(a) HOUSE **(b)** OFFICE



(c) MUSEUM

Figure 4.5: Dynamic obstacles with spawn order (red), and agent deployment positions (blue) in each map.

We test the performance of BICLARE for varying configurations. Additionally, we compare its performance to that of the Dynamic Frontier-Led Swarming (FSP) algorithm [42]. For better comparison, we also combine FSP with our quadtree and confidence implementation: FSP-QT, and observe how that performs against BICLARE-PP and BICLARE-WF. Tran et al. discuss how they ensured grouping and alignment in their swarm by setting ω^c and ω^a. In our experiments, we found that setting both to 0 ensured the best results. For a fair comparison, we tested FSP both with $\omega^c = \omega^a = 0$ and with $\omega^c = 0.23$ and $\omega^a = 0.5$. The latter will be referred to as FSP-G.

4.1.3. Real-life proof of concept environment

Our real-world experiment is conducted on a homogeneous set of small robots (Fig. 4.6). Each agent runs a Raspberry Pi Pico 2W [25], which has two 150MHz cores and 520kB of SRAM and is commercially available from €6.50. Obstacle detection was done with four HC-SR04 ultrasonic sensors, which typically cost €3.00. The robots propelled themselves with two TT DC gearbox motors [1]. For pose estimation, each agent features a Bitcraze CrazyFlie 2.1 drone[5]. These drones did not apply any propulsion or run any part of the algorithm and were solely used for pose estimation. The Bitcraze Loco position system [6] was used for the localization of the agents. This uses ultra-wideband to localize the drones. Their orientation was obtained from the drone's IMU. The drone communicates the location and orientation with the Pico over UART.

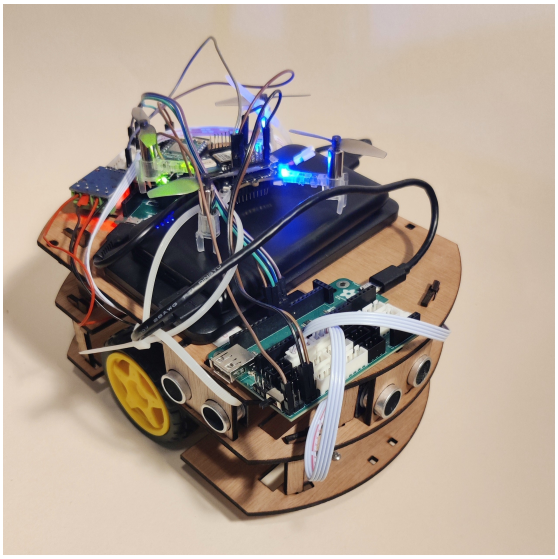


Figure 4.6: Robot used in the real-life experiment. On top is a CrazyFlie 2.1.

Fig. 4.7 shows the environment in which the experiment was run. T_{end} was set to 200 seconds.

4.1.4. Parameters

The parameters that are constant overall in simulated experiments can be found in Appendix A unless stated otherwise. Distance values are given in meters. Their values were tuned manually. Performance was better with $\omega^c = \omega^a = 0$ as this encouraged agent separation for increased coverage. Note that not all parameters are used in both BICLARE and FSP. All algorithms ran at 16 ticks per second. We used homogeneous swarms of Pi-Puck robots [50]. These agents

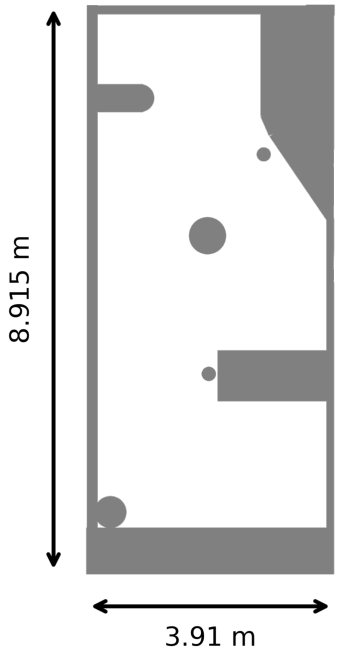


Figure 4.7: Map used for the real-life experiment. $\rho = 0.284375$, $AA = 32.348m^2$, $IA = 3.882m^2$

feature two parallel wheels that allow them to rotate in place. They have a radius of 3.62cm, an inter-wheel distance of 5.65cm, and a mass of 400g. The robots' max speed and rotation speed were set to 0.53 m/s and 1.77 rad/s, respectively.

The real-life robots also ran at 16 ticks per second. They achieved a maximum speed of 0.23 m/s, had a radius of 10.5cm, and a mass of 400 grams. They also featured two parallel wheels with an inter-wheel distance of 12.4cm. The parameter configuration for the real-life experiment can be found in Appendix B.

4.2. Performance Metrics

All experiments were repeated 5 times, with random seeds 1-5, and the average of each metric was taken over all repetitions. While their deployment position was constant, each agent's deployment orientation was randomized for each repetition. We ran experiments with varying swarm sizes (N_A) of 2, 4, 6, 10, and 15 agents.

4.2.1. Coverage Metrics

Coverage is a significant metric for exploration. The coverage percentage $CP(t)$ at time t is defined as

$$CP(t) = \frac{N_{cc}(t)\rho^2}{AA} \tag{4.12}$$

where $N_{cc}(t)$ is the number of covered cells in the agent's map. Note that when there are pose estimation or distance sensor inaccuracies, cells outside the map borders or in IA might be added to an agent's map. These are counted towards N_{cc}. Cells merged by our algorithm are

divided into cells of size ρ.

We also compare the coverage metric of the final gathered map from the agent, in which we do ignore cells outside the arena, and in IA:

$$CP_m = \frac{N_{mc}\rho^2}{AA} \tag{4.13}$$

where N_{mc} is the number of covered cells inside the map boundaries and outside IA in the map.

To compare which configuration results in the fastest increase in coverage, we also include the average coverage percentage during the mission, ACP:

$$ACP = \frac{\sum_{t=0}^{T_{end}} CP(t)}{T_{end}} \tag{4.14}$$

As officially FSP considers cells with fully evaporated pheromones uncovered, we include previously covered but evaporated pheromone-cells in N_{cc} for a fair comparison.

4.2.2. Confidence Metrics

For BICLARE it is interesting to observe the average certainty at time t, $AC(t) = |\varphi(z,t) - 0.5|$. To compare which configuration can ensure the highest overall certainty, we define the mean average certainty during the entire mission:

$$AAC = \frac{\sum_{t=0}^{T_{end}} AC(t)}{T_{end}} \tag{4.15}$$

4.2.3. Mapping accuracy

To evaluate the accuracy of the created maps, we use the precision and recall metrics. Here, we define cell z in a generated map to be an obstacle cell if $\varphi(z,t) < 0.5$, and free if $\varphi(z,t) \geq 0.5$. Splitting the actual map up in cells of size ρ, we define cells to be truly occupied when obstacles cover 10% of their area. Given the number of correctly identified obstacle cells TP, falsely identified obstacle cells FP, and falsely identified free cells FN:

$$\text{precision} = \frac{TP}{TP + FP} \tag{4.16}$$

and

$$\text{recall} = \frac{TP}{TP + FN} \tag{4.17}$$

To determine TP, FP, and FN, only cells in N_{cc} that were at any point reachable during the experiment are considered. To get a more general impression of the accuracy of the created map for comparison between algorithms, we use the F_1-score, which is the harmonic mean of precision and recall:

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.18}$$

For each experiment, we take the map of the agent who finished first ($S_i = 4$), at its finish time.

4.2.4. Obstacle avoidance

We report the number of agent-obstacle collisions and the number of inter-agent collisions. The former shows the average collisions with static or dynamic obstacles per agent. The latter is also shown on average per agent, meaning that two agents colliding is one inter-agent collision per agent.

4.2.5. Efficiency

To determine the efficiency of our algorithm, we compare the total travelled path and estimated battery usage by each agent throughout the mission. The battery estimation is based on the estimated power drawn by the motors and from sending and exchanging messages. We also track the distribution of observations throughout the map by counting the number of times each cell is observed. One observation of a cell is defined as a sensor ray crossing or being intersected in that cell. To determine the memory efficiency of our quadtree, we compare the number of quadtree nodes with the number of cells in the map if it were a matrix. .

4.2.6. Return rate

We report the number of swarm agents that successfully returned to the deployment site. We define an successful return for agent A_i when $\|p_i - p_{i,t_0}\| < 2\text{m}$.

5 Results

For each metric, the average was taken over all repetitions for each configuration. The standard deviation over all repetitions was also calculated. Plots show the standard deviations over the repetitions per experiment if not stated otherwise. When aggregating over one or more variables, we show the pooled standard deviation where applicable.

5.1. Experiment 1: Varying sensor error

We ran experiments with sensor error levels $f_e \in \{0, 0.5, 1, 1.5\}$, and evaluated the performance of our algorithm with these values.

Parameter	Values
R_{comm}	[∞]
P_{loss}	[0]
T_{φ}	[100]
T_{spawn}	[100,180,∞]
f_e	[0,0.5,1,1.5]
R_f	[∞]
N_f	[∞]
N_s	[∞]

Table 5.1: Variable parameter combinations for experiment 1

Aggregating results over all maps, N_A and T_{spawn}, we can see from Figs. 5.1 and 5.2 that all algorithms achieve similar CP_m and $F1$ when $f_e = 0$, except for FSP-QT. However, when the sensor error increases, the scores of FSP-G and FSP decrease at a higher rate than BICLARE. Both BICLARE algorithms and FSP-QT have high $F1$ scores with low variance for each value of f_e. We can see that BICLARE-PP and BICLARE-WF perform similarly, with greatly overlapping pooled standard deviations. The same observation can be made of FSP and FSP-G. From Fig. 5.3 we can observe that BICLARE ensures a higher return rate on average. Interestingly, at higher f_e values, FSP has increased the average return rate. However, there is a significantly increased variance.

In Fig. 5.3, we can see that BICLARE-PP ensures the average highest return rate of robots.

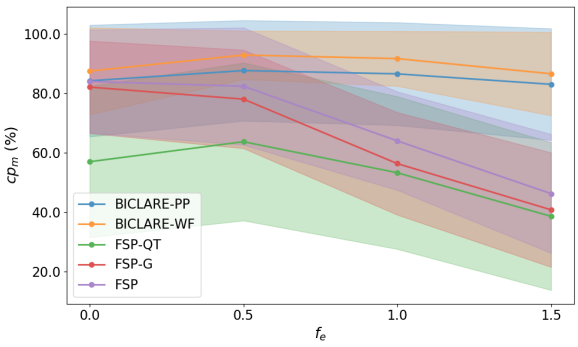


Figure 5.1: CP_m as a function of f_e, averaged over all maps, N_A, and T_{spawn}.

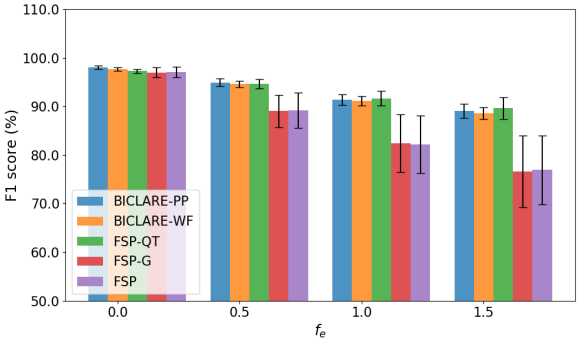


Figure 5.2: $F1$ as a function of f_e, averaged over all maps, N_A, and T_{spawn}.

While the FSP algorithms already have higher pooled standard deviations at $f_e = 0$, for all algorithms, the variance of the return rate increases with the noise.

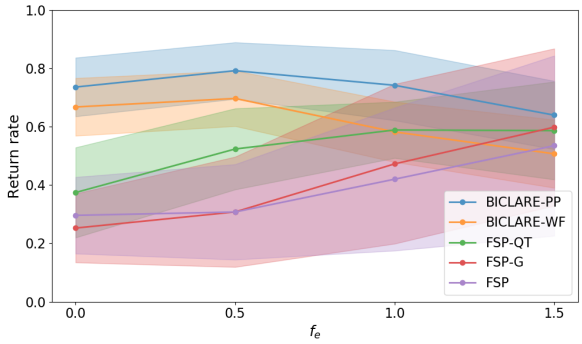


Figure 5.3: Average return rate as a function of f_e, averaged over all maps, N_A, and T_{spawn}.

We can see from Figs. 5.4 and 5.5 that BICLARE tends to have further traveling agents and higher battery usage. However, all algorithms have a significant pooled standard deviation for these metrics.

Fig. 5.6 shows three maps from experiments with the highest sensor error level $f_e = 1.5$, for different swarm sizes and dynamic object inter-arrival times.

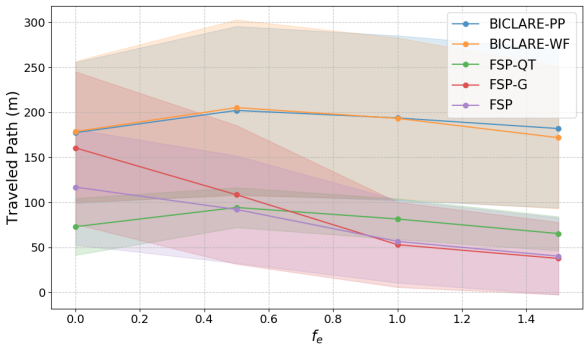


Figure 5.4: Traveled path as a function of f_e, averaged over all maps, N_A, and T_{spawn}.

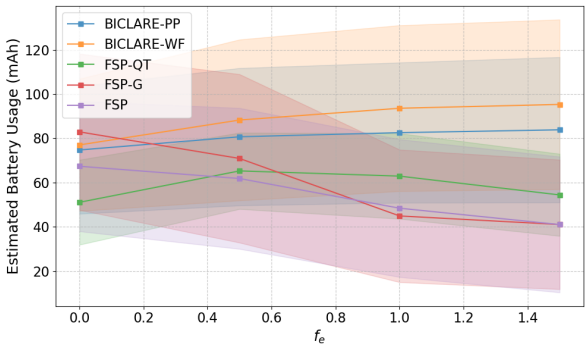


Figure 5.5: Estimated battery usage as a function of f_e, averaged over all maps, N_A, and T_{spawn}.

(a) HOUSE_TILTED with $N_A = 2$ and $T_{spawn} = 0$. $F1$ score is 87.2% and CP_m is 78.9%.

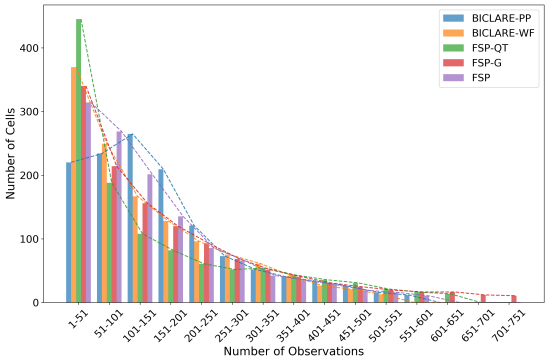
(b) OFFICE with $N_A = 6$ and $T_{spawn} = 180$. $F1$ score is 88.8% and CP_m is 91.9%.

(c) MUSEUM_TILTED with $N_A = 15$ and $T_{spawn} = 100$. $F1$ score is 89.6% and CP_m is 99.8%.

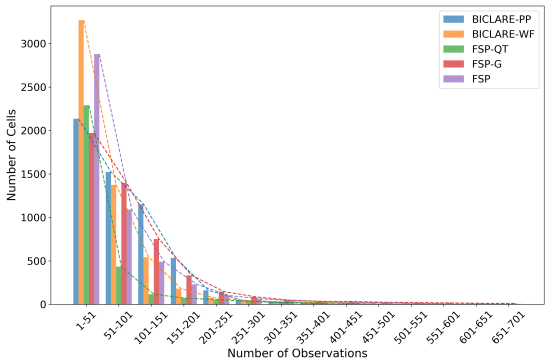
Figure 5.6: Some maps extracted from certain experiments with $f_e = 1.5$

In Fig. 5.7 we compare the algorithms by looking at the observation count distribution for the cells in the agent's map, aggregated over all values of T_{spawn}. To enhance visual clarity, we truncated the right tail of the distribution by removing all bins after the last bin with a weight larger than 10 cells. This excluded up to 8% of cell counts. Fig. 5.7a shows that, as opposed to the other algorithms, the BICLARE-PP algorithm resulted in the highest frequency of cells, not in the lowest observation bin (1-51), but rather in the 101-151 bin. This seems to happen for all $N_A > 2$ configurations in both HOUSE and HOUSE_TILTED. In no other map or algorithm is this observed. FSP-QT shows the steepest slopes between the bins. In Fig. 5.7b, we see that BICLARE-PP and FSP-G have observably smaller slopes between adjacent bins than BICLARE-WF and FSP. We observed that this slope difference decreased when the swarm size increased. Finally, in 5.7c, we do observe similar slopes. We observed similar trends for the other experiments in both MUSEUM and MUSEUM_TILTED. However, with lower swarm sizes, BICLARE-PP resulted in lower slopes and a lower total amount of cell observations. FSP-QT had very few cell observations overall. In Fig. 5.8, we can see that FSP-QT got some of the agents stuck in some corners, reducing the

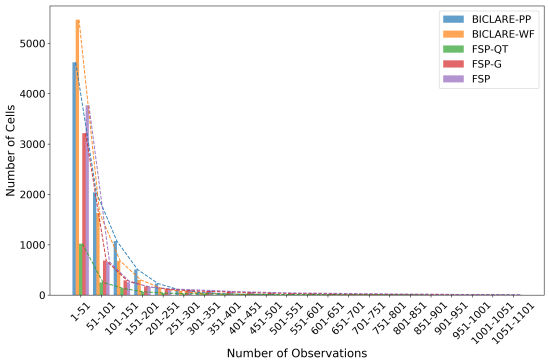
overall cell observations and coverage. The agents with BICLARE-PP resulted in much more balanced cell observations.



(a) HOUSE_TILTED, $N_A = 15$, $f_e = 0$

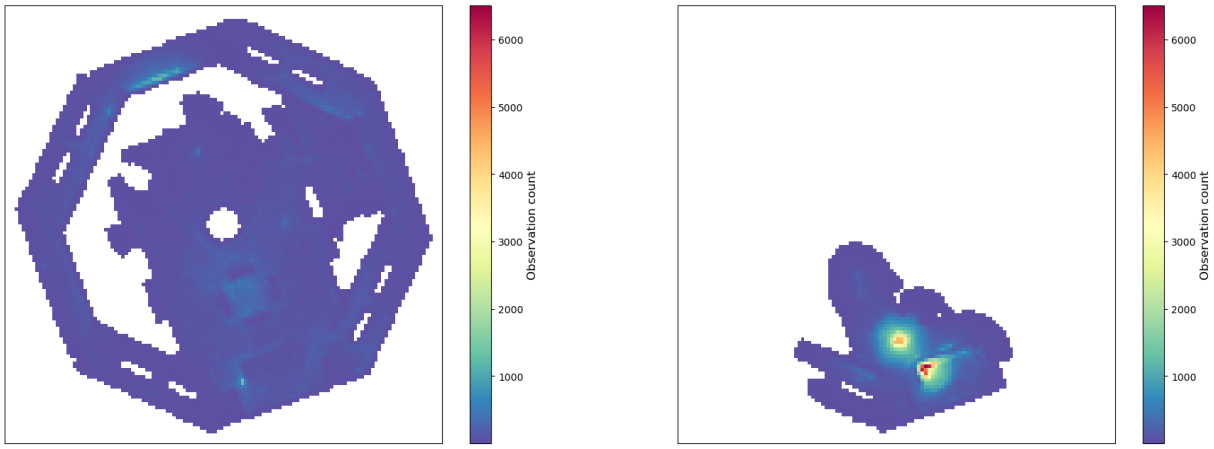


(b) OFFICE, $N_A = 4$, $f_e = 0$



(c) MUSEUM_TILTED, $N_A = 6$, $f_e = 1$

Figure 5.7: Cell observation count distribution. Averaged over T_{spawn}.



(a) BICLARE-PP

(b) FSP-QT

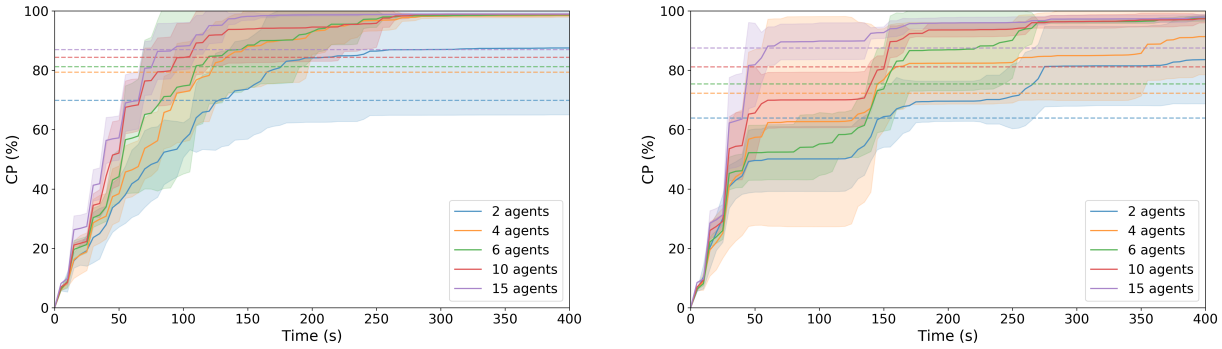
Figure 5.8: Heatmaps of cell observations for one of the experiments with $f_e = 0$, $T_{spawn} = 0$, and $N_A = 6$.

Table 5.2 shows the ACP and AAC for $f_e = 0$ across all maps and for all swarm sizes. We can see that the _TILTED variants of the maps often result in lower ACP scores for all algorithms.

In turn, AAC scores show the opposite relation. We also clearly observe that the number of agents in the swarm greatly influences both ACP and AAC. FSP-QT performs evidently worse than its FSP counterparts. BICLARE-PP and BICLARE-WF also mostly ensure higher AAC values than FSP-QT, even at similar ACP scores. Overall, BICLARE-PP always ensures the highest AAC, and BICLARE-WF most often has the highest ACP score. Fig. 5.9 shows $CP(t)$ up to T_{end} for BICLARE-WF and FSP. BICLARE-WF ensures a much smoother increase of CP than FSP. The latter seems to increase coverage in intervals.

Table 5.2: Performance of different algorithms across all maps and swarm sizes, for $f_e = 0$.

Map	N_A	ACP					AAC		
		BICLARE-PP	BICLARE-WF	FSP-QT	FSP-G	FSP	BICLARE-PP	BICLARE-WF	FSP-QT
HOUSE	2	**69.40**	66.68	49.12	61.19	61.68	**0.2023**	0.2001	0.1543
	4	**82.74**	78.30	68.23	70.00	69.33	**0.2450**	0.2259	0.1849
	6	**84.78**	84.41	75.09	75.77	77.05	**0.2733**	0.2420	0.1977
	10	84.65	85.67	80.47	84.49	**86.71**	**0.3053**	0.2769	0.2551
	15	**88.73**	87.77	84.56	85.44	87.47	**0.3203**	0.2896	0.2812
HOUSE_TILTED	2	61.01	**69.86**	43.69	56.87	63.91	**0.2145**	0.2002	0.1979
	4	75.80	**79.37**	56.19	71.58	72.29	**0.2470**	0.2342	0.2191
	6	80.11	81.25	63.43	64.11	**75.41**	**0.2730**	0.2550	0.2423
	10	**84.86**	84.34	70.35	77.09	81.13	**0.3071**	0.2772	0.2676
	15	**87.27**	86.99	74.63	83.83	87.48	**0.3195**	0.2850	0.2867
OFFICE	2	53.92	**57.30**	44.51	46.25	50.83	**0.1660**	0.1337	0.0915
	4	67.42	**71.77**	46.31	59.04	57.78	**0.2006**	0.1747	0.1007
	6	73.09	**79.32**	61.54	64.25	77.50	**0.2192**	0.1815	0.1443
	10	83.45	83.38	65.01	79.74	**82.05**	**0.2402**	0.2033	0.1839
	15	85.48	**88.94**	69.31	83.88	85.70	**0.2688**	0.2201	0.2244
OFFICE_TILTED	2	45.07	**58.64**	44.41	46.86	32.53	**0.1808**	0.1473	0.0971
	4	49.75	56.43	50.11	51.34	**57.84**	**0.2163**	0.1732	0.1594
	6	63.49	**71.34**	53.18	66.97	61.59	**0.2257**	0.1859	0.1676
	10	71.84	**80.88**	56.03	72.96	72.22	**0.2543**	0.2236	0.2015
	15	83.63	**86.32**	63.14	80.34	80.57	**0.2653**	0.2257	0.2126
MUSEUM	2	23.52	32.05	10.63	**37.70**	28.38	**0.1161**	0.0988	0.1055
	4	37.90	49.41	21.80	41.98	**54.26**	**0.1246**	0.1006	0.0956
	6	50.59	**66.90**	21.59	60.86	59.26	**0.1435**	0.1191	0.1068
	10	65.76	65.63	24.85	61.48	**70.62**	**0.1612**	0.1387	0.1180
	15	71.99	**75.52**	39.41	77.58	75.12	**0.1841**	0.1497	0.1245
MUSEUM_TILTED	2	24.47	30.81	12.39	29.69	**31.63**	**0.1297**	0.1118	0.1096
	4	37.47	41.72	16.49	45.82	**51.68**	**0.1451**	0.1229	0.1257
	6	46.49	**63.00**	28.44	47.66	57.10	**0.1538**	0.1361	0.1114
	10	57.32	**65.91**	30.01	66.06	68.27	**0.1748**	0.1434	0.1142
	15	69.12	**74.24**	40.99	65.72	80.32	**0.1948**	0.1579	0.1360



(a) BICLARE-WF **(b)** FSP

Figure 5.9: CP over time in HOUSE_TILTED with $f_e = 0$ and $T_{spawn} = 0$. The dotted lines indicate ACP.

Fig. 5.10 shows the average maximum number of nodes and leaves that were in each agent's quadtree throughout the experiments for BICLARE-PP. We chose the experiments with $f_e = 1.5$

and $T_{spawn} = 100$ as those settings result in the least amount of quadtree merging. Furthermore, the swarm size with the highest average max leaves and nodes was chosen for each map. The number of leaves and nodes is compared to the number of cells in a matrix, like one constructed with FSP. Note that FSP uses two matrices, one for coverage and one for obstacles. We can see that, especially in OFFICE, BICLARE's quadtree stores considerably fewer leaves and nodes than is required if a matrix is used.

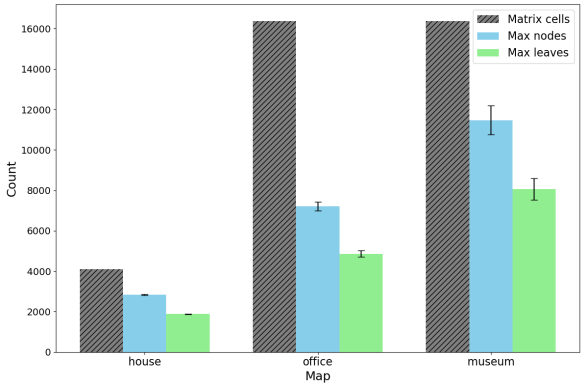


Figure 5.10: The maximum number of leaves and nodes for maps HOUSE with $N_A = 10$, OFFICE with $N_A = 15$, and MUSEUM with $N_A = 15$. $f_e = 1.5$ and $T_{spawn} = 100$.

We can see from Table 5.3 that BICLARE-PP, BICLARE-WF, and FSP-QT collide with more obstacles than FSP-G and FSP. This table shows the average number of collisions with obstacles and with other agents for each sensor error level, averaged over all maps, swarm sizes, and dynamic obstacle inter-arrival times. We observe significantly more collisions with obstacles than with other agents.

BICLARE-PP		
$\mathbf{f_e}$	Obstacle	Agent
0.0	1.217 ± 0.380	0.009 ± 0.038
0.5	1.536 ± 0.474	0.005 ± 0.023
1.0	3.194 ± 1.072	0.005 ± 0.019
1.5	6.954 ± 3.453	0.011 ± 0.025
BICLARE-WF		
$\mathbf{f_e}$	Obstacle	Agent
0.0	2.646 ± 1.602	0.048 ± 0.110
0.5	3.336 ± 2.287	0.025 ± 0.061
1.0	7.811 ± 4.449	0.025 ± 0.057
1.5	13.133 ± 6.877	0.028 ± 0.052
FSP-QT		
$\mathbf{f_e}$	Obstacle	Agent
0.0	1.331 ± 0.830	0.037 ± 0.113
0.5	1.290 ± 0.926	0.017 ± 0.053
1.0	1.952 ± 1.254	0.020 ± 0.074
1.5	2.685 ± 1.677	0.031 ± 0.087
FSP-G		
$\mathbf{f_e}$	Obstacle	Agent
0.0	0.044 ± 0.032	0.142 ± 0.544
0.5	0.052 ± 0.047	4.430 ± 14.031
1.0	0.044 ± 0.041	0.192 ± 0.368
1.5	0.031 ± 0.038	0.064 ± 0.136
FSP		
$\mathbf{f_e}$	Obstacle	Agent
0.0	0.055 ± 0.051	0.056 ± 0.231
0.5	0.061 ± 0.054	3.146 ± 10.645
1.0	0.039 ± 0.032	0.082 ± 0.178
1.5	0.040 ± 0.054	0.014 ± 0.042

Table 5.3: Average number of collisions of agents with obstacles and agents. Averaged over all maps, N_A, and T_{spawn}.

We measured the amount of collisions with BICLARE-PP when $R_p = \rho + 0.3362$ and $R_p = \rho + 0.4362$ in Table 5.4. We can see that the number of collisions with obstacles slightly decreased, while the inter-agent collisions slightly increased.

BICLARE-PP		
$\mathbf{f_e}$	Obstacle	Agent
0.0	0.785 ± 0.324	0.030 ± 0.078
0.5	1.150 ± 0.301	0.024 ± 0.074
1.0	2.900 ± 0.918	0.017 ± 0.046
1.5	5.449 ± 2.288	0.024 ± 0.067

Table 5.4: Average number of collisions of agents with obstacles and agents for BICLARE-PP with $R_p = \rho + 0.3362$. Averaged over all maps, N_A, and T_{spawn}.

5.2. Experiment 2: Computation-saving parameters

In this experiment, we change the frontier search range R_f, the maximum frontier regions N_f, and the maximum route length N_s. These values save computation time, and we evaluate their

effects. We evaluate combinations of the parameters noted in Table 5.5.

Parameter	Values
R_{comm}	[∞]
P_{loss}	[0]
T_{φ}	[100]
T_{spawn}	[100,180,∞]
f_e	[0,1]
R_f	[5,15,∞]
N_f	[20,∞]
N_s	[30,∞]

Table 5.5: Variable parameter combinations for experiment 2

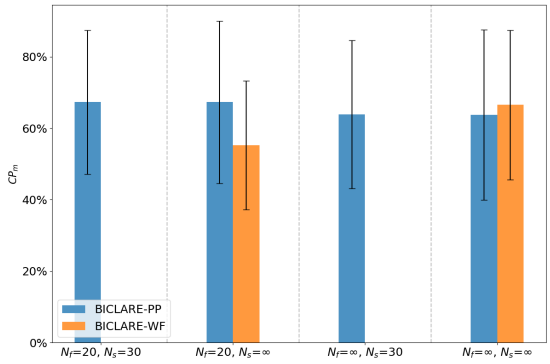
We tested the effect of R_f on the CP_m and $F1$ performance of all algorithms. For the BICLARE algorithms, N_f and N_s were set to ∞ to isolate the effect. The results are displayed in Table 5.6. We can see that for the smaller-sized map HOUSE_TILTED with $N_A = 2$, both BICLARE algorithms and FSP-QT show a slightly decreased CP_m score when $R_f = 15$ compared to $R_f = 5$, while FSP-G shows a slight increase. For six agents, both BICLARE algorithms show an increase from $R_f = 5$ to 15, with a more prominent change from BICLARE-WF. The FSP-based algorithms show no consistent trends. For 15 agents, there is a negligible change across all algorithms. The $F1$ score also shows insignificant change. For our medium-sized OFFICE_TILTED map, BICLARE-WF's CP_m score peaks when $R_f = 15$ and $N_A = 2$. The FSP algorithm's coverage seems to decrease when R_f increases. However, with $N_A = 6$, both BICLARE algorithms and FSP-QT have the lowest score with $R_f = 15$, while the performance of the other FSP algorithms increases. With a swarm size of 15, the coverage score increases with R_f for the BICLARE algorithms, FSP-G, and FSP. FSP-QT's score decreases when R_f increases. With our largest-sized MUSEUM_TILTED map, both BICLARE algorithms showed peak coverage when $R_f = 15$ with 2 and 6 agents, but the FSP algorithms show decreased performance with increased R_f. With 15 agents, FSP-QT shows decreased coverage while the other algorithms have increased coverage with increased R_f. Overall, the $F1$ score showed very little variability ($\leq 0.63\%$).

We can see from Fig. 5.11 that the ∞ assignment to N_f and N_s results in lower scores in both coverage and accuracy for BICLARE-PP than when they are 20 and 30, respectively. However, all values fall within each other's pooled standard deviation. For BICLARE-WF, $N_f = \infty$ ensures higher coverage, outperforming BICLARE-PP, but a similar $F1$ score than when $N_f = 20$.

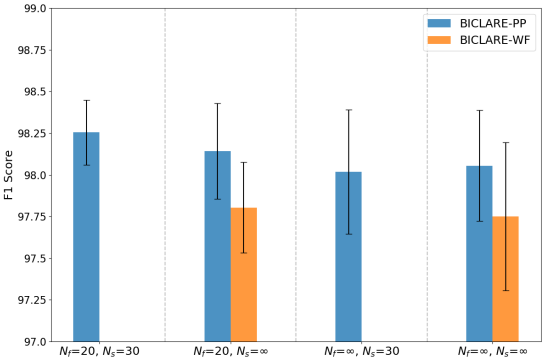
Overall, higher standard deviations were observed for lower swarm sizes. Interestingly, $f_e = 1$ resulted in a lower standard deviation than no sensor errors throughout all maps and configurations.

N_A	R_f	Map	CP_m					F1 score				
			BICLARE-PP	BICLARE-WF	FSP-QT	FSP-G	FSP	BICLARE-PP	BICLARE-WF	FSP-QT	FSP-G	FSP
2	5	HOUSE_TILTED	88.37	92.24	59.55	73.12	84.66	97.55	97.57	96.54	95.69	95.57
		OFFICE_TILTED	65.95	84.82	44.70	63.11	42.49	98.25	97.55	97.89	97.70	97.50
		MUSEUM_TILTED	32.31	52.07	13.79	41.87	44.05	97.95	97.11	97.62	97.90	98.10
	15	HOUSE_TILTED	83.42	87.92	52.88	77.41	84.52	97.49	97.26	97.05	95.65	95.59
		OFFICE_TILTED	65.22	86.01	49.79	59.01	41.32	98.25	97.79	97.47	97.58	97.62
		MUSEUM_TILTED	41.58	52.23	13.50	40.68	38.65	98.33	97.25	97.88	98.02	98.11
	∞	HOUSE_TILTED	83.42	87.92	52.88	77.41	84.52	97.49	97.26	97.05	95.65	95.59
		OFFICE_TILTED	65.93	85.98	49.20	59.01	41.32	98.22	97.73	97.60	97.58	97.62
		MUSEUM_TILTED	33.68	51.60	13.50	40.68	38.65	98.08	97.36	97.88	98.02	98.11
6	5	HOUSE_TILTED	98.32	94.59	78.87	76.84	98.75	98.22	97.96	97.30	95.29	95.44
		OFFICE_TILTED	90.86	97.79	64.18	79.30	78.94	98.47	98.27	98.01	97.33	97.36
		MUSEUM_TILTED	67.37	67.87	33.87	58.15	80.16	98.50	97.68	97.42	97.88	97.59
	15	HOUSE_TILTED	98.78	98.65	78.20	77.96	98.56	98.26	97.97	97.16	95.44	95.39
		OFFICE_TILTED	89.55	96.67	61.77	87.72	82.69	98.34	98.33	98.08	97.28	97.20
		MUSEUM_TILTED	70.58	91.14	30.41	65.81	86.70	98.37	97.74	97.80	97.67	97.45
	∞	HOUSE_TILTED	98.78	98.65	78.20	77.96	98.56	98.26	97.97	97.16	95.44	95.39
		OFFICE_TILTED	90.77	97.12	61.81	87.72	82.69	98.31	98.36	98.02	97.28	97.20
		MUSEUM_TILTED	68.03	88.99	32.54	65.51	85.08	98.45	97.92	97.81	97.69	97.49
15	5	HOUSE_TILTED	99.14	98.90	93.05	99.32	98.49	98.52	97.92	97.53	95.29	95.58
		OFFICE_TILTED	96.84	97.82	81.27	97.64	97.73	98.48	98.53	98.36	97.03	97.02
		MUSEUM_TILTED	93.96	95.76	65.52	91.61	99.17	98.63	98.03	98.11	97.33	97.24
	15	HOUSE_TILTED	99.14	98.88	89.75	98.81	98.32	98.56	97.74	97.46	95.52	95.60
		OFFICE_TILTED	98.04	97.87	78.35	97.79	98.32	98.52	98.43	98.05	96.99	97.01
		MUSEUM_TILTED	92.90	96.92	50.17	92.80	99.72	98.68	98.18	97.48	97.33	97.22
	∞	HOUSE_TILTED	99.14	98.88	89.75	98.81	98.32	98.56	97.74	97.46	95.52	95.60
		OFFICE_TILTED	98.05	98.00	73.65	97.79	98.32	98.54	98.46	98.17	96.99	97.01
		MUSEUM_TILTED	96.63	97.63	49.50	93.00	99.48	98.68	98.11	97.53	97.34	97.28

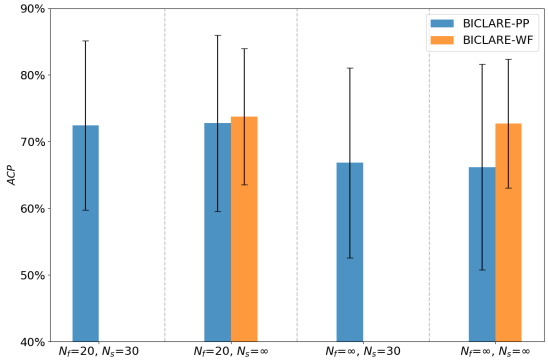
Table 5.6: Comparison of the effect of R_f on CP_m and F1 score metrics across different environments. $f_e = 0$ and $T_{spawn} = 0$. Standard deviations are between 0.04 and 1.19 for CP_m and 0.09 and 0.33 for F1 score.



(a) CP_m for $N_A = 2$



(b) $F1$ score for $N_A = 2$



(c) ACP for $N_A = 6$

Figure 5.11: CP_m and $F1$-score for $f_e = 0$, $T_{spawn} = 0$ and $R_f = \infty$, averaged over all maps. As N_s is only applicable when PeriPlanner is used, BICLARE-WF shows no value when $N_s = 30$.

5.3. Experiment 3: Environment volatility and pheromone evaporation

In this set of experiments, we observed the effects of different inter-arrival times of new obstacles and different evaporation rates. We evaluated a variety of combinations from Table 5.7.

Parameter	Values
R_{comm}	[∞]
P_{loss}	[0]
T_{φ}	[50,100,150,∞]
T_{spawn}	[100,180,∞]
f_e	[0,1]
R_f	[∞]
N_f	[∞]
N_s	[∞]

Table 5.7: Variable parameter combinations for experiment 3

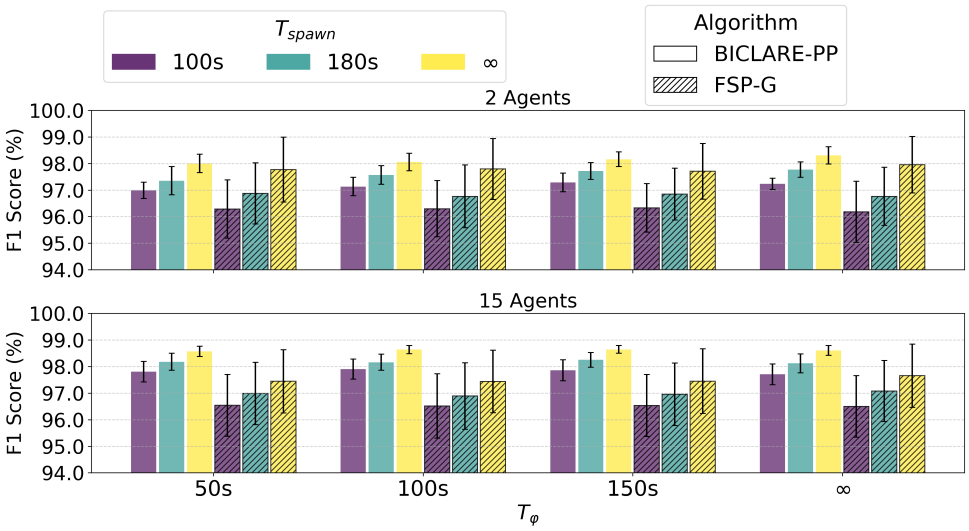
Fig. 5.12 shows the effects of T_{spawn} and T_{φ} on exploration performance with BICLARE-PP and FSP-G. We can see that a faster-changing environment (lower T_{spawn} value) results in a slightly lower $F1$ score for both algorithms. We observe similar patterns, as shown, for $N_A = 4, 6, 10$, which show decreased differences with larger swarm sizes. T_{φ} seems to have little effect on $F1$ score for both algorithms. We observe minor increased $F1$ performance with $N_A = 15$ when $T_{spawn} = 100$ and $T_{\varphi} = 100$ with BICLARE-PP compared to the algorithm's other results, but this increase falls well within the pooled standard deviations. However, CP_m and ACP scores show slight improvement for BICLARE-PP with increasing evaporation times, but deterioration for FSP-G.

5.4. Experiment 4: Communication range and loss

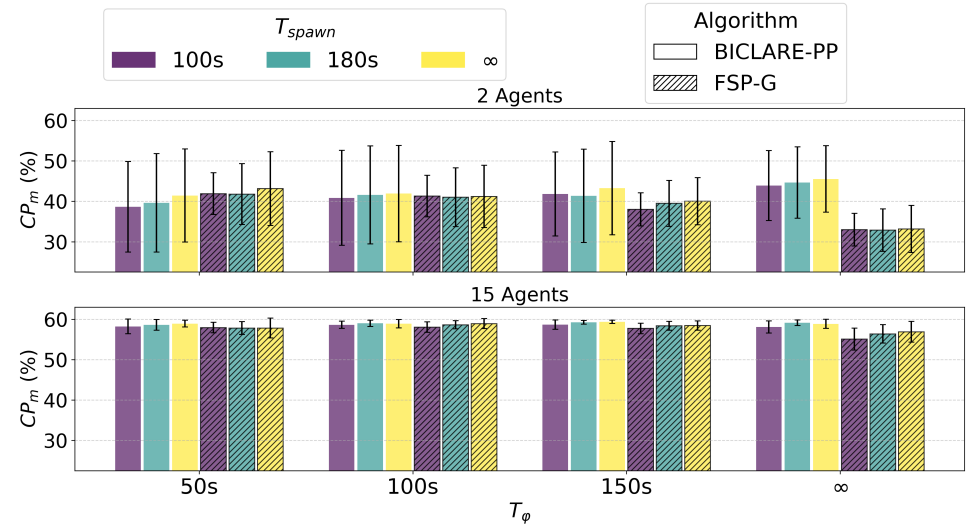
We tested the effects of communication range and message loss probability on performance, using the parameter values stated in Table 5.8.

Parameter	Values
R_{comm}	[5,10,15,∞]
P_{loss}	[0,0.25,0.5]
T_{φ}	[100]
T_{spawn}	[100,180,∞]
f_e	[0,1]
R_f	[∞]
N_f	[∞]
N_s	[∞]

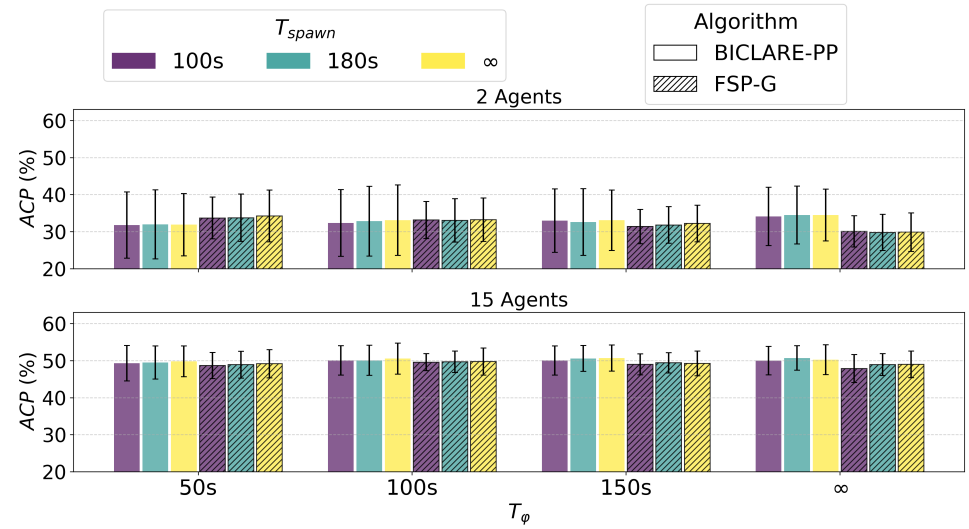
Table 5.8: Variable parameter combinations for experiment 4



(a) $F1$ score



(b) CP_m score



(c) ACP score

Figure 5.12: The effect of T_{spawn} and T_φ on $F1$, CP_m, and ACP score for $N_A = 2$ and $N_A = 15$ with $f_e = 0$. Plotted for BICLARE-PP and FSP-G. The scores are averaged over all maps. The error bars show the pooled standard deviation over the experiments for all maps.

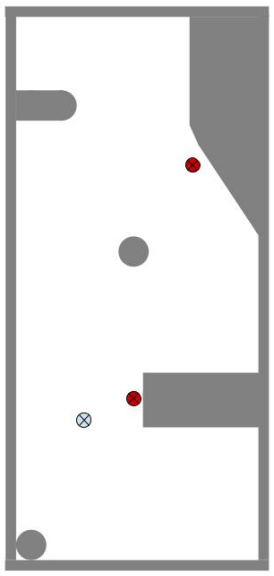
Table 5.9 shows the effects of R_{comm} and P_{loss} on $F1$, CP_m, and ACP score for algorithms BICLARE-PP and FSP-G. We tested with our most volatile environment configuration $T_{spawn} = 100$. We can see that P_{loss} has a slight positive effect on the $F1$ score for BICLARE-PP, being more evident when the swarm is larger. It has a slight negative effect on the $F1$ score for FSP-G. A larger P_{loss} also has a negative effect on CP_m and ACP scores for both algorithms, for all N_A values. Calculating the percentage decrease ($\frac{\text{score at} R_{comm}=5}{\text{score at } R_{comm}=\infty} - 1$) we see that BICLARE-PP shows a higher decrease in CP_m, with 5.23% on average, than FSP-G with 4.88%. Only at $N_A = 10$ and 15 BICLARE showed much less decrease (3.44% and 1.21% respectively) than FSP (6.09% and 12.11% respectively). The other metrics performed similarly. A larger R_{comm} shows no clear effect on the $F1$ score for either algorithm, but does show a clear positive effect on CP_m and ACP scores for both algorithms.

5.5. Experiment 5: Real-life proof-of-concept

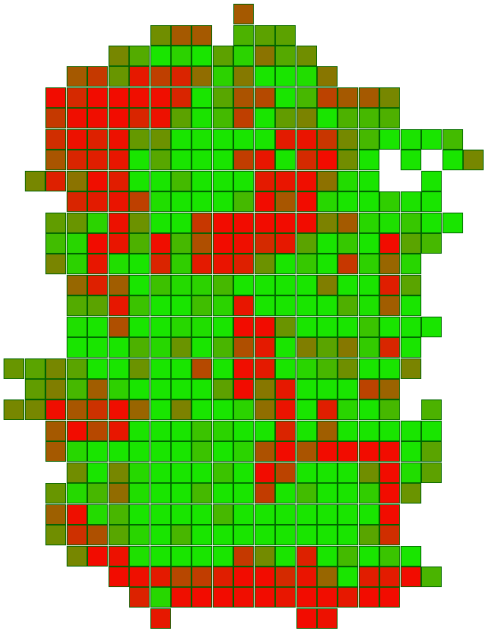
We ran a real-life experiment with four agents. Unfortunately, over time, three out of four agents stopped communicating after 16, 126, and 133 seconds, respectively. The first was removed from the map, while the others remained, functioning as dynamic obstacles. We observed that ultrasonic signals from other agents were picked up by an agent's own sensors, resulting in inaccurate readings. Fig. 5.13a shows the map including the final positions of the stranded agents. Fig. 5.13b and 5.13c show the surviving agent's confidence map at the end of the mission and the resulting final map, respectively. We can see that most of the map is covered, and the general outline of the map is clear. Throughout the mission, the surviving agent had zero collisions and successfully returned to its deployment position 34 seconds after the mission timer ended. The final map had a CP_m score of 87.0% and an $F1$ score of 84.6%.

N_A	R_{comm}	P_{loss}	F1 Score		CP_m		ACP	
			BICLARE-PP	FSP-G	BICLARE-PP	FSP-G	BICLARE-PP	FSP-G
2	5	0.0	96.87	96.30	58.01	62.18	42.03	45.94
		0.25	96.72	96.26	54.72	58.80	39.33	45.14
		0.5	97.25	96.06	57.08	57.75	40.74	42.83
	10	0.0	97.07	96.32	58.82	62.58	43.35	45.95
		0.25	97.03	96.23	58.56	59.60	41.89	45.56
		0.5	97.04	95.98	58.27	55.96	42.14	42.55
	15	0.0	97.09	96.27	60.71	61.87	44.35	46.11
		0.25	97.08	96.33	58.26	59.90	41.96	45.64
		0.5	97.10	95.97	58.00	55.59	42.55	42.49
	∞	0.0	97.13	96.30	61.69	62.54	44.73	46.29
		0.25	97.20	96.33	59.22	60.29	42.47	45.84
		0.5	97.08	96.04	57.95	56.16	42.43	42.68
4	5	0.0	97.45	96.43	72.44	75.31	53.10	54.08
		0.25	97.41	96.35	72.01	71.64	52.42	52.16
		0.5	97.29	96.24	68.42	71.29	50.24	50.52
	10	0.0	97.42	96.35	75.10	73.45	56.57	55.72
		0.25	97.53	96.24	74.55	74.87	56.48	55.28
		0.5	97.52	96.04	71.61	71.25	53.14	52.28
	15	0.0	97.59	96.42	77.02	73.65	57.29	55.51
		0.25	97.48	96.19	77.64	72.02	57.53	54.82
		0.5	97.41	96.16	75.25	71.74	54.94	53.48
	∞	0.0	97.57	96.37	77.60	73.40	57.09	55.43
		0.25	97.53	96.15	78.19	73.62	57.67	54.88
		0.5	97.54	95.99	75.00	70.79	54.70	53.33
6	5	0.0	97.61	96.40	81.14	77.21	61.34	56.04
		0.25	97.64	96.46	78.73	77.35	59.74	57.66
		0.5	97.39	95.96	75.84	76.55	58.10	56.02
	10	0.0	97.66	96.40	86.02	80.93	65.61	61.12
		0.25	97.62	96.23	84.47	82.94	63.85	61.42
		0.5	97.65	96.05	83.34	82.80	63.65	61.90
	15	0.0	97.58	96.45	84.29	81.40	64.91	61.96
		0.25	97.63	96.33	87.46	83.35	65.70	62.42
		0.5	97.64	95.99	83.91	82.42	64.25	63.34
	∞	0.0	97.61	96.49	86.18	80.45	66.03	61.95
		0.25	97.70	96.25	86.13	83.53	64.74	62.70
		0.5	97.65	96.05	85.94	82.75	64.76	63.47
10	5	0.0	97.78	96.69	91.41	86.03	69.76	67.18
		0.25	97.80	96.55	90.25	86.16	69.74	65.81
		0.5	97.68	96.15	88.86	83.70	66.97	61.28
	10	0.0	97.76	96.46	94.71	91.16	73.90	71.61
		0.25	97.86	96.45	93.16	89.99	73.75	71.06
		0.5	97.90	95.98	92.24	88.90	71.85	65.68
	15	0.0	97.79	96.48	93.65	92.38	73.53	72.16
		0.25	97.83	96.45	94.17	90.60	74.91	71.30
		0.5	97.91	95.96	93.81	89.46	73.22	67.07
	∞	0.0	97.75	96.47	94.23	92.79	74.26	73.10
		0.25	97.82	96.44	93.31	90.15	74.77	71.23
		0.5	97.87	95.89	92.62	89.58	72.63	67.15
15	5	0.0	98.04	96.88	96.40	86.72	77.56	70.96
		0.25	97.95	96.88	94.83	83.23	75.90	69.72
		0.5	97.99	96.21	96.54	82.12	75.96	64.74
	10	0.0	97.96	96.72	96.06	92.05	78.96	76.58
		0.25	97.97	96.56	96.96	91.71	79.55	74.81
		0.5	98.03	95.90	96.85	93.89	79.70	72.27
	15	0.0	97.89	96.61	97.31	93.98	79.62	77.44
		0.25	98.00	96.44	97.02	93.32	79.81	75.88
		0.5	98.07	95.74	97.65	93.94	79.24	72.62
	∞	0.0	97.90	96.51	97.27	96.10	80.07	79.14
		0.25	97.93	96.44	98.12	94.09	81.07	75.34
		0.5	98.04	95.69	95.96	96.65	78.25	73.70

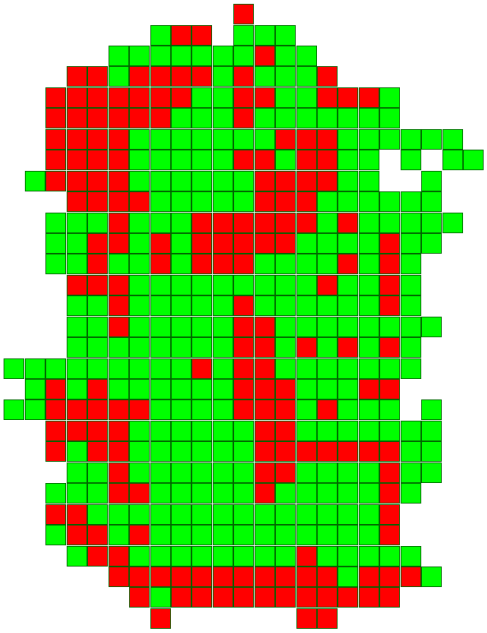
Table 5.9: Effect of R_{comm} and P_{loss} on $F1$, CP_m, and ACP score for different swarm sizes. Average over all maps. $f_e = 0$, $T_{spawn} = 100$



(a) Real final map including stranded robots (red), and the surviving robot (blue).



(b) Confidence map



(c) Final map

Figure 5.13: The actual map at the end of the real-life mission, the agent's confidence map, and the final inferred map for the real-life experiment.

6

Discussion

Both BICLARE algorithms ensure similar higher coverage overall and mapping accuracy compared to the FSP algorithms when sensor errors increase, achieving over 87% $F1$ score and 80% coverage on average. In the final maps, we can see that even in the noisiest areas, the general outline of the environment is clear. This shows the effectiveness of the confidence model in BICLARE's mapping algorithm. However, coverage shows high variability, where smaller swarm sizes achieve low coverage in big maps. BICLARE does result in much bigger total traveled paths and thus more battery usage than FSP. Battery usage increases with higher f_e values while the traveled path decreases. This might indicate robots grouping more, resulting in more exchanged messages. FSP showed a higher return rate with a higher f_e. This is because FSP also shows less coverage on average when sensor errors increase compared to BICLARE. This means the agents have displaced themselves less far from their initial deployment site, allowing easier return. So we can state that overall, BICLARE, especially BICLARE-PP, ensures a higher return rate of its swarm. We observe that all algorithms seem to follow a similar cell observation distribution. However, in one of the smaller-sized maps HOUSE_TILTED, BICLARE-PP's highest peak was not in the left-most bin. This indicates a higher level of re-exploration than the other algorithms. While this might affect total coverage, re-exploration is highly important in dynamic environments. Overall, having lower peaks indicates that fewer cells were explored; these agents spent most of the mission time close to walls or obstacles, limiting the number of cells covered by a single sensor measurement, as visible in Fig. 5.8. We have also seen that BICLARE's quadtree implementation saves considerable memory compared to FSP's 2-matrix implementation. The improvement is the most significant in our mid-sized OFFICE map. Overall, BICLARE-WF achieves higher coverage than BICLARE-PP, especially with bigger maps. However, BICLARE-PP ensures a higher return rate, suggesting that PERIPLANNER enables agents to consistently reach their targets.

From Table 5.2 we could see that the _TILTED variants of the maps resulted in lower AAC. This indicates that whenever the overall structure of the environment is not aligned with the map grid, as is often the case in real life, agents will be less confident about their surroundings. It is expected that AAC decreases whenever ACP increases, since N_{cc} is lower. We observe that FSP agents tend to increase their coverage in intervals. This, in turn, indicates intervals of re-exploration. However, from the $F1$ scores, we can conclude that there is no significant improvement in performance due to this re-exploration compared to the BICLARE algorithms.

The introduction of the quadtree seems to increase the number of collisions with obstacles significantly. Even with an increased obstacle avoidance radius, significantly more collisions

occur than with FSP and FSP-G. The confidence model does not appear to infer obstacles quickly enough, and the attempt to mitigate collisions by checking the front-facing sensor has not helped. If we compare the BICLARE algorithm collisions with FSP-QT, it may even contribute to additional collisions. The reduced coverage performance of FSP-QT compared to the other algorithms when R_f increases indicates that the confidence model causes agents to get stuck more often without wall-following or path planning. A lower R_f would result in closer targets, mitigating the issue. We further observe that reduced R_f increased BICLARE's performance in small maps, especially for small swarm sizes. The infinite frontier search range did not result in the best performance for any maps, and for medium- and large-sized maps, $R_f = 15$ performed the best.

Furthermore, when no sensor error is simulated, we can see that overall BICLARE achieves higher coverage than FSP for small and mid-sized maps, but scores significantly lower on large maps, especially with small swarm sizes. BICLARE-PP's improved performance when $N_f = 20$ and $N_s = 30$ indicates some flaws in BICLARE. These values result in not all target options being considered, proving that the frontier selection algorithm is not optimal. Most likely, this can be solved by re-tuning the parameters, especially k, l, m, and n. However, increased performance with lower values for R_f, N_f, and N_s does drastically reduce the worst-case time complexity of our algorithm, requiring fewer computational resources.

We have seen that BICLARE and FSP react to the volatility of the environment in a similar way, displaying a slight decrease in mapping accuracy when dynamic obstacles appear faster. Varying the evaporation time has little effect on coverage and mapping accuracy for any spawn rate of dynamic obstacles. This shows that even without evaporation, a high level of re-exploration takes place.

With bigger swarm sizes, a higher message loss probability resulted in better mapping accuracy performance. This indicates that new observations can be dominated by older information echoed between the agents, and thus, either received information should be discounted more, or alternate implementations for handling information by other agents should be explored. This suggests that less communication might increase reactability to environmental changes [40]. However, our experiments did not show increased mapping accuracy in our most dynamic experiments when the communication range decreased. Increased communication range positively affects the coverage for both BICLARE and FSP. This can be caused by the information mentioned above echoing, which might result in lower confidence, triggering re-exploration of the corresponding places. BICLARE performs better in scenarios with lower communication range than FSP for larger swarm sizes, whereas FSP performs better with smaller swarm sizes in these scenarios.

Our real-life proof of concept shows that the algorithm can successfully run on a low-cost Raspberry Pi Pico 2W in a small environment. However, performance seems inconsistent, with some robots failing during the experiment. We believe this is due to the limited available memory, even with our memory-efficient mapping implementation. Message exchange and handling, combined with the RTOS, required a significant portion of the available SRAM. Also, mapping errors due to interference from sensor signals originating from other agents represent a significant oversight in our implementation, and future work should account for these scenarios. Despite these complications, an adequate map of the environment was generated.

7

Conclusion

In this thesis, we proposed BICLARE, a bio-inspired collaborative and lightweight algorithm for robust exploration. Inspired by ant colony behaviour, BICLARE is designed for efficient exploration in complex scenarios to create maps for emergency responders, enabling them to understand and navigate unfamiliar terrain quickly. We designed our algorithm to be robust against sensor measurement errors and pose estimation errors common in the real world. It is also designed to work in dynamic environments like collapsing structures and to run on inexpensive hardware, making it suitable for disaster scenarios where limited budget and potential hardware losses are no rare occurrence. We compared BICLARE with the FSP algorithm by Tran et al., which provided the foundation of our implementation, demonstrating BICLARE's superior mapping accuracy in noisy conditions. It proved effective even in highly volatile and communication-limited scenarios. We presented the algorithm's potential on real-world hardware, showing adequate performance even with failing agents. Future work will focus on expanding the hardware trials, potentially slightly upgrading the microcontroller. It will also focus on handling interference between agents' sensors, for example, by varying the decrease of occupancy confidence when a cell is observed as occupied whenever neighbouring agents are present. Furthermore, future work will examine the potential domination of older information over new environmental changes. Finally, elaborate parameter optimization will be performed.

A

Simulation experiments constant parameter configuration

Parameter	Values
α_{φ}	0.05
ω^{a}	0
ω^{c}	0
ω^{s}	1.15
ω^{t}	0.3
Ω_{D}	1
Ω_{S}	0.1
ϕ_{Δ}	0.05
θ_{turn}	15
ζ	0.6
k	0.1
l	0.8
l_{free}	0.405465
l_{min}	-2.94444
l_{max}	2.19722

Parameter	Values
$l_{occupied}$	-0.619039
m	2.5
n	3
P_{o}	0.3
R_{a}	1
R_{c}	1.5
R_{d}	0.5
R_{o}	$\rho + 0.2362$
R_{p}	$\rho + 0.1362$
R_{s}	0.5
R_{sensor}	2
R_{x}	1
T_{f}	5
T_{map}	15
T_{sync}	5

B

Hardware experiment parameter configuration

Parameter	Values
α_{φ}	0.05
ω^{a}	0
ω^{c}	0
ω^{s}	1.15
ω^{t}	0.3
Ω_{D}	1
Ω_{S}	0.1
ϕ_{Δ}	0.05
θ_{turn}	20
ζ	0.6
k	0.1
l	0.8
l_{free}	0.120144
l_{min}	-2.94444
l_{max}	2.19722
$l_{occupied}$	-0.619039
m	1.5

Parameter	Values
n	3
N_{f}	20
N_{s}	30
P_{o}	0.3
R_{a}	1
R_{c}	1.5
R_{d}	0.5
R_{f}	3
R_{p}	$\rho + 0.205$
R_{o}	$\rho + 0.505$
R_{s}	0.5
R_{sensor}	1.5
R_{x}	1
T_{f}	5
T_{map}	10
T_{sync}	5

Bibliography

[1] Adafruit Industries. *Adafruit Brushless DC Motors 3777 Datasheet*. Accessed: 2025-04-01. 2025. URL: https://www.verical.com/datasheet/adafruit-brushless-dc-motors-3777-5912007.pdf.

[2] John Amanatides, Andrew Woo, et al. "A fast voxel traversal algorithm for ray tracing." In: *Eurographics*. Vol. 87. 3. 1987, pp. 3–10.

[3] Ralph Beckers, Jean-Louis Deneubourg, and Simon Goss. "Modulation of trail laying in the ant Lasius niger (Hymenoptera: Formicidae) and its role in the collective selection of a food source". In: *Journal of Insect Behavior* 6 (1993), pp. 751–759.

[4] Leonora Bianchi, Luca Maria Gambardella, and Marco Dorigo. "An ant colony optimization approach to the probabilistic traveling salesman problem". In: *Parallel Problem Solving from Nature—PPSN VII: 7th International Conference Granada, Spain, September 7–11, 2002 Proceedings 7*. Springer. 2002, pp. 883–892.

[5] Bitcraze AB. *Crazyflie 2.1*. Accessed: 2025-03-31. 2025. URL: https://www.bitcraze.io/products/old-products/crazyflie-2-1/.

[6] Bitcraze AB. *Loco Positioning System*. Accessed: 2025-03-31. 2025. URL: https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/.

[7] Oscar Bjurling et al. "Drone swarms in forest firefighting: A local development case study of multi-level human-swarm interaction". In: *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*. 2020, pp. 1–7.

[8] Carlo Pinciroli et al. *ARGoS: A Modular, Multi-Physics Simulator for Swarm Robotics*. Accessed: 2025-04-01. 2025. URL: https://www.argos-sim.info/index.php.

[9] Daniel Carrillo-Zapata et al. "Mutual shaping in swarm robotics: User studies in fire and rescue, storage organization, and bridge inspection". In: *Frontiers in Robotics and AI* 7 (2020), p. 53.

[10] Micael S Couceiro et al. "Darwinian swarm exploration under communication constraints: Initial deployment and fault-tolerance assessment". In: *Robotics and Autonomous Systems* 62.4 (2014), pp. 528–544.

[11] Rocco De Nicola, Luca Di Stefano, and Omar Inverso. "Multi-agent systems with virtual stigmergy". In: *Science of Computer Programming* 187 (2020), p. 102345.

[12] J -L Deneubourg et al. "The self-organizing exploratory pattern of the argentine ant". In: *Journal of insect behavior* 3 (1990), pp. 159–168.

[13] Leah Edelstein-Keshet, James Watmough, and G Bard Ermentrout. "Trail following in ants: individual properties determine population behaviour". In: *Behavioral Ecology and Sociobiology* 36 (1995), pp. 119–133.

[14] Firefighters and EMS Fund. *Fire Department Reductions Since January 2020*. May 2025. URL: https://www.fireandemsfund.com/fire-department-reductions/ (visited on 05/12/2025).

[15] Alex Fisher et al. "ColMap: A memory-efficient occupancy grid mapping framework". In: *Robotics and Autonomous Systems* 142 (2021), p. 103755.

[16] Gutama Indra Gandha and Dedi Nurcipto. "The Performance Improvement of the Low-Cost Ultrasonic Range Finder (HC-SR04) Using Newton's Polynomial Interpolation Algorithm". In: *Jurnal Infotel* 11.4 (2019), pp. 108–113.

[17] Cole Hengstebeck, Peter Jamieson, and Bryan Van Scoy. "Extending boids for safety-critical search and rescue". In: *Franklin Open* 8 (2024), p. 100160.

[18] Nancy G Hernadnez. "WestJEM Full-Text Issue". In: *Western Journal of Emergency Medicine: Integrating Emergency Care with Population Health* 18.6 (2017).

[19] Armin Hornung et al. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous robots* 34 (2013), pp. 189–206.

[20] Malek Itani et al. "Creating speech zones with self-distributing acoustic swarms". In: *Nature Communications* 14.1 (2023), p. 5684.

[21] Raphaël Jeanson, Francis LW Ratnieks, and Jean-Louis Deneubourg. "Pheromone trail decay rates on different substrates in the Pharaoh's ant, Monomorium pharaonis". In: *Physiological Entomology* 28.3 (2003), pp. 192–198.

[22] Albina Kamalova, Ki Dong Kim, and Suk Gyu Lee. "Waypoint mobile robot exploration based on biologically inspired algorithms". In: *IEEE Access* 8 (2020), pp. 190342–190355.

[23] Miquel Kegeleirs, David Garzón Ramos, and Mauro Birattari. "Random walk exploration for swarm mapping". In: *Annual conference towards autonomous robotic systems*. Springer. 2019, pp. 211–222.

[24] You-Dong Liang and Brian A. Barsky. "A new concept and method for line clipping". In: *ACM Transactions on Graphics (TOG)* 3.1 (1984), pp. 1–22.

[25] Raspberry Pi Trading Ltd. *Raspberry Pi Pico W and Pico 2 W Datasheet*. https://datasheets.raspberrypi.com/picow/pico-2-w-datasheet.pdf. Accessed May 11, 2025. 2024.

[26] Kimberly N McGuire et al. "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment". In: *Science Robotics* 4.35 (2019), eaaw9710.

[27] Márcio Mendonça et al. "Multi-robot exploration using dynamic fuzzy cognitive maps and ant colony optimization". In: *2020 IEEE international conference on fuzzy systems (FUZZ-IEEE)*. IEEE. 2020, pp. 1–8.

[28] Elijah J Morgan. "HC-SR04 ultrasonic sensor". In: *Nov* (2014).

[29] N.N. *Brandweer trekt aan de bel na mogelijke bezuinigingen: 'Voelen ons in de steek gelaten'*. May 2025. URL: https://www.ad.nl/rotterdam/brandweer-trekt-aan-de-bel-na-mogelijke-bezuinigingen-voelen-ons-in-de-steek-gelaten~ac52858a/ (visited on 05/12/2025).

[30] Amir M Naghsh et al. "Analysis and design of human-robot swarm interaction in firefighting". In: *RO-MAN 2008-The 17th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE. 2008, pp. 255–260.

[31] Jacques Penders et al. “A robot swarm assisting a human fire-fighter”. In: *Advanced Robotics* 25.1-2 (2011), pp. 93–117.

[32] Craig W Reynolds. “Flocks, herds and schools: A distributed behavioral model”. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, pp. 25–34.

[33] EJH Robinson et al. “Decay rates of attractive and repellent pheromones in an ant foraging trail network”. In: *Insectes sociaux* 55 (2008), pp. 246–251.

[34] Elva JH Robinson, Francis LW Ratnieks, and M Holcombe. “An agent-based model to investigate the roles of attractive and repellent pheromones in ant decision making during foraging”. In: *Journal of theoretical Biology* 255.2 (2008), pp. 250–258.

[35] ROBOTIS. *TurtleBot3 Features*. Accessed: 2025-03-12. n.d. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/.

[36] Muhammad Salman, David Garzón Ramos, and Mauro Birattari. “Automatic design of stigmergy-based behaviours for robot swarms”. In: *Communications Engineering* 3.1 (2024), p. 30.

[37] Naim Shandi, Jason M Merlo, and Jeffrey A Nanzer. “Decentralized picosecond synchronization for distributed wireless systems”. In: *IEEE Transactions on Communications* (2024).

[38] Andrew J Smith and Geoffrey A Hollinger. “Distributed inference-based multi-robot exploration”. In: *Autonomous Robots* 42.8 (2018), pp. 1651–1668.

[39] Jian Sun et al. “A novel UWB/IMU/odometer-based robot localization system in LOS/NLOS mixed environments”. In: *IEEE Transactions on Instrumentation and Measurement* (2024).

[40] Mohamed S Talamali et al. “When less is more: Robot swarms adapt better to changes with constrained communication”. In: *Science Robotics* 6.56 (2021), eabf1416.

[41] Guy Theraulaz and Eric Bonabeau. “A brief history of stigmergy”. In: *Artificial life* 5.2 (1999), pp. 97–116.

[42] Vu Phi Tran et al. “Dynamic frontier-led swarming: Multi-robot repeated coverage in dynamic environments”. In: *IEEE/CAA Journal of Automatica Sinica* 10.3 (2023), pp. 646–661.

[43] Vu Phi Tran et al. “Frontier-led swarming: Robust multi-robot coverage of unknown environments”. In: *Swarm and Evolutionary Computation* 75 (2022), p. 101171.

[44] Mark Van der Feyst, Eric Wissner, and James Petruzzi. *Residential Fire Rescue*. Fire Engineering Books, 2014.

[45] TWT Van Der Meer, E Verbree, and PJM Van Oosterom. “EFFECTIVE CARTOGRAPHIC METHODS FOR ASSISTING TACTICS CHOICE AND INDOOR DEPLOYMENTS DURING BUILDING FIRES–A CASESTUDY THE DUTCH FIRE BRIGADE”. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 42 (2018), pp. 655–660.

[46] K Vander Meer Robert and Leeanne E Alonso. “Pheromone directed behavior in ants”. In: *Pheromone communication in social insects*. CRC Press, 2019, pp. 159–192.

[47] Yiheng Wang, Alei Liang, and Haibing Guan. “Frontier-based multi-robot map exploration using particle swarm optimization”. In: *2011 IEEE symposium on Swarm intelligence*. IEEE. 2011, pp. 1–6.

[48] Stephanie Wendt, Nico Kleinhoelting, and Tomer J Czaczkes. “Negative feedback: ants choose unoccupied over occupied food sources and lay more pheromone to them”. In: *Journal of The Royal Society Interface* 17.163 (2020), p. 20190661.

[49] David Yanguas-Rojas et al. “Victims search, identification, and evacuation with heterogeneous robot networks for search and rescue”. In: *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*. IEEE. 2017, pp. 1–6.

[50] York Robotics Laboratory. *ARGoS3 Pi-Puck Repository*. Accessed: 2025-04-01. 2025. URL: https://github.com/yorkrobotlab/argos3-pipuck.

[51] Boyu Zhou, Hao Xu, and Shaojie Shen. “Racer: Rapid collaborative exploration with a decentralized multi-uav system”. In: *IEEE Transactions on Robotics* 39.3 (2023), pp. 1816–1835.