Research and Implementation of Decentralized Multi-Robot Coordination Methods Applied to Urban Search and Rescue

by

F. Pizarro Bejarano

Supervisor: Goldie Nejat

April 2021

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Abstract

Coordination is a fundamental component of multi-robot teams. However, coordination is challenging in urban search and rescue (USAR) operations due to limited communication. This thesis provides an overview of multi-robot coordination research and implements an algorithm capable of efficiently coordinating a team of robots exploring a space despite limited communication and danger to robots. First, existing research in the field of multi-robot coordination and exploration was analyzed to find an algorithm appropriate to USAR scenarios. This algorithm was then implemented and shown to be effective at coordinating exploration in USAR environments, regardless of communication levels. This implementation will serve as a benchmark with which to compare future multi-robot USAR exploration research.

Acknowledgements

This thesis confirmed in me a love of research and robotics. It has been the most academically satisfying and inspiring project in my undergraduate experience, one that has pushed me to undertake graduate research. I would like to firstly thank Dr. Goldie Nejat for accepting me as her student and working with me to choose a topic that interested me. Her drive for socially assistive robotics drew me to her research and motivated me to work my hardest on this thesis. Also, I would like to thank Aaron Tan for meeting with me regularly and giving me feedback on all my deliverables and presentations, regardless of their length. His advice was critical in completing my project and learning more about academic research. Thank you both. Your assistance and dedication have been instrumental in my thesis and my future as a robotics researcher.

Table of Contents

List of Figures

2 Literature Review

3 Implementation

4 Results and Discussion

1 Introduction

1.1 Background

Urban search and rescue (USAR) is a type of rescue operation requiring locating and extracting victims from collapsed structures [1]. Robots have been used to improve the efficacy of search and rescue operations while limiting the danger to human crisis managers [2, 3]. One use of robots is deploying a large team to map the area, detect hazards, and locate victims so human crisis managers can safely and efficiently extract them from the rubble [2].

Multi-robot teams are more effective and robust than single robots but coordinating groups of robots remains a challenge [2]. There are two main approaches to multi-robot coordination: centralized and decentralized architectures [4]. Centralized architecture involves using a central agent that has access to all the robots' information and environment data to make decisions and plan globally optimal search plans. However, this is difficult in USAR environments due to unreliable communication [22], ineffectual scaling to larger teams [4], and vulnerability to issues in the central agent [4]. In contrast, decentralized control allows each robot to plan its course. Coordination with other robots is done through observation and limited communication [4]. Decentralized methods are more adaptable to dynamic situations and robust to failure but lead to suboptimal solutions [4].

1.2 Objectives

Decentralized multi-robot coordination seeks to minimize overlapping efforts between robots to find the most optimal plan for finding victims despite lack of information. For scenarios like USAR where communication is unreliable [22], robots may need to infer the progress and goals of teammates from their current position and trajectory. USAR presents unique challenges for multi-robot coordination due to unreliable communication [22], high danger to robots [2], and the time pressure related to finding victims [2].

This thesis will review relevant research done in the field of decentralized multi-robot coordination, determine a suitable algorithm that can be applied to USAR, and evaluate its efficacy when applied to USAR. By measuring the efficacy of current research when applied to USAR, I hope to tackle the challenges of decentralized coordination in USAR and determine robust metrics to improve future algorithms. The candidate solution chosen to be implemented will serve as a baseline against which future multi-robot USAR work can be compared.

1.3 Approach

I will perform a literature review of relevant methods, focusing on decentralized multi-robot task allocation, decentralized multi-robot exploration, and implicit coordination. The most appropriate solution will be selected, and a measure of the efficacy of the algorithm will be determined. Finally, the algorithm will be applied to simulated teams of robots in cluttered environments similar to those found in USAR.

2 Literature Review

Various approaches to multi-robot exploration and task-allocation were reviewed to determine an appropriate candidate solution that will be applied to USAR. Each solution makes differing assumptions about the robot team, the availability of communication, and the specific task to be performed. The core assumption made for USAR is the lack of reliable communication between robots due to debris and radio channels being overwhelmed by emergency personnel [22]. Communication may be intermittent, have limited range, or be completely obstructed by walls and rubble [20, 22]. The candidate solution chosen should not only function when communication is unreliable but should be optimally suited for USAR operations.

To find such a solution, the field was divided using two categories: the type of decision-making architecture (centralized, decentralized, or hybrid) and the coordination strategy (explicit or implicit). Centralized architectures have a single agent that decides the goals of each robot in the team, compared to decentralized architectures where each robot decides its own goals [4]. Decentralized architectures are more robust to failure and unreliable communication [4] but must coordinate goals between robots to avoid duplicating work. This coordination can be explicit where the goals are clearly defined and the robots agree on how to distribute them [41], or implicit where the robots predict the goals of other robots and individually choose how to work together most efficiently [41]. Explicit coordination typically requires more communication as robots must negotiate and settle on a task allocation rather than sharing minimal information in implicit coordination strategies [41].

Since USAR operations suffer from unreliable communication [22] and a high risk of physical damage to robots [2], decentralized architectures are necessary. Implicit coordination strategies are preferred to further reduce dependence on communication. However, the entire state of the field was analyzed to better understand current research and possible solutions.

This literature review first discusses the differences in decision-making architectures in section **2.1 Decision-Making Architectures**. This discussion is broad since centralized architectures are not appropriate to USAR. Then, in sections **2.2 Explicit Coordination** strategies and **2.3 Implicit Coordination Strategies,** specific algorithms for decentralized coordination are considered. Finally, in section **2.4 Candidate Solution** a specific algorithm is chosen and discussed. This algorithm will be implemented in USAR scenarios to be used as a baseline for comparing the effectiveness of multi-robot decentralized USAR algorithms.

2.1 Decision-Making Architectures

There are two main approaches to coordinating teams of robots: centralized and decentralized. Additionally, several approaches fall somewhere in between and have been labelled hybrid architectures. There is a spectrum from completely centralized to completely decentralized, and due to unreliable communication in USAR operations, candidate solutions must be highly decentralized.

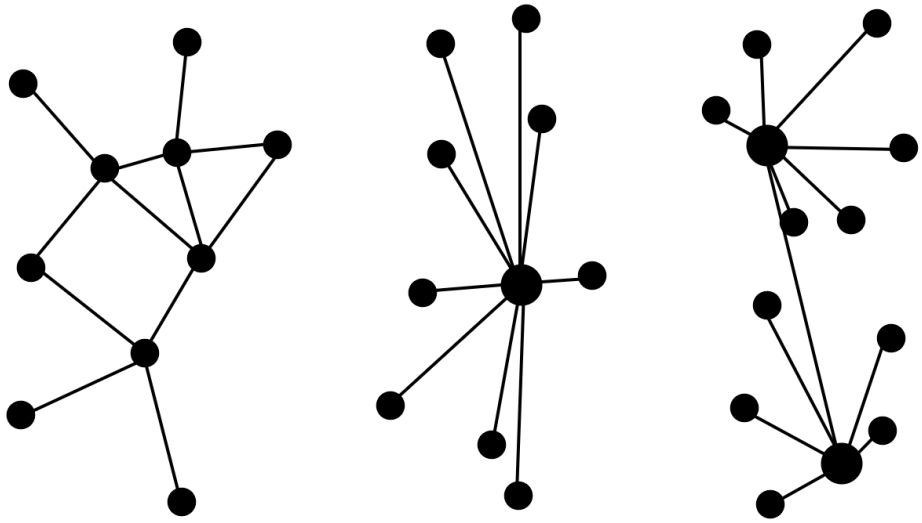

Figure 2.1: The three different multi-robot architectures. Decentralized (left) where each robot is equal and communicates with neighbours, centralized (centre) where all robots communicate to a central agent, and a hierarchical hybrid architecture (right) where the team is split into smaller centralized teams.

2.1.1 Centralized

Centralized architecture involves using a central agent, either a static computer or one of the robots in the team, to make decisions for all the robots in the team [4]. This central agent has access to all the robots' information and all available environment data. Since the central agent has access to all available data, it is capable of planning globally optimal solutions [4]. Centralized architecture has been applied to various problems such as task allocation [17] and path planning [19].

Centralized architectures generally suffer from ineffectual scaling to larger teams and vulnerability to issues in the central agent [4]. This is compounded in a USAR scenario due to

unreliable communication [22] and increased risk to central agents due to the dangerous environment [2]. If each robot in the team sends all sensor readings to the central agent, it would require prohibitively large bandwidth for a large team. Additionally, acting upon all the data from a large team would require very large computational resources. Lag in communications or disruptions would greatly affect the entire team, freezing it completely when it cannot receive orders from the central agent. In USAR environments, communication is very unreliable [22]. For these reasons, it is necessary to use methods less reliant on communication and more robust to failures in robot team members.

2.1.2 Decentralized

Decentralized control allows each robot to plan its course with adjustments based on observation and communication with the other robots [4]. Generally, it involves little direct communication between robots, usually only to neighbouring robots. The neighbourhood considered varies by algorithm and may also be defined by the range of possible communication, as long-range communication may be impossible in situations such as USAR [20, 22]. Decentralized methods are more adaptable to dynamic situations and robust to failure but lead to suboptimal solutions as each agent only has local information [4]. The low reliance on communication and robustness to failure make decentralized architectures optimal for USAR.

Although decentralized architectures do not preclude communication between robots as long as each robot plans its course, algorithms relying on heavy use of inter-robot communication nullify one of the core benefits of decentralized architectures. In this paper, algorithms that require each robot to be connected to every other robot and thus make decisions on global information will be considered hybrid architectures, discussed in section **2.1.3 Hybrid Architectures**.

Some algorithms, such as [5, 40], use no communication whatsoever. Instead, each robot only operates on knowledge that it acquires through interaction with the environment. This makes the system very robust as a failure in any robot as it will not affect the team. However, work may be duplicated. Systems that communicate to adjacent robots to jointly plan actions [6, 7, 8] are considered decentralized as they only require communication with adjacent robots. However, in large teams where there may be many adjacent robots, this also may lead to excessive communication and computationally expensive decision-making. Research such as [9,

41] has shown that mixing explicit and implicit coordination leads to equally good performance at a greatly reduced communication load. Finally, many systems are flexible in that they can work with any level of communication and remain effective by making use of implicit coordination [11, 25].

2.1.3 Hybrid

Decentralized solutions that require global coordination or select sub-team leaders to coordinate tasks will be considered hybrid architecture between centralized and decentralized.

Architectures that divide the robot team into smaller centralized teams each with a team leader are classic examples of hybrid architectures. These are called hierarchical architectures as there are two or more levels of authority in the team [4]. Hierarchical architectures have been applied to USAR as they are less communication expensive than centralized architectures [20].

Decentralized architectures that require complete communication between the whole team, such as [12, 13, 21], will also be considered hybrid architectures. Some algorithms are specifically designed to maintain this global communication network even when communication is unreliable [14, 15]. Other algorithms get around communication constraints by having all the robots periodically meet and jointly plan all goals [16]. Although the robots are still making decisions themselves, the main benefits of reduced communication loads and decentralized decision-making no longer apply. Additionally, significant errors in any one robot will affect the decisions of all other robots, not just adjacent ones. Due to these reasons, algorithms that require global communication or planning with the entire team will be considered hybrid architectures.

2.2 Explicit Coordination

When two or more robots need to coordinate in a decentralized architecture, there are two main approaches: explicit and implicit. Explicit coordination involves communicating directly with other robots and deciding on tasks together [41]. Implicit coordination involves inferring the plans of other robots either through limited communication or observation [41].

Although explicit coordination involves communication, in general communication is far more limited than what is necessary for centralized architectures since it is still a decentralized approach [4]. Usually, it only involves local communication. Some notable exceptions require global communication and thus should be considered hybrid architectures [12, 13]. However, for

USAR it is preferable to have limited communication, making implicit coordination advantageous as it requires significantly less communication [41].

2.2.1 Negotiation

Explicit coordination relies on every robot agreeing to the tasks itself and others in the vicinity will do. This is most clearly solved using negotiation, where the robots calculate the reward and cost of each task and distribute them such that the cost for the team is minimized. This method approaches global optimality but may require large calculation and communication loads as the cost for each robot to perform the tasks must be calculated [7]. A classic example of multi-robot exploration, the M+ algorithm [6], reduces the number of tasks by only considering optimality in the next time increment.

2.2.2 Market Strategies

To reduce communication and calculation loads of complete consensus and optimality analysis, various strategies based on market economics have been developed. These include auction or bidding strategies [8, 18], and trade approaches [7]. In these strategies, the robots often take on different roles, sometimes acting as auctioneers and sometimes as bidders. Since these methods create hierarchies, albeit temporarily, and thus rely on decisions made by other robots they can be considered hybrid architectures.

In Gerkey and Mataric's well-known auction method [18], an auctioneer robot would declare a task and the metric determining the fitness of a robot to achieve that task. Each robot that can feasibly accomplish the task calculates its fitness and provides it as a 'bid' to the auctioneer. Finally, the auctioneer decides on the winner and monitors their progress with the task. The approach improves upon previous research by communicating anonymously and assuming imperfect communication. However, it still requires high communication [7]. Trade approaches [7] attempt to lower communication requirements by having buyers ask for a task to be allocated to them rather than waiting for auctions, and sellers deciding winners based on messages received. Buyers choose desired tasks greedily based on the easiest task to perform [7].

2.3 Implicit Coordination

Implicit coordination involves inferring the goal of neighbouring robots through limited communication or observation [41]. Rather than explicitly dividing goals between each other, which may be time-consuming and require large communication and computational loads, each robot predicts what the others may do leading to emergent behaviour [41]. This is far more extendable to large groups of robots but is often used for less sophisticated tasks [41]. This is the most decentralized approach to multi-robot coordination.

2.3.1 Broadcast Position and Trajectory

Implicit coordination uses information on the actions and locations of other robots to predict their goals, and thus choose appropriate tasks. The simplest information that a robot can gather to determine the goals of other robots is their position and the map of what they've explored. From this, a robot can determine possible goals and avoid exploring areas that have already been mapped. Each robot will determine its goals by taking into account the goals of others in varying ways, such as by targeting only frontiers that it can target more efficiently than any other robot [27, 28], or decreasing rewards or potential around other robots [25, 31] to spread out the team.

Many algorithms using this approach have each robot share a great deal of data about their position, path, and area explored. Algorithms such as [11] directly share odometry and sensor readings, while [23, 24, 25] only share current position and local maps. These approaches are designed to work even when communication completely drops by inferring the position of robots that cannot be directly contacted.

2.3.2 Best Fit Task

Algorithms may decide their goal by deciding which frontier or goal they are more capable than other robots of achieving. The seminal work by Yamauchi [27] presents an approach where robots share perceptual information to build shared maps and thus attempt to distribute frontiers for exploration. However, this implicit coordination approach allows for frontiers to be explored again by other robots [28]. [28] improves upon frontier allocation by distributing robots optimally throughout the map rather than simply towards the closest frontier. [26] explores a similar idea for generic tasks rather than frontiers. This is useful as USAR is not

purely an exploration task, but also involves tasks of identifying and interacting with victims. However, calculating the best fit task to complete at each timestep is computationally expensive.

2.3.3 Approximation Methods

Rather than directly calculating the optimal task or frontier for each robot to pursue, many algorithms use an approximation to spread out the team. These methods benefit from less calculation and increased reactivity.

A simple approach is to repel robots away from other robots and attract robots to desired locations, such as frontiers. Algorithms such as [30] do this directly using potential fields overlaid on the environment, allowing the robot to 'flow' to low potentials near frontiers and far from other robots. [25] makes use of Markov decision processes solved using the value iteration algorithm [36], adding positive rewards to frontiers and negative rewards to areas around other robots. Finally, [29] makes use of a particle filter to coordinate a multi-robot search of a target by having each robot share random particles, allowing each robot to create local particle distributions of the target's location. These methods do not directly calculate the optimal task to perform but instead use an appropriate surrogate problem, such as maximizing reward or minimizing potential.

Knowledge of how the other robots will act is a necessity of implicit coordination. Many of the algorithms analyzed work only on homogenous teams that assume identical robots with the same algorithm governing their decision-making. However, multi-robot teams may be heterogeneous teams composed of very different robots, such as drones scanning the area and rovers locating victims. These heterogeneous teams may require more complex coordination algorithms [4], such as [9] which learns the behaviour of each member of the team. After learning a model of the behaviour of each member, it can plan based on what its teammates are likely to do.

2.3.4 Biomimicry

A less common approach to coordination is biomimicry, usually mimicking swarm insect communication. These methods usually do not use any direct electronic communication but communicate in different ways. [31] proposes using a communication system based on the 'waggle dance' of bees, who communicate with each other based on intricate movements in the

air. The paper describes various messages that can be communicated by UAVs by creating shapes in the air. This is useful for teams of UAVs that need to share information but cannot communicate directly. Other approaches use chemicals [32] to create trails that other robots can use for various purposes, similar to how ants navigate paths and leave messages using pheromones. Coordination through changing the environment or leaving messages in the environment is known as stigmergy [31, 32].

However, biomimicry is limited in its general application as these movements and implicit communication strategies can only communicate simple pre-programmed messages rather than maps, odometry information, or sensory information. Also, they typically require additional sensing capabilities, such as computer vision to determine waggle dance movements or chemical sensors to detect artificial pheromones.

2.4 Candidate Solution

2.4.1 Choosing a Candidate Solution

From all the approaches considered, one algorithm needs to be implemented to serve as a baseline for future USAR research. Ideally, the algorithm would be optimally suited to USAR by having the following characteristics:

1. Designed for unreliable communication, such as intermittent loss of communication, no long-range communication, or communication being blocked by obstacles and walls. However, short-range line-of-sight communication is assumed to function at least intermittently.
2. Robust to hardware failure in any member of the robot team. Due to the perilous nature of USAR, robots are likely to get physically damaged or destroyed during the operation [2]. The team must continue functioning even when one or more members of the team have been physically damaged.

These criteria naturally lead to highly decentralized solutions that use implicit coordination, reducing communication to a minimum and being highly robust to failure. Despite using implicit coordination, many of these algorithms still share significant amounts of information in the form of perceptual and odometry information [11, 23, 27, 28]. However, several solutions [24, 25] only share position and local maps.

Additionally, more practical criteria need to be examined. The chosen algorithm needs to apply to USAR, and thus should be mainly an exploration algorithm with some ability to perform tasks. This is compared to algorithms such as [26, 29] which are purely search and retrieval algorithms. Also, the algorithm should be simple and clearly explained, yet demonstrated to be highly effective.

Considering all these criteria, [25] was chosen to be implemented as the candidate solution. This paper, titled "Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes" is a fully decentralized algorithm that only shares local maps and positions. When a robot cannot communicate with another robot, it calculates a probability distribution of the other robot's position. Additionally, the algorithm was developed over several years and has several papers written on it by the same authors [10, 25, 33, 34, 35] culminating in a refined algorithm in 2015. The papers are clear and allow for re-implementing the algorithm. Finally, the algorithm won 2nd place in the CAROTTE Challenge, a challenge testing single and multi-robot autonomous exploration under communication constraints [10]. This algorithm's low communication, use of implicit coordination, clarity, and demonstrated effectiveness make it an optimal candidate. Additionally, it can be easily modified to prioritize different tasks such as identifying or interacting with victims, similarly to what was done in [35].

The algorithm described in [10, 25, 33, 34, 35], which will be called the "candidate solution" for the remainder of this document, is a Markov decision process planning algorithm applied to exploration. The first version of this algorithm was explored in [10], which considered only the single robot case and discussed possible extensions to multi-robot exploration. [33] laid out the algorithm for multi-robot search, and [34] demonstrated its capabilities. [25] further documents its communication infrastructure. Finally, [35] discusses optimally photographing the environment, which is part of the CAROTTE Challenge this algorithm was designed for. However, this is not a requirement of USAR, so this extension will not be used.

2.4.2 Single Robot Exploration

The first paper written for the candidate solution [10] implements single robot exploration, which forms the basis of the later papers on multi-robot exploration. To fully understand the complete multi-robot algorithm, it is necessary to first consider only single robot

exploration.

The algorithm works by creating an occupancy grid representation of the environment as it explores, where every cell is either free space, an obstacle, or unknown. This is referred to as the pixel layer or pixel map, as it can be represented as an image where every pixel is a cell. Many different sensors can be used for this purpose, but the paper uses a laser rangefinder. The algorithm was developed to be effective with cheap robots and sensors, such as a single laser rangefinder, rather than higher-end cameras or other expensive sensors [10]. This is doubly important for multi-robot USAR as robots are likely to be damaged or destroyed during the operation [2], and thus it may be financially infeasible to use teams of expensive robots.

A hexagon grid representation of the map is created based on the pixel map. This overlays a hexagon grid on the pixel map and groups pixels in each hexagon such that each hexagon is labelled free, occupied, or unknown, just like the pixel map. While the sensors update the pixel map, decisions are made on the hexagon grid. This is because the hexagon grid reduces the state space of the problem by grouping pixels. Additionally, rather than defining orientation as a continuous variable, the robot's orientation can be constrained to be one of the six directions of the hexagon, further reducing the state space of the problem.

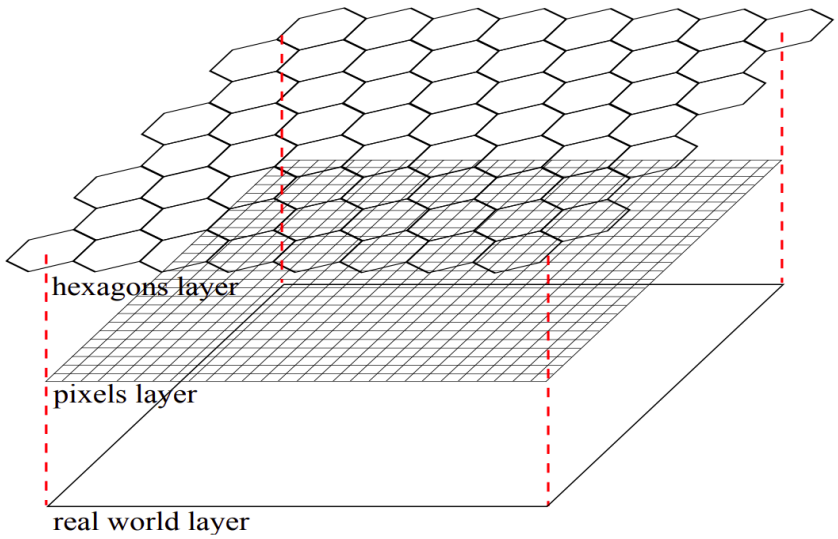

Figure 2.2: The real world is scanned and converted into a pixel map which simultaneously updates the hexagon decision layer. [10]

As the pixel and hexagon layers are updated, revealing free and occupied areas in the environment, rewards are propagated through the hexagon layer. A reward signifies how much useful information the algorithm believes can be gained by travelling towards that hexagon. Rewards are an integral part of the Markov decision process, as each robot will explore the environment by seeking to maximize its reward. Rewards are propagated around borders between free hexagons and unknown hexagons, as this is where the robot will uncover more of the environment. Only free hexagons can have rewards because it may not be safe to enter an unknown hexagon as it may be occupied, causing a collision.

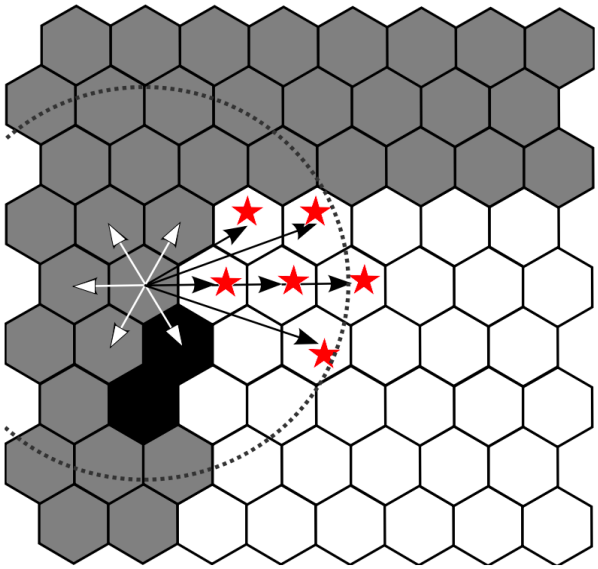

Figure 2.3: For every unknown (grey) hexagon bordering a free (white) hexagon, rewards propagate to free hexagons within a certain radius. Occupied (black) hexagons block this propagation. White arrows symbolize impossible propagations while black arrows symbolize successful propagations, and the red stars represent the rewards. [10]

To move between hexagons, the actions the robot can take must first be defined. The six orientations of a hexagon simplify this as it constrains the robot to only rotate between those six orientations. Other than rotating, the robot can move forward from one hexagon to an adjacent one, and any combination of those actions. Some robots may be able to do other actions, such as strafing, but this is not considered in this research as this is less common. However, it is assumed the robot can rotate in place. The set of actions can be changed to account for different robot capabilities.

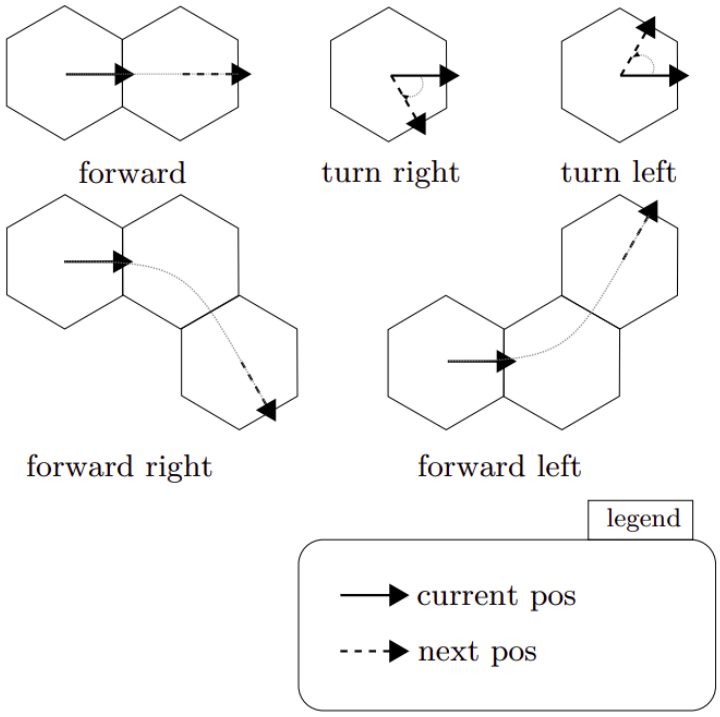

Figure 2.4: Set of possible robot actions. 'Forward right' and 'forward left' can be further decomposed into simpler actions. [10]

Each robot decides the next action by solving a Markov decision process (MDP). For a single robot, this MDP is defined by the tuple $\{S, A, T, R\}$ where:

- S is the set of all possible states, where each state is a position and orientation. A position is defined as a hexagon and the orientation is one of the six orientations of the hexagon. The position represents the centre of the robot, and the orientation is defined with respect to this centre hexagon.
- A is the set of all actions, as defined in Figure 2.4.
- T is the transition function, defining the likelihood of being in state s' after performing action a from state s. This can be simplified by assuming it is deterministic: impossible if the robot would hit an obstacle and certain otherwise. However, the robot would no longer consider the possibility that its actions may fail, such as by slipping.
- R is the reward function, determining the reward at each state as defined in Figure 2.3. To simplify the problem, the reward is dependent only on position, not on orientation. [10]

Once the MDP has been completely defined, it can be solved. The candidate solution solves it using the value iteration method [36], but MDPs can be solved in many ways. Rather than calculating a completely optimal solution across the entire map, which would be computationally expensive, the value iteration algorithm only considers the next H actions, where H is called the horizon [10]. The horizon can be changed during operation if desired. It can be increased to improve accuracy and reach distant rewards or decreased to reduce computation.

The core calculation for the value iteration algorithm can be seen below, where the true value of each state is calculated. Once these values have been calculated, the robot simply chooses the action that moves it to the adjacent state with the highest value [10].

$$V(s) = R_{exploration}(s) + \gamma \sum_{s'}^{S_H} T(s, a, s')V(s') \quad (1)$$

Where:

- $V(s)$ is the true value of state s
- $R_{exploration}(s)$ is the reward at state s
- γ is the discount factor, which weighs the value of future events
- S_H are all the states reachable in the next H actions
- $T(s, a, s')$ is the probability of transitioning from state s to state s' after doing action a

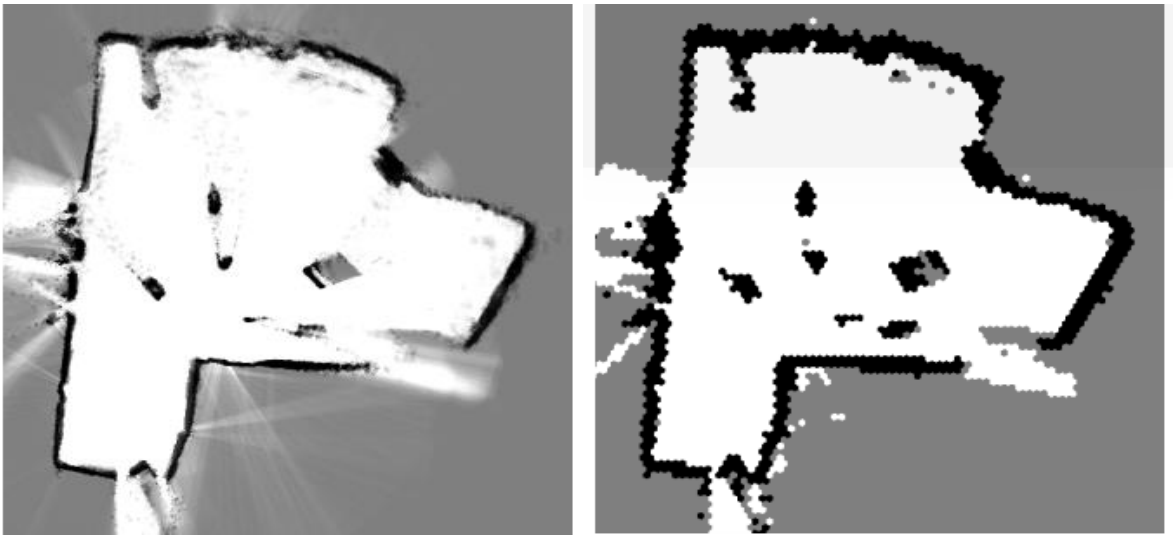

Figure 2.5: Final pixel layer (left) and hexagon layer (right) after complete exploration. White represents free space, black represents occupied space, and grey represents unknown space. White space outside of the room has been observed through partially open doors. [10]

2.4.3 Multi-Robot Coordination

The candidate solution was designed for the CAROTTE Challenge, which only allows for limited bandwidth and temporary communication loss [25]. The solution was designed with these limitations. The robots communicate their local maps which are merged to create a shared map, preventing duplicated work. Additionally, robots share their positions but not their goals or trajectories. Each robot must infer the actions of the others using the shared map and the current and past positions of the other robots [25].

Assuming perfect communication, the MDP each robot solves remains largely the same except for the reward function. To encourage robots to not target areas that other robots can explore more effectively, a repulsive MDP is defined. This repulsive MDP sets a positive reward in the state where other robots currently are, and no reward anywhere else. Then, by solving this MDP, the robot generates the repulsive value of each state within the horizon. Finally, the reward at each state of the original MDP is modified by subtracting the value calculated in the repulsive MDP. In this way, the robot is attracted to areas of high information gain as before and repulsed by other robots. The balance between these two reward mechanisms can be tuned by changing the amount of reward given in the repulsive MDP [25].

$$V_{i,repulsive}(s_i) = R_{i,repulsive}(s_i) + \gamma \sum_{s'}^{S_H} T(s_i, a, s') V_{i,repulsive}(s') \quad (2)$$

Where:

- $V_{i,repulsive}(s_i)$ is the repulsive value at state s_i for robot i
- $R_{i,repulsive}(s_i)$ is the repulsive reward for robot i, which is positive at the state where any other robot currently is and zero everywhere else

Using $V_{repulsive}(s_i)$ and $R_{exploration}(s_i)$, the final reward function is calculated as:

$$R_i(s_i) = R_{exploration}(s_i) - V_{i,repulsive}(s_i) \quad (3)$$

The final MDP assuming perfect communication is:

$$V_i(s_i) = R_i(s_i) + \gamma \sum_{s'}^{S_H} T(s_i, a, s') V_i(s') \quad (4)$$

However, this only works when there are no errors communicating with distant robots. If each robot has communicated with the other robots at least once, it can use their previous

position to estimate their current position. If a robot has never communicated with another robot, it cannot coordinate with it as it does not know it exists. Given the last known position of another robot, the robot must make use of distributed value functions (DVFs) to estimate the value each robot places on each state and the probability it is currently at each state. DVFs were introduced to distributed robotics by [38]. Each robot can estimate another robot's value function 'emphatically' because the algorithm assumes a homogeneous team, and thus each robot uses the same algorithm [25]. This could potentially be extended to heterogeneous teams but will not be necessary for this investigation.

It is computationally expensive to calculate the value functions of all the other robots and the probability they have moved to every state. This can be simplified by only considering robots in one's neighbourhood. However, this will sacrifice accuracy. The approach of calculating every other robots' value function is termed the "Pessimistic" DVF while only considering neighbouring robots is the "Optimistic" DVF. The general form of the value function is:

$$V_i(s_i) = R_i(s_i) + \gamma \sum_{s'}^{S_H} [T(s_i, a, s') V_i(s') - \sum_{j \neq i}^{N} f_{ij} Pr(s' \mid s_j, \Delta t_j) V_j(s')] \quad (5)$$

Where:

- N is the number of robots
- f_{ij} is a weighing factor on how important the value function of robot j is to robot i. This can be set to 0 for robots that are not neighbours for the Optimistic approach.
- s_j is the last known state of robot j
- Δt_j is the amount of time since robot j has communicated its location
- $Pr(s' \mid s_j, \Delta t_j)$ is the probability robot j explores or occupies state s' given that Δt_j time ago it was observed at state s_j
- $V_j(s)$ is the value function of robot j

This DVF approach works even for perfect communication by setting Δt_j to zero since every robot communicates with every other robot at each iteration. In fact, this is the final approach used by [25] for all levels of communication. $Pr(s' | s_j, \Delta t_j = 0) = Pr(s' \mid s_j)$ thus represents the probability robot j chooses to soon explore state s' given it is currently located at state s_j.

Calculating $Pr(s' \mid s_j, \Delta t_j)$ is complex but can be achieved by determining the probability the robot takes a trajectory from the previous state to the new state. This calculation is done in the candidate solution by using Voronoi diagrams, which significantly reduce the number of possible trajectories. Voronoi diagrams are also used in the final algorithm to plan safe paths through the environment (see Figure 2.6). This is described below [25]:

$$Pr(s' \mid s_j, \Delta t_j) \; = \; \eta \; \sum_{\tau}^{Traj(s_j, s')} Pr(\tau, \Delta t_j) \tag{6}$$

Where:

- η is a normalizing constant
- $Traj(s_j, s')$ is the set of trajectories from s_j to s'
- $Pr(\tau, \Delta t_j)$ is the probability robot j follows trajectory τ in time Δt_j

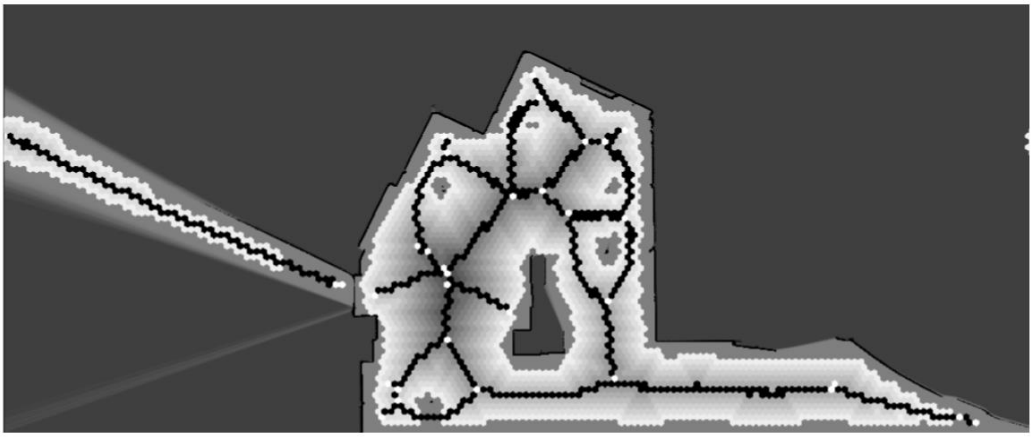

Figure 2.6: A Voronoi diagram in the hexagon layer generated in the candidate solution. Black curves are Voronoi edges and white points are Voronoi nodes. [33]

These papers only considered intermittent communication loss, but USAR operations usually suffer from generally unreliable communication with little to no long-range communication [22]. This algorithm works regardless of whether communication loss is complete, intermittent, or limited in range and thus can be applied to USAR.

Additionally, this algorithm does not cover local coordination, such as when two robots are trying to fit through a small doorway. Although the robots are repelled from one another, occasions may occur when robots are forced into close contact, and this will require a different coordination technique. [25] employs a different algorithm ([39]) for this situation but this will not be implemented in this thesis.

2.5 Conclusion

The field of multi-robot coordination spans a wide variety of approaches that make varying assumptions about the availability of communication and computation. Centralized architectures can optimally perform a task or explore as all available information is used by a central computer to plan the actions of all agents. However, it is computationally expensive and communication heavy, and very susceptible to failure [4]. Decentralized architectures allow each agent to perform its decision-making, but in doing so reach sub-optimal solutions. However, decentralized architectures are generally more robust, allow for larger teams, and have lower requirements for computation and communication [4]. USAR suffers from unreliable communication [22] and high danger to robots [2], making decentralized architectures ideal.

Decentralized robots must share information and coordinate to perform tasks and reduce duplicated work. This coordination can be explicit, where the robots agree on task allocation, or implicit, where the robots use limited information to predict the goals of the other robots. Implicit communication generally uses far less communication as only essential information needs to be shared [41]. Additionally, it is more robust to complete communication loss as a robot can infer the paths of other robots and allocate new tasks to itself without communicating directly with other robots [41]. For these reasons, implicit coordination is preferred for USAR.

A candidate solution was chosen from decentralized architectures that make use of implicit coordination. Various practical considerations, such as applicability to USAR, demonstrated effectiveness, and reproducibility, were also considered when choosing a candidate solution to implement. The algorithm “Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes” [25] was chosen as it satisfies these criteria and is flexible to modifications for USAR. The algorithm uses a planning approach based on solving a Markov decision process to move robots toward frontiers while avoiding areas other robots may explore. If communication is lost with a neighbouring robot, it will infer its current position from its last known position. This algorithm will be implemented in simulated USAR scenarios and used as a baseline for future research into multi-robot USAR.

3 Design and Implementation

The algorithm is implemented in a grid world in Python. This is because the candidate solution is a coordination algorithm concerned with high-level tasks such as exploration targets. It is not concerned with low-level control such as path planning, odometry, or velocity control. As a result, a simple grid representing obstacles and free space suffices. Python was chosen due to its portability to the Robot Operating System (ROS) and the simplicity for prototyping. In the future, this implementation may be ported to a complete 3D simulation in ROS by replacing the inputs to the algorithm with ROS topics. This is further explored in section **4.5.5 Moving to ROS.** However, a complete simulation in ROS is not the purpose of this thesis.

3.1 Pixel Map and Hexagon Grid Layers

The candidate solution splits the environment into three layers: the real world, the pixel layer, and the hexagon grid layer (see Figure 2.2). The real world in this project is simply an image of the map. It is then converted from a grayscale image into a pixel layer where every pixel is either free, occupied, or unknown. The pixel map is divided into larger hexagons encompassing many pixels to create the hexagon layer where decisions are made. This dramatically reduces the state space of the algorithm and allows for orientation to be defined as one of the six orientations of a hexagon.

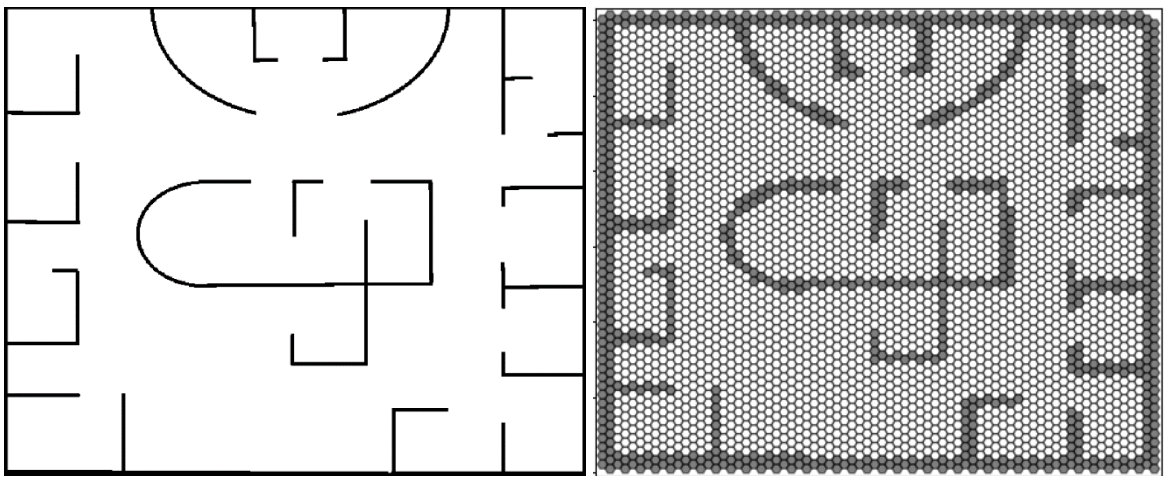

Figure 3.1: A pixel map (left) of an office environment taken from [25], and the hexagon grid (right) generated from the pixel map.

The hexagon grid was defined as a dictionary of custom Hex objects containing the state (i.e. free, unknown, or occupied) and other valuable information such as reward. The dictionary is indexed by the axial coordinates of the hexagon. Since dictionaries in Python are implemented as a hash table, their indexing is theoretically O(1) [44], dramatically reducing computation time for the algorithm. Any update to the pixel layer done by scanning the environment immediately updates the hexagon layer, changing the states and rewards of hexagons.

3.2 Simulating Rangefinders

Once the robot has access to a pixel layer and a hexagon layer, it needs to scan the environment to update the pixel layer. The actual implementation of the rangefinder is arbitrary as many different rangefinders could be used. For this simulation, the robot perfectly measures the distance to the closest object directly in front of it, revealing its field of view when rotating from one orientation to the next. This is calculated using Bresenham's Algorithm [37]. Each scan reveals free pixels and obstacle pixels, which are updated in the pixel layer. The pixel layer is then used to update the hexagon map.

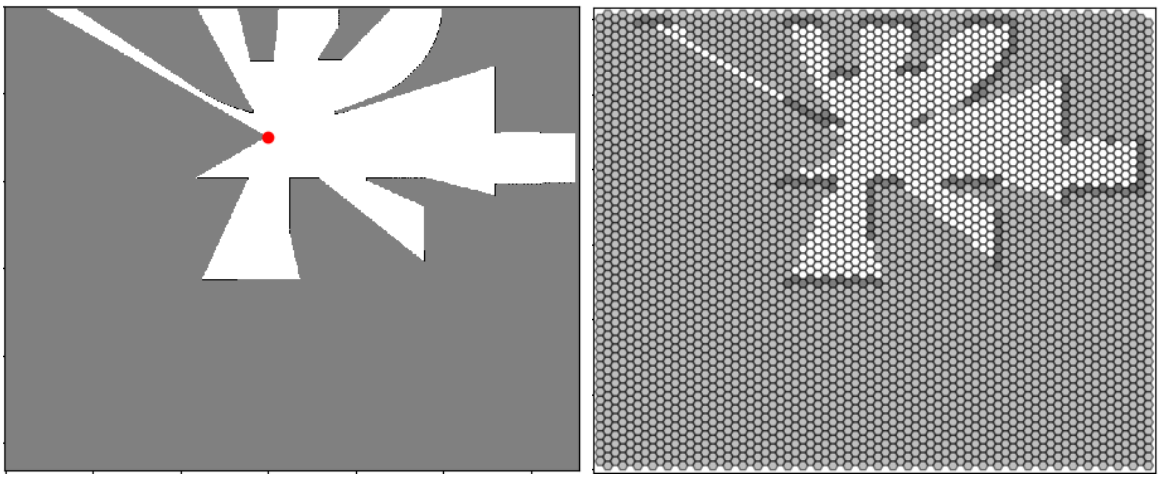

Figure 3.2: The robot observes the environment as it rotates, uncovering free and occupied pixels. The pixel map (left) was generated when the robot did a 300-degree rotation. The hexagon grid (right) was then updated.

If this algorithm is moved to ROS, a pre-built mapping module can be used to replace this section. This is outlined in sections **4.5.3 Realistic Mapping** and **4.5.5 Moving to ROS.**

3.3 Propagating Rewards

As the robot scans its environment, rewards need to be propagated near frontiers between free spaces and unknown spaces, as seen in Figure 2.3. This reward propagation is done whenever the hexagon layer is updated. This is done by clearing the rewards from the entire map and then checking every unknown hexagon. If the hexagon has a neighbour within the desired radius that is free, and there is a clear (unoccupied) path between the unknown hexagon and the free hexagon, then a static reward is set on the free hexagon. The reward has been chosen to be the same regardless of how far the free hexagon is from the unknown hexagon (as long as it is within the desired radius) to follow the algorithm. Since the frontiers retreat as the robot approaches because the robot observes the unknown hexagons, it is not necessary for the robot to directly reach the frontier hexagons. Rather, the robot should be encouraged to look in that direction. Due to this, the robots rarely ever reach any rewards, but this does not affect the algorithm.

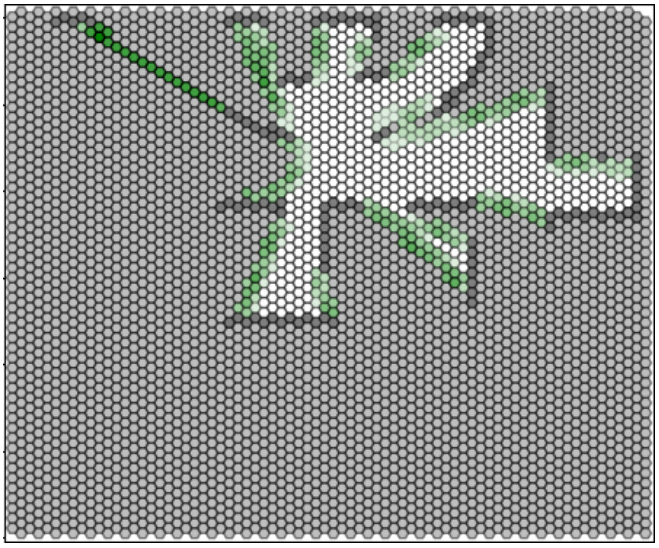

Figure 3.3: Rewards (green) are propagated in the free hexagons near unknown hexagons, to a radius of two hexagons. The intensity of the green indicates how much reward there is at that location. Higher rewards signal more information can be gained by approaching that location.

3.4 Implementing the Markov Decision Process

The initial paper for the candidate solution outlines the MDP for single robot exploration [10], described in section **2.4.2 Singe Robot Exploration**. To solve the MDP, first the states, actions, transition function, and reward function are defined. The hexagon grid defined in section

3.1 Pixel Map and Hexagon Grid Layers defines all the states, as a state is simply the hexagon the robot is occupying and the orientation it is facing on that hexagon. The actions are defined as moving forward, turning left, or turning right. The transition function is defined with regards to a variable called $noise$. The desired action is performed with a certainty of $1 - noise$, and a random action is performed with a probability of $noise$. This makes the robot more cautious when near undesirable areas such as walls or other robots. However, it was found that introducing this noise into the MDP significantly slowed the convergence of the algorithm. Thus, for testing the noise was set to zero. For real operations, where avoiding collisions is vital, the noise should be increased to reflect the actual probability of failure. Since in this simulation the robot's actions never failed, increasing $noise$ did not improve the accuracy of the algorithm. The transition function may be updated for larger robots in section **4.5.1 Simulating Large Robots**, as the robot will take up more than one hexagon. Finally, the reward function has been accomplished in section **3.3 Propagating Rewards**. Thus, the MDP has been defined and can be solved using the value iteration algorithm [36].

The value iteration algorithm is guaranteed to converge to the optimal solution given enough time [36]. The convergence speed and accuracy rely on two tuneable parameters, the discount factor γ, and the horizon H. When a policy is being computed with value iteration, only up to H future actions are considered to reduce computation. The discount factor weights the importance of future events as compared to maximizing the reward in the present. In experiments in [10], the discount factor was found to have little effect on the algorithm. However, the horizon has a large effect on the behaviour and runtime of the algorithm. A smaller horizon reduces computation but also may reduce the optimality of the algorithm as it only considers its immediate surroundings. Additionally, if there are no rewards within H actions of the robot, it will become stuck as it will ignore distant rewards. In [10], it was found that there are usually many rewards surrounding the robot, and H can be kept small. It is dynamically increased if the robot ever finds itself stuck too far away from any rewards.

By testing it was found that having a small horizon (set to a distance of 20 hexagons) worked well. However, if no rewards are within that horizon then it must expands not only to consider the nearest reward but also the entire path towards the nearest reward. Expanding the horizon to accommodate the path was necessary since the value iteration algorithm iteratively calculates the value of intermediate states, so if the path is not included in the horizon the value

from a reward hexagon will not be propagated back to the robot's current state. Also, it was found that a static discount factor was unable to perform well both when rewards were close and when rewards were very distant. If rewards were very distant and the discount factor was low the value iteration algorithm would not propagate rewards back to the robot since the value of the states would converge to zero before reaching the robot. However, if the discount factor was too high it would disregard small rewards around it, greatly affecting total mission time as the robot needed to make a second pass around the entire environment to pick up small rewards that had been ignored. In practice, it was found that increasing the discount factor proportionally to the length of the path to the nearest reward allowed the robot to operate well during the entire exploration while still being able to tune the robot's greediness as desired. This is further explained in section **4.4.1 Tuning Parameters.**

The results of the single robot exploration algorithm can be seen below. It efficiently explores the map by disregarding small rewards (such as corners that have not been fully observed) in favour of searching the majority of the space quickly. The value function can be seen below in green, where a darker green indicates a higher value. The value function only extends to a horizon of 20 hexagons, unless increased to reach distant rewards.

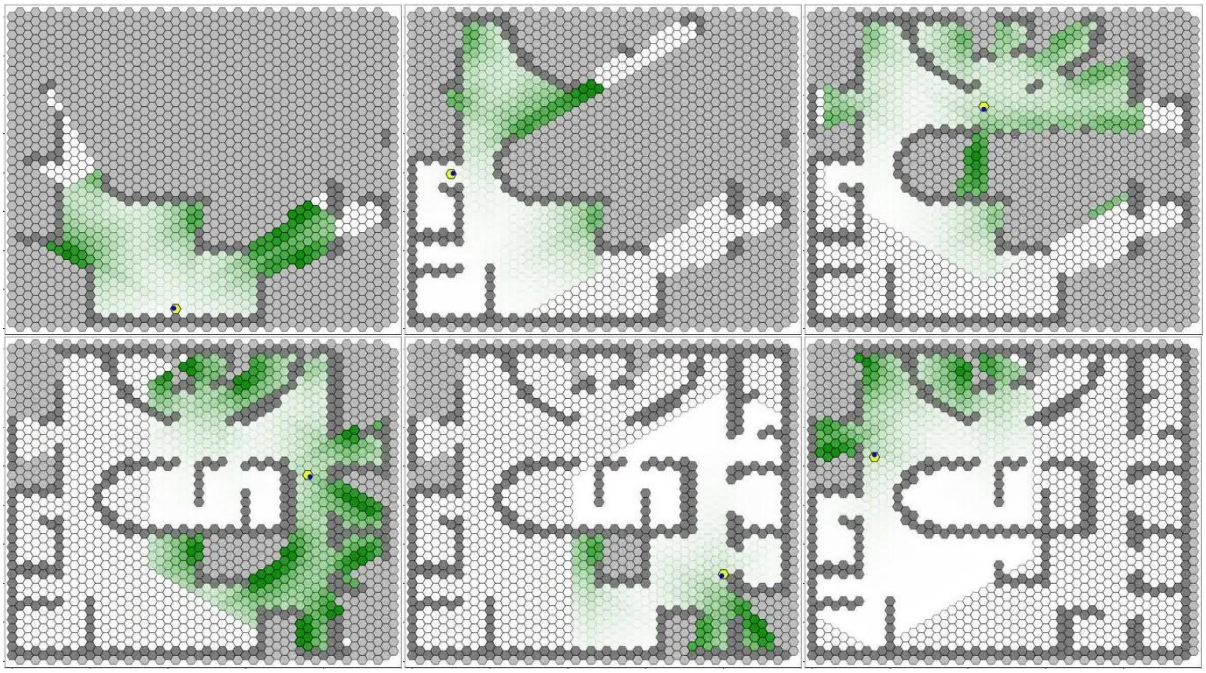

Figure 3.4: Single robot exploration using the MDP. The robot is represented by the yellow hexagon and the blue dot indicates the front of the robot. Value is in green, with darker green indicating a higher value.

3.5 Multi-Robot Coordination

To coordinate with the team, each robot communicates its position and map, merges in received maps, and uses the received positions of other robots to estimate their future destinations. The DVF formulation from section **2.4.3 Multi-Robot Exploration** (equation 5) was used for both perfect and imperfect communication. Thus, the value function of every known robot in the horizon of the current robot is calculated. This along with the probability that the other robot will explore a state given its last known position and the time elapsed since the last communication is used to estimate likely areas the other robot will explore. The repulsive MDP formulation is used to calculate the value function of other robots. This value function is calculated while the robots are communicating but is not updated when the robots stop communicating as it is unknown how the value function of the other robot will evolve. A visualization of the prediction calculation is included in Figure 3.6. The entire process for multi-robot coordination for a given robot is given in Figure 3.5.

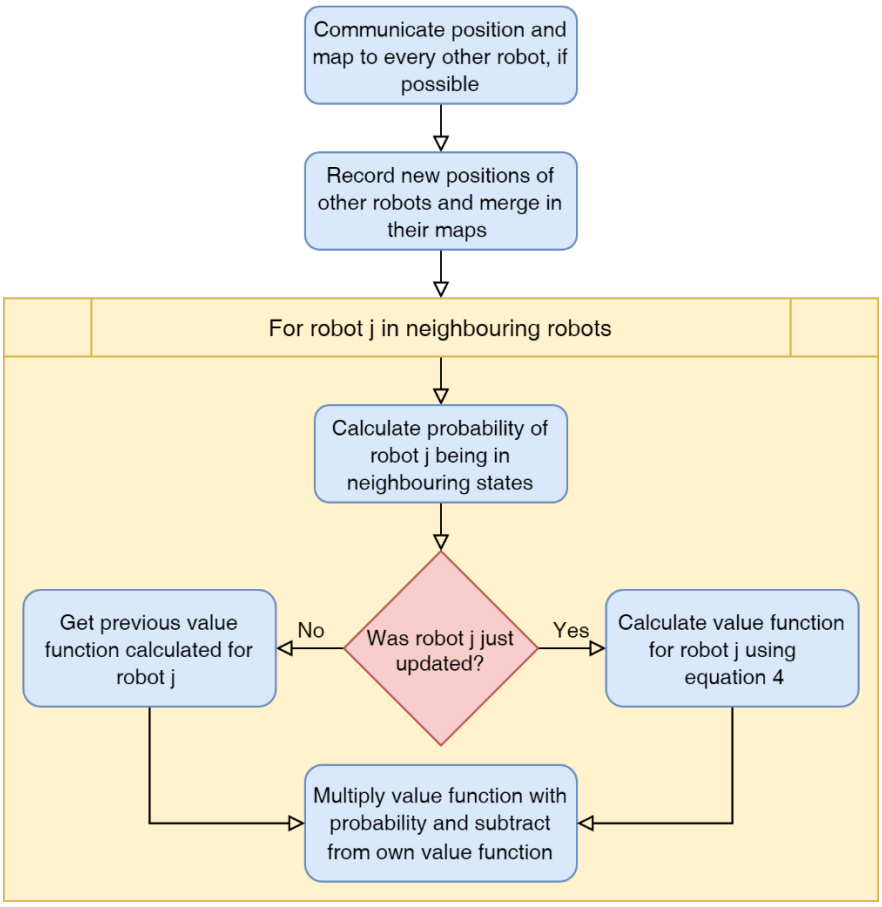

Figure 3.5: Process for a given robot to coordinate with its teammates.

To estimate the probability the robot is targeting or currently occupying neighbouring states, the paper utilizes Voronoi diagrams. However, this calculation of probability was not described in detail by the candidate solution [25] and assumes robots move along Voronoi edges, which is not the case for our solution. Instead, a novel probability calculation was developed. The probability that a robot is occupying or targeting a state is zero if that state is farther than a certain distance from the last known location. This distance, called the *exploration horizon*, grows proportionally to the time since the robots last communicated. Thus, if the robots have not communicated in a long time, the area that the other robot may explore is far larger. The probability the other robot is occupying or targeting a state within the exploration horizon is inversely proportional to the distance between the last known location and the state. Also, the probability is inversely proportional to the number of free states within the exploration horizon. This is because if there are very few states within the exploration horizon (such as if the other robot was in a narrow corridor), the likelihood that the other robot is in any of these few states is far higher than if the robot was last seen in a large open space.

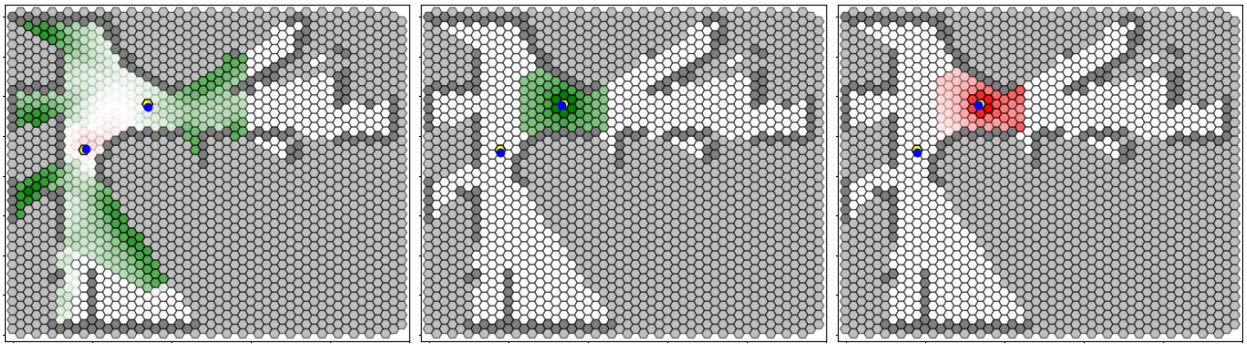

Figure 3.6: Visualizing the prediction step. (left) the estimated value function for the rightmost robot. (centre) the probability of the robot exploring neighboring states. (right) the value function multiplied with the probability function.

Now that each robot has estimated the value function of every other robot, multiplied it by the probability they are going to explore their neighbours, and subtracted this from their value functions, they are ready to explore in a coordinated fashion. The coordinated exploration can be seen below where the robots clearly avoid one another. Positive valued hexagons are green where darker green indicates higher value. Negative valued hexagons are red where darker red indicates a more negative value. Each robot places strong negative values around the other robot as long as it is within the horizon considered by the value iteration algorithm.

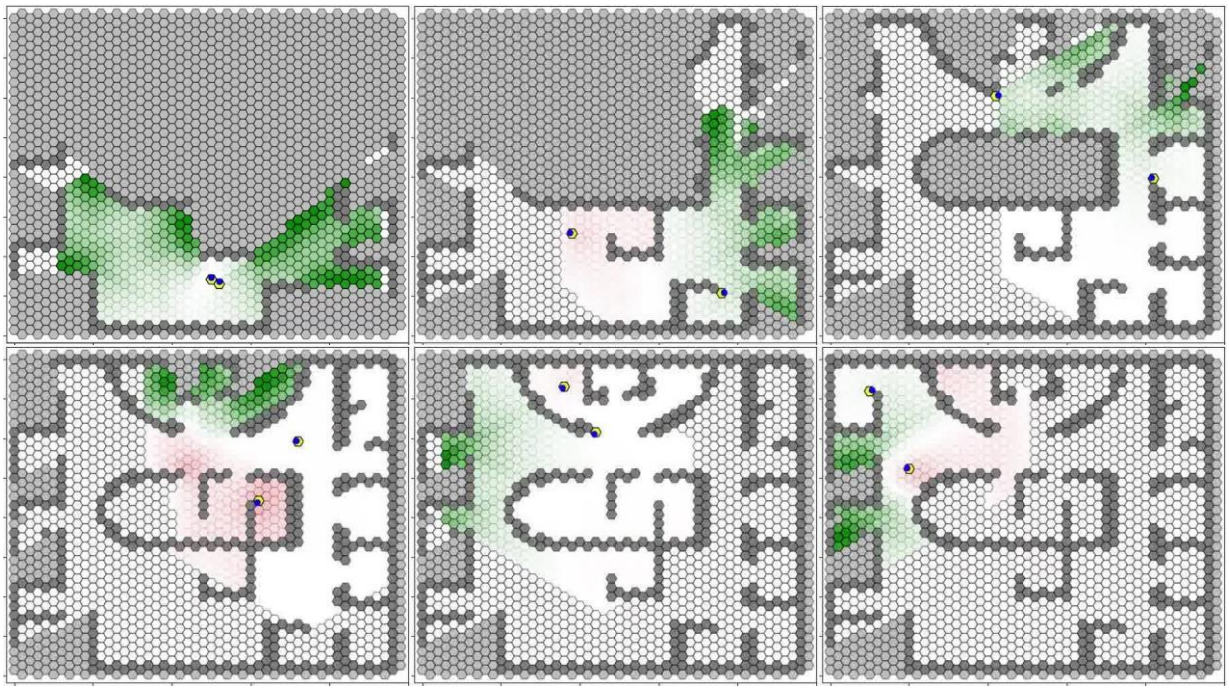

Figure 3.7: Multi-robot exploration using the MDP, from the perspective of one of the robots. The robots are represented by the yellow hexagons and the blue dots indicate the front of the robots. Positive value is in green, and negative value in red. Darker green indicates more value, and darker red indicates a more negative value.

3.6 Designing USAR Scenarios

Now that the robots coordinate effectively, a scenario that resembles USAR needs to be created. Depending on the robot's functionality, such as being able to identify victims using computer vision, the algorithm itself could be updated. This is explored in section **4.5.6 Adapting the Algorithm for USAR Scenarios.** For this thesis, only the exploration task is considered.

To create an exploration scenario that reflects the challenges of USAR, two of the aspects of USAR need to be simulated:

1. A cluttered map with many potential exploration trajectories that collide, making coordination vital to efficiently explore and avoid collisions.
2. Communication constraints that mimic real-world USAR communication failure.

To create a cluttered map, maps used in USAR research were analyzed. However, these

maps were rarely as cluttered and real USAR scenarios. Many papers, including [25], utilized simple office layouts with all furniture and objects removed. A cluttered map was designed by editing one of the office layouts from [25], removing walls and adding rubble. By removing walls, the number of trajectories that can be used to move between any two points was increased, as the original map was very divided between two corridors with little chance of two robots colliding. This, along with the added clutter, created a more realistic scenario for USAR. Additionally, added obstacles decreased communication when communication was blocked by obstacles, increasing the difficulty of the scenario.

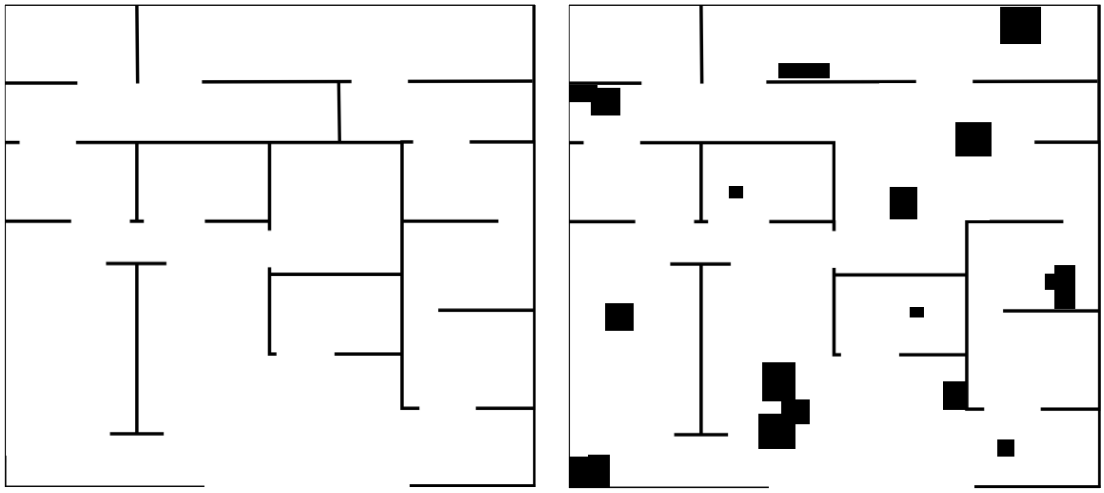

Figure 3.8: (left) a map from [25], (right) modified map to be more cluttered and have more possible collisions.

Communication failures happen in a multitude of ways in real USAR scenarios. In [25] they test communication failure by having all communication go down for a set amount of time at set intervals. Two other methods of communication failure are communication being blocked by obstacles and communication only extending to a certain radius. All three of these have been implemented.

3.7 Greedy Approach

To properly measure the efficacy of the candidate solution, a naïve baseline algorithm was implemented. A simple greedy approach was programmed where each robot moves directly towards the closest reward. First, the nearest reward is found using a breadth-first search. Then, a

single action is done in pursuit of this desired new state, either moving forward or rotating. After each action, the environment is scanned again, and the desired path is recalculated. This is extended to multi-robot exploration by allowing the robots to transmit their maps and merge in received maps, similarly to the candidate solution. However, other than map merging there is no change to the base algorithm in the case of multi-robot exploration; the greedy approach will simply move the robots towards the nearest reward as before. This can be seen below.

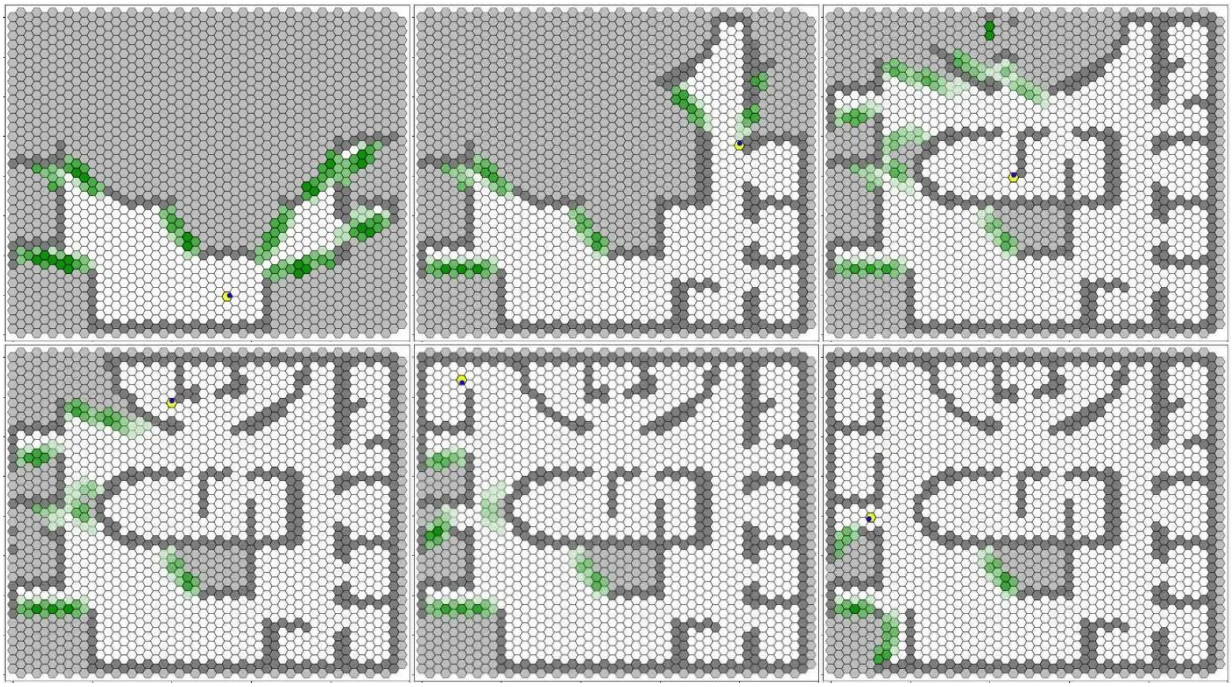

Figure 3.9: Single robot exploration using the greedy approach where the robot moves towards the closest reward. The robot is represented by the yellow hexagon and the blue dot indicates the front of the robot. Rewards are in green, with darker green indicating higher reward.

4 Results and Discussion

4.1 Benchmarking Success

Now that the candidate solution has been completed, the next step is to determine how to benchmark success as compared to other algorithms. Metrics for human and robot USAR operations were researched. Since this thesis focused on multi-robot exploration and not other aspects of USAR, such as assisting victims, metrics related to coordinated exploration in limited communication environments were chosen. [25] measures the effectiveness of its algorithm based on two metrics:

1. **Total mission time:** The time required for the team to completely explore the area.
2. **Cumulated Local Interactions:** The cumulated time any two or more robots were within one meter of each other.

Ideally, total mission time is minimized. However, for USAR, due to the time pressure to find victims sooner rather than later, it is equally or more important to explore a majority of the space early on rather than completing the entire exploration in less time. For example, an algorithm that explores 90% of an area far faster than another algorithm but takes longer to reach 100% may be preferable since victims may become unresponsive if they are not located and rescued in a set amount of time (e.g. the first day). For these reasons, not only is total mission time measured but the rate of exploration is measured to determine how rapidly each algorithm explores. This is measured as a percent of the environment explored per iteration of the algorithm.

Local interactions are undesirable for two reasons. Firstly, it indicates the robots are not spreading out and exploring the area as effectively as possible. Ideally, the robots would uniformly spread out into the environment and not need to interact closely unless it is necessary, such as passing through a single doorway. Secondly, when communication is guaranteed, two robots interacting locally can make use of local coordination to prevent collisions. However, when communication fails the robots have no way of coordinating, which may cause collisions or cause robots to get stuck. These collisions may damage the robots. For these reasons, local interactions are inefficient and dangerous. Local interactions are measured as the cumulated number of iterations where two or more robots are near each other.

4.2 Testing Methodology

With metrics clearly defined, the algorithm is ready to be tested. Firstly, a testing methodology must be defined that will properly test the algorithm's efficacy and limitations. For these tests, several aspects need to be defined:

1. The number of robots in a team
2. The map or maps which the robots will explore
3. The starting locations on the map
4. The levels of communication to simulate
5. The algorithms to compare

Ideally, a large variety of tests with different parameters would be performed. However, due to time restrictions, it was decided to test only with two robots on the cluttered map in Figure 3.8. More robots increase the computation time significantly because more maps need to be merged and value functions estimated per robot on every iteration. Also, each robot computes its value function sequentially rather than each robot working in parallel, so additional robots add time to each iteration even with no communication. Moreover, to justify larger teams a large map would need to be used. Large maps dramatically increase the state space, both increasing the amount of time required to compute value functions and the number of iterations required to fully explore the environment. Due to this, it was infeasible to test with many robots. To properly test larger teams, the code would need to be parallelized as is explored in section **4.5.4 Parallelization**, and a much larger map would need to be employed.

At first, testing was done with the robots starting at positions uniformly spread throughout the environment. However, this was deemed unrealistic as in a USAR operation the robots would likely start near each other, not dispersed within a collapsed building. Also, this reduced the need for cooperation as robots were already spread out. Instead, the robots all started near each other in the 'lobby' area of the chosen map. This starting area can be seen in Figure 4.4 as a green semi-circle. To add a degree of randomness, the robots start in random positions in the starting area.

Communication can fail in a variety of ways in real USAR scenarios, and several of these have been programmed. The code can accommodate full communication, no communication, communication blocked by obstacles, communication that extends to a certain radius, and

intermittent communication, as well as combinations of these. Full communication and no communication were tested. However, for testing purposes, the only communication failure that was tested was communication being blocked by obstacles. This was chosen as it realistically replicates short-range communication methods being blocked by concrete and rebar, a common difficulty in USAR [22]. Additionally, due to how cluttered the testing environment in Figure 3.8 is, the robots communicate infrequently if communication is blocked by obstacles. Thus, this method of limiting communication presents a more challenging communication failure than the other communication failure methods and produces representative results of the efficacy of the algorithm in communication limited environments.

Finally, this algorithm is meant to serve as a benchmark against which to compare future multi-robot exploration algorithms. Ideally, it would be tested against a variety of other algorithms to determine if it is an appropriate choice as a benchmark. However, due to time limitations, only the greedy approach was implemented in section **3.7 Greedy Approach**. Although the algorithm is simple, it is efficient in that no actions are wasted since it always moves robots directly towards rewards and very systematically clears areas. The candidate solution is less direct and may waste actions avoiding areas with other robots or due to inaccurate value functions. To surpass the naïve greedy approach, the candidate solution needs to effectively coordinate with the team. Thus, the greedy approach is an appropriate baseline with which to compare the candidate solution.

Additionally, [25] confirms that predicting the locations of other robots when not directly communicating with them is effective by testing against a variation of their algorithm. In communication limited environments, [25] tests a version of their algorithm where if a robot loses communication with a teammate, it assumes it does not exist. In a sense, the robots always assume perfect communication and do not account for limited communication by skipping the prediction step. This non-predictive variation was also tested in this thesis to determine the efficacy of the prediction step when communication is limited.

In summary, the tests will use the following parameters:

1. The team will consist of two robots.
2. Testing will be done on the cluttered map found in Figure 3.8.
3. The robots start in random locations of the starting area shown in Figure 4.4.
4. Tests will happen with full communication, communication blocked by obstacles,

and no communication.

5. The full candidate solution and the greedy approach from section **3.7 Greedy Approach** will be compared. Additionally, the non-predictive variation of the candidate solution will also be tested for the limited communication tests.

To ensure reproducibility and accurate results, each test will be run 10 times and the results will be averaged.

4.3 Results

Using the testing methodology above, a total of 70 tests were done: 10 tests per algorithm (greedy and candidate solution) per communication method (full communication, limited communication, no communication) plus 10 tests for the non-predictive candidate solution variation using limited communication. For the results below, Markov Decision Process (MDP) refers to the candidate solution, and MDP – Ind is the independent (non-predictive) variation of the candidate solution. These results are averaged over 10 trials.

4.3.1 Rate of Exploration

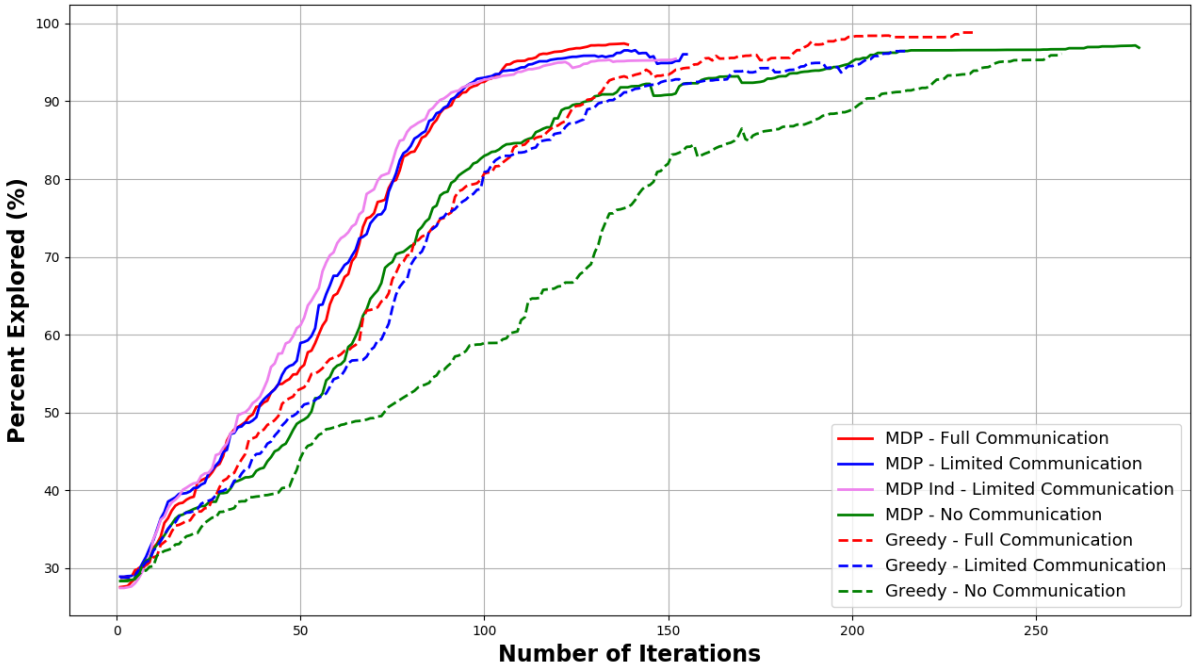

Figure 4.1: The percent explored per iteration for all 7 tests.

For the candidate solution, it is not necessary to communicate on every iteration since the two robots communicate when they are close and coordinate to spread out. Since the robots spread out very effectively even if communication is blocked by obstacles (as can be seen in Figure 4.4), performance does not degrade much when communication is limited. However, for the greedy approach, limiting communication counterintuitively bolsters performance. This may be because the main failure of the greedy algorithm is that the robots choose to target the same reward, and thus move towards the same location and follow the same path for large sections of the exploration. This can be seen in Figure 4.5. Breaks in communication allow the robots to de-sync and pursue different rewards, improving total mission time.

More surprising is that when communication is limited, the non-predictive variation of the candidate solution performs slightly better than the complete predictive approach. Since the difference is relatively minor this may be due to random variation. However, the major caveat in this result is that in one run, that was excluded, the non-predictive version got permanently stuck. One robot would turn a corner, see the other robot was there, and retreat. However, after retreating behind the corner, it would forget the other robot was on the other side. It would reattempt to turn the corner. The other robot was similarly stuck. This is a major problem of the non-predictive approach. This was the only test that was excluded from the results.

Importantly, the candidate solution outperforms the greedy approach regardless of communication. Even without any communication, the candidate solution explores far faster than the greedy approach, exploring on par with the greedy approach when communication is perfect. This is because the value iteration algorithm maximizes the reward according to the discount factor and has been tuned to explore more of the environment sooner at the expense of total mission time. However, this caused the candidate solution with no communication to have the longest total mission time. If desired, the candidate solution can be tuned to be greedier by decreasing the discount factor.

4.3.2 Local Interactions

The candidate solution explores faster than the greedy approach in every case. However, the difference between the ability of the two algorithms to reduce the number of local interactions is more drastic, as seen below in Figure 4.2. The candidate solution causes the robots to rarely interact outside of the starting area as long as the robots can communicate when they

are close to one another. For full communication and limited communication, the greedy approach has more than 20 times the number of local interactions as the candidate solution, and more than 15 times the non-predictive variation of the candidate solution. Even with no communication at all, the candidate solution has less than a third of the local interactions of the greedy approach simply because it is inherently less deterministic and thus the robots tend to follow one another less.

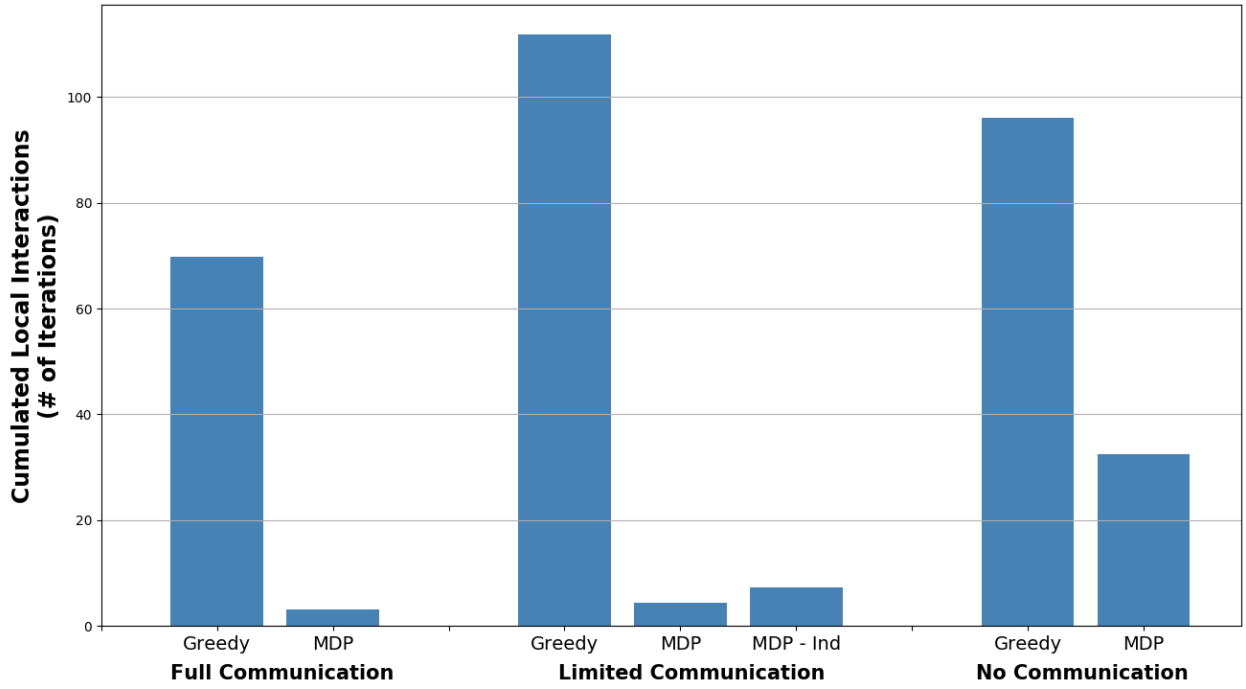

Figure 4.2: The number of iterations where the two robots are locally interacting for all 7 tests.

Interestingly the number of local interactions for the greedy approach is least with full communication and highest when communication is limited. This is the opposite result of the exploration rate, where the greedy approach with limited communication outperformed the full communication scenario. This may be due to the very high variability in the greedy tests; the greedy approach performs terribly if the robots synchronize at some point during operation, as they begin targeting the same reward and lose all the benefit of having two robots. However, if the robots never synchronize, the greedy approach performs similarly to the candidate solution. Since the greedy approach has extremely high variability compared to the consistency of the candidate solution, the conflicting results relating to the greedy algorithm with full and limited communication may be due to chance. The variability in the results is further explored in the

next section, **4.4.3 Variability**.

More tests would have to be performed to confirm these results more precisely, but it is clear from these results that local interactions are dramatically reduced when using the candidate solution as compared to the greedy approach. This is further visualized by observing sample trajectories for the approaches in section **4.3.4 Analyzing Trajectories.**

4.3.3 Variability

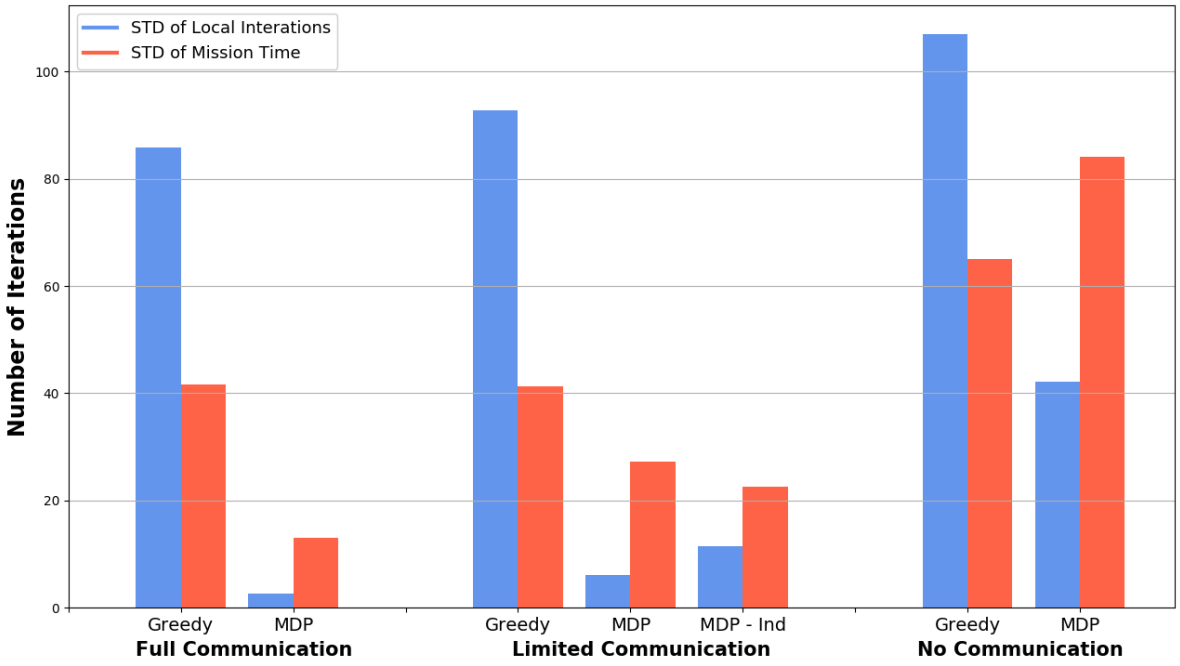

Figure 4.3: Standard deviation of both cumulated number of local interactions and total mission time for all 7 tests.

Generally, the greedy approach had far more variability in terms of the number of local interactions and total mission time compared to the candidate solution. As can be seen above in Figure 4.3, the candidate solution was far more consistent in the number of local interactions than the greedy approach, especially with full and limited communication. Also, the candidate solution was more consistent than the greedy approach with full and limited communication in terms of total mission time but surprisingly had more variation when there was no communication. This may be because the worst-case mission runtime of the greedy approach is bounded by greedy single robot exploration, which only requires one pass through the environment. However, the worst-case runtime for the single robot candidate solution involves

passing through the environment twice since it may need to return to every area to collect small rewards that were previously ignored. Due to this, the variability in the no communication case of the candidate solution is very high.

4.3.4 Analyzing Trajectories

To better understand the results above, the trajectories of the robots were plotted for random runs of each test. Some indicative trajectories are below.

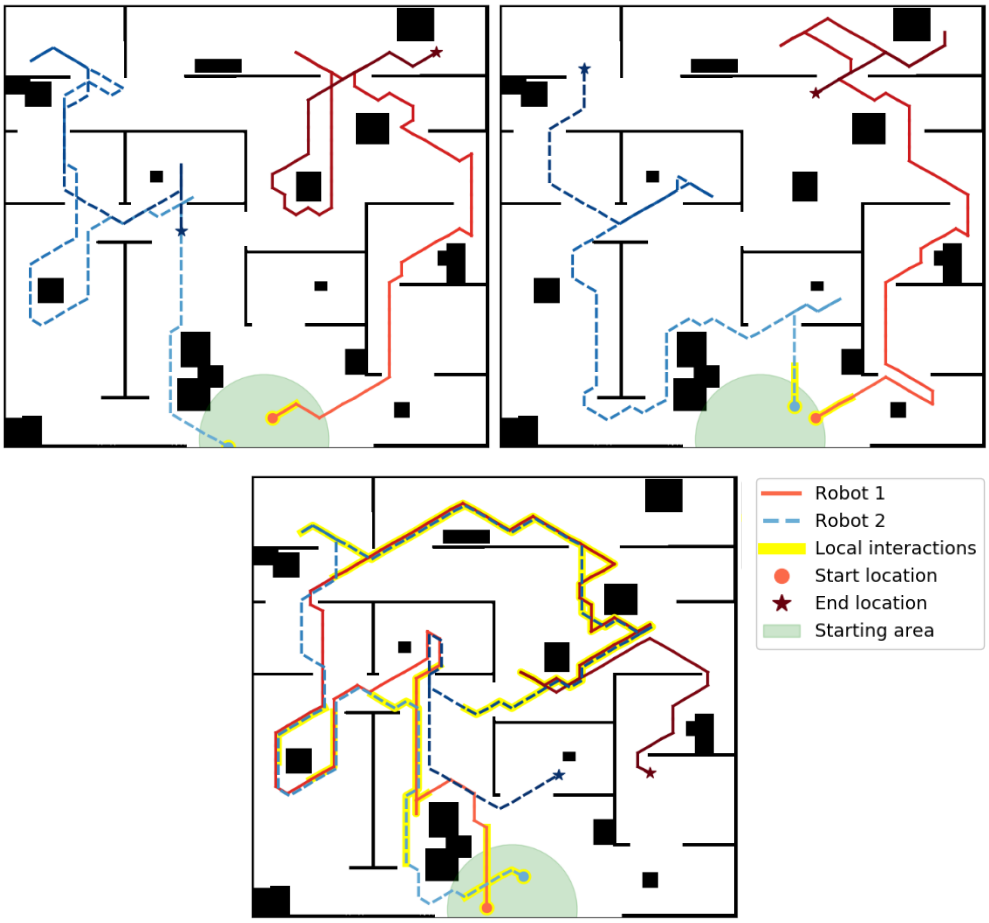

Figure 4.4: Trajectories for the candidate solution. (top left) Full communication. (top right) Limited communication. (bottom) No communication.

As can be seen in the sample trajectories for the candidate solution in both full and limited communication, it very effectively moves the robots away from each other. The only local interaction occurs at the beginning of the trajectory in the starting area. The non-predictive variation results in similar trajectories. However, in the case of no communication, the robots

have overlapping trajectories that cause them to locally interact on many occasions. Even these overlapping trajectories are more varied than the nearly identical trajectories of the greedy robots when they become synchronized, as can be seen in Figure 4.5.

Figure 4.5: Trajectories for the greedy approach. (top left) Full communication. (top right) Limited communication. (bottom) No communication.

The greedy approach sometimes explores very efficiently, such as in the full communication example above. However, the robots often become synchronized such as in the limited communication example and follow nearly identical trajectories. Due to this, the greedy approach has very high variability regardless of communication. In the no communication example, the robots merely chose similar routes since the exploration of one does not affect the other and their maps are not being merged. However, the no communication tests have the highest variability in number of local interactions and total mission time since the performance of the algorithm is completely dependent on starting locations.

4.4 Limitations

The candidate solution has been shown to effectively increase exploration rate and reduce local interactions drastically regardless of communication. Additionally, it accomplishes these results more consistently than the greedy approach. However, it has significant limitations that require increased effort to resolve. The algorithm needs to be tuned sufficiently well that it explores consistently without getting stuck. Additionally, the algorithm's computation needs to be limited such that it is tractable, but the value function needs to be sufficiently accurate to guarantee effective exploration. These two limitations are explored below.

4.4.1 Tuning Parameters

The candidate solution has many tuneable parameters which need to be chosen carefully to ensure success. Incorrectly tuning these parameters lead to the robots becoming stuck, either because the value function at their location was too flat and thus the robots did not move or because the robots blocked each others' path and could not coordinate how to move past each other. Several of these problems were fixed by automatic tuning methods. A list of tuneable parameters, their effects, and any automatic tuning methods are described below. Additionally, local coordination methods described in section **4.5.2 Local Coordination** would assist when robots need to move around each other assuming they can communicate.

1. **Radius of Rewards:** This is the hexagon radius to which rewards are propagated. As can be seen in Figure 3.3 it was chosen to be 2. A larger radius of reward propagation may allow the robot to find rewards faster as whenever it enters a hexagon with rewards it pauses its main algorithm and instead turns to scan the relevant unknown hexagons. This was not tested and likely depends on the size of the overall map.
2. **Weight of Rewards:** The amount of reward that is propagated is irrelevant in the single robot case, as only the ratio between rewards matters. However, in the multi-robot case, the balance between positive rewards and negative rewards around other robots needs to be tuned. Three sources of rewards need to be tuned: the positive rewards near frontiers ($R_{exploration}(s_i)$) in equation 1, the negative rewards ($R_{i,repulsive}(s_i)$) used to estimate the value functions of other robots in equation 4, and the weighting of other robots (f_{ij}) in the DVF formulation of equation 5. It was found that, regardless of tuning, when

rewards were few or far the negative rewards overpowered the positive rewards, causing the robots to ignore rewards. This was balanced by making sure the positive rewards were scaled to always be more than twice the cumulated value of negative rewards, which completely prevented the robots from getting stuck.

3. **Free Hexagon Balance:** In [10], a hexagon is considered free if the ratio between free and unknown pixels in that hexagon is greater than a threshold T_r. The smaller this ratio, the more optimistic the algorithm is that a partially explored hexagon is truly free. This speeds up exploration but occasionally causes the robot to collide with walls and produces less accurate maps.
4. **Discount Factor:** The discount factor controls how highly the robots weigh the importance of future events. A small discount factor makes the robot more greedily search for close rewards, while a discount factor near 1 makes the robot consider distant rewards more highly. As described in section **3.4 Implementing the Markov Decision Process,** the discount factor needs to increase as the path length to the nearest reward increases. The discount factor was set according to the following equation:

$$\gamma = 1 - 80/max(20, pathLengthToClosestReward)^2 \quad (7)$$

   Using this equation, the discount factor is set to 0.8 when rewards are close, but as rewards become very far the discount factor approaches 1. This solved the issue of the robots getting stuck due to the value function becoming flat very far from rewards.
5. **Noise:** Noise controls the probability that a robot's action will fail and instead a different random action will be performed. It was found that having any noise greatly slowed the computation of the value function, and thus for testing noise was set to zero. For real operation, it would need to be set to a realistic value that reflects the real possibility that certain actions will fail. For example, if it is known that forward movement often fails due to a sandy floor, it should be reflected in the transition function. This will likely involve a more complex formulation of the transition function than a single noise value for all actions.
6. **Computation Parameters:** Several parameters balance accurately computing the value function with computation time. These include the maximum number of iterations used in calculating the value function, the convergence criteria, the horizon H described in

section **3.4 Implementing the Markov Decision Process**, and the *exploration horizon* to consider when predicting the future destinations of other robots as described in section **3.5 Multi-Robot Coordination**. These had to be significantly limited to make testing possible within the time limit, but the algorithm's performance was not significantly hindered. For larger maps, higher noise, or more robots, it may be important to drastically change these parameters as more computation will be necessary to accurately estimate the value function. However, if the code is parallelized, as described in section **4.5.4 Parallelization,** the code may run fast enough that limiting accuracy will no longer be necessary.

4.4.2 Computational Cost

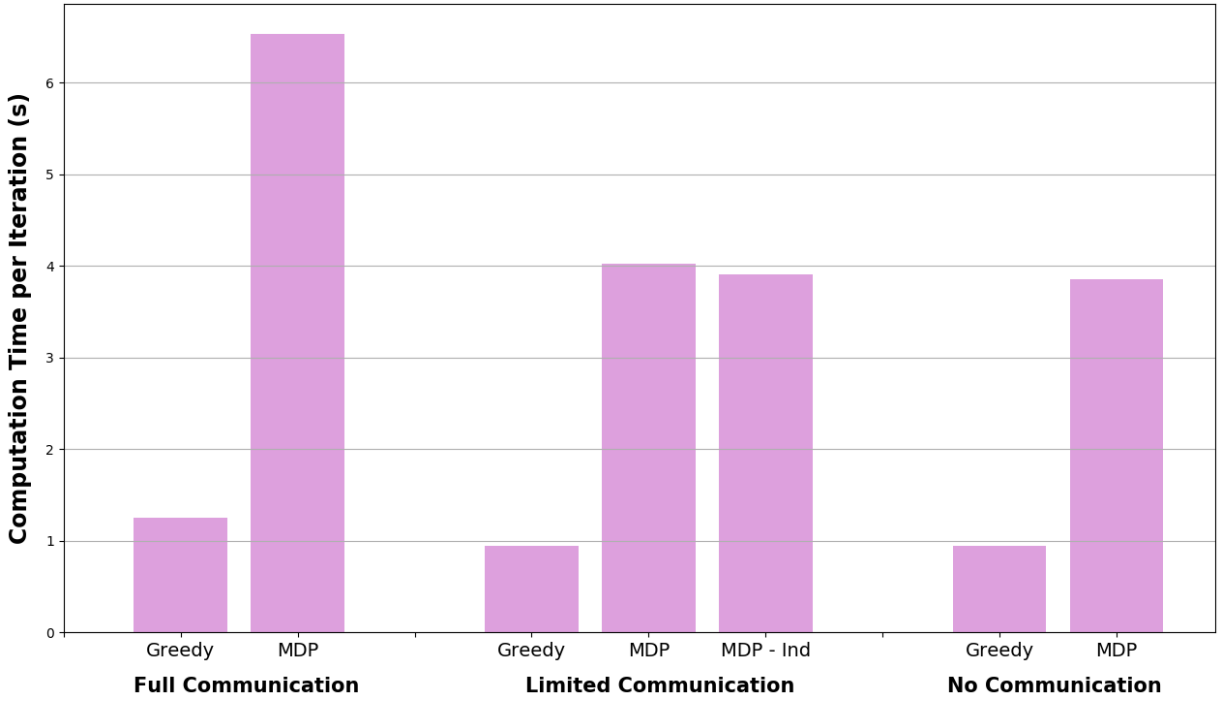

Figure 4.6: Computation time per iteration of the algorithm for all 7 tests.

The candidate solution is more computationally expensive than simpler methods, such as the greedy approach. When fully communicating, the candidate solution takes almost 6 times more time per iteration than the greedy approach. This decreases significantly when communication is limited because the algorithm does not require recalculating the value function of neighbouring robots as often. The difference in computation time between the full candidate

solution and the non-predictive variation is very minor, demonstrating that the additional step of prediction is computationally inexpensive.

When implemented on real robots, several factors would change the computational complexity of the candidate solution. Firstly, the computation time reported in Figure 4.6 was determined when the robots are operating sequentially and with no parallelization. On real robots, the computation would be split between robots and different aspects of the algorithm would be parallelized. This is discussed in section **4.4.4 Parallelization.** This should drastically reduce the computation time and make the algorithm more robust to larger teams as each robot would calculate its value function simultaneously rather than sequentially. However, larger teams would also mean that each robot needs to estimate the value function of many robots and merge in the maps of many robots, significantly increasing the computation time. Also, the addition of noise in the model significantly slows down convergence when calculating the value function, as discussed in section **4.4.1 Tuning Parameters**. For efficient use of this algorithm when applied to large teams in large environments, it may be necessary to use a programming language that can execute far faster, such as C++.

4.5 Future Work

4.5.1 Simulating Large Robots

Most robots are not sufficiently small to fit inside the hexagons. [10] advises on using hexagons that are smaller than half the size of a doorway, approximately 15cm in diameter. This is because if two hexagons are large enough to cover a doorway and the edges of the hexagons overlap with the door frame it will cause both hexagons to become occupied, and thus the doorway will appear like a solid wall. In general, this means robots will typically occupy several hexagons. The robot used in the candidate solution covers 18 hexagons (see Figure 4.1).

Larger robots increase the risk of colliding with obstacles and walls. To account for this, the transition function defined in section **3.4 Implementing the Markov Decision Process** needs to be modified. For a single hexagon robot, the transition function only checks the robot moved to an adjacent free hexagon. For a larger robot, it must consider that the entire area of the robot is free as it moves. To accomplish this, a robot mask is introduced, defined as the hexagons that contain the robot. If any of these hexagons would hit an occupied or unknown hexagon during movement, the transition function will indicate this action will fail. The mask can be

increased to guarantee no collisions will occur.

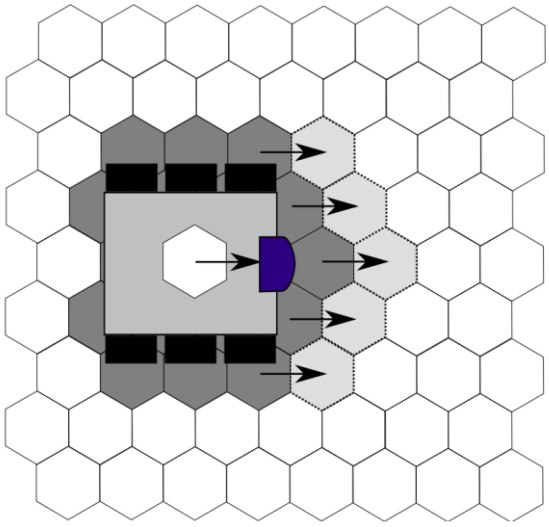

Figure 4.7: The dark grey hexagons represent the robot mask, and the light grey hexagons show the required free cells for the 'forward' action to succeed. The white hexagon in the centre of the robot is used to represent its state.

4.5.2 Local Coordination

The candidate solution managed to mitigate local interactions, almost never interacting beyond the starting location as long as the robots can communicate when they are near each other. However, two scenarios may cause the robots to interact.

Firstly, communication may fail entirely, causing the robots to collide. Since there is no way for the robots to realize there is another robot nearby (short of computer vision or other sensors), the robots will rely on anti-collision systems (e.g. lidar, bumper sensors) to prevent and react to collisions.

Secondly, if the robots need to get close to one another to pass through a doorway or narrow passageway and can communicate, a local coordination strategy should be employed. [25] uses a multi-agent MDP [39] to coordinate the robots. This results in a variety of behaviours depending on the situation, such as the robots following each other through a narrow corridor, or a robot freezing or backtracking to allow another robot to pass. For real operation, this method or another local coordination method will need to be employed.

4.5.3 Realistic Mapping

Currently, the implemented algorithm does not dynamically increase the map size if the robot encounters an area beyond the initial map size. Also, it assumes ideal mapping in that the sensors have no noise and completely scan an area. For real operation, a realistic mapping technique would need to be implemented. However, this would also require a more robust map merging strategy. Many algorithms for map merging exist, so it is necessary to choose an appropriate algorithm. Finally, the code would need to be updated such that the map dynamically increases in size when the robot encounters an area beyond the original estimated map size.

Several modules already exist in ROS for realistic mapping and map merging, such as *gmapping* [42] and *multirobot_map_merge* [43]. These should replace the written code as they are more realistic and robust.

4.5.4 Parallelization

Currently, all the code is synchronous and sequential. The first robot communicates to the second, which updates its map, then the second communicates to the first robot, which updates its map. Then the first robot calculates its value function, moves, and scans its new environment. Then the second robot does the same. This is unrealistic for two reasons.

Firstly, it would be inaccurate to assume that the robots are working synchronously, as in real operation each robot would have an onboard computer. Since the robots would not be synchronized, the code must be changed to account for the fact that communications from other robots may come at any time. This is simplified in ROS since ROS has powerful libraries for distributed computation.

Secondly, mapping, moving, and decision-making for each robot should be separated into independent threads as described by the candidate solution [10]. This allows each robot to move and scan simultaneously while also continuously computing its value function. This should speed up the computation by making use of parallel processing.

4.5.5 Moving to ROS

To create realistic simulations in three-dimensional environments, as well as complete implementation for real robots, this algorithm should be implemented with the Robot Operating

System (ROS). The current implementation has been designed to be sufficiently modular that several sections can be replaced entirely by ROS modules, such as mapping being replaced by *gmapping* [42] and map merging being replaced by a module such as *multirobot_map_merge* [43]. Other aspects will also have to be changed to connect with the robots' sensors and actuators, including scanning and movement. However, the core algorithm should not have to be changed as it is a high-level decision-making algorithm that calculates the value function on the hexagon layer. Since ROS is compatible with Python 2.7, which was used to program this implementation, major sections of the currently implemented code should be able to be ported to ROS without change.

4.5.6 Adapting the Algorithm for USAR Scenarios

The candidate solution is designed for general multi-robot exploration, not specifically for USAR. The solution could be adapted to not only explore the area but focus on finding victims. This can be done by increasing the reward near potential victims. These victims may be identified through various methods in a complete implementation, such as computer vision. The algorithm will then increase the reward around potential victims such that the robot approaches sufficiently close to make a positive identification. Other tasks may also be created, such as speaking with targets or communicating information back to human crisis managers. The requirements of the USAR algorithm will depend on the situation and robot capabilities. The candidate solution can be modified with relative ease for new tasks if the tasks can be defined or encouraged by changing rewards, such as approaching victims. Tasks that require more than modifying the reward function would be difficult to implement with the candidate solution.

5 Conclusion

Multi-robot teams have the potential to greatly assist in urban search and rescue operations [2, 3]. Many algorithms have been proposed to coordinate robots to efficiently explore; however, the challenges involved in real USAR render many of these algorithms inadequate. In section **2 Literature Review,** the field of multi-robot exploration and coordination was reviewed and an algorithm [25] appropriate to the limited communication found in USAR [22] was selected. In section **3 Design and Implementation,** this algorithm was implemented in a grid world in Python. By combining a decentralized decision-making architecture with an implicit coordination strategy, the algorithm effectively coordinates teams of robots regardless of communication limitations. It uses a Markov decision process planning approach with positive rewards near frontiers and negative rewards around known and predicted locations of other robots to maximize the area searched while minimizing exploration overlap.

Experiments with multiple levels of communication in section **4 Results and Discussion** demonstrated that the chosen algorithm explores faster than a naïve approach when it can communicate with nearby robots. Additionally, the algorithm greatly reduces the number of local interactions between robots in all communication levels. Due to its simplicity, scalability to larger teams, and robustness to communication failure, this algorithm will serve as a benchmark against which to compare future multi-robot USAR research. The next steps for this research are to implement the algorithm in the Robot Operating System for more realistic simulation and eventual deployment on physical robots.

6 References

[1] G. D. Cubber, et al, “Chapter 1: Why do we need search and rescue robots?” in Introduction to the Use of Robotic Tools for Search and Rescue, IntechOpen, Aug-2017. Available: https://www.intechopen.com/books/search-and-rescue-robotics-from-theory-to-practice/introduction-to-the-use-of-robotic-tools-for-search-and-rescue

[2] Liu, Y., Nejat, G., “Robotic Urban Search and Rescue: A Survey from the Control Perspective,” Journal of Intelligent & Robotic Systems, Mar-2013. Available: https://link.springer.com/article/10.1007/s10846-013-9822-x

[3] J. Wong and C. Robinson, “Chapter IV: Findings” in Urban Search and Rescue Technology Needs, National Criminal Justice Reference Service, Nov-2004. Available: https://www.ncjrs.gov/pdffiles1/nij/grants/207771.pdf.

[4] N. J. Zhi Yan, “A Survey and Analysis of Multi-Robot Coordination” (Section 8: Decision- making: Centralized Versus Decentralized), International Journal of Advanced Robotic Systems, Jan-2013. Available: https://journals.sagepub.com/doi/10.5772/57313.

[5] Omidshafiei S, Agha–Mohammadi A, Amato C, Liu S, How JP, Vian J, “Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions”, The International Journal of Robotics Research. March-2017. Available: https://journals.sagepub.com/doi/full/10.1177/0278364917692864

[6] Botelho, Silvia & Alami, Rachid. “M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement”, Proceedings - IEEE International Conference on Robotics and Automation. Feb-1999. Available: https://www.researchgate.net/publication/3802817_M_a_scheme_for_multi-robot_cooperation_through_negotiated_task_allocation_and_achievement

[7] Zhi Yan, Nicolas Jouandeau, Arab Cherif, “Multi-robot Decentralized Exploration Using A Trade-based Approach”, 8th International Conference on Informatics in Control, Automation and Robotics. Jul-2011. Available: https://hal.archives-ouvertes.fr/hal-02311608/

[8] Capitan J, Spaan MTJ, Merino L, Ollero A, “Decentralized multi-robot cooperation with auctioned POMDPs”, The International Journal of Robotics Research. Jun-2013. Available: https://journals.sagepub.com/doi/10.1177/0278364913483345

[9] Z. Xu, R. Fitch and S. Sukkarieh, "Decentralised coordination of mobile robots for target tracking with learnt utility models," IEEE International Conference on Robotics and Automation, Karlsruhe. 2013. Available: https://ieeexplore.ieee.org/document/6630846

[10] Simon Le Gloannec, Laurent Jeanpierre, Abdel-Illah Mouaddib, “Unknown Area Exploration with an Autonomous Robot using Markov Decision Processes”, TAROS. 2010. Available: https://hal.archives-ouvertes.fr/hal-01438198

[11] Fox, Dieter & Ko, Jonathan & Konolige, Kurt & Limketkai, Benson & Schulz, Dirk & Stewart, Benjamin, “Distributed Multi-Robot Exploration and Mapping”, Proceedings of the IEEE. Jan-2005. Available: https://www.researchgate.net/publication/224773220_Distributed_Multi-Robot_Exploration_and_Mapping

[12] Khalil Mohamed, Ayman El Shenawy, Hany Harb, “A Hybrid Decentralized Coordinated Approach for Multi-Robot Exploration Task”, The Computer Journal, Volume 62, Issue 9. Sep-2019. Available: https://doi.org/10.1093/comjnl/bxy107

[13] Viseras, Alberto & Xu, Zhe & Merino, Luis, “Distributed Multi-Robot Cooperation for Information Gathering Under Communication Constraints”, International Conference on Robotics and Automation. Apr-2018. Available: https://www.researchgate.net/publication/324442098_Distributed_Multi-Robot_Cooperation_for_Information_Gathering_Under_Communication_Constraints

[14] Hollinger, Geoffrey & Singh, Sanjiv, “Multi-Robot Coordination with Periodic Connectivity”, Proceedings - IEEE International Conference on Robotics and Automation. Jun-2010. Available: https://www.researchgate.net/publication/224155746_Multi-Robot_Coordination_with_Periodic_Connectivity

[15] Nestmeyer, Thomas & Franchi, Antonio & Bülthoff, Heinrich & Giordano, Paolo, “Decentralized Multi-target Exploration and Connectivity Maintenance with a Multi-robot

System", Autonomous Robots. May-2015. Available: https://www.researchgate.net/publication/277023216_Decentralized_Multi-target_Exploration_and_Connectivity_Maintenance_with_a_Multi-robot_System

[16] S. Kemna, J. G. Rogers, C. Nieto-Granda, S. Young and G. S. Sukhatme, "Multi-robot coordination through dynamic Voronoi partitioning for informative adaptive sampling in communication-constrained environments," IEEE International Conference on Robotics and Automation. 2017. Available: https://ieeexplore.ieee.org/abstract/document/7989245

[17] Liu C., Kroll A, "A Centralized Multi-Robot Task Allocation for Industrial Plant Inspection by Using A* and Genetic Algorithms", Artificial Intelligence and Soft Computing. 2012. Available: https://link.springer.com/chapter/10.1007/978-3-642-29350-4_56

[18] Gerkey, Brian P. and M. Mataric, "Sold!: auction methods for multirobot coordination.", IEEE Transactions on Robotics and Automation. 2002. Available: https://www.semanticscholar.org/paper/Sold%21%3A-auction-methods-for-multirobot-coordination-Gerkey-Mataric/73fa4688e14e23a304b907957701c49d0ee13de7

[19] R. Luna and K. E. Bekris, "Efficient and complete centralized multi-robot path planning," International Conference on Intelligent Robots and Systems. 2011. Available: https://ieeexplore.ieee.org/abstract/document/6095085

[20] J. de Hoog, S. Cameron and A. Visser, "Dynamic team hierarchies in communication-limited multi-robot exploration," IEEE Safety Security and Rescue Robotics, Bremen. 2010. Available: https://ieeexplore.ieee.org/abstract/document/5981573

[21] Hollinger, Geoffrey & Singh, Sanjiv & Djugash, Joseph & Kehagias, Athanasios, "Efficient Multi-Robot Search for a Moving Target". International Journal of Robotics Research. Feb-2009. Available: https://www.researchgate.net/publication/220121820_Efficient_Multi-Robot_Search_for_a_Moving_Target

[22] R. Murphy, J. Casper, J. Hyams, M. Micire and B. Minten, "Mobility and sensing demands in USAR," IEEE International Conference on Industrial Electronics, Control and Instrumentation. 2002. Available: https://ieeexplore.ieee.org/document/973139

[23] Anderson, M., Papanikolopoulos, N, “Implicit Cooperation Strategies for Multi-robot Search of Unknown Areas”, Journal of Intelligent Robotic Systems. 2008. Available: https://link.springer.com/article/10.1007/s10846-008-9242-5

[24] Benavides, Facundo & Ponzoni Carvalho Chanel, Caroline & Monzón, Pablo & Grampín, Eduardo, “An Auto-Adaptive Multi-Objective Strategy for Multi-Robot Exploration of Constrained-Communication Environments”. Applied Sciences. Feb-2019. Available: https://www.researchgate.net/publication/330983636_An_Auto-Adaptive_Multi-Objective_Strategy_for_Multi-Robot_Exploration_of_Constrained-Communication_Environments

[25] Laëtitia Matignon, Laurent Jeanpierre, Abdel-Illah Mouaddib, “Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes”, Association for the Advancement of Artificial Intelligence (AAAI) Conference. 2012. Available: https://hal.archives-ouvertes.fr/hal-00971744/

[26] Wei, C., Hindriks, K.V. & Jonker, C.M, “Dynamic task allocation for multi-robot search and retrieval tasks”, Applied Intelligence. 2016. Available: https://link.springer.com/article/10.1007/s10489-016-0771-5

[27] Brian Yamauchi, “Frontier-based exploration using multiple robots,” In Proceedings of the Second International Conference on Autonomous Agents. 1998. Available: http://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/integrated1/yamauchi_frontier_based_explor.pdf

[28] Bautin A., Simonin O., Charpillet F, “MinPos : A Novel Frontier Allocation Algorithm for Multi-robot Exploration”, Intelligent Robotics and Applications. 2012. Available: https://link.springer.com/chapter/10.1007/978-3-642-33515-0_49

[29] Mottaghi, R., Vaughan, R, “An integrated particle filter and potential field method applied to cooperative multi-robot target tracking”, Autonomous Robotics. 2007. Available: https://link.springer.com/article/10.1007/s10514-007-9028-9

[30] Liu, Tsung-Ming & Lyons, Damian, “Leveraging area bounds information for autonomous decentralized multi-robot exploration”, Robotics and Autonomous Systems. Aug-2015.

Available: https://www.researchgate.net/publication/282623801_Leveraging_area_bounds_information_for_autonomous_decentralized_multi-robot_exploration

[31] Das, Barnali & Couceiro, Micael & A.Vargas, Patricia. “MRoCS: A new multi-robot communication system based on passive action recognition”, Robotics and Autonomous Systems. Aug-2016. Available: https://www.researchgate.net/publication/321463218_MRoCS_A_new_multi-robot_communication_system_based_on_passive_action_recognition

[32] G. A. Cardona, D. Yanguas-Rojas, M. F. Arevalo-Castiblanco and E. Mojica-Nava, "Ant-Based Multi-Robot Exploration in Non-Convex Space without Global-Connectivity Constraints," 18th European Control Conference (ECC). 2019. Available: https://ieeexplore.ieee.org/abstract/document/8796034

[33] Matignon, Laëtitia & Jeanpierre, Laurent & Mouaddib, Abdel-illah. “Distributed Value Functions for Multi-Robot Exploration: a Position Paper”, IEEE International Conference on Robotics and Automation. 2011. Available: https://www.researchgate.net/publication/235642260_Distributed_Value_Functions_for_Multi-Robot_Exploration_a_Position_Paper

[34] Laëtitia Matignon, Laurent Jeanpierre, Abdel-Illah Mouaddib, “Distributed Value Functions for Multi-Robot exploration”, IEEE International Conference on Robotics and Automation. 2012. Available: https://hal.archives-ouvertes.fr/hal-00969562

[35] Laëtitia Matignon, Laurent Jeanpierre, Abdel-Illah Mouaddib, “Decentralized Multi-Robot Planning to Explore and Perceive”, Workshop on Multi-Agent Coordination in Robotic Exploration, European Conference on Artificial Intelligence. 2014. Available: https://hal.archives-ouvertes.fr/hal-01497823

[36] R. Bellman, “Dynamic programming : Markov decision process,” Princeton University Press. 1957.

[37] J. E. Bresenham, "Algorithm for computer control of a digital plotter," IBM Systems Journal. 1965. Available: https://ieeexplore.ieee.org/abstract/document/5388473

[38] J. Schneider, W.-K. Wong, A. Moore, and M. Riedmiller, “Distributed value functions”, Proceedings of the Sixteenth International Conference on Machine Learning. 1999. Available: https://www.semanticscholar.org/paper/Distributed-Value-Functions-Schneider-Wong/bd34c45174e121ae7fb495c3758a7de4e6c966c4

[39] Boutilier, C, “Planning, learning and coordination in multiagent decision processes”, In TARK. 1996. Available: https://www.cs.toronto.edu/~cebly/Papers/tark96.pdf

[40] H. Zhang, J. Chen, H. Fang and L. Dou, "A Role-based POMDPs Approach for Decentralized Implicit Cooperation of Multiple Agents", 13th IEEE International Conference on Control & Automation. 2017. Available: https://ieeexplore.ieee.org/document/8003110

[41] Yan Z, Jouandeau N, Cherif A, “A Survey and Analysis of Multi-Robot Coordination”, International Journal of Advanced Robotic Systems. Dec-2013. Available: https://journals.sagepub.com/doi/full/10.5772/57313

[42] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard, “Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters”, IEEE Transactions on Robotics. 2007. Available: https://ieeexplore.ieee.org/document/4084563

[43] Jiří Hörner, “Map-merging for multi-robot system”, Thesis from Charles University in Prague, Faculty of Mathematics and Physics. 2016. Available: https://is.cuni.cz/webapps/zzp/detail/174125/

[44] Python Software Foundation, “DictionaryKeys – Python Wiki”, Python Wiki. 2021. Available: https://wiki.python.org/moin/DictionaryKeys