Independent Activity and Local Opportunity for Dynamic Robot Team Management in Dangerous Domains

by

Seth Fiawoo

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
July 2019

Thesis advisor

John E. Anderson

Author

Seth Fiawoo

Independent Activity and Local Opportunity for Dynamic Robot Team Management in Dangerous Domains

Abstract

Dangerous domains are a challenge for teams of heterogeneous robots, since robot losses may involve the loss of particular skills that might be rare in the domain. Previous research has resulted in a framework that allows teams to rebalance and recruit from the environment. However, there is currently no consideration of situations where agents may at times provide more useful work globally by not joining a team, or situations where it might be discovered that types of work might be associated with a given locality. My thesis extends this framework to give agents the ability to refuse to join teams and work for times on their own, by considering current satisfaction in the use of their skills, the likely rarity of their skills, and the distribution of places those skills are used in the environment. I examine this work in a simulated Urban Search and Rescue domain. My results show that in scenarios where a robot's special skills are rare and tasks requiring those skills are only available at a few fixed locations in the environment, a robot is more useful if it suspends its team commitment to make itself available to all teams.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank God for seeing me through this chapter; it wasn't without its challenges, but, His sustenance pulled me through. Next, I would like to thank my advisor, Dr. John Anderson. Words cannot express how appreciative I am for your level of patience, guidance, support, and insight. Even during your recovery, when others would have taken a break, you were constantly available, using your superior reviewing skills to help me finalise a thesis I am very proud of. I am really grateful to have been your student. God bless you! A big thanks also to my committee members for taking time of their schedules to review this thesis and provide useful insights.

Finally, I would like to thank family and friends for all their support and encouragement during this process of obtaining my MSc.

This thesis is dedicated to God, family, and friends who never questioned my ability to successfully complete this journey. Thank you for all the love, patience, kindness, support and encouragement.

Chapter 1

Introduction

1.1 Chapter Overview

This chapter gives some background information on my research as well as introducing concepts and terminology that are used throughout my thesis. Next, I present my motivation for conducting this research and the questions my research will answer. Finally, I give an overview of my solution approach and an outline of the rest of my thesis.

1.2 Introduction

In the wake of a disaster, Urban Search and Rescue (USAR) teams arrive on the scene to rescue victims that may be injured or trapped in damaged urban structures. Such an environment is very dangerous to human rescuers (e.g. fire, further structural collapse) [Murphy, 2000]. To minimise the risk to human life, one avenue of research involves substituting teams of robots in place of humans, and having those robots search and map the existing structure and search for victims [Arkin and Berkey, 1997]. This is a strong challenge to artificial intelligence (AI) and robotic technology, in that the environment is unpredictable, changes dynamically, and is equally as dangerous to robots as humans [Kemp et al., 2007; Anderson and Papanikolopoulos, 2008]. This danger makes the use of heterogeneous teams important: the likelihood of losing any one robot is high, and cost can be better balanced with risk by having fewer more elaborate units and additional cheaper, simpler robots that can be more easily risked [Parker and Tang, 2006; Gage et al., 2004; Guerrero and Oliver, 2012]. The unpredictable nature of the domain also demands a broad skill set that is difficult to expect from any single robot [Kiener and von Stryk, 2007].

The nature of this domain presents an extreme challenge to the management of a team [Parker and Kannan, 2006; Rogers III et al., 2013]. As robots are damaged or destroyed, teams must adapt, and must also be able to integrate new members over time (e.g. encountering a robot lost from another team, or joining with robots that are released specifically as replacement units) [Gunn and Anderson, 2015]. The domain itself also adds technological challenges: communication can be sporadic and unreliable, and teams must do the best they can under conditions where information (e.g. assignment of a task, a notification of a new team member) may not be received by all members. Previous work in our lab [Gunn and Anderson, 2013] has resulted in a framework for collaboration that supports dynamic teamwork under the assumption that robots will be lost or destroyed, possibly found or replaced, and that roles on a team (including leadership) must sometimes be reassigned to best suit current membership. The framework operates in a distributed fashion, with local encounters driving both task discovery and membership changes on and among teams (discovering new agents, realizing agents are lost, encountering teams and balancing membership with them) [Gunn and Anderson, 2015]. Task assignment (i.e. domain work) proceeds dynamically as the mixture of robots on a team changes [Gunn and Anderson, 2013]. All this is done under conditions of limited communication reliability. More recently, we have moved to more active forms of recruitment in teams, recognizing that a reliance on simply encountering others will not be enough for situations where agents with a specific skill or ability are desperately needed. We have added support for a range of active recruitment strategies. These strategies involve varying degrees of effort, from sending recruitment messages while doing useful work to an active physical search for an agent with a specific ability [Nagy and Anderson, 2016]. They also vary in commitment: task-based recruitment borrows a robot to do one specific task, while role-based recruitment involves switching teams to take on a particular role [Nagy and Anderson, 2016]. In all the work thus far, there has been the assumption that members are always available to be on some team, and that it always makes sense for a lone agent to join a team when asked. There are a number of important situations in which this is not the case, however (beyond settings that do not apply here, such as when team members cannot trust one another [van de Vijsel and Anderson, 2004]).

1.3 Motivation

In a situation where some skills are rare, multiple teams may be competing for the same few agents, and teams moving a significant distance away from one another cannot easily have members be temporarily recruited or otherwise shared with other teams, possibly leading to lower global performance. From an individual's standpoint, a skill might be crucial but needed infrequently, meaning joining a team would lead to the robot spending much time moving with a team and using its rare skill only infrequently. An example of this in USAR is an agent equipped for fire-fighting: fires may need to be put out for a team to move through a given area, but may be encountered only sporadically. Such a robot could still be of use to a team for other activities, but a possibly rare skill would still be underutilized. Permanently travelling with such an agent is also not necessarily in the best interests of the team, since work must be devoted to coordination at all times even when the skill is not required.

While fires may be unpredictable, other activities may be spatially constrained. If a robot can lift others up a flight of stairs, for example, the utility of travelling with a team depends on the frequency with which climbable areas are encountered, just as the fire-fighting example above. Beyond this, however, there may be only one or two fixed places to climb over rubble and stairs, in which case it would be better to remain near those locations rather than travelling with a team. Conversely, once time has passed with little work (e.g. teams have moved on to other floors), remaining in that location would be less productive.

Balancing the possibility of being on a team with independent activity in both the above situations requires a sense of self-motivation beyond simply having a shared goal: it involves knowing one's particular skills are being put to good use. The latter of the two examples above also involves looking at the locations where skills are being used to possibly better leverage them in cases where those skills are not plentiful. The purpose of my thesis research is to extend our framework to support these types of activities, and to study these phenomena in the USAR domain.

The framework as it existed before my own work supports a robot being able to switch into an open role on a team for which it is more suited, and provides a range of effort indicating how much energy a robot should spend in trying to recruit other robots for tasks it cannot perform. The extended framework supports a spectrum of possibilities for team-based activity, from always joining a team and moving with it [Gunn and Anderson, 2015], to being temporarily recruited [Nagy and Anderson, 2016], to deciding to operate independently where that is advantageous and shift to other points on this spectrum when circumstances change. The best point in this spectrum for any agent to be on depends on the range of skills in the environment and their rarity, which in turn depends on the milieu of agents currently in the environment and the changes in this population over time.

For example, if there is an influx of stair-climbing robots, individuals with this skill should recognise this over time and become more willing to stay with a team even if the work being done does not utilise this skill, and gravitate again to working at particular locations if this skill once again becomes rare. The intended outcome is a better use of robots both off and on teams, and a stronger global outcome while still relying on local decisions made over time.

1.4 Terminology

In this section, I describe some technical terms that I use throughout my thesis.

- *Task*—A task as used within my work, refers to a basic unit of useful work that can be performed by a single robot [Gunn and Anderson, 2015]. Some examples of tasks are extinguishing a fire, clearing debris, identifying a victim. [Krieger et al., 2000; Kiener and von Stryk, 2007]
- *Role*—For the purposes of my research, a role is defined as a position in a team a robot can fill and is described by the types of tasks a robot filling that position can perform [Gunn and Anderson, 2015]. An example is a fire-fighting role. A robot occupying this role should be able to put out fires, navigate uneven terrain and through debris and note potential victims as the robot moves [Nagy, 2016].
- *Unique skill*—There are certain tasks within the environment which are only capable of being fulfilled by one particular type of robot. Any skill used while performing such a task makes the skill a unique one. For instance, only a DebrisBot may be able to remove debris and in that case, this makes the skill associated with clearing debris unique.
- *Satisfaction*—Within the context of my work, satisfaction is a measure of how much utility a robot derives from performing tasks within the environment. A robot gets satisfaction when it is able to complete tasks (whether discovered or assigned). Completing different tasks gives a robot a different satisfaction value. For instance, a robot performing a task that requires the use of a unique skill will gain more satisfaction from completing that task than completing a

task which does not make use of a unique skill. I track two kinds of satisfaction in my work: 1) a robot tracking its own satisfaction, and 2) a leader tracking its team members' satisfaction by counting the number of task assignment requests members have accepted.

- *Rare skill*—A dangerous domain such as a USAR environment constantly evolves over time and this can cause robots to break down. At some point, there might be certain tasks discovered where there are not enough robots to perform newly discovered tasks. I define a rare skill as one where there is an excessive number of requests compared to the number of robots readily available to satisfy those requests.

- *Team*—A team as used within my work, refers to a group of robots that have a varying range of abilities, that have incomplete knowledge of the team's membership and its collective abilities, and have a common goal. Every team has a leader responsible for coordinating members and assigning tasks to them. A robot on its own (lost robots and replacements) is considered to be a team of one, with that robot being the leader. All robot teams in my work have a common overarching goal and thus do not compete with other teams in the domain while working towards this goal [Krieger and Billeter, 2000; Pitonakova et al., 2014]. Team members in my work are not selfish [Dutta and Sen, 2003] and are always expected to accept tasks assigned to them (provided their task queue is not full). However, when a team member has a rare skill, it will reject task requests that will not make use of that rare skill so it can potentially serve other teams and reduce a bottleneck. While this raises issues of a robot being selfish, a robot cannot refuse tasks because harm might come to it while performing that task. Though it might be useful for robots within a dangerous domain to measure the risk associated with tasks before accepting them, that is beyond the scope of my thesis.

- *Recruitment*—Recruitment is the process of requesting that a task be performed, either by a specific robot or whichever is available, and is in itself a task [Gage et al., 2004]. Every recruitment request employs a different strategy ranging from chance encounters (passive) to broadcasting messages (concurrent) to performing a physical search (active), with the aim of finding an agent more suitable to take on a newly discovered task the recruiter cannot perform [Nagy and Anderson, 2016]. The above-mentioned range indicates how much effort a robot puts into finding a better suited robot for a newly discovered task. The points within this range defined here are referred to as *recruitment strategies* in my research. Robots in my framework are able to reject recruitment requests when the skill being requested is a rare one. Recruitment allows a team to acquire new skills temporarily (task-based) or permanently (role-based), in order to perform work that previously could not be completed by the team.

- *Willingness*— Depending on what a robot's satisfaction value is at any time, a robot may accept or reject recruitment requests. I define a robot's tendency to accept or refuse a recruitment request with consideration of its current satisfaction as the robot's *willingness.* In my work, there are three levels to a robot's willingness - *willing, somewhat willing, and not willing.* A robot is said to be willing if its satisfaction is so low that it believes it has to be aggressive (leaving its current team) about increasing its satisfaction. A somewhat willing robot has moderate satisfaction and is open to accepting recruitment requests. A robot is not willing to accept recruitment requests if it realises it has a currently rare skill.

1.5 Research Questions

In this section, I present the research questions I will be answering using the approach I describe in Section 1.6

1. **To what degree does tracking robot satisfaction affect the total teams' performance with respect to the overall mission in a USAR domain?**
2. **For specialised tasks that require the use of a robot's unique skill, is it more useful to have robots expending effort to stay with a team or should these efforts be redirected towards areas where its special skills may be needed?**

1.6 Solution Approach

My work builds on prior research done in our lab where teams of heterogeneous robots were able to dynamically balance their team membership by *role switching* and exchanging members based on physical encounters between teams [Gunn, 2011], and having robots put in varying levels of effort to search for better suited robots to fulfil some discovered task for which they themselves will not perform adequately [Nagy, 2016]. However, in all of this work, there was no consideration of whether a robot was useful within the context of contributing to the teams' overall goals. The focus was mainly on identifying what was lacking within an individual team, and trying to supply the missing robot skill.

My work focuses on a robot being able to track its utility within a team and use this knowledge acquired over time to determine whether it is better off staying on its current team, to be more or less willing to be recruited by another team, or ultimately leave its current team to become independent. In a similar vein, team leaders are also able to track how well a member is performing within the team and make an appropriate decision - keeping a member on a team or letting them go. A robot will also be able to identify when there is a shortage of its skill within the environment (whether its skill is in high demand). With access to such information, a robot can make more informed decisions (that is, making itself available to all teams if there is a global shortage of a skill it possesses, or staying with its team if none of its skills are in high demand). When a robot realises it is not being productive enough, it could be that the part of the environment being currently explored does not make use of its skill set and thus its satisfaction would diminish over time. What if the robot had visited a previous location where it had been useful? It makes sense that in trying to increase its satisfaction, it will go back to places where its skill was utilised the most (or to new areas that bear similarity to places where it was previously satisfied), with the aim of improving its satisfaction (there's probably more work to be done there, or places like that, and that work will bring satisfaction to the robot). Having a high utility at that location, a robot will be motivated to limit its activity to that location. Being at that location over a period, if the robot's satisfaction reduces, then naturally the robot will explore other avenues to improve its satisfaction. This could involve being more open to accepting recruitment requests or going to a location with a likely higher satisfaction.

My thesis extends an existing framework for dynamic team management to account for situations where agents may be more useful operating independently of a team, either because rare skills would be infrequently utilized moving with individual teams, or where skills are most useful in particular locations in the domain. The framework allows agents to be as independent as the evolving situation (including the current mix of tasks and robot types) demands, and increase global effectiveness.

1.7 Thesis Organisation

The remainder of my thesis is outlined as follows:

- **Chapter 2: Related Work**–Reviews existing literature related to my work with regard to team management, robot assessment and evaluation, and robot recruitment.

- **Chapter 3: Methodology**–In this section, I describe satisfaction, skill rarity and locality as it relates to my work. I also give an overview of my robot and team design.

- **Chapter 4: Implementation**–Provides a detailed description of components (robots, skills, and tasks) I implemented to enable me to evaluate my framework.

- **Chapter 5: Testing and Evaluation**– Describes experiments I used to evaluate my framework and discusses the results I obtained from running these experiments.

- **Chapter 6: Conclusion**–uses results from the preceding chapter to answer the research questions posed in Section 1.5 within the context of my thesis. I also discuss possible directions for future work.

Chapter 2

Related Work

The primary focus of my thesis is to develop and evaluate a framework that allows robots to estimate how useful they are within a team setting (i.e. contribution towards the overall team's goals) in a rapidly evolving domain such as USAR. A robot would use this information in conjunction with knowing the commonality of its skills in the environment and whether there is a region within the environment that is likely to require the use of any rare skill.

In this chapter, I introduce related background work and describe concepts from these works that are similar and also indicate how they differ from my work. I group the related works by similar concepts used. The first category is *dynamic team management and formation*. In this group, robots work in teams and dynamically adjust team membership when specific events occur in the environment [Gunn and Anderson, 2015; Dasgupta and Cheng, 2016; De Rango et al., 2018; Krieger et al., 2000].*Robot assessment and evaluation* deals with works that involve robots keeping track of information regarding how they are performing within the environment and taking the requisite actions to improve *performance* [Gage et al., 2004; Simonin and Ferber, 2000; Guzzi et al., 2018]. The final group deals with *recruitment in multi-robot systems* where robots will ask for assistance from other robots to complete some discovered task [Nagy, 2016; De Rango et al., 2018; Krieger et al., 2000].

2.1 Dynamic Team Management and Formation

One of the concepts pivotal to my work is team formation and maintenance. This is a really broad area of study and a huge variety of approaches and strategies have been used to form and maintain teams in order to accomplish goals. In this section I discuss previous works where team membership changes over time or based on some event occurring. That is, a robot may leave a team or a team may accept new members according to defined membership rules (for instance a robot not heard from after some time may be considered to have left the team [Gunn and Anderson, 2015]) or an event occurring, such as encountering an obstacle [Dasgupta and Cheng, 2016].

Kiener and von Stryk [2007] presented a framework for task allocation in teams of heterogeneous robots. They demonstrated their approach with a team comprising a wheeled Pioneer robot and a humanoid robot. The team's objective was to follow a ball down a hallway and kick it into a goal. The two robots maintained knowledge of their unique abilities and by using these at the right time, they were able to complete the task when neither of them could have done it on their own. Although this is similar to my research, there are significant differences. My work involves a higher number of tasks with varying priorities at different locations, and therefore task allocation decisions are not immediately obvious, unlike the above-mentioned work where the task was well-defined at the beginning of the team's mission and there was an obvious mapping as to which agent should do what.

Coviello and Franceschetti [2012] presented an algorithm for team formation with agents in one of two classes: leaders and followers. A leader recruits team members within its locality given a predefined constraint (the number of followers a leader can recruit). In forming a team, a leader sends requests to followers within its communication range and a follower can accept or reject a request. If a follower receives multiple team requests, the follower selects a team at random to join. Unlike my work, this does not involve making any rational decisions based on skill balance, task diversity, and agent population. It is also much less sophisticated than my framework in terms of the limited roles an agent can take on within a team.

Dasgupta and Cheng [2016] provided a framework for dynamically dividing up a team of robots into smaller sub-teams to navigate around arbitrarily-shaped obstacles. The concept of weighted voting games (every agent is assigned a weight and the group of agents whose total weight exceeds a given threshold, dictates the course of action [Elkind et al., 2007]) was applied. Once an obstacle was encountered, the robots divided themselves into smaller groups in such a way that each sub-team could navigate around the obstacle. Once all sub-teams were clear of the obstacle, they merged back into a larger team. The concept of maintaining formation around obstacles is well-explored (e.g. [Balch and Arkin, 1998; Fredslund and Mataric, 2002]). However, the approach presented by Dasgupta and Cheng [2016] results in fewer sub-teams and faster decision-making. With regard to allowing robots to leave their existing teams and form new ones, my work is similar to the aforementioned work.

However, in my framework the team management strategies I employ are for acquiring skills to complete specific tasks and not for satisfying physical constraints from the spatial context. Also, my framework operates in a much richer domain in terms of potential tasks, and a domain that is dangerous to robots.

Gunn and Anderson [2015] implemented the original framework upon which my work is based. In their approach, robots were responsible for identifying human victims and performing team maintenance duties as the need arose. Robot loss was a significant risk due to challenging conditions (debris and communication failures) in the environment. To mitigate the effect of these losses, Gunn and Anderson [2015] introduced replacement robots and used *role switches* – agents could change their roles to cover gaps in skills on a team, even if they were not perfectly suited to these new roles. They also relied on encounters between teams (single agents are considered to be a team of one) where members could be exchanged (including a single agent merging with a larger team) so that post-encounter, teams had a more effective set of skills. This approach was later referred to as *passive recruitment* in Nagy [2016] to distinguish it from other types of agent recruitment. Robot recruitment in Gunn and Anderson's [2015] approach was entirely passive: only robots or teams encountered by chance could influence role changes, and these changes would be permanent until another role change was deemed necessary. The framework developed by Gunn and Anderson [2015] provides some core components which I use in my work and these are described in more detail in Section 2.4.1.

My work extends the original framework by considering when there is global shortage of a particular skill within the environment and affords robots the option of rejecting team membership requests to make themselves available to accept task requests from all teams within the domain.

2.2 Robot Assessment and Evaluation

Gage et al. [2004] introduced an approach that uses an affective model for team recruitment. In their approach, robots broadcast requests for help and other robots receiving these requests either respond (or not) in part on the basis of a *shame value*. This *shame value* is calculated based on a robot's suitability to perform the requested task. Robots that do not respond to requests have their shame value increase over time while those responding to requests have their shame value reset to zero. The shame value thus motivates robots that ignore requests to be more likely to respond positively to subsequent requests. It also makes the best-suited robots respond first, and the less-suited ones respond later (as shame increases). My work does not use shame as a motivator, but instead takes a different approach by tracking an agent's satisfaction at the current use of its particular skill set. It also takes a broader view than Gage et al. [2004] by considering this not just as a response to individual tasks but for broader choices on team membership or independence – staying on a team or leaving it.

Simonin and Ferber [2000] proposed a satisfaction and altruism model that incorporates cooperative behaviours between multiple agents. The focus of their research was the resolution of spatial conflicts that arose from having multiple agents in the environment. To resolve conflicts, Simonin and Ferber [2000] used a satisfaction signal to determine an agent that was involved in a conflict course of action. In their model, an agent took actions to try to maximise its satisfaction based on either the agent's self-interest or the collective interests of agents. To help them achieve this, they divided agent satisfaction into three components: personal satisfaction, empathy satisfaction (average satisfaction of agents nearby), and interactive satisfaction (satisfaction obtained from interacting with other agents). An agent only broadcasts its interactive satisfaction within a bounded distance from itself, so other agents would know whether the broadcasting agent needed help or if it were being obstructive to the broadcasting agent. Each agent in their work has its own tasks, but agents could also cooperate to complete tasks. My work also models satisfaction, but as a balance between skills being used while on a team versus the opportunities that exist elsewhere, as opposed to whether one is hindering another agent. My work is also non-competitive in that there is a common goal in USAR, with agents having different immediate tasks but a global goal that all agents contribute toward. Moreover, encountering other agents in my framework is an important part of building and maintaining teams and a natural occurrence in the world, as opposed to something to be strictly avoided.

Guzzi et al. [2018] presented a framework that models human emotion in multi-robot systems. They used a navigation task to show how coordination in multi-robot systems could improve by stacking artificial emotions on top of an existing navigation framework. An agent in their work has an *internal state* which is characterised by: 1) a set of *abilities*, 2) a *personality*, and 3) an *energy level* which is time dependent. An agent's current internal state determines its active *emotional state* which in turn determines what action a robot would take, referred to as its *behaviour*. In order to avoid oscillations in the emotional state of a robot, an agent keeps its active emotion until the activation for that emotion falls below a *low* threshold or another emotion's level rises above a *high* threshold; the thresholds are dynamically determined.

2.3 Recruitment in Multi-Robot Systems

De Rango et al. [2018] implemented a protocol in wireless sensor networks to coordinate a swarm-based robot team for discovering and disarming mines. In their work, the swarm continually explores an unknown environment while actively looking for mines. In their work, disarming a mine is too complex a task for a single robot to execute, and so there is the need to request help from others. There are no pre-set teams in their experiments and teams are only formed temporarily upon the discovery of a mine, with the constraint being how many robots from the swarm are needed to disarm the mine. Once a robot discovers a mine, it begins recruiting other robots by leaving a trail to the found mine. The actual disarmament of the found mine begins when there are enough team members to carry out the task. Krieger et al. [2000] performs similarly, but allows robots to recruit others to follow them to known locations where food is available. In both of these works, teams are formed temporarily based on certain events – the discovery of a mine and the discovery of a known food location, respectively. In forming temporary teams, each robot acts as a leader when an event occurs. There are significant differences between these works and my own, however. First, these operate in homogeneous domains and do not have to consider particular abilities or skills. Secondly, robots can only accept recruitment requests in the above cited works, whereas my framework provides mechanisms that allow for a robot to reject a recruitment request.

Nagy and Anderson [2016] extended the original framework developed in [Gunn and Anderson, 2015] and added two types of recruitment mechanisms: *role-based* recruitment allowed a member to join a new team to fill an ongoing role, while *task-based* recruitment let a new member join only for the purposes of performing a single task. The latter allows an agent to essentially be borrowed temporarily from another team. Nagy and Anderson [2016] also introduced two levels of activity for performing recruitment: concurrent recruitment and active recruitment. In *concurrent recruitment*, a robot would continue performing its tasks while it broadcast messages at an interval requesting help with a task it could not perform because its task queue was full or it did not meet the task's minimum suitability requirement. During *active recruitment*, a robot that had been assigned a task it was not capable of performing would physically search for a robot that could perform the task. This is a much higher level of commitment that pauses active work on other tasks in favour of recruitment. In Nagy and Anderson [2016], there is no consideration of the rarity of any skill in the environment or the locality of any task, and there is no reason for an agent to reject recruitment requests other than being too busy. My work extends the existing framework by considering whether any of a robot's particular skills is currently rare. If any are, it stops accepting recruitment requests, with the sole purpose of satisfying tasks related to the use of its rare skill(s). Also, in addition to tracking if a robot's skill is rare, robots in my framework also track locations that would likely have tasks requiring the use of its rare skill(s). A robot will make a decision as to whether to stay and fulfil requests with respect to that particular location or continue with its team. Robots thus have a say in whether they are recruited or not. The framework developed by Nagy and Anderson [2016] forms a core part of my work and is described more extensively in Section 2.4.2.

2.4 Existing Framework

My work rests upon the same core framework used in Gunn [2011] and Nagy [2016], and so several mechanisms and concepts from this need to be described here. *Tasks* are units of useful work that can be performed by a single robot, and the act of deciding to allocate a task to an agent, or consider one's own ability to take a task on, involves a *suitability expression* matching a robot's view of its own abilities to that task. Each robot has a queue of pending tasks it has committed to carrying out. *Roles* are positions in a team a robot can fill and are defined by the types of tasks a robot filling that position needs to be able to perform. In assigning tasks, agents are searched for by roles first. If no one filling that role is available to take on the task, then a much broader search can be done using known abilities. A robot can change its role within a team if there exists a role on the team for which it is better suited and which it currently does not occupy; this is referred to as *role switching* [Gunn and Anderson, 2013]. Recruitment strategies refer to the type and amount of effort a robot will put into finding a well-suited robot for a discovered task. My work will use three existing recruitment strategies – passive recruitment from [Gunn and Anderson, 2013] and concurrent and active recruitment from [Nagy, 2016], as well as two recruitment levels: task- and role-based recruitment from [Nagy and Anderson, 2016].

2.4.1 Gunn Framework

This section overviews the core parts of the original Gunn framework [Gunn and Anderson, 2015] that are central to my work.

2.4.1.1 Attributes, Tasks, and Roles

All robots within the framework are described using a set of characteristics and capabilities referred to as *attributes*, and these define the abilities a robot should have to be able to carry out a specific task. Using attributes, a robot is not only able to determine to what degree a task lies within its own capabilities, but equally provide an estimation of the capabilities of other robots and their match to the task. An example of an attribute is a robot having a debris remover.

Tasks are units of useful work that can be performed by a single robot. Although all robots contribute towards a larger goal which comprises completing different tasks, all the individual tasks in my framework require a single robot to complete and do not require the pulling together of resources from multiple robots [Krieger et al., 2000; Kiener and von Stryk, 2007]. This means that complex tasks are broken into simpler pieces that are ultimately carried out by different robots.

To enable robots to keep track of assigned tasks, every robot maintains a *task queue* – a priority queue of tasks a robot has either discovered or has been assigned, ordered by importance.

Every task within the framework uses a *minimum requirements expression* and a *suitability expression* to deal with the potential limitations of robot skills. A minimum requirements expression specifies the minimum set of attributes a robot should possess to successfully carry out that specific task. For example, a robot discovering an *extinguish-fire* task but not being in possession of a fire extinguisher cannot put out the fire, therefore it will leave that task to someone more capable. After a robot meets the minimum requirement for a task, the next step is to estimate how successful the robot would likely be if it attempted to perform the task. This is where suitability expressions are useful. A suitability expression assigns weights to the various attributes specified by a task's minimum requirements and evaluates to a numeric value. This provides an effective way for deciding the most suitable robot for a particular task from a group of robots. The greater a robot's suitability value, the more well-suited that robot is for the task. Intuitively, robots not meeting the minimum requirements for a task have a suitability of zero. Even though we can determine how capable a robot would be if it attempts to perform a task by using suitability expressions, the framework provides *roles* as a shortcut to simplify the task allocation process (this is described further in Section 2.4.1.3).

Roles are positions within a team a robot can fill and are described by a collection of tasks that a robot occupying that role is normally expected to perform. When a robot occupies a role, it is assumed that robot can perform all the tasks that role encompasses. However due to the changing nature of the environment, as well as the potential for damage, a robot could occupy a role for which it is not currently adequately equipped. This could be caused by a team having skill deficits or communication problems, or a robot that was properly equipped when the task was assigned could since have become damaged. Even though roles serve as a shortcut in allocating tasks, it is also possible for a robot occupying a certain role to be assigned tasks not associated with the current role it occupies. This can happen when there is literally no one else, for example. Every task within a role has a weight associated with it that describes the importance of that task within the context of the role being occupied. For a robot to be considered suitable for a role, it must meet the minimum suitability requirements of all tasks contained in that role. To keep robots active even when they run out of work, all roles have an associated default *idle task* to which a robot reverts when its task queue is empty.

2.4.1.2 Team Management and Maintenance

To give a robot a sense of its current team membership, every robot regularly broadcasts its location to the other members of its team, so they are aware of it. This helps to keep knowledge of team structure among robots more up to date than would otherwise be possible. If a robot is not heard from after a certain period, it is considered to be no longer part of the team (i.e. at least temporarily lost). For a domain where wireless communication is unreliable, there is a likelihood some broadcast messages would not be received by all team members. This can potentially let robots incorrectly assume members have been lost or are damaged. Rebalancing roles within a team because of such a scenario can lead to robots occupying suboptimal roles or to the redundant assignment of roles within the team unknowingly. However, as better information becomes available such a situation will be rectified.

An ever-changing domain such as USAR poses a tremendous amount of risk to robots operating within the environment. Teams could lose members not only because of communication challenges, but due to the fact that the terrain itself poses a threat to robots – resulting in damage or an inability to move (becoming stuck). In such scenarios it becomes necessary to send replacement robots into the environment occasionally so existing teams can fill roles which were rendered vacant.

For a team to know what roles are lacking, it is important to define what makes up an *ideal team* [Nagy, 2016]. The purpose of an ideal team definition is to serve as a yardstick for teams to help determine how best to restructure or integrate new potential members into the existing team's structure. An ideal team usually has an overall balance of skills to enable it to function well in a given environment. In this framework, an ideal team is defined at the beginning of a mission and is specified by a human; it would be interesting to have robots refine their own ideal teams based on what tasks are most likely encountered and what losses are likely to occur, but that is beyond the scope of my research. Due to the dangers associated with the USAR domain, a team would typically not perfectly match the ideal team requirements except at the onset of a mission. Formally, an ideal team is represented as a list of desired roles, the minimum and maximum number of robots to occupy these roles and the relative importance of the roles within the context of the overall team. A sample ideal team definition is shown in Figure 2.1, depicting a single leader role, one or two robots that can verify victims, and several small explorer robots.

Teams are able to know what roles are lacking by performing occasional *role switch checks* [Gunn and Anderson, 2015], where team members evaluate their current role against their knowledge of the current team's structure. When determining if a robot

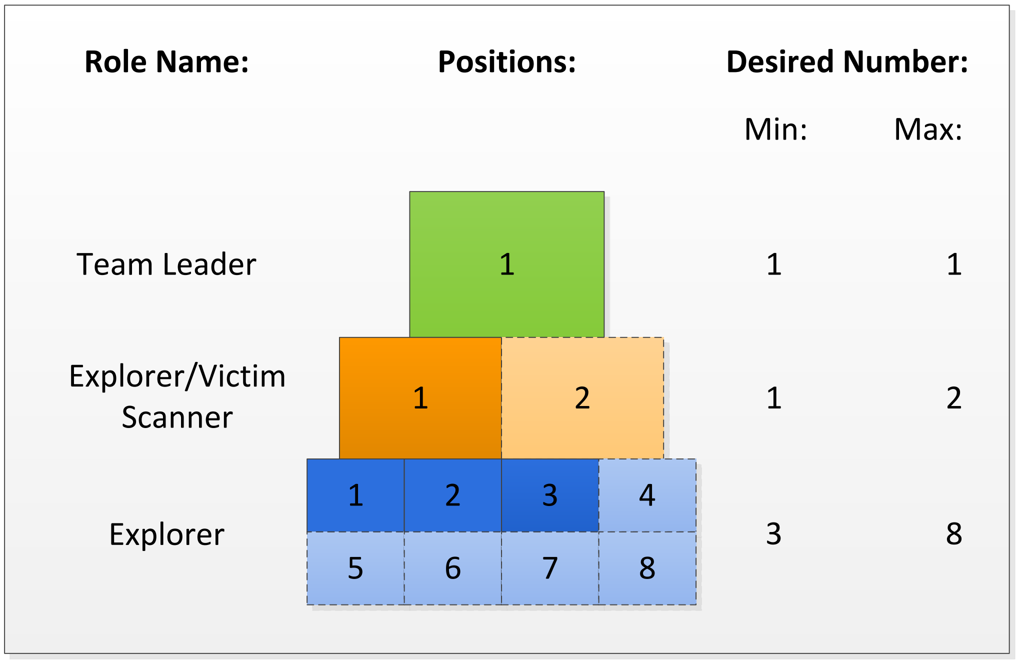

Figure 2.1: A sample ideal team definition with three robot types. Adapted from [Gunn, 2011]

should switch to any role, the robot creates a list of underfilled roles, ranking them by their *role score value*, which is the sum of the robot's suitability for that role and the role's importance (recall that every role has an importance value describing the significance of the role to the ideal team definition). If there is a role with a higher role score than that of the role the robot currently occupies, it switches to that role and notifies its team members, otherwise it sticks to its current role. *Role switching* allows for team rebalancing to ensure that team members occupy roles to which they are most suited. However, it does not help in scenarios where a team has lost a significant number of members and has to recoup its losses. To reacquire skills lost by a team, the original framework relies upon random encounters with other teams (including stray single robots lost from other teams or released individually after the start of the operation) [Gunn, 2011].

The original framework [Gunn, 2011] provides a mechanism whereby two teams use the opportunity upon encountering one another to rebalance team membership and recoup lost skills. This is referred to as *passive recruitment* in the subsequent framework. A team encounter starts with two robots observing one another and exchanging information regarding their current individual teams' structures. The more computationally-capable of the two robots is selected to perform a team merge and redistribution process provided it is well-suited to carry out the process. If suitable, the more computationally-capable robot then determines how robots on the two teams should be distributed such that each team is as close as possible to the ideal team definition. There are two possible outcomes from this distribution process: 1) two teams both being closer to meeting the ideal team definition or 2) one team formed by integrating members of both teams (this covers the case of encountering single-robot teams). Once a suitable distribution has been determined, the more computationally-capable robot communicates this to the other robot. Both the robots are then responsible for instructing their teammates to change roles or teams as appropriate.

2.4.1.3 Task Management

All robots in the framework are responsible for identifying and completing tasks to the extent their attributes will allow (Section 2.4.1.1). Tasks (discovered or assigned) are executed in order of task priority to ensure that the most important tasks are performed first.

A robot's task queue contains a list of tasks that are yet to be completed by that robot, ordered according to task priority and time when the task was added to the queue. Therefore, tasks which have the highest priority and have been in the task queue the longest are selected for execution first. When a robot completes a task, it removes that task from the queue and selects the current oldest, highest-priority task for execution. A robot inserts tasks into its task queue by either discovering a task on its own, or being assigned a task by its team leader. Using a priority queue to store pending tasks ensures that the most important tasks are completed first. However, it is possible that while executing a task, a robot might become aware of another task having a higher priority. The Gunn framework addresses such an occurrence by using *task pre-emption*, where a newly inserted task takes precedence over a currently-executing one if it has a higher priority. Robots query their task queues occasionally, to make sure no new higher-priority tasks have been inserted while in the middle of completing a task. In the event that a more important task is discovered, the robot will immediately stop work on the currently-executing task and begin execution of the newly-discovered higher-priority task.

When a robot comes across a new task, it first evaluates its suitability for that task. If it meets the minimum requirements for that task, the robot will insert the task into its queue as long as its task queue is not full. If the robot does not insert the newly-discovered task either because it is not suitable or it has a full task queue, the robot will attempt to inform the leader of the discovered task. In the Gunn framework, only team leaders are able to assign tasks, hence it is necessary to inform the team leader so the task can be assigned to a suitable robot (Nagy [2016] extends this by allowing non-leaders to assign tasks through *task-level recruitment* – refer to Section 2.4.2.2).

Robots that cannot add discovered tasks to their queue (because they are not suitable enough or do not have any room for additional tasks) transmit these tasks to the leader for appropriate delegation. Upon successful receipt of the message, the team leader must assign these tasks to one of its team members. There are two ways by which a leader attempts to assign tasks to members [Gunn, 2011]. The team leader first tries using role heuristics to assign tasks in order to minimise the level of effort and communication required (*role-based task assignment*). If the process fails, a more extensive search (*exhaustive task assignment*) is done to find a suitable robot. A team leader's task allocation process is illustrated in Figure 2.2

Role-based task assignments incorporate the use of roles to reduce how much effort a team leader puts into assigning tasks. Leaders use their knowledge of roles associated with a task in conjunction with the robots currently filling those roles to help identify likely candidates to assign the task to [Gunn, 2011] (recall from Section 2.4.1.1 that roles are defined by a list of task types). Before assigning a task, a team leader creates a list of known team members occupying an appropriate role and sends them each a message simultaneously, requesting they complete the specified task. Upon receiving this request, a robot will take into consideration the task type and its current workload. Once a robot determines that the task is within its capabilities and it has room in its workload, the robot responds to the leader indicating its suitability and the *cost* (calculated in terms of a robot's distance to the task location). Otherwise, a robot sends a rejection response indicating that it

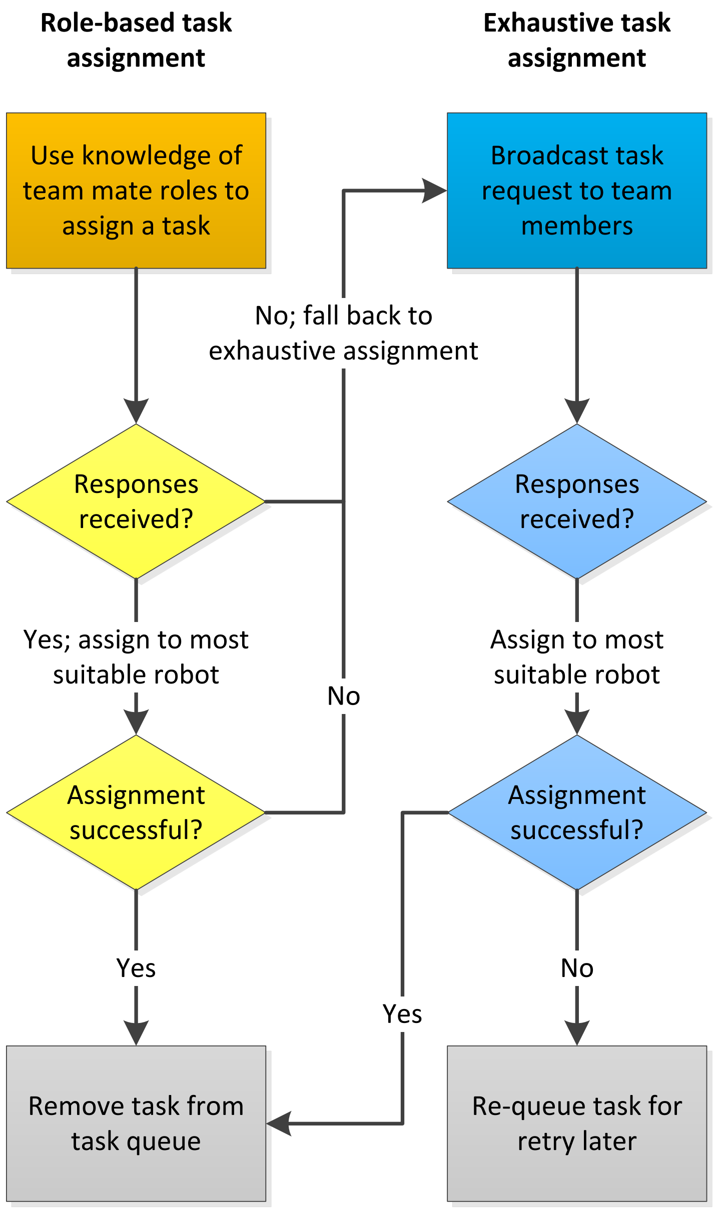
Role-based task assignment
Exhaustive task assignment
Use knowledge of team mate roles to assign a task
Broadcast task request to team members
No; fall back to exhaustive assignment
Responses received?
Responses received?
Yes; assign to most suitable robot
Assign to most suitable robot
No
Assignment successful?
Assignment successful?
Yes
No
Yes
Remove task from task queue
Re-queue task for retry later

Figure 2.2: The process a team leader goes through to find a suitable robot for a task. Adapted from [Nagy, 2016]

cannot execute the task if its task queue is full or it is not suitable enough.

The leader has a time frame within which it expects responses to the task completion request from team members, and responses received outside of this window are ignored. Responses received within the window are processed and the robot which indicated the highest suitability for the task is sent a confirmation message. In cases where multiple robots indicate the highest suitability, the robot indicating the least task cost is sent the confirmation message. The robot that receives the confirmation message adds the task to its queue and sends an acknowledgement back to the leader indicating it accepted the task. When a leader receives this acknowledgement, it considers the task successfully assigned and deletes the task from its queue. In the event that a leader does not receive any responses meeting the minimum task suitability within the time frame (this covers rejection responses, no responses at all, or a mix of both), it switches to *exhaustive task assignment*.

Exhaustive task assignment is computationally expensive and communication intensive, thus it is only used when role-based task assignment fails. A team leader using exhaustive task assignment broadcasts a wireless message that can be received by any team member. Just as in the role-based task assignment, team members respond with their suitabilities and cost, and the leader assigns the task to the responding robot with the highest suitability. If, by using this strategy, a leader does not find a suitable robot to assign the task, the leader reinserts the task into its queue for reassignment at a later time.

2.4.1.4 Schemas

The use of *schemas* is a very popular approach within the autonomous mobile robotics field [Arkin, 1987], and the Gunn framework [Gunn, 2011] used a schema-based approach to interpret raw data gathered from the environment by robots to determine what set of actions robots had to take. The Gunn framework uses *perceptual schemas* to perceive the environment and passes the data collected to the *motor schemas* which determine what sequences of action a robot will take. Using motor schemas, robots are able to exhibit complex behaviours by combining different motion vectors. In a scenario where an obstacle is in between a robot and a goal location, the robot will generate a repulsive vector away from the obstacle and an attractive vector towards the goal upon receiving data from the perceptual schema. The summation of these two vectors results in a vector that guides the robot around the obstacle and towards the goal. I implement additional schemas to the ones that already existed in the framework, and these are described in Section 4.6.

2.4.2 Active Recruitment Controller (ARC) Framework

The ARC framework [Nagy, 2016] extended the Gunn framework by introducing mechanisms by which robot teams could be aggressive about bringing in new skills to the team either permanently (*role-level*) or temporarily (*task-level*).

2.4.2.1 Recruitment Spectrum

The recruitment spectrum characterizes the different strategies by the amount of effort a robot will use when trying to recruit others. The ARC framework discretised the range of effort robots can commit to their recruitment tasks. These strategies range from *passive* – where a robot will only attempt recruitment upon chance encounters with other robots within the environment [Gunn and Anderson, 2013], to *concurrent* in which a robot will simultaneously attempt to recruit while performing another task, and finally to *active* where a robot will halt any currently-executing task solely to work on recruitment [Nagy, 2016]. The recruitment spectrum depicting the varied amount of effort robots use when performing recruitment is shown in Figure 2.3.

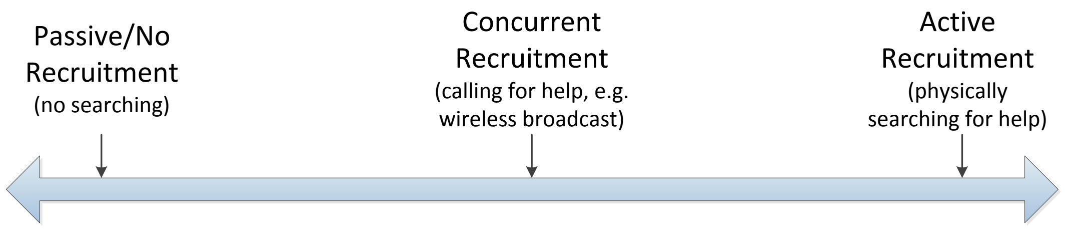

Figure 2.3: The recruitment spectrum. Adapted from [Nagy, 2016]

Passive recruitment has already been described in Section 2.4.1.2.

For concurrent recruitment, the aim is to increase the likelihood of a robot coming across other useful robots within the domain while still maintaining its focus on doing useful work. Robots employing this strategy continue normal execution of their tasks, with the only difference being that they send out occasional broadcasts to find and recruit other robots if they are available.

In order to maximise the likelihood of encountering useful robots in the environment, a robot places more emphasis on searching rather than completing any pending tasks it might have. A robot in this state uses the active recruitment strategy. Unlike the previous two states on the spectrum, active recruitment is a task itself and thus can be assigned from one robot to another. Robots in the framework can only execute a single task at a time, hence a robot performing an active recruitment cannot execute any additional task. The aggressive nature of this strategy significantly increases the chances of encountering a useful robot (but limits other useful work while doing so). Once an active recruitment task is created, it is inserted into the robot's task queue in order of the task's priority. The priority of a recruitment task is determined by the type of recruitment (role-level or task-level). Role-level recruitment tasks have the highest priority, thus when such a task is inserted into a robot's task queue, the robot halts any currently executing task to perform the just inserted active recruitment task. For task-level recruitment tasks, the priority depends on that of the task being recruited for.

2.4.2.2 Recruitment Strategies

There are different ranges of effort a robot will employ when attempting to recruit, and there are two levels of recruitment which indicate whether the skills being acquired via recruitment will be temporary or permanent. Task-level recruitment is a shallow kind of recruitment and involves searching for a robot with some desired skills and requesting that it execute some specific task. The other, which is role-level, is a more exhaustive search in which a team leader requests that a robot locate another robot to occupy a role in the team that the leader has determined to be under filled. A robot using any of these two levels of recruitment can apply any range of effort defined on the recruitment spectrum (Section 2.4.2.1) when fulfilling a recruitment request. An outline showing these strategies and how they are used is shown in Table 2.1

Table 2.1: Recruitment strategies employed in fulfilling role or task requirements. Adapted from Nagy [2016]

	Passive	**Concurrent**	**Active**
Task-level	wait for robot to join team by chance	while performing regular work, look for a robot to complete a task	halt regular work and search for a robot to complete a task
Role-level	wait for robot to join team by chance	while performing regular work, look for a robot to fill a role	halt regular work and search for a robot to fill a role

Task-level recruitment allows a robot to enlist the help of other robots to complete a task for which it is not well suited. A robot's search for suitable robots is not restricted to communication within its current team, implying that a recruiting robot can assign a task to a robot outside of its current team. This process does not require sanctioning from the recruiting robot's team leader and therefore all robot types can perform a task-level recruitment. This process is similar to the regular task assignment process discussed in Section 2.4.1.3.

Role-level recruitment enables a skill (set of skills) that are deemed essential for the successful operation of a team to be acquired on a permanent basis. The concept of an ideal team (Section 2.4.1.2) has a significant bearing here since it helps in identifying roles in a team, which by the definition of an ideal team should be present but have no robots satisfying those roles. As part of its team-management duties, a team leader does occasional missing role checks to determine which roles, if any, are absent based on its current knowledge of the team structure. Though team leaders might have an imperfect knowledge of the team structure, it is assumed they will have the most complete of the team and are thus the obvious choice for performing missing role checks. Once a missing role check is completed, a leader selects the most important underfilled role from the missing role list and assigns a team member with a role-level recruitment task. That team member becomes responsible for finding another robot to fill that missing role on the team. The process is illustrated in Figure 2.4

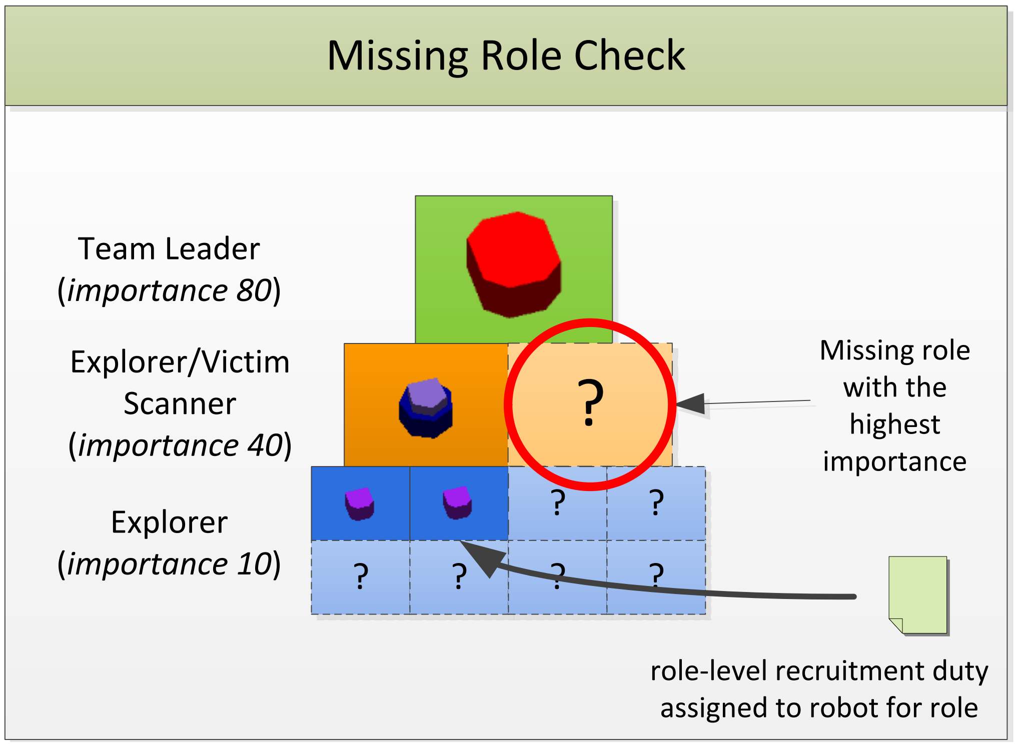
Missing Role Check
Team Leader (*importance 80*)
Explorer/Victim Scanner (*importance 40*)
Explorer (*importance 10*)
?
Missing role with the highest importance
role-level recruitment duty assigned to robot for role

Figure 2.4: A missing role check and a role-recruited task generated as a result of the role check. Adapted from Nagy [2016]

2.4.2.3 Marker Manager

The ARC framework [Nagy, 2016] introduced the use of *markers* to help robots know which victims had already been seen, to prevent redundant victim confirmation tasks being generated for victims that other robots had already come across. When a robot comes across a potential victim, it drops a marker by that victim so other robots know that victim is in the process of being or has already been confirmed (the framework has no way of differentiating actual real victims from false victims after they have been confirmed, and future work could look into ways by which robots can know when a victim has been confirmed). The *marker manager* is responsible for handling this process of robots dropping markers near victims and also responsible for robots without any markers requesting for some from others.

2.5 Conclusion

In this chapter, I have discussed works related to mine and also described modules from the existing framework which are central to my implementation. Having introduced all the major components that I have reused from the existing framework and are pivotal to my work, I will now describe the concepts I have added in Chapter 3.

Chapter 3

Methodology

3.1 Chapter Overview

This section discusses my design concepts and the approach I took to answering my research questions.

In order to understand the work I have performed, an introduction to the core concepts of the initial framework and developments to it that precede my own are provided in Section 2.4.

3.2 My Framework

There are a number of factors that could be considered when trying to improve the performance of teams in dangerous domains, and my work considers how three such factors – satisfaction, skill rarity, and task locale – can be leveraged to improve the performance of the existing framework. By focusing on these factors, I can determine when it is advantageous for an agent to be willing to switch from its current team to another team or become completely independent of any team and be responsible for its own tasks. An overview of my framework is shown in Figure 3.1

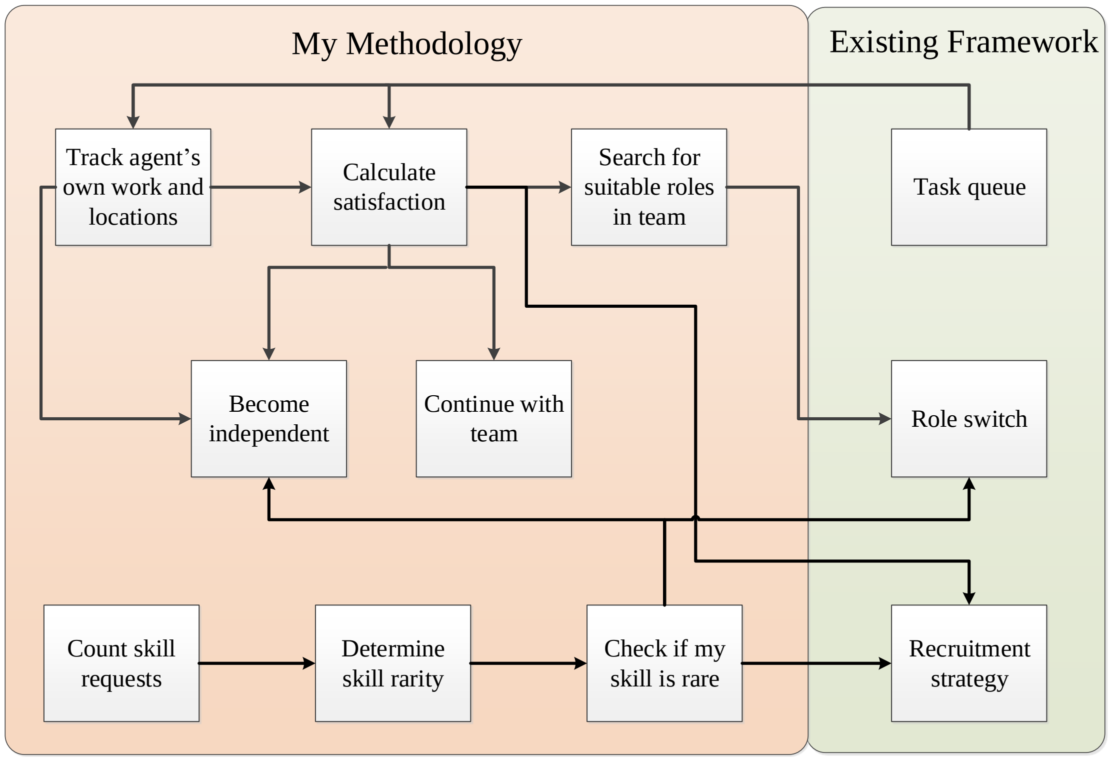

Figure 3.1: An overview of my framework

The goal of my thesis is to develop and investigate how a robot can provide an estimation of its utility with respect to the team's overall goals and come up with strategies that can be used to improve a robot's productivity within its current team. To enable me do this, I have introduced a means by which a robot can provide an estimation of how useful it is within a team. I refer to this in my work as *robot satisfaction* and I provide more details in the following section.

3.2.1 Robot Satisfaction

For a robot to judge whether it would be of greater use on a different team or working independently, or is more productive in its current situation, it must have a means of judging whether its skills would be more useful under each scenario [Arkin, 1995]. Similarly, a team leader must also consider the utility of an individual when deciding if further work should be devoted to coordinating a given robot as part of a team. This section introduces the concept of *robot satisfaction*, a metric for evaluating and comparing this utility.

In any team setting, team members are required to make some meaningful contribution towards achieving the team's goals. A limitation of the Gunn and ARC frameworks is that they do not provide any effective mechanism by which a robot can measure its usefulness to a team by way of its contribution towards the team's collective goal. The framework as it existed before ensures that a robot is always executing some task by using *idle tasks* – default tasks robots fall back on when they have no discovered or assigned tasks available. The issue with this is that idle tasks as defined in the framework hardly make use of a robot's unique skills. It is also assumed in the previous framework that all team members would be useful in their respective teams. However, this might not be the case, as a team member could be stuck performing its idle task because the area currently being explored by its team has no tasks requiring the use of its special skills.

My work approaches satisfaction from two perspectives, 1) the individual's own satisfaction, and 2) the team leader's satisfaction with any team member. I use a *satisfaction expression* to represent how satisfied an agent is with its contribution towards its team's goals, and how satisfied a team leader is with the members on its team.

3.2.1.1 Tracking individual satisfaction

Calculating an agent's satisfaction involves weighting tasks completed over time (where heavier weight is given to those using rare skills). It also involves looking at the task queue to consider future satisfaction that will be provided by known upcoming tasks. Finally, a robot also considers the number of requests from others to use its skills that had to be rejected in order to continue with its current team. A robot would reject an incoming request from outsiders in favour of staying on its current team if the incoming task request does not require the robot to use its rare skills and its current satisfaction value is above the threshold. Each agent keeps track of its satisfaction and depending on its current satisfaction value, it will take the appropriate course of action. To enable me to prescribe how much effort a robot should put into increasing its satisfaction value, I have introduced the concept of *willingness*, which helps a robot determine how aggressive it should be with regard to improving its current satisfaction value. I have discretised the amount of effort a robot will expend trying to improve its satisfaction value into three *willingness states* –*not willing, somewhat willing, and willing.*

A robot would also set its *willingness* to this state when one of the following scenarios occur: 1) one of the robot's skills has become a rare one in the environment (Section 3.2.2), 2) the robot's current satisfaction value is above *base satisfaction* (a range of values I determined through initial experimentation. Details of this

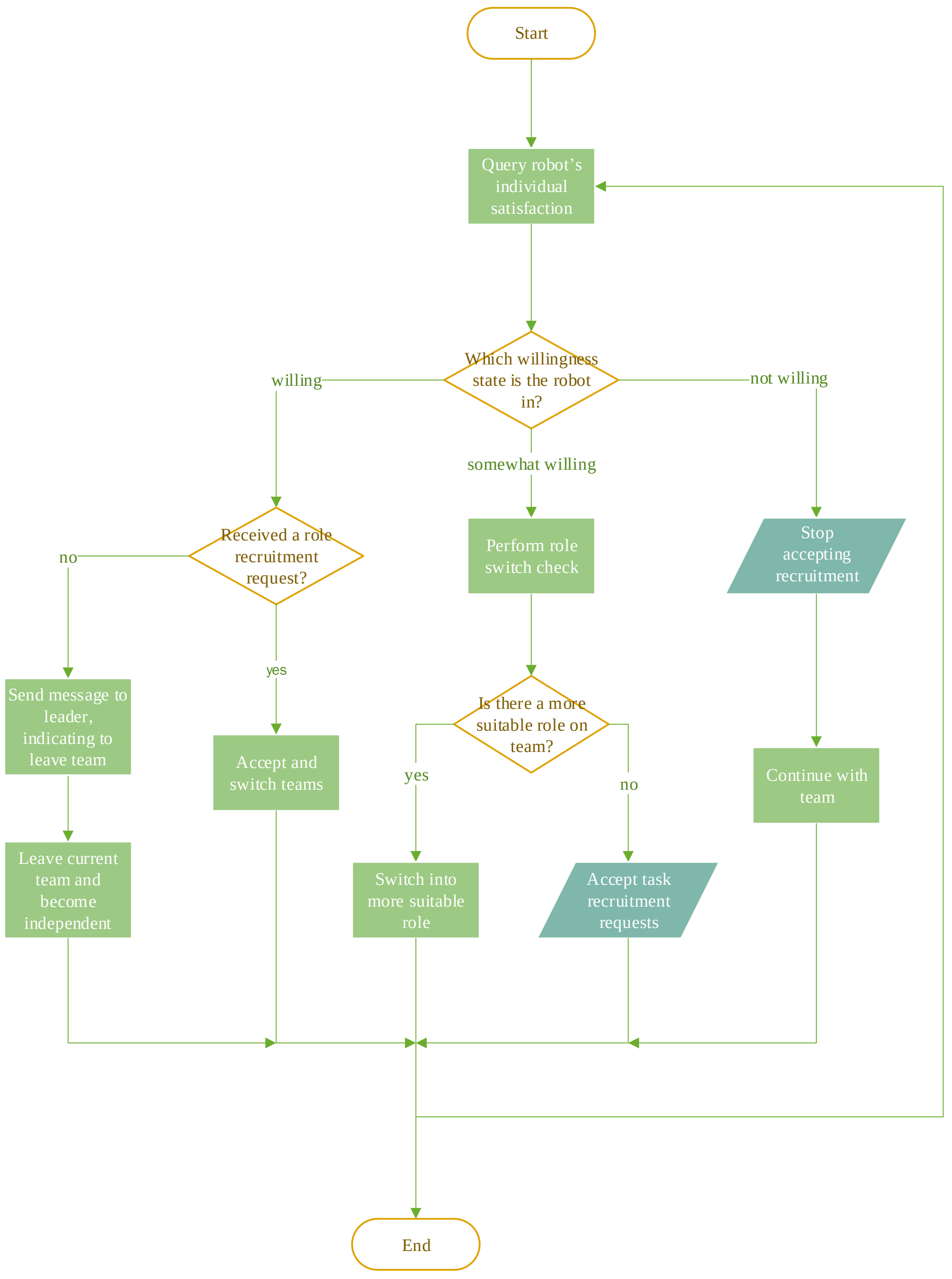
Start
Query robot's individual satisfaction
Which willingness state is the robot in?
willing
not willing
somewhat willing
Received a role recruitment request?
Perform role switch check
Stop accepting recruitment
no
yes
Send message to leader, indicating to leave team
Accept and switch teams
Is there a more suitable role on team?
yes
no
Continue with team
Leave current team and become independent
Switch into more suitable role
Accept task recruitment requests
End

Figure 3.2: The decision-making process of a robot involving its current satisfaction

experiment are presented in Section 4.9). When a robot has its state set to *not willing* because of the former, it makes itself available for recruitment by all teams within the environment involving its rare skill(s), by accepting task-based requests involving the use of those skill(s). On the other hand, a robot in the *not willing* state, because it has a high satisfaction value, will continue with its team because it is actively contributing towards its current team's goals and does not need to actively improve its satisfaction.

When a robot is in the *somewhat willing* state, its current satisfaction value hovers around the base satisfaction. In this state, a robot expends a moderate amount of effort into improving its current satisfaction by either accepting task-level recruitments or performing occasional role switch checks (Section 2.4.1.2) to find a better role if one exists.

For a robot to be in the *willing* state, it means its current satisfaction value falls below the base satisfaction. This most likely means the robot had not been using its unique skill(s) for most of the period within which its satisfaction was calculated. In this state, a robot will do more outside its current team by taking a more aggressive approach to increase its satisfaction – this includes accepting role-based recruitment [Nagy and Anderson, 2016] from other teams, or permanently moving to other teams of its own initiative, or leaving all teams to work independently. If an agent in the *willing* state receives a role-level recruitment request, it will accept the request with the aim of improving its satisfaction value – i.e., lack of current satisfaction will allow it to be recruited to another team permanently or go off on its own (become a team of one). For the latter, a robot sends a message to the leader that it is relinquishing its team membership. The team leader, upon receiving this message, uses its own evaluation of the robot (described in Section 3.2.1.2) to determine whether it should let go of that robot. If the leader determines the robot is not useful enough in terms of its contribution, it terminates the robot's membership and updates its knowledge of the team's current membership accordingly. Then, it sends a message back indicating that the robot is no longer part of the team. Upon receiving this message, the robot then wanders off in hopes of being recruited by another team. If the leader deems the member useful, it sends a message to the member telling it to set its willingness flag to *somewhat willing*. This is done to try to prevent an industrious member (from the leader's perspective) from leaving because it deems itself not useful enough. In the event that, due to communication challenges, a leader does not receive a member's message requesting to leave the team, the member still wanders off but only after 15 seconds have elapsed since it sent the message. The team leader, not hearing from this robot after some time, will consider the robot lost and update the team membership appropriately.

A robot updates its satisfaction value every 30 seconds within my implementation but consideration of current satisfaction in an agent is triggered by one of two events: being idle for a period of time, or receiving a recruitment request from a team other than its own. The latter will cover the situation where an agent is fully occupied but is not using skills that are currently rare. If an agent receives a recruitment request and its satisfaction value is below the threshold, it will accept the request with the aim of improving its satisfaction value – i.e., lack of current satisfaction will allow it to be recruited to another team permanently, or go off on its own. If the agent's current satisfaction hovers around the base satisfaction, it will perform a role switch check to find out if there is a role it is better suited for available on the team. If one exists, it switches into that role otherwise it becomes more receptive of task-based recruitments from outside its team. In the case where the agent's satisfaction value is above the threshold, it will continue performing its tasks with its current team since it will be satisfied with the work it is doing. This process is illustrated in Figure 3.2. From the opposite perspective, a continually idle agent might also risk being dropped from its team because the work involved in coordinating it is greater than would be achieved by temporarily recruiting an agent for a specific task when needed (Section 3.2.1.2).

3.2.1.2 Tracking team members' satisfaction

In addition to team members tracking their individual satisfaction, team leaders also monitor to some extent the amount of work their team members have successfully completed within a specified time period. In a team setting, a team member being satisfied with its output on a team does not necessarily translate into a team leader being content with the amount of work the team member is performing. If a team member repeatedly rejects task assignments from its leader, the leader would deem that robot not useful enough and would eventually let go of that robot since the cost of coordinating that team member outweighs the benefits that robot brings to the team (from the leader's perspective). Once a role becomes vacant, it allows for the team to recruit a new member by performing missing role checks (discussed in Section 2.4.2.2) with the hope of recruiting a more industrious or capable replacement. A leader tracks

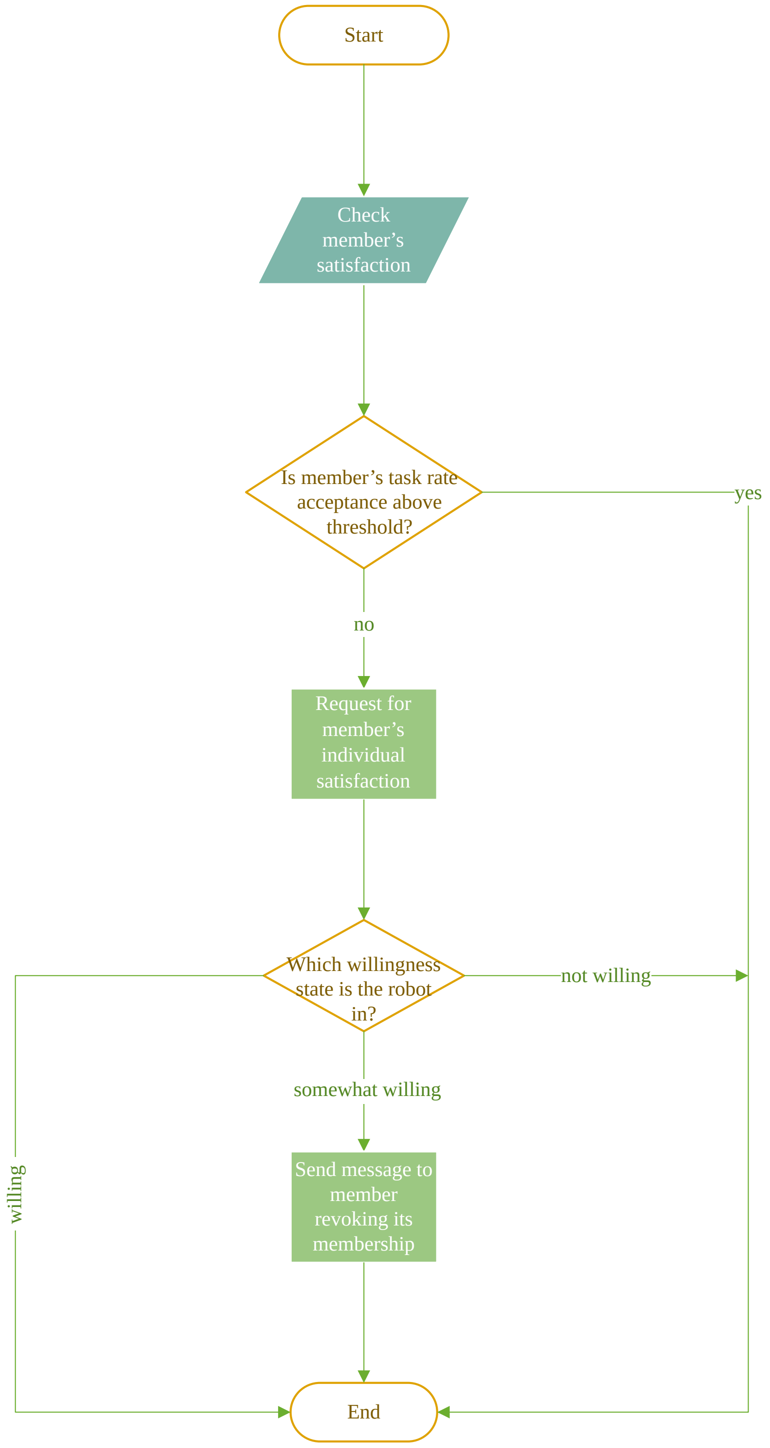
Start
Check member's satisfaction
Is member's task rate acceptance above threshold?
yes
no
Request for member's individual satisfaction
Which willingness state is the robot in?
not willing
somewhat willing
willing
Send message to member revoking its membership
End

Figure 3.3: A leader determining whether a member is useful or not

its team members' usefulness to the team by counting the number of completed tasks reported by its team members. By doing this, a leader is also able to recognise when robots which have become separated and are no longer within communication range of the team. If the leader's evaluation of any member falls below a particular value, the leader sends a message to that robot requesting its current willingness state. After receiving a response from the robot, the leader makes a final decision. If a robot responds with a message that it is in the *somewhat willing* state, the leader sends another message telling the robot to set its willingness state to *willing*. This would allow the robot to eventually leave the team, making room for the team to recruit a more capable robot to fill the soon-to-be vacant role. In scenarios where the robot responds with a *willing* or *not willing* state, the leader takes no further action. A flowchart illustrating this process is shown in Figure 3.3.

3.2.2 Rare Skills in the Environment

During a rescue mission, some skills may become rare as a result of agents getting damaged or lost, with fewer replacements with that skill available. Whether a skill is common or rare can affect the value of an agent to a team significantly.

Since any robot's picture of the environment and the other robots in it is imperfect, any measure of the availability of those with a particular skill will also be inaccurate. I estimate this by tracking the number of recruitment requests an agent receives that require a particular skill. In addition to the number of individual requests received, the number of teams requesting this skill is also tracked (one team might need an agent's skill just because of its current situation, but multiple teams requesting would indicate a more global shortage). Once an agent recognizes that it has a currently rare skill, it will be more likely to respond positively to recruitment mechanisms involving that skill - either temporarily leaving a team often enough that it may lose touch, cease being a member and work independently, or leave a team explicitly to join another team (role-based recruitment [Nagy and Anderson, 2016]). This will be coupled with the satisfaction mechanism described above: being dissatisfied with the use of one's skills and at the same time knowing those skills are rare should increase the likelihood of leaving a team. In my work, I focus on specialised skills (ones which only one type of robot has) within the environment. Although common skills could also become rare in the event that a majority of robots having that common skill get damaged, leaving just a few robots to fulfil tasks requiring that skill, my work does not investigate the latter phenomenon. Within my framework, a robot determines it has a rare skill if more than one team is requesting it to perform a task involving that skill and the tasks are not located within the same *task locale* (task locales are described in Section 3.2.3.1). If a robot realises it has a rare skill, it sets its willingness flag to *not willing* and will reject any task requests that do not involve the use of its currently rare skill including requests coming from its own team. This allows for a robot to make itself available to all teams for the length of period during which it deems it has a rare skill. Once the skill is no longer rare, it attempts to find and rejoin its team. If it is unable to find its team after some time, it wanders off on its own with the aim of recruiting other robots along the way or being recruited by another team. This process is shown in Figure 3.4

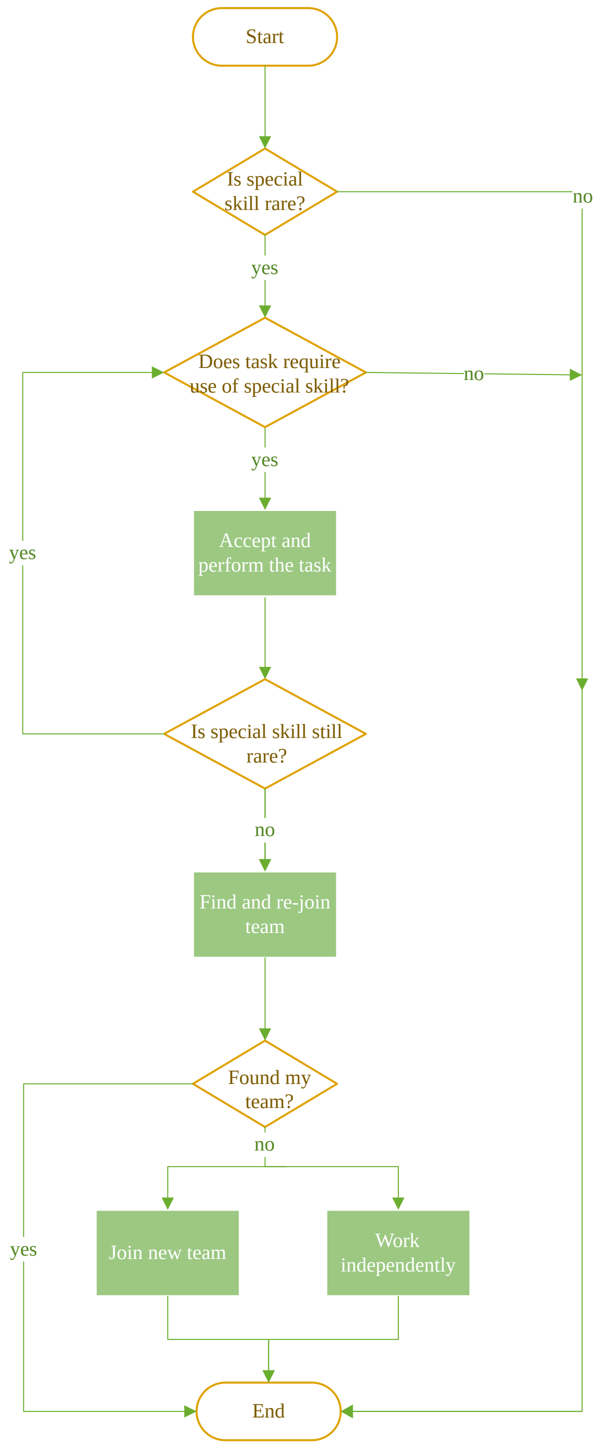

Figure 3.4: A robot checking if it has a rare skill, and taking the requisite actions

3.2.2.1 Unique skill

Some tasks in the environment require special skills in order to complete. Such tasks can only be performed by a specific type of robot because of their specialised nature. Any skill required to complete such a task is referred to as a unique skill in my framework. As an example, an *extinguish-fire* task requires that a robot have a fire extinguisher and only a fire bot has this particular attribute, making the skill associated with extinguishing a fire a unique one. For a robot to identify if its unique skill might be rare, it counts the number of requests it receives from different robots that want to utilise its unique skill. There is the possibility that different robots might discover the same task requiring a unique skill and this does not count as two separate requests. To be able to tell when different task requests might be referring to the same task, I use a *task locale* (described further in Section 3.2.3.1), which helps identify requests coming from the same region.

3.2.3 Robots' Sense of Locality

A robot needs some sense of where the work it is performing takes place (localisation), to know if moving about with a team is likely to produce greater satisfaction than performing tasks locally.

To achieve this, agents not only keep track of what tasks they perform but also *where* they perform these tasks. If an agent has specialised skills beyond general exploration, and currently has a low satisfaction level, it has the ability to accept recruitment requests involving that location in order to raise satisfaction. This allows location-based activity to use the same satisfaction mechanism employed to consider independence vs. team membership. If an agent stays long enough at a given location it will eventually be lost from its original team (when the team moves on), and its satisfaction can continue to be maintained by performing local tasks (i.e., operating independently). If the skill used for these tasks becomes less rare, or if fewer opportunities are available (e.g. a stair-climbing robot is at a staircase but teams have moved on and no one is making use of them), it will eventually lose satisfaction and be more receptive of other types of requests, or begin wandering through being idle. It will thus move to a different part of the environment that may be more fruitful, or rejoin some team, or gather new individuals and form a larger team as time goes on. Similarly, if its skill becomes less tied to a given location (e.g. a staircase collapses so there is no new stair-climbing work), it will influence the robot away from its current position and toward other team-based activities.

3.2.3.1 Task locale

In my framework, robots divide the environment as they explore into small grids ($3\text{m} \times 3\text{m}$) and keep track of their activity within each region. I refer to these regions as *task locales* in my research. Within a task locale, a robot tracks the work it successfully completed in that region and its *location satisfaction* — the cumulative sum of a robot's individual satisfaction gained from completing work in a task locale. The purpose of doing this is to provide a robot with a list of options should it need to be aggressive about increasing its satisfaction or when it is let go by its team. Task locales are sorted in order of their location satisfaction, with the region producing the highest satisfaction being first. When a robot wants to increase its satisfaction, the robot will revisit areas where it has previously done useful work to check whether new tasks have become available (because of the dynamic nature of the environment) or it left some work undone because of its commitment of staying with its team. For a robot not to get caught in a cycle of continually revisiting already explored regions, a task locale is visited only once via its own effort (this does not include instances where a robot revisits a locale accidentally through random exploration), and only the top three regions for location satisfaction are visited. This allows for robots to wander off to unexplored regions after some time.

3.3 Conclusion

I have described my methodology in this chapter alongside portions of the existing framework that I used to enable me to implement my own. The next chapter gives a detailed description of my implementation as well as that of existing components my work relies upon.

Chapter 4

Implementation

4.1 Introduction

In order to evaluate my framework, it requires an embodied implementation in a domain in which experimentation can take place. The purpose of this chapter is to describe important elements of that implementation. Since my research is an extension of existing work [Gunn, 2011; Nagy, 2016], it is necessary to briefly describe components from the previously mentioned frameworks because they serve as the starting point for my implementation.

4.2 USAR Domain

Urban Search and Rescue involves the location and extraction of human casualties trapped within a collapsed structure and as my work focuses on the USAR domain it is necessary to introduce some concepts that are central to this domain and what the mission within such an environment is. Since such environments are at risk of further collapse and damage (placing victims and rescuers in more danger), locating and rescuing casualties are time-sensitive missions.

All robots in my framework work towards achieving two main goals within a simulated USAR environment. These are 1) find as many human casualties as possible, and 2) explore as much of the environment as they can. In a real world scenario, information garnered by robots will be relayed to a human rescue team to be used as a guide to extract casualties promptly.

Although it would have been interesting to do my implementations on physical robots in the real world, the focus of this research is to develop a multi-robot framework that encompasses various concepts revolving around team management and coordination, hence the reason for using a simulated world. Also, it requires a great amount of effort and cost to build robots and create a realistic USAR setting, along with the potential danger to experimenter(s) from this setting. Finally, the amount of time that would be needed to run the number of experiments I performed would be well over a year, not including downtime due to damage and destruction of robots. The aforementioned reasons are why I implemented my work in a simulated environment (similar to [Gunn, 2011; Nagy, 2016]).

4.3 The Simulated Environment

I implemented and tested my framework using the Stage API [Vaughan, 2008] which is a well-established multi-robot simulator used to implement the existing portions of this framework. Using a simulated environment like Stage to evaluate my work is advantageous because all experiments can be repeated, and they can be run faster than in real-time [Vaughan, 2008]. In addition, using simulation allows me to focus on the core part of my research rather than the many issues that could arise from keeping a large team of heterogeneous physical robots operating consistently across many experimental trials.

The Stage API provides interfaces for creating virtual environments, as well as robots and other objects that can interact in these environments. The behaviours and interactions between these objects are implemented using a programming interface that Stage provides. Figure 4.1 shows an example of a randomly-generated USAR environment, with Figure 4.2 showing a closer view that highlights important objects used in my work.

Similar to the work from which mine extends [Gunn, 2011; Nagy, 2016], every experimental run starts with two robot teams entering the environment through openings that represent doorways.

4.3.1 Environment Objects

- **Fire**: Red objects depicted in Figure 4.2 represent occurrences of fire within the environment and can only be put out by robots with fire-extinguishing equipment

- **Stairway**: In my framework, I implement a multi-storey structure, and stairways are the medium by which robots can move from one level to another. Robots without stair climbing capabilities that want to use a stairway have to request to be carried by a robot which is able to traverse stairs.

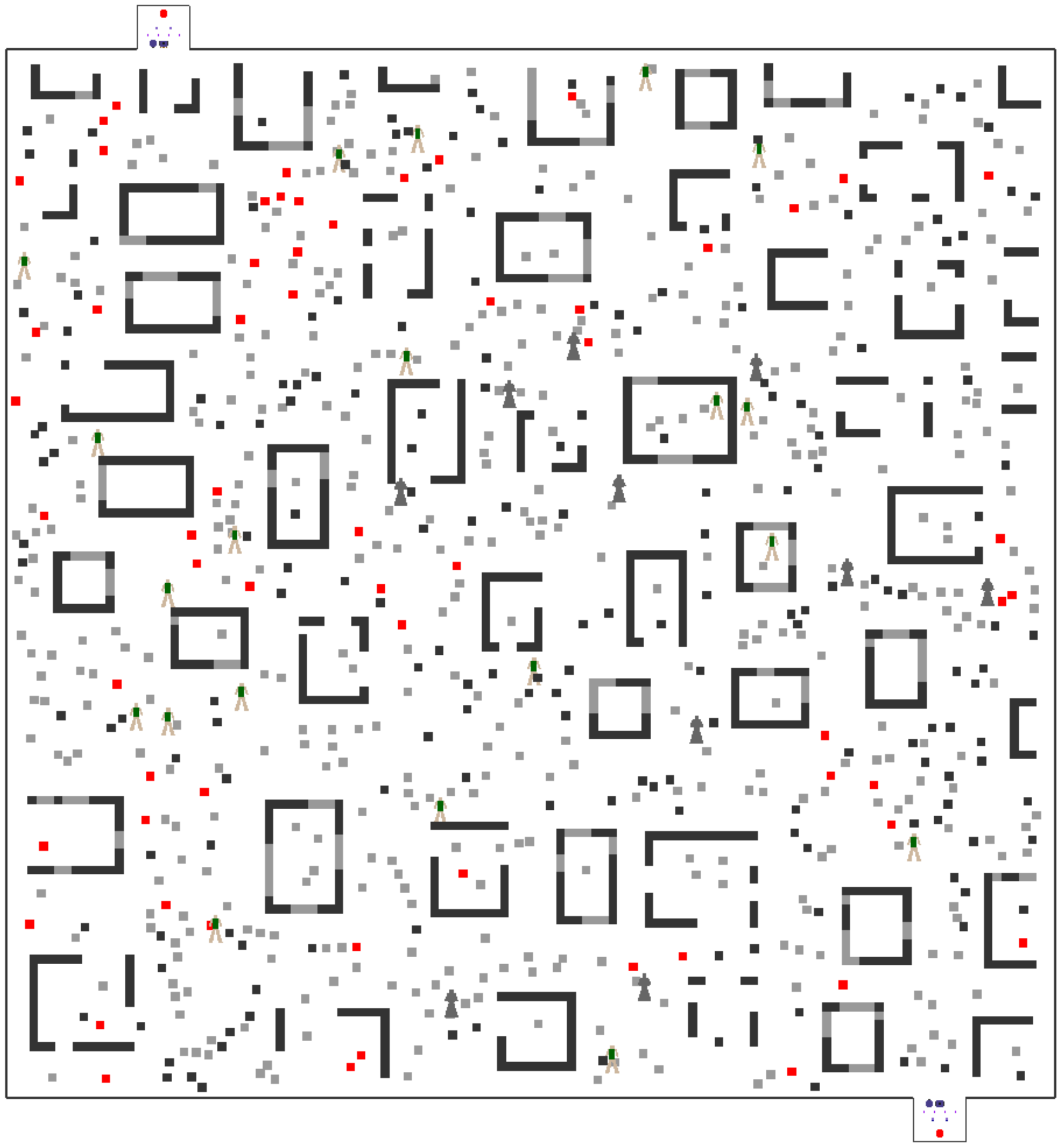

Figure 4.1: Sample of a randomly generated world in my framework

- **Removable debris**: Light grey objects as shown in Figure 4.2 represent low-lying debris. Robots with a tracked drive system are able to traverse such objects whereas robots with wheeled drives are forced to navigate their way around because they might get stuck trying to go through. Robots with debris-removal equipment are able to clear such debris so robots with wheeled drive can go through. Also, they are able to help robots that get stuck.

- **Fixed obstacle**: Tall dark grey objects represent debris and serve as obstacles and thus are not passable by any robot within the environment. A robot that encounters this type of obstacle must navigate around it.
- **Victim**: Objects representing victims are randomly placed within the environment. These objects represent human casualties that are in need of help. There are two type of victim objects - ones that represent real victims and the other represents false victims. The false victims are debris configuration that resemble real victims; enhanced sensors are required to be able to distinguish a false victim from real victim.
- **Marker**: This is an object that a robot carries and drops near a detected potential victim if it does not detect any markers close to the victim. Each robot carries a maximum of two markers and can request a marker from another robot if its marker supply runs out.

4.4 Robot Types

This section describes the six robot types used in my implementation. The MaxBot, MidBot, and Minbot were developed in [Gunn, 2011] and are used in my work. [Nagy, 2016] added the DebrisBot to his framework and I use this robot type in

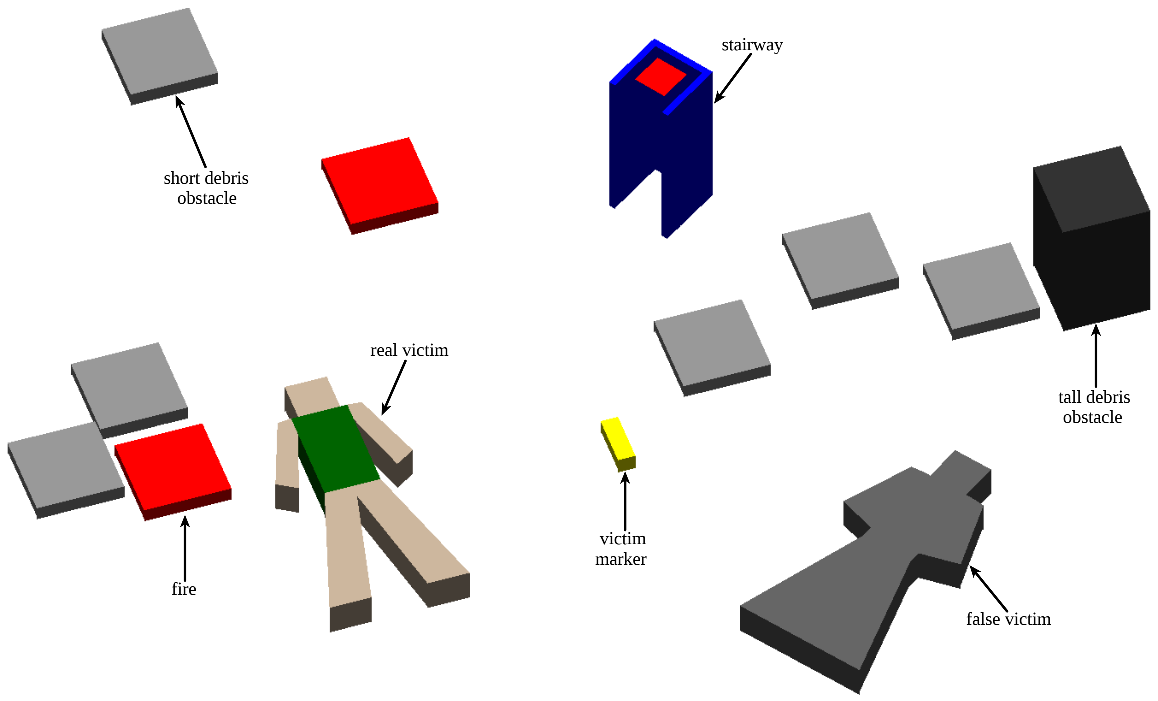

Figure 4.2: Objects within an environment as used in my research

my implementation as well. I have added two additional robot types to my framework, the FireBot and the StairBot. Figure 4.3 shows the different robot types used in my framework.

4.4.1 MinBots

The MinBot is a very small highly expendable robot with very basic victim-sensing abilities and can be used for potential victim discovery and exploration. It has a limited array of sonar sensors which it uses to avoid obstacles and update its map of the environment. MinBots do not have robot detection sensors, thereby limiting the level of interaction they can have with other robots [Gunn, 2011].

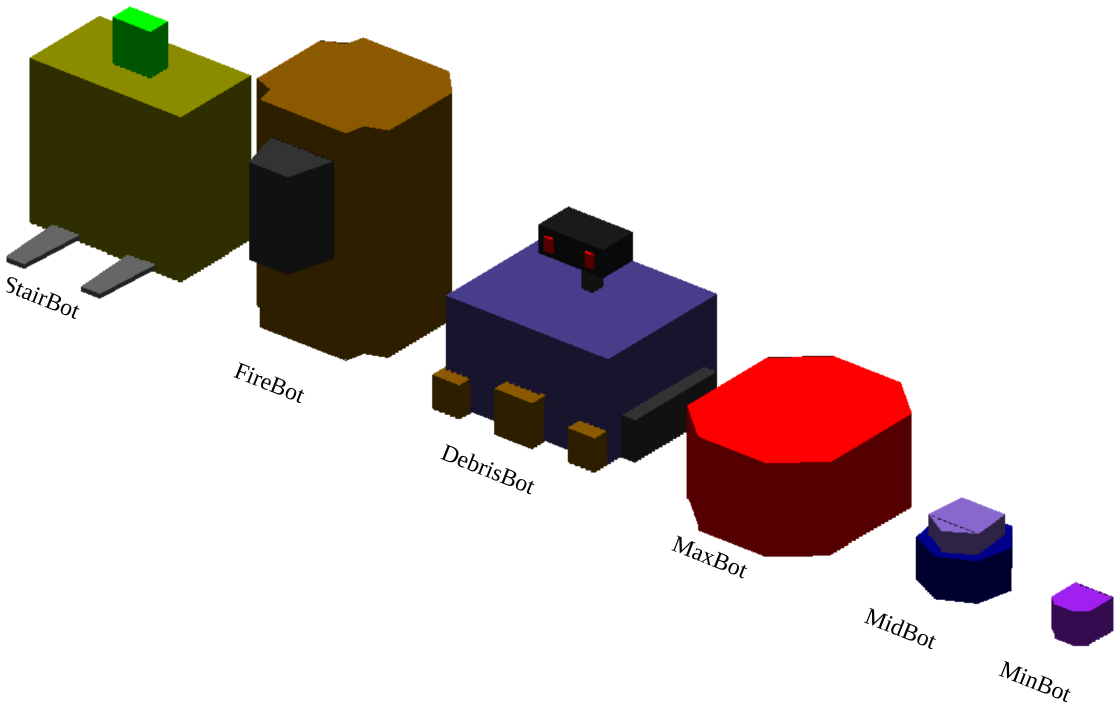

Figure 4.3: The different robot types used in my framework

Because of their limited computational and memory capabilities, MinBots are not able to assign tasks, combine shared environment maps, detect frontiers, generate frontier exploration tasks, or perform path-finding algorithms. Their inability to perform these tasks makes them inadequate to fill any leadership roles. MinBots are not allowed to assign tasks within the framework but they are able to perform recruitment. Although a MinBot is allowed to execute recruitment tasks, the absence of a planner or a shared team map means it cannot give other robots path information when recruiting them for tasks. This increases the chances of a robot recruited by a MinBot getting stuck or lost because of the lack of path information that would guide the recruited robot around any obstacles on its path.

4.4.2 MidBots

MidBots are much larger robots compared to MinBots and have improved computational power and memory, thus making them capable of occupying leadership positions assuming no better suited robots are available [Gunn, 2011].

This robot has advanced victim-sensing abilities and can correctly distinguish real humans from false victims at close distances (4.0m). In addition, MidBots can detect frontiers, generate frontier tasks, maintain a combined team environment map, and perform task assignment. The MidBot, however, lacks path-planning abilities and tasks assigned by the MidBot would not include path information. This means that assigned robots could get stuck while navigating to their assigned task locations.

4.4.3 MaxBots

MaxBots have large memory capacities and high computational capabilities (the ability to detect frontiers, maintain shared team maps, and assign tasks) making them adequately suitable for leadership roles. Despite these abilities, a MaxBot has basic victim sensing abilities and must rely on a Midbot to help it correctly distinguish real from false victims. The MaxBot is also equipped with a tracked drive system that enables it to traverse low-lying debris [Gunn, 2011].

4.4.4 DebrisBots

The DebrisBot is a robot equipped with debris-removing equipment used to break down and remove low-lying debris. DebrisBots have two sonar arrays to allow them to differentiate between removable low-lying debris and much taller obstacles that cannot be cleared.The DebrisBot has basic victim sensing capabilities to allow it to distinguish between low-lying debris and potential victims. DebrisBots also have a robot detector so as not to mistake robots for low-lying debris [Nagy, 2016].

The DebrisBot is a very specialised robot for performing debris-removal tasks and is unsuitable for leadership roles. It cannot detect frontiers, does not maintain a shared environment map, and lacks path planning and task-assignment capabilities.

4.4.5 FireBots

I implemented the FireBot, a highly-specialised robot whose main purpose is to extinguish any fire it encounters in the environment. FireBots are equipped with fire extinguishers which blast a chemical mix onto any fire nearby, eventually putting it out (the simulation does not deal with different types of fires that would require different extinguishing methods). The FireBot attempts to extinguish fires when it is performing the *extinguish fire* task. To simulate the extinguishing of a fire, I implemented additional components within Stage to simulate the effect activating the extinguisher has on the fire until it is eventually extinguished (A FireBot spends a maximum 6 seconds trying to put out a fire). For FireBots not to be damaged while attempting to put out fires, I set the range of the fire-extinguishing equipment to 4.0m.

FireBots are equipped with fire-sensing capabilities (I used Stage's *fiducial sensor* model to represent a fire sensor) and have a tracked drive system to enable them to traverse through low-lying debris. They also have a sonar array placed 20cm from its base that they use to detect tall obstacles. To reduce the possibility of driving over victims (and furthering injuring them) or running over smaller robots, FireBots are equipped with both basic victim sensing and robot detection capabilities.

This robot is poorly suited for leadership as it lacks planning abilities, does not maintain shared map knowledge, cannot detect frontiers, and does not have task-assignment capabilities.

4.4.6 Stairbot

The StairBot is a very specialised robot whose main task is to transport robots between different floor levels in a multilevel environment. It is equipped with a *teleporter* which enables it to carry a single robot along, up or down a level using a stairway. To simulate the movement between multiple levels in the environment, I implemented extra facilities within the Stage simulator. All stairways have a transit time (which simulates how long it takes a robot moving from one level to appear on another level) of 4 seconds. StairBots activate their stair-climbing ability only if there is a pending request from a robot to be stair lifted and the StairBot along with the requesting robot are within 30cm of a stairway. This is done to prevent StairBots from ascending or descending stairways any time they are within the vicinity of one.

I use Stage's fiducial sensor to create a *stairway detector* which I use to abstract the identification of stairways in the environment. Similar to the FireBot, the StairBot has a tracked drive system and is equipped with both a robot detector and a basic victim detector.

The lack of planning abilities, inability to detect frontiers, lack of task-assignment capabilities, and inability to maintain a shared team map of the environment makes it a poor candidate of choice to occupy a leadership role.

4.5 Attributes, Tasks, and Roles

4.5.1 Attributes

As mentioned in Section 2.4.1.1, each robot within the framework is described using a set of characteristics and capabilities known as attributes, that allow robots to estimate their own abilities as well as that of others. These attributes are divided into three main categories: physical, computational and sensory attributes.

4.5.1.1 Physical Attributes

Physical attributes define properties of the robots that are useful for determining features such as speed, size, how they navigate through the environment, as well as any special equipment they possess. Table 4.1 below shows these attributes and their values for every robot type in my framework. I have added the *fire extinguisher* and *teleporter* to the previous framework [Gunn, 2011; Nagy, 2016] to enable robots to know whether they can put out fires or stair-lift other robots.

Table 4.1: Physical properties and attributes of robots used

	MinBot	MidBot	MaxBot	DebrisBot	FireBot	StairBot
locomotion	wheeled	wheeled	tracked	tracked	tracked	tracked
length×width(cm)	10×10	20×20	44×38	40×50	45×35	40×40
expendability	1.0	0.25	0.05	0.1	0.1	0.1
debris remover	no	no	no	yes	no	no
fire extinguisher	no	no	no	no	yes	no
teleporter	no	no	no	no	no	yes
marker count	2	2	2	2	2	2

4.5.1.2 Computational Attributes

Computational attributes are heuristic definitions of all the possible capabilities of robots and are defined within a robot's control software. They give an indication of whether a particular robot is equipped with the necessary computational and memory resources before it attempts to execute a task.

4.5.1.3 Sensory Attributes

Sensory attributes specify what type of sensors robots are equipped with. Table 4.3 shows the different robot types and the sensors they are equipped with.

Table 4.2: Capabilities of robots used

	MinBot	MidBot	MaxBot	DebrisBot	FireBot	StairBot
victim tracker	yes	yes	yes	yes	yes	yes
frontier finder	no	yes	yes	no	no	no
maintain team map	no	yes	yes	no	no	no
assign tasks	no	yes	yes	no	no	no
planner	no	no	yes	no	no	no

4.5.2 Task Types

4.5.2.1 Gunn Framework Tasks

The tasks described in this section are those which were developed in the original framework [Gunn, 2011] and are used directly in my own implementation.

- *Explore*: This is the lowest priority task within the framework and involves robots randomly exploring the environment. All robots perform this task in their idle state, except for the DebrisBot.

- *Explore Frontier*: A robot performing this task moves to a specified location, explores the area briefly, and reports any findings to its team leader. Only robots with a frontier finding capability can generate this type of task.

- *Find Team*: This task is used by replacement robots when they are initially

Table 4.3: Physical properties and attributes of robots used

	MinBot	MidBot	MaxBot	DebrisBot	FireBot	StairBot
victim sensor	basic	advanced	none	basic	basic	basic
robot sensor	no	yes	yes	yes	yes	yes
fire detector	none	basic	basic	basic	advanced	basic
stairway detector	yes	yes	yes	yes	yes	yes
sonar sensor	5	10	3	14	8	12
sonar range	4m	6m	2m	6m	6m	6m
laser rangefinder	none	none	yes	none	none	none
victim marker detector	yes	yes	yes	yes	yes	yes

released into the environment. Robots executing this task will travel inwards along the bearing it was introduced into the environment for a total of five minutes, or until another team is encountered which it can join (or possibly form a new team with another robot).

- *Find Victim*: This task is used to determine a robot's suitability for tracking victims and it is not explicitly placed on the task queue.

- *Confirm Victim*: Robots with advanced victim detection sensors use this task to move to a specified location where there is a potential victim and confirm whether it is an actual victim.
- *Manage Team*: This task is used to represent the requisite capabilities needed for a robot to be able to manage a team (e.g. make task assignments)
- *Encounter*: This task serves as a guide for the team merge and redistribution process when two teams encounter each other within the environment.

4.5.2.2 ARC Framework Tasks

The tasks defined here were implemented in the previous framework [Nagy, 2016] and are used directly in my work.

- *Begin Role Recruitment*: This task is used to initialise the role-recruitment process when active recruitment is turned on.
- *Find Robot*: This task is used to locate other robots when active recruitment is enabled. A robot performing this task will perform a random walk to increase its chances of finding a useful robot.
- *Unguided Debris Removal*: This is the idle task for the DebrisBot and it involves preforming a random walk, and removing any low-lying debris within its field of view.
- *Guided Debris Removal*: This task involves a DebrisBot moving to a specified location (presumably one where another robot may be stuck) and attempting to remove debris.

- *Donate Marker*: This task guides the marker donation process in the case where one robot has agreed to donate a victim marker to another robot.
- *Wait For Marker*: This is a task that guides the actions of a robot that has requested a marker and is awaiting a donation from another robot. If after some period, the requesting robot does not receive a marker from the donating robot, the requesting robot gives up the task with the hope of getting a marker from another robot.

4.5.2.3 My Framework

I developed several specialised task descriptions to deal with the challenges I faced during my implementation, and these tasks are overviewed in this section.

Extinguish Fire The *extinguish fire* task is a very specialised task: only robots equipped with fire extinguishing equipment (Table 4.1) can perform it. In my implementation only the FireBot can execute this task. This task involves a FireBot moving to a specific location and attempting to put out any fire at that location. This task relies on the *detect fire* perceptual schema to keep track of actual fires and remove fires when they are extinguished. Also locations which were incorrectly reported to have fires are discarded.

When a robot gets to a location where a fire is located, it pauses any active motor schemas and attempts to put out the fire by discharging the extinguisher at it. FireBots in my framework are given a maximum of three attempts (each attempt lasts for 2 seconds in my work) to put out a fire, so they do not spend so much time trying to put out a single fire. If a robot uses up its three attempts and the fire is still visible, the robot marks this as a failed attempt and abandons that task. A FireBot also abandons the task if it spends more than 30 seconds while attempting to move to the fire location. After the FireBot abandons the task, it gets the next active task from its task queue to perform. If the fire is extinguished before the three attempts are up, the robot marks that task as successfully completed and moves on to performing the next task in its queue.

Manage Stairway This specialised task can only be executed by a robot with a *teleporter*, hence only StairBots are able to perform this task. In this task, a StairBot limits its movement close to a stairway and never goes beyond some maximum radius, which I define as 2m in my implementation. This task is generated by a StairBot whose team membership comprises one robot (i.e. the StairBot itself) when it comes within detection range of a stairway and its satisfaction value is below the satisfaction threshold (Section 4.8.2). As long as the robot's satisfaction value is above the satisfaction threshold (Section 4.8.2), the robot will camp at this location and perform *stair lift* tasks. When the Stairbot's satisfaction drops because it has not performed any *stair lift* tasks for some time, the robot deletes this task and goes on to perform the next task on its queue. This new task should enable the robot to move to other regions in the environment (e.g. to ultimately find a new staircase elsewhere where it can do the work for which it is purposed).

Stair lift A robot within the environment that encounters a stairway and does not have a *teleporter* can request to be moved onto another floor. The robot pauses all its active motor schemas once this request is made. Such a request generates a *stair* *lift task* which contains location information and this is embedded in the message request. Since StairBots are the only robot types in my implementation that have *teleporters*, only they can respond to such a request.

If a *stair lift* task becomes a StairBot's active task, the StairBot will attempt to move to the location from which the request came by using the *move to location* schema (Section 4.6.2). When the robot gets within detection range of the stairway or the requesting robot, it activates the *move to robot* schema (Section 4.6.2) so it can physically meet with the requesting robot. Once the StairBot meets with the requesting robot, it activates its *teleporter* and carries the robot up or down a level depending on the robot request. A robot requesting a stair lift waits for 30 seconds and if it does not physically come into contact with the responding StairBot, it resumes its motor schemas and goes on to perform the tasks on its task queue. On the other hand, a StairBot that takes more than 30 seconds to physically meet the requesting robot, abandons that task and deletes it from its task queue. It then selects a new task to execute from its task queue.

4.5.3 Role Types

In this section I describe all roles that are used in my work. The following roles already exist in the previous framework: *team leader*, *coordinator/explorer*, *explorer*, and *debris remover*. As part of my implementation, I have added two more roles – *fire extinguisher*, and *stair lifter*. Recall from Section 2.4.1.1 that roles are used as a heuristic to streamline the task-assignment process. Suitability values are calculated for every robot-role combination using the attributes of the different robot types

2.4.1.1. These values are used to determine how well a robot occupying a role satisfies the task requirements of that role [Gunn, 2011]. Table 4.4 shows all the possible robot-role suitability combinations. As indicated by [Gunn, 2011], the importance of these suitability values are seen when used relatively to one another and not in the actual numeric suitability values themselves. All tasks making up a role are assigned hand-tuned weights which indicate how important a task is to that role. For example, a Debris Remover role places a high emphasis on the *guided debris removal* task, thus robots with a debris remover (DebrisBots) will have a higher suitability value for the Debris Remover role. There are scenarios where a robot fails to meet any minimum suitability requirements and thus scores a suitability of zero.

Table 4.4: The possible robot-role combinations with their suitablities

	MinBot	MidBot	MaxBot	DebrisBot	FireBot	StairBot
Leader	11	83	106	26	26	23
Coordinator/ Explorer	24	136	52	32	32	27
Explorer	124	56	52	52	52	17
Debris Remover	0	10	10	110	10	10
Fire Extinguisher	5	15	15	15	115	15
Stair Lifter	5	15	15	15	15	105

4.5.3.1 Team Leader

The team leader is responsible for guiding and coordinating the efforts of all robots on its team. This typically involves assigning tasks, maintaining a combined shared map of the environment that its team has explored, and detecting and assigning frontiers to be explored by team members. Of all the robots in my work, the MaxBot is best-suited for this role. This is because it possesses frontier-finding software which helps to efficiently target areas to be explored rather than targeting random locations. The MaxBot also has a planning module which enables it to build paths to locations of interest (one where a potential victim possibly is), to help guide robots to these locations. In addition to the previously mentioned capabilities, the MaxBot has the ability to assign tasks, and this is a necessary capability when it comes to managing a team.

The MidBot is somewhat suited for the leadership role as it possesses much of the same computational abilities as the MaxBot with the planning module excluded. This means a robot assigned a task by a MidBot is more likely to get stuck or lost while navigating to the task location. The MinBot, DebrisBot, FireBot, and StairBot lack the basic capability for this and are thus are very unsuited for this role.

4.5.3.2 Coordinator/Explorer

Robots in this role are primarily responsible for performing victim verification tasks, exploration tasks, and also guiding the team redistribution and merging process in the event that two teams encounter one another. The MidBot is the only robot equipped with advanced victim-detection capabilities and is therefore the best-suited for this role.

The MinBot, MaxBot, DebrisBot, FireBot, and StairBot are poorly suited for this role as they only have basic victim-tracking capabilities. The MaxBot is equipped to handle the team distribution process while the FireBot and StairBot are not adequately equipped to handle the team distribution process. However, they have a robot detector which will help them perform team redistribution if necessary.

4.5.3.3 Explorer

Robots occupying this role are mainly expected to perform exploration tasks as directed by a team leader, usually with the aim of finding potential victims in unexplored areas. Exploration tasks could be potentially dangerous, resulting in a robot becoming damaged or lost, and thus the MinBot is an ideal fit for this role. This is because it is highly expendable (i.e., it is cheap and easily replaceable).

The other robot types have the necessary capabilities required by this role but because they are better suited to fill more advanced or specialised roles, they have lower suitability values for this role.

4.5.3.4 Debris Remover

This role involves clearing low-lying debris from the environment and robots filling this role are required to have debris removal equipment. This requirement makes the DebrisBot the only robot well-suited for this role.

Since none of the other robots possess a debris remover, they are totally unsuited to occupy such a role.

4.5.3.5 Fire Extinguisher

Robots in this role are expected to detect and put out detected fires within the environment. Specifically, robots in this role are expected to perform the *extinguish fire* and should possess fire-detection and fire-extinguishing abilities. In my implementation, the FireBot is the best suited robot type for this role as it has advanced fire-detection capabilities and fire-extinguishing equipment.

The MaxBot, MidBot, DebrisBot, and StairBot are poorly suited for this role because of the lack of fire-extinguishing equipment, but they do have a basic fire-sensor. The MinBot lacks both fire-detection and fire-extinguishing capabilities and is thus utterly unsuited for this role.

4.5.3.6 Stair Lifter

This is a very specialised role that involves a robot equipped with teleporting capabilities transporting another robot from one level of the environment to another. For a robot to be able to carry out this task, it should possess a stairway-detector and a teleporter. This makes the StairBot the best-suited robot type for this role in the framework.

All the other robots in my implementation possess stairway-detection capabilities but lack that of teleporting and are thus poorly suited for this role.

4.5.4 Ideal Team

As mentioned previously in Section 2.4.1.2, the ideal team is used as a guide to help determine how best to restructure or integrate new potential team members, and it describes the desired number of each robot role within a team. In my work, an ideal team comprises 1 *team leader*, 1-2 *coordinators/verifiers*, 3-10 *explorers* (as defined in Gunn [2011]), and 1-2 *debris removers* [Nagy, 2016]. For the new roles I have added, I specify that a team should have 1-2 *fire extinguishers* and 1 *stair lifter*. The ideal team definition is shown in Figure 4.4.

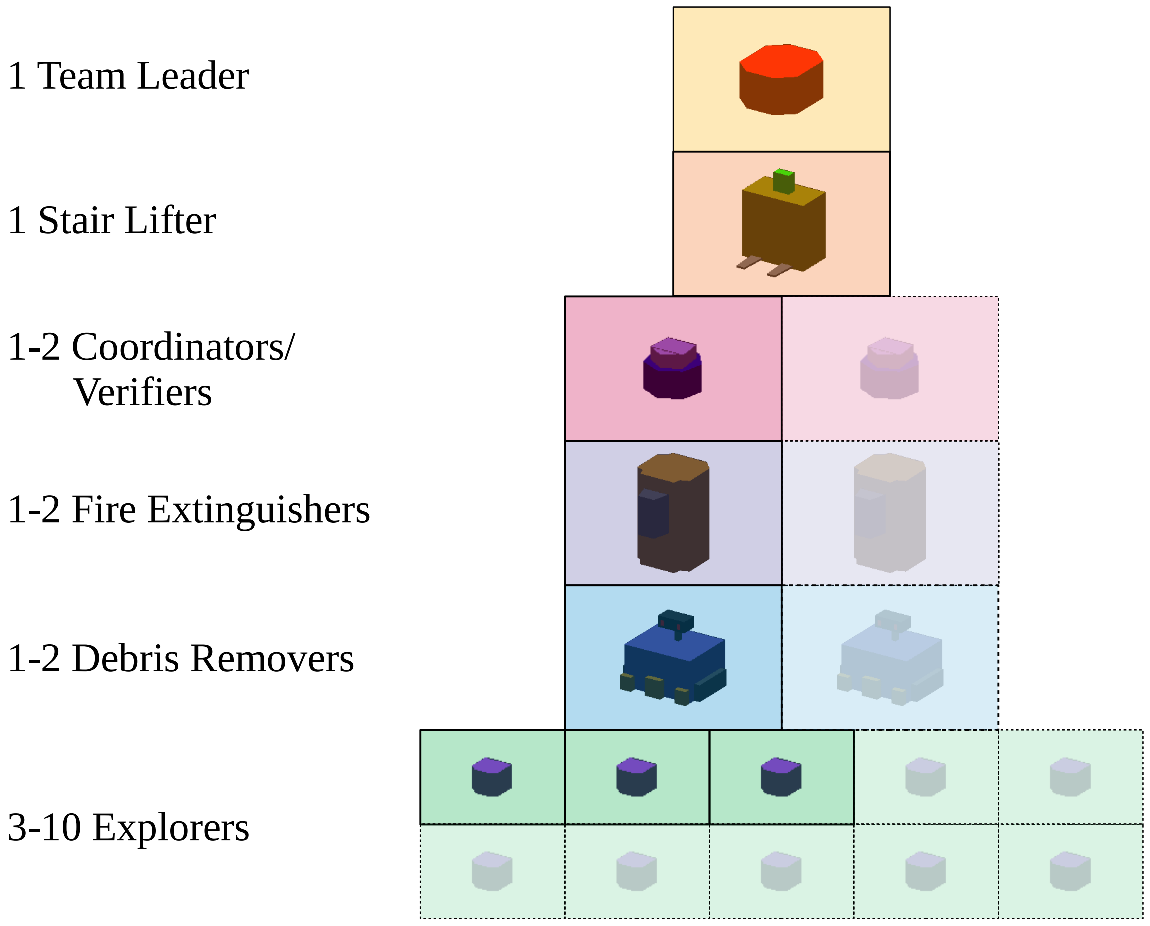

Figure 4.4: An ideal robot team as used in my implementation.

The number of robots filling the first three roles were based on values obtained via experimentation in Gunn [2011]. For the *debris remover* role, I used the same number of robots specified for the role as defined in Nagy [2016]. I chose to specify a minimum of 1 and a maximum of 2 *fire extinguishers* in my ideal team definition to allow for stray FireBots to be integrated into the team. I specified just 1 *stair lifter* in my ideal team definition because I only have a maximum of two levels in my experimental environment and two stairways connecting them, thus 1 StairBot on a team would be able to fulfil any stair-lifting requests. Any stray StairBots that cannot be integrated into a team because that role is already occupied would be available to be used by all teams within the environment. In other environments, a greater number of StairBots would be more suitable.

4.6 Autonomous Control

Here, I briefly describe the low-level control mechanisms for robots used in my implementation. All but the *detect fire* and *detect stairways* schemas were already implemented in the existing framework [Gunn, 2011; Nagy, 2016].

4.6.1 Perceptual Schemas

These are used to process sensory inputs from the robots so a robot can have a meaningful representation of the environment.

- *Localization*: This schema tracks a robot's position and orientation. All members of the same team use the same coordinate system which is centred at the teams initial position in the environment. Different teams have their own translation vectors so that location information can be exchanged between teams. This schema also determines if a robot is stuck on some debris and generates a *guided debris removal* task if the robot is still stuck after 5 seconds.

- *Process Range Data*: This schema is used to get information about nearby obstacles by processing data from rangefinding sensors.
- *Detect Debris*: This schema uses the localization schema to determine if a robot is stuck and then logs that there is an obstacle at that location. This information helps a robot avoid that obstacle again in the future.
- *Detect Obstacles*: This schema uses information from the *process range data*, *detect debris*, and *detect robots* perceptual schemas to generate a list of obstacles encountered in the environment.
- *Detect Robots*: This schema uses the robot detector sensor to build a list of currently observed robots.
- *Detect Victims*: This schema uses input from the victim detector sensor to build a list of currently observed victims.
- *Detect Lost*: This schema tracks how far a robot travels while attempting to move to some location. Once the robot travels five times the required distance without reaching its destination, the robot deletes the current task from its queue. In the case when the task is a confirm victim task, rather than deleting, the robot re-queues the task to attempt at a later time.
- *Detect Markers*: This schema uses input from the marker detector sensor to build a list of currently observed victim markers in the environment.

- *Detect Fire*: I implemented this schema to build a list of currently observed fires using input from the fire detector sensor.
- *Detect Stairways*: I implemented this schema to build a list of currently observed stairways using input from the stairway detector sensor.

4.6.2 Motor Schemas

Motor schemas are used to determine what sequences of action a robot will take based on current information from its perceptual schema.

- *Avoid Obstacles*: This schema generates a repulsive force intended to make a robot steer clear of any detected obstacles.
- *Move To Location*: This schema enables a robot to move to a specified location in the environment. If path information is available, then the robot uses it, otherwise it uses a reactive approach to get to its destination.
- *Turn In Place*: This schema generates an action vector that instructs a robot to rotate in a clockwise direction without moving forward or backward.
- *Random*: This schema helps prevent a robot from getting stuck by generating a small random magnitude motion vector that points the robot in a new direction.
- *Recover Stuck*: This schema uses the *localization* perceptual schema to recognise when a robot is stuck. In such a scenario, an action vector commanding the robot to back up and attempt to free itself is generated.

- *Seek Debris*: This schema uses information from the *detect obstacles* perceptual schema to generate an action vector that attracts a robot towards short obstacles while repelling it away from tall obstacles (non-removable debris) or objects that are not debris (e.g. victims).

- *Move To Robot*: This schema generates a motion vector that guides one robot towards another robot using information from the *detect robots* schema ensuring that the two robots physically meet.

4.7 Framework-Specific Modules

This section describes the functional modules that are used to support my implementation. Except for the *location and rare skill manager*, all of these modules are already implemented in the existing framework [Gunn, 2011; Nagy, 2016].

- *Encounter Manager*: The *encounter manager* module handles the team merge and redistribution process when two robots encounter one another in the domain.

- *Knowledge Manager*: The *knowledge manager* tracks any information that is known to the robot. This includes but is not limited to the robot's own attributes, attributes of other known robots, as well as its current team structure.

- *Communication Manager*: This module is responsible for processing all wireless messages that a robot receives from others.

- *Role Manager*: The *role manager* is responsible for managing all role- and team-related processes. This involves performing role switch checks and handling role-level recruitments.
- *Task Manager*: This module handles all a robot's tasks as well as information about these tasks. It also manages the task queue and all communications and processes pertaining to task assignment.
- *Location and Rare Skill Manager*: This module manages information (tasks and task satisfaction) regarding a task locale as well as information about whether a robot has a rare skill. When a robot is looking to increase its satisfaction and as a result it leaves its team, this manager provides a list of the top three locales for the robot to visit with the aim of increasing its satisfaction. This manager also alerts a robot to the possibility of the robot having a rare skill in the environment.

4.8 Mission-Specific Modules

This section describes the modules that provide the required logic to help robots achieve high level goals in the USAR domain.

- *Environment Mapper*: This module is responsible for maintaining a shared team map of the environment by integrating results obtained from exploration.
- *Frontier Finder*: This module is used to generate *explore frontier* tasks by examining a robot's map of the environment. To determine areas to explore, an occupancy grid-based approach is used.

- *Victim Tracker*: The victim tracker module maintains and updates a robot's list of currently known victims as well as their statuses (potential or actual).
- *Planner*: This is used to calculate paths to locations of interest (e.g., victims) that assigned robots could use so as not to get stuck or lost in their attempt to move to the goal.
- *Marker Manager*: This module is responsible for handling the marker release process when victims are detected in the environment.

4.8.1 Satisfaction Manager

I implemented the *satisfaction manager* to handle the tracking of a robot's satisfaction and use this information to determine a robot's course of action.

4.8.1.1 Satisfaction Expression

Recall from Section 1.4 that I define satisfaction as an estimation of a robot's usefulness on its team. In order to formulate a *satisfaction expression*, I make two general assumptions, and they are: 1) a robot gains more satisfaction from working on a task for which it is well-suited, and 2) the longer a robot spends performing a task, the more its satisfaction is likely to reduce.

To formulate the satisfaction expression, I first estimated the satisfaction a robot derives from completing a single task (Equation 4.1) by using how long it took the robot to successfully complete that task and the robot's suitability for that task.

$$TS = \frac{S_{task}}{\Delta t} \tag{4.1}$$

TS refers to the satisfaction a robot gets from completing a specified task, S_{task} is the suitability of the completed task, and Δt represents the time duration for the task (i.e. when the robot started executing the task and when it successfully completed it). I also define a robot's task rejection rate as the likelihood that a robot would reject a task if it is requested to perform one. This is shown in Equation 4.2.

$$TR(t_i) = \frac{\sum_{n=t_{i-1}}^{t_i} T_{reject}(n)}{\sum_{n=t_{i-1}}^{t_i} T_{request}(n)} \tag{4.2}$$

TR represents the task rejection rate (i.e. a ratio between the number of rejected task requests and the total number of task requests received), T_{reject} is a task request that a robot rejected, and $T_{request}$ is a request to complete a task.

I then used Equations 4.1 and 4.2 to define a robot's own satisfaction at any time t, as shown in Equation 4.3

$$RS(t_i) = \frac{\left(\sum_{n=t_{i-1}}^{t_i} TS(n) + TS_{future}\right) * TR(t_i)}{t_i - t_{i-1}} \tag{4.3}$$

Here, $RS(t_i)$ is the robot's overall satisfaction at a particular instance of time; TS is the task completed satisfaction as defined by Equation 4.1; TS_{future} refers to how much satisfaction a robot could derive upon completing some task in its task queue. In principle, I use Equation 4.1 to estimate the satisfaction of a task in the task queue that is not currently being performed. Since this task has not yet been performed, a robot estimates how long it would take to complete the task by using how long it would take it to get to that task location based on its current speed (I assume a straight path with no obstacles) and how long it will ideally take to complete the task. For example, consider a DebrisBot having a *guided debris removal* task as a future task on its task queue. In the framework, a DebrisBot has a maximum of 5 seconds to remove a piece of debris once it gets to the debris location. The estimated time it will take a DebrisBot to complete this task, assuming it will take it about 10 seconds to get to the task location based on its current speed is thus 15 seconds. TR is the robot's rejection rate defined in Equation 4.2.

4.8.2 Tracking Robot Satisfaction

In my implementation all robots calculate their satisfaction every 30 seconds using the satisfaction expression defined in Equation 4.3. As mentioned in Section 3.2.1.1, I have provided discretised states which indicate how aggressively the robot should be attempting to rectify its lack of satisfaction. In my implementation I create a *willingness* flag which can hold three possible values: *SM_STAY* which refers to the not willing state, *SM_WILLING* refers to the somewhat willing state, and finally *SM_LEAVE* which refers to the willing state.

Every 30 seconds a robot in my framework will calculate its current satisfaction on its team and based on this value the robot will set its willingness flag accordingly. To know what value to set the willingness flag, a robot compares its current satisfaction value to the satisfaction threshold. The satisfaction threshold I used in my work was obtained from initial experimentation by hand tuning values. In preliminary experiments (I describe this in Section 4.9), I realised that using a value of 140 resulted consistently in the highest number of actual victims being successfully identified; thus I settled on this value. Future work could explore the option of using learning techniques to have each robot determine their own satisfaction thresholds, since what one robot may see as satisfactory might not necessarily be satisfactory to another. If a robot's current satisfaction falls between 110 and 170, the robot sets it willingness flag to *SM_WILLING*. A robot's satisfaction falling below 110 would set its willingness flag to *SM_LEAVE* and a value above 170 sets the willingness flag to *SM_STAY*.

If a robot's willingness flag is *SM_STAY*, the robot is satisfied with the work it is doing and it will not take any active measures to improve its current satisfaction. In such a scenario, the robot will likely not be receptive of task requests coming from outside its team (this is not the case when the robot's skill has become rare, as will be discussed in Section 4.8.2.3). In the case where the willingness flag is set to *SM_WILLING*, a robot will perform a role switch check to see if there is an open role which suits it better than the one it currently occupies. If there is, it switches into that role. Also, a robot in this state will be more receptive of task-based recruitments (but not role-based recruitment) with the aim of increasing its current satisfaction value. Finally, a robot with its willingness flag set to *SM_LEAVE*, will be more receptive of role-based recruitments which would essentially let it switch teams. A robot that remains in this state and does not receive any role-based recruitments for 15 seconds sends a message to its team leader indicating that it wants to leave the team and the leader upon receiving this message employs the mechanism described in Section 4.8.2.1 to determine whether it should let go of that member or not.

4.8.2.1 Tracking Members

In addition to a robot being able to track its own satisfaction, a leader also keeps tabs on what its team members do. A leader tracks its team members' productivity using Equation 4.4.

$$LS(r_j, t_i) = \frac{\sum_{n=t_{i-1}}^{t} T_{accept}(r_j, n)}{\sum_{n=t_{i-1}}^{t} T_{request}(r_j, n)} \tag{4.4}$$

LS is the team leader's satisfaction with a member at a particular time. Here, r_j is the robot whose satisfaction is being calculated by the leader and t_i refers to the time period at which a leader tracks a member's satisfaction, T_{accept} refers to a task that was successfully assigned to a member, and $T_{request}$ is a task a member has been requested to complete.

In my work, if a robot's acceptance rate of work from its team leader's perspective falls below 0.5, the leader sends that particular robot a message requesting its current willingness state. A robot receiving such a message will then check its willingness flag and send a response back to the leader. If the leader receives a message indicating that the member's willingness flag is set to *SM_STAY*, it means the member is currently satisfied with its work on the team and the leader does not take any further action. If the response a leader receives has the member's flag set to *SM_WILLING*, the leader follows up with another message telling the member to set its willingness flag to *SM_LEAVE*, thus allowing the robot to be more accepting of role-based task recruitments or eventually leave the team with the aim of improving its satisfaction. If the leader receives a message indicating the member's willingness flag is set to

SM_LEAVE, the leader does not take any action since the member is already poised to take a more aggressive approach to increase its satisfaction on its own. A team leader calculates its team members' satisfaction every 60 seconds in my framework.

4.8.2.2 Tracking Location Satisfaction

In my implementation, I divided the environment into 3×3 grids which I refer to as *task locales* (See Section 3.2.3.1). A robot tracks all tasks it has performed within a task locale as well as the satisfaction it has gained from completing these tasks. Location satisfaction is calculated as a running sum of the satisfaction a robot gets upon completing a task (same as Equation 4.1) within a particular region. The location and rare skill manager stores this information and also logs all tasks completed within this region. Task locales are sorted in order of decreasing satisfaction. When a robot is let go by its team leader because its work was not satisfactory enough, the location and rare skill manager provides a list of the regions with the top three satisfaction values. A robot revisits these task locales with the aim of finding new work (which has since become available or it was not able to perform because of its commitment to staying with its team). Since robots could get stuck in a cycle of revisiting task locales, in my implementation robots are allowed to actively (i.e. through their own effort and not random exploration) revisit task locales only once.

4.8.2.3 Identifying Skill Rarity

To help me track how rare a skill is at some particular time, a robot tracks how many unique skill (See Section 3.2.2.1) requests it received over time from different robots and the teams those robots are currently on. In my implementation, receiving a request from more than one team indicates that there is a shortage of that particular skill within the environment. When a robot realises that a skill it possesses has become rare, the robot sets its willingness flag to *SM_STAY*. If a robot has its willingness flag set to *SM_STAY* and it has a rare skill, the robot stops accepting all recruitment requests (even those from its own team) except for those that request the use of its rare skill. When that skill is no longer rare, the robot will attempt to find its team. If it is unsuccessful, it joins another team or continues working independently.

4.9 Preliminary Experiment

To give me an idea of what value to use as the threshold for satisfaction such that it would lead to the highest performance, I conducted an initial experiment to investigate the performance of my framework when the satisfaction value threshold is set to different values. I used the percentage of victims successfully confirmed as the metric because the ultimate goal of any rescue mission is to save as many human casualties as possible.

I experimented with ten different base satisfaction values range from 40 to 220 using a step size of 20. In this preliminary experiment, I used a communication success rate of 60%. For the other experimental configurations, I used all recruitment approaches (active, concurrent, and passive), the three possible probabilities of robot failure (major, minimal, and no failures), and whether replacement robots were available or not. I used identical sets of 20 random seeds for each experimental configuration, ultimately resulting in 3600 different trials which were run approximately 36 at a time using Amazon Web Services [Amazon Web Services, Inc., 2019]. I averaged out all the results from trials that had the same satisfaction threshold. The graph shown in Figure 4.5 is a plot of the percentage of victims successfully identified using each of the ten base satisfaction values.

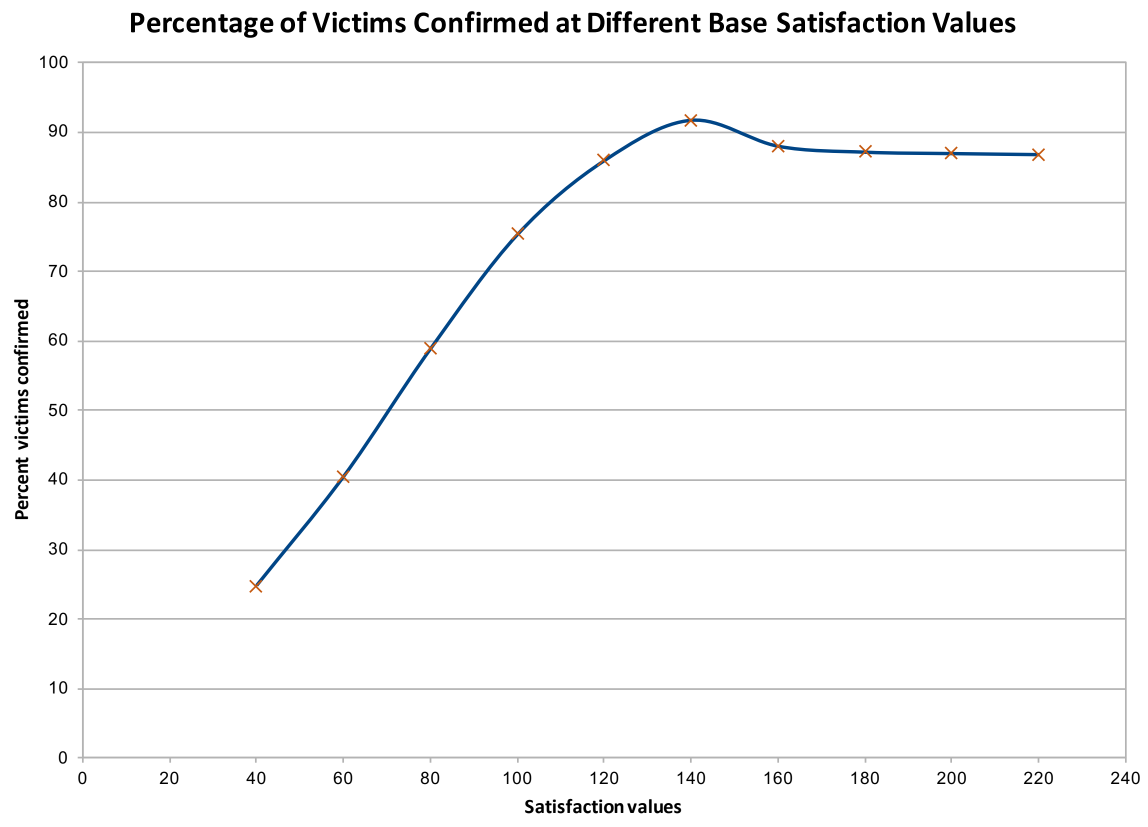

Figure 4.5: Results from the preliminary experiments to determine what satisfaction value to use as the base

The highest performance was achieved when the threshold for satisfaction was around 140. I used a range from 110 to 170 as the base satisfaction values in my implementation. In my framework, I use the values around the base satisfaction to indicate a robot is in the *somewhat willing* state. Any satisfaction values above the base satisfaction means the robot is in the *not willing* state and values below the threshold indicate the robot is in the *willing state*.

4.10 Conclusion

This chapter has described my implementation within a USAR domain that would allow me to evaluate my framework. In the next chapter, I describe the experiments I used in evaluating my work.

Chapter 5

Testing and Evaluation

5.1 Introduction

To evaluate the effectiveness of my solution approach, I compared it to the previous versions of the framework [Gunn, 2011; Nagy, 2016] developed in our lab. Experiments were conducted in a manner similar to these earlier works, using a simulated USAR domain. I evaluated my implementation by considering factors such as wireless communication reliability, the recruitment strategy used, and the probability that a robot could experience failures (whether temporary or permanent) during a mission.

In the following sections, I restate the research questions my work aims to answer and then describe the metrics used to evaluate my work. I then describe my experimental design, present the results from my experiments, and discuss my findings.

5.2 Review of Research Questions

Recall my research questions from Section 1.5.

1. **To what degree does tracking robot satisfaction affect the total teams' performance with respect to the overall mission in a USAR domain?**
2. **For specialised tasks that require the use of a robot's unique skill, is it more useful to have robots expending effort to stay with a team or should these efforts be redirected towards areas where its special skills may be needed?**

To answer my first research question, I conducted a set of experiments with satisfaction tracking (Section 4.8.1) turned on while varying parameters affecting communication reliability and robot failures for the three different recruitment approaches in the existing framework. For the second question, I introduced an extra floor level in my test environment and conducted experiments under varying levels of communication reliability, different recruitment approaches, use of replacement robots.

5.3 Evaluation Metrics

To evaluate the performance of my implementation, I track four metrics during every experimental trial. They are *percentage of real victims successfully confirmed, and known to any leader*, *percentage of area coverage as seen by any leader*, *number of extinguished fires*, and *number of successful stair lifts*

In my implementation, I keep track of the number of victims correctly classified by each robot. A correctly classified victim is one that has been confirmed as an actual victim as opposed to a false victim (Section 4.5.2.1). To know how many victims were correctly classified, I use the number of positively confirmed victims reported to a team leader, because team leaders are expected to have the most complete picture of all work done by the team's members. Relying only on information from team leaders connotes that if a leader becomes damaged close to the end of an experimental run, the information it had would be lost assuming there was not enough time for another robot to take on the leadership role. I do not discard the information garnered by this robot and still gather statistical information from this damaged robot. This is the same approach that was used in the existing framework [Gunn, 2011; Nagy, 2016].

I track how much of the environment has been explored by merging map exploration data from all team leaders. I do this by finding the union of all area coverage reported by team all leaders. This is the same metric used in the existing framework ([Gunn, 2011; Nagy, 2016]). For a number of reasons, this metric is fairly conservative. Coverage data gathered by replacement robots are not used until they become part of a larger team; also, information acquired by individual robots may not be communicated to a team leader successfully before an experimental run ends.

To track the number of fires extinguished, I get the total number of fires extinguished by all FireBots within the environment. I examine how the value of this metric changes as environmental conditions (communication success rates, robot failures, etc.) change.

Successful stair lifts are tracked by counting the total number of times StairBots are able to carry other robots to a different level of the environment. This metric is important as stair lifts require coordination between two robots, and the success of this operation is dependent on the current environmental conditions. Also, since stairways are permanent fixtures in the environment, tracking this metric gives insight as to whether having robots stationed at such locations is more beneficial than having robots committed to moving with their teams.

5.4 Environment Generation

I used a tool developed by Wegner [2003] and modified by Gunn [2011] and Nagy [2016] to generate my world files. I made additional modifications to this tool to include fire objects, FireBots and StairBots. Aside from that, the environmental parameters used in generating a world were identical to those used in Nagy [2016]. All environments measure 60m $\times$ 60m, with about 15% of the environment covered by fire, debris and obstacles.

Ten negative and twenty positive victims are randomly placed in the environment to satisfy the following constraints: victims are at least 0.8m from a wall, and victims do not overlap each other. This approach is identical to that used by Gunn [2011] and Nagy [2016].

Robot team placement was done in a similar manner to what was done in the existing framework [Gunn, 2011; Nagy, 2016]. Two separate robot teams start out from openings on opposite ends of the environment, with one team at each opening. In configurations where replacement robots are used, they are released from equally spaced positions along the inner perimeter of the environment.

Before selecting my candidate environments to use for my experimental runs, I first generated 50 world files, after which I went through and hand-picked three environments which I was extremely confident satisfied my selection criteria. The criteria I used were the same as those used by Gunn [2011] and Nagy [2016] and are listed below:

- Any environments that had a great amount of removable debris and fire blocking the entrance used by initially deployed teams were discarded because it would require a lot of effort and time clearing these debris and extinguishing these fires. This would ultimately contribute to lowering the team performance since a majority of the time could be spent in trying to clear the entrance. However, environments that had very limited quantities of removable debris and fire were considered because DebrisBots and FireBots could clear the debris and extinguish the fire immediately after the mission started.

- Environments that had large obstacles blocking the entry paths of replacement robots were discarded to reduce wasted effort while the robot attempted to circumnavigate these obstacles.

- Any environments that had regions of highly clustered debris or obstacles were discarded since these were supposed to be reasonably distributed across the environment.

- I discarded any environments that had areas containing highly clustered victim populations to avoid any bias during victim confirmation and ensure that roughly the same amount of effort was required when locating and confirming victims in the environment.

The three world files I finally selected for my main experiment are shown in Appendix A.

5.5 Environment Set-up

Using the three environments selected, I designed a factorial experiment to evaluate the effectiveness of my implementation. I had to generate a world file (Stage's representation of an environment) for every experimental configuration because limitations of the Stage simulator required that experimental parameters (communication, success rate, probability of robot failure, etc.) be placed in world files.

Stage guarantees repeatability by using random seeds that are included in a world file. Thus, running any particular world file multiple times will generate the same set of results every time when given the same seed. This was a very useful feature when I had to rerun some experiments for observation purposes. Similar to the approach used by Nagy [2016], I used identical sets of 50 random seeds for each experimental configuration. This resulted in 50 world files for every combination of independent variables I considered. Using the same set of random seeds is to ensure that differences in team performances across different experimental runs would not be attributed to random numbers that inadvertently favoured some trials. All the trials were run on a *c5.9xlarge* Amazon EC2 instance [Amazon Web Services, Inc., 2019] which had 36 cores and 72GB of RAM using Amazon Web Services, with about 36 world files being run concurrently.

5.5.1 Independent Variables

All the independent variables except the *satisfaction configuration* variable used in my experiments are taken from Nagy [2016] to enable comparison with that work. The independent variables used in my experiments are described below:

Communication Reliablity : This is the probability that any wireless message sent will successfully be delivered to its intended recipients. Similar to previous frameworks [Gunn, 2011; Nagy, 2016], I use communication success rates of 100%, 60%, and 20%.

Recruitment Configuration : This defines what recruitment mode was used in the experiment. The different recruitment modes available are *active*, *concurrent*, and *passive* recruitment.

Replacement Robots : This controls whether replacement robots are used during a trial or not. When replacement robots are used, there are an additional 10 MinBots, 2 MidBots, 1 MaxBot, 1 DebrisBot, and 1 FireBot available as replacements. Replacement robots are released into the environment after five minutes of simulation have elapsed. Except for the FireBot, this setup is identical to that used in Nagy [2016].

Robot Failure This variable controls the likelihood that a certain robot type will experience some failure –temporary or permanent– during trials. Probability of failure is specified for each robot type used in my implementation and these are shown in Table 5.1. There are three different configuration failure levels used in my experiments: *no failures*, *minimal failures*, and *major failures*. These are the same levels used in Nagy [2016].

Satisfaction Configuration : This variable controls whether satisfaction tracking is turned on or off.

5.6 Additional Experiment

I conducted another experiment where I used a world containing two levels (shown in Appendix A) representing two storeys in the environment. This additional experiment was done particularly to study the effect of having robots with special skills becoming independent of all team commitment when their skill became rare. For a skill to be rare, it means there are not enough robots to satisfy the current demand of tasks requiring that particular skill. For the configuration where replacement robots were used, I substituted four replacement MinBots with two FireBots and two StairBots. Aside from this modification, all other parameters were the same as those used in the main experimental set-up. With no replacement robots used, there were two FireBots and StairBots each in the environment, and when replacement robots were used that number increased to five each. I did not run any trials with satisfaction tracking turned off, because robots are able to determine when a particular skill they have may be rare only when satisfaction tracking is turned on.

5.7 Main Experiment Results

This section provides an overview of the results obtained from running the experiments described in Section 5.5. Similar to Nagy [2016], my set-up was

level	model	prob. permanent failure	prob. temporary failure	avg. % total time failed
none	MinBot	0.000	0.000	**0.000%**
	MidBot	0.000	0.000	**0.000%**
	MaxBot	0.000	0.000	**0.000%**
	DebrisBot	0.000	0.000	**0.000%**
	FireBot	0.000	0.000	**0.000%**
minimal	MinBot	0.000	0.008	**14.9%**
	MidBot	0.000	0.006	**11.9%**
	MaxBot	0.000	0.004	**9.0%**
	DebrisBot	0.000	0.008	**14.9%**
	FireBot	0.000	0.008	**14.9%**
moderate	MinBot	0.002	0.014	**25.1%**
	MidBot	0.002	0.012	**20.9%**
	MaxBot	0.002	0.010	**18.5%**
	DebrisBot	0.002	0.014	**25.1%**
	FireBot	0.002	0.014	**25.1%**

Table 5.1: Robot types and the different configurations of failure probability.

such that statistics were garnered at the end of every trial run using the metrics described in Section 5.3. The results presented here show the impact that tracking robot satisfaction has on the different recruitment modes, as well as the impact of communication and robot failures, and the effect of using replacement robots. I use the configuration in which satisfaction tracking is turned off as the baseline to which I compare my implementation (when satisfaction tracking is used).

5.7.1 Victims Successfully Confirmed to Leaders

Figures 5.1 to 5.3 show the percentage of victims identified to team leaders using different recruitment modes with varying communication success rates, with satisfaction tracking turned on or off, with varying probabilities of robot failure, and with replacement or no replacement robots.

My implementation made significant performance gains against the baseline when using a passive mode of recruitment for configurations that had a communication success rate greater than 20% (Figure 5.1). Using a passive recruitment approach, teams are only able to recruit new members onto their team through chance encounters. Implying that, unless a member got lost, it basically remained on the same team until its team encountered a different team, triggering the merging and redistribution of skills between the two teams. By allowing robots to leave or switch teams when they realised they were not being productive, an opportunity was provided for these robots to explore other areas of the environment rather than expending energy remaining with their respective teams. When the communication success rate was at 100%, replacements robots were used and using a passive approach

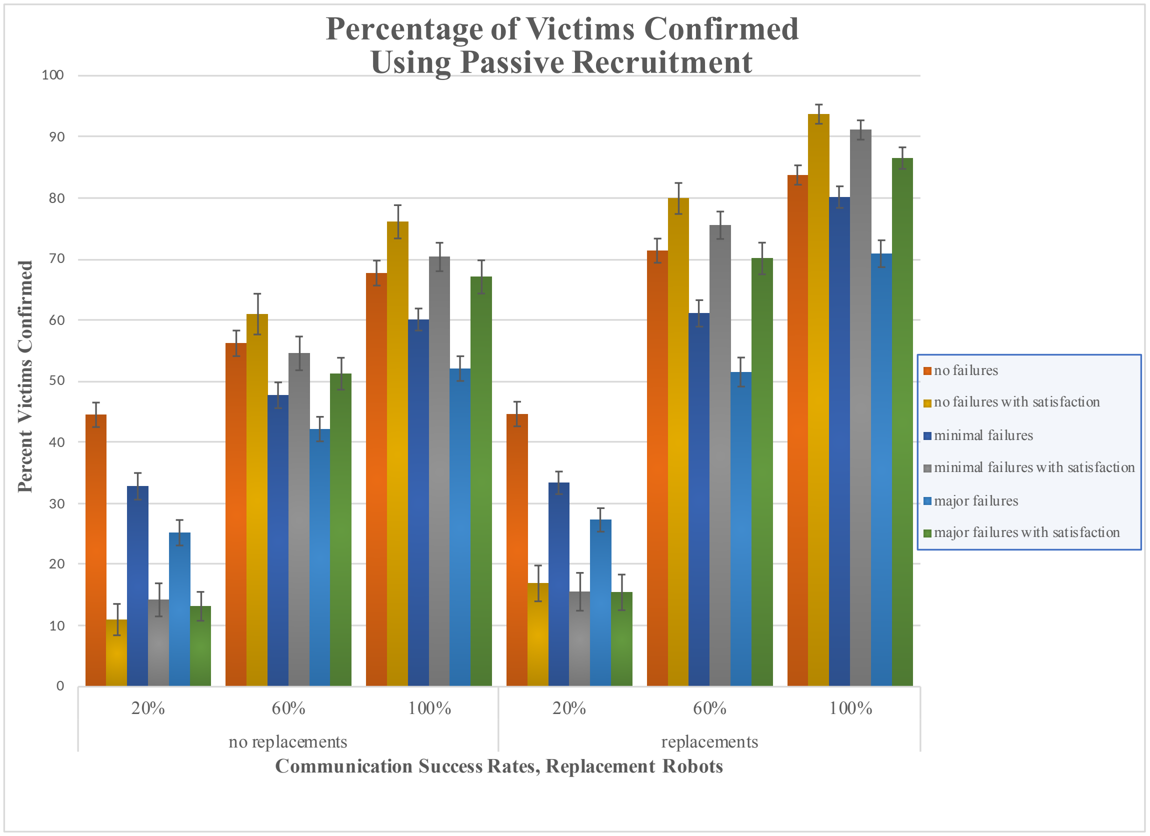

Figure 5.1: Percentage of victims identified using passive recruitment, with and without replacement robots

to recruitment, my framework performed almost similarly to the configuration that had the highest overall performance – active recruitment mode (Figure 5.3), with replacement robots and a communication success rate of 100%.

For all recruitment modes, and whether replacements were used or not (even though my implementation did not perform as well as or better than the baseline for most of the configurations), there were significant decreases in the performance gap when communication success rates increased above 20%. For example, when active recruitment was used with no replacements and for all failure configurations, the baseline performed about 11% better than my framework. However, when the

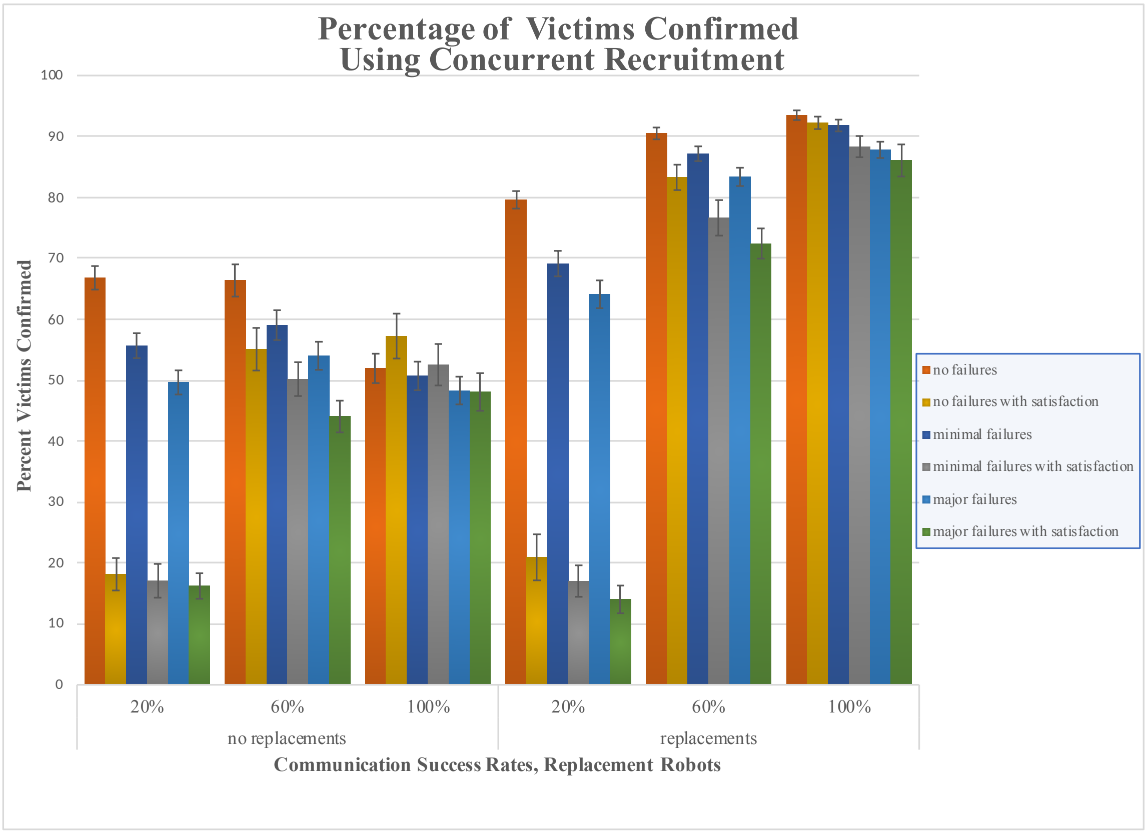

Figure 5.2: Percentage of victims identified using concurrent recruitment, with and without replacement robots

communication success rate increased to 100% for the same configuration, the baseline performed about 4% better than when satisfaction tracking was turned on.

For configurations that used a communication success rate of 60%, my framework's performance was significantly below the baseline when there were no replacements available. However, with the introduction of replacement robots, performance of my framework improved and the baseline performed only slightly better than mine. In configurations where communication success rates were at 100% and replacements robots were used in both active and concurrent modes of recruitment, my implementation performed similarly to that of the baseline. With satisfaction tracking turned on, there was a slight improvement over the baseline when concurrent recruitment was used with no replacement robots at a communication success rate of 100% (as shown in Figure 5.2).

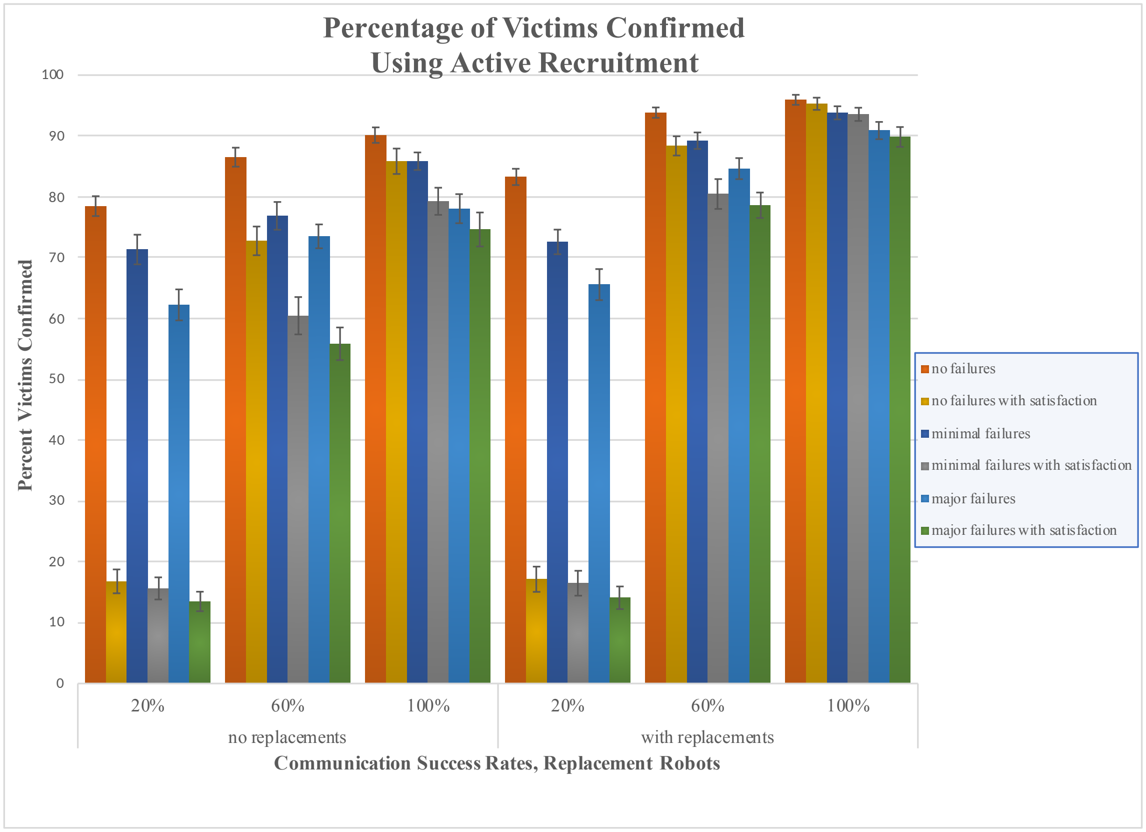

Figure 5.3: Percentage of victims identified using active recruitment, with and without replacement robots

My system performed poorly when compared to the baseline where communication success rates fell to 20% in all three recruitment strategies, as seen in Figures 5.1 to 5.3, whether replacement robots were used or not. This is just an indication that my implementation requires some minimum communication success rate before any reasonable performance is seen. I attribute this poor performance to the fact that when team leaders track their members satisfaction, they only rely on how many task assignments the members have accepted within a certain time frame to determine whether a particular member is useful or not to the team. Upon further investigation, I realised that leaders were almost never satisfied with their team members in the poorest communication category because they hardly ever heard back from them due to lost messages. This resulted in leaders letting go of team members often enough that it resulted in a complete breakdown of teams. Also, because of the poor communication reliability, it is likely that team members let go of by leaders do not receive the message from the leader terminating their membership and continue transmitting updates to the (former) leader.

5.7.2 Area Coverage

For the passive recruitment approach, which relies upon chance team encounters, my system (when satisfaction is used) performed slightly better than the baseline, with performance increasing as communication success rates increased (Figure 5.4). My approach provides avenues that allow a robot to actively wander off away from its team in search of greater satisfaction when its current satisfaction value falls below the threshold. This increases the robot's chances of exploring new areas in the environment and any information gathered while it was independent is passed along to a team leader when that robot is recruited to join a team.

For the active recruitment mode (Figure 5.6), except for communication success rates of 20%, the performance of the baseline and my implementation were similar for all configurations excluding when communication success rates were at 60% and no replacements were used. One possible reason for this is that using 30 minutes as

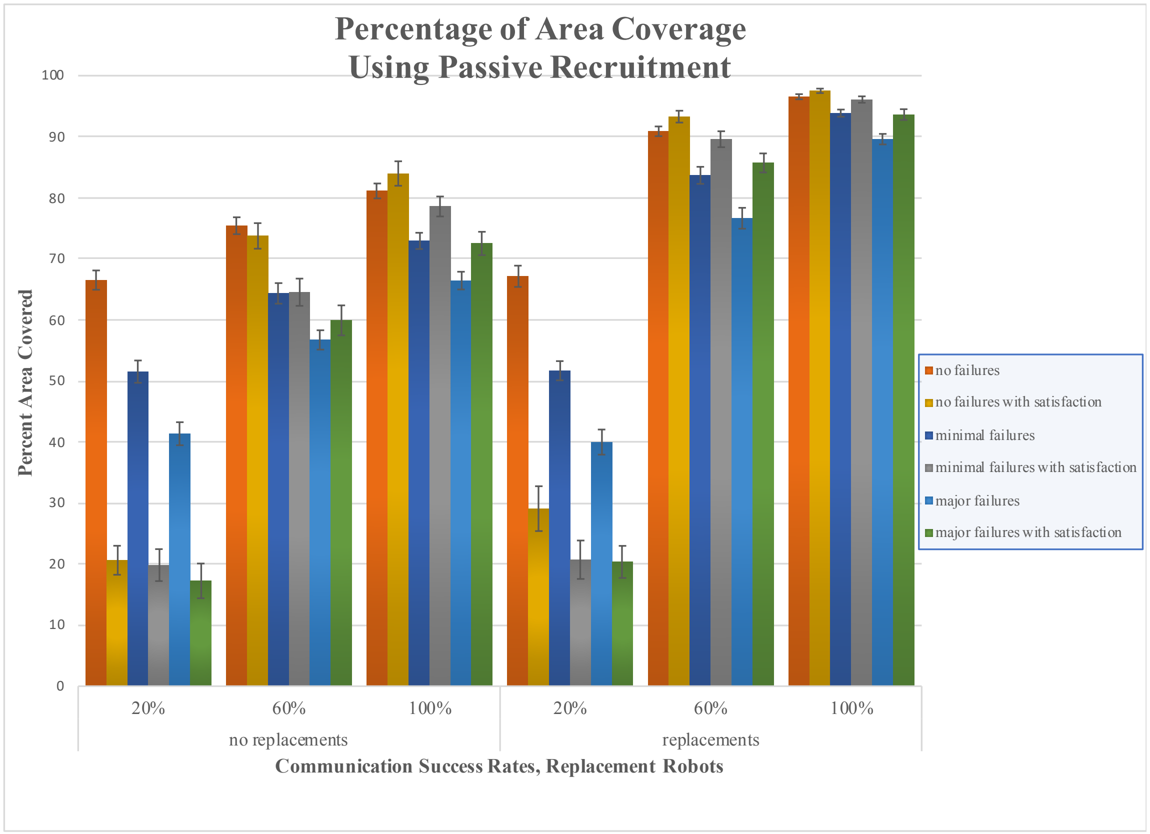

Figure 5.4: Percentage of area coverage using passive recruitment, with and without replacement robots

the simulation time is not sufficient for isolated robots that can not manage a team map to find a team they could join in order to transmit their exploration data to that team's leader.

When replacement robots were available, there were significant improvements in robot teams' performance. For the same communication success rates and whether replacements were used or not, active recruitment (Figure 5.6) performed better compared to other strategies followed by concurrent then passive. However for a 100% communication success rate and use of replacement robots, performances of concurrent and passive recruitment were almost identical.

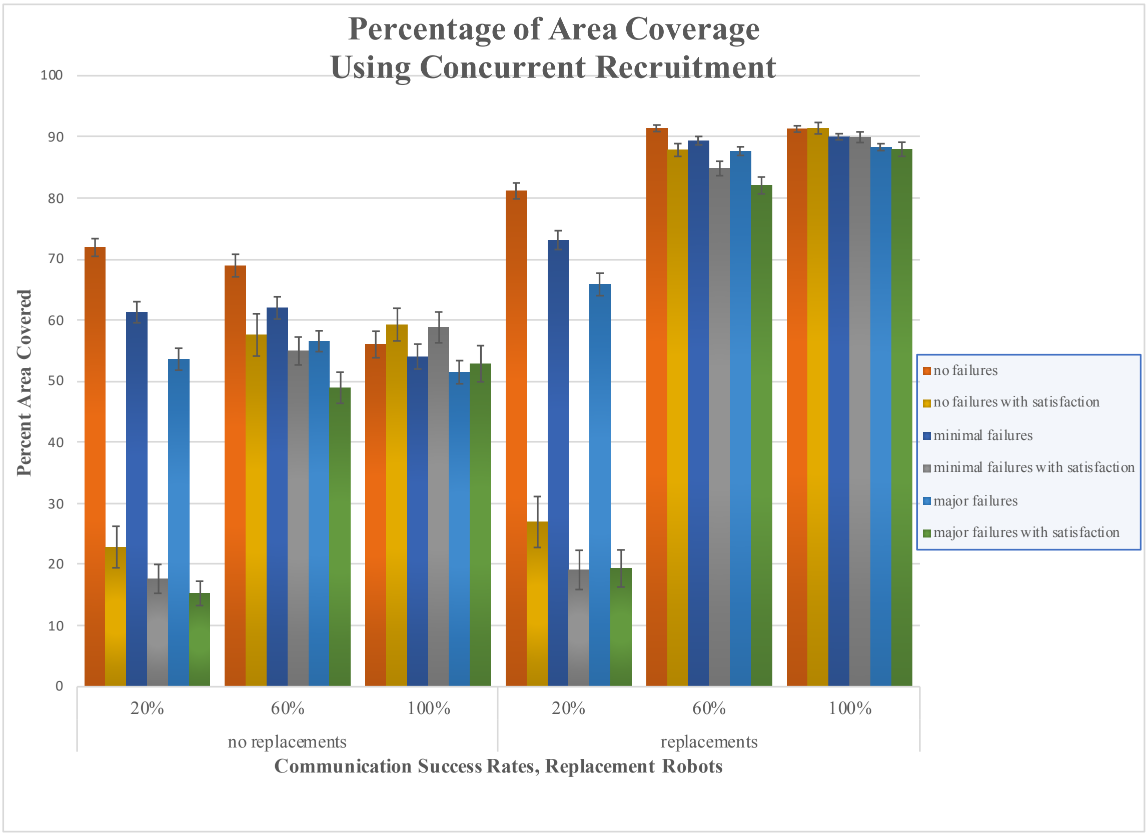

Figure 5.5: Percentage of area coverage using concurrent recruitment, with and without replacement robots

When no replacements were used, with communication rates of 60% and 100%, passive recruitment (see Figure 5.4) performed better than concurrent recruitment (Figure 5.5). This is likely due to the duplication of effort that occurs, as robots that are not leaders are able to recruit other members via task- or role-level recruitment (Section 2.4.2.2) without a leader's intervention. Since such members do not have a wider perspective of outstanding work, tasks can be redundantly assigned to multiple robots, resulting in wasted effort. This same phenomenon was observed and reported by Nagy [2016]. On the other hand, passive recruitment approaches rely on a single leader to make any decisions regarding task assignments and hence there is

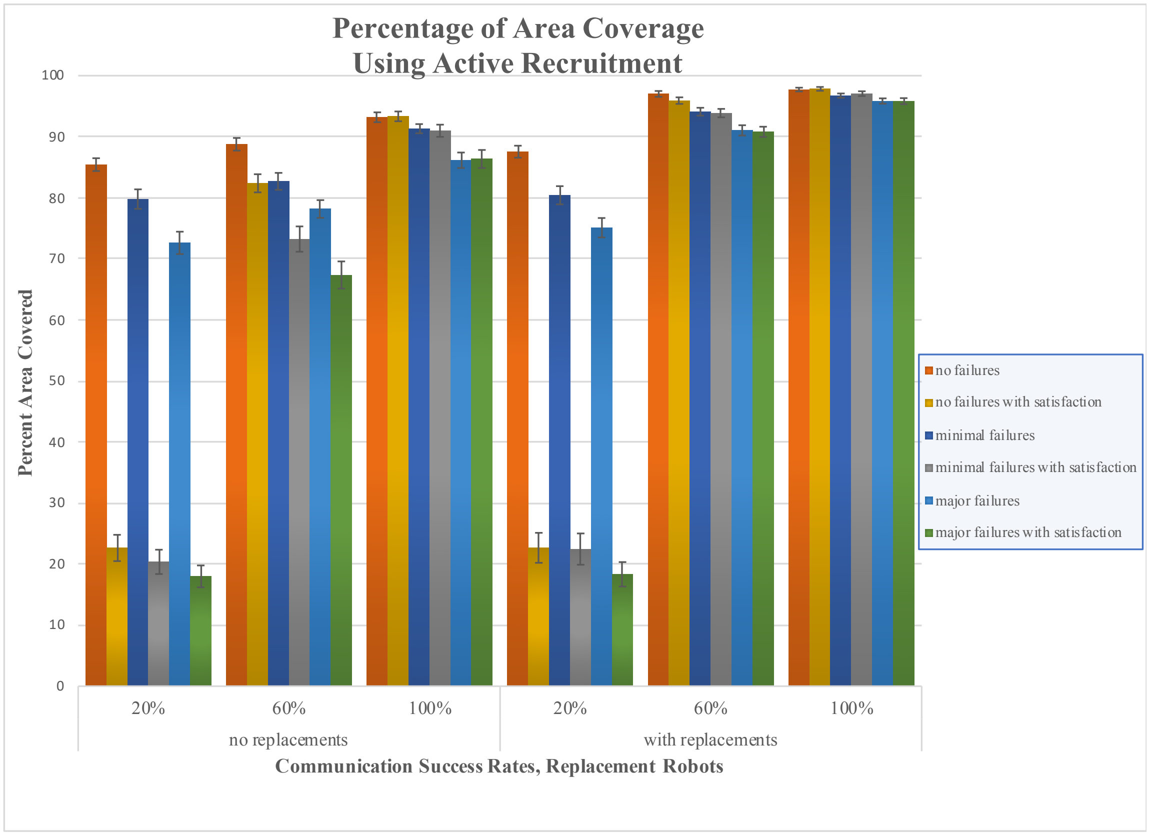

Figure 5.6: Percentage of area coverage using active recruitment, with and without replacement robots

a lesser chance of tasks being redundantly assigned. However, there is then greater vulnerability in losing a single leader

Similar to my results from Section 5.7.1, my system did not perform well when communication success rates were extremely low, as compared to the baseline using the same configurations. When communication success rates fall to 20%, not only does it become more difficult to communicate exploration results to a team leader, but also to manage a team (team rebalancing, receiving or accepting task assignments, etc.). A possible reason for this is that a leader relies on acceptance of task assignments by members to deem a member useful to the team. Due to the high rate of message delivery failure, it is likely that a leader that does not receive acknowledgement messages of a team member accepting a task, would determine that member is not being useful enough, and would withdraw that robot's membership; this would likely lead to many single-member teams. Since not all robot types are able to merge team maps (Section 4.4), using my conservative approach of merging maps of only leaders with the capability of maintaining a team map, a lot of information is lost. When communication success rates were above 20%, my system performed consistently better than when they were at 20% or less. A reason for this is because more messages relating to team management were successfully delivered, and as such, teams were better maintained.

5.8 Additional Experiment Results

As mentioned in Section 5.6, I performed additional experiments to help me investigate whether robots tracking their satisfaction were more inclined to stay at locations that would require the use of their specialised skills rather than continuing to travel with a team. These additional experiments involved monitoring how many stair lift requests StairBots successfully completed (shown in Figures 5.7 and 5.8) and how many fires were extinguished by FireBots (shown in Figures 5.9 and 5.10).

5.8.1 StairBots Results

When replacement robots were used (see Figure 5.8), the number of successful stair lifts was significantly reduced when compared to configurations that used no replacement robots (Figure 5.7). A possible reason for this dip in performance is that having replacement robots reduces the likelihood of skills becoming rare during the trials. Even though having more StairBots available should possibly have resulted in more stair lifts being performed, it is likely that StairBots were not stationed close enough to stairways and had to travel some distance to fulfil these requests. Since my implementation gives approximately 30 seconds for StairBots to execute stair lift requests, it is possible that a number of requests were abandoned because StairBots could not reach their destinations before the allotted time elapsed.

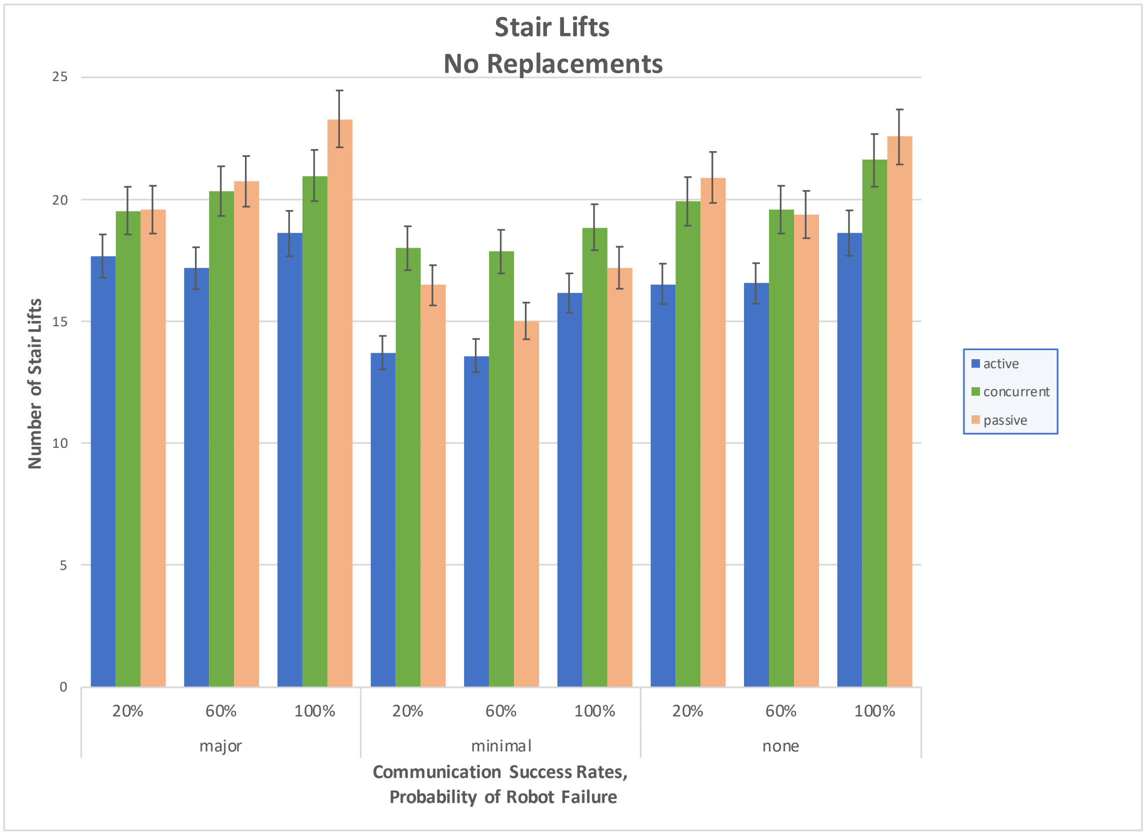

Figure 5.7: Number of stair lifts successfully completed when satisfaction tracking was turned on for all configurations without the use of replacement robots

Active recruitment strategies had the poorest performance compared to using passive or concurrent recruitment for the same configurations of probability of robot failure, communication success rate, and replacement robots (see Figures 5.7 and 5.8). This is likely because when using active recruitment, StairBots might perform physical searches that take them away from where stairways are located. Also, a StairBot navigating to a stairway in response to a stair lift request may be recruited along the way for more important tasks (for instance, finding a robot to confirm an actual victim). Even in the event it is not recruited, travelling some distance in a USAR domain increases the possibility of any robot becoming stuck on an obstacle.

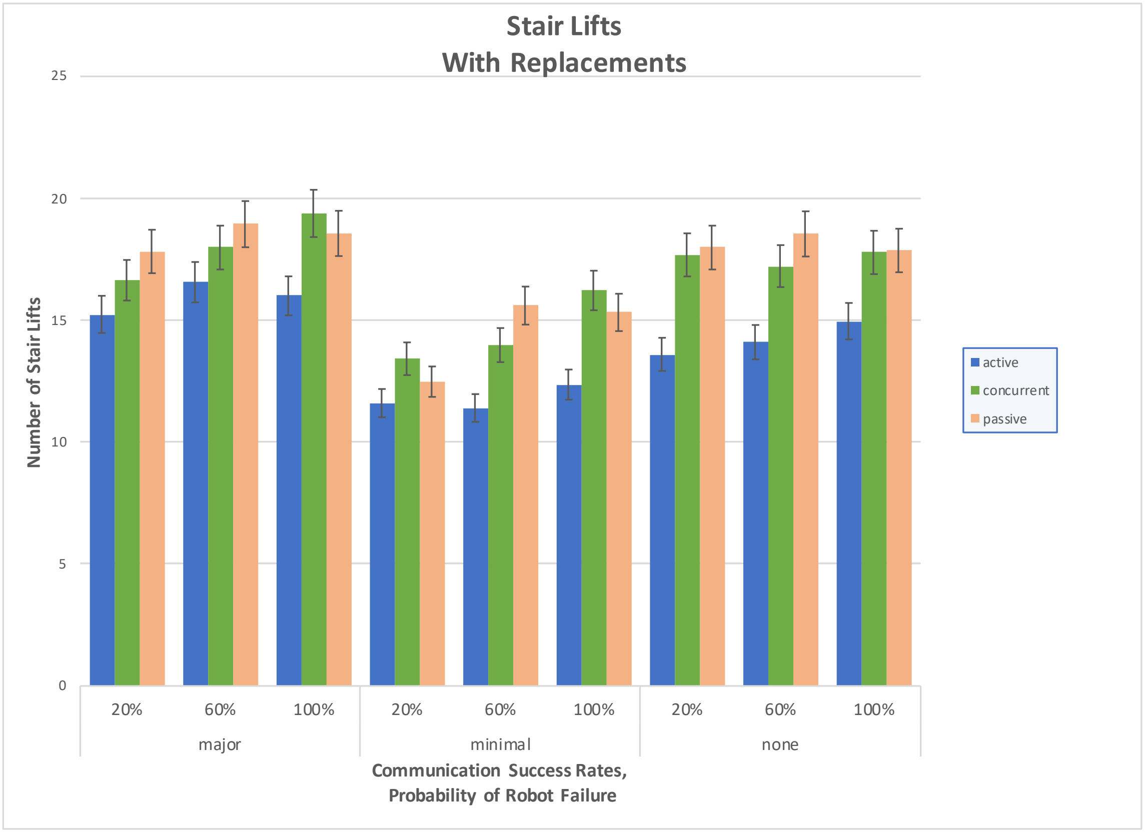

Figure 5.8: Number of stair lifts successfully completed with satisfaction tracking turned on for all configurations when replacement robots were used

Configurations that had a moderate amount of robot failures performed the worst when compared to the other robot failure probability configurations (refer to Figures

5.7 and 5.8). One reason I think this is so is that with minimal robot failures, at certain times during the experimental runs, there were fewer StairBots (compared to when there were no robot failures) available to respond to stair lift requests, but not quite to the extent that it resulted in the stair lifting ability becoming rare. In configurations where there was a high probability of robot failure, there were slightly higher number of stair lift requests fulfilled when compared to configurations where there were no robot failures. This is likely because having major robot failures could result in a skill being rare for most of a trial. Such a situation would result in StairBots being stationed at stairways till they were unsatisfied with their output at that location. Since a StairBot performing stair lifts results in it using its unique skill, it will be greatly rewarded with a very high satisfaction, which translates to a longer time spent at that location. This increases the probability of a StairBot being always stationed at a stairway and thus being able to respond to a majority of stair lift requests.

5.8.2 FireBots Results

Unlike the results seen in Section 5.8.1, configurations which had major robot failures performed the worst when dealing with FireBots, compared to the other configurations of robot failure probability. This is likely because fires are scattered throughout the environment and are not restricted to specific locations in the environment, so more robot failures translates to fewer opportunities to find fires and ultimately results in fewer fires being extinguished.

For the different recruitment configurations (Figures 5.9 and 5.10), the active recruitment mode resulted in the most fires being extinguished. A possible reason for this is that when using active modes of recruitment, robots perform a physical search for a robot to recruit. This increases the opportunity for FireBots to encounter fires and attempt to extinguish them.

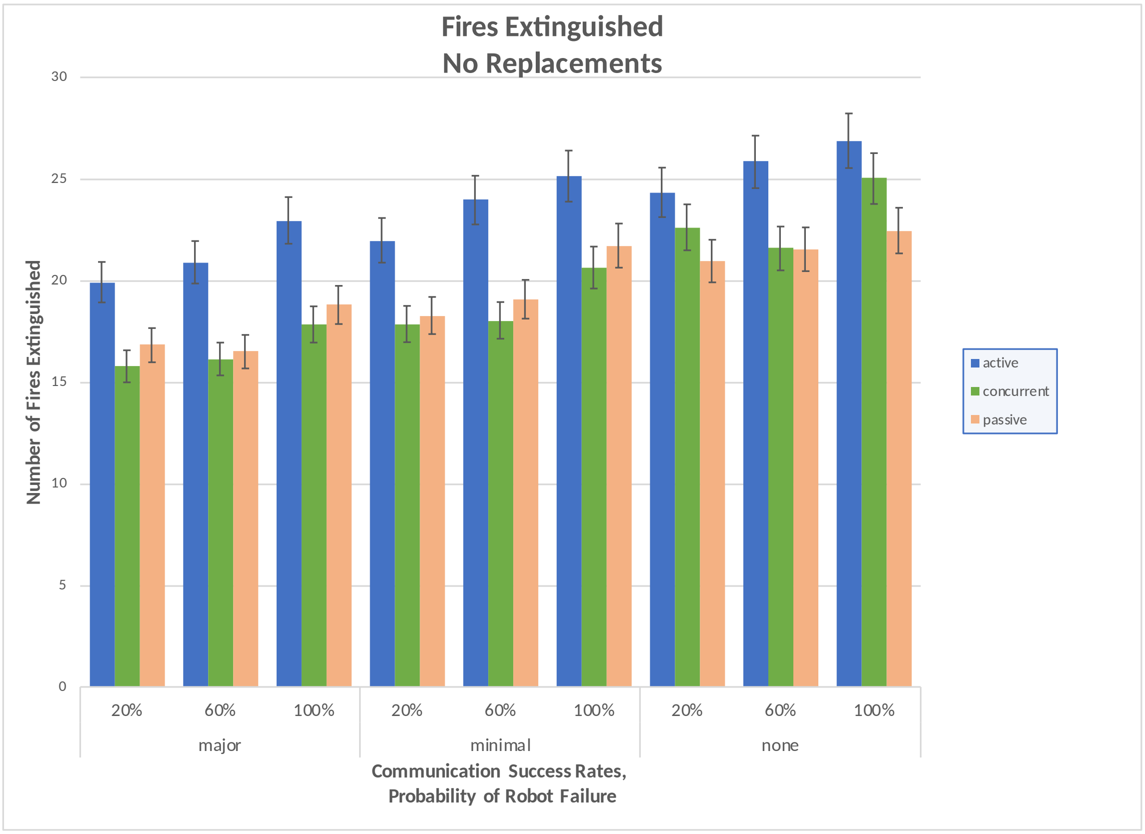

Figure 5.9: Number of fires successfully extinguished when satisfaction tracking was turned on for all configurations without the use of replacement robots

In configurations when replacement robots were used (Figure 5.10), FireBots extinguished more fires compared to the same configurations when no replacement robots were used (Figure 5.9). Having more FireBots in the environment increases the likelihood of these FireBots finding fires and extinguishing them, thus the increased performance when replacement robots are used over when replacements are not used.

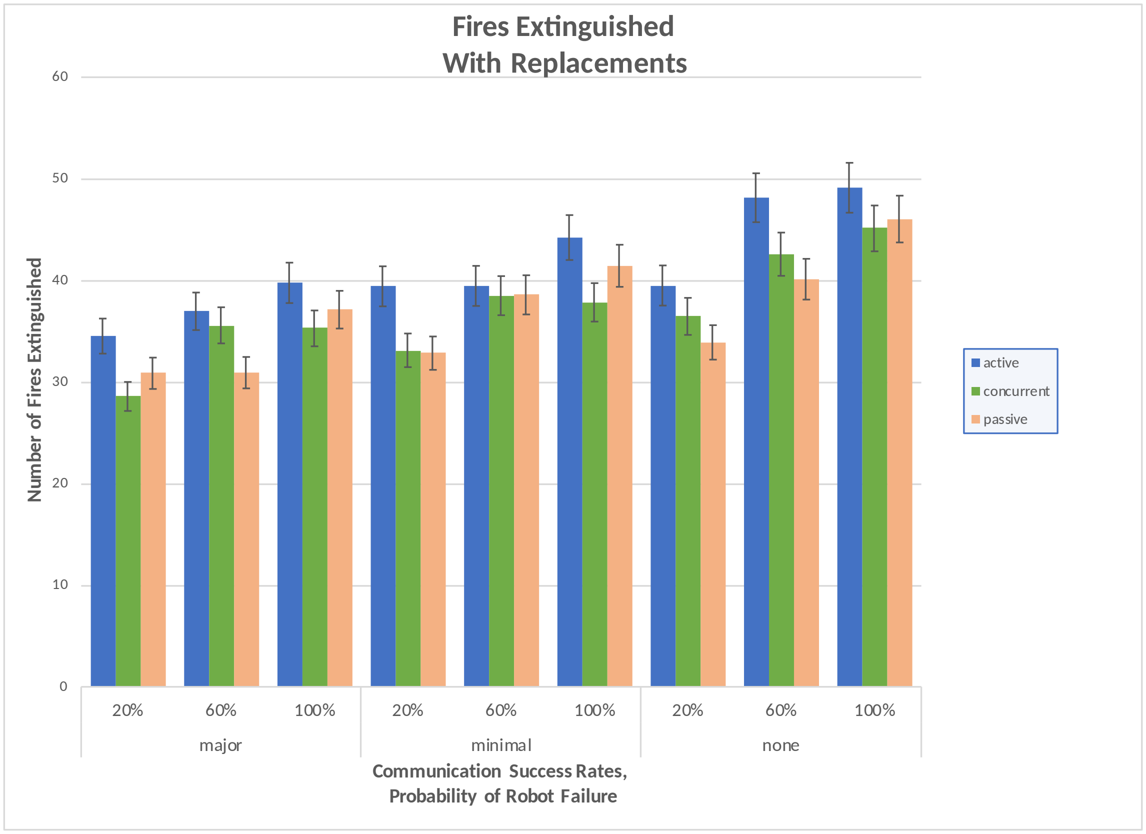

Figure 5.10: Number of fires successfully extinguished with satisfaction tracking turned on for all configurations when replacement robots were used

5.9 Analysis

From the results, it is seen that as communication rates increase, the performance of my system increases significantly. When there is a very high message delivery failure rate my system performed poorly and could lead to complete team breakdown. I attributed this to the fact that team leaders might have let go of certain team members who had not actually become aware their team membership had been withdrawn. One suggestion to improve the performance would be to also use task assignments that were rejected in determining whether team members are useful or not. This can be made possible by using learning techniques to find what weights should be assigned to parameters being used to estimate the satisfaction value. These weights would be based on the current environmental conditions.

The reason I think when using active recruitment, the baseline generally outperformed my system for the overall team mission is that by actively searching for robots to recruit for a task, it increases a robot's chances of exploring new areas of the environment. Though my approach sometimes results in a robot leaving its team, this only happens when a team leader is not satisfied with that robot's work or the robot is itself not happy with the contribution it is making towards the team's goals. This might not happen quite as often as active recruitment requests. Also, in my implementation robots are able to reject recruitment requests if the robot is considered to be useful to the team. In the previous framework [Nagy, 2016], the only time robots rejected recruitment requests was when their task queues were full.

For scenarios where a robot's skill was required at a permanent location within the environment (in this case a stairway), using an active approach to recruitment produced the worst results. I think this is so because if a robot happened to accept a recruitment request due to the fact that its satisfaction value was low, the robot would likely be pulled away from this location that required the use of its unique skill. Until a time when its satisfaction fell below the threshold or its unique skill had become rare, the robot would not be committed to satisfying tasks involving its special skill. This was seen in the case of the additional experiments conducted using the StairBots. For configurations when replacement robots were used, having no robot failures and having major robot failures performed almost similarly and both performed better than when there was a moderate amount of failures. This indicated that in environments that are unforgiving, it would be as beneficial having robots track their satisfaction as introducing replacement robots. With no failures and use of replacement robots, there is a low likelihood of a robot's skill becoming rare, because there will be enough robots available to satisfy any requests. When the need for a robot's skill was not restricted to a specific location and scattered all over the environment, performance was highest when replacement robots were used with an active recruitment mode, indicating that for tasks that occur all over the environment, tracking satisfaction did not provide any added benefits. For specialised tasks that require robots being stationed somewhat permanently at a certain location in the environment, robots that will be expected to perform such tasks should not have an active mode of recruitment enabled. The framework as it currently exists does not support using multiple recruitment strategies in a single trial and future work could add such a functionality.

5.10 Conclusion

In this chapter, I have presented and discussed the performance of my methodology. I discussed the metrics I used to evaluate my framework as well as discussed my results and finally pointed out the strength and weaknesses of my approach.

Chapter 6

Conclusion

6.1 Chapter Overview

In this chapter, I begin with a review of my research questions and summarise how the results I described in Chapter 5 answer them. I present the main contributions of my thesis in Section 6.3, and follow this with a discussion of directions for future work in Section 6.4.

6.2 Answers to Research Questions

In this section, I review my research questions which I first posed in Section 1.5, and discuss how the results I obtained from my experiments answer these questions.

1. **To what degree does tracking robot satisfaction affect the total teams' performance with respect to the overall mission in a USAR domain?**

   From my results, significant performance gains are seen when satisfaction tracking is used alongside passive recruitment approaches as opposed to when it is not used. By tracking satisfaction, robots can leave teams or be let go by leaders with the aim of improving their satisfaction. Passive recruitment approaches do not provide such opportunities outside of encountering other teams or team members wandering far off from their teams and becoming lost. Hence, robots spend a significant amount of effort trying to stay with their teams. No significant improvements are seen though when satisfaction tracking is used in conjunction with concurrent or active recruitment strategies. This is because these approaches already provide adequate opportunities for robots to leave their respective teams via role- or task-level recruitment (Section 2.4.2.2), reducing the need for consideration of satisfaction. With communication success rates at 20%, using satisfaction tracking resulted in a complete breakdown of teams, in turn leading to low performance. This illustrates that my implementation needs a certain minimum message transmission success (somewhere between 20% and 60%) in order to perform reasonably. A future experiment could examine what a minimal communication success rate would be more precisely.

2. **For specialised tasks that require the use of a robot's unique skill, is it more useful to have robots expending effort to stay with a team or should these efforts be redirected towards areas where its special skills may be needed?**

   As conditions in the environment deteriorate (such that robots become more easily damaged, leading to a possible high demand of certain skills), allowing robots to avail themselves to be used by all teams proved very useful when the skills these robots possessed were required at certain fixed locations in the environment. For tasks that were scattered over the environment, there was no significant gain in terms of how many tasks a robot used its specialised skill to perform. This is thus greatly dependent on the nature of the environment. Had the test environment been a larger number of storeys, we likely would have seen a stronger advantage to independent StairBots, for example. Many public buildings are organised around one or more central stairwells, making this applicable to the real world. Once again, a future experiment could examine the effect of the number of teams and the number of levels in a larger structure.

6.3 Contributions

This section summarises the key contributions made by my research. These are:

1. A methodology by which robots are able to estimate how useful they are to their teams and take appropriate action when they determine that they are not being useful on their current teams.

2. A methodology by which robots are able to identify when specialised skills they possess may be rare and allow them to suspend or re-rank team commitments during that period.

3. An implementation of my methodology that shows the potential benefits of my framework in a simulated multilevel USAR domain.

4. A framework which can be further improved upon by others for future research, including other areas of team-based USAR.

6.4 Future work

The evaluation of my framework revealed some areas where improvements could be made to further contribute to this field.

6.4.1 Future Implementation Suggestions

First of all, it would be insightful to have my work implemented on real robots. My work in simulation made a number of simplifications to facilitate my research, but implementation on physical robots in the real world would require more elaborate (and expensive) solutions for these. This would raise additional operational challenges that were not seen during simulation pertaining to localisation, errors in sensor data, debris removal, etc.

Another improvement that can be made is in the area of simulation realism. In my implementation, robots are able to perform somewhat complex computations in a single simulation time step, as all robot types use the processing power of the computer implementing the simulation. I believe the simulation could be made more realistic by imposing memory and computational restrictions on the different robot types. This would mean optimising and adjusting algorithms for the different robot types based on their memory and processing power.

An area that could also be improved upon is that of robot perception. To facilitate work on my research, I used Stage's in-built fiducial sensors to help easily identify certain objects (fire, stairways, etc.) within the environment. This is not as trivial a task for robots in the real world, especially under disaster conditions. Implementing my system on physical robots would require that such abstracted perceptors be replaced with an appropriate implementation of robot perception. For example the advanced fire detectors would need a minimum of three sensory inputs (heat, smoke, and vision) to be able to actually confirm the presence of a fire.

The Stage simulator as it currently exists does not offer a lot of object dynamics and also does not provide a rich level of object interactions such as pushing or carrying objects. Another shortfall of the Stage simulator is that as a 2.5D simulator it does not offer in-built support for multilevel environments, and to be able to simulate multi-storeys in my work, I had to use two world files side-by-side. Porting my work to a simulator such as Gazebo [Koenig and Howard, 2004] which offers more complex physics could be used to evaluate my work more rigorously.

6.4.2 Suggestions for Future Research

The results of my experiments in Section 5.9 raised some questions with regards to robots being able to estimate their utility on a team and also being able to identify when there was a global shortage of a robot's special skills. In this section, I discuss directions for future research as revealed by my work.

In my implementation, there is no granularity in terms of robot failure. That is, a robot is either functional or not. The binary nature of representing robots in this regard indicates that a robot can be of no use when damaged. This however, may not be so in the real world, as robots could have certain individual components fail yet remain useful in other ways. For example, a FireBot that has its extinguisher damaged could end up performing tasks which are only associated with MinBots. Even though a robot identifies as a certain robot type and would be assumed to be able to carry out a certain task based on its type, extensions could be made to my work such that a robot before accepting a job would check to see if it possesses the functional components needed to be able to carry out that task.

The ability for robots to determine a risk factor associated with performing a certain job would also be a very useful addition to my methodology. Since some robot types are very specialised in my work, having these robots perform tasks that could potentially damage them would impact perfomance assuming these robots get damaged executing a task (especially in scenarios where special skills they possess may be rare). Though the use of replacement robots curtails this to some extent, I think it would be interesting to have the satisfaction expression consider the risk factor associated with performing a task as well as how rare that robot's skill is within the environment. Given that this is a time-constrained emergency situation, there is also the question of balancing these types of computations with actively getting work done in the environment.

In my implementation, I assume that all robots' satisfaction values are on the same scale. However, this might not be the case, as robots' suitabilities vary and I use a robot's suitability for a task in my satisfaction expression (Section 4.8.1.1). I therefore think it would be useful to have robots use learning techniques to come up with their own scale for determining whether they are satisfied with the work they are doing.

Ideal teams in my work are defined at the onset of the mission by the researcher and stay the same over the course of a trial. Generally all robot teams in the framework are always looking to have their membership be as close to the ideal team definition as possible. However, as the environment changes, maintaining the same ideal team definition might not be useful enough. That is, the concept of an ideal team may change over time. Having teams refine the ideal team definition over the duration of the mission based on what tasks are most likely encountered and what losses are likely to occur might prove useful. It would be a challenge focusing on the as-yet-unexplored parts of the environment, rather than tailoring an ideal team for work that is already done and which may not be seen again.

A limitation of my work is that robots are not able to select their own individual configurations based on information they gather from the environment. Whatever configuration is used applies to all robots during the trial (for instance if active recruitment is turned on, all robots in that trial run would employ active recruitment approaches). However, from my results, certain configurations used with certain robot types resulted in poor performance or did not significantly improve performance. For example, from Section 5.8.1, StairBots under passive recruitment approaches performed significantly better than under active recruitment approaches when satisfaction tracking was turned on, indicating that StairBots should mostly use passive recruitment strategies (or possibly operate independently). Learning techniques could be used here to help robots identify what configuration to use depending on the current conditions of the environment. Future research could explore the possibility of having robots set their own configurations during a mission and examine the effect this has on overall team performance.

6.5 Conclusion

This research has shown some benefits associated with having robots estimate their utility within a team setting and taking appropriate actions to increase their contributions when they realise they are not useful enough. Also, my research demonstrated the utility of having members put all team commitments on hold when they determine there is a possibility that certain special skills they possess are in high demand globally and make themselves available to be used by all teams within the environment. Even though my contributions are significant, my research has uncovered avenues of future work which I think would enhance my work further.

Appendix A

Experimental Environments

Figures A.1 to A.3 shown below are the environments within which I conducted my main experiments (Section 5.5).

Figure A.4 shows the multi-storey environment I used in conducting my additional experiments described in Section 5.6

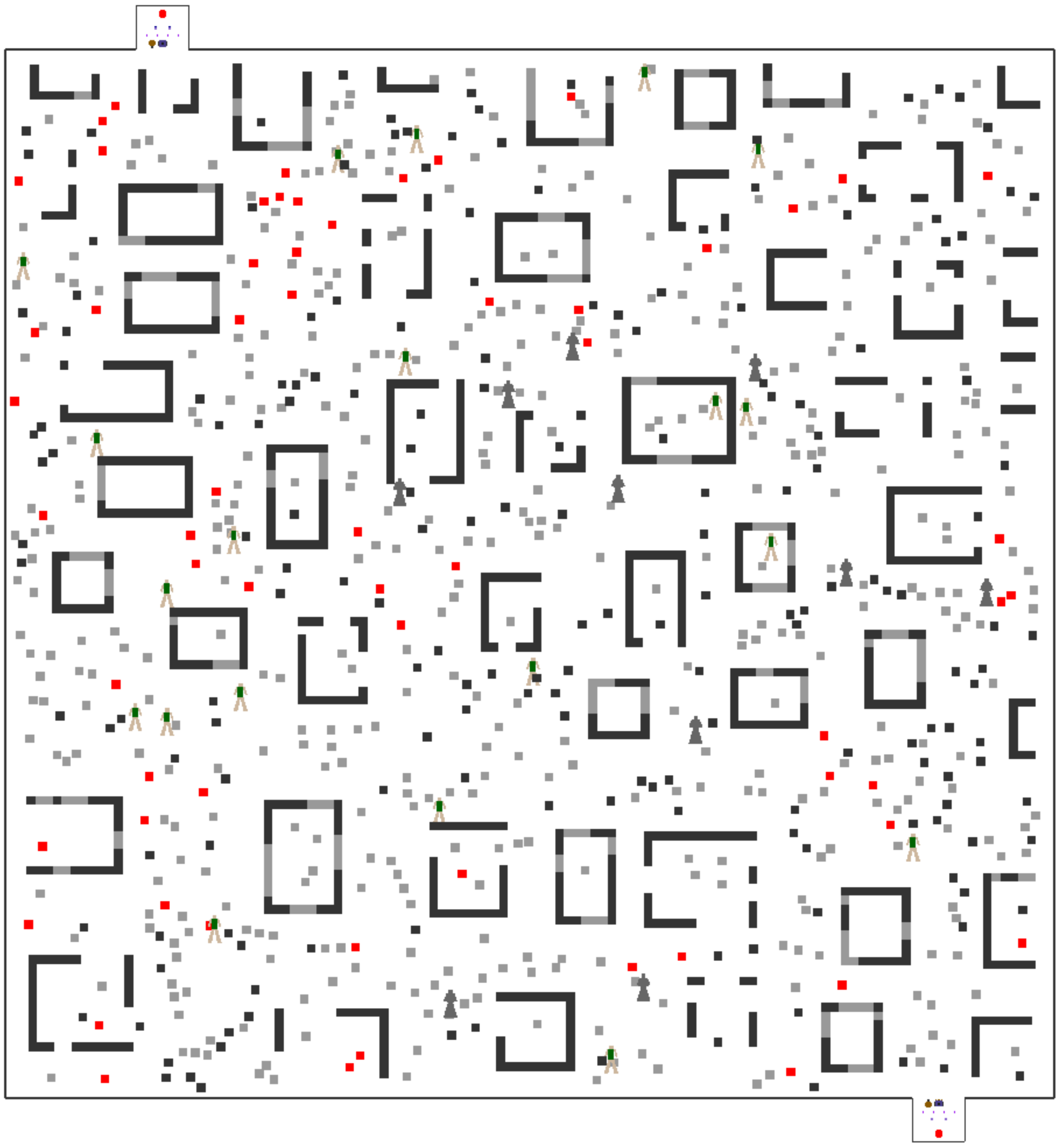

Figure A.1: Experimental environment 1

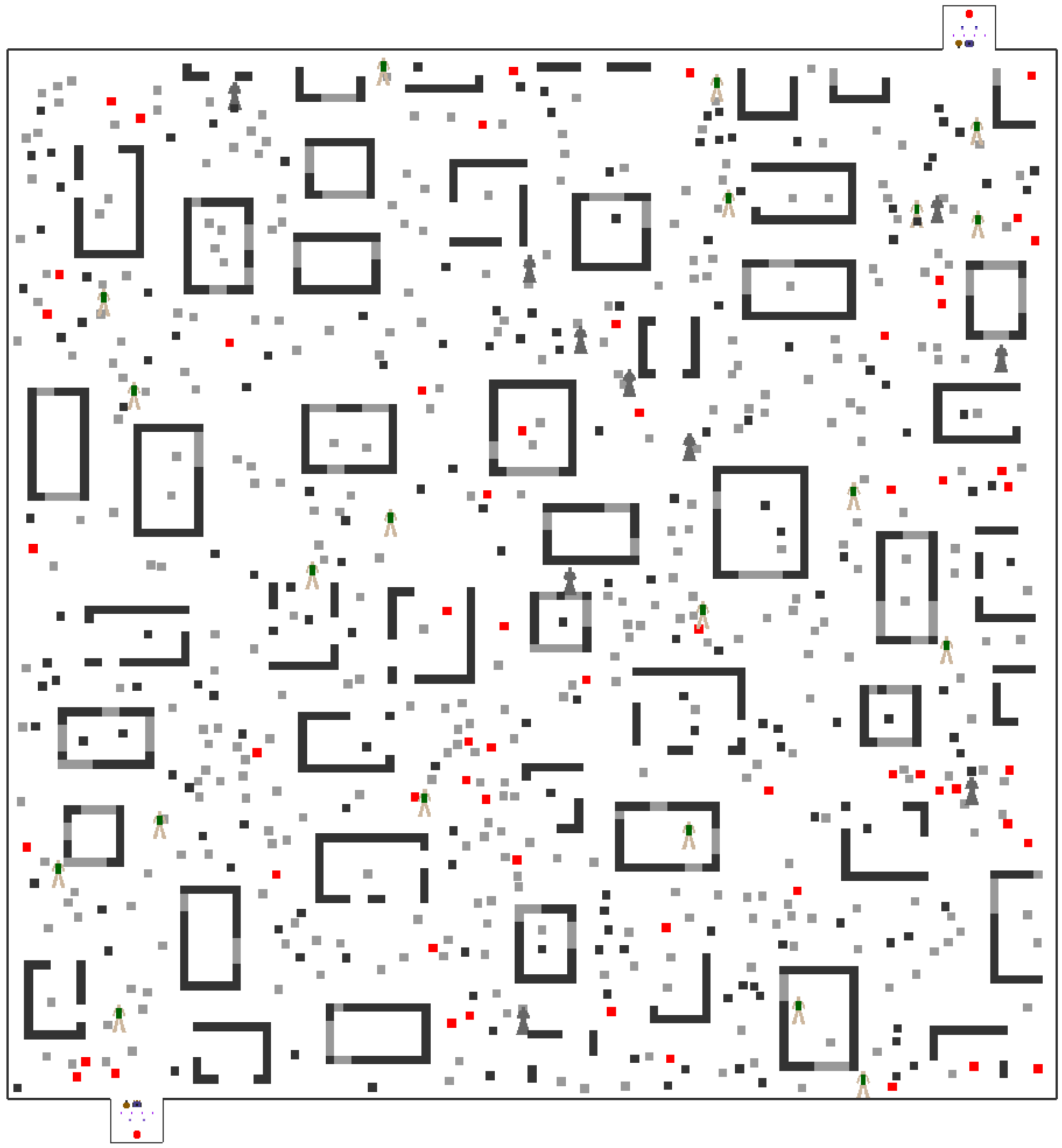

Figure A.2: Experimental environment 2

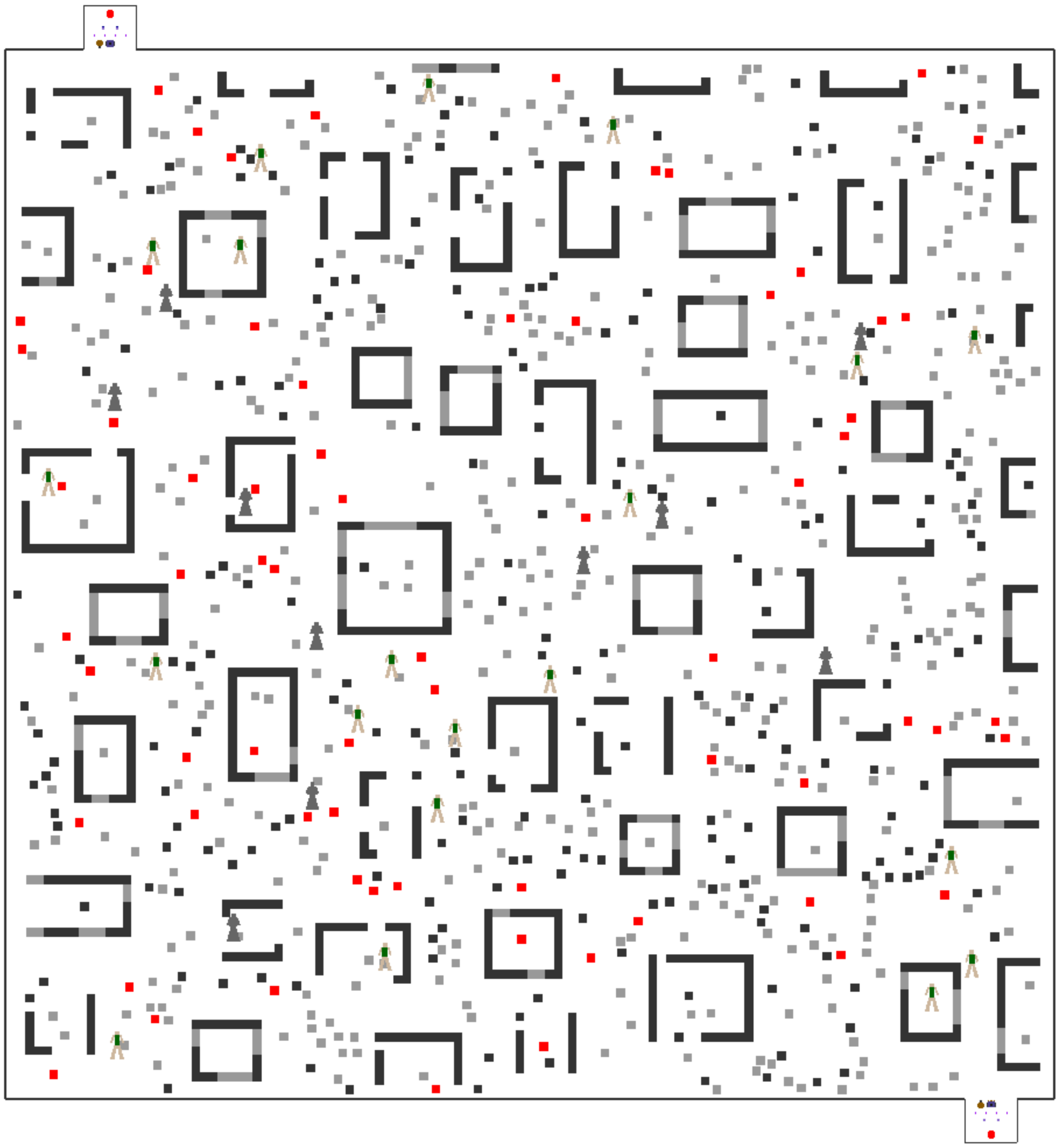

Figure A.3: Experimental environment 2

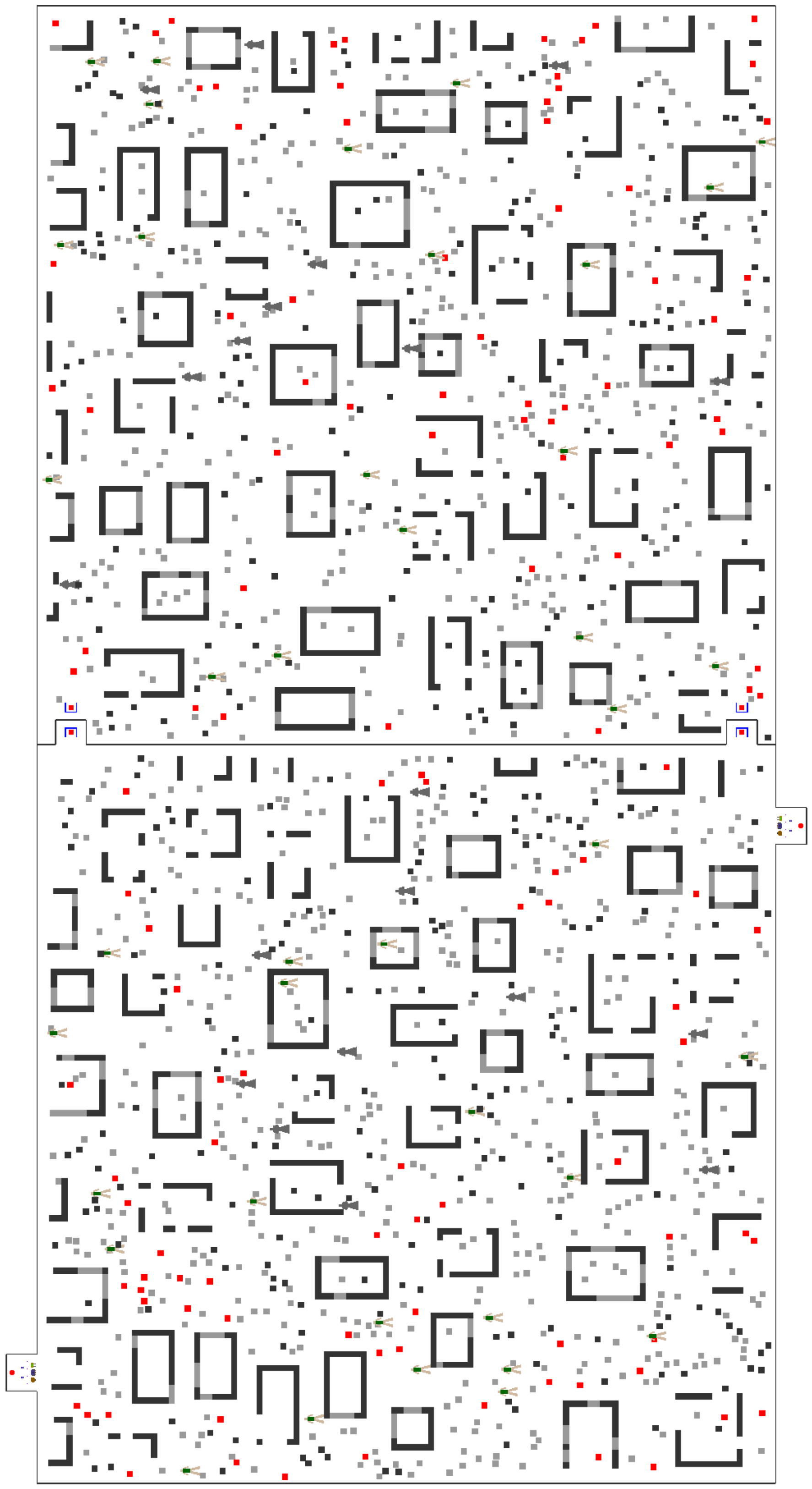

Figure A.4: Experimental environment with multiple storeys

Bibliography

Amazon Web Services, Inc. Amazon EC2 Instance Types - amazon web services, 2019. URL `https://aws.amazon.com/ec2/instance-types/`.

M. Anderson and N. Papanikolopoulos. Implicit cooperation strategies for multi-robot search of unknown areas. *Journal of Intelligent and Robotic Systems*, 53(4):381–397, 2008. ISSN 0921-0296. doi: 10.1007/s10846-008-9242-5. URL `http://dx.doi.org/10.1007/s10846-008-9242-5`.

R. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, pages 264–271, March 1987. doi: 10.1109/ROBOT.1987.1088037.

R. Arkin. Reactive robotic systems. In *The handbook of brain theory and neural networks*, pages 793–796. MIT press, Cambridge MA, 1995.

R. Arkin and G. Berkey. *Robot Colonies*. Springer US, 1997. ISBN 978-1-4757-6451-2.

T. Balch and R. C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

L. Coviello and M. Franceschetti. Distributed team formation in multi-agent systems: Stability and approximation. In *Proceedings of the 51st IEEE Conference on Decision and Control (CDC 2012)*, pages 2755–2760, Maui, HI, December 2012. doi: 10.1109/CDC.2012.6426198.

P. Dasgupta and K. Cheng. Dynamic multi-robot team reconfiguration using weighted voting games. *Journal of Experimental & Theoretical Artificial Intelligence*, 28(4): 607–628, 2016. doi: 10.1080/0952813X.2015.1020575. URL `https://doi.org/10.1080/0952813X.2015.1020575`.

F. De Rango, N. Palmieri, X.-S. Yang, and S. Marano. Swarm robotics in wireless distributed protocol design for coordinating robots involved in cooperative tasks. *Soft Computing*, 22(13):4251–4266, Jul 2018. ISSN 1433-7479. doi: 10.1007/s00500-017-2819-9. URL `https://doi.org/10.1007/s00500-017-2819-9`.

P. S. Dutta and S. Sen. Forming stable partnerships. *Cognitive Systems Research*, 4 (3):211–221, 2003.

E. Elkind, L. A. Goldberg, P. Goldberg, and M. Wooldridge. Computational complexity of weighted threshold games. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*, volume 22, page 718. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

J. Fredslund and M. J. Mataric. A general algorithm for robot formations using local sensing and minimal communication. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 2002.

A. Gage, R. Murphy, K. Valavanis, and M. Long. Affective task allocation for distributed multi-robot teams. Technical Report CRASAR-TR2004-26, Center for Robot Assisted Search and Rescue, University of South Florida, 2004.

J. Guerrero and G. Oliver. Multi-robot coalition formation in real-time scenarios. *Robotics and Autonomous Systems*, 60(10):1295 – 1307, 2012. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/j.robot.2012.06.004. URL `http://www.sciencedirect.com/science/article/pii/S0921889012000942`.

T. Gunn. Dynamic heterogeneous team formation for robotic urban search and rescue. Master's thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, December 2011.

T. Gunn and J. Anderson. Effective task allocation for evolving multi-robot teams in dangerous environments. In *Proceedings of the 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technologies (IAT-2013)*, pages 231–238, Atlanta, GA, November 2013. doi: 10.1109/WI-IAT.2013.114.

T. Gunn and J. Anderson. Dynamic heterogeneous team formation for robotic urban search and rescue. *Journal of Computer and System Sciences*, 81(3):553–567, 2015. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/j.jcss.2014.11.009. URL `http://www.sciencedirect.com/science/article/pii/S0022000014001500`.

J. Guzzi, A. Giusti, L. M. Gambardella, and G. A. Di Caro. A model of artificial emotions for behavior-modulation and implicit coordination in multi-robot systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 21–28, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5618-3.

doi: 10.1145/3205455.3205650. URL `http://doi.acm.org.uml.idm.oclc.org/10.1145/3205455.3205650`.

C. C. Kemp, A. Edsinger, and E. Torres-Jara. Challenges for robot manipulation in human environments [grand challenges of robotics]. *Robotics Automation Magazine, IEEE*, 14(1):20–29, March 2007. ISSN 1070-9932. doi: 10.1109/MRA.2007.339604.

J. Kiener and O. von Stryk. Cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007 (IROS).*, pages 959–964, San Diego, CA, October 2007. doi: 10.1109/IROS.2007.4399291.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, Sep. 2004. doi: 10.1109/IROS.2004.1389727.

M. J. Krieger and J.-B. Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30 (1):65–84, 2000.

M. J. B. Krieger, J.-B. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406(6799):992–995, 2000.

R. R. Murphy. Marsupial and shape-shifting robots for urban search and rescue. *Intelligent Systems and their Applications, IEEE*, 15(2):14–19, Mar 2000. ISSN 1094-7167. doi: 10.1109/5254.850822.

G. Nagy. Active recruitment in dynamic teams of heterogeneous robots. Master's thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, October 2016.

G. Nagy and J. Anderson. Active recruitment mechanisms for heterogeneous robot teams in dangerous environments. In *Proceedings of the 29th Canadian AI Conference*, 2016.

L. E. Parker and B. Kannan. Adaptive causal models for fault diagnosis and recovery in multi-robot teams. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2703–2710, Beijing, China, October 2006. doi: 10.1109/IROS.2006.281993.

L. E. Parker and F. Tang. Building multirobot coalitions through automated task solution synthesis. *Proceedings of the IEEE, special issue on Multi-Robot Systems*, 94(7):1289–1305, July 2006. ISSN 0018-9219. doi: 10.1109/JPROC.2006.876933.

L. Pitonakova, R. Crowder, and S. Bullock. Understanding the role of recruitment in collective robot foraging. 2014. URL `http://eprints.soton.ac.uk/364829/`.

J. G. Rogers III, C. Nieto-Granda, and H. I. Christensen. Coordination strategies for multi-robot exploration and mapping. In *Experimental Robotics*, volume 88 of *Springer Tracts in Advanced Robotics*, pages 231–243. Springer International Publishing, 2013. ISBN 978-3-319-00064-0. doi: 10.1007/978-3-319-00065-7_17. URL `http://dx.doi.org/10.1007/978-3-319-00065-7_17`.

O. Simonin and J. Ferber. Modeling self satisfaction and altruism to handle action selection and reactive cooperation. In *Proceedings of the 6th International Conference On the Simulation Of Adaptive Behavior (SAB 2000)*, pages 314–323, Cambridge, MA, 2000.

M. van de Vijsel and J. Anderson. Coalition formation in multi-agent systems under real-world conditions. In *Proceedings of the AAAI-04 Workshop on Forming and Maintaining Coalitions and Teams in Adaptive Multiagent Systems*, San Jose, CA, July 2004.

R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2): 189–208, 2008. ISSN 1935-3820. doi: 10.1007/s11721-008-0014-4. URL `http://dx.doi.org/10.1007/s11721-008-0014-4`.

R. Wegner. Balancing robotic teleoperation and autonomy in a complex and dynamic environment. Master's thesis, Department of Computer Science, University of Manitoba, July 2003.