Dynamic Heterogeneous Team Formation for Robotic Urban Search and Rescue

by

Tyler J. Gunn

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
December 2011

Thesis advisor

John E. Anderson

Author

Tyler J. Gunn

Dynamic Heterogeneous Team Formation for Robotic Urban Search and Rescue

Abstract

Using teams of robots to complete a task provides a number of advantages over the use of a single robot. Multiple robots are able to complete tasks faster, and provide redundancy in case of equipment failure or loss. Teams of robots with different capabilities and physiologies are beneficial because they allow a team to provide a high level of overall functionality while striking a balance between the cost and complexity of the robots. Previous work tends to focus on the use of pre-formed teams of robots, with little attention to the formation and maintenance of the team itself. An environment such as a disaster zone presents numerous challenges to robotic operation, and it can be expected that the nature of a team will change due to, for example, malfunctions and the introduction of replacement equipment. I developed a framework to support the maintenance of teams of heterogeneous robots operating in complex and dynamic environments such as disaster zones. Given an established team, my work also facilitates the discovery of work to be done during the team's mission and its subsequent assignment to members of the team in a distributed fashion. I evaluated my framework through the development of an example implementation where robots perform exploration in order to locate victims in a simulated disaster environment.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

I would like to begin by thanking my advisor, Dr. John Anderson, for guiding me through this process. Your guidance enabled me to keep focused and on track throughout the various steps of the way, and your reviewing skills helped me produce a thesis I am truly proud of. I would also like to thank my committee, who ultimately had to review this thesis so close to the holiday season.

Finally, I'd like to thank my friends and family, who have put up with my often lackluster ability to manage time between the various competing priorities and responsibilities encountered during the process of writing this thesis.

This thesis is dedicated to my wife, Angela, whose continual encouragement and support made it possible to dedicate the time and energy required to make this thesis a success. Angela, you never once questioned my ability to make it through this journey, and for that I am extremely grateful.

This thesis is also dedicated to my daughter, Erica, who always put a smile on my face as she sat enthralled watching "Daddy's robots" wander around the computer screen, nor did she ever question why Daddy needed to work on his thesis, rather than play "princess".

Chapter 1

Introduction

1.1 Introduction

The goal of this research is to provide a framework to allow robots with different capabilities (heterogeneous robots) operating in complex and dynamic environments to form and maintain teams in order to identify and complete work in these environments.

Robots operating in any real-world environment, even a laboratory or factory floor, have many challenges to contend with, such as noisy and inaccurate sensor data. Localization is imperfect, even where positioning systems such as GPS are available, and algorithms to intelligently interpret visual data are computationally expensive and inaccurate. Operation in hazardous environments such as those presented by the exploration of other planets and disaster zones must additionally deal with the fact that robots can be damaged or destroyed and new potential team members may arrive at arbitrary times as replacements, or as organizations commit new equipment.

The nature of many domains means that communication between robots is short range, unreliable and sporadic in nature: in disaster areas, for example, infrastructure can be heavily damaged, and the nature of debris itself can interfere with wireless communication.

Robots operating in these complex and dynamic environments should be able to take advantage of physical proximity to communicate their presence to one another and negotiate an appropriate team structure. My thesis research involves the development of a framework to facilitate this method of coordination, and examines its effectiveness in the context of a heterogeneous team of robots operating in a simulated disaster environment.

In this chapter I will begin by providing background and motivation for my research. Second, I will provide a high level description of my approach and how it benefits robots operating in complex and dynamic environments. Next, I will describe *Urban Search and Rescue*, an example of a complex and dynamic environment which I use as a grounded context for my teamwork research, as well as a formal evaluation environment. Finally, I will describe the research questions this thesis poses and provide an outline for the remainder of this thesis.

1.2 Motivation

Teamwork is an important part of our day to day lives. Tasks can be completed faster by dividing the work amongst team members, and this principle applies equally whether considering groups of humans or robots. Groups of robots have been shown to complete work significantly faster than independent individuals in applications

such as scavenging and exploration [Shell and Mataric, 2006; Rosenfeld et al., 2006; Ma et al., 2006; Rooker and Birk, 2007], for example.

There has only recently been a focus on how to form teams in situations where the individuals involved are self-motivated. Current works tend to involve high level tasks that do not translate well to the real world (e.g. package delivery in abstract space [Dutta and Sen, 2003; van de Vijsel and Anderson, 2005]). There is little consideration of the challenges present in a physical implementation on real robots, such as communication distance or reliability [Vig and Adams, 2005]. Further, work involving teams of heterogeneous robots (e.g. Kiener and von Stryk [2007]; Howard et al. [2006a]) are usually restricted to relatively controlled environments such as a laboratory and rely on fixed team structures determined in advance.

Aside from being able to complete a task faster, there are many benefits that can be realized when robots with different capabilities (*heterogeneous* robots) form a team. Using a single type of robot (*homogeneous* robots) with all possible capabilities required does not make sense in situations where there is a clear division of skills. In a manufacturing environment, for example, it would not make sense to have a single type of robot that welds, cuts, and moves goods around a warehouse. Even though such a robot could accomplish all tasks, the redundant equipment used for others would be a significant economic waste. Beyond the issue of expense, redundant equipment also requires additional energy to carry and provides greater likelihood of robot failure.

An example of a domain where heterogeneity is beneficial is urban search and rescue (USAR). According to Murphy et al. [2000a], rescuers identify the availability

of different robot physiologies as an important consideration when operating in a USAR domain. Wheeled robots are able to search open areas quickly, while tracked robots such as the iRobot PackBot are advantageous for climbing stairs and over areas of light debris [Yamauchi, 2004]. Further specialized physiologies, such as the snake-like robot developed by Murai et al. [2008], can be used for searching tight enclosed areas. Each robot physiology is well suited for use in certain situations, making it advantageous to have a variety of robot physiologies available.

There is also a utility in robots having access to a number of different sensory facilities, both from the standpoint of suitability to particular tasks and cost-benefit ratio. A laser scanner is both excellent for mapping and significantly more expensive than other forms of sensing, for example, making it cost prohibitive to equip every member of a team where some are expected to be lost are destroyed. Another example is the sensory equipment to detect the presence of victims in the environment. Burion [2004] studied sensors suitable for victim detection and found that the fusion of results from multiple sensors shows significantly improved victim detection performance. Equipping all robots with these capabilities would increase the overall cost and complexity of a search and rescue robot considerably.

1.3 Terminology

Before moving on to an overview of my approach, I will define some common terms that have specific meanings in the context of my work.

- *Agent* - Russell and Norvig [2003] define the term *agent* as something which perceives its environment using sensors in order to act upon the environment

using actuators. This broad definition can be applied to humans, who perceive their environment through their senses, and act using their limbs. Software agents in the general sense could read input from a database and display results on a computer monitor. For the purposes of my work, I define an agent as a mobile robot equipped with sensors which enable it to perceive the environment it operates in and actuators that allow it to move in and interact with the environment. I also make the important distinction that an agent consists of its control software and the robotic platform it operates on; the control software cannot transfer from one robot to another. The terms agent and robot will sometimes be used interchangeably.

- *Environment* - the environment is the area in which robots operate, collaborating towards the overall goal of the team.

- *Heterogeneous robot* - a heterogeneous robot is one in which differences in physiology, sensory equipment, and computational or memory capabilities exist between robots. My work focuses on the physical differences between robots that make them suited to performing particular types of work. I do not consider differences in robot control software or behaviour primitives when comparing robots.

- *Mission* - in the context of my work, a mission is an objective known to all robots operating in the environment. The robots work together to accomplish the mission.

- *Role* - a robot assumes a role on a team in order to determine the general

collection of tasks that the robot is normally expected to be able to perform. A robot well suited to fill a role has the necessary capabilities (e.g. sensory, computational, physiology) to complete the tasks that the role normally expects. Differences in robot capabilities mean that other robots may only be able to fill the role in a limited capacity, or not at all.

- *Task* - a task is a clearly separable unit of useful work that a robot can carry out in order to make overall progress towards the team's mission. In my work, a single robot carries out a task in isolation and does not require the assistance of any other robots to complete it (this is in contrast to tasks such as box-pushing in which robots require tight coordination to complete a task [Mataric et al., 1995]).

1.4 Approach

My thesis research involves the design and implementation of a framework that allows robots operating in a complex and dynamic environment to form and maintain teams in order to identify and complete work in these environments. My framework is intended for use in challenging domains involving area coverage tasks, using robots with potentially different skill sets. Robots employing this framework are able to dynamically structure themselves into teams in a decentralized manner. The framework allows robots to recognize situations where the team structure should be modified to accommodate the discovery of new robots and the departure or failure of existing robots. Further, my framework enables robots to identify work to be completed in

the environment in the form of tasks, and then to determine the appropriate robot to carry out these tasks.

Robots using my framework coordinate using short-range unreliable wireless communication. My framework takes advantage of the broadcast nature of wireless communication to allow robots to gather knowledge of other robots operating in the environment both through periodic announcement broadcasts and inspection of wireless traffic between other robots. This helps ensure knowledge of the current team structure disseminates amongst all members of the team, while reducing the overall amount of bandwidth dedicated to the dissemination of this knowledge.

My framework improves over existing frameworks that rely on reliable wireless communication and have fixed team structures where the mapping between robots and the work to complete is determined in advance (e.g. Howard et al. [2006a]; Kiener and von Stryk [2007]). The design of such frameworks makes it difficult for them to cope with changes in team structure and losses of communication that occur in hazardous environments such as a disaster zone.

Tasks in my framework are determined in advance of a mission, and describe the units of work that robots can perform. To determine the robot best suited to carry out a task, a task has both a minimum requirements and suitability expression defined in terms of the attributes of a robot. The minimum requirements determine the absolute minimum set of capabilities a robot requires in order to carry out the task. For robots that meet the minimum requirements of a task, the suitability expression determines the degree to which a robot is suited to carry out that task. Describing tasks in this manner forms the basis on which robots can reason about the best available team

member to carry out a specific task and assists in indicating when the current team structure is less than ideal or when new members should be sought out.

To facilitate efficient task allocation and assign a general responsibility of duties, I use roles to describe the set of tasks a robot filling the role is normally expected to be able to perform. Since roles are defined in terms of the tasks normally expected of them, and tasks are defined in terms of attributes a robot must possess in order to complete them, a robot can determine its suitability to fill a role. Using roles provides a short-cut when assigning tasks to other robots; a task can be assigned to any robot who occupies a role that is normally expected to be able to carry out the task.

Finally, my framework defines a desired team in terms of the number of robots and the roles they will occupy. A special purpose team coordinator role is present on each team. A robot occupying the team coordinator role provides a coordination point for all robots operating on the team. The team coordinator is responsible for assigning tasks identified by members of its team to the robots best suited to carry them out. The team coordinator attempts to ensure that each teammate has a small backlog of tasks to complete, helping agents remain productive in times of communication outages. All teammates have default tasks that define work they can perform on their own in the absence of instructions from the team coordinator. As the structure of a team changes due to team mates arriving and leaving, the agent occupying the team coordinator role can change as best suits the resulting team structure. This ensures that the team coordination responsibilities are not fixed to a single agent that is a single point of failure, and allows a team to adapt such that the best available agent performs these duties at any one time. The choice of desired team composition is

domain dependent and should take into account the general mix of work expected to be performed in the domain, as well as the capabilities of agents that will be available on a team.

Because this framework is intended for application in dangerous or challenging environments, it assumes that any agent could suffer a failure or become separated from its team at any point [1]. In a damaged building, for example, a structural collapse could isolate the agent from its team or even destroy it entirely. A failure of this nature would potentially incapacitate a team if the failing agent occupied the team coordinator role. To allow teams to continue operation in such circumstances, agents using my framework periodically perform a *role check*. During the role check, the agent uses its knowledge of the agents on its team to determine the role it should fill at the current time. In this way, agents are able to adjust the role they occupy on their team to compensate for the loss of team mates. The desired team definition also allows an agent to identify situations where the addition of another agent to its team would be advantageous. As the number of agents filling each role changes, those where the number of agents filling the role falls below the definition of a desired team are given a higher priority to fill when replacements of other teams are encountered in the environment.

My framework also supports the operation of multiple cooperating teams within an environment. Agents that encounter one another in the environment take advantage of visual and radio contact in order to exchange mission progress information. This helps reduce the chance of unintended redundant effort between teams. During

[1] Realistically, however, this could happen even in the safest of domains and does not preclude the use of this framework in everyday environments as well.

an encounter the agents also share knowledge about their teams and perform a redistribution of team members. The redistribution of team members helps ensure that teams are re-shaped closer to the desired team structure. If a team loses its leader, for example, it has the opportunity to pick up a better suited leader from another team it encounters. Alternatively, an agent might also become lost or separated from its team. Upon encountering another agent or group, it has the opportunity to become a member or form a new team. This is advantageous as the agent is able to continue performing useful work for another team, while at the same time distributing knowledge from one team to another.

1.5 Urban Search and Rescue

In order to study dynamic team formation in a grounded manner, a representative domain must be chosen. A good example of a highly challenging domain in which agents can be of value is in the aftermath of a natural or man-made disaster. This is commonly known as *Urban Search and Rescue* (USAR) [FEMA, 2000], and involves exploring damaged structures to locate and assist human casualties. I evaluate my work using a simulated USAR environment in which robotic agents must coordinate in order to locate victims while generating a map of the disaster zone.

Murphy et al. [2000a] describe the mobility and sensory difficulties present in such an environment. Debris and uneven terrain can make it difficult for a robot to navigate, and can cause a robot to become stuck. The collapse of a building results in many small *voids* that larger robots would be unable to explore. Existing floor-plans and maps of the environment are not useful as structural changes to the

environment as a result of the disaster may have caused considerable changes to the configuration of walls and spaces. Further, progressive collapses can occur during the rescue operation further changing the expected layout.

During the robotic USAR operations at the World Trade Center disaster, Casper and Murphy [2003] found that rescuers had a difficult time visually identifying features in images due to dust covering all areas of the environment; the current state of the art in image processing would not have fared better. According to Murphy et al. [2000a], readings from sonar sensors can be hindered by the presence of sharp angles in the environment and varying surface materials. Laser range finders offer improvements in these areas, but suffer issues with reflective surfaces. Their cost also precludes their widespread use.

According to Wong et al. [2004], the use of robotics in USAR promises to help rescuers provide a more timely and efficient response to disasters. Robots are not subject to the same constraints that human rescuers are; they can be sent to perform search and rescue operations in conditions deemed unsafe for human rescuers. Robots do not suffer from fatigue during continual use. They are always alert and ready to perform their duties.

From an academic standpoint, USAR robotics provides researchers with numerous difficult problems to solve. USAR robotics is a multidisciplinary research field, involving work in artificial intelligence, human robot interaction, psychology, and engineering. The Computing Research Association [2002] identifies the creation of computer technologies to minimize the impact of disasters as a grand challenge in computing. Further research in USAR robotics is recommended to aid in disaster

response.

1.6 Research Questions

I will use the approach and domain outlined in the previous sections to answer the following research questions:

1. **Can my framework provide teams operating in dynamic environments with the ability to adequately cope with changes in team structure and composition (i.e. due to loss and failure of team members, and encountering other teams and teammates in the environment)?** I will compare the relative performance of agents using my framework against a baseline approach where agents cannot switch roles and teams. An increasing level of probabilistic agent failure will be used to demonstrate the ability of my framework to cope with the changes in team structure that will occur due to failures. Further, replacement agents will be introduced into the environment, which can be discovered by existing teams.

2. **Can my framework help mitigate the negative affect of unreliable communication on coordination efforts between agents?** This question can be answered by observing the difference in performance while the communication reliability is varied when using my framework versus the baseline approach where agents cannot switch roles and teams.

3. **Is my framework able to cope with failure of a team's leadership structure?** Inducing a failure in leadership at set points allows for observation

of the performance benefits realized by using my approach versus the baseline.

1.7 Thesis Organization

The remainder of this thesis is structured as follows:

- **Chapter 2 - Related Work** - reviews literature related to this thesis for topics such as teamwork, exploration, search and rescue and other peripheral areas such as mapping techniques.

- **Chapter 3 - Methodology** - discusses my teamwork and task assignment framework.

- **Chapter 4 - Implementation** - discusses the implementation of my framework for use in the simulated USAR environment in which I evaluate my work. Provides detailed descriptions of the elements of my framework that are coded specifically for this domain. Also provides detailed descriptions of the peripheral components required to operate in a USAR environment such as, for example, the victim tracker and frontier finder modules.

- **Chapter 5 - Evaluation** - describes the experiments I use to evaluate my work, and discusses the results obtained.

- **Chapter 6 - Conclusion** - answers the thesis questions within the context of my experimental results, providing further discussion and ideas for future directions this work can take.

Chapter 2

Related Work

My work covers a number of topics within the general field of multi-agent systems. Specifically, my work covers elements of multi-agent cooperation in area coverage tasks. My methodology supports this cooperation through the formation and maintenance of teams, and task allocation within those. As my example implementation involves performing an urban search and rescue mission in an inherently complex and dynamic environment, there are a number of peripheral research areas with which my work is concerned. Some examples of these include autonomous robot control, environment mapping, and distributed exploration.

This chapter introduces important related research, and describes the areas where my work is similar to and differs from these related works. Section 2.1 begins with a selection of topics related to coordination in multi-agent teams. The section begins by reviewing related work involving the formation and maintenance of multi-agent teams, and the use of roles to set expectations of agents on a team. Related work in the area of task allocation is presented, and a description of current works involving

effective heterogeneous teams is provided.

Section 2.2 examines related research in the field of urban search and rescue robotics, both from the perspective of the types of robots desirable to perform this mission, and current works to support coordination of robots in a rescue environment.

Finally, Section 2.3 covers a variety of topics peripheral to the development of my framework, yet important to the development of my example implementation. This includes topics such as autonomous robot control, mapping, and distributed exploration.

2.1 Coordination in Multi-Agent Teams

This section reviews related literature dealing with coordination in multi-agent teams. It reviews related literature dealing with key areas related to my work: roles, task allocation, heterogeneous teamwork, and team maintenance.

2.1.1 Team Maintenance

Until recently, there has not been a large focus on how to form and maintain teams of robotic agents. Most previous works assume teams were formed in advance and will not change during the course of operation (e.g. [Rekleitis et al., 2004], [Bruce et al., 2003], [Lau et al., 2009], [Boonpinon and Sudsang, 2007], [Giannetti and Valigi, 2006], [Li et al., 2007]). This section reviews previous multi-agent systems work dealing with the formation of mutually beneficial partnerships between robots. These works provide a basis for the type of teamwork found in my work.

Dutta and Sen [2003] studied cooperation between heterogeneous agents in an abstract package delivery domain. Their approach assumes agents are self-interested and can form mutually beneficial partnerships with other agents. The agents cooperate using the principle of reciprocity; an agent can only expect to get out of a partnership what they have contributed themselves. That is, an agent's willingness to provide assistance to another is dependent on the history of interaction between the two. In contrast to teams in my work, the partnerships between agents occur between two agents only and are limited in scope to the interactions between those agents. Since agents are self-interested, there is no imperative for agents to form larger aggregations like the teams in my work. My work assumes agents are not self-interested and have a common goal. Finally, the package delivery domain Dutta and Sen [2003] use to study their work has been so strongly abstracted from the real world as to make it unrealistic for a physical robotic domain. The impact of communication between agents is not considered, and there is an inherent assumption each agent can communicate with any other. This is in contrast to real-world domains where communication is unreliable and limited by the range of a robot's transmitter. Further, Dutta and Sen [2003]'s package delivery domain abstracts physical space to a series of spokes emanating from a central hub. This abstraction lacks the realism of working in a true 2-D or 3-D environment, and does not concern itself with issues such as localization in the environment. In my work, teams involve a larger number of agents than Dutta and Sen [2003]'s partnerships out of necessity, both from the perspective of parallelism in operation and redundancy of agents.

van de Vijsel and Anderson [2005] build on the work by Dutta and Sen [2003], and attempt to increase realism in the team formation process by assuming agents have conflicting goals, multiple teams can exist, and agents learn of others based on their interactions. Similarly, my work assumes multiple teams can exist, but I do not assume agents have conflicting goals. Although learning about other agents through interactions is a broadly applicable concept to robotic domains, the underlying domain choice is again too abstract to be suitable for use in a real world domain. van de Vijsel and Anderson [2005] uses a package delivery domain where agents work in a two dimensional grid environment. Although this is more realistic than Dutta and Sen [2003]'s package delivery domain, it again does not consider issues such as perception of agents in the environment, localization, or the impact of limited range unreliable communication.

Brooks and Durfee [2003] propose using *congregations* as a means of supporting cooperation between agents. Similar to teams in my work, congregations provide a means for agents with similar needs to identify one another in order to have repeated and meaningful interactions. Brooks and Durfee [2003] assume agents in a congregation will fill different roles over the span of their membership, take on different tasks, and interact with a wide number of agents. Although these assumptions agree with my use of teams, there is still an underlying assumption that an agent's participation in a congregation is dependent on the gains it can achieve from being a member of the congregation. In my work, I assume agents cooperate selflessly in order to advance the overall mission of the team. An agent's membership on a team is not dependent on the gains it can receive from being a member of the team, but rather based on

where the agent feels it is best suited to contribute. Further, Brooks and Durfee [2003] studied their congregation approach in an information economy domain where agents act as consumers of articles offered by different producers. As with Dutta and Sen [2003] and van de Vijsel and Anderson [2005], this domain lacks the level of difficulty expected in any real-world domain in which robots operate, making this approach unsuitable for difficult domains such as USAR.

George et al. [2010] developed a method to form coalitions between members of a larger team of unmanned aerial vehicles (UAVs) operating in a region. These coalitions facilitate the cooperative engagement of a target by a subset of the overall team. Where a UAV determines it does not have the required resources to engage a target, it broadcasts a request for the resources it requires to engage its target. UAVs in radio range hear the request and respond with a bid indicating the resources available, and an estimated time to travel to the target location. George et al. [2010] assumes communication is limited in range and propose techniques for propagating information among the team members, given messages can propagate a greater distance through the user of multi-hop message routing. Although their work occurs in a more real-world domain, it focuses on the formation of sub-teams intended for the completion of a single task and does not deal with the formation of the overall team of UAVs. In contrast, my work assumes tasks are carried out by a single agent. A method such as George et al. [2010]'s might be useful to add the ability for robots in my work to form sub-teams in order to cooperatively complete tasks, but it is not sufficient on its own to form and manage long term teams as members are lost and new members arrive.

Cheng and Dasgupta [2010] studied a technique to form teams among groups of robots operating with the overall goal to explore an area. They examine the hypothesis that smaller teams of robots cooperating in an environment can more effectively explore the area than a single larger team. Their work uses elements of game theory to study how robots can partition themselves into smaller teams in a manner such that the resulting teams perform the optimal exploration of the environment, minimizing the overlap in coverage between robots. They have evaluated their technique using mathematical models in an abstract domain, but plan to study their technique in a simulated robotic domain in the future. As such, this approach remains a mathematical abstraction that has yet to be demonstrated to be practically applicable. In addition, while their approach to forming teams aims to form teams which are mathematically closer to optimal, my work relies on heuristic descriptions of a desired team to guide the team formation process. However, techniques such as Cheng and Dasgupta [2010]'s could be used to enhance the decisions made during my team merge and redistribution process (Section 3.7).

2.1.2 Roles

The concept of using roles as a means to set expectations of agents is common in multi-agent systems research. This section begins by examining possible variations and interpretations of the general term, and continues by describing related uses of roles with appropriate comparisons to my work.

It is important to gain an understanding of what the term *role* means, and the possible variations that are possible. Odell et al. [2003] studied the concept of roles in

the context of human organizations, drawing influences from the behavioural sciences. They distinguish between emergent roles, where the behaviours and operations of roles are learned through operation of the multi-agent system, and imposed roles where the roles are pre-determined in advance. Since imposed roles mimic how roles in human organizations operate, Odell et al. [2003] argue that where humans must interact with a multi-agent system, it may be advantageous to use an imposed role structure. The roles in my work are imposed by the specific tasks (or units of work) that agents perform (future work could study the use of emergent roles).

Odell et al. [2003] define the *horizontal specialization* of roles as referring to the breadth of operations that an agent filling a role can perform. Roles that have too much horizontal breadth result in a tendency towards homogeneous robots and a limited set of resulting roles. Instead, Odell et al. [2003] argue that role design should tend towards having a variety of limited capability roles, which helps promote heterogeneity in agents. Although the roles in my work tend towards defining a broad range of operations an agent can perform, I do not assume that agents are able to complete all operations expected of the role, leading to each agent having a degree of suitability to fill a role.

According to Odell et al. [2003], the *vertical specialization* of a role determines the degree to which an agent filling the role relies on other robots to coordinate its activities within the role. Vertically narrow roles result in an agent relying heavily on other agents for direction, while vertically broad roles focus on the direction and coordination of other agents. In my work, roles fall in the middle of the spectrum as the team coordinator role provides general direction to agents which are expected to

perform work independently.

Similar to my work, Odell et al. [2003] define a role assignment as a mapping of an agent to a role, and a position as a specific role assignment within a group (team). A number of positions make up a group (team). Similarly, I define a desired team in terms of the number of positions that make up a team.

With an understanding of roles and properties that can be used to describe them, the remainder of this section examines some related works involving roles and compares their use to my work.

Xu and Xia [2009] developed a system to facilitate assignment of roles to robots operating in complex domains without the use of explicit communication between robots. They evaluate their work in the context of simulated robotic soccer. Similar to my work, Xu and Xia [2009] describe a role as being defined by the activities that a robot performs while filling that role. However, Xu and Xia [2009] do not abstract the activities robots perform into tasks. In the robotic soccer domain, Stone and Veloso [1999] have previously shown the nature of the domain is such that the work involved is determined by areas of the playing field. As a result, the use of tasks would be questionable, unless the tasks were bounded to a specific area of the playing field. Xu and Xia [2009] assume all robots are homogeneous and have the capabilities to fill any role equally well, where my work assumes robots are heterogeneous and each robot can fill a role to a varying degree. Further, Xu and Xia [2009] assume a team is defined in terms of a set of roles, where each robot fills a single role on the team. In contrast, my work assumes each role can be occupied by many individuals and a team is defined by the roles and number of individuals occupying each role. Similar

to my example implementation, robots are assigned roles preferentially based on their location in the environment (I assign tasks preferentially to the robot closest to the task location). Finally, Xu and Xia [2009] assume a lack of explicit communication among teammates, requiring robots to be able to observe all other members of their team in order to form an accurate world model on which the role determination is based. This is appropriate in a robotic soccer domain as the physical environment in which the robots operate is limited in size and all robots can reasonably be expected to observe one another in order to form a world model. This assumption is not realistic in domains such as USAR where robots are spread out and there are a large number of obstacles present in the environment.

Using a robotic soccer domain, McMillen and Veloso [2006] studied an approach to assigning roles to players on a team. In their work, roles act as descriptions of the work a robot filling that role will complete. McMillen and Veloso [2006] consider role allocation to be analogous to task allocation. In effect, they consider roles to strictly define the work to be completed by a robot filling a role. Further, they use roles to define the area a robot operates in when filling that role. This is in contrast to my work where roles are less strict and only set heuristic expectations of the work to be completed. Further, my work does not place constraints on the areas in which a robot can operate based on the role it fills. McMillen and Veloso [2006] define plays to determine the series of actions a team must take given the current position of the players and the soccer ball. The plays define the roles that must be filled to complete the play. In contrast to my work where roles are filled by robots in a decentralized manner, McMillen and Veloso [2006] assign roles through a leader robot

who determines which robot should fill each role based on their location on the field. The use of plays provides the potential for the roles on a team to change during the game, where in my work the roles required of a team are determined by the definition of a desired team.

This section has examined the concept of roles and their origin in behavioural sciences. It also examined some related works involving roles and illustrated the similarities and differences between those works and my own. The next section examines how tasks can be allocated to members of a team.

2.1.3 Task Allocation

Given a team of agents cooperating towards a common goal, the process of task allocation is concerned with choosing which agent(s) on a team should take on which tasks. This section examines task allocation approaches related to the approach I use in my work. I begin with a description of the Contract Net approach, a framework that forms the foundation for the other approaches I outline below.

Davis and Smith [1983] developed the Contract Net approach to support the completion of tasks with a distributed collection of agents. The framework is one of the major foundations of multi-agent systems research, and is still implemented and extended in systems today [FIPA, 2002]. The Contract Net approach treats distributed task allocation as a contract negotiation between an agent with a task requiring assignment, and agents that negotiate to ensure the completion of the task. The framework assumes any agent can act as a contractor (an agent bidding on a task) or a manager (an agent with a task requiring completion). Using the Contract Net

approach, tasks are announced with a description of the work to be completed, the criteria an agent must meet in order to consider bidding on the task, and the criteria used by the manager to evaluate bids. Contractors evaluate whether they meet the necessary criteria, and respond with a bid based on the criteria. A bid criterion could be, for example, a measure of the estimated time the contractor expects it would take to complete the described task. The manager evaluates all of the bids it receives and assigns the task to the best bidder.

My approach to task allocation is similar to the Contract Net approach to the degree that similar issues of contracting underlie much modern multi-agent systems research. Each agent in my approach is responsible for evaluating its individual suitability to complete the task and reporting back to the task assigner. However in contrast to the Contract Net approach, agents do not communicate the details of the tasks requiring completion, as the nature of the tasks to be completed is determined in advance of operation. Further, my work uses two passes of contract negotiations to perform task assignment. In the first phase, role assumptions are used to narrow the pool of bidders to only those expected to be able to complete the task; this helps reduce communication traffic and frees unsuited bidders from having to prepare a bid. Where the first phase fails to result in the contract being awarded to an agent, the second phase opens the bidding to all agents nearby (Sections 3.5.3.1 and 3.5.3.2).

Finally, the Contract Net approach assumes a contractor can further decompose a task and sub-contract the components to other agents [Davis and Smith, 1983]. In my work, I assume tasks are completed by a single agent without being broken down further. A task may ultimately be routed to another agent if the agent accepting it

fails, however.

Building on the work of Davis and Smith [1983], Gerkey and Mataric [2002] developed an auction-based method for distributed task allocation amongst teams of heterogeneous robots. Similar to my work, it is assumed robots are subject to failures, communication between robots is unreliable, and task executors are responsible for completion of a task without supervision by the assigner. Task allocation occurs in a decentralized manner in that each agent is responsible for assigning its own tasks. Communication between robots uses a *publish/subscribe* approach. Messages are published based on *subjects* that describe the capabilities required to complete tasks. Robots subscribe to subjects based on the capabilities they possess. When a robot announces a task, it publishes an announcement based on the capabilities required to complete the task. The announcements include a bid evaluation method each subscriber to the message uses to evaluate its suitability to complete the task. Subscribers respond with their evaluated bid for the completion of the task; the assigner chooses the winning bidder and assigns the task. Task completion is time-bounded and the assigner is responsible for monitoring the progress of the task by a task executor.

Gerkey and Mataric [2002]'s approach of agents bidding on tasks is similar in spirit to how agents in my work respond to task announcements with their suitability to complete a task. However in my work the task assigner uses roles as a means of filtering the agents the task announcement is addressed to, where Gerkey and Mataric [2002] relies on the publish/subscribe communication model to limit the agents considered for bidding.

Kiener and von Stryk [2007] present a framework for the cooperative completion of tasks by teams of heterogeneous robots. Kiener and von Stryk [2007]'s framework achieves this by modeling the individual tasks of the overall mission, and storing the degree to which each of the robots can perform these tasks. The capabilities of (only) a single humanoid and single wheeled robot are determined in advance, along with weights identifying the suitability of each to all possible tasks. This information allows a central controller to allocate tasks to each robot.

While the tasks involved are significant in that they involve fine motion control and interaction (e.g. parking a wheeled robot while a humanoid mounts it), this is still very primitive in terms of task allocation. The broadly different robot skills and task demands in Kiener and von Stryk [2007]'s work results in a predefined set of tasks with only one logical way to map these tasks to the robots in their system. Their approach also requires constant communication with a central controller, where my work performs task allocation in a distributed manner. My framework also assumes there are many possible mappings between robots and tasks. Tasks will be discovered as robots move through the environment, rather than allocated by a central controller. The design of my approach will also suppose adding robots to existing teams, as well as forming a team from an initial collection of robots. Further, my work does not have the requirement that reliable communication is available between team members for successful task completion.

My work, however, uses a similar approach to mapping tasks as that of Kiener and von Stryk [2007], in terms of recording knowledge about tasks and robot capabilities and attempting to match these, albeit in a more sophisticated way. My matching is

necessarily heuristic in nature, because of a much greater number of possible match combinations. I also extrapolate robot capabilities into defined roles which determine the types of tasks suitable for a particular role to perform.

2.1.4 Task Completion

In my work, each agent maintains a prioritized list of tasks it will carry out. The task-list in my work is similar in concept to an *agenda*. An agenda is a broadly used concept in artificial intelligence research, originating from the Speech Acts Theory as a way of describing how the human mind tracks pending speech acts it is considering vocalizing [Cohen and Perrault, 1979]. Agendas are commonly used in belief-desire-intention (BDI) architectures to manage the actions an agent is considering given its current beliefs about the environment and intentions / goals [Bratman et al., 1988]. Matelln and Borrajo [2001], for example, uses an agenda to specify the actions a robot will execute next. Actions are added by the robot, actions themselves, and other cooperating robots. Pokahr et al. [2005] define a framework to allow an agent to deliberate about its goals in order to plan the required actions to achieve them. They use an agenda to track the pending actions an agent considers given its current goals.

2.1.5 Effective Heterogeneous Teams

This section reviews a selection of related works involving teams of heterogeneous robots cooperating effectively in real-world domains. These works illustrate the benefit of heterogeneity in robot types, and provide inspiration to the choice of robots

used in my example implementation.

Parker et al. [2003] developed a system to automatically deploy a sensor network using heterogeneous robots. Resourceful *leader* robots guide the overall deployment of the network, while *follower* robots act to provide direct movement instructions to the sensor nodes necessary to keep them in formation and deploy them. In contrast to the work I propose, the teams, formation, and deployment positions of the sensor nodes are all pre-computed in advance and rely on the presence of reliable communication to a central processing unit. Changes in team structure due to the loss of robots are accounted for by looking up a new pre-computed deployment pattern and adjusting the formation accordingly. No attempt is made to recover lost robots. Further work by Howard et al. [2006a] investigates the performance of the system with a simpler team structure (eliminating the follower robots), and sensor node robots with basic autonomous navigation capabilities. During their experiments, they found the leader robot would lose track of the simpler robots due to issues detecting them in the environment. Howard et al. [2006a] argues this is further evidence that fault tolerance must be planned into any real-world implementation.

The framework developed by Kiener and von Stryk [2007], introduced in Section 2.1.3, to support task allocation to robots is also interesting from the perspective that it involves heterogeneous robots cooperating on a team towards a common goal. They demonstrate their framework with a humanoid and wheeled robot, and task the pair with following a soccer ball (held by a human) down a corridor, and then kicking it into a goal. Neither robot is well suited to completing the entire mission on its own. The humanoid robot is able to see and track the ball, but has limited locomotion.

The wheeled robot has no vision capabilities, but has fast locomotion. Kiener and von Stryk [2007] demonstrate that the two robots can solve the problem by working together: the humanoid rides on the wheeled robot to the ball, then dismounts and deposits the ball in the net. Although their work uses heterogeneous robots, it is demonstrated using only two robot types. In contrast, my framework will operate with many types of robots. Finally, Kiener and von Stryk [2007] use a fixed team structure which is determined in advance, where in my work I assume the dynamic nature of the environment will result in changes to team membership as the mission progresses. Further, I assume robots are able to form and disband teams in order to adapt to the changes in the available mix of robot types.

Dorigo et al. [2011] developed the *swarmanoids* architecture as a means of encouraging research into swarm robotics in real-world domains. Similar to Kiener and von Stryk [2007], Dorigo et al. [2011] use heterogeneous robots to carry out complex cooperative tasks in a real-world domain. Their work uses three robot types which cooperate to complete the mission of locating a book on a shelf and retrieving it. A flying *eye-bot* robot has the ability to autonomously fly in indoor environments and uses vision to detect the book on a shelf and direct the rest of the team to retrieve it. *Foot-bot* robots are traditional wheeled robotic platforms, equipped with the necessary sensors to navigate autonomously through the environment without colliding with obstacles. Finally, a *hand-bot* robot includes a gripper suitable for gripping the target book from a shelf, and a magnetic tether system that allows it to attach to a metal ceiling and raise itself up the book shelf in order to grab the book. The hand-bot robot type is unique in that it has no locomotion capabilities of its own;

it depends on foot-bot robots to move it into position to retrieve the book from the shelf. Although their example task of retrieving a book from a shelf is an impressive demonstration of heterogeneous robots cooperating to complete a mission, the heterogeneity between the robot types precludes them from being used in a combination other than their demonstrated configurations. In the absence of foot-bot robots, for example, a hand-bot robot would be unable to move; it could not link up with an eye-bot to move around. The current swarmanoids implementation places restrictions on how robots can be combined, and leaves little opportunity to adapt to changes in available robot types. In my work, I assume a certain degree of overlap in the capabilities of the various robot types. This allows teams to adapt to scenarios where a more capable robot type has been lost, and provides the ability to continue operation in a potentially degraded state using a less capable robot as a substitute for the lost robot.

2.2 Urban Search and Rescue

This section reviews a selection of related topics in the field of urban search and rescue robotics. I first examine work to covering the properties of robots necessary to support effective operation in a USAR domain, and then describe test-beds designed to foster research through the establishment of standardized example environments in which USAR robots can be tested. Finally, I examine related works to accomplish the USAR mission through the use of robots, and examine related control methodologies and uses of heterogeneous robot types.

Schlenoff [2005] studied the properties of robots used in USAR to develop an ontology that describes their physical, functional and operational characteristics. Their goal was to develop a standardized description of USAR robots in order to facilitate the development, testing and evaluation of robots for use in USAR operations. My framework describes robots in terms of attributes that describe a robot's suitability for executing tasks in domains such as USAR. The attributes I use in my example implementation are similar to those in Schlenoff [2005]'s work and describe the locomotion, sensory and computational capabilities of the robots. A real-world implementation of my work could use the ontology created by Schlenoff [2005] in order to provide a commonly understood description of the robots used.

To promote research into USAR robotics, the National Institute of Standards and Technology (NIST) developed a series of robotic USAR test arenas, which are used in the RoboCup Rescue competition [Jacoff et al., 2003]. These test arenas provide a real-world physical simulation of conditions a robot can be expected to encounter while operating in a disaster zone. Three standard configurations represent environments of increasing difficulty, and provide simulated victims that must be located. The goal of the NIST test environments is to provide a standardized means to evaluate and compare USAR robots. Jacoff et al. [2003] also describes the USAR mission used in the RoboCup Rescue competitions. Similar to my example implementation, robots must explore an environment while building a map and identifying the location of victims found. Further, in my implementation I evaluate the performance of my framework in terms of the percentage of victims located and the percentage of the environment explored. This is similar to the performance metric used in RoboCup

Rescue which assigns a point-score based on the number of victims located and quality of the map created.

Reich and Sklar [2006a,b] make use of both robots and wireless sensor nodes in order to locate victims within an environment and to subsequently localize them. Similar to my work, an assumption is made that the sensor nodes are simple, inexpensive, expendable, and available in larger quantities than the more capable robots. The sensor nodes detect victims in the environment through some means (their example implementation uses a simulated victim detection approach), and broadcast the presence of victims to neighboring sensor nodes. Sensor nodes pass along this information, tracking the number of hops to the detected victim. As knowledge of a victim propagates through the network, robots begin to move toward the closest sensor node with the lowest victim hop-count. The robots, knowing their starting location, can thus determine the location of the victims. In my work, I assume robots use a local coordinate system based on their starting location and that the members of a team reconcile differences in their coordinate system, lending to a shared understanding of locations in the environment. Knowledge of detected victims is propagates to other agents as a specific location, rather than using communication hop-counts.

Carnegie [2007] developed a robotic USAR system which uses a team consisting of three different classes of robots. The largest robots act to carry the smaller two to the edge of the disaster zone. The middle size robots are responsible for deploying the smallest robots into the environment. Similar to the work I propose, the smallest robots are intended to be inexpensive enough that they are expendable. Task allocation in Carnegie [2007]'s system assumes the availability of high bandwidth

communication and a preformed team structure – two constraints my work does not assume.

Dissanayake et al. [2006] developed the CASualty robotic search and rescue system. CASualty makes use of three different classes of robots with different physical forms and sensor capabilities, suitable for operation on different terrain types. Similar to the work I propose, Dissanayake et al. [2006] make use of an occupancy grid mapping approach and frontier-based exploration. Although their approach uses multiple heterogeneous robots to accomplish the search and rescue task, coordination between the robots relies entirely on human operators.

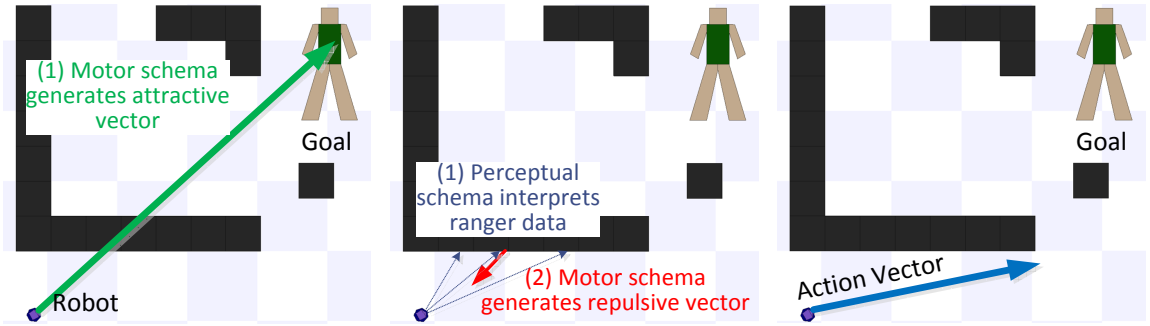
Wegner and Anderson [2004] implemented a robotic control system for use in a USAR domain. Their work focuses on the issue of blending instructions from a human operator with the instructions generated by robots' own autonomous control system. The system also provides the capability to identify situations where a robot requires the assistance of the human operator (e.g. a robot getting stuck). Gauthier and Anderson [2005] extend this work by investigating situations where robots can assist one another. Neither of these works employs heterogeneous robots as I intend to, and more importantly, neither performs any type of autonomous team management.

Scone and Phillips [2010] recognize that in a rescue environment communication will necessarily be short range and limited in nature. They assume a robot will be designated as a coordination point through which other robots will report the discovery of victims in the environment to rescuers residing outside the environment (presumably this robot has increased communication facilities that enables it to communicate over a longer distance). Since robotic rescue aims to find as many victims,

as quickly as possible, Scone and Phillips [2010] study the impact of robots moving to report their results to the coordination point on the overall mission. When a robot moves to report to the coordination point, it is not actively searching for victims. Conversely when a robot is exclusively searching for victims, it is potentially delaying the delivery of information regarding the presence of victims it has already found back to the coordination point. They found that using a biologically inspired foraging strategy, a balance between exploring and reporting results to the coordinator could be made, minimizing the overall time to find and report the location of the victims in the environment. Although my work does not explicitly deal with the impact of reporting exploration results back to the rescue team, my framework does attempt to ensure that a single agent on a team maintains a complete map of the environment along with the location of the victims. Strategies such as Scone and Phillips [2010]'s could be used to ensure that team coordinators regularly report their results to the rescue team.

2.3 Experimental Domain

Since my example USAR implementation involves the use of robotic agents, there is also background to my thesis in terms of robotic control and other problems traditionally associated with mobile robots, such as mapping. Although the primary focus of this thesis is the agent interaction / teamwork aspect of my approach, it is important to provide a necessary foundation in related areas necessary for a team of robotic agents to operate in a disaster environment. This section provides necessary background for robotic control software, environment mapping techniques, and



(a) (b) (c)

Figure 2.1: The goal exerts an attractive force on the robot while obstacle hits exert a repulsive force. The vector sum becomes the action vector the robot uses to move.

general exploration approaches.

2.3.1 Autonomous Control

The most commonly-used approaches to control in modern mobile robotics are behaviour-based approaches, where control logic is grouped into interacting packages for specific behaviours. An example of this is Arkin [1987]'s Schema-based approach. Arkin uses *perceptual* schemas to filter incoming sensory information into a format suitable for use by *motor* schemas, which determine the action the robot will take based on the current perceptual information. Schemas can be organized hierarchically, and can stimulate or inhibit one another's outputs, allowing for complex behaviour to emerge from the interaction of simple components.

Figure 2.1 shows an example of using schemas to direct a robot to a goal, while avoiding obstacles. In 2.1a, a motor schema directs the robot to the goal location by generating an attractive vector proportional to the distance from the goal loca-

tion. Obstacle avoidance is facilitated through a perceptual schema 2.1b(1) which interprets the distance readings from the robot's sensors to identify the location of obstacles. The output of this perceptual schema feeds into a motor schema 2.1b(2) which generates a repulsive vector to guide the robot away from the obstacles. The vector-sum of the motor schemas results in the action vector 2.1c which guides the robot towards the goal while avoiding obstacles. Beyond this global vector summation, schemas may be arranged to specifically reinforce or inhibit the output of other schemas in particular.

I use a schema-based approach in my example implementation to provide robots with the ability to interpret and process raw sensory data, and provide a set of motion primitives such as *move to location* and *random wander* motor schemas. The implementation of my framework provides a deliberative layer than enables and disables schemas as necessary based on the robot's current task. This goes beyond a pure schema-based approach itself, making my behavioural implementation a hybrid architecture, analogous to others who have also added deliberative mechanisms to schema-based models (e.g. AuRA [Arkin and Balch, 1997]).

2.3.2 Mapping

Since one of the primary goals of robots in my work is to generate a map of the environment, an effective means of representing the map information is required. The major approaches to mapping can be divided into topological mapping [Goedemé et al., 2008], which creates a map that specifies the connections between areas, and occupancy-grid based mapping [Elfes, 1989], which divides the map into a grid

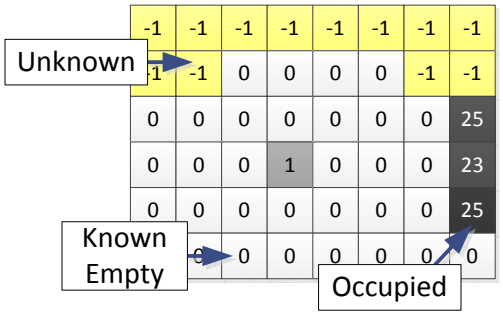


Figure 2.2: Occupancy grid maps indicate the certainty that an obstacle is present in an area represented by a cell.

with each cell representing a fixed amount of space. Each cell in the grid indicates the degree of confidence that an obstacle occupies that space. I will be taking the latter approach for my mapping, because of the ease of creating and communicating occupancy grids. While this is the less sophisticated of the two, and would not likely be used in real-world USAR, mapping is a peripheral element in my research, and this method will support operation in my simulated environment while keeping the scope of this work reasonable.

In my implementation of occupancy grids, Equation (2.1) illustrates how the value of a cell, $C_{x,y}$, represents the degree of confidence there is an obstacle present in the location that cell represents.

$$C_{x,y} = \begin{cases} -1 & \text{nothing is known about (x,y);} \\ 0 & \text{the cell is unoccupied;} \\ 1..M & \text{the cell may be occupied.} \end{cases} \tag{2.1}$$

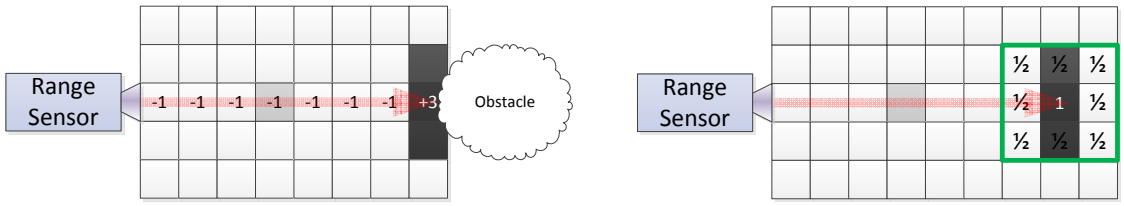
Where $C_{x,y} = -1$, the robot has no knowledge of whether there is an obstacle or not at the location (x, y); the area is unexplored. A value of 0 represents a certainty

that a cell is unoccupied. Values from 1 to M represent an increasing degree of confidence a cell contains an obstacle. The value M represents a maximal degree of confidence a cell contains an obstacle – my implementation uses $M = 25$. Figure 2.2 shows a small occupancy grid map with unknown (unexplored) space, known empty space, and areas of differing obstacle confidence.

Gutmann et al. [2005] extends the basic two-dimensional occupancy grid map by adding a height measure to each cell. This means that each cell records not only the confidence an obstacle occupies a space, but also the height of the obstacle in that space. Gutmann et al. [2005] calls this a 2.5D occupancy grid map, and uses it for obstacle avoidance with humanoid robots traversing a standard floor surface. The height information allows the humanoid robot to plan its steps given the presence of obstacles in the environment that it can potentially step on to or over. In my example implementation, I use a 2.5D occupancy grid map to differentiate between areas of the environment covered by low debris suitable for navigation by tracked robots, versus open areas of the environment suitable for navigation by wheeled robots.

2.3.2.1 Updating the map

Robots update an occupancy grid map of the environment to indicate the potential presence of obstacles. A robot equipped with a bump sensor, for example, could update an occupancy grid map when it collides with an obstacle. The robot could later drive over the same location, decreasing the confidence of an obstacle being present at that location. The same principle also applies when considering observations reported by other robots; if another robot reports a location as empty, this can decrease the



(a) Updating the confidences based on a scan.

(b) The growth rate operator updates the confidence of obstacles based on a cell's neighbors.

Figure 2.3: Updating an occupancy grid map using HIMM.

confidence of an obstacle being present at a location. Observations made by other robots are generally given a lesser importance to those made first-hand, as there is inherently some degree of uncertainty due to errors in localization. That is, a second robot may report an obstacle at a given location thinking it is at that location, when in fact it is somewhere else.

Robots building maps of an environment will typically make use of range-finding sensors to build a map of the environment. Range-finding sensors, such as sonars and laser range scanners, provide a means of estimating the distance from the robot to the nearest obstacle in the path of the scan beam. Sonar sensors are generally fixed to the robot's body and return a range reading in the direction the sensor points. Laser scanners sweep a laser beam across the scanner's field of view and return range readings at fixed intervals; a SICK laser scanner, for example, returns 180 readings over a 180 degree field of view. Given that the robot knows its location on the occupancy grid map and the direction the ranger reading occurred, it can update the occupancy grid map. All range-finding sensors have a maximum scan range; if

no obstacle is encountered within the maximum scan range these sensors typically return a maximum distance value.

Borenstein and Koren [1991] developed the Histogrammic in Motion Mapping (HIMM) algorithm to update the occupancy grid cells in a straight line between the robot's location up until the end of the scan. Figure 2.3a illustrates this process. Each cell along the laser scan line up until the end of the scan is decremented in confidence by 1; this makes sense as the ranger beam did not encounter an obstacle. If the end point of the scan is not at the maximum range of the sensor, the end point is increased in confidence by +3 as it is presumed the point is an obstacle. The schema uses *Bresenham's line algorithm* to determine the points between the robot location and the end of the scan line that require updating. Bresenham [1996]'s line algorithm is a computer graphics algorithm used to efficiently rasterize a line on a grid between two points.

Since HIMM increases confidence at a single point, a fast moving robot will tend to report few successive scans for the same location. This can lead to low confidence values for obstacles while the robot is moving. To help grow confidence in the presence of obstacles, HIMM uses a growth rate operator when it updates the obstacle cell at the end of the scan. In addition to increasing the confidence by 3, the end cell is also increased in confidence by $\frac{1}{2}$ the confidence of each neighboring cell.

I use HIMM to update the occupancy grid maps that robots in my implementation build. HIMM is advantageous as repeated scans build confidence in the presence of obstacles. Where an erroneous sensor reading reports the presence of an obstacle, subsequent scans will reduce the confidence in the presence of that obstacle causing it

to fade off the map. This is also helpful to prevent robots from mapping the presence of other teammates as obstacles; subsequent scans of the same location will cause the presence of the robot to fade away in confidence.

2.3.3 Localization and Shared Coordinate Systems

The reactive autonomous control approach introduced in Section 2.3.1 provides a robot with the ability to avoid collisions with obstacles, while navigating to a destination in the environment. Further, the mapping techniques introduced in Section 2.3.2 provide a robot with the ability to build a representation of the environment where the robot (or its team) has been. For a robot to navigate effectively in an environment, it must also be able to determine its location within that environment [Murphy, 2000]. *Localization* is the process by which a robot attempts to determine its location in an environment. According to Murphy [2000], localization is either *iconic*, meaning the raw sensor data is used to determine a robot's location, or *feature-based*, meaning identifiable features such as corners or doorways are identified and used to determine the robot's location.

For a single-agent scenario, a robot could use its own map to provide reference points for determining its location. In a multiple-robot scenario, such as that involved in my work, the multiple robots involved require a common understanding of the meaning of spatial positions they refer to [Wiebe and Anderson, 2009]. For example, each robot creating its own map may have a completely different starting point, yet ultimately will have to combine these maps in order to produce cooperative results. A robot indicating a goal to travel to a specific location will similarly have to refer to that

location using a reference that will have meaning to a receiving agent. This requires some form of shared coordinates (either a common external frame of reference, or a means of translating one agent's frame of reference to another). It also requires that each agent have a reasonable and continually updated estimate of its own location in the environment, so that a given location can be interpreted relative to the robot's own.

The most well-known common frame of reference for geographic information is the Global Positioning System (GPS). GPS provides an absolute, external frame of reference, which a GPS receiver can use to determine the latitude and longitude of the receiver. Since GPS receivers all use the same external frame of reference, robots equipped with GPS receivers have a shared understanding of the meaning of locations they communicate to one another. Even in a forgiving outdoor environment however, GPS still requires sophisticated correction schemes for making accurate large-scale maps with multiple robots [Reid and Braunl, 2011].

GPS is far less forgiving in other environments, for example, such as inside buildings with concrete walls. Radio interference and debris also make GPS unsuitable for robots operating in a disaster zone. Further, in environments such as exploring a mine or another planet, GPS signals will not available at all. Without a common external frame of reference (such as provided by GPS), robots must rely on other methods to determine their location.

A number of probabilistic localization techniques exist which allow a robot to localize itself in its map of the environment. These techniques use a combination of odometry information, and matching the robot's current sensor readings against

the robot's map in order to probabilistically establish the robot's location. These approaches can use only the information provided by one agent (e.g. [Burgard et al., 2011], [Guanghui et al., 2007]), or can take advantage of the perspectives of multiple agents inhabiting the same environment. Approaches in the latter category, such as those of Fox et al. [2000] and Martinelli et al. [2005], use mutual observation between robots to allow robots in close proximity to share data in order to aid in their respective localization efforts. The encounter provides a common reference point which both robots use to establish a common localization. Probabilistic approaches such as these account for noisy and inaccurate sensor data, and are ideal for real-world implementations.

A robot operating in an unknown environment attempting to localize itself, while also building a map of the environment (Section 2.3.2) faces further challenges. Localizing using a map that is under construction is difficult, as the map will be an incomplete representation of the environment. The incomplete map can result in a lack of firm reference points necessary to facilitate localization. This problem is known as *simultaneous localization and mapping* (SLAM), and involves a robot both building a map of the environment it explores, and localizing itself with that map as it operates. Basic SLAM techniques, such as those by Diosi and Kleeman [2004] and Montemerlo et al. [2003], are intended for use by a single robot. The SLAM problem has also been studied extensively when considering multiple robots cooperatively mapping and localizing within an environment (e.g. [Pfingsthorn et al., 2008], [Andersson and Nygards, 2008]).

Aside from probabilistic approaches to establishing a shared understanding of

locations among a team, it is also possible to build one up over time. Wiebe and Anderson [2009], for example, studied three approaches to enable robots to establish shared understandings about locations in the environment. Their first approach uses chance encounters between robots to establish a shared meaning for the location where the robots met. Their second approach establishes shared understandings based on uniquely identifiable areas of the environment which both robots observe. Finally, their third approach recognizes common features such as doorways or hallways, and establishes a shared meaning between robots based on these.

It is often desirable to introduce simplifications to single and multi-robot localization, in order to facilitate research in an unrelated area. Anderson and Papanikolopoulos [2007, 2008], for example, use a deployment pattern where all robots enter the environment one at a time, in the line, providing a shared localization ability through their shared common origin. The RoboCup small size league uses a camera positioned over the playing field to identify and localize the players, using a shared coordinate system [Zickler et al., 2010]. Rooker and Birk [2007] demonstrated their approach to maintaining wireless communication among team members in simulation, assuming all robots have a shared coordinate system and perfect localization.

Since localization and establishing a shared coordinate system between robots is a peripheral area to my focus on adaptive teamwork, I use the built-in localization provided by the Stage simulator to provide each robot with its absolute location within the environment [Gerkey et al., 2003]; this provides a global shared coordinate system and perfect localization, similar to Rooker and Birk [2007]. To simulate conditions necessary for each team to establish its own shared coordinate system, members of

a team begin operation in a common location, similar in spirit to Anderson and Papanikolopoulos [2007]. This ensures robots on a team have a common starting frame of reference. I do *not* assume that all teams start at the same coordinates, however, so teams will create maps based on different origins, and agents on different teams will refer to locations with different designations, requiring the same translations that would be necessary in the real world.

While not requiring a complete multi-agent localization algorithm in my approach is a simplification over the real world, it involves a peripheral area of research upon which adaptive teamwork does not directly depend. I have attempted in my work to leave open the means by which localization could be performed, to allow any approach to localization to be substituted into this. For example, although in my implementation all robots share the same coordinate system through the Stage simulator, I use encounters between robots to emulate the coordinate system reconciliation (similar to Martinelli et al. [2005]'s approach) that would be required if robots localized in a different manner. I assume two robots are only able to reason about communicated locations and information if they have previously encountered one another in the environment, and completed a coordinate reconciliation step (Section 4.7.1.1).

2.3.4 Multi-Robot Mapping

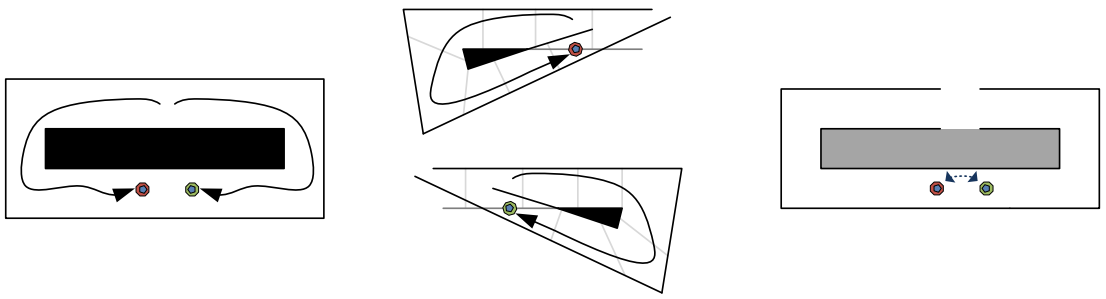
Given techniques for a single robot to construct a map of the environment in which it operates (Section 2.3.2), and techniques robots can use to establish a shared understanding of their location within the environment (Section 2.3.3), it is desirable to construct a single unified map of the environment, based on the maps each individual

builds.

Since my work uses the localization API provided by the Stage simulator, all robots have shared, perfect localization. This means the maps robots build will all be based on the same coordinate system, and map merging becomes a simple operation of combining the values for cells with the same coordinates (Section 4.8.1.2).

In reality, differences in coordinate systems makes map merging a more complex operation than in my implementation. Multi-robot mapping is a well-studied area, and many useful solutions have been studied to account for differences in coordinate systems between robots. An example is Carpin [2008]'s approach to allow multiple robots exploring an area to merge their individual occupancy grid maps into a single combined map. Carpin's approach uses image processing techniques to identify areas where the individual maps overlap, and a series of translations and rotations are then applied to fit the map pieces together.

Imperfect localization can also result in areas where a robot's map ends up over-



1) Two robots navigate a circulator corridor, creating separate maps as a series of connected patches

2) Errors in localization while constructing each robot's map would cause an overlap if the patches were fit together.

3) Encountering one another provides a common reference point on both maps, allowing the maps to be fit together, correcting the overlapped areas.

Figure 2.4: Example of robots merging map patches stored in a manifold.

lapping itself, such as illustrated in Figure 2.4(1) and 2.4(2). A potential solution to this is Howard et al. [2006b]'s approach to simultaneous localization and mapping, which represents maps as a series of map patches overlaid on a manifold structure. The map patches are able to logically overlap in the manifold structure, and when robots encounter one another (Figure 2.4(3)), the robots use the common reference point of the encounter, which allows patches to be correctly merged together, resolving overlaps and loops in the stored map data. Because my simulation uses perfect localization, such techniques are not required, but could be substituted into the framework I present when implemented in a physical environment.

2.3.5 Frontier-Based Exploration

Aside from providing humans with a view of the environment being explored, a map can also be used to help guide the team's exploration efforts.

Yamauchi [1997] developed a frontier-based approach to exploring an unknown space using an occupancy grid map. Frontier-based exploration works by detecting the boundaries between empty explored space and unexplored space on the occupancy grid map. A frontier is an area where a robot could potentially gain more information about the environment; by continually navigating to frontiers the robot is able to continually build and expand its occupancy grid map of the environment. Frontier-based exploration has been shown to be an effective multiple robot exploration technique (see [Yamauchi, 1998; Poernomo and Ying, 2006; Ma et al., 2006; Rooker and Birk, 2007]), and I will be incorporating a frontier-based exploration approach in directing robots to explore the environment.

While there are a host of other interesting low-level problems in mobile robotics that could be addressed in this thesis, the main focus of my work is on high-level team formation and adaptation. I will therefore be using tools within the simulation package I will employ to provide adequate solutions to other problems (such as localization) to allow me to focus my efforts on the core of this thesis.

Chapter 3

Methodology

The main contribution of my thesis is the development of a framework enabling teams of heterogeneous agents to form teams and adapt to changes in team structure, while operating in a complex and dynamic environment. The purpose of this chapter is to describe the approach my framework takes in a domain-independent manner. While I make reference to some of the particular features of domains that make them challenging or dangerous to robots, and draw examples from USAR on occasion, it is important to emphasize that my framework is not intended to be confined to one particular domain. Other domains involving the use of multiple agents with potentially different skill-sets and area coverage in a challenging environment are ideal candidates for my framework. Examples include exploring another planet, clearing a mine field, and performing surveillance. At the same time, my framework and its implementation is inspired by particular aspects of real-world difficulties associated with multi-robot systems, and so I begin with a brief discussion of some of these, before providing an overview of the framework itself.

In my example implementation, agents are embodied as autonomous robots equipped with the necessary capabilities to operate in a disaster environment. In such an environment there is a high degree of probability that robots will become damaged or destroyed making it cost prohibitive to equip every robot with all capabilities. This is common in many challenging domains, and for this reason my framework assumes the capabilities of the robots falls onto a spectrum ranging from poorly equipped to well equipped. I also assume that there will be many poorly equipped robots, and few well equipped robots. The manner in which robots are equipped is domain and mission dependent, but I assume that the poorly equipped robots would be equipped with the least expensive sensors, simplest drive capabilities and minimal computational power. I also assume that robots will be specialized to support the overall mission. In a USAR domain, for example, some robots might be equipped with complicated sensory equipment capable of confirming the presence of a victim in the environment (e.g. a combination of sophisticated life signs detection, such as CO2, heat, as well as vision), while others might be specialized towards penetrating difficult-to-access areas. Clearing a minefield, on the other hand, could use a large number of expendable robots to search for mines and a smaller number of specialized robots to clear the mines the exploring robots find.

My methodology involves mapping agents to tasks in a manner similar in spirit to that of Kiener and von Stryk [2007], but with two important differences. First, task allocation is decentralized. Rather than using a fixed central controller to identify and assign tasks, an agent on a team occupying the role of *team coordinator* is responsible for the management of work to be completed. This responsibility is not fixed, and

shifts between team members to take into account the failure of team members and the arrival of new ones. Second, task allocation in my approach makes use of a formal definition of *roles*, where Kiener and von Stryk [2007] perform task allocation by directly mapping tasks to robot capabilities.

The use of roles promotes efficiency in task allocation in a similar way that roles in human organizations do: they allow assumptions of the abilities of the agents that fill them, thereby avoiding exhaustively considering all underlying skills of all potential agents. Exhaustive matching is impractical in all but the simplest situations such as that implemented by Kiener and von Stryk [2007](Section 2.1). As the number of agents and capabilities increases, the memory and computational cost of an exhaustive search is too great for the available time and the limited computing facilities of an agent. Beyond the computational complexity of matching itself, there is also the issue of decentralization, since no one agent will always have a consistent view of the world. Supporting an exhaustive matching process would entail the cost of making this view consistent as well.

There are some complications inherent with using roles in this manner. Since an agent may fill a role to which it is not entirely suited, it is possible that some tasks are best done by others despite normally being associated with that role. If the task warrants, an assigner can start with the role assumption and spend the extra effort to match beyond that to mission-specific attributes (a more detailed discussion can be found in Section 3.5.3.1). For example, there may be any number of agents with the equipment to perform some specified task, but beyond this it might be more desirable to find the one of these that is closest to the location associated with that task. It

is also possible that an assignee knows it cannot complete a task due to equipment failure, or that its current workload is not conducive to taking on another task. In such a situation, it can inform the assigner that it cannot complete the task and the assigner will attempt to assign the task to another agent. The task assignment process assumes communication is unreliable and that assignees can fail at any time; this ensures that task assignment does not get stuck waiting for an agent due to communication or equipment failure.

Over the course of operation in a difficult and dynamic domain, it is reasonable to expect that the nature of a team will change, both by losing members (e.g. disability, recruitment to other teams) and gaining them (encountering new agents, meeting and exchanging members with other teams). An agent could easily be destroyed by rolling over a mine while clearing a minefield for example, or become stuck on loose ground while mining or exploring another planet. Equally, another jurisdiction may arrive at a disaster site with new agents that can join those already exploring for human victims. My work provides provisions for agents to change roles as necessary (possibly changing teams as well) in these situations. This means, for example, if the agent occupying the team coordinator role fails, another agent can step in to fill that role, or other agents may recognize the lack of possibilities for leadership and seek out another team, causing the restructuring or complete breakup of the team itself. Further, an agent that becomes separated from its team is able to switch to a new role and join another team it encounters.

Similar to Odell et al. [2003], in my framework a *desired team* is defined by the number of agents and the roles they fill on the team in order to complete the mission.

The desired team defines the desired minimum and maximum number of agents filling each role; these limits are chosen at design-time to ensure that the team can handle the expected mix of tasks present in the environment.

The following sections describe the operation of my framework from the perspective of the team as a whole and the individual agent. First, an overview of the major interactions within a team are presented in order to provide a high level overview of what my framework does to support the maintenance of teams (Section 3.2) and the distribution of work amongst teammates (Section 3.3). Next, the relationship between attributes, tasks and roles is explained to provide a basis for how my framework accomplishes its major operations in a distributed manner (Section 3.4). Finally, Sections 3.5, 3.6 and 3.7 provide a detailed description of the algorithms and interactions of my framework in terms of the concepts of attributes, tasks and roles.

3.1 Framework Overview

To fully understand the operation of my framework and how it supports distributed team maintenance and task management, it is necessary to describe the high-level operations and interactions the framework supports. These high level descriptions provide an understanding of what the team as a whole does, as well as the major interactions between agents using my framework. Subsequent sections answer the question of how my framework accomplishes these operations.

Figure 3.1 shows the operation of my framework from the perspective of the team, while Figure 3.2 shows how this translates to individual agents on a team. The primary objectives of my framework are *team maintenance* and *task management*.

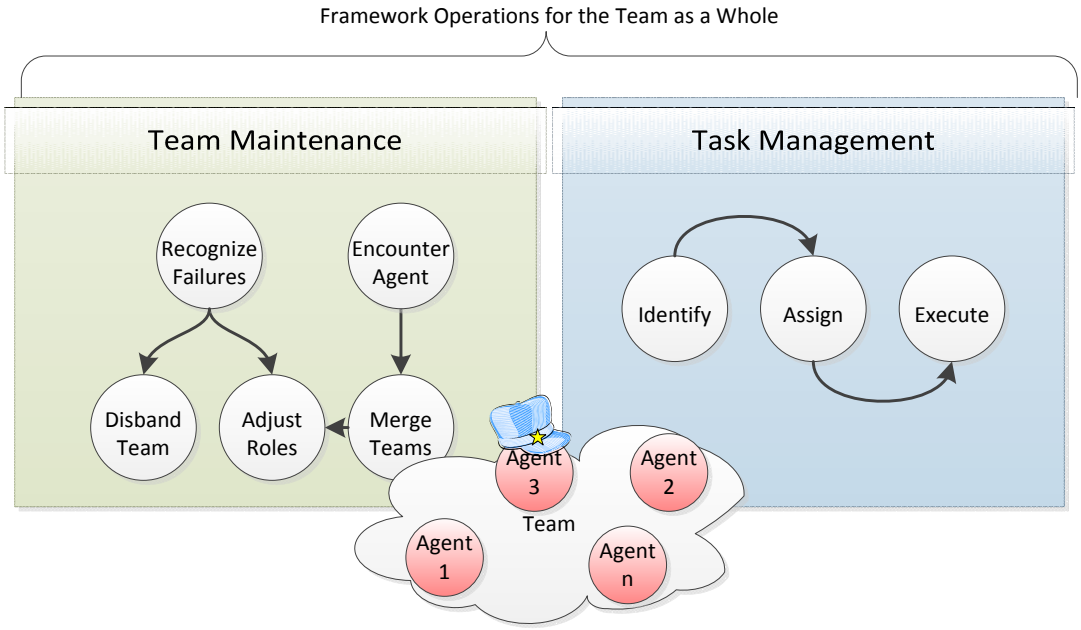


Figure 3.1: The operation of my framework, from the perspective of the team as a whole.

The team maintenance operations focus on the formation and maintenance of teams, and their reaction to changes in team structure. The task management operations focus on the identification and assignment of tasks to the most suited members of the team. Since roles in my framework are defined in terms of the tasks normally expected of them, a deficiency in team structure can result in a situation where task assignments are suboptimal; the team maintenance operations attempt to correct deficiencies in team structure so that a higher level of suitability can be achieved when assigning tasks.

The first team maintenance operation a team performs (Figure 3.1) is to recognize and respond to agent failures. In a disaster environment, for example, robots could be damaged, destroyed, or become separated from their teams. In response to a

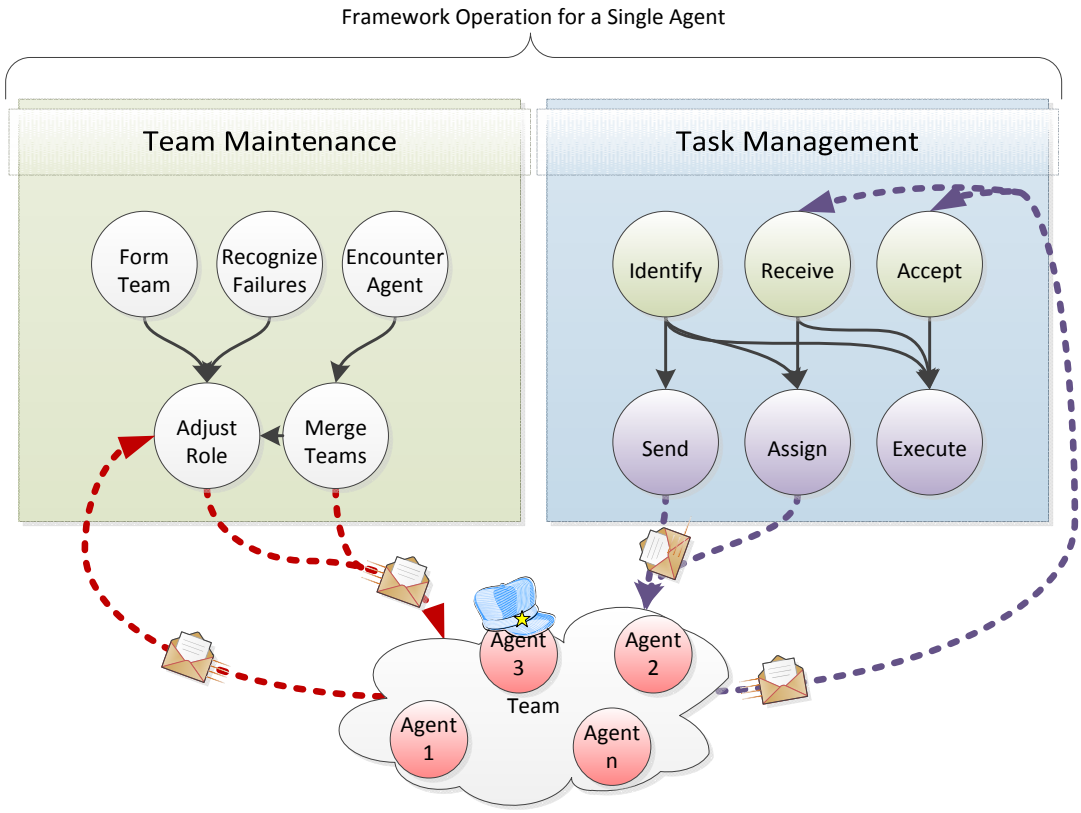


Figure 3.2: The operation of my framework, from the perspective of an individual agent.

recognized failure, the team will either adjust the roles of the remaining agents on the team, or disband the team completely. This is accomplished in a distributed manner at the agent level (Figure 3.2). Agents actively monitor the status of their teammates and when they determine a failure has occurred are able to initiate an adjustment in their own role accordingly.

Team maintenance also occurs when agents encounter one another in the environment. From the perspective of an established team (Figure 3.1), encountering another team provides the opportunity for teams to merge or re-distribute the agents between the teams. Robotic agents on different teams in close physical proximity to

one another, for example, are able to visually identify each other and take advantage of short-range communication between each other. My framework provides the agents with the ability to learn about each other's teams and to exchange agents between teams in order to improve the operating conditions of both.

Aside from providing established teams with the ability to merge / re-distribute agents, team maintenance also supports the formation of new teams from individual agents operating in the environment as they encounter one another. In my framework, individual agents are not treated differently from agents operating as a part of a larger team. Such agents are the sole member of the team and fill the role of team coordinator. A lone agent well suited to filling the team coordinator role will tend to retain that role and build up its team as it encounters agents, where a long agent poorly suited to filling the team coordinator role will tend to cede that role for a more suitable one on another team it encounters.

Thus, using my framework, teams are a fluid aggregation of agents, where the agents switch roles within the team and change teams as necessary to make the best use of the available agents. As agents change roles and teams, the overriding goal is to form stable teams that meet the definition of a desired team as closely as possible.

From the perspective of an existing team using my framework, a collection of *task management* activities are performed, as shown in Figure 3.1. As the team identifies tasks, these are assigned to the best suited team members who carry them out. In my framework, tasks are considered to be atomic units of work that a single agent will complete in isolation. As shown in Figure 3.2, the identification of tasks occurs at the agent level. As agents identify tasks, they are sent to the agent filling the team

coordinator role. Each task is assigned to the best suited agent, who carries it and reports the results back to the team coordinator.

Where the current team complement allow the feasible assignment and completion of a task (i.e. no agent could possibly complete it, even in a mediocre way), the task is retained for future completion, as long as the task remains relevant. Future assignment attempts ensure that the task has an opportunity to be carried out when the team complement supports it.

With a high level overview of the interactions that occur in my framework established, the following Sections 3.2 and 3.3 describe the team maintenance and task management interactions shown in Figures 3.1 and 3.2 in more detail to provide a better understanding of what my framework does. This provides an appropriate foundation for the detailed discussion of how attributes, tasks and roles operate and the detailed algorithms used to accomplish team maintenance and task management, which form the remainder of this chapter.

3.2 Team Maintenance

Before discussing the operation of my framework further, it is necessary to have some understanding of the high-level relationship between tasks and roles in my framework (more formal descriptions of the representations of these will appear in Sections 3.4.2 and 3.4.3, with further implementation details in Chapter 4). A task specifies the set of capabilities an agent must possess in order to take on a task. The task requirements are formulated such that it is possible to calculate a suitability value for any one agent with a unique set of capabilities to complete that task (Section 3.4.2).

Roles, on the other hand, are defined in terms of the set of tasks an agent filling that role will normally be expected to be able to perform. This means the set of capabilities an agent requires to fill a role is ultimately determined by the tasks normally expected of that role. Since task requirements are formulated such that an agent's suitability to execute a task can be calculated, it follows that an agent's suitability to fill a role is the aggregate of its suitability to complete each task normally expected of the role.

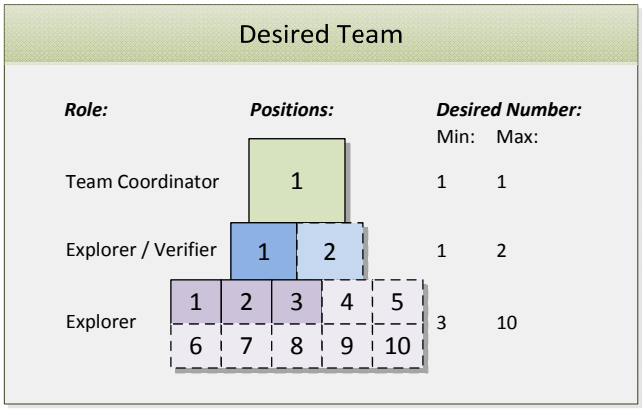


Figure 3.3: The definition of a desired team describes the roles and number of agents filling each role. The definition defines a minimum and maximum number of agents desired in each role.

The concept of a *desired team* is central to team maintenance in my framework. Figure 3.3 depicts the concept of a desired team using the configuration from my example USAR implementation. The desired team identifies the number and roles of the agents that make an effective team. It describes the team composition that the team maintenance operations aim to achieve. The desired team composition is dependent on the domain and the capabilities of the agents and is determined by a

human in advance of operation. The desired agent mix is chosen to ensure the agents coordinating on a team are able to effectively complete the team mission (future work could have the team learn or adjust the desired agent mix as it operates). Within a team, *team coordinator* is a special purpose role responsible for directing the overall operation of the team. The team coordinator role designates the responsibility of assigning tasks to a single agent and provides a single coordination point where the results of tasks are collected.

The goal of team maintenance is to ensure agents fill roles on the team which result in the team approximating the definition of a desired team as closely as possible. Adjusting the roles agents fill on the team can occur when an agent failure is recognized, or when a new agent is found and a team merge and redistribution occurs.

3.2.1 Recognizing Failures

Carlson and Murphy [2003] studied the reliability of research and field robots and found equipment and software failures to be very common during operation. In an environment hostile to the operation of robotic equipment, such as the exploration of another planet or in the aftermath of a disaster, it is essential not only to use robust equipment, but to ensure that the operation of a system is resilient to such failures.

In my framework, where an agent is able to recognize that it has become disabled in some manner and cannot continue operation, it informs its teammates who record the failure. In my example USAR implementation, for example, agents that become stuck and are unable to free themselves inform their teammates, who can then adjust their roles accordingly.

Although self-recognition of failures is an active area of research (e.g. see Verma et al. [2004]), it is often not possible for an agent to determine itself if it has failed. In my USAR implementation, agents can become lost when they wander out of wireless radio range of their teammates or suffer from equipment failure. In a real-world scenario, other failures are possible due to structural collapses, fires or flooding. Even very basic failures can sometimes be hard to determine: the result of being in a dark room can look very much like vision failure, for example. My framework helps agents detect scenarios where teammates have become separated from the team, or have suffered a complete failure, by tracking the last time an agent has heard from the other agents on its team. Agents that have not been heard from in a specified period of time are considered to no longer be a member of the current team. In my example implementation, agents rely on monitoring the wireless communication messages between each other to track when each teammate has been heard from. My framework primarily concerns itself with coping with the complete failure of agents; establishing a more complete failure model able to compensate for partial agent failures would be useful future work.

A team recognizes the failure of agents and responds to these failures by adjusting the roles of the remaining agents accordingly. Using my framework, an agent is responsible for determining if any of its teammates have failed, and adjusting the role it fills on the team in response (a *Role Check*). Agents change roles in an attempt to ensure the team matches the definition of a desired team a closely as possible. Further, role changes ensure that the team coordinator role is filled by the best suited agent. The recognition of failures is implicit in that an agent adjusts the role

it fills on their team in response to deficiencies it detects in its current view of the team complement. An agent periodically determines the best role for it to fill and implements a role change if the role is different from the one it currently fills. The agent informs its teammates of the change, which can trigger subsequent role checks to occur for other members of the team. To help prevent two agents from simultaneously determining their role and implementing the same role change, a random interval is added between role checks to minimize the chance of this occurring. It is important to note that each agent is responsible for changing roles on its own, and can do so without the explicit "permission" of any other agent. It is, however, also possible for an agent to be instructed to change roles by another agent such as in the case where a team merge occurs (Section 3.2.2).

Since I assume communication is unreliable, messages used to communicate the current team structure are not guaranteed to reach every member of the team. This results in agents on a team having a view of the overall team structure that may differ from the actual team structure. In periods where communication is more reliable it can be expected that each individual team member will have a more accurate view of the team structure, while in periods of poor communication it can be expected that the individual's view of the team structure diverges from reality. As a result, an agent can potentially initiate a role change that is sub-optimal due to having an incomplete view of its team structure. These deviations are expected in the context of the team of a whole, and compensated for in part by defining a desired team in terms of a range of agents filling each role. Restricting the role checks each agent performs to occur on a periodic basis also helps to prevent the team from continually

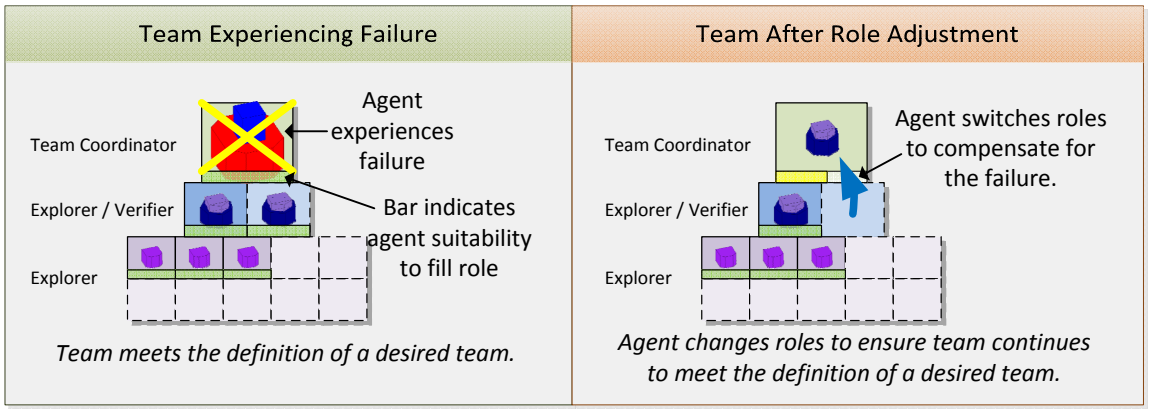


Figure 3.4: A team adapts to the failure of an agent occupying the team coordinator role.

restructuring itself as its view of the team structure changes. Such delays between reevaluation are often useful in multi-agent systems, to avoid unnecessary oscillation. Anderson and Baltes [2006], for example, note that it can be used both for choosing which opponents to block on a field, and what the most appropriate play should be in robotic soccer, while avoiding explicit communication between agents.

Figure 3.4 uses the roles and agent types from my example USAR implementation to illustrate the role adjustment that occurs in response to the failure of an agent. The specific details of the tasks used to define these roles, as well as differences in the different robot types are presented in Chapter 4; they are discussed here at a high level to provide an example of how a team responds to failures. Prior to the failure, Figure 3.4 shows agents filling roles in quantities deemed ideal given the definition of a desired team. The green bars below each agent show the degree of suitability for each agent to fill their current role. An agent's suitability to fill a role is calculated by matching the individual agent's capabilities against the requirements of the tasks

normally expected of the role (Section 3.4.3.1). One of the agents filling the *explorer / verifier* role recognizes the failure of the agent filling the team coordinator role and adopts that role in response. Others with similar suitability will be unlikely to re-examine their roles at the same time, and so conflict is unlikely. In the event two agents attempt to adopt this role at the same time, the next role check by either agent would rectify the situation. Where the agents are not equally suited to fill the role, the less suited agent would choose a new role during the role check, and the better suited agent would remain in the role. Where both agents are equally suited, the agent with the highest robot identification number (Section 3.4.1) chooses another role. Since the agent adopting the team coordinator role is not fully suited to fill that role, it can be expected to do so in a reduced capacity compared to the failed agent. The team, however, can continue operation despite this. When an agent better suited to fill the team coordinator role joins the team, the less suited agent will cede the role to the better suited one in a similar fashion. Since agents do not perform role checks at the same time, it is unlikely that multiple agents will adopt the same role simultaneously. However (as explained above), should this occur the agents will correct the situation during a subsequent role check.

Because of the assumption that there will likely be few very-well equipped agents, the failure of one such agent, as in this example, will often result in a less suitable agent taking its place at least temporarily. In Figure 3.4, the *explorer* role is filled by agents that are so simplistic in nature as to be entirely unsuited to fill the team coordinator role. In a scenario where only agents of this type remain on the team after the failure of a well suited team coordinator, one would fill the role of team

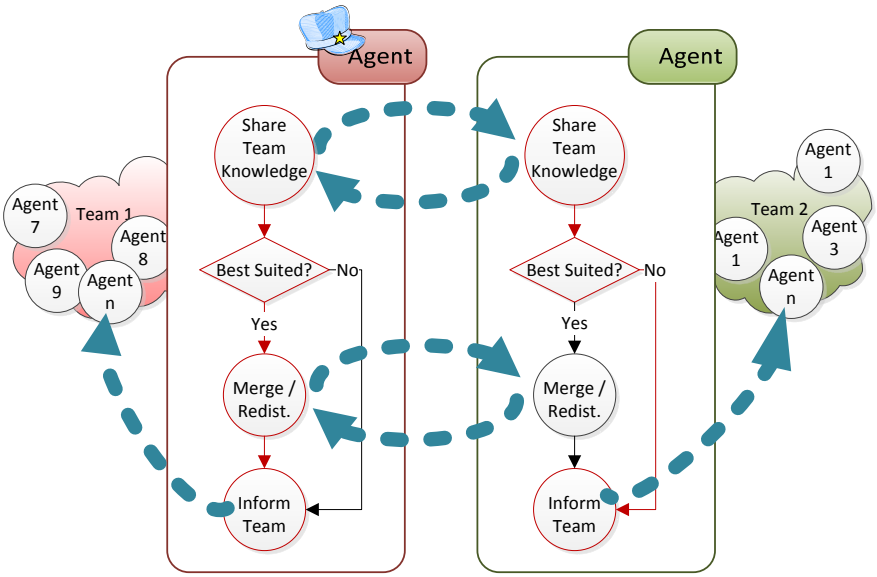


Figure 3.5: Two agents encounter each other and perform a merge on behalf of their respective teams.

coordinator. Because the agent type lacks the ability to perform any meaningful team coordinator activities, the result would be a gradual degradation in the performance of the team. Without a team coordinator to assign meaningful work, the agents will eventually resort to their default idle behaviour of wandering. This will result in the agents wandering away from one another and each agent not having heard from its teammates in some time. At this point, it will form a new team with itself as the only member, making it a likely candidate to be picked up by an established team that encounters it in the future.

3.2.2 Encountering Agents

Teams encountering one another have the opportunity to exchange teammates so both teams better meet the definition of a desired team, or to combine them

into a single team (the simplest case of which is picking up a single individual that has no other teammates). Using my framework, this occurs when individual agents on different teams encounter one another. In my example USAR implementation, agents may sense one another's presence while performing their own tasks, by virtue of being in close physical proximity. During such encounters the two teams also have the opportunity to share mission-specific data between themselves. In my example USAR implementation, an encounter between teams provides the opportunity to share information about the area that each team has explored, and the location of any potential victims that have been found so far.

Figure 3.5 shows the operations that occur when two agents encounter each other and perform a merge and redistribution of their respective teams. The two encountering agents first exchange information about their teammates. This includes the roles each agent occupies, as well as a description of the capabilities of each agent. The two agents encountering each other act as representatives for their teams and negotiate the merge and redistribution on behalf of their respective teams. There are computational expectations for performing this task that are similar to those of the team coordinator, and so the two agents encountering one another must determine their respective degrees of suitability to this task. The agent with the best suitability to this will perform the merge and redistribution (if necessary), provided the agent's abilities meet a defined threshold (since neither encountering agent may in fact be particularly well-equipped). If neither agent qualifies, the merge and redistribution does not take place. In my example implementation, for example, only two of the agent types are qualified to perform the merge and redistribution operation.

Intuitively, the agents on both teams filling the team coordinator role would be the ideal agents to perform the team merge and redistribution operations. However, unreliable communication and potentially large distances between the encountering agents and their respective team coordinators make this undesirable in practice. Performing the negotiation over long distances is problematic as unreliable communication makes failure likely. Further, moving the team coordinator agent closer would prevent that agent and the encountering agents from completing other useful work in the interim. Using the encountering agents ensures the merge and redistribution is performed by agents local to one another. This is a trade-off, as the agent performing the merge will potentially be less suited from a computational and memory standpoint than the team coordinator agent from either team. Further, there will likely be some discrepancies in the agent's personal view of the current team complement compared to the team coordinators'. However, it is not unreasonable to expect just as large (or larger) discrepancies due to communication issues inherent in negotiating with the actual team coordinators from a distance.

The representative agent chosen to perform the merge and redistribution combines information regarding the other representative agent's teammates with information about its own teammates. It uses this information to form new teams by iteratively finding the best suited agent to fill a role on the new teams and assigning that agent to its best suited role and team (details of this process are found in Section 3.7). The definition of a desired team guides the agent selection process to ensure that the most important roles where the desired minimum requirements have not yet been met are filled first. The result will be either a single combined team, or two teams with agents

swapped between teams to ensure both teams match the definition of a desired team. Where both teams already meet the definition of a desired team prior to the merge and redistribution operation, no changes are made to either team.

After the merge and redistribution operation, the other agent is informed of the role and team changes that its teammates must implement as a result. Both encountering agents are then responsible for informing their own teammates of the changes that must be implemented. Details of the team merge and redistribution operation can be found in Section 3.7. Example scenarios illustrating this process are available in Section 3.7.2.

3.3 Task Management

Given an established team, my framework supports the identification, assignment and completion of tasks. As shown in Figure 3.2, every agent plays a part in the task management process. Each agent is responsible for identifying tasks in the environment to the degree that its sensory equipment allows it to do so. Where an agent is able to carry out a task it identifies by itself and its workload permits, it will carry out the task. Otherwise, the agent sends tasks to the member of its team filling the team coordinator role. The team coordinator receives the task and will assign it to another member of the team for completion.

Task assignment first uses roles as a shortcut to determine the agent(s) to consider assigning the task to (a detailed description of role-based task assignment is in Section 3.5.3.1). Since a role is defined in terms of the tasks normally expected of it, requests are sent only to those agents filling a role normally expected to be able to complete

the task. Agents respond with an estimate of their own suitability toward completing the task. The task is then assigned to the best suited agent.

When task assignment request messages are sent to agents, it is possible that poor communication conditions prevent the requests from being received by every agent. It is also possible that the responses agents send back to the requests are not received by the task assigner. No attempts are made to re-try or re-send messages used in the task assignment process; agents impacted by communication failures are thus unlikely to be assigned the task. This means although there may potentially be many agents whom are expected to be able to complete a task, only a subset of these agents will participate fully in the negotiation. This distributed model assumes that in such cases, a task left undone will be rediscovered by another agent.

Where a task fails to be assigned after using the role-based task assignment operation, due to communication difficulties or a lack of suitable agents, a second task allocation process is used. The second task allocation process operates similarly, except without using roles. In the second process, called *exhaustive task assignment*, a broadcast message is sent to all agents on the current team in range. The responses and assignment occurs in the same manner as when roles are used as a shortcut. The second task allocation process provides agents impacted by communication failure in the first process with another chance to participate in the task assignment process. It also broadens the pool of agents able to participate in the assignment process to account for the fact the task assigner may have an incomplete view of the roles each member of its team currently fills.

After carrying out a task, the agent completing the task reports the results to the

agent filling the team coordinator role. Reporting task completion results to the team coordinator ensures that agent has the most complete picture of the work completed in order to aid in prioritizing future work.

The preceding sections introduced the major operations of my framework and provided an overview of its operation in terms of the team as a whole and the individual agents that make up the team. The following section explains the concept of attributes and expressions and how they are used in determining the agents best suited to carry out tasks. Finally, the manner in which roles are defined in terms of the tasks expected of them is explained and how this relates to the definition of a desired team.

3.4 Attributes, Roles and Tasks

In my framework, *attributes* are used to describe the capabilities of agents and the requirements of tasks. Figure 3.6 (explained in detail in the subsections that follow) shows how attributes are used to form expressions that describe the capabilities an agent requires in order to carry out a task. Roles are defined in terms of the different types of tasks normally expected of an agent occupying that role. The roles and number of agents expected in each form the definition of a desired team. Finally, the capabilities of an agent determine the attribute values it has, which determines the role it will fill on a team.

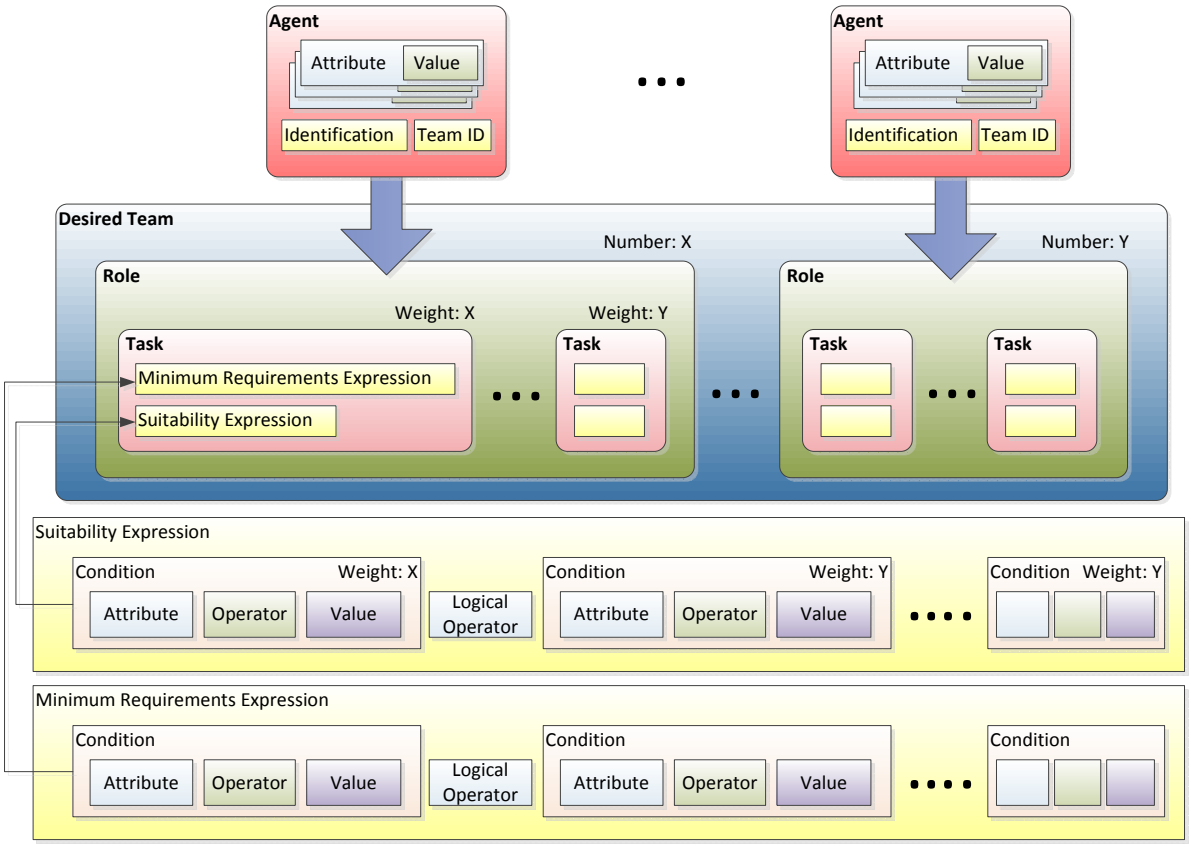


Figure 3.6: Overview of how attributes make up the knowledge necessary to define expressions, tasks and roles.

3.4.1 Attributes

For agents to meaningfully coordinate with one another about work to perform, they require a common understanding of the properties of themselves and others. As shown in Figure 3.6, attributes describe the capabilities and properties of agents relevant to the completion of tasks. Agents also have an identification and team ID property. The agent identification is a unique number identifying each agent in the mission, and the team ID is the identification number of the team the agent currently is a member of.

The choice of attributes is mission specific and shared between all agents using my framework in advance of operation. Examples of attributes might include, for example, the diameter or a robotic agent (useful for determining if it can fit into a space), whether an agent possesses a specific sensor type or has a particular locomotion capability. In my framework, each agent maintains a list of attribute-value pairs that describe its capabilities. For the robotic agents in my example implementation, this includes physical properties, sensory capabilities, and computational facilities. These values are determined at agent design-time and provide the necessary information for an agent to describe its capabilities to other agents. Agents share their attribute-value lists with their teammates in order to form a distributed view of the overall team composition. This shared knowledge provides a basis for agents to reason about who should carry out tasks and which role the agent should best fill at any one time.

The attributes that describe an agent can have boolean or numerical values, as best suits the nature of the attribute. Boolean attributes are useful to indicate if an agent has a specific feature. The *HasMap* attribute, for example, identifies if an agent maintains a map of its surroundings. Numerical attributes are useful to describe physical properties of an agent. An example might be an attribute that describes the width of a robotic agent in meters. Numerical attributes can also be used to represent enumerated properties. For example, the physiology of a robotic agent could be described such that 1=tracked, 2=wheeled, and 3=humanoid. Section 4.4 describes the specific agent attributes I use in my example implementation.

3.4.2 Tasks

In my framework, tasks represent units of work that contribute towards the progress of a mission. These are divided into types based on the kinds of work that the overall framework will carry out. These types are domain dependent and defined and represented by humans when a domain is chosen. In contrast to Kiener and von Stryk [2007] where multiple agents coordinate on a single task, in my work a single agent carries out a task in isolation. Examples of task types in my example implementation include *explore frontier*, *find victim*, and *confirm victim*.

The *minimum requirements* and *suitability expression* elements shown in Figure 3.6 determine the attributes an agent must possess in order for it to be able to carry out a task of that type. The *priority* element describes the importance of one type of task in relation to the other types in the mission.

Agents recognizing work to be done create tasks as instances from these task type descriptions. It is possible the agent that creates a task may not have the capabilities required to complete it. Defining a task in terms of its minimum requirements and a suitability expression provides information agents require to reason about whether they can or cannot carry out a task themselves, and subsequently who is best suited to carry out a task given the current team complement (Section 3.5.2 describes how task assignment occurs using these attributes).

3.4.2.1 Minimum Requirements

The *minimum requirements* for a task type, illustrated in Figure 3.6, defines the absolute minimum capabilities an agent must possess in order to be able to carry

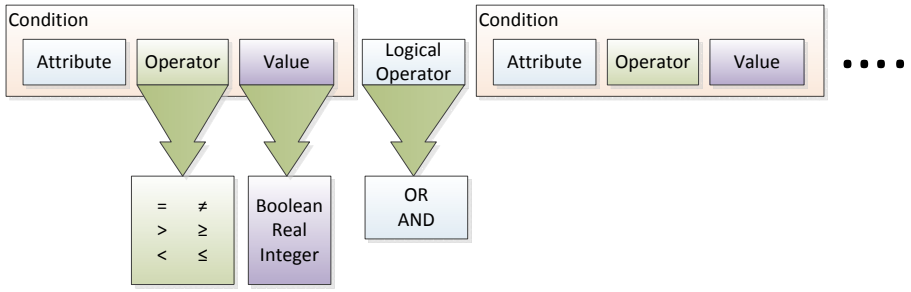


Figure 3.7: Expressions are built from conditions separated by logical operators.

it out. As shown in Figure 3.7, expressions are constructed of *conditions* separated with logical operators. Each condition consists of an attribute, a comparison, and a value. Conditions support attribute / value comparison using the operators $>$ (greater than), $\geq$ (greater than equal), $<$ (less than), $\leq$ (less than equal), $=$ (equal), $\neq$ (not equal).

Equation (3.1) illustrates a simple minimum requirements expression (M) from my example USAR implementation. To be able to carry out this task, the $HasRanger$ or $HasLaser$ attribute of the agent must be $true$ and the $Clearance$ attribute must have a value > 0.5.

$$M = (HasRanger = true \vee HasLaser = true) \wedge Clearance > 0.5 \tag{3.1}$$

3.4.2.2 Suitability Expression

A task type's *suitability expression*, shown in Figure 3.6, is used to calculate the *suitability* of an agent to carry out that type of task. The suitability expression is used to calculate a numeric value that describes the degree to which an agent is suited to carry out a task, and allows for reasoning about which agent is best suited to carry

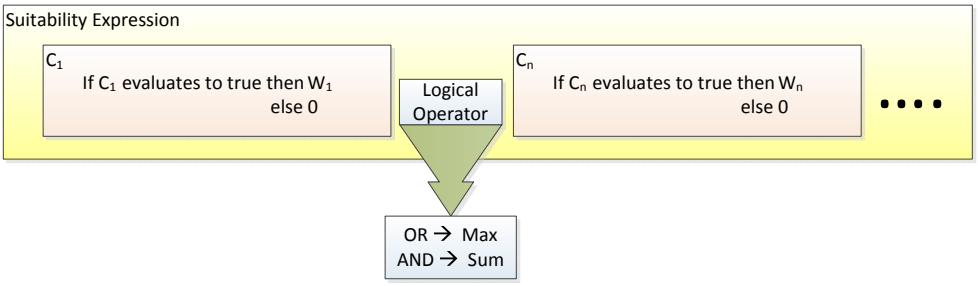


Figure 3.8: When condition C_i evaluates to true, it generates a value W_i. Where the *and* operator separates conditions, the values are summed. Where the *or* operator separates conditions, the result is the maximum value.

it out.

Similar to the minimum requirements, the suitability expression consists of conditions combined with logical operators. Each condition is weighted to reflect its contribution to the evaluation of the suitability expression against a set of agent attributes. Where an agent does not meet the minimum requirements of a task, the suitability expression always evaluates to zero.

As shown in Figure 3.8, conditions C_i in a suitability expression that are satisfied (evaluate to true) generate a value equivalent to their weight W_i, while conditions C_i that are not satisfied (evaluate to false) generate a value of zero. The evaluated weights for conditions combined with the *and* logical operator are added together. For conditions combined with the *or* logical operator, the result is the maximum weight of the evaluated conditions. The net effect is that conditions combined with the *and* operator increase the suitability for every condition that is met, while the *or* operator acts to increase suitability based on the most valuable condition met.

Equation (3.2) illustrates the suitability expression (S) for the *Explore Frontier* task in my example implementation. These tasks are best suited for completion by the smaller, low-cost robots as their expendable nature makes them well suited to operation in unknown environments. The ($HasLaser[10] = true \vee HasRanger[15] = true$) sub-expression favours an agent with a low-cost sonar ranger. Similarly, the ($Expendability[60] > 0.5 \vee Expendability[10] > 0.2 \vee Expendability[0] > 0.01$) sub-expression assigns increasing suitability to more expendable agents. Finally, the weights from the sub-expressions and the $RobotRadius[20] < 0.2 \wedge HasMap[5] = true$ portion of the expression are added together to determine the overall suitability.

$$\begin{aligned} S = ((&HasLaser[10] = true \vee HasRanger[15] = true) \wedge \\ &(Expendability[60] > 0.5 \vee Expendability[10] > 0.2 \\ &\vee Expendability[0] > 0.01) \wedge \\ &RobotRadius[20] < 0.2 \wedge HasMap[5] = true \end{aligned} \tag{3.2}$$

3.4.2.3 Task Priority

The *priority* attribute of a task type (see Figure 3.6) indicates its relative importance in relation to other task types. A higher priority in one task relative to another means the higher priority task should be carried out first. While a task is carried out, it may be pre-empted if a higher priority task arrives. In my example implementation confirming a hypothesized victim, for example, takes priority over exploring a frontier.

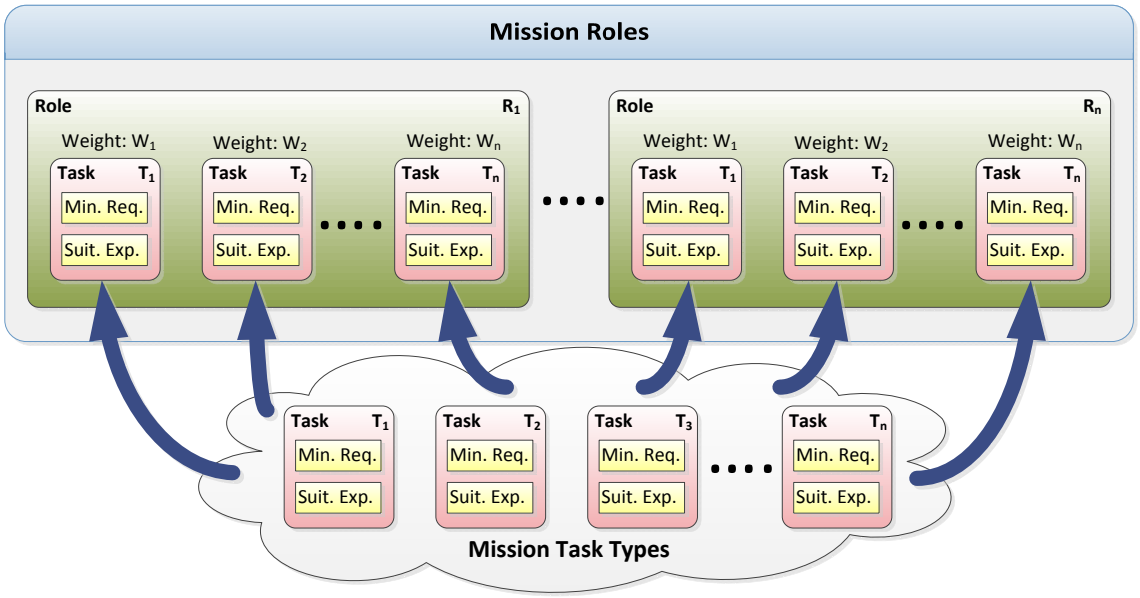


Figure 3.9: A role is defined by the tasks normally expected of an agent filling it.

3.4.3 Roles

As illustrated in Figure 3.6, a *role* is defined by the tasks that any robot occupying the role is normally expected to be able to perform, such that each role has a list of the tasks expected of it. Thus, roles are a heuristic description of the types of work a robot filling the role will normally carry out. Roles are heuristic both in the sense that an agent occupying a role may not be able to complete all of those tasks, and in the sense that the tasks listed may be a representation of the types of work normally expected of the role. This is in contrast to previous work where roles have been used to define other individual parameters, such as the area to cover in soccer [McMillen and Veloso, 2006; Stone and Veloso, 1999], or the tasks that one is permitted to perform [Zambonelli et al., 2003].

A role R_i can be defined as in Figure 3.9. Where $M = \{T_1, T_2, \cdots, T_n\}$ are the

task types expected in the current mission, a role R_i is defined by the subset of the mission tasks that are normally expected to be carried out by an agent filling the role R_i. Each expected task is assigned an importance weighting that determines how important it is that an agent filling the role can carry out that type of task. When determining the suitability of an agent to fill a role, the importance of the task acts to weight the agent's suitability for the task in the overall role suitability calculation (see Section 3.6.1). Tasks that an agent is expected to encounter the most often are assigned a higher importance weighting.

3.4.3.1 Suitability of an Agent to Fill a Role

Given a role R_i is defined in terms of the tasks $\{T_1, T_2, \cdots, T_n\}$ normally expected of a robot filling that role, it is possible to determine a numerical suitability for an agent to fill R_i given the agent's attributes.

Algorithm 1 Determining the suitability of an agent to fill a role.

Require: $T = \{T_1, T_2, \cdots, T_n\}$, the tasks normally expected of role R

Require: $W = \{W_1, W_2, \cdots, W_n\}$, the weights of tasks R expected of role R.

Require: $A = \{A_1, A_2, \cdots, A_n\}$, the attributes and values that describe an agent

$S = 0$

for all $T_i \in \{T_1, T_2, \cdots, T_n\}$ **do**

$E_i \leftarrow$ evaluate suitability expression of T_i against A

$S = S + E_i W_i$

end for

return S, suitability of agent to fill role R

Algorithm 1 shows the process used to determine the suitability of an agent with attributes and values A to fill a role R. From Figure 3.9, $T = \{T_1, T_2, \cdots, T_n\}$ defines the set of tasks that are normally expected of role R. Each task $T_i \in R$ has an importance weighting W_i that determines the importance of the task in relation to the other tasks expected of the role.

For each task T_i, the suitability expression of the task is evaluated against the attributes and values in A (see Section 3.4.2.2 for more on evaluating suitability expressions of a task). This value is multiplied by the importance weighting W_i for the task and added to S, the overall suitability of the agent to fill the role.

The result is a sum of the agent's suitability to carry out each task expected of the role, weighted by the relative important of the tasks in the role.

3.4.4 Desired Team

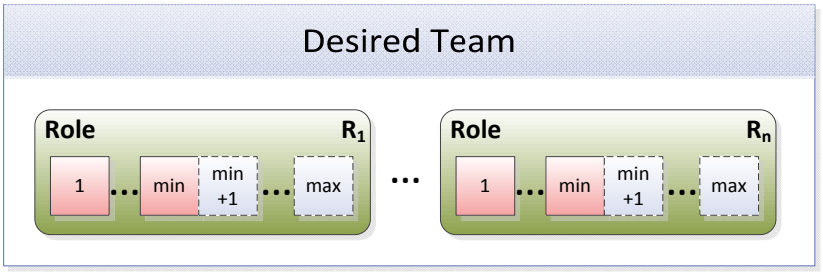


Figure 3.10: A desired team is defined by the roles and range of agents desired in each role.

Given the roles $\{R_1, R_2, \cdots, R_n\}$ defined for the current mission, a desired team is defined by the range of agents that should be present on a team to perform the mission effectively. As shown in Figure 3.10, for each role R_i a minimum and maximum

number of agents are specified. The minimum values establish the absolute minimum complement of agents filling roles necessary to accomplish the mission. In any domain there is always a point of diminishing return after which the addition of more agents results in a minimal performance gain. Rosenfeld et al. [2006] found that in any physical domain there will be an upper limit on the number of agents that can operate in close proximity before interference between agents begins to inhibit overall group performance. Computational, memory and communication limits of agents will also impose constraints on the maximum size of a team in terms of the effort required to manage the team and process the results of less capable agents. The maximum values for all roles collectively enforce an upper bound on the number of agents filling the roles on the team.

3.5 Task Management via the Task List

Section 3.3 introduced the manner in which my framework identifies and assigns tasks to agents on a team in a distributed manner. Section 3.4 discussed roles and how they are defined in terms of the types of tasks normally expected of them. Further, it described how the requirements of tasks are expressed in terms of minimum requirements and suitability expressions. This section brings together these concepts and explains in detail how task management occurs via the *task list*. It also describes in detail how roles are used in the task assignment process and how task assignment works.

Using my framework, each agent maintains a prioritized list of tasks. This task list allows an agent to track new tasks it discovers in the environment, participate in

the negotiation of new work to carry out, and assign tasks that it cannot carry out by itself to other agents. Further, the task list provides a source of tasks for the agent to carry out itself.

3.5.1 Carrying Out Tasks

Tasks on the task list are maintained in order of task priority, followed by the time the task was added to the list. An agent will carry out the highest priority, oldest task first. If while carrying out a task, a higher priority task arrives than the current one, the agent suspends the current task and begins carrying out the task with the higher priority. The suspended task remains in the same position on the agent's task list and will resume once the higher priority task has been completed. This helps to ensure that more important tasks receive prompt attention. In my example USAR implementation, for example, a robot should stop exploring an area if a task arrives requesting it to confirm the presence of a victim in the environment.

If an agent runs out of tasks, it will carry out an *idle task*. The idle task specifies the default behaviour of the agent in the absence of specific work to complete. In my example USAR implementation, for example, in the absence of a victim to confirm or a specific area to explore, agents fall back to a random wander behaviour. This helps ensure that in periods of unreliable communication or when a team suffers a failure of its team coordinator that agents have a task that is useful in their domain to fall back upon until they are able to receive more meaningful work. In the case of wandering, the agent can also discover new tasks while performing this fallback behaviour, so this will likely be useful in a broad range of domains.

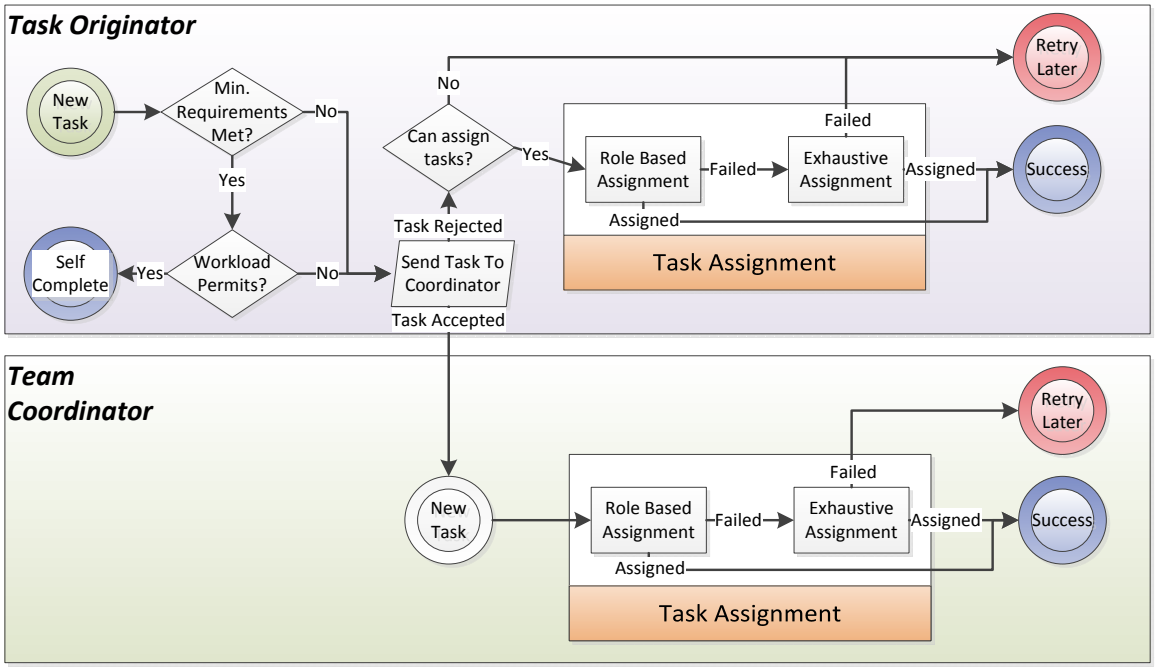


Figure 3.11: Task assignment processing.

3.5.2 Adding New Tasks to the Task List

When an agent identifies a new task that requires completion, it adds it to its task list. At this point, a determination is made if the agent will carry out the task itself or send it to another agent for completion (see Figure 3.11). The agent first determines if its own attributes satisfy the task minimum requirements (See Section 3.4.2.1). If satisfied, the agent determines the number of tasks on its list of the same type that is to carry out. If the count is less than a configured threshold (I use 5 in my work), the agent will mark the task for completion by itself. The task count threshold helps to spread the workload among all the agents on the team. The threshold applies per task type to ensure that an agent will complete a balance of different task types. For example, in my implementation it would be undesirable for an agent that can confirm

victims to refuse those tasks because it already has a backlog of frontiers to explore. This mechanism could still be made more elaborate in the future, considering domain-specific attributes of tasks (e.g. considering the distance between different exploration tasks might lead an agent to reject another such task on that basis rather than the overall quantity).

If a task cannot be carried out by the current agent (due to workload or unmet minimum requirements), the task will be sent to the team coordinator for assignment.

3.5.3 Assigning Tasks

In my work, I assume the assignment of tasks is the responsibility of the agent filling the team coordinator role. Figure 3.11 shows the handling of a task which will be sent to the team coordinator for assignment. The agent attempts to send the task to the team coordinator (Section 3.5.4). If the team coordinator accepts the task, it will first attempt to assign the task using the role-based task assignment process (Section 3.5.3.1). If no suitable agent is found using this process, an attempt is made using the exhaustive assignment process. It is also possible for the team coordinator to refuse the task, in which case the agent will attempt to assign the task on its own, providing it has the necessary capabilities to perform task assignment.

To keep the resources required to track tasks out for assignment, and to prevent overwhelming the wireless communication, an agent assigning tasks will only attempt to assign a fixed number of each task type at any one time. In my implementation, I limit the task assignment process to 5 instances of each task type.

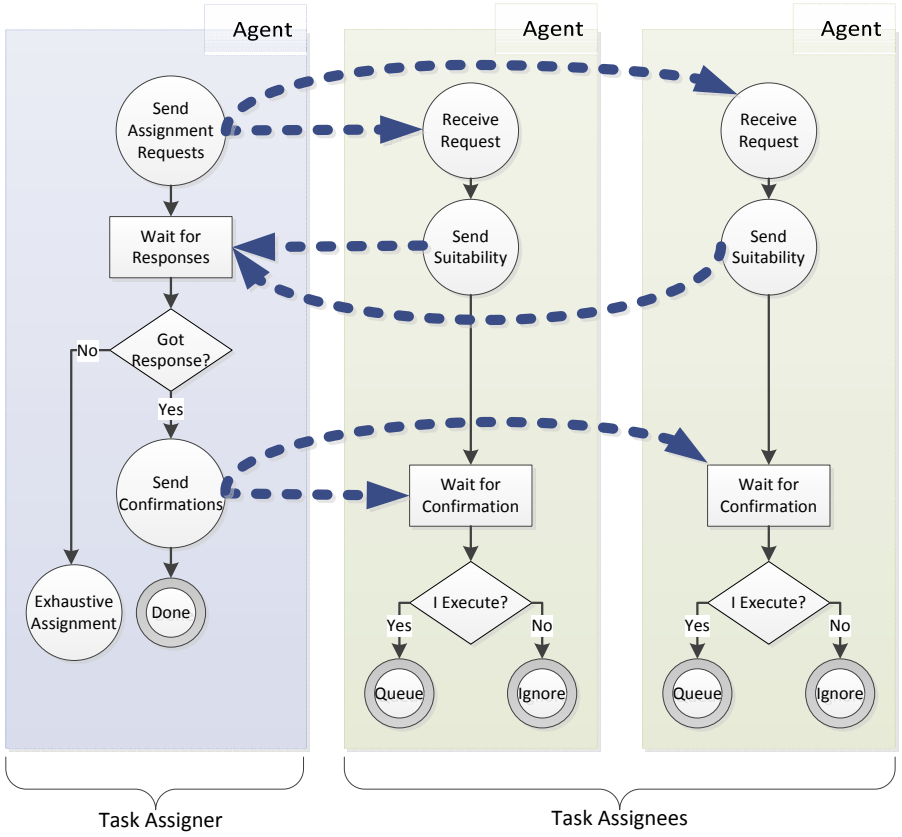


Figure 3.12: Overview of an agent performing task assignment. role-based and exhaustive task assignment use the same general approach.

3.5.3.1 Role-based Task Assignment

An agent with the capability of performing task assignment will first attempt to assign the task using *role-based task assignment*. *Role-based assignment* of tasks takes advantage of the fact that the definition of a role is in terms of the tasks an agent in that role is normally expected to carry out. This eliminates the need to match task attributes exhaustively against the attributes of each team member, speeding the task assignment process. Role-based assignment is also advantageous because the agent performing task assignment can send messages addressed to specific recipients rather

than broadcasting a request and hoping someone responds. Role-base task assignment occurs in three phases: *sending task assignment requests*, *waiting for responses*, and *sending confirmations*.

As illustrated in Figure 3.12, during the first phase the task assigner sends task completion requests to all agents it is aware of on the current team who are in a role expected to be able to complete the task. The assigner iterates through the agents $\{K_1, K_2, \cdots, K_n\}$ known to be on its team, offering the task to all potential assignees simultaneously. If the role agent K_i is known to occupy is expected to be able to complete the task to be assigned, the assigner sends a task assignment request to agent K_i. The potential assignee agent K_i processes the incoming task assignment request and responds based on its current workload and capabilities.

The potential assignee first determines its suitability to carry out the task by evaluating the task minimum requirements and suitability expression of the task against its own attributes (Sections 3.4.2.1 and 3.4.2.2). If the agent satisfies the minimum requirements for the task, it checks its task list to see how many other tasks of the same type it has queued for completion. If the number of tasks already queued is within a defined limit (in my work, five), the agent sends a response back to the task assigner indicating its acceptance of the task, as well as its calculated suitability to carry out the task. If the minimum requirements of the task are not met or the task would exceed the robot's workload for that type, the agent sends a task rejection response indicating it cannot carry out the task. Tasks an assignee accepts are added to the assignee's task list and marked to indicate that the task is accepted from another agent pending the receipt of a confirmation message. The assignee will

not carry out the task until it receives confirmation. The counting of similar task types to balance workload is a primitive estimator; in any real-world domain there are any number of other factors that could influence the distribution of workload. For example, the estimated time to complete each task could be incorporated into the workload determination to ensure an agent's pending tasks will keep it busy for some determined period of time.

During the second phase of role-based task assignment, the assigner waits for responses to the task assignment requests for a period of time (in my implementation, two seconds). Incoming assignment responses are logged along with each potential assignee's self-reported suitability to carry out the task. My framework also supports a mission-specific cost when determining to whom a task should be assigned. In my example implementation, all tasks occur at a physical location so the cost is the distance from the task location to the location of the assignee agent. The mission-specific cost factor is used to break ties between equally suited agents when determining which should carry out the task.

In the third phase of role-based task assignment, the assigner agent reviews the responses to determine the best agent (if any) to assign the task to. An acceptance confirmation message is sent to the potential assignee whom responded with the highest suitability. Where multiple agents have the same suitability, the agent with the lowest cost (i.e. closest distance in my example implementation) to carry out the task is chosen. The assignee whom receives the acceptance confirmation marks the task as ready for completion and sends an acknowledgment message to the task assigner indicating it received the confirmation. Where the task assigner does not

receive an acceptance confirmation message, it assumes the task assignment failed and proceeds to the exhaustive task assignment process (Section 3.5.3.2).

It is, however, possible the acknowledgment was sent by the assignee but did not arrive due to communication failure. In such a scenario, there is a potential for the same task to be assigned to multiple agents. In the worst case there will be some duplication of effort due to multiple agents executing the task. In the case of an exploration task, for example, the path both agents travel to arrive at the exploration location will differ, resulting in an opportunity for both agents to discover new work along the way. In other domains, duplication of effort may be undesirable. An implementation using my framework targeting such a domain would need to explicitly take measures to minimize the negative impact caused by duplicate work. This could involve checking whether work has already been completed prior to starting (i.e. an agent assigned a task to cut a lawn in an area could check the height of the grass to determine if it has already been cut).

At the same time the acceptance confirmation is sent to the agent the assigner has chosen to complete the task, task cancellation messages are sent to all other task assignees to indicate the task was assigned to someone else. Upon receiving these messages, the assignees remove the task from their task lists and do not consider it for completion. A timeout ensures that an assignee will not wait for an acceptance or cancellation message for too long. If an assignee has not received either message before the timeout, the task is removed from its task list and no longer considered for completion by that assignee.

Where the task assigner fails to receive an acceptance confirmation message indi-

cating that an assignee has accepted the task, it will attempt to use exhaustive task assignment (Section 3.5.3.2) to assign the task. Similarly, if the assigner received only rejection responses from all assignees, it will also use exhaustive task assignment to attempt to assign the task.

3.5.3.2 Exhaustive Task Assignment

If role-based task assignment fails to result in the assignment of a task, the agent falls back to using exhaustive assignment. Exhaustive task assignment is not optimal as it places extra load on the team to process the assignment requests and track responses. It also necessitates a greater reliance on communication between agents.

In exhaustive task assignment, role information cannot be relied upon to provide guidance in determining the agents to assign a task to. Exhaustive task assignment uses the same three phases as role-based assignment. The distinction is that in exhaustive task assignment, a task assignment request is broadcast to any agent in range that can hear it – no specific addressing occurs. Agents that receive the task assignment request and are on the assigner's team send back responses as in role-based assignment.

If after role-based and exhaustive assignment a task is still unassigned, it is re-queued on the assigner's task list so that an attempt can be made to assign it later.

3.5.4 Sending Tasks to the Team Coordinator

Agents rely on the team coordinator to perform task assignment. The team coordinator should have the most complete knowledge of the team complement and

be the best equipped to perform the task assignment. To send a task to the team coordinator, an agent identifies which agent it knows is the team coordinator, sends a message with the task to be assigned, and awaits a response. Where sending a task to the team coordinator fails, the agent will attempt to assign the task by itself if possible.

In order to determine to whom the task should be sent, the agent uses its current knowledge of its teammates to identify the agent it thinks is filling the team coordinator role. It sends a message to that agent with the task requiring assignment. If the recipient is the team coordinator, and it has the capability of assigning tasks, it sends a response indicating that it received the task successfully and will assign the task on behalf of the sender. Having successfully sent the task to its team coordinator, the agent removes the task from its task list. It is also possible that the agent identified as the team coordinator is not the team coordinator, such as in the case when another agent has assumed that role but communication issues prevented the agent from learning about the role change. In such a scenario, the agent will periodically attempt to re-send the task until its knowledge of its teammates correctly identifies the team coordinator.

It is also possible the agent filling the role of team coordinator lacks the computational and memory capabilities required to perform the task assignment operation described in Section 3.5.3. This might happen if the team coordinator agent fails and a poorly suited agent, incapable of assigning tasks, assumed the team coordinator role. In such a scenario, the members of the team would be responsible for retaining tasks they identity until the team complement includes an agent capable of filling the

team coordinator role with the necessary capabilities to perform the task assignment operation.

To help cope with unreliable equipment and communication, an agent sending a request to the team coordinator to assign a task monitors the request for a defined period (two seconds in my implementation). If the sender does not receive a response in a timely manner, it attempts to assign the task by itself if possible, and failing that re-queues the task in its own task list later processing. Failures can occur if the sender has wandered out of range of the team coordinator, or if the team coordinator has failed, for example.

3.6 Role and Team Determination

In Section 3.2.1, the concept of an agent determining its role and team membership was introduced as a means of responding to the failure or loss of agents from the team. To ensure a team is resilient to situations such as these, each agent periodically determines the best role for it to fill at any one time. In this way, the team adjusts the roles of its members in a distributed fashion using knowledge of the current team structure shared among all teammates. At the same time a role determination occurs, each agent also evaluates its current team membership to recognize situations where it should form a new team of its own.

3.6.1 Role Determination

Each agent periodically (every 30 seconds in my implementation) evaluates its current role within the context of all agents that it currently knows about. This

Algorithm 2 Determining a new role for an agent.

Require: $\{K_1, K_2, \cdots, K_n\}$, the known agents on my team.

for all $R_i \in \{R_1, R_2, \cdots, R_n\}$, the roles for the current mission. **do**

$S \leftarrow$ suitability of agent to fill role R_i

$W \leftarrow$ role importance weighting

Track role with highest $(S + W)$

end for

Switch to role with highest $(S + W)$

enables it to detect situations where the current team complement has strayed away from the desired team composition. An example is where the team coordinator becomes damaged and another agent will take over that role. It also allows the agent to adjust its role when another agent switches into the role it currently fills. The agent may have been filling a role it is not ideally suited for and another agent has switched into that role. In such a scenario the first agent may be able to switch to a role to which it is better suited.

As illustrated in Algorithm 2, role determination consists of the agent iterating through all roles for the current mission and determining the best role for it to occupy at the current time, given the other agents it knows about. For a role R_i, the agent first determines its own suitability S to occupy the role (see Section 3.4.3.1 for information on calculating an agent's suitability to fill a role). A role importance weighting W is then calculated based on the number of agents currently occupying this role compared against the definition of a desired team and the importance of the role. The agent tracks the role with the highest $S + W$ value and switches to that role if it differs

Algorithm 3 Determining role importance weighting for an agent to fill a role.

Require: R role to determine important weighting

Require: K known agents on agent's team

Require: S suitability of agent to fill role R

Find N = number of other known agents K in this role on my team

D_{min} = minimum number of agents filling this role on a desired team

D_{max} = maximum number of agents filling this role on a desired team

I_r = the importance weighting of the role R on a desired team

if $N < D_{min}$ **then**

$W \leftarrow \frac{D_{min}-N}{D_{min}} I_r$

else if $N \geq D_{min} \wedge N < D_{max}$ **then**

$W \leftarrow \frac{D_{max}-N}{2D_{max}} I_r$

else if $N \geq D_{max}$ **then**

if another agent $K_i \in K$ fills role R and its suitability is less than S **then**

$W \leftarrow I$

end if

end if

return W the role importance weighting

from its current one.

Algorithm 3 shows how an agent calculates the role importance weighting W for a role R. The purpose of the role importance weighting is to encourage an agent to fill a role to which it is less suited when the number of agents filling that role has fallen below what is specified in the definition of a desired team.

If N, the number of agents occupying the role R_i, is less than D_{min}, the desired minimum number of agents occupying role R_i, the role is assigned an importance weight $W = \frac{N_{min}-N}{N_{min}} I_r$, where I_r is the importance of the role from the definition of a desired team. The result is a decreasing importance weighting for the role R_i as the number of agents in the role approaches the minimum desired N_{min}.

If N, the number of agents occupying the role R_i, is between D_{min}, the desired minimum number of agents occupying role R_i, and D_{max}, the desired maximum number of agents occupying role R_i, the role is assigned a lesser importance weighting to reflect the fact that its minimum has already been met. In this case, $W = \frac{D_{max}-N}{2D_{max}} I_r$.

Finally, if N, the number of agents occupying the role R_i, is greater than or equal to D_{max}, the role is currently filled according to the definition of a desired team. In such a circumstance, the agent attempts to determine if it is better suited to be occupying the role R_i than any of the other agents on its team already occupying that role. The agent does this by looping through the known agents $\{K_1, K_2, \cdots, K_n\}$ on the current team and finding the agent occupying the role R_i with the lowest suitability to occupy that role. If all agents are better suited to occupy the role, the role is not considered as a candidate for the agent to switch to and $W = 0$. If a less suited agent occupying the role is found, the importance weight of the role is assigned $W = I$.

If at the end of processing a better suited role was found, the agent switches to that role and announces the role change to its teammates. Any agents occupying the role which receive the role switch announcement will initiate a role check as well. In this way, if an agent is better suited to fill a role than a team member it knows of

in that role, that team member will re-evaluate its role given that the better suited agent now occupies that role.

Performing a role check depends on an agent's knowledge of its current team composition. Since agents rely on communication to discover the composition of their team, unreliable communication results in inconsistencies between the agents' view of the current team structure compared to the actual team structure. As a result, an agent performing a role check may be making its decision based on inaccurate information. An agent's view of its current team structure could lack information about new team members who have joined the team, or it could include information about team members who have left the team or failed.

Where an agent does not have knowledge of a member of its team the agent might, for example, switch to a role it considers unfilled despite the existence of a better suited team member whom has already filled the role. In such a scenario, the agent's inconsistent view of the current team structure results in it adopting a different role than it would if its knowledge of the current team structure included the agent it was not aware of. As communication improves, however, the agent would learn of the better suited agent and adjust its role accordingly.

Similarly, an agent's knowledge of its current team composition could include a teammate who has failed or left the team. In scenarios such as these the agent may continue to fill its current role despite there being an under-filled role. However where an agent has not heard from a teammate in some time it removes the teammate from its knowledge of its current team composition. The agent will consider the missing teammate during its next role check. Similarly, other members of the team will

recognize the departure of the teammate and attempt to compensate during the next role check.

The definition of a desired team specifies a range of agents filling each role, which will account for some suboptimal role switches. In the worst case there is a potential for the roles agents fill on a team to deviate from the definition of a desired team. This is to be expected, however, as the definition of a desired team is intended to act primarily as a guideline to define a team structure considered ideal for operation of the team. Deviations from the definition of an ideal team can result in duplication of effort, or a delay in executing some types of tasks. However, it is desirable for a team to continue operation in a degraded state rather than ceasing to function altogether.

3.6.2 Team Determination

Prior to the agent evaluating its role within its current team, it also evaluates its current team membership. If the agent has not heard from any members of its current team in some time, it will form a new team with itself as the only team member (for information on how agents forget team members in my example implementation, see Section 4.7.2 in Chapter 4). By the time an agent decides to form a new team of its own, it will have already in effect left its team. Its previous teammates may have already forgotten about it and adjusted the team structure to compensate. Further, had the agent been able to find its way back to its team, it would have as a part of completing its assigned tasks.

An agent forms a new team when it becomes isolated, to ensure that a team consists primarily of agents in close physical proximity. If agents maintained their

previous team identity, a situation could arise where geographically separated groups of agents consider themselves members of the same team. The result would be a fragmentation of the team where each fragment consists of very different team members, each with its own different goals. The team fragments will behave as if they were actually different teams, and it makes sense to identify them as such. By changing team identity when an agent becomes separated from its team, any new team forming as a result will be identified as a different team, preventing the issue of a fragmented team identity. If an agent becomes separated from its team and were to encounter its previous team again, it would re-join that team if space was still available and its skill set was still required.

3.7 Team Merging and Redistribution

This section describes the process and algorithm used to perform the team merge and redistribution process which occurs when agents encounter one another. The section begins with a discussion of the interactions between the encountering agents and the communication required between them. A discussion of the impact of communication failures and inconsistencies in an agent's knowledge of its team structure follows in Section 3.7.1.

As introduced in Section 3.2.2, when agents encounter one another while performing operations in their environment, an opportunity arises for the agents' teams to exchange mission progress and information about one another's teams. The encountering agents act as representatives for their respective teams and are responsible for performing the merge and redistribution operations.

Algorithm 4 Merging and redistributing teams occurs by clearing both known teams and iteratively re-adding members.

Require: $M = \{M_1, M_2, \cdots M_n\}$, the agents on merging agent's team.

Require: $O = \{O_1, O_2, \cdots O_n\}$, the agents on other agent's team.

$K = M \cup O$, known agents on both teams.

$M = O = \emptyset$; clear both known teams.

while agents exist in K that are not assigned to a team **do**

$K_i \leftarrow$ agent with highest suitability to fill a role on team M or team O.

Add agent K_i to best suited team, M or O.

end while

Agents can only be involved in a single encounter at a time. An agent observing multiple other agents will attempt to initiate an encounter with the first observed agent. Similarly, an agent receiving multiple requests to initiate an encounter will participate in the first received encounter for which it received a request. Attempting to perform multiple encounters at the same time would necessitate a means of synchronizing the merge and redistribution operations. This would unnecessarily complicate the team merge and redistribution process, and increase the chance of communication failures impacting the process as a whole.

Recall from Section 3.2.2, the representative agents first exchange information about one another's teams. Since the team merge and redistribution operations are negotiated by the representative agents, and these operations are similar in nature to the activities expected of an agent filling the team coordinator role, they use their suitability to fill that role as a means of determining which agent should perform the

actual merge and redistribution calculations.

Each representative agent determines its own suitability to fill the team coordinator role given it knows its attributes $A_{self} = \{A_1, A_2, \cdots, A_n\}$. The agent uses Algorithm 1 (see page 77, Section 1) to calculate the numerical suitability, S_{self}. The agent also calculates the suitability of the other representative agent, S_{other} to fill the team coordinator role. Both representative agents calculate these values using the attributes of the other agent found in the previously exchanged team information.

A team merge and redistribution suitability threshold determines the minimum suitability required to execute the team merge and redistribution algorithm. The threshold recognizes the fact there is a bare minimum set of capabilities required for an agent to be able to complete the team merge and redistribution calculations. Where the agent's own suitability S_{self} is less than the threshold, it will rely on the other agent to complete the team merge and redistribution calculations. Where neither agent's suitability is greater than or equal to the threshold, the encounter is terminated without making any changes to either team.

The representative agents both determine if $S_{self} > S_{other}$. If the agent's own suitability is greater than the other agent's, it will consider itself responsible for completing the team merge and redistribution calculations. Where both agents determine they are equally suited, ties are broken by choosing the agent with the higher identification number (recall from Section 3.4.1, agents have a unique identification number).

Algorithm 4 describes the process used to perform the team merge and redistribution operation. Of the two encountering agents, the one that is best suited to

fill the role of team coordinator is responsible for executing Algorithm 4. Each of the encountering agents calculates its suitability to fill the team coordinator role and exchanges it at the same time it that it sends the other agent information about its own team (for more information on determining an agent's suitability to fill a role, see Section 3.4.3.1). For the purposes of Algorithm 4, $M = \{M_1, M_2, \cdots M_n\}$ is the set of agents known to be on the team same team as the agent performing the merge, and $O = \{O_1, O_2, \cdots O_n\}$ is the set of agents known to be on the other agent's team.

The goal of the team merge and redistribution algorithm is to re-shape the two teams or combine them into a single team, as determined by the definition of a desired team. The algorithm begins by combining M and O into K, and clearing M and O; the agents from both teams are pooled and then iteratively placed back on to a team in the role that suits them best.

While there are still agents in K left to process, the algorithm iteratively finds the agent with the highest suitability to fill a role on either team M or team O, and adds that agent to the best suited team. Algorithm 5 describes the manner in which the highest suitability agent is found. For each agent $K_i \in K$, the algorithm determines the suitability of the agent to fill each of the roles for the current mission on both team M and team O. The suitability calculation used is the same one an agent uses to determine its own suitability to fill a role on its current team (see Algorithm 2 in Section 3.6.1). This ensures that where appropriate a less suited agent can fill a role if required. Algorithm 5 tracks the agent K_i that is best suited to fill a role on either M or O. Since the algorithm always finds the best suited agent, sub-optimal role assignments are avoided unless necessary.

Algorithm 5 Finding the agent to add to the merged teams.

Require: $K = \{K_1, K_2, \cdots, K_n\}$, known agents on both teams.

Require: $M \leftarrow$ the agents on the merger's team.

Require: $O \leftarrow$ the agents on the other agent's team.

$B \leftarrow$ null, the best suited agent to join one of the teams.

$B_s \leftarrow$ null, suitability of best suited agent.

for all $K_i \in \{K_1, K_2, \cdots, K_n\}$ **do**

for all $T \in \{M, O\}$ **do**

for all $R_i \in \{R_1, R_2, \cdots, R_n\}$, the roles for the current mission. **do**

$S \leftarrow$ weighted suitability of K_i to fill role R_i on team T

Track agent K_i with best suitability S.

end for

end for

end for

return K_i, the agent best suited to join one of the two teams.

The process continues until all agents have been added to teams. Where an agent is equally suited to a role on team M or team O, the algorithm will keep the agent on its previous team to prevent agents unnecessarily changing teams every time a merge and redistribution occurs. This is important as my framework is designed to operate in environments where communication between agents is unreliable. Minimizing team changes ensures that agents do not waste time attempting to communicate a team switch that has no positive impact on either team.

Following the merge and redistribution operation, the agent executing the merge

and redistribution algorithm sends the role changes and team membership changes applicable to the other team to the other representative agent. The two representative agents negotiating the team merge then send instructions to their teammates indicating the required role changes and team membership changes. Upon receipt of the role and team information, the teammates switch their current role and team as specified. The required role changes and team membership resulting from the team merge and redistribution algorithm are not used by either of the representative agents to update their knowledge of the current team structure – relying on this information completely would mean assuming the changes specified were successfully carried out, which ultimately may or may not be the case. Instead, agents implement role change and team membership change instructions addressed specifically to them, and ignore any other such messages. As agents implement the role change and team membership change instructions, they immediately broadcast a message indicating their new role and team membership information. Agents receiving these broadcasts update their knowledge of the current team structure accordingly. Subsequent communications sent by the agents also include the new role and team membership information, ensuring the new information has a higher chance of being overheard by all team members despite unreliable communication.

3.7.1 Coping with Failures and Inconsistent Knowledge

The description of the team merge and redistribution process and algorithms in the previous section describe the expected process in ideal conditions where knowledge of the team structure is consistent between the agents on a team and communication

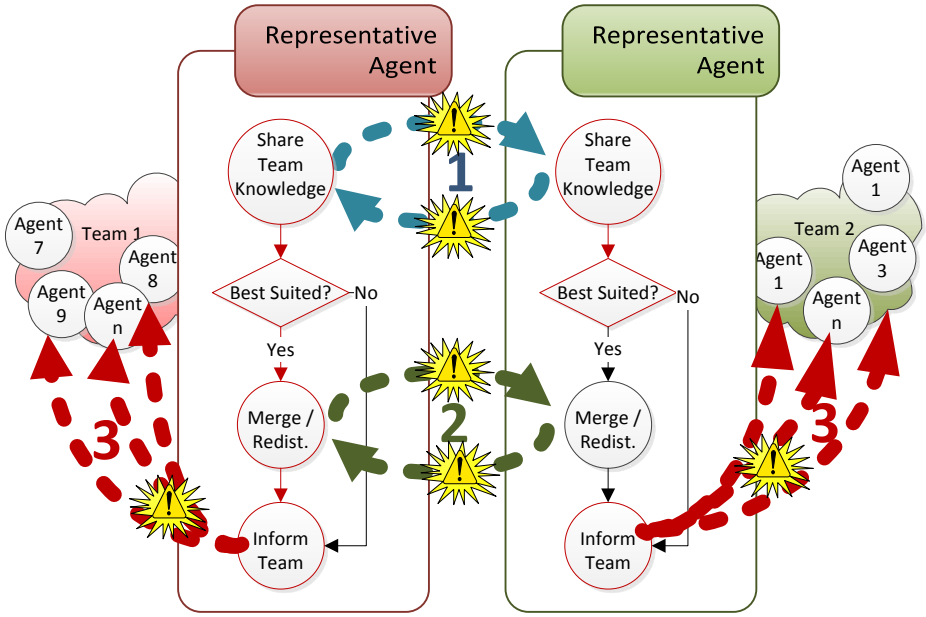


Figure 3.13: Communication failures can impact the merge and redistribution at three points during the process.

failures do not impact the process. This section describes the impact of inconsistencies in an agent's knowledge of its team structure, and the impact of communication failures at various points in the process.

Communication failures have the potential to impact the team merge and redistribution process either by directly interrupting the process itself, or indirectly by causing inconsistencies in the knowledge each representative agent has of its team composition. Inconsistencies in a representative agent's knowledge of its team composition can cause it to make decisions it would not otherwise make if it had complete knowledge of its team, consistent with that of all teammates.

As shown in Figure 3.13, communication failures can directly impact the merge and redistribution operation by interrupting the exchange of messages between the

representative agents and their team at three critical points. Failures can occur while exchanging information about each other's teams (1), communicating the required role and team changes (2), and when each representative informs its respective teammates of required team and role changes (3). Timeouts are used in the major steps of the process to ensure an agent will not wait indefinitely for messages it is expecting from the other agent.

When the representative agents first exchange information about one another's teams (Figure 3.13 (1)), a failure will cause the merge and redistribution process to terminate. The agents have the opportunity to try again during their next encounter. No role changes or team membership changes will occur in this scenario.

A communication failure occurring when a representative informs a member of its team of a role change or team membership change (Figure 3.13 (3)) will prevent that agent from implementing a role change or team membership change. In such a scenario, the agent will learn of role changes or team membership changes implemented by other agents on both teams and update its knowledge of the current team complement accordingly. The agent will adjust the role it fills on its team based on this knowledge during its next role check. Depending on how poor communication conditions are, there is a potential for multiple role changes or team membership changes to not be implemented as required. The role check operations performed by the agents ensure the team attempts to adjust its structure given the role changes and team membership changes which were successfully implemented.

As indicated in Figure 3.13 (2), a communication failure can occur when the representative agent completing the merge and redistribution calculation informs the

other representative of the role changes and team membership changes required. In such a scenario the representative agent sending the merge instructions would begin informing its own teammates of the required changes, while the other representative agent would never receive the merge instructions required of its team. The result is a scenario where members of one team implement role changes and team membership changes which are inappropriate given the other team is not implementing any changes.

Since the role change and team membership change instructions are only used by the agents implementing the instructions, neither team will update the knowledge of its team complement with the unimplemented changes. The impact of not implementing changes will vary, depending on the exact nature of the changes. For example, an agent might be instructed to switch roles on its team in order to accommodate an agent from the other team. If the agent from the other team never receives the instruction to switch teams, the role vacated to accommodate the agent switching teams will remain unfilled. In such a scenario, future role check operations would lead to the recognition of the vacant role, and the agent that switched out of it (or some other agent) would be instructed to fill that role if it resulted in a better team structure. In the meantime, the team may have been forced into a sub-optimal configuration. Further, the team is unable to gain the benefit of the new agent which was to join its team.

Although in the previous example the team is able to easily repair itself through a role check, it is possible to imagine scenarios where this is not possible. The team merge and redistribution operation could, for example, determine an agent on team A

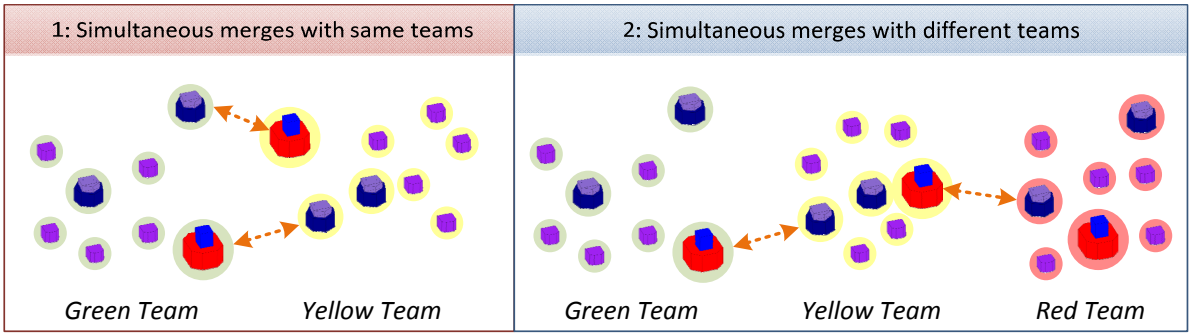


Figure 3.14: Simultaneous merge scenarios can occur due to multiple encounters between the same teams, or between different teams.

is to swap roles with an agent on team B. If the agent on team A fails to receive the required role change and team membership change instructions, the agent on team B would change teams and assume the role the agent from team A was supposed to vacate. This could leave team A with an overfilled role, and team B with an underfilled role. This would result in both teams operating in a degraded state. Team A might have room for the extra agent in another role on its team, and would implement such a change during the next role check. Team B, however, would need to encounter another team (or encounter team A again, at a later time) or lone agent in order to replace the missing agent.

Aside from communication failures during the team merge and redistribution, it is also possible for multiple encounters between agents to occur at the same time. Figure 3.14 shows two ways in which simultaneous encounters can occur. These scenarios are illustrations of the types of simultaneous encounters that might occur.

In the first scenario (Figure 3.14(1)), multiple agents on each team encounter multiple agents on the other team at the same time. Given the knowledge each of

the representative agents possesses of their own team composition will likely vary, the multiple pairs of representatives might potentially arrive at different required role changes and team membership changes. Further, the agents implement the role changes and team membership changes in the order they are received, resulting in the potential for parts of either of the role changes or team membership changes from the simultaneous merges being implemented. As with the communication failure scenarios explained above, the agents on both teams will adjust their own roles given the role changes and team changes that actually do get implemented, ensuring both teams adapt as required. The representatives, however, will likely have similar knowledge of their team structure and will thus attempt to implement similar role changes and team membership changes. In some scenarios, the simultaneous encounters could actually compensate for communication failures during the merge and redistribution operations, acting to cross-check each other.

In the second scenario (Figure 3.14(2)), a team simultaneously encounters agents on different teams. The result is a scenario where a single team attempts to perform the merge and redistribution operation with two other teams at the same time. In scenarios such as these, each representative would only have knowledge of their team as it was at the start of the encounter. Similar to the first scenario, role changes and team membership changes will be implemented by the representative agents, potentially causing conflicting changes within the teams. As with the first scenario, the teams will all adapt as agents learn of the changes and complete role checks.

In reality, it is not likely for three large teams to be involved in a simultaneous merge scenario such as this. Agents will tend to be geographically spread out as they

complete tasks, and it is expected teams would tend be distributed geographically somewhat as well. This makes sense as, for example, rescuers would not insert all rescue robots in the exact same location. The teams would be introduced in multiple locations, and exploration should ideally direct the teams and robots within them to spread out in order to explore the area faster. It is important to note, however, work in laboratories sometimes assumes robots begin operation in a common location (e.g. [Anderson and Papanikolopoulos, 2007]). This provides robots with a common localization points for robots, but is not practical for actual rescue work.

A more likely case of the second scenario in Figure 3.14 would be where a team encounters two lone agents at the same time. In such a scenario, the team could decide to adopt both agents. If both agents ended up in the same role, there is a possibility that role ends up overfilled. If there was space for one of the agents in another role on the team, the required change would occur when the agents perform a role check. If there is no space for the extra agent in another role, the role would remain overfilled until another team is encountered, or a vacancy is created due to the failure of another team member. A detailed discussion of a scenario such as this is found in Section 3.7.2.5.

In any of the previous scenarios, there is a possibility communication failures could result in the team implementing role changes and team membership changes that unintentionally result in agents filling a role such that the number of agents filling that role exceeds the maximum per the definition of a desired team. During subsequent role check operations, the agents in the overfilled role could switch to another role if space was available in a suitable role.

It is, however, possible there are no appropriate unoccupied roles to which extra agents could switch. A team can respond to this situation in a number of different ways: The team could continue operating with overfilled roles, the extra agents could be forced to leave the team, or the team coordinator could use the team merge and redistribution algorithm (Section 3.7) on the current team complement, splitting the team into two or more new teams which meet the definition of a desired team.

In the first response to overfilled roles, the team continues operating with some roles overfilled. The next time another team is encountered, the agents in excess of the role maximum have the opportunity to switch to another team. Further, during operation it is possible an agent may become separated from the team or suffer a failure, bringing the team closer to the definition of a desired team. Handling overfilled roles in this manner might be reasonable if encounters between teams occur frequently, and agents will frequently suffer failures or become lost. In my example implementation, I chose to use this approach, as the number agents available of each type means it is unlikely for there to be a large surplus of the more capable agent types so as to overfill their best suited roles. I also chose a large range of desired agents to fill the role to which the least capable agents are best suited, so that in most scenarios there would be room for extra agents resulting from inappropriate merge operations.

The second potential response to overfilled roles involves agents recognizing situations where there is no longer space for them on their current team, and leaving the team on their own. The agents could recognize this during the period role check operations. Departing the team would provide the agents with an opportunity to join

another team, build up a new team of their own from other agents encountered, or to join another team with space. Agents departing the team in this manner also act to provide a means of distributing their former team's knowledge to a new encountered team.

The third potential response to overfilled roles involves the agent filling the team coordinator role recognizing its team has overfilled roles, and initiating the team merge and redistribution algorithm, using its current team as input to the algorithm. The algorithm would provide the necessary role changes and team membership changes to correct the overfilled roles. This would result in the team splitting into two smaller teams, with the agents distributed between the teams. This approach raises the possibility of similar issues as when two teams are merged, and the role change and team membership change instructions are not fully implemented. Investigating these latter two responses is identified as future work (Section 6.4).

The next section illustrates some common scenarios and shows how the team merge and redistribution algorithm adjusts the two teams as a result.

3.7.2 Examples of Team Merge and Redistribution

The following examples illustrate some common scenarios where merging and redistributing teams can occur. The examples use the roles and agent models from my implementation (see Sections 4.5.3 and 4.4 for more information, but the degree of detail presented here suffices for the context of examples). There are three roles illustrated; team coordinator, victim verifier, and explorer. The examples visualize the desired team structure as a series of boxes representing the number of each role

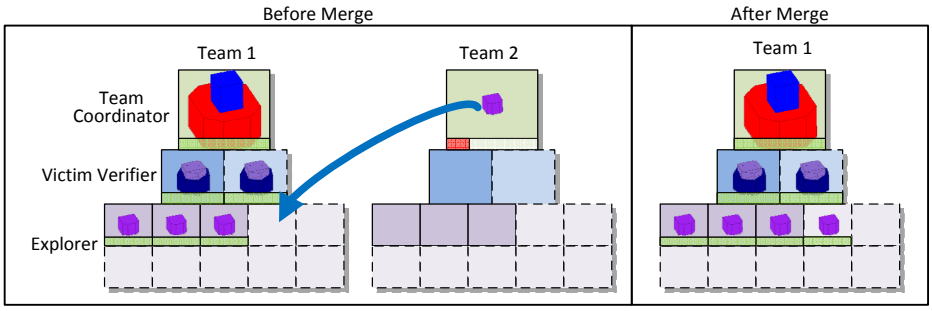


Figure 3.15: An established team encounters a lone agent unsuited to filling the team leader role. The robot joins the established team in its optimal role of explorer.

to fill (e.g. Figure 3.15). The boxes with dashed lines represent role requirements above the minimum desired. A colored bar below each agent illustrates the level of suitability that agent has to fill the role it is in. These examples are not intended to be an exhaustive illustration of all possible team merge operations that can occur, but instead serve to highlight some of the more common merges that would occur in a dynamic and challenging environment.

3.7.2.1 Encountering Supplementary Agents

Figure 3.15 illustrates a situation when an established team (team 1) encounters a lone agent filling the team coordinator role of its own team (team 2). This lone agent could be a replacement released into the environment that has come across an existing team, or an agent that has become separated from its team. The team merge results in the lone agent joining team 1 and assuming its optimal role, explorer, on that team.

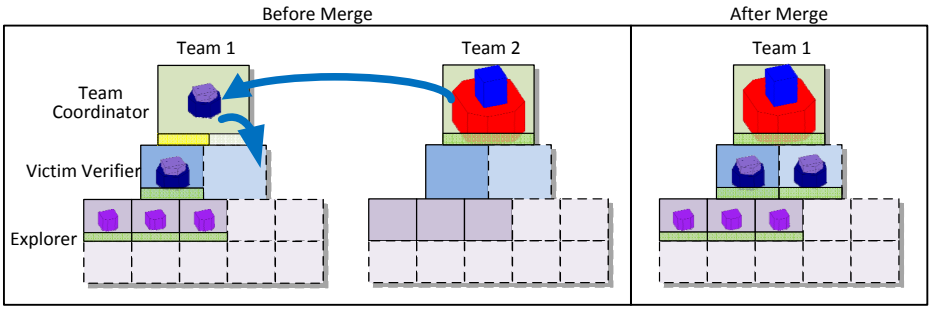


Figure 3.16: A team with a suboptimal team coordinator encounters a replacement agent better suited to that role. The suboptimal team coordinator cedes its role to the replacement and takes on its optimal role of victim verifier.

3.7.2.2 Encountering Replacement Agents

Figure 3.16 illustrates a scenario where team 1, a team with a suboptimal team coordinator, encounters a lone agent that is better suited to that role. Such a scenario could occur if, for example, the leader of team 1 fails or becomes separated from its team; in such a scenario another agent would assume the team coordinator role, albeit at a reduced capacity. Team 2 consists of a single agent well suited to filling the team coordinator role; this could be a replacement agent or an agent that has become

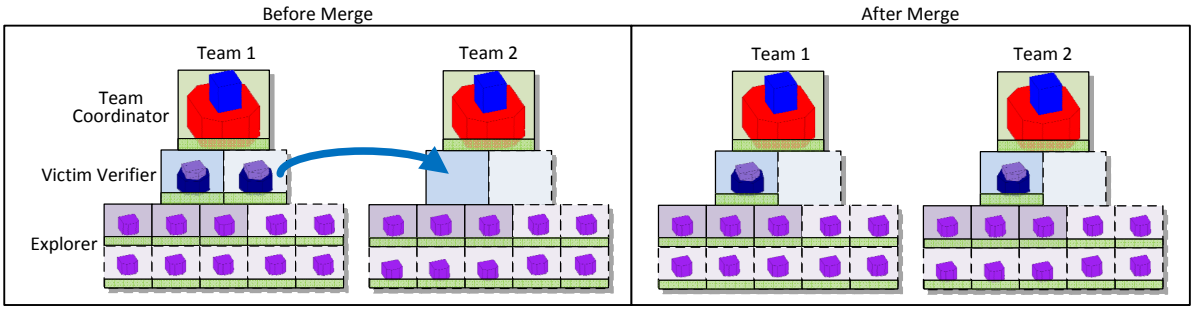


Figure 3.17: A team with the victim verifier role unfilled encounters a team with two and obtains one for itself.

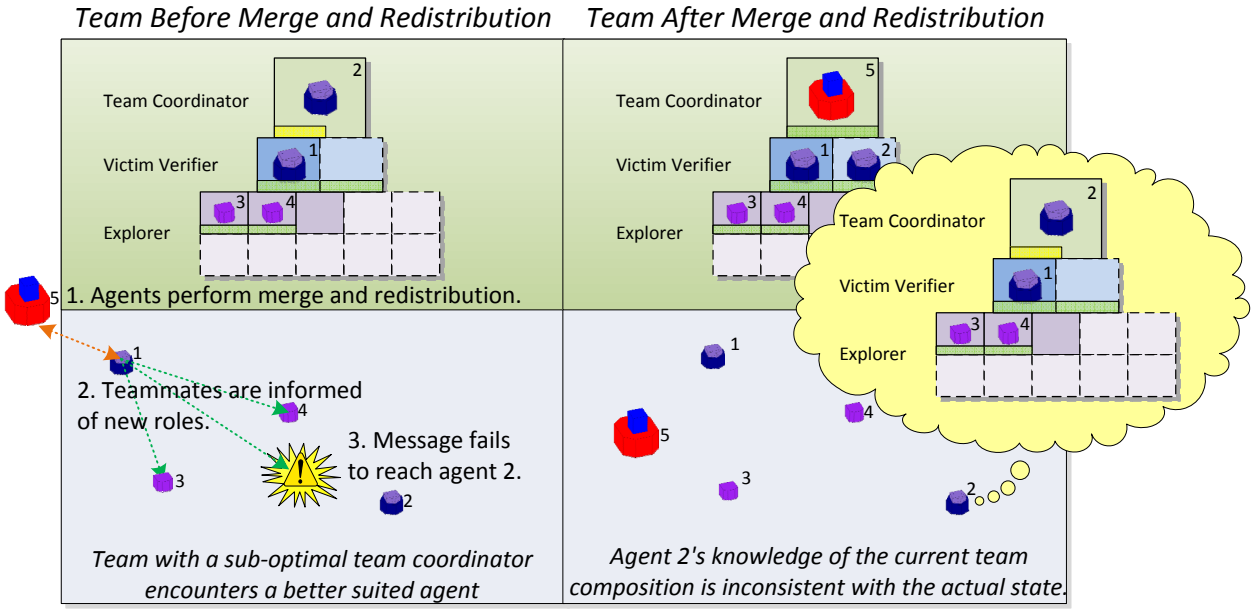


Figure 3.18: A merge and redistribution where an agent does not learn of a required role change due to communication failure.

separated from its team. When teams 1 and 2 merge, the agent from team 2 assumes the team coordinator role on team 1, and the current team coordinator on team 1 assumes its optimal role as a victim verifier.

3.7.2.3 Redistributing Teams

Figure 3.17 illustrates a scenario where team 2 has not met its minimum requirements for the victim verifier role. Upon encountering team 1, one of the agents occupying the victim verifier role on team 1 switches to team 2 so that it can meet the minimum requirements for that role as per the definition of a desired team.

3.7.2.4 Role Check after Team Merge and Redistribution

Figure 3.18 shows a scenario where a team merge and redistribution operation occurs, but communication failure prevents a member of the team from being informed of the new team structure. In Figure 3.18, prior to the merge and redistribution the team is operating in a degraded state. Agent 2 fills the role of team coordinator on the team. Agent 1 encounters a lone agent (agent 5), and acts as a representative to perform the merge and redistribution of the teams. It is determined agent 2 will cede the team coordinator role to agent 5, who will join the team. Agent 2 will switch to the explorer / verifier role. Agent 1 informs the team of the required role changes. Agent 2, however, does not receive the message informing it to switch from the team coordinator role to the explorer / verifier role.

Figure 3.18 shows the team after the merge and redistribution. Agents 1, 3, 4 and 5 are aware of the new team configuration and know agent 5 is the new team coordinator. Since agent 2 failed to receive the message notifying it of the changes, its knowledge of the current team composition is now inconsistent with the actual state. This means both agents 2 and 5 believe themselves to be the team coordinator. This inconsistency will correct itself when agent 2 learns about the existence of agent 5, who is filling the team coordinator role. At that point, agent 2 will perform a role check causing it to switch to the victim verifier role as originally planned.

3.7.2.5 Simultaneous Team Merge and Redistribution

As shown in Figure 3.19, a team with four agents has an opening in the victim verifier role. Agents 1 and 2 both encounter lone agents operating in the environment.

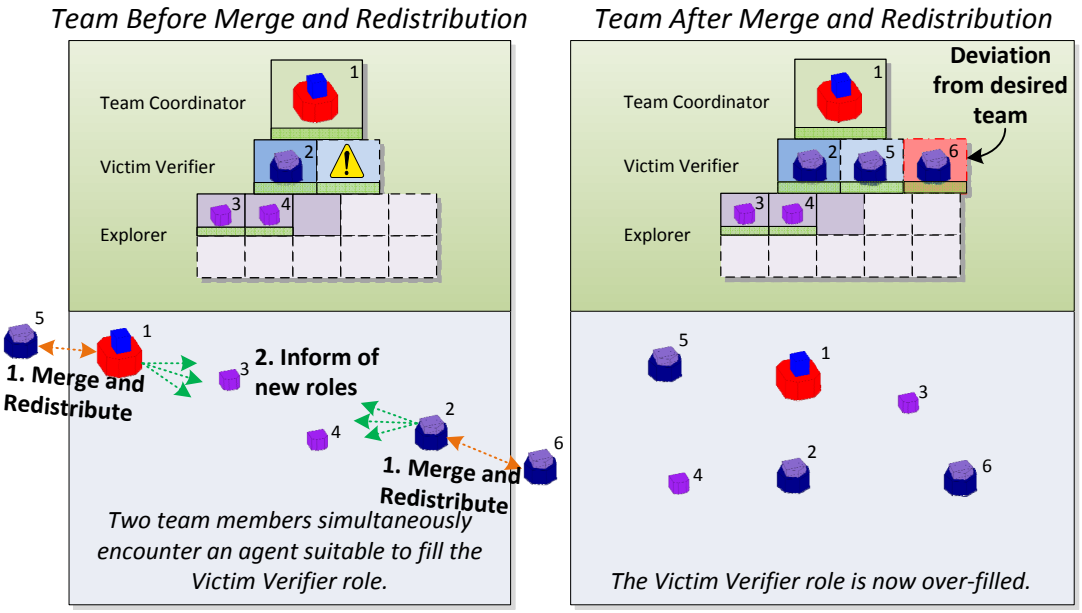


Figure 3.19: Two lone agents simultaneously join the team, resulting in a deviation from the definition of a desired team.

Agent 1 encounters agent 5 and initiates the team merge and redistribution operation. At the same time, agent 2 encounters agent 6 and initiates the team merge and redistribution operation. Both agent 1 and agent 2 recognize the agent they encounter as a suitable agent for the unfilled victim verifier role on the team. As a result agent 5 and 6 become members of the team and fill the victim verifier role. Figure 3.19 shows the result of the team merge and redistribution. Agents 2, 5, and 6 all fill the victim verifier role on the team. This results in a situation where the team complement has deviated from the definition of a desired team. The first agent of 2, 5, and 6 to perform a role check will recognize the role it fills no longer meets the definition of a desired team and will choose to fill the Explorer role, despite the fact it is less suited to that role.

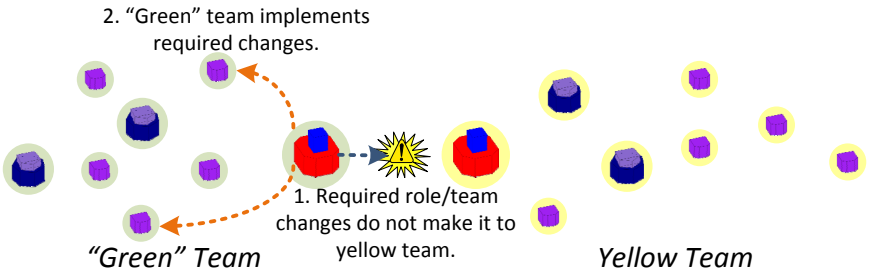


Figure 3.20: Two teams meet and attempt to merge. Communication failure prevents one of the teams from hearing the results of the merge.

It is also possible to imagine a scenario where all roles on the team were filled by agents. In such a scenario, agents 2, 5, and 6 would remain in the victim verifier role and the team would fail to meet the definition of a desired team. The next time another team is encountered, there would be an opportunity for one of the extra agents filling the victim verifier role to switch to the other team.

3.7.2.6 Team Merge and Redistribution Impacted by Communication Failure

Figure 3.20 depicts a scenario where two teams (the *green* team and the *yellow* team) perform a merge and redistribution operation. The representative agent on the green team performs the merge and redistribution calculations and sends the required role and team changes to the representative agent on the yellow team. The message, however, does not make it to the yellow team representative due to a communication failure. As a result, the green team agents implement the required role changes and team membership changes, while the yellow team agents do not due to the failed communication.

In the end only the required changes to the green team are implemented, while

the yellow team implements no changes. As agents on both teams learn of the role changes and team membership changes that succeeded, the teams will adjust the roles of their members as agents perform their periodic role checks.

3.8 Conclusion

This chapter has provided an in-depth description of the major operations that my framework performs in order to form and maintain teams of agents operating in a dynamic and changing environment. The concept of attributes, and how they are used to define the requirements for tasks was described, as well as how roles are defined in terms of the tasks normally expected of them. Finally, the detailed algorithms that operate on the attributes, tasks, roles and teams were presented to describe how the framework accomplishes its team maintenance and task management objectives.

The next chapter describes my example implementation of my methodology. The agents, attributes, tasks, and roles that I use in my implementation are described, along with the mission-specific task algorithms and robot control software, and the simulated environment in which my agents perform. This simulated environment was used to run a series of experiments evaluating this framework in a complex domain, and the presentation of these results follows in Chapter 5.

Chapter 4

Implementation

This chapter describes the Urban Search and Rescue implementation I developed in order to support a controlled evaluation of my framework for adaptive and flexible teamwork in complex environments. Section 4.1 provides an overview of the robots I use in my implementation, and the mapping between robots and roles. Section 4.2 describes the operational concept of my example implementation, and explains how the robots cooperate as a team in order to accomplish the USAR mission. Section 4.3 examines the simulated environment I use in my implementation. The simulated victims and the manner in which robots detect these victims are described, as well as the approach I used for robots to detect the presence of one another in the environment. Section 4.4 describes the different models of simulated robots and the sensory and computational resources available to them. In Section 4.5, I describe the attributes, roles and tasks the robots use accomplish the USAR mission as described in Section 4.2. Finally, Section 4.7 describes framework and mission-specific modules in my implementation.

4.1 Implementation Overview

In order to study the methodology described in Chapter 3, my example implementation is grounded in the domain of Urban Search and Rescue. I assume teams of heterogeneous robotic agents are coordinating to explore damaged structures in order to build a map of the environment and to locate human casualties, and that robots will be lost and new robots will be released into the environment as time goes on.

To keep the scope of my work manageable, my implementation operates in a simulated disaster environment. Using simulation to study my methodology is appropriate as the primary focus of my framework is to support teamwork and coordination between agents, rather than complete accuracy in USAR in particular. Setting up a genuine disaster environment, having enough heterogeneous robots, and controlling trials in such an environment would also be impractically difficult. The approach of using a simulated USAR environment for multi-agent research is well established; Wegner and Anderson [2004] studied how to blend human instructions with those of an autonomous control module, Gauthier and Anderson [2005] studied agents providing assistance to one another, and Eghbali and Sharbafi [2010] studied path planning algorithms inspired by ant colony behaviours in simulated USAR environments, for example. A simulated environment provides the ability to run large numbers of simulations faster than real-time and ensures simulation results are repeatable [Vaughan, 2008]. It is important to ensure that I can run a sufficiently large number of experimental trials to demonstrate my approach. I examine the benefits of a simulated implementation further in Section 4.3.

To perform the USAR mission, I assume that a large number of simple, inex-

pensive, and expendable robots are available. These robots (which I will refer to as *MinBots* for convenience) are small in size, have inexpensive sensors, and have limited computational and memory resources. The MinBots rely on wheels for locomotion. Their small size and wheeled physiology limits their operation to areas relatively clear of debris, and precludes them from navigating over obstacles (though not around them). I assume the MinBot robot type is computationally limited, precluding it from performing operations such a maintaining a map of the team's exploration progress, assigning tasks, and determining new areas to explore. I also assume the availability of a small number of more complex, expensive robots. These physically larger robots (called *MaxBots*) have expensive sensors, and ample computational and memory resources. Similar to the Packbot robot developed by iRobot, MaxBots use a tracked drive system that allows them to navigate over small obstacles and through areas of debris without becoming stuck [Yamauchi, 2004]. Finally, I also assume the availability of robots (called *MidBots*) with capabilities and size that fall between the MinBots and MaxBots. Similar to the MinBots, the MidBots rely on wheels for locomotion and cannot drive over larger obstacles. For more information on these specific robot types, see Section 4.4.

The choice of robot types introduces heterogeneity in terms of the progressively smaller size of the robots used, the computational and sensory capabilities of each, as well as differences in methods of locomotion. Although my framework allows for variations in sensory equipment between individual robots of each general type, my implementation assumes that all robots of a type are equipped the same. MidBots, for example, are equipped with specialized equipment to enable more accurate detec-

tion of victims in a debris field. I have chosen these three to provide the necessary capabilities for operation within USAR domain, and to provide a reasonable challenge in terms of the possibilities for task allocation to the algorithms described in Chapter 3. It is entirely possible there can be further variations in robot types within each of these categories, or additional categories for other domains, and my framework supports such variations.

My implementation assumes robots have a probability of failure, and that a robot will fail completely, rather than individual components failing. An interesting area of future work (Section 6.4), however, would be to implement partial failures, such as those studied by Carlson and Murphy [2003]. The individual components of a robot could have a probability of failing, with robots able to recognize these failures and adjust their advertised capabilities accordingly.

To support search and map creation in a USAR environment I define the following roles: *team coordinator*, *explorer/verifier*, and *explorer*. The *team coordinator* role has the responsibility of directing the overall operation of a team, directing the assignment of tasks to the various team members, and maintaining a map that represents the collective exploration of the team. Robots occupying the *explorer* role are expected to be able to explore an area specified by the team coordinator, and to find potential victims in the environment. Robots occupying the *explorer/verifier* role are expected to be able to confirm the identification of victims in the environment, as well as perform exploration. Further details of these roles from a knowledge/implementation standpoint may be found in Section 4.5.3.

In my implementation, a desired team would have one robot in the *team coordi-*

nator role, one to two robots in the *explorer/verifier* role, and three to ten robots in the *explorer* role. A team structure such as this allows exploration to spread out further from the team coordinator, provides a reasonable level of redundancy between robots, ensures that a reasonable mix of capabilities is present on a team in order to complete its mission, and does not overwhelm the team coordinator with too many robots to coordinate.

Although there is an ideal mapping between the three robot types and the roles in my work (MaxBot → team coordinator, MidBot → explorer/verifier, MinBot → explorer) there will be many times where such a mapping will not be possible, because the appropriate robot is not available (not present or unable to do additional work). Each type could attempt to take on any role, however, to varying degrees of success. If a MinBot is on its own, it must act as its own team coordinator, for example, and would recognize when it encounters any other type of robot to whom it would be best to cede this role. In such a scenario, the limited capabilities of the MinBot would result in it doing little else other than attempting to find a team to join and storing a map of the area explored to report to a new team. Similarly, a MaxBot (operating on its own or as part of a team that already had a functional team coordinator) might perform low-level exploration expected of the *explorer* role, but would switch to the *team coordinator* role when the opportunity arose (e.g. the current coordinator becomes nonfunctional, or a team with a poor coordinator is encountered). The diverse capabilities of the MidBots make them suitable to fill a wide range of roles, albeit in a reduced capacity compared to other more specialized robots. A MidBot could fill the *team coordinator* role in the absence of a MaxBot, for example, with

the expectation the reduced computational and storage capabilities would result in it not being as effective at handling the team. Similarly, when filling the *explorer* role its increased size would prevent it from gaining access to smaller areas that a MinBot would have no problem exploring. Sensor equipment variations between robots will also result in situations where a robot is occupying a general role and may be more useful for specialized tasks (e.g. verifying the presence of a victim).

This section has provided a high level overview of the robots, tasks and roles in my implementation. The following sub-section describes the manner in which robots cooperate in order to create a map of the environment while identifying victims.

4.2 Operational Concept

This section describes the interactions between robots in the context of the two major goals of the mission (Section 2.2): locating and identifying victims, and exploring the environment while building a map, so that victims can ultimately be reached by humans. Finally, the operational knowledge maintained by a team is described, to provide an understanding of how the team protects against loss of data due to failure of the team coordinator.

4.2.1 Locating and Identifying Victims

In my example implementation, positively identifying a victim in the environment is a capability afforded to a limited number of robots on a team. Other robots on the team are able to identify the potential presence of victims in the environment, but must rely on suitably equipped team mates to confirm the presence of the victim.

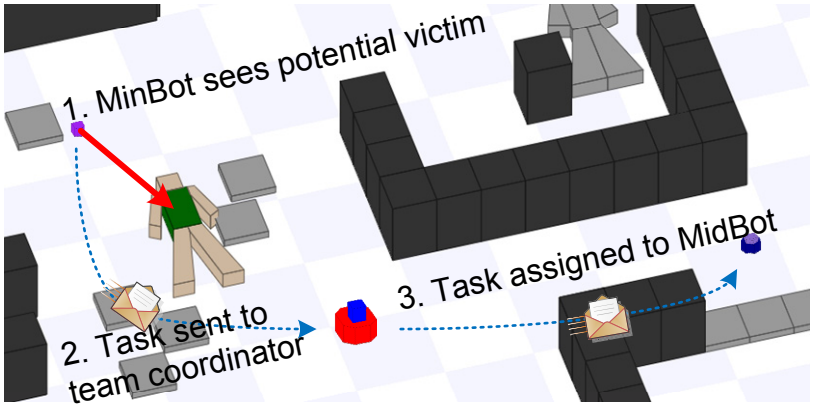


Figure 4.1: A MinBot detects a potential victim.

Of the three types of robots in my implementation, the MinBots and MidBots have the necessary capabilities to detect the potential presence of victims in the environment. I assume the MinBots use a low fidelity victim detector which is only able to detect the *potential* presence of a victim in the environment. In a real-world implementation, this might be accomplished through an infrared sensor to detect heat signatures similar to a human body (in which case similar temperature heat sources could be mistaken for a human victim), or through analysis of images from a camera (in which case it is possible to image debris configurations resembling a human victim). The MidBots are able to positively detect the presence of a victim in the environment (more information on victim detection can be found in Section 4.3.2). In a real-world implementation, the MidBots would use a number of complementary sensor technologies (i.e. heat, visual, sound) to achieve a reasonable level of certainty a human is present. Alternatively, the MidBots could be equipped to rely on a human operator remotely confirming the presence of a potential victim (e.g. similar to [Wegner and Anderson, 2004]). Given there will be a larger number of MinBots

than MidBots, it is expected the MinBots will detect the potential presence of the majority of victims, which the MidBots will subsequently confirm.

Figure 4.1 shows a typical sequence of interactions which occur when a MinBot detects a potential victim. Upon detecting the potential victim, the MinBot generates a task to confirm the presence of the potential victim. The task identifies the location of the potential victim (Figure 4.1(1)). The MinBot does not have the necessary capabilities to carry out the victim confirmation task, so sends it to the robot filling the *team coordinator* role on its team (Figure 4.1(2)); in this case a MaxBot type robot.

The MaxBot type robot performs task allocation for this new task (Section 3.5.3), and assigns it to a nearby MidBot robot with the necessary capabilities to carry out the task (Figure 4.1(3)). The MidBot accepting the task will move to the location specified in the victim confirmation task and use its superior sensory capabilities to confirm the presence of the victim. The results are passed back to the robot filling the *team coordinator* role to update the list of victims it maintains (more information on tracking victims is found in Section 4.8.3).

In a scenario where a MidBot robot initially detected the victim, it would carry out the victim confirmation task itself, moving in closer as necessary to confirm the presence of the victim. The results obtained by completing the task are still passed to the team coordinator to ensure there is a centralized list of the victims found.

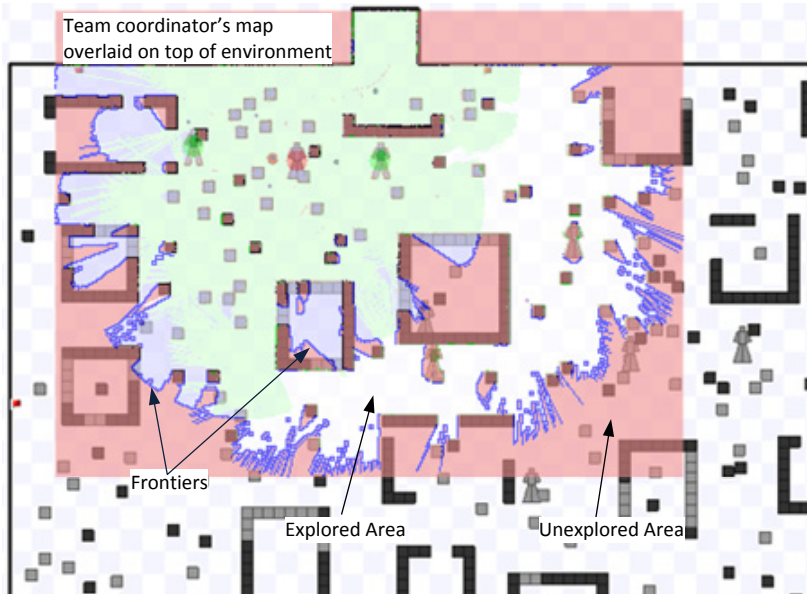


Figure 4.2: The team coordinator uses its map to identify *frontiers*, transitions between explored and unexplored space.

4.2.2 Exploration

In my implementation, exploration of the environment is directed by the robot filling the *team coordinator* role. A robot of the MidBot and MaxBot types filling the *team coordinator* role uses a frontier-based approach to guide the exploration. These robot types also have the capability of building an occupancy grid map of the environment based on the exploration results reported by the members of the team. The team coordinator identifies *frontiers* and generates *explore frontier* tasks for these locations (Section 4.8.2). Frontiers are non-obstacle boundaries between explored and unexplored space, and represent areas where the team stands to gain more knowledge about the environment.

Although both the MidBot and MaxBot robot types have the capability of iden-

tifying frontiers, the MaxBot robot type can use a path-planning algorithm (Section 4.8.4) when assigning frontiers to robots. The path-planning algorithm allows the MaxBot to make better decisions about which robot is closest to the frontier, and takes into account the fact debris may block some areas, making them inaccessible to robots incapable of traversing the debris.

Figure 4.2 shows a simulated USAR environment with the team coordinator's map overlaid on top of it. The pink areas on the map represent unexplored areas. The blue lines indicate frontiers the team coordinator has generated *explore frontier* tasks for.

The team coordinator assigns the frontier exploration tasks to robots on the team, who complete them and report back with the results. Carrying out a frontier exploration task involves moving to the frontier location, and exploring for some time. The robot reports back with the map data acquired during the exploration, assuming the exploration was successful (it didn't get stuck or lost, for example).

4.2.3 Operational Knowledge

The operational knowledge a team updates during the course of the USAR mission consists of the occupancy grid map of the environment and the list of victim locations. Although the robot filling the role of *team coordinator* expected to have the most complete picture of the operational knowledge, all other robots with the capability of maintaining their own copy of the operational knowledge will attempt to do. The MidBot and MaxBot robot types have the necessary capabilities of merging the results of exploration tasks into their own maps, as well as the capability of tracking a list

of victim locations and statuses. When these robots overhear wireless messages sent from other teammates to the current team coordinator, they use the messages to update their own operational knowledge as well.

This means a team meeting the description of a desired team in my implementation could have up to three copies of the team's operational knowledge. This is advantageous as it ensures the knowledge the team gains during operation is not vulnerable to the failure of a single robot. If the robot filling the *team coordinator* role fails and a MidBot or MaxBot robot assumes the *team coordinator* role, the distributed operational knowledge ensures the team does not have to start the mission over.

The downside to maintaining copies of operational knowledge among members of the team is that they are unlikely to be consistent. A robot could temporarily be out of range of a transmitting robot, and not receive updated knowledge. Further, communication failures could prevent the reception of information by some teammates. Inconsistencies in the operational knowledge mean if the current team coordinator fails, it is possible the robot taking over the *team coordinator* role has different information about the team's progress than the original coordinator. Although there is a potential for some duplication of effort to re-collect the lost knowledge, the team does not have to restart the entire mission due to the failure.

In addition to sharing knowledge among a team, teams which have physically encountered one another, and established a translation between their respective local coordinate systems (Section 4.3.1), are able to use overheard wireless messages from each other to update their respective operational knowledge. This helps ensure

multiple teams operating in the environment are able to benefit from knowledge they hear when within communication range. The limited range of wireless communication in my implementation means this will not occur frequently, unless the teams are operating in the same geographical area.

4.3 Simulated Disaster Environment

Although verification of my framework using real robots would be ideal, there are a number of advantages to using simulation as a means of performing experimentation. According to Cohen et al. [1989], experimentation using a simulated environment is advantageous as a simulated environment provides the necessary control over the environment to ensure experiments are repeatable. This is not only advantageous during experimentation, but also to promote comparison with methods developed by other researchers. Cohen et al. [1989] also argue simulation is advantageous as it allows the researcher access to a wider variety of environments than they would otherwise have access to, and allows environments to be constructed to test very specific scenarios. This level of control over the experiment can very difficult to achieve in the real world. It would be difficult and time consuming, for example, to ensure any debris and obstacles moves during the mission are moved back to the exact same starting locations, lighting conditions are identical, and the robots are all positioned in the exact same starting locations. Further, Etzioni and Segal [1992] argue simulated agents are advantageous as a vehicle for multi-agent systems research as they allow the peripheral problems associated with physical robots (i.e. sensors, actuators) to be simplified in order to facilitate experimentation.

Verification of my framework is accomplished in simulation using the Stage module of the Player/Stage package. Player/Stage is a robot control and simulation package, widely used in multi-robot systems research. Player provides a common interface for interacting with robots and their various sensory devices, while Stage provides simulated robots and sensors that operate in a simulated environment [Gerkey et al., 2003]. The Stage simulator has been verified against real robots, and provides adequate simulation resolution to ensure that results in simulation parallel those seen with real robots.

Control software coded against the Player module has the advantage of being easily moved from simulation to real robots. However, using Player requires that simulations be run in real time, and it means although simulation runs are deterministic they are not repeatable (the order of execution for robot control software is influenced by the operating system scheduler). As described by Vaughan [2008], using the Stage simulator alone provides two key advantages important to my work: the ability to run simulations much faster than real time, and repeatability between simulation runs. Verification of my approach necessitates a large number of simulation runs, making it infeasible to run them in real time (running my experiment trials in real time would take 188 days (Section 5.5)). The repeatability of simulation runs in Stage is particularly important during experimentation as it ensures that changes in an experimental variable are the actual cause of differences in experimental results, rather than random variation between runs.

Although my work is coded against the Stage simulator, the application programmer interfaces (APIs) provided by Stage and Player are similar enough that

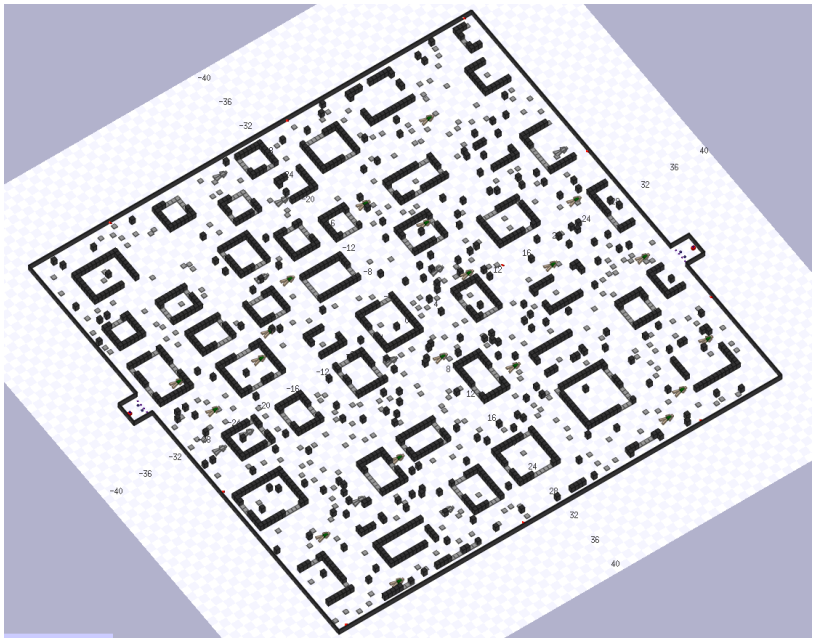


Figure 4.3: Example simulated USAR environment.

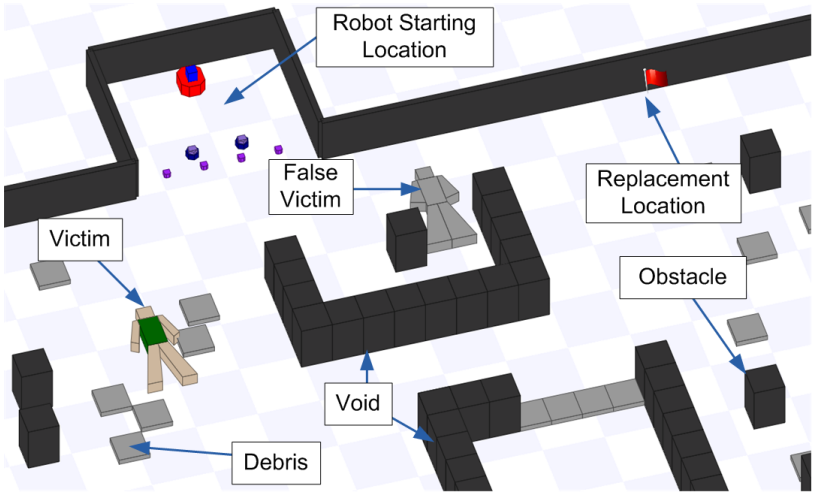


Figure 4.4: Major features of a simulated USAR environment.

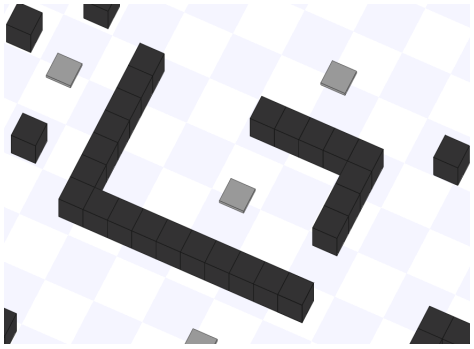
retrofitting my approach to work through Player would not require a significant effort, making translation to a real-world implementation possible.

Figure 4.3 shows a high-level view of a simulated USAR environment in my work.

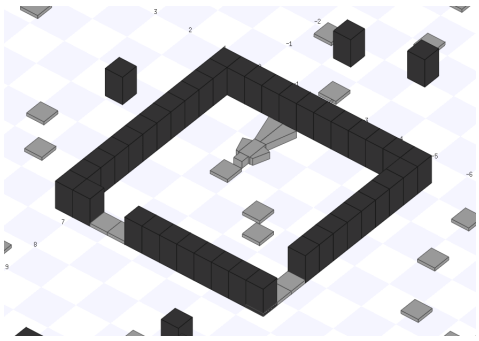
The major features of these simulated environments are highlighted in Figure 4.4. A team of robots begins operation in a walled area along the perimeter of the environment. The robot team starting locations are evenly spaced along the perimeter of the environment. Robots are placed in the starting boxes (emulating an insertion point into the environment such as a window or door) with the MaxBots in back, MidBots in the middle and the MinBots in front. The arrangement of robots is chosen to ensure the robots with sensors capable of perceiving others on the team are placed in the best position to use those sensors. This emulates a common starting deployment, such as the one used by Anderson and Papanikolopoulos [2007] in their work, allowing robots on a team to establish a shared coordinate system based on a fixed reference point. Further discussion on localization and the coordinate systems used in my implementation are found in Section 4.3.1.

Replacement robots begin operation at evenly spaced intervals along the perimeter of the environment. Replacement robots are introduced at the replacement locations after a fixed delay to simulate the insertion of replacement robots into the environment during the rescue operation.

The light-grey areas in Figure 4.4 emulate low-lying debris in the environment. In a real USAR environment this could be small piles of bricks or debris that a robot with greater mobility, such as afforded by a tracked drive system, could explore. In my simulated USAR environment, debris forms an obstacle to robots with a wheeled drive system (i.e. the MidBot and MinBot robot types). The MaxBot robots are able to drive through the low-lying debris due to their tracked drive system. Obstacles which are impassable to all robots are also scattered throughout the environment (shown



(a) A void with no debris blocking entrances; accessible by any robot.



(b) A void with debris blocking entrances; accessible only by tracked robots.

Figure 4.5: Voids or rooms in the environment.

as taller, dark grey boxes in Figure 4.4. In a real USAR environment this could be larger obstacles such as office furniture, or partially collapsed building structures.

Rectangular *voids* are scattered throughout the environment. Larger voids simulate the presence of rooms or cubicles in a typical office environment, while smaller voids simulate small enclosed areas created by failure of the building structure. Randomly sized voids are positioned in the environment, and have a variable number of openings. Some voids, as in Figure 4.5a, are accessible to all robots. Approximately 60% of all voids have low-lying debris blocking all openings, as illustrated in Figure 4.5b. These voids require a robot with the ability to traverse debris in order to be explored.

By default, the Stage simulator supports specifying whether an object in the environment acts as an obstacle to robots (all walls, for example, are marked as obstacles). To simulate differences in robot drive systems, I modified the Stage simulator so obstacles can also specify an obstacle type, and robots can identify which types of

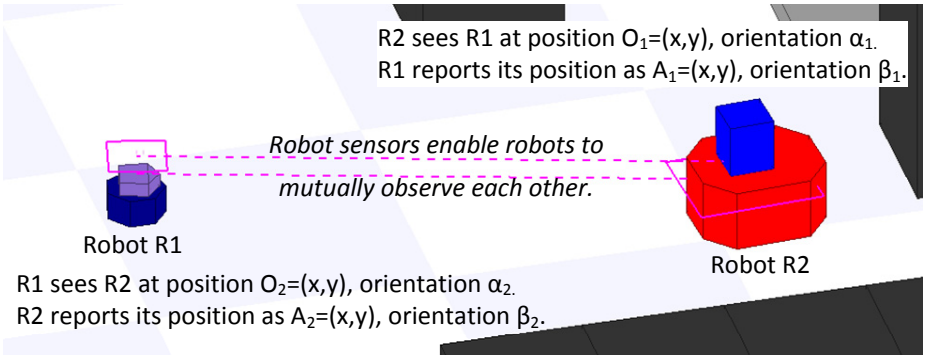


Figure 4.6: Robots mutually observing one another reconcile coordinate systems.

obstacles they collide with. In my simulated environments, each obstacle is marked with an obstacle type of *debris* or *wall*. Robots with a tracked drive system only collide with obstacles with an object type of *wall*, but can drive through obstacles with an obstacle type of *debris*. Likewise, robots with a wheeled drive system collide with obstacles with either a *wall* or *debris* obstacle type.

Realistically, a robot traveling through debris would be expected to travel slower than it could in open areas. To simulate this, robots with a tracked drive travel at half their normal speed when navigating through areas of debris.

4.3.1 Localization and Coordinate Systems

As described in Section 2.3.3, for robots on a team to cooperate effectively, its members must be able to refer to locations in space in a manner understandable to all members of the team. I assume all members of the same team share the same local coordinate system. This is provided by the stage simulator's API and allows me to remove considerations of localization from my implementation. As stated in Section 2.3.3, however, any localization scheme can be added to replace this.

To emulate a shared localization between members of a team, I assume members of the same team begin operation in a specific starting formation (similar to Anderson and Papanikolopoulos [2007]). The starting formation emulates conditions which would provide a single fixed reference point, allowing the team to establish a shared coordinate system.

Each team may start at a different origin, and thus, while there will be consistent localization within teams, there will still be different coordinate systems between teams, as would occur in team-independent localization in the real world. Thus, it is necessary for robots on different teams to be able to reconcile differences between their respective shared coordinate systems. Moving to the real world would mean the reliance on the Stage simulator's localization API would need to be removed. An implementation in a USAR test bed could, for example, use a probabilistic approach (e.g. [Fox et al., 2000] or [Martinelli et al., 2005]).

In my implementation, teams establish a translation between their respective local coordinate systems when two robots encounter one another in the environment. A robot equipped with a *robot detector* (Section 4.3.3) is able to determine (in its team's local coordinate system) the position and orientation of another robot it observes in the environment. The observed robot reports its position and orientation, in its team's local coordinate system, to the observing robot. This information can be used to establish a translation between points referenced in each team's local coordinate systems.

Figure 4.6 illustrates how robots from two teams encountering one another in the environment are able to use their robot detectors to establish a translation between

their team's local coordinate systems. Robot R1 observes robot R2 at location $O_2 = (x, y)$, with orientation α_2, in the local coordinate system shared by R1's team. Robot R2 reports its location to robot R1 as $A_2 = (x, y)$, with orientation β_2. Similarly, robot R2 observes robot R1 at location $O_1 = (x, y)$, with orientation α_1, in the local coordinate system shared by R2's team. Robot R1 likewise reports its location to robot R as $A_1 = (x, y)$, with orientation β_1.

A robot can only switch teams during an encounter, which ensures a translation between the team coordinate systems is established. When a robot switches teams, it continues to use its original team's coordinate system, but applies a translation to coordinates communicated by its new team. With any further team switch, the translation to the new team's coordinate system replaces the translation to the previous team's coordinate system.

Moving my implementation to a controlled USAR test bed (e.g. the NIST test bed [Jacoff et al., 2003]), using real robots, would be a useful avenue for future work, and would provide an increased level of realism in terms of localization and coordinate reconciliation for mapping.

4.3.2 Victims

Victims are randomly placed in my environment and represent individuals that may be trapped and/or injured. The detection of victims in a real-world USAR environment is a challenging problem unto itself potentially involving the fusion of readings from heat, sound, vision and other sensors [Murphy et al., 2000a]. As the purpose of my work is to study coordination in a USAR environment, I have made

necessary abstractions to the process of victim detection in my simulation.

To include an element of sensory heterogeneity in my work, I have implemented two types of simulated *victim detectors*. A *basic victim detector* allows a robot to detect the potential presence of a victim in the environment. In a real-world implementation, a basic victim detector would ideally be very simple and inexpensive so it could be used on the inexpensive and expendable MinBot robots. Such a sensor could identify heat signatures in the environment which mimic the temperature of a human body, for example, or flag something resembling a human form. Potential victims found using this sensor would ultimately be confirmed using a more complicated and expensive victim detector, to avoid sending human rescuers to investigate based on only a weakly confirmed hypothesis.

In my USAR simulation, a *full-featured victim detector* allows a robot to confirm potential victims in the environment. In a real-world implementation such a victim detector would employ multiple sensors and rely on complex sensor fusion to provide a more accurate identification. This sensor would necessarily be more expensive and it would be desirable to equip a smaller number of robots with these sensors. Information about the implementation of the victim detectors can be found in Section 4.9.2.

Figure 4.7 illustrates the simulated victims in my environment. Figure 4.7a shows a debris configuration that represents a *false victim* in the environment. The *basic victim detector* cannot distinguish between a false victim and a *true victim* as illustrated in Figure 4.7b. The full-featured victim detector can distinguish between the two types of victims. Thus, readings reported by the basic victim detector must be

(a) A debris configuration resembling a victim.

(b) An actual victim.

Figure 4.7: The simulated victim types allow for heterogeneity in sensors for victim detection.

confirmed using the full-featured victim detector.

4.3.3 Robot Detection

When robots encounter one another in the environment, they take advantage of being in close physical proximity and act as representatives for their team to initiate the merge and redistribution operation (Section 3.2.2). In my example implementation, I assume some robots are equipped with *robot identifier* sensors. A *robot identifier* allows one robot to detect the presence of another robot, and determine the identification, position, and orientation of the robot in relation to itself (detailed information on the implementation of *robot identifiers* is found in Section 4.9.3).

The *robot identifier* is important as it provides a means to recognize the identity of a robot in close physical proximity, which is a prerequisite to addressing messages to nearby robots using wireless communication. The *robot identifier* also provides the ability to determine the position and orientation of an observed robot. This information can be reconciled against the location and orientation the observed robot reports, in order to reconcile differences between local coordinate systems.

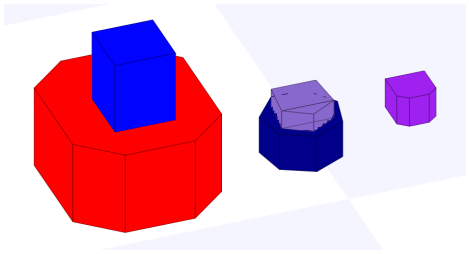


Figure 4.8: The three simulated robot types in my work.

To initiate the merge and redistribution operation between two robots, one of the two encountering robots must have a *robot identifier*. If neither robot has a *robot identifier*, the robots cannot recognize the presence of one another in the environment and thus cannot initiate the merge and redistribution operation.

4.4 Robot Types

This section describes in detail the characteristics and capabilities of the three types of robots I use in my example implementation. The MaxBot type is based on the *pioneer* robot model predefined in the Stage simulator, and the MidBot and MinBot types were created for my implementation.

Figure 4.8 illustrates the appearance of the three simulated robot types in my implementation. The *MaxBot* on the left is the largest robot and has the most powerful sensory and computation facilities, as well as a tracked drive system enabling it to move easily through debris. The *MidBot* in the middle is smaller, and is equipped with lesser computation and sensory facilities. It also relies on a less robust wheel-drive. Finally, the *MinBot* on the right represents the most limited robot in my work. Similar to the MidBot, it employs a wheeled drive, and has a further reduced

	Minbot	Midbot	Maxbot
Locomotion	Wheeled	Wheeled	Tracked
Width/Length	10 cm x 10 cm	20 cm x 20 cm	38 cm x 44 cm
Victim Sensor	Basic	Full	No
Robot Sensor	No	Yes	Yes
Sonar Sensors	5	10	3 (debris)
Sonar Range	4 m	6 m	2 m
Laser explorer/verifier	No	No	Yes
Laser Range	-	-	6 m

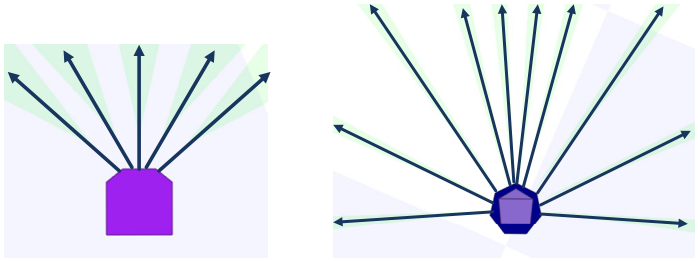
Table 4.1: Robot Types and Characteristics

	Minbot	Midbot	Maxbot
Victim Tracker	Yes	Yes	Yes
Frontier Finder	No	Yes	Yes
Maintain Team Map	No	Yes	Yes
Task Assignment	No	Yes	Yes
Planner	No	No	Yes

Table 4.2: Robot Capabilities

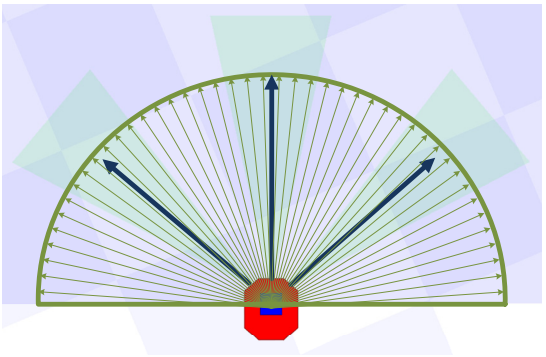
complement of sensors and computational power.

Tables 4.1 and 4.2 show the physical properties, sensory equipment and general capabilities of the three types of robots I use in my work. The following sub-sections



(a) MinBot sonar rangefinders.

(b) MidBot sonar rangefinders.



(c) MaxBot sonar and laser rangefinders.

Figure 4.9: Sensory equipment of the three robot types in my work.

describe each type of robot in detail.

4.4.1 MinBots

In my work, the MinBot robot type is the smallest robot, with the least sensory and computational abilities. Similar to Carnegie [2007], the MinBot should be a robot that is inexpensive enough as to be expendable. I assume these robots are small in size; approximately 10cm x 10cm, and use a wheeled drive system. The use a wheeled drive system means that the MinBot robots are restricted to relatively open areas with little debris.

As shown in Figure 4.9a, MinBot robots are equipped with five sonar sensors arranged on the front of the robot. The sonars have a maximum range of 4.0 meters. The MinBot robots use these sonar sensors both for obstacle avoidance, and to update their own occupancy grid map of the environment. I assume that MinBot robots lack the ability to detect other robots in the environment, meaning they cannot detect the presence of other teams in order to initiate an encounter between the teams (recall one of the two robots must have the ability to identify robots for this to occur).

Further, MinBot robots are equipped with basic victim sensors (Section 4.3.2), capable only of detecting the potential presence of victims in the environment. A better suited robot would need to confirm the presence of these potential victims.

Finally, the limited computational and memory capabilities of MinBot robots make them unsuitable to perform the duties a robot filling the *team coordinator* role normally performs. For example, the MinBot lacks the ability to analyze its occupancy grid map to detect frontiers and generate frontier exploration tasks. It is also not able to maintain the team's combined map of the environment; the MinBot can track only the area it has explored itself. The MinBot's lack of computational facilities also prevents it from assigning tasks to other robots on its own; it must rely on a better equipped robot to do this on its behalf (Section 3.5.4).

4.4.2 MidBots

The MidBot robot type is sized between the smallest and largest robot types, at 20cm x 20cm. Similar to the MinBot robot type, the MidBots are equipped with a wheeled drive system.

Figure 4.9b shows the 10 sonar sensors a MidBot is equipped with; the number and position of the sensors afford the MidBot a larger field of view, and their increased range (6m) allows for the detection of obstacles at a greater distance than the MinBot. As with the MinBot, the sonar sensors are used both for obstacle avoidance and to update the occupancy grid map.

Unlike MinBots, the MidBot robots are equipped with a *robot identifier* sensor which enables them to detect the presence, orientation and identity of other robots in the environment. This means the MidBot robots are able to *detect robots* on other teams in order to initiate a merge and redistribution operation between the two teams.

MidBot robots are equipped with full featured victim sensors (Section 4.3.2), capable of detecting the potential presence of victims in the environment. In contrast to the basic victim sensor found on the MinBot, the full featured victim sensor on a MidBot is able to confirm the presence of a potential victim at closer ranges. This makes the MidBot robot especially well-suited to carrying out the *confirm victim* tasks identified by teammates.

Compared to the MinBots, the MidBots are equipped with a greater level of computational capabilities. The MidBots have the computational capabilities required to maintain the team's combined map of the environment. This means they are able to use exploration results reported by teammates to build a combined map of the environment. The MidBots can then use this combined map along with the frontier detection algorithm to identify frontier exploration tasks, guiding the teams exploration. Further, the MidBots are able to perform task assignment, allowing for the assignment of discovered tasks to other teammates. The MidBot, however, lacks the

capability to execute the path-planning algorithm. This means its task assignments will be suboptimal compared to a MaxBot's. The lack of path planning capabilities means a robot may be assigned a task in a location it cannot access. Further, the lack of planner generated way-points makes the assignee more likely to fall into a navigational local minima inherent to the reactive navigation. According to Murphy [2000], a local minima is an area where the sum of the action vectors from the motor schemas results in a vector with zero magnitude. This might occur if, for example, the action vector attracting the robot to a destination is negated by the action vector directing the robot to avoid an obstacle. When the magnitude of the action vector is zero, the robot stops moving.

4.4.3 MaxBots

The MaxBot robot type is the largest of the three robot types, measuring 38cm x 44cm (the default size of the Pioneer model in the stage simulator). In contrast to the other robot types, the MaxBot is equipped with a tracked drive system. The tracked drive system affords the MaxBot the ability to traverse not only through open areas, but also over areas of low lying debris.

From a sensory standpoint, the MaxBot robot type is the best equipped of the three robot types. As shown in Figure 4.9c, the MaxBot robots are equipped with a laser scanner and a small complement of sonar sensors. The laser range scanner has a maximum range of 6m, similar to the sonars on the MidBot. Although most real laser scanners provide up to 180 readings over a 180 degree scan range, I assume the laser scanners on the MaxBot provide 45 readings over a 180 degree scan range. The

reduced number of scans helps minimize the computational overhead inherent with a laser scanner in the Stage simulator. A MaxBot uses the laser scanner for obstacle avoidance and to update its map of the environment.

The MaxBot's sonar sensors are positioned lower on the body of the robot to provide a means of detecting the presence of low lying debris in the environment. These sensors are only used to update the robot's occupancy grid map to indicate the presence of areas of low lying debris.

While the MaxBot robot type has very good sensory capabilities, I assume that its capabilities are geared towards identifying robots in the environment, making them ideally suited to performing coordination activities. The MaxBot is equipped with a *robot identifier* sensor, but it does not have the ability to detect the presence of victims in the environment. Since the MaxBot is ideally performing coordination activities, I presume it would be unwise to explicitly send the MaxBot in harm's way to confirm potential victims. Instead, I place this responsibility on the MidBot robot type.

MaxBots are equipped with the highest level of computational facilities, and are able to perform frontier detection, maintain the team's map and perform task assignment. In addition, the MaxBot robot type can take advantage of a path-planning algorithm when assigning tasks to robots on its team. The path-planning algorithm affords the MaxBot with the ability to determine if a teammate may be unable to reach a location due to known debris along the way. The path-planning algorithm also means the MaxBot is able to provide task assignees with a list of way-points to help them navigate to the location of a task.

The next section relates the characteristics of the robot types to the attributes, tasks and roles in my example implementation.

4.5 Attributes, Roles and Tasks

This section describes the attributes which define the characteristics and capabilities of the robots in my example implementation. It then describes the tasks in terms of minimum requirements and suitability requirements expressions formulated using these attributes. The roles in my example implementation are defined in terms of these tasks, and related to the definition of a desired team.

4.5.1 Attributes

Recall from Section 3.4.1 that attributes define the mission-specific properties of agents used to define the necessary capabilities an agent must possess in order to complete a task. The robots in my implementation use the following attributes to describe their physical properties and computational capabilities.

4.5.1.1 Physical Properties

These attributes describe physical properties of the robots, useful for determining if a robot can access areas of the environment, and how long it will take them to get to a location.

- *Physiology* – an enumerated attribute of the robot used to describe how it moves about the environment. Valid values for my implementation are *wheeled*,

representing a robot with a wheeled drive, and *tracked*, representing a robot with a tracked drive. I assume tracked robots are able to navigate through areas of low lying debris, where wheeled robots are confined to open areas with no debris.

- *Radius* – a floating point attribute describing the maximum radius of a robot, measured in meters. Can be used by the planning algorithm to determine if a robot can fit through a space. For the MinBot robot type, 0.05m. For the MidBot robot type, 0.10m. For the MaxBot robot type, 0.44m.

- *Speed* – a floating point attribute indicating the maximum speed of a robot, measured in meters per second. Useful for determining how long a robot will take to travel between two points when choosing which robot should be assigned a task. In my implementation, all robot types have a speed of 0.4 meters / second.

- *Expendability* – a floating point attribute ranging from 0 to 1.0 indicating how expendable the robot is. This attribute is a heuristic measure of how willing we are to send a robot into harm's way. In my implementation, the MinBot robot type has the highest expendability (1.0), while the MaxBot has the lowest (0.05). The MidBot robot type falls in between with an expendability of 0.25.

- *RobotModel* – a special-purpose enumerate attribute which identifies the specific model of the robot (MinBot, MidBot, MaxBot). This attribute is used during one of my experiment base-cases where there is a fixed mapping between tasks and the robot models (Section 5.5.1.2).

4.5.1.2 Computation Capabilities

These attributes are heuristic descriptions of the capabilities of the robots. They provide an indication whether a particular robot has the necessary computational and memory resources necessary to perform the more computationally and memory intensive operations of the mission.

- *CanFindFrontiers* – boolean attribute which determines whether the robot can use the frontier-finder module to analyze its map in order to generate frontier exploration tasks. Frontier detection involves analyzing the robot's map to detect transitions between explored and unexplored space. It also involves tracking the status of the generated tasks. The frontier detection algorithm and tracking of tasks would require a robot with significant computational and memory requirements (i.e. the MaxBot and MidBot robot types).

- *CanMaintainTeamMap* – boolean attribute which determines if a robot has the capability to maintain a combined map of the team's exploration progress. Robots with this capability are able to merge the maps reported by other robots as a result of carrying out *explore frontier* tasks into their own map. Although in my example implementation map merging is a relatively trivial operation, in a real-world implementation the map merging algorithm would need to account for differences in coordinate systems and attempt to compensate for noisy and inaccurate data. The MaxBot and MidBot robot types both have the capability to merge maps, in order to maintain a combined map of the environment with the team's exploration progress.

- *CanAssignTasks* – boolean attribute which indicates whether a robot can use the role-based and exhaustive task assignment algorithms (Sections 3.5.3.1 and 3.5.3.2) to assign tasks to other members of the team. These operations can be computationally intensive and require significant memory resources to track the status of ongoing task assignment operations, and are best handled by a robot with ample computational and memory facilities (i.e. the MaxBot and MidBot robot types).

- *HasPlanner* – boolean attribute indicating whether a robot is able to use the path-planning algorithm to generate navigational paths between locations on the map. A robot with this capability is better able to assign tasks as it can determine if an assignee is able to reach the destination and suggest a series of way-points to the robot which will help prevent it from getting stuck in navigational local minimas. This algorithm is the most computationally intensive in my implementation, and is reserved for those robots with the highest level of computational and memory capabilities (i.e. the MaxBot robot type).

- *HasMap* – boolean attribute indicating whether a robot is able to maintain its own occupancy grid map of the environment. In my example implementation, true for all robot types. It is, however, possible to imagine a model of robot which does not have the capability of generating an occupancy grid map. Such a robot could simply cache sensor readings without putting in the effort to assemble them into a map.

4.5.1.3 Sensory Capabilities

This section describes the attributes robots use to describe their sensory capabilities.

- *HasRangeFinder* – boolean attribute indicating whether the robot is equipped with a sonar rangefinder. Indicates the robot has 1 or more sonar sensors. The MinBot and MidBot robot types rely on sonar sensors for navigation and mapping. The MaxBot robot type has a small complement of sonar sensors lower on its body to help map the presence of debris.

- *HasLaser* – boolean attribute indicating whether the robot is equipped with a laser range finder. In my implementation, the MaxBot robot type is equipped with a laser range finder.

- *HasVictimDetector* – boolean attribute indicating whether the robot is equipped with a victim detector sensor. Indicates whether the robot has either a basic or full-featured victim detector (Section 4.3.2). Both the MidBot and MinBot robot types in my implementation have a victim detector.

- *VictimDetectorCanConfirmVictims* – boolean attribute indicating whether a robot's victim detector can positively confirm victims. The basic victim detector is unable to confirm victims, whether the full-featured vector detector can. Only the MidBot robot type in my implementation has a victim detector capable of confirming victims.

- *HasRobotIdentifier* – boolean attribute indicating whether a robot is equipped

with a *robot identifier* (Section 4.3.3). A *robot identifier* enables a robot to initiate an encounter with another robot it observes in the environment.

4.5.2 Tasks

This section describes the task types used in my example implementation in terms of the attributes describes in the previous section.

4.5.2.1 Explore

The *explore* task type performs undirected, autonomous exploration. Robots carrying out this task wander in a randomly chosen direction for a period of time while avoiding obstacles. The robot updates its occupancy grid map as it explores. The explore task is the idle task which all robots carry out in the absence of other work. As such, the priority of the explore task is set to the lowest of all task types. Further, the explore task will never be sent to another robot for completion; each robot carries out its own explore task.

Equation 4.1 describes the minimum requirements for the explore task. A robot carrying out the task must have at least a sonar or laser rangefinder. The suitability requirements (Equation 4.2) assign an equal suitability to a robot with a laser or sonar rangefinder. The minimum requirements and suitability expression for the explore task ensure any robot can carry out the task in the absence of other work.

$$M_{exp} = \{HasLaser = true \vee HasSonar = true\} \tag{4.1}$$

$$S_{exp} = \{HasLaser[100] = true \vee HasSonar[100] = true\} \tag{4.2}$$

4.5.2.2 Explore Frontier

The *explore frontier* task type involves moving to a location, performing exploration, and reporting the results back to the team coordinator (the robot carrying out the task could in fact be the team coordinator itself, and would report the results to itself). Instances of this task type are created by the frontier finder module (Section 4.8.2).

Equation 4.3 specifies that a robot must have at minimum a laser or sonar rangefinder in order to carry out an *explore frontier* task. Equation 4.4 increases a robot's suitability for robots with an occupancy grid map ($HasMap[5] = true$). Robots with a sonar rangefinder are given a preference over ones with a laser scanner ($HasLaser[10] = true \vee HasSonar[15] = true$). The intention is to favor robots with less expensive sonar sensors, because those with better sensors are more likely to be of use doing higher-level sensory tasks such as verifying victims. Finally, an increasing suitability is assigned to robots with a greater expendability factor ($Expendability[60] > 0.5 \vee Expendability[10] > 0.2 \vee Expendability[0] > 0.01$).

$$M_{fron} = \{HasLaser = true \vee HasSonar = true\} \quad (4.3)$$

$$S_{fron} = \{HasMap[5] = true \wedge$$
$$(HasLaser[10] = true \vee HasSonar[15] = true) \wedge$$
$$(Expendability[60] > 0.5 \vee Expendability[10] > 0.2 \vee Expendability[0] > 0.01)\} \quad (4.4)$$

4.5.2.3 Find Team

When replacement robots begin operation, an instance of the *find team* task type is used to help guide them deeper into the environment in search of a team to join. The schema guides the robot travel along the bearing in which it was introduced into the environment, while avoiding obstacles. The robot continues in that bearing for five minutes, until another joins the robot's team, or until the robot joins another team. Ultimately, the environment will contain walls and other obstructions which prevent the robot from strictly continuing in the exact bearing the robot was inserted into the environment. The bearing acts as a general guide to steer the robot into the environment.

The *find team* task type has no minimum requirements or suitability requirements, as any replacement robot can take it on. I assume replacement robots are introduced at the edge of the environment and are oriented roughly towards its center. *Find team* tasks guide replacement robots into the environment where they are more likely to encounter other teams already operating in the environment.

4.5.2.4 Find Victim

The *find victim* task type represents the capabilities required for a robot to find victims in the environment. As such, it is not a task which the robot explicitly puts on its task list. In my implementation, robots with the capability to do so continually search the environment for victims regardless of the tasks they execute. Thus, the *find victim* task type represents whether a robot has the capability to use the victim tracker module (Section 4.8.3).

The minimum requirements (Equation 4.5) specify the robot must have a victim detector. Robots with either the basic or full-featured victim sensor (Sections 4.3.2 and 4.9.2) are capable of finding victims. The suitability expression of the task (Equation 4.6) specifies a slight increase in suitability for robots that do not have a victim detector capable of confirming the identity of victims. Although the difference in suitability ultimately does not impact which robots can or cannot use the victim tracker module, the difference in suitability feeds into the role suitability calculations and can have an impact on a robot's overall suitability to fill a role.

$$M_{fvic} = \{HasVictimDetector = true\} \tag{4.5}$$

$$S_{fvic} = \{VictimDetectorCanConfirmVictims[5] = false \wedge HasVictimDetector[95] = true\} \tag{4.6}$$

4.5.2.5 Confirm Victim

The *confirm victim* task type defines the work necessary to confirm the presence of a potential victim located in the environment. The basic victim detector and full-featured victim detector are both capable of detecting potential victims at a distance (Section 4.9.2). Robots equipped with the full-featured victim detector are able to confirm the presence of potential victims upon moving closer to the location of the potential victim. The victim tracker module (Section 4.8.3) creates instances of the *confirm victim* task type when a potential victim detected in the environment requires confirmation.

The task involves moving to a specified location, confirming or refuting the pres-

ence of a victim at that location, and reporting the results back to the team coordinator.

The minimum requirements of confirm victim task type (Equation 4.7) state a robot must have a victim detector capable of confirming potential victims. The suitability expression (Equation 4.8) indicates the same requirements, with the majority of the suitability factor attributed to the ability to confirm potential victims.

$$M_{fvic} = \{HasVictimDetector = true \wedge VictimDetectorCanConfirmVictims = true\} \quad (4.7)$$

$$S_{fvic} = \{VictimDetectorCanConfirmVictims[80] = true \wedge HasVictimDetector[20] = true\} \quad (4.8)$$

4.5.2.6 Manage Team

The *manage team* task type is a meta-task representing the general set of capabilities required for a robot to manage the overall USAR mission. The manage team task type acts as a general description of the capabilities required for a robot to effectively fill the *team coordinator* role (Section 4.5.3.1). It is primarily used as a component of the role definitions to identify the desired capabilities of a robot filling the *team coordinator* role.

Equation 4.9 indicates a robot must have at minimum the ability to build an occupancy grid map, and the capability of merging maps in order to manage a team. The MidBot and MaxBot robot types in my implementation are both candidates to perform the team management responsibilities. The suitability expression (Equation

4.10) gives preference to robots that are able to use the planner module (i.e. the MaxBot type).

$$M_{mt} = \{HasMap = true \wedge CanMergeMaps = true\} \tag{4.9}$$

$$S_{mt} = \{HasMap[20] = true \wedge CanMergeMaps[10] = true \wedge CanFindFrontiers[40] = true \wedge HasPlanner[30] = true\} \tag{4.10}$$

4.5.2.7 Encounter

The *encounter* task encapsulates the operations necessary for two robots to encounter one another in the environment. The *encounter* task type serves three purposes. First, it provides the encountering robots with the ability to directly share their knowledge of the mission. Second, although my current implementation uses a shared coordinate system, the *encounter* task is also an opportune time for robots to reconcile differences between one another's coordinate systems and re-localize. Third, this task provides robots on different teams with the opportunity to complete the team merge and redistribution operation (Section 3.7). The *encounter* task is unique in my implementation as it involves two robots cooperatively completing the encounter. Section 4.7.1 describes the task in detail and explains how both robots cooperate to complete the encounter.

Encounter tasks are generated by the encounter manager module (Section 4.7.1) when a robot with a *robot identifier* sensor observes another robot. The encounter manager will only generate an *encounter* task if the robot has not completed a previous encounter within 1 minute. This prevents robots from continually engaging in

Task Types	Roles: Team Coordinator	Explorer	Explorer / Verifier
Explore	3%	5%	5%
Explore Frontier	4%	60%	10%
Find Team			
Find Victim	3%	35%	5%
Confirm Victim			50%
Manage Team	80%		20%
Encounter	10%		10%
	100%	100%	100%

Table 4.3: Tasks normally expected of the roles in my example implementation.

encounters with one another.

The *encounter* task type has no minimum requirements (Equation 4.11), and a suitability expression (Equation 4.12) which assigns full suitability to a robot with a *robot identifier*. The choice of minimum requirements and suitability expressions ensures a robot with a *robot identifier* can encounter another robot lacking a *robot identifier*, yet still cooperatively complete the encounter.

$$M_{mt} = \{\} \tag{4.11}$$

$$S_{mt} = \{HasRobotDetector[100] = true\} \tag{4.12}$$

4.5.3 Roles

As previously introduced in Section 3.4.3, a role provides a heuristic description of the tasks a robot filling that role is normally expected to carry out. This section describes the roles I use in my example implementation. To accomplish the USAR

	Roles		
Robot Type	Team Coordinator	Explorer	Explorer / Verifier
MinBot	11	124	24
MidBot	73	56	86
MaxBot	96	52	42

Table 4.4: Calculated suitability of robot types to fill each role, based on the attributes of the robot types.

mission, I define the roles *team coordinator*, *explorer*, and *explorer/verifier*. Table 4.3 shows the roles I use in my example implementation and which of the task types described in Section 4.5.2 are normally expected of each role. The percentages shown in the table are a heuristic description of the types of tasks normally expected of each role. The percentages describe the relative importance of being able to carry out each type of task for a robot filling that role. The values in my implementation were set at values I deemed reasonable, and then refined through experimentation to verify the behaviour of robots performing role check operations, and teams undergoing the merge and redistribution operation. An area of potential future work would be to study applying techniques to enable the expected task mix to be refined as the robots operate (Section 6.4).

Table 4.4 shows the calculated suitability of each robot type to fill the roles in my example implementation. The suitability is an unsigned integer, representing the relative suitability of a robot to fill a role, compared to the other robot types. The actual numerical suitability values are not as important as the values relative to oth-

ers. Reading across the rows of the table, the suitability values may be compared to determine the relative suitability of a robot type for each of the roles. A MinBot, for example, is strongly suited to the *explorer* role, and weakly suited to the *team coordinator* role. Reading down the columns of the table, the suitability values represent the relative suitability of each robot type to fill that role. Sections 4.5.3.1, 4.5.3.2 and 4.5.3.3 discuss the work normally expected of a robot filling each role, and the implication of the role being filled by each of the robot types in my example implementation.

4.5.3.1 Team Coordinator

A robot filling the *team coordinator* role is expected to guide the overall USAR mission for a team. As shown in Table 4.3, the task mix normally expected of a robot filling the *team coordinator* role consists primarily of the *manage team* task type (Section 4.5.2.6), and a small element of exploration. The robot filling the *team coordinator* role is expected to be able to explore frontiers, and will thus assign some *explore frontier* tasks to itself, as its team coordination responsibilities allow. The team coordinator will ideally use the frontier finder module (Section 4.8.2) to identify *explore frontier* tasks based on areas requiring exploration. It assigns these tasks to members of the team for completion. The team coordinator will use the results of the *explore frontier* tasks to add knowledge to its occupancy grid map, which drives the identification of further *explore frontier* tasks.

The team coordinator is also responsible for assigning tasks identified by other team members. This typically involves assigning *confirm victim* tasks (Section 4.5.2.5),

found by team members using their victim tracker modules (Section 4.8.3), to appropriate members of the team.

The best suited robot to fill the *team coordinator* role is the MaxBot robot type (Table 4.4). In my example implementation, the MaxBot can make use of the planner module (Section 4.8.4) when assigning tasks to members of the team. This ensures a task is not assigned to a robot that is not able to move to the task location (i.e. the path is blocked by known debris or some other obstruction) and gives a more accurate picture as to which robot is closest to the task location. Further, the team coordinator uses the planner to provide a series of way-points which the task assignee can use to navigate to the task location. The way-points help reduce the chance of the robot becoming stuck in local minima while navigating, or wasting time traveling down the wrong path. An implementation on real robots would likely result in a larger advantage to the MaxBot robot type than in my simulated implementation. Where the MaxBot robot type would be likely to use a powerful computer, the MidBot might rely on a lower powered embedded computing platform. These differences in computational abilities are not reflected in my current implementation; future work to increase realism in my simulation would need to emulate the simulated time it takes to complete various operations as would be expected on real hardware (Section 6.4). Further, my implementation does not impose different memory constraints on the simulated robot types. Differences in available memory between the MidBot and MaxBot types would impact the size of maps each could maintain, and the amount of knowledge each was capable of remembering.

Due to computational and memory constraints, a MidBot robot filling the *team*

coordinator role will not be able to use the planner module. Not having access to the planner module means the MidBot robot type will not be able to consider whether assignees are knowingly unable (from the map) to access the task location, and will rely on the Cartesian distance when considering the closest robot. Further, a lack of way-points to the task location results in the assignee relying on purely reactive navigation to the task location. An assignee could become lost or stuck in a local minima while attempting to navigate to the task location.

The MinBot robot type is the poorest suited robot to fill the *team coordinator* role. Since the MinBot cannot merge maps resulting from exploration tasks, a unified view of the environment cannot be maintained. Further, the MinBot lacks the capability to use the frontier finder module, which means the team cannot perform exploration in a coordinated manner. The MinBot type cannot assign tasks, which means any *confirm victim* tasks identified will not be assigned to an agent capable of completing them. Should a MinBot robot fill the *team coordinator* on a larger team for an extended period of time, the result will be a gradual breaking up of the team as teammates fall back to performing unguided exploration. This will result in the team gradually spreading apart until the teammates are no longer in radio range of one another.

4.5.3.2 Explorer/Verifier

A robot filling the *explorer/verifier* role is expected to perform exploration under the direction of the team coordinator, and to carry out victim verification tasks. The weights of the expected tasks (Table 4.3) place an emphasis on the *confirm victim* tasks, representing the relative important of this type of task for a robot filling the

explorer/verifier role. The inclusion of the *manage team* task type in the expected task mix acts to encourage robots with some ability to manage a team to fill the *explorer/verifier* role. The intention is to encourage the team composition to include some robots which could act as a backup for the robot filling the *team coordinator* role.

The mix of tasks and weights make the MidBot robot type the best suited to fill the *explorer/verifier* role (Table 4.4). The MidBot robots have a full-featured victim detector capable of confirming the identity of potential victims in the environment, making them ideal for completing *confirm victim* tasks. A MidBot has the capability of assigning tasks and is able to use the frontier finder module, making them an ideal backup robot to fill the *team coordinator* role if required.

The next best suited robot type to fill the *explorer/verifier* role is the MaxBot robot type (Table 4.4). The MaxBot robot type lacks a victim detector of any kind, meaning it is unable to execute the *confirm victim* tasks which make up a large part of the expectation of the role. However, a MaxBot filling the *explorer/verifier* role provides a backup to the robot filling the *team coordinator* role, should that robot fail. Since it can be expected there will be far fewer MaxBot robots present in the environment than MidBots, a MaxBot would not realistically fill the *explorer/verifier* role for long. Strong team coordinators would likely be picked up by other teams lacking that capability.

The least suited robot to fill the *explorer/verifier* role is the MinBot robot type (Table 4.4). As with the MaxBot robot type, the MinBots lack the capability to confirm potential victims, and further do not make suitable replacement team co-

ordinators. The range of expected agents filling the *explorer* role on a team means surplus MinBots are more likely to fill the *explorer* role.

4.5.3.3 Explorer

As shown in Table 4.3, robots filling the *explorer* role are primarily expected to perform exploration under the direction of the team coordinator. Robots filling this role are also expected to search for potential victims while exploring.

The MinBot robot type is best suited to fill the *explorer* role (Table 4.3), attributable to its highly expendable nature. The MinBot robot type is also simplistic in nature, and lacks most of the capabilities of the MidBot and MaxBot robots which make them better suited to fill other roles. The plentiful quantity of MinBot robots compared to the other robot types make it likely the *explorer* role will be filled by a MinBot robot over the other types.

4.5.4 Desired Team

Recall from Section 3.4.4 that a desired team defines the roles and range of each which should be present on a team in order to complete the mission. Figure 4.10 shows the definition of a desired team used in my example USAR implementation. I assume a single robot will fill the *team coordinator* role, 1 to 2 robots will fill the *explorer/verifier* role, and 3 to 10 robots will fill the *explorer* role. The teams I use in my evaluation begin operation as shown in Figure 4.10, with one MaxBot filling the *team coordinator* role, 2 MidBots filling the *explorer/verifier* role, and 4 robots filling the *explorer* role.

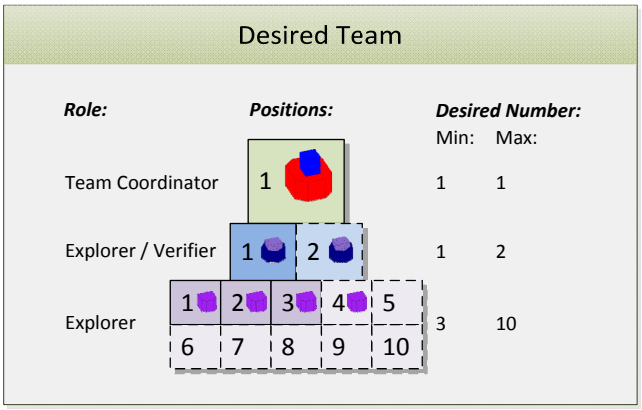


Figure 4.10: A desired team in my example implementation. Teams begin with 1 MaxBot, 2 MidBots, and 4 MinBots filling the roles.

This mixture of robot types was chosen to provide a reasonable level of redundancy for robots filling the *team coordinator* role, and to ensure a sufficient number of robots are available to execute the *confirm victim* tasks identified over the course of the mission. The small range of desired robots (1 to 2) filling the *explorer/verifier* role was chosen to ensure a level of redundancy for confirming victims, while ensuring a team does not end up with a large number of highly capable agents. The wide range of robots desired in the *explorer* role (3 to 10) reflects the fact MinBot robots are the most plentiful, yet unreliable agents in my implementation. Replacement or lost MinBot robots could be encountered at any time, and the large upper range in the *explorer* role provides ample room to accommodate these robots. Further, MinBot robots can become lost or disabled, reducing the number of robots available on the team. The relatively large difference between the upper and lower range for the *explorer* role accommodates these variations.

The definition of a desired team for my example implementation was arrived at

through preliminary experimentation, given the framework described in Chapter 3, the control elements described in Section 4.6, and the domain described in Section 4.3. The composition of a desired team would likely differ in a real-life implementation, and would need to draw on research studying the use of real robots in USAR (e.g. [Murphy et al., 2000a]).

4.6 Autonomous Control

This section describes the autonomous control system used by robots in my example implementation. The section begins with a description of the schema-based approach I use to process sensory input and generate a corresponding action vector to guide the robot's movement. I also describe how the autonomous control system interacts with the framework-specific software to add a deliberative layer to the reactive control provided by the schemas.

Figure 4.11 shows the perceptual schemas in my implementation which interpret the raw sensor data from the victim detector, *robot identifier* and various range sensors. The figure also shows the motor schemas which provide the framework software with navigational primitives enabling the robot to move to locations, avoid obstacles, and recover from situations where it has become stuck. This approach is analogous to the schema-based approach developed by Arkin [1987].

The outputs from some of the perceptual schemas (Figure 4.11) feed into mission-specific modules in the framework software which are responsible for identifying tasks based on the perceived sensory information. Carrying out a task provides input to the autonomous control system to enable and disable motor schemas as required by

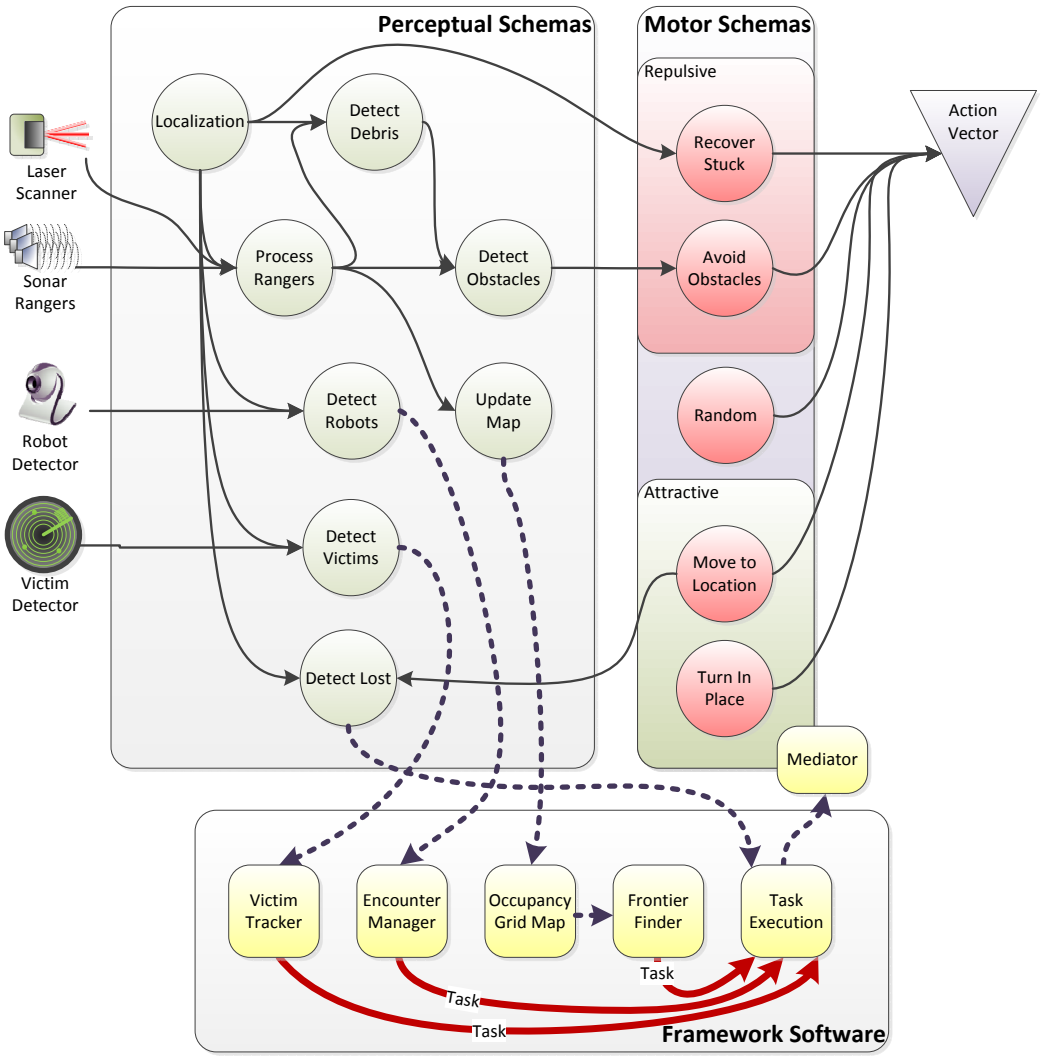


Figure 4.11: The interactions between the schemas in my implementation and the framework software.

the task. Thus, the mission-specific modules and task execution provide a deliberative layer on top of the basic schema-based autonomous control system, making my approach analogous to other hybrid architectures, such as AuRA [Arkin and Balch, 1997].

The following sections explain the operation of the individual schemas pictured

in Figure 4.11 and describe how they interact with one another and the framework software.

4.6.1 Perceptual Schemas

4.6.1.1 Localization

The *localization* perceptual schema (Figure 4.11) is responsible for determining the current location of the robot, in the robot's local coordinate system. In my work all robots share the same coordinate system, and determine their location using a Stage simulator API call (Section 4.3.1).

The *localization* perceptual schema determines the current position, $P_{robot} = \{x, y\}$, and orientation, α_{robot}, the robot faces. The robot position and orientation are inputs to all other perceptual schemas as they must know the robot's location and orientation in order to perform their function.

In addition to reporting the position and orientation of the robot, the *localization* perceptual schema also detects situations where the robot has become stuck. The Stage API provides an *IsStalled* property on the robot that determines if it has collided with some obstacle which prevents it from moving. An implementation on real robots could, for example, report the robot as stuck if its position has not changed in some time despite the robot receiving motor inputs directing it to move.

4.6.1.2 Process Range Data

As shown in Figure 4.11, the *process range data* perceptual schema provides other perceptual schemas with range data from sonar and laser rangefinders. In my im-

plementation, the *process range data* perceptual schema primarily provides a single representation of the sonar and laser scan data retrieved from the Stage API. An implementation of this schema on real robots would need to perform more advanced sensor fusion. According to Diosi and Kleeman [2004], sensor fusion is the process by which information from different sensor types is combined, in order to reduce the incidence of false readings and to take advantage of complementing characteristics of the different sensor types. A laser range scanner, for example, has difficulties detecting mirrors and glass doors, while a sonar sensor is able to detect these obstacles.

In my implementation, the *process range data* schema uses data from any sonar sensors and laser range scanners defined for the robot to create a combined list of rangefinder data. Each reading consists of a start point, direction and distance reading for the rangefinder scan, $S_{scan} = (x, y, z), \alpha_{scan}, D_{scan}$, relative to the robot. This establishes where the sensor is located on the robot's body, in which direction the scan beam went, and the measured distance reading. For scan beams from a laser rangefinder, α_{scan} is the direction of the laser beam as it was emitted from the scanner.

4.6.1.3 Detect Debris

Using input from the *localization* and *process range data* perceptual schemas, the *detect debris* perceptual schema identifies low-lying obstacles which a robot is unable to detect using its sonar or laser rangefinder sensors. The schema adds the debris location to a rolling list of debris locations.

Figure 4.12 illustrates how the *detect debris* perceptual schema works. If the *localization* perceptual schema indicates the robot has become stuck (Section 4.6.1.1),

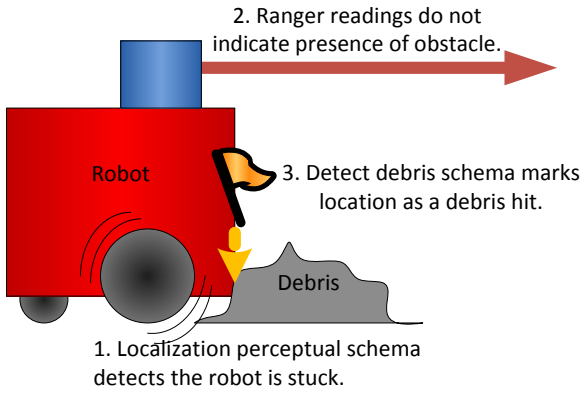


Figure 4.12: The *detect debris* perceptual schema allows a robot to track the location of debris its rangefinders cannot see.

the schema loops through all rangefinder readings reported by the *process range data* perceptual schema to determine if any reading is within a short distance of the robot. A short distance reported by a rangefinder indicates the potential presence of a large obstacle such as a wall. In this case, the schema would not report the presence of debris. If, however, no rangefinder reading indicates the presence of a large obstacle, the schema will log the robot's current location as a debris hit. The fact the robot got stuck while the rangefinders returned no obstacle hits indicates the robot hit an obstacle it could not detect.

The *detect debris* perceptual schema enables robots which are incapable of navigating through low-lying debris to detect the location of this debris, in order to avoid a collision with this location in the future (Section 4.6.1.4). Where a robot has the capability to navigate through the debris, it would not become stuck and would continue navigating unimpeded without logging the location as debris. I equip the MaxBot robot type with a small complement of sonar rangefinders positioned low on the robot's body, enabling it to sense and map debris it encounters in the environment

(Section 4.4.3).

4.6.1.4 Detect Obstacles

The *detect obstacles* perceptual schema uses the range and direction readings from the *process range data* (Section 4.6.1.2), *detect debris* (Section 4.6.1.3), and *detect robots* (Section 4.6.1.6) schemas to generate a list of obstacles the robot should avoid. The schema generates a list of vectors in magnitude / direction format specifying the distance and direction of each detected obstacle, $O = \{\vec{O_1}, \vec{O_2}, \cdots, \vec{O_n}\}$. $\vec{O_i} = \begin{bmatrix} D_i \\ \\ \phi_i \end{bmatrix}$, where D_i is the distance and ϕ_i is the direction to obstacle O_i.

When interpreting the rangefinder readings, only those rangefinder readings where the distance returned is less than the maximum range of the rangefinder are added to the list of obstacles. The height at which the rangefinder readings were taken is also taken into account when adding them to the list of obstacles. Where the robot has a wheeled drive system, both low-lying debris and higher obstacles are added to the list. Where the robot has a tracked drive system, I assume the robot can drive over obstacles less than a height of 9cm.

Similarly, only those debris points within a fixed distance of the robot (in my implementation, 2 meters) are added to the list of obstacles. Any robots in range are added to the list of obstacles.

The result is a consolidated list of vectors specifying the distance and direction to all obstacles the robot should avoid.

4.6.1.5 Update Map

The *update map* perceptual schema uses the readings from the *process range data* (Section 4.6.1.2) and *detect debris* (Section 4.6.1.3) schemas to update the robot's occupancy grid map. The detect robot schema is not used to update the map as it would be undesirable to specifically show the position of robots on the map. More information on how the occupancy grid map is updated is found in Section 4.8.1.1.

As shown in Figure 4.11, the schema updates the robot's occupancy grid map of the environment using the rangefinder readings and debris hits found.

4.6.1.6 Detect Robots

The *detect robots* perceptual schema uses the *robot identifier* to generate a list of robots currently observed. As shown in Figure 4.11, the output from the *detect robots* perceptual schema feeds into the encounter manager (Section 4.7.1). The encounter manager is a framework-specific module, responsible for initiating encounters with observed robots in the environment.

The *detect robots* schema maintains a list of vectors in magnitude / direction format specifying the distance and direction of each detected robot, $R = \{\vec{R_1}, \vec{R_2}, \cdots, \vec{R_n}\}$. $\vec{R_i} = \begin{bmatrix} D_i \\ \\ \phi_i \end{bmatrix}$, where D_i is the distance and ϕ_i is the direction to robot R_i.

4.6.1.7 Detect Victims

The *detect victims* perceptual schema uses the victim detector (Section 4.9.2) to generate a list of potential victims currently observed. As illustrated in Figure 4.11, the output from the *detect victims* perceptual schema feeds into the *victim*

tracker (Section 4.8.3). The victim tracker is a mission-specific module in my example implementation, responsible for creating *confirm victim* tasks based on the victims requiring confirmation observed by the robot.

The *detect robots* schema maintains a list of vectors in magnitude / direction format specifying the distance and direction of each detected victims, $V = \{\vec{V_1}, \vec{V_2}, \cdots, \vec{V_n}\}$. $\vec{V_i} = \begin{bmatrix} D_i \\ \phi_i \end{bmatrix}$, where D_i is the distance and ϕ_i is the direction to victim V_i.

4.6.1.8 Detect Lost

The *detect lost* perceptual schema tracks the distance a robot has traveled while carrying out a task that involves moving to a location through the use of the *move to location* motor schema (Figure 4.11). The perceptual schema is reset when the destination for the move to location motor schema is set. At this point, the schema calculates the distance from the robot's current location to the destination. As the robot moves towards its destination, the *detect lost* schema tracks the cumulative distance the robot travels. Despite the use of way-points to help guide a robot to its destination, changes to the environment due to structural collapse or inaccurate knowledge used in the planning process can still result in a robot failing to reach its destination. Where the task was assigned by a robot not capable of using the planner module, no way-points would be available to help guide the robot to its destination, further increasing the chance of the robot getting lost. The *detect lost* perceptual schema provides the robot with a means of detecting scenarios where it has traveled much further than necessary to reach its destination.

If the robot has traveled more than five times the expected distance, the schema

generates a signal which the framework software uses as a cue to indicate the robot has become lost. The five times expected distance factor was arrived at through initial experimentation, and takes into account the fact a robot will not be able to travel in a straight line distance to arrive at a task location. Future work could improve on this by having the robots on a team adjust this value down, based on how far over the expected distance robots typically end up traveling to arrive at their destination.

When the robot determines it has become lost, execution of the current task is suspended and the robot moves on to the next task. Where the suspended task is an *explore frontier* task, the task is abandoned and the exploration results obtained so far are sent back to the team coordinator. Future exploration from the perspective of other agents could reveal another route to reach the frontier, or result in the frontier having been explored. Where the suspended task is a *confirm victim* task, the task is re-queued and the robot will attempt to complete it at a later time.

4.6.2 Motor Schemas

This section lists the motor schemas the robot uses to generate action vectors which guide the robot to move to locations, avoid obstacles, and recover from situations where it has become stuck.

4.6.2.1 Avoid Obstacles

As illustrated in Figure 4.11, the *avoid obstacles* motor schema uses the obstacles identified by the *detect obstacles* perceptual schema (Section 4.6.1.4). It generates a corresponding action vector repulsing the robot away from the identified obstacle

Algorithm 6 Determining the obstacle avoidance action vector.

Require: $O = \{\vec{O_1}, \vec{O_2}, \cdots, \vec{O_n}\}$, where $\vec{O_i}$ are the vectors pointing to each obstacle.

Require: M = maximum obstacle distance

$\vec{A} \leftarrow$ empty action vector.

for all $\vec{O_i} \in \{\vec{O_1}, \vec{O_2}, \cdots, \vec{O_n}\}$ **do**

$$\vec{R} = \begin{bmatrix} \frac{(M-D_i)^2}{Mn} \\ \\ \pi + \phi_i \end{bmatrix}$$

$$\vec{A} = \vec{A} + \vec{R}$$

end for

return $\vec{A}$

points.

Algorithm 6 shows how the schema generates the resulting action vector. The algorithm loops through the n vectors pointing to obstacles found by the *detect obstacles* perceptual schema and calculates the average repulsive force required to avoid the obstacles. The repulsive forces are in the opposite direction of the detected obstacles.

Assuming a maximum obstacle distance (M) of 10 meters, Figure 4.13 illustrates how the repulsive force exerted by obstacles grows exponentially as a robot nears the obstacles. The exponential growth used to determine the repulsive force of obstacles was determined through experimentation. The exponential function results in the robot ignoring distant obstacles, while reacting strongly to close obstacles. This helped to minimize oscillations I observed when using a linear obstacle avoidance function; robots would tend to cycle back and forth through tight areas.

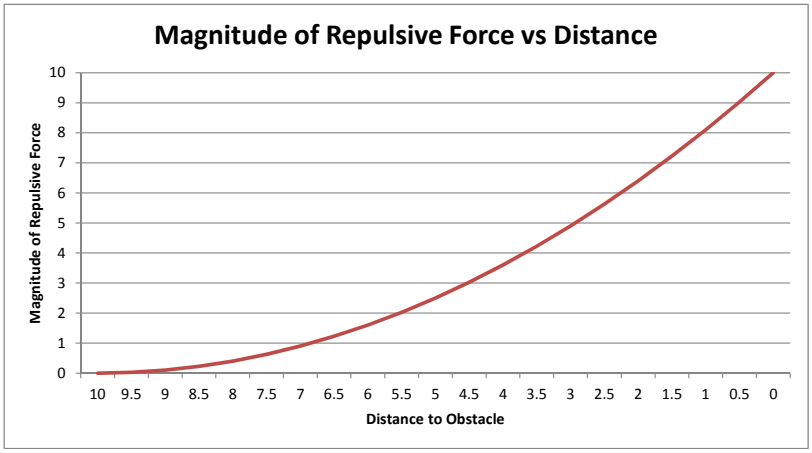


Figure 4.13: The repulsive force grows exponentially as the robot nears obstacles.

4.6.2.2 Move To Location

The *Move to Location* motor schema enables a robot to move to a specified location. As shown in Figure 4.11, it does not depend on any perceptual schemas for input. Instead, the schema acts to guide the robot through a series of way-points, determined by the task the robot is currently carrying out, to arrive at its final destination. Where a robot was assigned the task by a team coordinator incapable of using the planner module, the way-points list would consist of only the task location itself (since only the planner module can produce way-points).

Tasks involving moving to a location may have either a task location or a set of way-points to the location provided by the path planner. When the robot begins carrying out the task, however, it may be in a different position than that assumed by the robot planning the path because of stale information or robot movement. The first way-point on the path is thus not necessarily the most efficient place for the robot carrying out the task to begin traveling. The closest way-point, and thus the place

Algorithm 7 Finding closest way-point.

Require: $Y = \{W_1, W_2, \cdots, W_n\}$, where W_n is the ending way-point.

Require: D current destination.

if No current destination, D **then**

$D \leftarrow$ closest way-point W_i in Y to robot's location.

Remove $\{W_1 \cdots W_i\}$ from Y

end if

$A \leftarrow$ action vector attracting robot to D

to begin moving toward is found using Algorithm 7. The way-points prior to this are removed from the plan. The robot navigates through the remaining way-points in order until it reaches its destination.

Given the robot's current destination, Equation 4.13 illustrates the attractive action vector A which guides the robot to the destination. D_g is the distance between the robot and the destination, and ϕ is the angle between the robot and the destination coordinates.

$$A = \begin{bmatrix} D_g \\ \phi \end{bmatrix} \tag{4.13}$$

4.6.2.3 Turn in Place

The *turn in place* motor schema generates an action vector which commands the robot to turn in place in a clockwise direction. The motor schema is used by the *encounter* task (Section 4.7.1.1) to turn a robot in place so it can detect a robot which has requested an encounter.

The schema generates an action vector pointing in the direction $\frac{\pi}{2}$. The turn in place motor schema is handled specially by the autonomous control system to command the robot's motors to turn the robot in place without moving forward or backward.

4.6.2.4 Random

The *random* motor schema helps prevent the robot from becoming stuck in a local minima.

The *random* motor schema generates a small magnitude action vector pointing in a randomly generated direction in the range $[-\pi, \pi]$. A new direction is chosen every second. Adding this small random vector to the robot's overall action vector ensures the magnitude will never sum to zero, preventing the robot from getting stuck in a local minima resulting from a zero magnitude action vector.

4.6.2.5 Recover Stuck

The *recover stuck* motor schema uses the "stuck" indicator provided by the *localization* perceptual schema (Section 4.6.1.1) to recognize a situation where the robot has become stuck. In response, the *recover stuck* motor schema generates an action vector commanding the robot to back up and attempt to free itself. When the *recover stuck* motor schema generates an action vector, it suppresses the output of all other motor schemas.

The schema attempts to free the robot by "wiggling" the robot while commanding it to move backwards. To "wiggle" the robot, the schema generates a new random direction pointing behind the robot in the range $[\frac{-\pi}{4}, \frac{\pi}{4}]$. A new value is generated

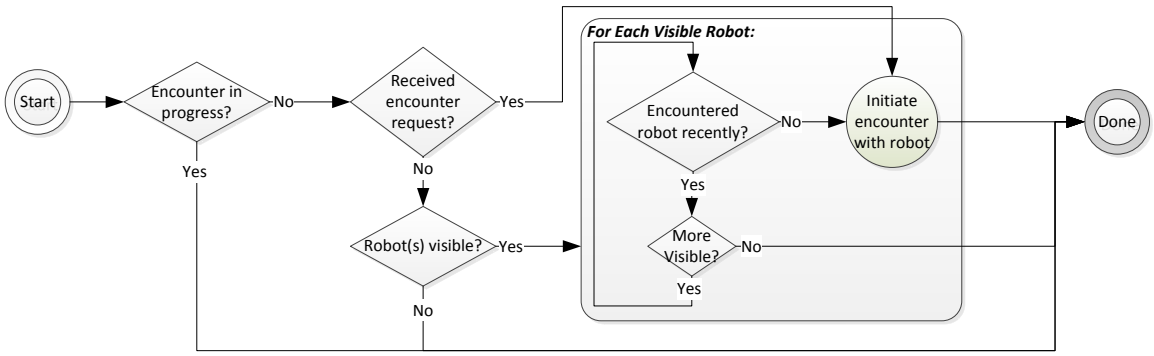
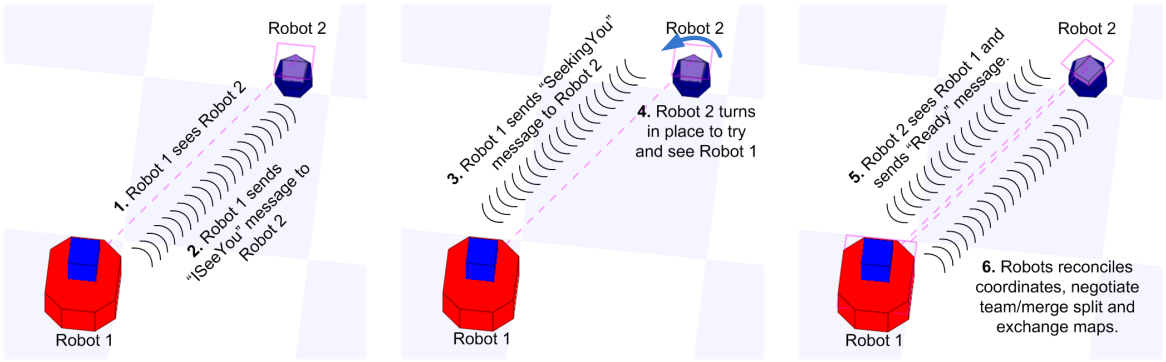


Figure 4.14: Encounter manager.

twice per second. If the *localization* perceptual schema reports the robot has become freed, the *recover stuck* motor schema commands the robot to reverse in the current direction for 2 seconds. At this point, the schema resets itself and stops generating an output. The robot can then continue normal autonomous movement. The *detect debris* schema will have recorded the location where the robot got stuck, which would then be included in the action vector generated by the avoid obstacles motor schema.

4.7 Framework-Specific Modules

This section describes modules I implemented to support the methodology described in Chapter 3. These modules provide supporting functionality necessary for the robots in my example implementation to manage teams and distribute tasks to one another.



(a) (b) (c)

Figure 4.15: Illustration of an encounter between two robots with robot sensors.

4.7.1 Encounter Manager

This section describes the encounter manager module, which runs on each robot and is responsible for identifying opportunities to perform an *encounter* with another robot. In my implementation, an encounter provides robots on two teams with an opportunity to perform the team merge and redistribution operation described in Section 3.7. In my implementation, an encounter also provides two robots with the ability to reconcile any differences between their team's local coordinate systems, and provides an opportunity for the robots to make use of close physical proximity to share mission-specific knowledge.

An encounter is a coordinated maneuver which takes place between two encountering robots. Both robots involved in the encounter carry out *encounter* tasks, and use wireless communication to coordinate the encounter. Figure 4.15 illustrates the sequence of events which occur when a robot encounters another robot in the environment. In Figure 4.15a(1), Robot 1 observes Robot 2 and initiates an encounter

by sending an "I See You" message to state its intention to initiate an encounter (Figure 4.15a(2)). Robot 2 has a *robot identifier* sensor (Section 4.3.3) and will attempt to turn in place until Robot 1 is observed. If Robot 2 has a *robot identifier* sensor, it sends a "Seeking You" message to Robot 1 to acknowledge the encounter request message and state that it is maneuvering into position for the encounter (Figure 4.15b(3)). Robot 2 turns in place until Robot 1 is observed (Figure 4.15b(4)). Where Robot 2 does not have a *robot identifier* sensor, it will simply stop moving as it cannot attempt to sense the other robot. Finally, Robot 2 sends a "Ready" message to Robot 1 indicating it is in position and ready to begin the encounter.

The encounter manager uses the *detect robots* perceptual schema (Section 4.6.1.6) to determine when a robot is being observed by the *robot identifier* sensor. The encounter manager tracks the time of last encounter for each robot it knows of, ensuring the robot on which it runs does not repeatedly engage in frequent encounters with the same robots. As shown in Figure 4.14, the encounter manager iterates through the list of observed robots reported by the *detect robots* perceptual schema, and determines if an opportunity exists to perform an encounter with one of the robots. If an observed robot has not been previously seen, or the time since the robot was last involved in an encounter is greater than a set time (1 minute in my implementation), the encounter manager attempts to initiate an encounter with that robot. It does this by creating a new *encounter* task, specifying an encounter is to be initiated with the observed robot. The *encounter* task has a higher priority than the other task types, and the robot begins carrying it out immediately, suspending any other work.

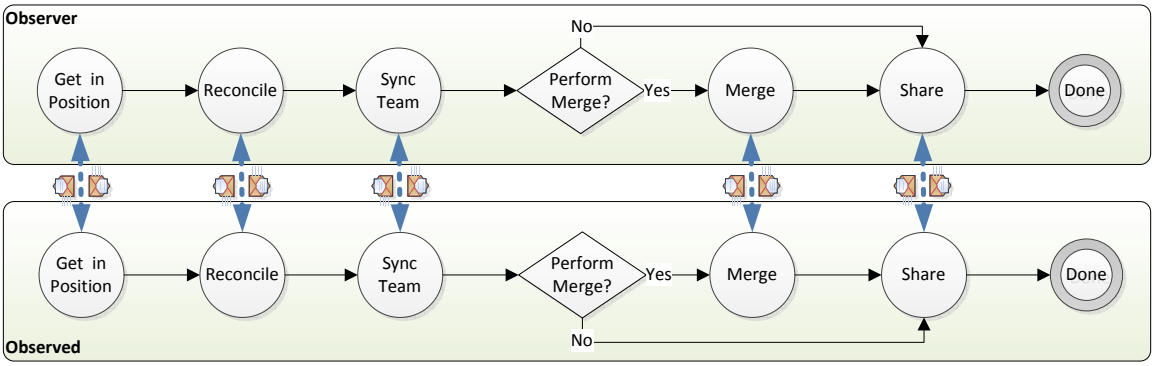


Figure 4.16: Coordination between *encounter* tasks of encountering robots.

The encounter manager also tracks incoming requests via wireless communication to initiate an encounter. As shown in Figure 4.14, if the robot is not already engaged in an encounter it will add an *encounter* task to its task list specifying a request was received to initiate an encounter. If the robot is already engaged in an encounter, it replies with a message indicating it is currently busy and cannot participate in the requested encounter.

4.7.1.1 Encounter Task

Both of the encountering robots carry out their own *encounter* task. The two tasks coordinate using wireless communication between the two robots, as illustrated in Figure 4.16. The figure refers to two robots, the *observer* and the *observed*. The observer is the robot that initiated the encounter by observing the observed robot. All of the steps in the encounter process make use of timeouts, ensuring the robots do not wait for one another indefinitely, should communication or equipment failures occur (a comprehensive discussion of the impact of failures and uncertain knowledge on the team merge and redistribution process can be found in Section 3.7.1).

The first step the *encounter* tasks accomplish is to ensure both robots are in position for the encounter. The robots position themselves to ensure they can both observe one another using their respective robot sensors, and when in position stop moving. Where only the observer robot has a robot sensor (i.e.. in the case where a MidBot or MaxBot observes a MinBot), the observed robot has no ability to position itself so the observer is in view. In this case, the observed robot simply stops moving to allow the observer to get a stable position and orientation reading using its *robot identifier* sensor.

If the observed robot has a robot sensor, it will get into position by turning in place until the observer is within view of its *robot identifier* sensor. While rotating, the observed robot periodically sends a "Seeking You" message to the observer robot, ensuring it is aware the robot is attempting to move into position. Once the observer robot is visible, the robot stops turning and sends a "Ready" message to the observer.

As shown in Figure 4.16, when both robots are in position, the *encounter* tasks move to the *reconcile* step. The reconcile step provides the robots with an opportunity to reconcile differences in their teams' local coordinate systems (Section 4.3.1). Each robot sends a message to the other, indicating its current location and orientation in its own local coordinate system. The combination of this and the position and orientation of the other robot reported by the *robot identifier* provide sufficient information for the two robots to establish a translation between their respective coordinate systems.

Following the coordinate reconciliation step, the *encounter* tasks move to the *sync team* step (Figure 4.16). During the sync step, each robot sends the knowledge it has

of its team to the other robot. This ensures both robots have an accurate picture of the composition of the two teams, which is a precursor for performing the team merge and redistribution operation. If the robots are on different teams, they will perform the team merge and redistribution operation as described in Section 3.7.

The final step in the encounter (Figure 4.16) involves the encountering robots sharing their current mission knowledge with one another. Each robot sends its current environment map to the other robot. Where the recipient has the capability to merge maps (i.e. the MaxBot and MidBot robot types), the map information is combined with the robot's own map of the environment. Where the encountering robots are on the same team, this helps ensure consistency in mission knowledge among teammates. Where the encountering robots are on different teams, the knowledge exchange helps synchronize the mission knowledge between the different teams.

After the completion of the steps of the *encounter* tasks, each robot removes the respective *encounter* task from its task list, and the time of the encounter is recorded to ensure the robots do not immediately engage in another encounter.

4.7.2 Knowledge Manager

My example implementation includes a knowledge manager module responsible for maintaining a robot's operational knowledge of its own attributes, and of the other robots operating in the environment. The knowledge manager is also responsible for maintaining the static knowledge pre-programmed into the robots at design time. This includes the robot's attributes, the task types, roles, and knowledge of the desired team definition.

The knowledge manager makes use of the robot's wireless communication facilities to synchronize knowledge of the team's composition among all members of the team. The knowledge manager accomplishes this using intentional broadcasts of self information, and by gaining knowledge from wireless traffic the robot overhears. Similar to Legras and Tessier [2004], I rely on the broadcast nature of wireless communications so that a robot makes use of all messages it overhears, even the ones not explicitly addressed to it. Further, the knowledge manager is able to "forget" knowledge of teammates when no communication has been received for some time; this enables the robot to cope with situations where a teammate has become lost or damaged (Section 3.2.1).

Each robot's knowledge manager periodically (every 15 seconds in my work) uses the wireless communication system to broadcast a message to all robots in range. This message contains the robot's identification, current team affiliation, current role and suitability to fill that role, and a list of the robot's attributes. Any robots within radio range overhear the broadcast and pass it to the knowledge manager. The knowledge manager updates its operational knowledge of other robots to include the new information. The knowledge manager timestamps the time when it has last heard from each robot. The timestamps provide the knowledge manager with the basis for "forgetting" old knowledge.

In addition to the intentional broadcasts a robot makes, all messages sent to other robots include the robot's identification, current team affiliation, current role, and suitability to fill that role. This small subset of information ensures team and role changes are more likely to be overheard by members of a robot's team. The full list

of robot attributes does not form a part of every message, as they would increase the size of messages considerably. Further, the robot attributes in my implementation are static, so there is not an imperative to continually resend them (though this would change if future work modeled the current state of various attributes and abilities).

Knowledge older than a configured age (in my implementation, I used three minutes) is "forgotten", and is no longer used by the robot when considering role and team switches (Section 3.6), considering potential task assignments (Section 3.5.3), for determining who is leading the current team, or when performing a team merge and redistribution operation (Section 3.7). The ability to forget knowledge in this manner ensures that a robot bases team merge and redistribution decisions on knowledge of the robots it is most confident are active members of the team (Section 3.7). Robots which become disabled are forgotten by their team, allowing teammates to adjust roles appropriately during periodic role-checks (Section 3.6.1). Similarly, if a robot becomes separated from its team, it will forget its old team and become more willing to join up with another team it encounters in the environment (Section 3.6.2).

4.7.3 Communication Manager

The *communication manager* module handles the delivery and receipt of messages between robots. It assumes that robots use a wireless radio capable of providing connectionless, broadcast-based communication between robots. Thus, robots are able to hear messages from all robots within their current radio range. Messages can be either broadcast and unacknowledged, or directed and acknowledged. The *communication manager* provides the capability to address sent messages to specific robots in radio

range, and to track the successful receipt of those messages using an acknowledgment message. In my work, I assume messages not impacted by communication failures are received by all robots in radio range of the sender. This includes messages which are addressed to a specific robot. Although only the intended recipient will act upon the message, other robots in radio range are able to use the sender's information to update their knowledge of the current team (Section 4.7.2). Further, robots capable of doing so will incorporate knowledge gained from confirming victims and exploring areas into their own operational knowledge (Section 4.2.3).

Wireless communication in my implementation is facilitated by the Stage simulator. To simulate unreliable communication, I assume any message a robot sends has a probability of being successfully sent to the robots in radio range. Section 4.9.1 describes the simulated wireless communication used in my implementation. The probability of being successfully sent is an experimental parameter, which is varied in the experiments I performed to evaluate my implementation (Section 5.5).

4.7.3.1 Acknowledged Messages and Timeouts

All messages a robot addresses to another robot contain a message sequence number. The message sequence number allows the *communication manager* to track whether or not a sent message was successfully received by the recipient. When a recipient receives a message addressed to it, it immediately sends back an acknowledgment message with the same sequence number. Upon receipt of the acknowledgment, the sender marks the message as being received successfully. It is, however, possible for either the sent message or the acknowledgment from the recipient to be lost due

to communication failures. The *communication manager* also tracks a send timeout for all sent messages (in my work, two seconds). If an acknowledgment has not been received within the send timeout, the sent message is marked as having failed delivery. As one of the goals of my work is to study the impact of unreliable communication on coordination between robots, my implementation does not include the capability to re-send messages when acknowledgments fail. In a real-world implementation, the *communication manager* would increase the chance of delivery through retries and other means. Rather than assuming unreliable communication can be made better, I attempt to use the available communication without relying on resends or other techniques.

The acknowledgment of sent messages is useful to allow a framework module to determine if a message was received successfully. For example, when one robot observes another in the environment, it sends an "I See You" message to the robot it observes, which is the trigger for initiating an *encounter* maneuver between the two robots. When the *communication manager* receives an acknowledgment to the "I See You" message, the *encounter* task knows that it can safely proceed to the next step in the encounter maneuver (Section 4.7.1.1).

Where appropriate, I attempt to deal with message delivery failures in other ways. A robot exploring an area, for example, could have moved out of range of its team coordinator, preventing it from reporting back the final results of its exploration. In such a scenario the robot would continue on to the next task, and report back the aggregate results from both exploration tasks. This helps to ensure effort is not lost due to communication failures.

4.8 Mission-Specific Modules

This section describes the mission-specific modules in my example implementation. These modules provide the necessary logic to enable robots to explore the environment, identify frontier exploration tasks, build a map, and identify victim confirmation tasks.

4.8.1 Mapping

One of the goals of my search and rescue system is to create a map of the environment a robot explores; an occupancy grid provides a means of describing the environment a robot explores in terms of the certainty that an obstacle exists at a location [Elfes, 1989]. In a real USAR environment a three dimensional mapping approach would be necessary. For my purposes, I use an occupancy grid structure capable of storing not only the certainty an obstacle is presence in a location, but also the height of the obstacle. My approach is similar to work by Gutmann et al. [2005] and results in a 2.5 dimension representation of the environment.

Since robots can begin operation anywhere in the environment, and the extent to which a robot can explore in any direction is unknown, it is important to represent the occupancy grid map in an efficient manner. Since each robot's coordinate system begins at point $(0, 0)$ in a Cartesian plane, a robot exploring a 100m x 100m environment would need to allocate a 200m x 200m map to account for the fact it could start operation at any point in the environment. I account for this by using a lightweight matrix of pointers, which point to dynamically allocated *occupancy grid patches*. Figure 4.17 illustrates this process. Initially the matrix of pointers does not reference any

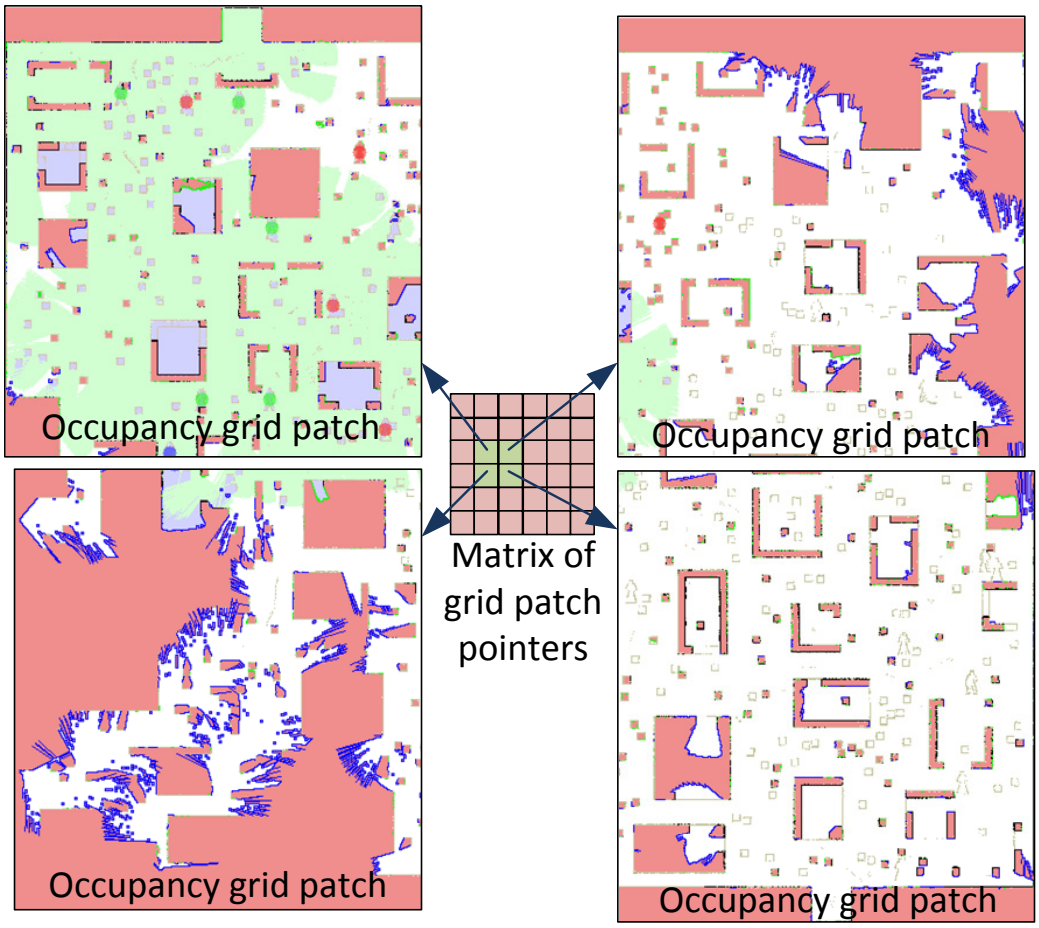


Figure 4.17: To support efficient storage of the map, a lightweight matrix of pointers references the occupancy grid patches, which are allocated as needed.

patches. As the robot explores and a need arises to expand the map, new fixed-size occupancy grid patches are allocated in the pointer matrix. In my implementation, the pointer matrix can accommodate up to 100 x 100 grid patches, each 500 x 500 cells in size. In my experiments, I used a map resolution of 0.08 meters x 0.08 meters per cell. This means the occupancy grid map can represent an environment up to 2000 meters in each direction from the robot's starting location. It would be possible, however, to modify my occupancy grid implementation to facilitate the expansion of

the pointer grid, allowing the map to be expanded as required.

4.8.1.1 Updating the Occupancy Grid Map

Each cell in the occupancy grid map consists of two single-byte integers, representing the height and degree of certainty an obstacle is present at that height. As previously introduced in Section 2.3.2.1, I use the HIMM (Horizontal in Motion Mapping) approach to update the occupancy grid map with scans from sonar or laser rangefinders. Bresenham [1996]'s line algorithm is used to rasterize the scan lines onto the occupancy grid.

Algorithm 8 Applying a rangefinder scan to the occupancy grid map.

Require: $P = (x, y, z)$, the coordinates of the origin of the scan

Require: D, the distance of the rangefinder scan

Require: R, the maximum range of the scanner

Require: M, the occupancy grid map to update

$E \leftarrow$ end point of scan based on P and D

$U \leftarrow P$

while $U \neq E$ **do**

if at end of scan ($U = E$) and scan distance less than max range ($D < R$) **then**

Increase confidence at P by 3

else

Decrease confidence at P by 1

end if

$U \leftarrow$ next point on line, according to Bresenham's algorithm

end while

Given the starting point of the rangefinder scan (P) and the distance of the rangefinder scan (D), Algorithm 8 describes the process used to update the certainty values in the occupancy grid map.

Individual cells in the occupancy grid are updated using Algorithm 9. The algorithm accepts a point (P) on the map, specifying the (x, y) coordinates of the point to update, and the height (z) the corresponding reading was taken at. Where a rangefinder scan is applied to the map, z will be the height of the rangefinder sensor which generated the scan. The algorithm also requires the change in certainty to be applied (Δ). A positive Δ value indicates an increase in confidence an obstacle is present, while a negative Δ value indicates a decrease in confidence an obstacle is present.

As described in Algorithm 9, if Δ represents a decrease in certainty, the certainty at cell C_p will only be decreased if the height of the rangerfinder is the same or lower than the height (H_o) stored in the occupancy grid for cell C_p. This allows the occupancy grid map to increase the certainty of a low-lying obstacle, while ignoring a decrease in confidence at a higher point. If Δ represents an increase in certainty, the certainty at cell C_p is increased; if the rangefinder height (z) is higher than the height (H_o) stored in the occupancy grid for cell C_p, the height at C_p is set to z.

Figure 4.18 illustrates the operation of Algorithm 9 for a robot with both a laser rangefinder positioned high on its body, and a sonar rangefinder positioned low on its body. The dashed blue box represents a cell at coordinates (x, y) in the occupancy grid map (M), which stores the certainty C_p an obstacle of height H_o is present at that location. For this example, assume $C_p = unknown$ initially. At point (x, y), the

Algorithm 9 Updating the certainty an obstacle is present at a location.

Require: $P = (x, y)$, the point in the map to update the confidence

Require: z, the height of the rangefinder used to take the reading

Require: Δ, the change in confidence

Require: M, the occupancy grid map to update

$C_p \leftarrow$ certainty of cell in M at (x, y)

$H_o \leftarrow$ height of obstacle as recorded in M at P

if decreasing certainty ($\Delta < 0$) and z is the same height or lower than H_o **then**

$C_p = C_p + \Delta$

else if increasing certainty ($\Delta > 0$) **then**

$C_p = C_p + \Delta$

if z is higher than H_o **then**

$H_o = z$

end if

end if

sonar rangefinder has detected debris, and updates M with an increase in confidence of $\Delta_s = +3$, at height z_s. This sets $C_p = +3$, and $H_o = z_s$. Also at point (x, y), the laser rangerfinder detects empty space, and updates M with a decrease in certainty of $\Delta_s = -1$, at height z_L. Since Δ_s represents a decrease in certainty, and $z_L > H_o$, C_p is not changed, and remains +3.

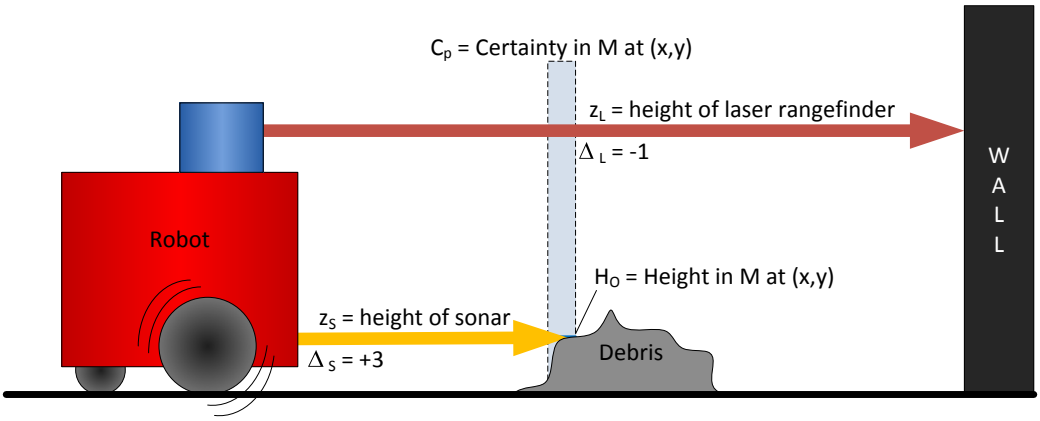


Figure 4.18: Example of updating the occupancy grid using sensors readings from two heights.

4.8.1.2 Merging Maps

As robots carry out *explore frontier* tasks (Section 4.5.2.2), they send the results of their exploration to the team coordinator, in the form of an occupancy grid map. As previously described in Section 2.3.4, combining maps generated by multiple robots is a non-trivial operation. Errors in localization and noisy sensor data, for example, lead to errors in the map merging process, potentially rendering the resulting map useless.

In my example implementation, since I assume members of a team share the same coordinate system (Section 4.3.1), robots on the same team can communicate maps with one another, and overlay the maps in order to facilitate merging. As robots from different teams encounter one another in the environment, they gain the ability to translate their coordinate systems, opening the possibility of merging maps between teams. The merging of maps between teams occurs not only with the information

exchanged during the encounter (Section 4.7.1.1), but also when map communications between members of another team are overheard (Section 4.2.3).

Since robots use the Stage API to localize themselves, and all agents effectively share a common coordinate system, I assume that a robot can only merge another robot's map into its own if it has previously encountered a member of that robot's team, establishing a translation between their respective coordinate systems (Section 4.3.1). This means merging maps is a simple manner of overlaying the two occupancy grid maps, once coordinates are appropriately translated. Although this is unrealistic for a physical environment without adding a means of overcoming the error involved in reconciling coordinate differences between agents, it is sufficient for demonstrating my adaptive team management framework.

Algorithm 10 Merging maps.

Require: M, the occupancy grid map.

Require: S, an occupancy grid map to merge into M.

for all $P[x, y] \in S$ **do**

  if $P[x, y]$ is not unknown **then**

    if $P[x, y]$ is known empty space **then**

      $C[x, y]$ = known empty.

    else

      Update certainty of $C[x, y]$ using Algorithm 9, with $S[x, y].Height$, $S[x, y].Certainty$.

    end if

  end if

end for

Given a robot's map (M), and a received map to merge in (S), Algorithm 10 describes the process used to merge maps. For each cell $P[x, y]$ in S which does not represent unknown space, the corresponding cell $C[x, y]$ in M is found. If $P[x, y]$ represents known empty space, $C[x, y]$ is set to known empty space. Likewise, if $P[x, y]$ represents an obstacle of varying certainty, Algorithm 9 is used to update the certainty of $C[x, y]$, with the height from $S[x, y]$, and the change in certainty being the certainty of $S[x, y]$.

4.8.2 Frontier Finder

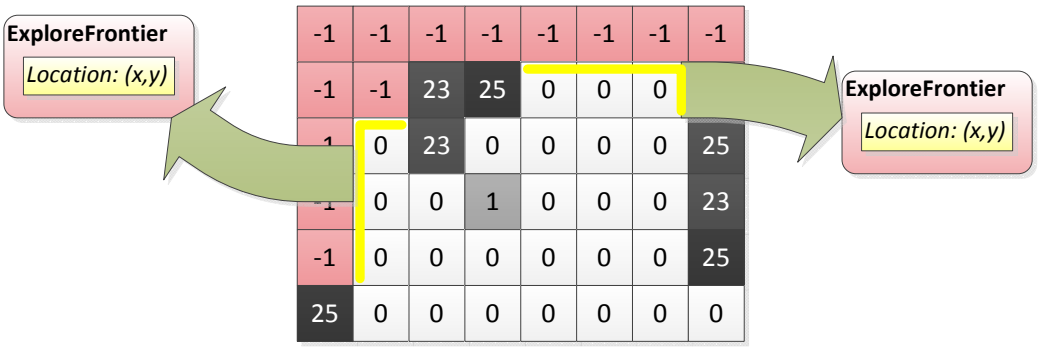


Figure 4.19: The frontier finder identifies frontiers exploration tasks.

As described in Section 2.3.5, my implementation uses a frontier-based approach to guide a team's exploration efforts. A frontier is a transition between explored empty space and unexplored empty space, and represents a location where a robot can move to potentially gain knowledge of the environment Yamauchi [1997]. As illustrated in Figure 4.19, the robot filling the *team coordinator* role uses the frontier finder module to identify frontiers in the robot's occupancy grid map and generate *explore frontier* tasks. As robots report back the result of exploring frontiers to the

team coordinator, the frontier-finder marks the frontiers as explored.

The frontier finding algorithm runs periodically (every 10 seconds in my implementation) to inspect the robot's occupancy grid map to identify new frontiers. The frontier finder also removes frontiers which no longer exist due to updates made to the map since the last execution of the algorithm.

Algorithm 11 Updating frontiers.

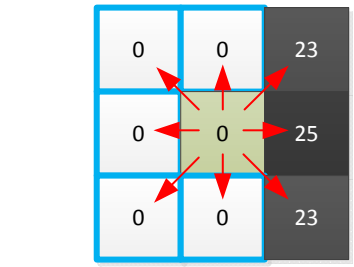
Require: M, the occupancy grid map, with all cells marked unvisited.

```
for all C[x, y] ∈ M do
  if C[x, y] is not visited, is not part of a frontier, and is a frontier cell then
    Mark C[x, y] visited.
    Create new frontier F starting at C[x, y].
    V = empty stack
    Push C[x, y] onto V
    while V has cells do
      T[x, y] ← Pop top cell from V
      Add cell T[x, y] to frontier
      Push neighboring frontier cells of T[x, y] not visited, or part of a frontier
    end while
    Record size of F and create frontier task
  else if C[x, y] is not visited, is part of a frontier, and is not a frontier cell then
    Update frontier containing C[x, y]
  end if
end for
```

Algorithm 11 describes how new frontiers are identified and unnecessary frontiers are removed. The algorithm begins by assuming all cells in the map are unvisited / unprocessed. For each cell ($C[x, y]$) in the occupancy grid map M, the algorithm first determines if $C[x, y]$ has been marked as visited (each cell is visited only once during each algorithm execution). If the cell has not been visited, is not already part of a frontier, and is a *frontier cell*, it is pushed onto a stack (V) for processing. A new frontier (F) is created, and the algorithm follows the frontier by adding all adjacent frontier cells to the frontier.

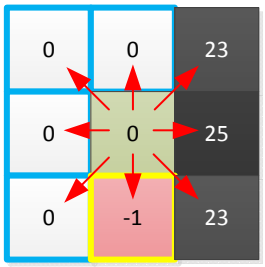
If the algorithm encounters a cell $C[x, y]$ which has not been visited, is marked as part of a frontier, but is not a frontier cell, it updates the frontier containing $C[x, y]$ to decrease its size. This ensures as the map changes due to the addition of new information, existing frontiers are adjusted appropriately. The new knowledge could result in an existing frontier shrinking in size, or being removed entirely. If a frontier is removed, the robot broadcasts a message indicating the task associated with the removed frontier is now invalid. The robot whom the task was assigned to will remove the task from its task list to prevent duplication of effort. Communication failure could result in the task cancellation message failing to be received by the task assignee, in which case the assignee would still explore the frontier, duplicating effort. The duplication of exploration effort still provides the opportunity for new knowledge to be gained, as each robot will likely take a different route to arrive at the frontier. Further, re-exploring an area helps to increase the certainty of the map data for that area.

A cell $C[x, y]$ is considered a frontier cell if it is a transition from known empty



Cell is empty.
5 neighboring cells are empty.
No neighboring cells are unknown.
Cell is **not** a frontier cell.

(a)



Cell is empty.
4 neighboring cells are empty.
1 neighboring cell is unknown.
Cell **IS** frontier cell.

(b)

Figure 4.20: A cell is a frontier cell if it and one of it's neighbors are empty, and a neighbor is unknown.

space to unknown space. By definition, a cell is a *frontier cell*, if it represents empty space, one of its eight neighboring cells is known empty space, and one of its eight neighboring cells is unknown space. Figure 4.20a shows a cell that is not a frontier cell: The cell itself represents empty space, five of its neighboring cells are empty space, but no neighboring cells represent unknown space. There is no potential to gain knowledge of unknown space at that cell. Figure 4.20b shows a cell that is a frontier cell: The cell itself represents empty space, four of its neighboring cells are empty space, and one of its neighboring cells is unknown space.

When a robot completes an *explore frontier* task, it reports back the map information resulting from the exploration to the team coordinator. The team coordinator merges the results into its own map, and marks the frontier as explored.

My approach to frontier-based exploration assumes the team coordinator will identify a number of frontiers for exploration, and assign them to the robots on its team,

such that there will generally be a backlog of frontiers requiring exploration. This approach differs from work by Yamauchi [1998], where coordination is achieved by sharing map information and each robot is responsible for identifying frontiers on its own. Yamauchi [1998] makes no attempt to eliminate duplicate effort between robots. Other approaches can be used to help prevent duplicate effort, and it would be useful future work to investigate the efficacy of such approaches in helping to prevent the duplication of effort in my implementation. Poernomo and Ying [2006], for example, attempt to ensure robots are sufficiently spread out, so as to reduce the possibility of overlapping efforts. My frontier finder implementation could be modified to mimic this behaviour by attempting to assign the most geographically separated frontiers to different robots.

4.8.3 Victim Tracker

The *victim tracker* module is responsible for tracking victims in the environment, and identifying potential victims in the environment which require confirmation. It maintains a list of tracked victims, where victims are uniquely identified by their location in the environment. Information recorded about each tracked victim includes a location, status, and information about the associated confirmation task (if applicable) for that victim. The victim status can be: *potential*, *negative*, or *positive*. A status of potential means the victim tracker is not sure if there is a victim present at the victim's location. This is the case where the victim was detected using a *basic victim detector* (Section 4.9.2). A status of negative indicates a potential victim which has been confirmed (using a *full-featured victim detector* (Section 4.9.2)) as

not being a victim. A status of positive indicates a potential victim which has been positively confirmed as a true victim.

The *victim tracker* updates the status of victims identified by the *detect victims* perceptual schema (Section 4.6.1.7), and using information received via the wireless communication system. The *detect victims* perceptual schema provides a list of victims, and indicates whether each is categorized as potential, negative, or positive, as determined by the robot's victim detector sensor. The *victim tracker* loops through the reported victim locations and status and updates its list of tracked victims. If a new potential victim has been found, the *victim tracker* creates a new *confirm victim* task, which it adds to its task list (Section 3.5.2).

The team coordinator inspects *confirm victim* tasks it receives using the *victim tracker*, and removes tasks for victims which it already knows to be confirmed, or for which a different victim confirmation task already exists. This helps reduce the chance of duplicate effort where multiple robots identify a potential victim in the same location, or where a different robot finds a potential victim which has already been confirmed.

When a robot completes a *confirm victim* task, it reports its findings about the potential victim to the team coordinator. The team coordinator's *victim tracker* module uses this information to update its knowledge of the victim's status. All other robots with the capability to do so who overhear this message also update their knowledge of the victim. This helps ensure knowledge of victims is distributed among the team. In my implementation, I assume the MinBot robot type is capable only of tracking the victims it identifies, and the MidBot and MaxBot robot types are

both able to track all victims the team identifies. A real-world implementation of a MinBot would likely be severely memory-constrained, making it unlikely to be able to track the victims found by the entire team.

4.8.4 Planner

The planner module provides a robot with the ability to make use of its occupancy grid map to find a path from a starting location to a destination. My planner module implementation is based on the wavefront planner module included in the Player component of the Player/Stage software [Gerkey et al., 2003]. According to Murphy [2000], one advantage of wavefront planners is their ability to use a generated plan to find a path from any location in the environment to the destination. This is ideal in my work, as the planner module is used during the task allocation process. Given a task location, the planner module is able to efficiently determine an appropriate path and associated cost to reach the task location from any task assignee's location.

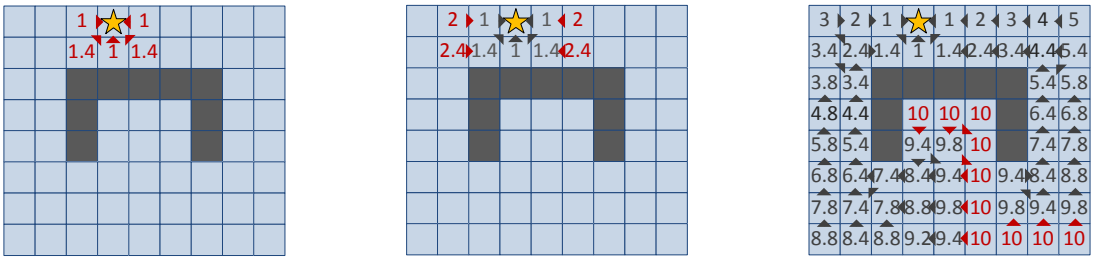
When assigning a task, the task assignment process first generates a plan for the task location. As robots respond to task allocation requests, they report their current location to the task assigner. The task assigner uses the planner to determine the cost (distance) involved with reaching the task location from each potential assignee's location. When an assignee is chosen to carry out the task, the planner is used to provide a series of way-points to help guide the assignee to the task location. The way-points do not directly dictate the exact movements the robot will make, but instead act to guide the robot to its destination while its reactive control system performs obstacle avoidance. This helps reduce the chance of an assignee getting lost

or stuck in a local minima while navigating to a task location.

The wavefront planner operates on a grid data structure (similar to the occupancy grid map) called a *configuration space* [Murphy, 2000]. The configuration space, or CSpace for short, represents the environment from the perspective of the planner. Similar to the occupancy grid map, it specifies the known areas where a robot can move to, and the areas known to be occupied by an obstacle. Unlike the occupancy grid, it does not specify a level of certainty for obstacles, nor does it distinguish between unknown space and obstacles. This makes sense, as the planner cannot assume the robot is able to travel to unknown spaces, or to destinations within an area flagged as an obstacle. The configuration space also takes into account how close to obstacles a robot is permitted to travel. In my implementation, I assume robots should not get closer than 30 cm to obstacles, to reduce the possibility of collision.

The planner begins at the destination cell, and marks each non-obstacle neighboring cell with the distance (path cost) to the destination cell. A pointer in each neighbor cell points back to the destination. The neighboring cells are then recursively populated. The result is a fully-populated CSpace grid where each cell indicates the distance to reach the destination cell, and the pointer in the cell points to the neighboring cell with the smallest distance.

Figure 4.21 illustrates the process through which the wavefront planner CSpace is populated, assuming each cell represents a 1m x 1m area. Figure 4.21(1) shows the first iteration of the algorithm. Here, each neighbor of the destination cell is populated with the distance to the destination. The pointers in each cell point to the destination. Figure 4.21(2) shows the second iteration. The distances expand



(1) Beginning at destination, populate neighbors with cost to destination.

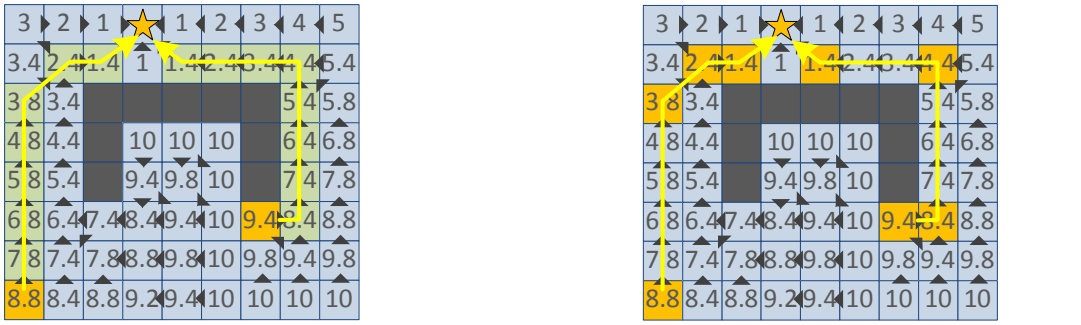
(2) Recursively populate costs of neighboring cells.

(3) Continue until all cells are populated.

Figure 4.21: Wavefront expansion.

outward from the destination, much like a wave in a pool of water. Here, the pointers in each cell point to the next closet cell to the destination. Figure 4.21(3) shows the result of the algorithm. Each cell in the CSpace is populated with a distance to the destination, and the pointers in each cell point to the next closest cell to the destination.

The cost (distance) to reach the destination is stored in each cell. As illustrated in Figure 4.22(1), finding the path from any location to the destination becomes a



(1) To find path, begin at start location and follow pointers to lowest cost neighbors.

(2) Way-points are the minimal set of points with no obstacles between them.

Figure 4.22: Wavefront path generation.

simple manner of following the pointers from the starting location to the destination. The figure illustrates this process for finding the path from two different locations to the destination. To provide a set of way-points to guide a robot from a location to the destination, the planner examines the path to find the minimal set of cells along the plan with no obstacles between them (illustrated in Figure 4.22(2)). This ensures the way-points are providing general guidance to the robot, rather than a prescriptive set of exact moves the robot must carry out to reach the destination.

4.9 Simulation Implementation Details

This section describes the manner in which I extended the Stage simulator to provide simulated limited range unreliable communication. It also describes the manner in which I implemented the *victim identifier* sensor and *robot detector* sensors, using the facilities provided by the Stage simulator.

4.9.1 Simulated Unreliable Wireless Communication

I assume that the robots in my work coordinate via unreliable short-range wireless communication. A real-world implementation might use an 802.15.4 wireless radio to support communication between robots. Radios of this nature can achieve a maximum transmission range of approximately 300 meters in open space [Baronti et al., 2007].

Older versions of the Stage simulator supported a simulated Wi-Fi model, which was capable of determining which robots are in radio range of one another. The Wi-Fi model also supported a basic broadcast message facility. As part of my work, I ported the older code from the Stage 2.0 code-base to Stage 3.2 version I used for

my work. The Stage Wi-Fi model supports simulation of radio propagation using the ITU indoor radio model. According to Seybold [2005], the ITU indoor radio model describes the propagation of radio signals inside rooms or closed areas. A distance power loss coefficient determines how much signal power is lost as radio waves propagate. Higher values indicate a higher degree of signal loss as the distance between the sender and recipient increases.

Algorithm 12 Determining if a robot is in radio range.

Require: $C \leftarrow$ power loss coefficient

Require: $P \leftarrow$ transmission power, in dB

Require: $S \leftarrow$ receiver sensitivity, in dB

Require: $f \leftarrow$ transmission frequency in MHz (2400 for Wi-Fi)

Require: $d \leftarrow$ Cartesian distance to robot

$L = 20 \log_{10}(f) + C \log_{10}(d) - 28$

if $L \leq (P - S)$ **then**

  return In range

else

  return Out of range

end if

Algorithm 12 describes how the modified Stage simulator determines if two robots are within radio range of one another. The algorithm calculates L, the power loss (in dB) of a signal transmitted at frequency f (in MHz), where the robots are separated by a cartesian distance of d meters. The power loss coefficient (C) determines the degree of power loss as distance increases – higher numbers indicate a higher degree

of power loss as distance increases. If the power loss between robots is less than the difference between the power of the transmitter (P), and the sensitivity of the receiver (S), the robots are considered to be in radio range. In reality, radio signals would not cut off at an exact distance and would vary based on the presence of obstacles and local interference. However, for the purpose of my work this radio model is sufficient.

In my implementation, I use a power setting of 4, a sensitivity of -70, and a power loss coefficient of 30. The power loss coefficient was chosen based on the ITU recommendations for an indoor office environment [ITU-R, 2003]. The power and sensitivity were arrived at through initial experimentation, and result in an effective communication range between robots of approximately 20 meters. Although wireless radios can achieve much greater transmission distances in open space, I assume that the difficult environment of a disaster zone would introduce considerable limitations on the maximum transmission range of radio signals. Further, it would be advantageous to minimize the radio transmission power in order to conserve robot battery life.

Given my work assumes wireless communication is unreliable, I extended the Wi-Fi model to support a parameter that determines the probability of a message being sent from a sender successfully. Algorithm 13 shows the algorithm I use to implement simulated unreliable message delivery. The parameter P_{sucess} determines the probability a sent message will be successfully delivered to the robots within radio range. The algorithm generates a random number R from 0 to 100. If the random number of greater than the $P_{success}$, the message is delivered to all robots in radio range. The sending robot does not know whether a sent message was successfully

Algorithm 13 Simulated unreliable message delivery.

Require: $N = \{N_1, N_2, \cdots, N_n\}$, robots in radio range.

Require: $M \leftarrow$ message to send.

Require: $P_{success} \leftarrow$ the probability of successful message send ($[0 \cdots 100]$).

$R \leftarrow$ random number in $[0 \cdots 100]$.

if $R > P_{success}$ **then**

for all $N_i \in \{N_1, N_2, \cdots, N_n\}$ **do**

Deliver M to N_i.

end for

end if

received.

4.9.2 Victim Detectors

I use the *fiducial finder* functionality provided by the Stage simulator to implement false and positive victims, and the two types of victim sensors (Section 4.3.2). Objects in the Stage simulator can be assigned a *fiducial type* and *fiducial ID* within that type. In my work, I assign a specific fiducial type for the identification of false and positive victims (another fiducial type is used to identify robots with the *robot identifier* sensor (Section 4.9.3)). Configurations of debris representing a victim (e.g. Figure 4.7a) are assigned a fiducial ID of 1, while true victims (Figure 4.7b) are assigned a fiducial ID of 2.

A fiducial finder in Stage is a sensor capable of detecting objects marked with a specific fiducial type, and subsequently returning the associated fiducial ID. This

is the rough equivalent of having a bar code to uniquely or categorically identify a marked object, and a scanner to read that code. The fiducial finder has a configurable *maximum range* and *identification range*. The *maximum range* identifies the maximum distance at which a fiducial finder can detect fiducials. Within the maximum range, a fiducial finder returns the coordinates and orientation of each object with the matching fiducial type. The identification range determines how close the robot must be to a fiducial in order to determine its fiducial ID. Outside the identification range, the fiducial ID is returned as "undefined", indicating the potential presence of a victim.

The *full-featured victim detector* is configured as a fiducial finder with a maximum range of 6.0 meters, and an identification range of 4.0 meters. This means at a distance between 4.0 and 6.0 meters the *full-featured victim detector* can identify the potential presence of victim, but because it cannot read its fiducial ID, it cannot distinguish between a true victim and a debris configuration resembling one. Between 0 and 4.0 meters, the *full-featured victim detector* can read the fiducial ID, making it capable of distinguishing between true victims, and debris configurations resembling victims.

The *basic victim detector* is configured as a fiducial finder with a maximum range of 4.0 meters, and an identification range of 0 meters. This means that at a distance between 0 and 4.0 meters, the *basic victim detector* can identify the potential presence of a victim. The identification range of 0, however, prevents the *basic victim detector* from differentiating between true victims and debris configurations representing victims. Both victim detector sensors have a 180 degree field of view in front of the robot.

4.9.3 Robot Identifier Sensors

In my work, the *robot identifier* sensor provides a robot with the ability to determine, in the robot's local coordinate space, the position and orientation of another robot in close physical proximity. It provides a means for a robot to initiate an encounter with a robot it observes in the environment (Section 4.7.1.1).

In their work, Howard et al. [2006a] outfit each physical robot with a cylindrical fiducial. The marker is encoded with the robot's unique ID and has a pattern around its circumference which is marked so that it appears differently, depending on the angle from which it is viewed. Using a camera, a robot can observe another robot in the environment to determine the identification and orientation of it in the environment, in relation to itself. Analogously, I employ simulated fiducials for a robot to identify and determine the relative location and orientation of another robot. I assume that all robots in my environment have fiducial markers similar to [Howard et al., 2006a]. Robots equipped with a *robot identification sensor* (i.e. the MidBot and MaxBot robot types) read the fiducials on robots to determine their position and orientation in the environment.

Similar to the victim detection approach described in Section 4.9.2, I make use of the fiducial finder functionality in the Stage simulator to implement my fiducial-based robot identification system. A unique fiducial type is delegated to robot identification. Each robot has a fiducial with the robot identification fiducial type, and a fiducial ID set to its unique robot ID. A Stage fiducial finder configured to recognize the "robot identification" fiducial type forms the *robot identifier* sensor.

The *robot identifier* fiducial finder is configured to have a maximum range of 6m in

which it can identify the position and orientation of other robots. The *robot identifier* is set to have a 180 degree field of view in front of the robot.

For robots that have a *robot identifier*, the Stage simulator reports the robot ID and relative position and orientation of any robots in the current field of view. These values are used by the *detect robots* perceptual schema (Section 4.6.1.6).

4.10 Conclusion

This chapter has described the Urban Search and Rescue implementation I developed in order to support an evaluation of my methodology. The following chapter describes the experiments I performed to evaluate my methodology against baseline conditions.

Chapter 5

Evaluation

5.1 Overview

In order to examine the effectiveness of my methodology (Chapter 3), experiments were conducted in the simulated USAR environment, described in Section4.3. I compared the performance of teams using my methodology against two baseline cases. In the first, robots were not able to implement role changes and team membership changes in response to changing conditions. I further compared these against a worst-case baseline, where in addition to roles and team membership being fixed, tasks were mapped in a fixed manner to the three robot types. I considered factors such as the availability of replacement robots, the degree to which communication would be successful, and the likelihood of robot failure.

This chapter begins with a review of my thesis questions (Section 5.2), and a description of the experiments I performed to answer these questions. Next, I describe how I generated the experimental environments (Section 5.4) in which I ran

my experiments. I then describe the design for the experiments (Section 5.5), along with the criteria I used to compare the effectiveness of my methodology against the baseline cases (Section 5.3). The results of my experiments are presented in Sections 5.7 and 5.8, and a discussion of my findings is found in Section 5.9.

5.2 Review of Research Questions

Recall my research questions from Section 1.6:

1. **Can my framework provide teams operating in dynamic environments with the ability to adequately cope with changes in team structure and composition (i.e. due to loss and failure of team members, and encountering other teams and teammates in the environment)?**

2. **Can my framework help mitigate the negative affect of unreliable communication on coordination efforts between agents?**

3. **Is my framework able to cope with failure of a team's leadership structure?**

My main experiment (Section 5.5) answers the first two questions, by studying the impact communication and robot failures have on the performance of teams. A second experiment with a smaller scope (Section 5.8) answers the third question, by introducing a failure in the leadership structure of a team at a pre-determined interval, and observing the performance of the team as it adapts.

5.3 Evaluation Criteria

To facilitate an evaluation of the performance of my methodology, I recorded two values at fixed times throughout each trial: the percentage of the environment covered, and the percentage of victims successfully identified. The *percentage of the environment covered* metric measures the total portion of the environment explored by both teams. The *percentage of victims successfully identified* metric describes the number of victims (Section 4.3.2) successfully identified by all teams. A victim is considered successfully identified when it has been correctly identified as either a false or positive victim (i.e. by a robot equipped with *full-featured victim detector* (Section 4.9.2). My implementation software captures these metrics every 20 seconds for the duration of each 30 minute trial.

The metrics are captured from the team coordinators of each team, excluding replacement robots that have not yet joined a team or formed a team of their own. I also exclude the individual map and victim knowledge from MinBots, as these robots are not capable of forming an aggregated view of a team's operational knowledge (Section 4.2.3). This can result in some knowledge gained by MinBots not being represented in the above metrics (i.e. if a MinBot was unable to communicate new knowledge to the team coordinator, due to it becoming lost or experiencing communication issues). I chose not to include this information, as I assume the successful transfer of knowledge to the team coordinator is an important indicator that a team is performing effectively, and because it takes a generally conservative approach to evaluating my work. Map data from the team coordinators is merged (Section 10) and victim knowledge is combined to determine the actual metrics above for the purposes

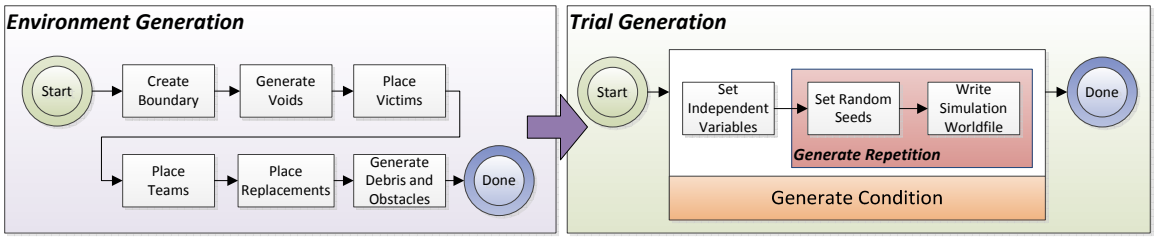


Figure 5.1: Experimental environments are generated using a two step process.

of evaluation.

5.4 Experimental Environment

This section describes the simulated disaster environments in which I ran my experiments. The manner in which the environments were constructed is described, as well as the experimental controls I used to help avoid bias between the environments.

All experimental environments were generated as world files suitable for use with the Stage simulator. The three robot models (MinBot, MidBot, and MaxBot) are defined in a common file, according to the specifications found in Section 4.4. This common file is referenced in each world file, ensuring all instances of a robot type have the same physical properties, sensory equipment, and capabilities.

5.4.1 Generating Environments

As previously stated in Section 4.3, I chose to evaluate my methodology in simulation for the consistency and repeatability it offers over performing experiments in a real-world domain. Running all of my experiments in the same environment is not ideal, as there could be biases inherent with the environment. For example, the

starting position of replacement robots and the configuration of debris could result in the replacements infrequently encountering other robots, making my methodology appear less effective than it is. Similarly, the starting position of replacement robots could result in the replacements encountering other robots easily, making my methodology appear more effective than the intended operating environment. Using multiple environments with the same difficulty of operation is ideal, as it helps to reduce bias inherent to the environment. The environment generator I used to generate my environments ensures all environments are similar in terms of the difficulty operation. The environment generator ensures the placement of environment elements occurs randomly, robot starting locations are evenly spaced along the perimeter of the environment, and the density of obstacles present in the environments are consistent.

Environments used in my work were generated using an environment generator application written in Java. The environment generator randomly creates an environment containing debris, obstacles, voids, victims, and robots. Parameters in the environment generator allow the properties of the generated environments to be configured. I used the environment generation application created by Wegner [2003] as starting point for my environment generator.

As shown in Figure 5.1, generation of environments occurs in two phases: *environment generation*, and *trial generation*. During the *environment generation* phase, the physical configuration of the environment is established. This includes the position of walls, voids, debris, obstacles, robots, and victims within the environment. The configuration of each generated environment is saved. The chosen environments (Section 5.4.2) are used as input for the *trial generation* phase, where the actual experimental

conditions are generated for each experiment.

I chose to use a two phase environment generation process to ensure I had an opportunity to review the environment configurations, prior to generating the complete Stage world files for each environment. The two phase environment generation process also provides the flexibility to generate more repetitions of an experiment, or entirely new experiments based on the same environment configuration.

Each generated environment is 60 meters by 60 meters in size, and has a boundary wall along each edge (e.g. Figure 4.3, p.129). After placing the perimeter boundary walls, the voids and rooms are placed in the environment (referred to as voids from this point forward). An environment has 50 rectangular voids, with randomly generated dimensions 5 – 12 meters wide, and 5 – 12 meters long. The voids are randomly positioned within the environment, ensuring none overlap.

Each void has 1 to 5 openings, randomly placed along the perimeter of the void. Each opening is from 1 to 2.5 meters wide. 60 percent of voids have all of their openings blocked by debris (Figure 4.5b, p.131) , meaning they are only accessible to robots with a tracked drive system. The openings in the remaining 40 percent of voids are not blocked by debris, and are accessible to any robot (Figure 4.5a, p.131).

As described in Section 4.3, robots begin operation in enclosed areas along the perimeter of the environment (Figure 4.4, p.129). This emulates an entry point in the environment where the robots are inserted, such as a window or a doorway. In my experiments, I used two teams, each with its own entry point. The first entry point is randomly placed along the perimeter of the environment. The other entry point is placed to ensure both entry points are evenly spaced along the perimeter of

the environment. Each team starts with one MaxBot robot, two MidBot robots, and four MinBot robots, arranged as illustrated in Figure 4.4.

After placing teams, starting positions are generated for the replacement robots along the perimeter of the environment (Figure 5.1). These starting positions are used for experimental conditions where replacement robots are present. 13 replacement robot starting positions are evenly spaced along the perimeter of the environment. The 13 positions are randomly assigned a robot from a pool of replacements, which includes 10 MinBots, 2 MidBots, and 1 MaxBot.

Each environment has 10 negative victims (debris configurations resembling victims; Figure 4.7a, p.136), and 20 positive victims (Figure 4.7b, p.136). Negative and positive victims are positioned randomly in the environment, such that they are at least 0.8 meters from a wall, and no victims overlap.

The final step in the environment generation process (Figure 5.1) is the placement of obstacles and debris (Figure 4.4, p.129). Each obstacle or debris object is 0.5 meters wide by 0.5 meters long. Obstacles and debris are created and positioned randomly, until the total area occupied by walls, obstacles, and debris covers 13% of the total environment area. 60% of the added objects are debris (impassable by wheeled robots), and 40% are obstacles (impassable by all robots). The obstacle and debris objects are positioned to ensure they are at least 1.2 meters from the wall of a void, and 0.6 meters from the nearest obstacle or debris object.

5.4.2 Choosing Environments

I generated a total of 40 environments, from which 3 environments were chosen to be used when performing my experiments. My selection process involved loading each environment in the Stage simulator, and running them for a few minutes of simulated time to observe the general movement of the robots. I eliminated any environment where:

1. The wall of a void blocks a team starting position. This configuration creates a narrow corridor through which the entire team must navigate before entering the broader environment. The high density of robots in the starting position, and the fact the robots use reactive navigation would result in significant time wastage before any useful work could be done. In a real USAR environment, this would be a situation where after a single false start, a team would be inserted elsewhere.

2. The wall of a void blocks a replacement robot starting position. Similar to the previous case, a narrow corridor is created which causes the replacement robot to waste considerable time attempting to free itself before it can even begin doing useful work.

3. The randomly distributed obstacles and debris are too clustered. In some cases, the random placement of obstacles and debris resulted in areas where the debris and obstacle configuration made navigation more inconsistent across the domain than would be desirable. The goal was to have a series of environments with roughly consistent navigational difficulties, as opposed to domains where

difficulties were more concentrated in some areas than others.

4. The randomly distributed victims are too clustered. Occasionally victims would be clustered, rather than evenly distributed throughout the environment. Again, the goal was to have domains that would be of roughly equal challenge, and this clustering would prevent that.

After considering the above situations, I chose three environments from the initial 40 that appeared to be of roughly equal difficulty. The independent variables, and the factorial design used in my main experiment (Section 5.5) made it impractical to use more than three environments (each environment adds 2700 simulation runs). Many repetitions of each experimental condition are run in each of the three environments, resulting in a comprehensive demonstration of my methodology compared to the baseline techniques. The three environment configurations I chose are shown in Appendix A.

5.4.3 Generating Repetitions

In order for me to make observations beyond those possible with aggregate data, I required the ability to visually observe the trials that make up this data as well. Watching trials as they ran was not desirable, as running Stage with the graphical user interface enabled results in a significant performance penalty (I ran my experiments with the graphical user interface disabled to speed simulation execution). Further, even with the graphical user interface enabled, simulations run up to 20 times faster than real-time, making it difficult to make fine grained observations. This means making observations about specific trials required the ability to re-run them perfectly,

with the graphical user interface enabled and the simulation speed slowed down.

The factorial design of my experiment (Section 5.5) assumes each experimental condition (all 162 combinations of the levels of the variables shown in Figure 5.2) will be run for 50 repetitions each. In each repetition, different communication failures and robot failures will occur. Further, the autonomous control software will generate different random motions, and use different recovery movements to become unstuck. Finally, the role check operation intervals occur for each robot at different intervals. These variations mimic what would be expected in repeated repetitions of an experiment in the real world.

Although operation of the Stage simulator ensures running the same world file multiple times will result in repeatable results [Vaughan, 2008], the use of random number generators in my robot control software means individual trials are not naturally repeatable. To ensure repeatability of the trials and facilitate re-running them for observation, the seed values for the random number generators are specified in the Stage world file for each trial. This ensures any single experimental trial can be run any number of times, generating the same results each time.

50 sets of different random number seeds were generated, and used in the 50 repetitions of each experimental condition. This ensures random factors are identical across all like numbered repetitions of each experimental condition. This means, for example, if robot 3 fails after 10 seconds, in repetition 1, it will fail at the same point in repetition 1 for all experimental conditions. The use of different random number seeds across repetitions ensures no repetition of an experimental condition will generate exactly the same results.

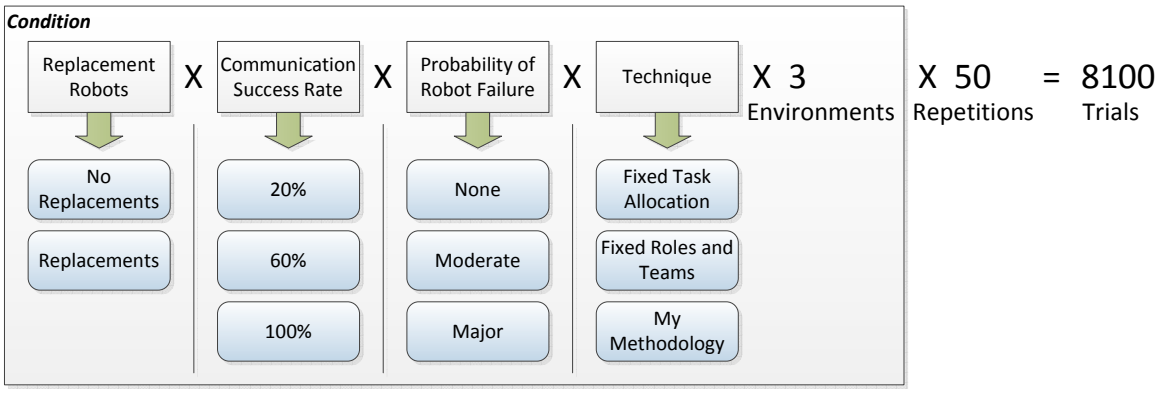


Figure 5.2: Main experiment variables and levels.

5.5 Experiment Design

For my main experiment, I compare my methodology against two base case conditions (Section 5.5.1). Since my methodology aims to cope with communication and robot failures, two of the independent variables in my experiment control the communication success rate (Section 5.5.2.2), and the probability of robot failure (Section 5.5.2.3). In addition to responding to failures, my methodology also allows a team to adapt to accommodate the arrival of replacement robots. I include another independent variable in my experiment, which determines whether replacement robots are available or not (Section 5.5.2.1).

As shown in Figure 5.2, I chose a factorial design for my main experiment. My experimental conditions consist of every unique combination of the three independent variables, and the three techniques. I performed 50 repetitions of each condition, in each of the 3 experimental environments, for a total of 8100 experimental trials.

5.5.1 Base Cases

I compare my methodology against two base cases. The base cases provide a basis for comparing my methodology against the kinds of techniques commonly used in current works involving teams of robots.

5.5.1.1 Fixed Roles and Team Membership

In the first base case, robots are not permitted to change roles and cannot switch teams. This means team structure is fixed: teams cannot gain team members, and team members cannot voluntarily leave the team. Further, robots are not able to change their roles or team membership to adapt to changes to the team as a result to failures. This is intended to provide a basis for comparison against my framework's performance from the standpoint of adaptive team management. Because tasks are still allocated in the base case using my task allocation methodology (Section 3.5.3), it should allow me to show how much of any performance improvement is due to improved team management.

This also allows some comparison with other work. This base case is similar in spirit to Parker et al. [2003]'s approach to coordinating the deployment of robots in an environment. In Parker et al. [2003]'s work, the leader, follower, and sensor robots can only ever carry out the responsibilities pre-determined for them at the start of the mission.

5.5.1.2 Fixed Task Allocation

The second base case uses fixed roles and team membership, and includes the restriction that there is a one to one mapping between tasks and robot types. This means each task type can only be carried out by one type of robot. Team coordination responsibilities can only be carried out by MaxBots, *confirm victim* tasks can only be carried out by MidBots, and *explore frontier* tasks can only be carried out by MinBots.

Using a fixed task mapping as well as no team management essentially provides a worst-case comparison for multi-robot exploration in this domain: I can compare my approach and the base case above to a situation where all agent interaction is completely inflexible.

Using a fixed mapping of tasks to robots in this manner is similar in spirit to Kiener and von Stryk [2007], where there is a single mapping of tasks to the robot types.

5.5.2 Independent Variables

5.5.2.1 Replacement Robots

The *replacement robots* independent variable aims to examine how my methodology deals with the availability of replacement robots. My experimental design treats replacement robots in a binary fashion (Figure 5.2): replacement robots are either available or unavailable. Where replacement robots are available, the environment includes 10 replacement MinBot robots, 2 replacement MidBot robots, and 1 replacement MaxBot robot. Replacement robots begin operation after 5 minutes. This

emulates a scenario where replacement robots are introduced after the start of the rescue operation to offset damage or loss, or where another rescue team has arrived with additional resources to augment the mission.

The case where no replacements are available assumes the robots available at the start of the mission are the only robots available. This case examines my framework's ability to allow existing teams to adapt to failures which have occurred, or the absence of a robot due to it becoming lost.

5.5.2.2 Communication Success Rate

The *communication success rate* independent variable determines the probability any given message will be successfully sent to all robots in radio range. It makes use of the simulated unreliable wireless communication facilities, which I added to the Stage simulator (Section 4.9.1). My experiment uses success levels of 20%, 60%, and 100% (Figure 5.2). The 20% success level represents an environment where communication is extremely unreliable, and a vast majority of messages are not sent successfully. The 60% success level represents a scenario where a majority of messages are sent successfully. This level is representative of conditions I would expect in a disaster environment, where communication is possible, but still unreliable. The 100% success level represents a scenario where all messages are sent successfully and allows a comparison of results with a scenario where communication success is not an issue.

5.5.2.3 Probability of Robot Failure

The *probability of robot failure* independent variable determines the probability that a robot will suffer either a temporary or a permanent failure in each time step of

the simulation trial (Figure 5.2). Both the probability of temporary failure, and the probability of permanent failure are specified separately for each robot type. When a failure occurs, a robot is immobile and its control software does not run for the duration of the failure, nor can the robot send or receive any communications. If the failure is permanent, the robot ceases operation for the remainder of the trial. The length of a temporary ranges from 180 to 240 seconds (3 to 4 minutes). Replacement robots cannot suffer any failures until they begin operation.

Algorithm 14 Determining if a robot fails during a time step.

Require: P_{perm} = probability $\in [0, 1]$ robot will fail permanently

Require: P_{temp} = probability $\in [0, 1]$ robot will fail temporarily

R = random real number $\in [0, 1]$.

if $R \leq P_{perm}$ **then**

Robot fails permanently

else if $P_{perm} < R \leq (P_{perm} + P_{temp})$ **then**

T = random integer $\in [180 \cdots 240]$

Robot fails for T seconds

end if

Algorithm 14 shows the process by which a robot determines whether it will suffer a failure in a time step. Given the probability the robot will fail permanently (P_{perm}) and the probability the robot will fail temporarily (P_{temp}), the algorithm chooses a random real number (R) from 0 to 1.0, inclusive. If $R \in [0, P_{perm}]$, the robot will suffer a permanent failure for the remainder of the trial. If $R \in (P_{perm}, P_{perm} + P_{temp}]$, the robot will choose a random integer (T) from 180 to 240, and remain in a failed

Level	Robot Model	Prob. Permanent Failure	Prob. Temporary Failure	Avg. % Total Time Failed
None	MinBot	0.0000	0.000	**0.0%**
	MidBot	0.0000	0.000	**0.0%**
	MaxBot	0.0000	0.000	**0.0%**
Moderate	MinBot	0.0000	0.008	**14.9%**
	MidBot	0.0000	0.006	**11.9%**
	MaxBot	0.0000	0.004	**9.0%**
Major	MinBot	0.0002	0.014	**25.1%**
	MidBot	0.0002	0.012	**20.9%**
	MaxBot	0.0002	0.010	**18.5%**

Table 5.1: Levels of the *probability of robot failure* independent variable.

state for that duration of time.

Table 5.1 shows the three levels of the *probability of robot failure* independent variable I use in my main experiment (Section 5.5), and the individual failure probability parameters for each robot type. The probability of failure parameters are set for each robot type such that the most reliable robot type is the MaxBot, and the least reliable robot type is the MinBot. The MidBot robot type falls in between the other two. I assume the more capable robot types will be more reliable, due to more expensive construction, and due to the fact they will likely be spending more time on coordination activities rather than exploration. The final column in Table 5.1 shows the average percentage of time during a mission each robot type can be expected to be in a failed state.[1]

The *none* level represents the case where robots do not suffer from permanent or temporary failures (Table 5.1). The failure probabilities of the *moderate* level are set so none of the robot types will suffer a permanent failure. The MinBot, MidBot,

[1]Average percentage of total time failed over 100 trials, each with a duration of 30 minutes.

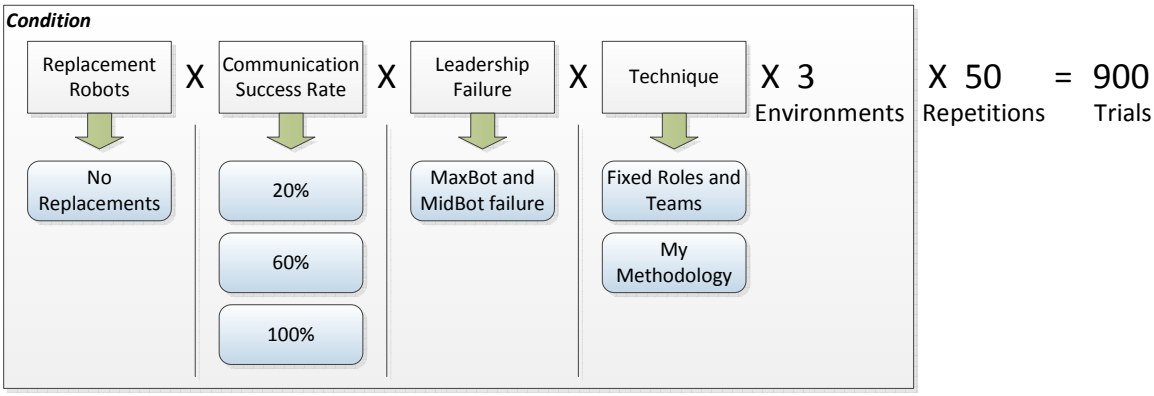


Figure 5.3: Leadership failure experiment variables and levels.

and MaxBot robot types will spend on average 14.9%, 11.9%, and 9.0% of the time failed at this level, respectively. The failure probabilities of the *major* level are set so that all robot types have a small probability of suffering from a permanent failure. The MinBot, MidBot, and MaxBot robot types will spend on average 25.1%, 20.9%, and 18.5% of the time failed at this level, respectively. I chose a low probability of permanent failure, as higher levels caused a high amount of variation in the average percentage of total time failed. This makes sense, as suffering a permanent failure early on in a trial versus late in the trial can have a considerable impact in the overall progress of the team. The leadership failure experiment, detailed in the next section, examines the impact of permanent failures in a controlled manner.

5.6 Leadership Failure Experiment

I performed an additional experiment to explicitly study the impact of failures experienced by the robots filling the *team coordinator* role on a team. Although my main experiment includes consideration of random robot failures throughout the

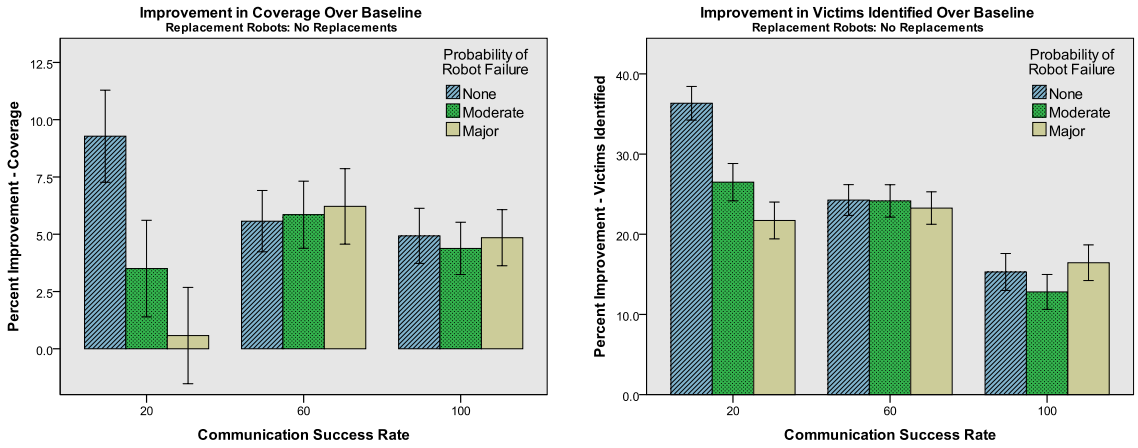
trials, these failures never occur at predictable times, making it difficult to make specific observations about individual failures. This new experiment introduces failures of robots capable of leadership roles at predetermined times, allowing for more effective analysis of their impact.

Figure 5.3 shows the design of this experiment. Since the purpose of the experiment was purely to study the impact of leadership failures on a team, I included only a single team, and no replacement robots. Similar to my main experiment, I varied the communication success rate (20%, 60%, 100%). I did not include random probabilistic failures for any robots. In all trials, the MaxBot fails after 10 minutes of operation, and one of the MidBots fails after 15 minutes of operation. This experiment compares my methodology against a scenario where the roles and team membership is fixed at the start of the trial, and cannot change (Section 5.5.1).

I ran each experiment condition 50 times, in each of the 3 environments from my main experiment (Section 5.5), for a total of 900 trials.

5.7 Main Experiment Results

This section describes the results I obtained from my main experiment. The results are grouped first by whether there were replacement robots available. Within each group, subsections show the impact of communication failure, and the impact of robot failures. Table B.1 (Appendix B.1) provides a cross-reference of the results for all combinations of the independent variables. On all charts in the sections that follow, error bars represent 95% confidence intervals.



(a) (b)

Figure 5.4: Factorial experiment, performance improvement in terms of coverage and victims successfully identified, where replacement robots are not available, compared to the *fixed roles and team membership* base case (Section 5.5.1.1).

5.7.1 No Replacement Robots

Figure 5.4 provides a summary of the performance improvements realized using my methodology, compared to the base case where roles and team membership are fixed (but not the allocation of tasks) (Section 5.5.1.1). The charts use statistics captured at the end of each experimental trial and show the performance improvement realized, for each level of the communication success rate and probability of robot failure independent variables.

5.7.1.1 Improvement in Coverage over Baseline

Figure 5.4a shows the improvements in coverage realized using my methodology, compared to the base case where roles and team membership are fixed (but not the

allocation of tasks). With the exception of the case where both communication is extremely unreliable (20% success rate), and robots experience high levels of failures (major failure probability), my methodology results in an average of approximately a 5% improvement in coverage over the baseline.

Where the communication success rate is 20% and robots do not suffer from failures (Figure 5.4a), my methodology showed a considerable improvement in performance over the baseline. When observing simulation runs at these levels, I noticed extremely unreliable communication caused task assignment to fail, leading to the members of a team falling back to undirected wandering. Where roles and team membership were fixed (but not the allocation of tasks), team members increasingly spread apart as they wandered, eventually losing contact with their team coordinator. Using my methodology, smaller grouping of agents were able to form as the team broke apart. Where a MidBot was able to take on the team coordinator role for a subset of the team, there was a chance for the smaller teams to still perform some useful work, despite the overall poor communication. In the instances I observed, however, robots still spend a large amount of time carrying out their *wander* idle tasks, indicating their efforts are not as effectively coordinated as they could be, due to the poor communication.

At the 20% communication success rate, the performance improvement quickly drops as the probability of robot failure increases (Figure 5.4a). Where the probability of robot failure was *major*, my methodology could result in a slight decrease in coverage (when the confidence interval is included), compared to the baseline technique. The sharp drop in performance indicates that a 20% communication success

rate is not sufficient to facilitate an effective coordination of the exploration efforts. The poor communication results in frontier exploration tasks frequently failing to be assigned, which means the team coordinator builds a large backlog of unassigned tasks (recall from Section 3.5.3, task assignment will only attempt to assign the 5 oldest tasks at one time). The poor communication conditions also prevent the results from the few frontier exploration tasks that do get successfully assigned from being reported back to the team coordinator. Without the knowledge gained from the frontier exploration tasks, the team coordinator will not remove older frontier exploration tasks which have become obsolete, resulting in tasks being assigned to explore frontiers which likely no longer exist.

At the 60% and 100% communication success rate, as the probability of robot failure increases, the improvement in the percentage of the environment covered does not vary significantly. Replaying some experimental trials revealed the random duration temporary failures of robots (Section 5.5.2.3) were short enough that teams did not recognize the failures and adjust their structure to compensate. The temporary failures, which are 3-4 minutes in duration, appeared to end before a team recognized the absence of the failed robot. A robot must not be heard from for 3 minutes before it is forgotten (Section 4.7.2). Useful future work would be to run an experiment to further investigate the impact of short duration failures, given different memory durations, and to investigate whether a team could fine-tune this parameter based on its experience. The leadership failure experiment results (Section 5.8), however, include predictable, long duration failures which allow for a better comparison between my methodology and the baselines. Short term failures did, however, sometimes result

in the failed robot's team moving out of communication range from the failed robot, resulting in it becoming separated from its team upon resuming operation.

5.7.1.2 Improvement in Victims Identified over Baseline

Figure 5.4b shows the improvement in the percentage of victims successfully identified using my methodology. The improvements achieved over the baseline were consistently higher than the improvements in the area covered.

At the 20% communication success rate, as the probability of robot failure increases, the percentage of victims successfully identified compared to the baseline drops similarly to the percentage covered, at the same levels of communication success rate and robot failures (Figure 5.4b). Despite this, however, the improvement is still greater than 20%, unlike with the percentage of the environment covered, where it is possible for a decrease in performance to occur when the confidence interval is considered. The relatively small density of victim confirmation tasks in an environment (there are at most 30) means that despite the extremely poor communication, a large backlog of victim confirmation tasks will not accumulate. Using my methodology, the formation of smaller teams as the larger teams break apart means the victim confirmation tasks have a better chance of being assigned. Further, when a MidBot assumes the team coordinator role, it has the ability to carry out victim identification tasks on its own, avoiding the need to assign these tasks to another robot.

At the 60% and 100% communication success rates, as the probability of robot failure increases, the performance improvement compared to the baseline does not vary significantly, similar to what was observed with the improvement in area covered.

The same observations made in Section 5.7.1.1 apply to the improvement in the percentage of victims successfully identified: the short duration of robot failures were the most likely cause of the small differences observed.

5.7.1.3 Impact of Communication Failures over Time

This section investigates the impact of the communication success rate on the percentage of the environment covered and percentage of victims successfully identified, over the duration of the trials. Since the intention is to investigate the impact of unreliable communication, these observations compare the impact of the communication success rate with a fixed probability of robot failure. Graphs showing the results for the other levels of the probability of robot failure variable are cross referenced in Table B.1 in Appendix B. To ensure my analysis of the impact of communication failure occurs in realistic conditions, I chose to highlight the results when the probability of robot failure is moderate.

Figures 5.5 and 5.6 show the average percentage of the environment covered and percentage of victims successfully identified over the duration of the 30 minute (1800 second) experimental trials, where the communication success rate is 20%. These graphs compare my methodology against both the base case where roles and team membership is fixed (but not the allocation of tasks) (Section 5.5.1.1), and the base case where task allocation is also fixed (Section 5.5.1.2).

The percentage of the environment covered (Figure 5.5) for my methodology and the base case where roles and team membership (but not the allocation of tasks) are fixed, at the same rate until 800 seconds (13 minutes) into the trials. Observing trial

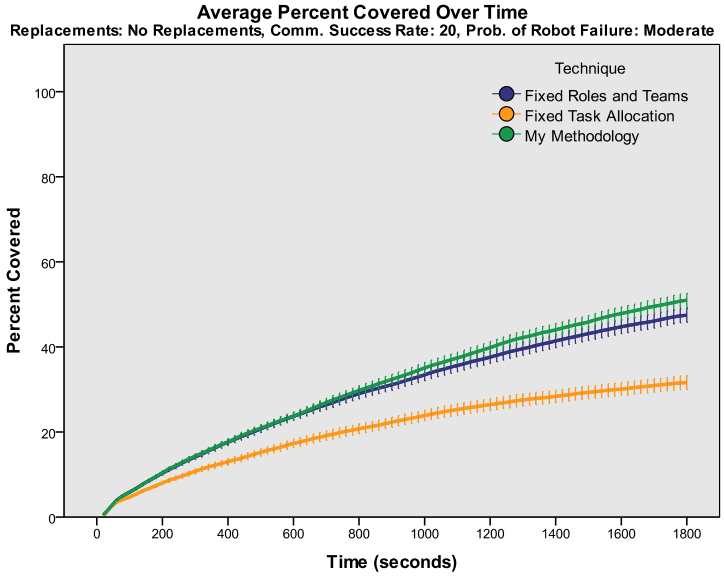


Figure 5.5: Main experiment, coverage over time where: no replacements are available, comm. success rate = 20%, and prob. robot failure = moderate.

runs, I noticed by this point in time the teams had spread out enough to cause robots to become separated from their team. Using my methodology, the separated team members were able to restructure themselves into smaller teams, where with the base cases, the separated agents would generally wander off, and be unable to contribute further. This allowed my methodology to achieve a minor gain in performance over the baseline where roles and team membership (but not the allocation of tasks) is fixed.

The base case where task allocation is fixed (as well as roles and team membership) performed considerably worse than the other cases (Figure 5.5). After only 100 seconds, the lack of communication provided a considerable hindrance to progress. The fixed mapping of tasks to robot types reduces the potential number of agents the explore frontier tasks can be assigned to, making it less likely for these tasks to be

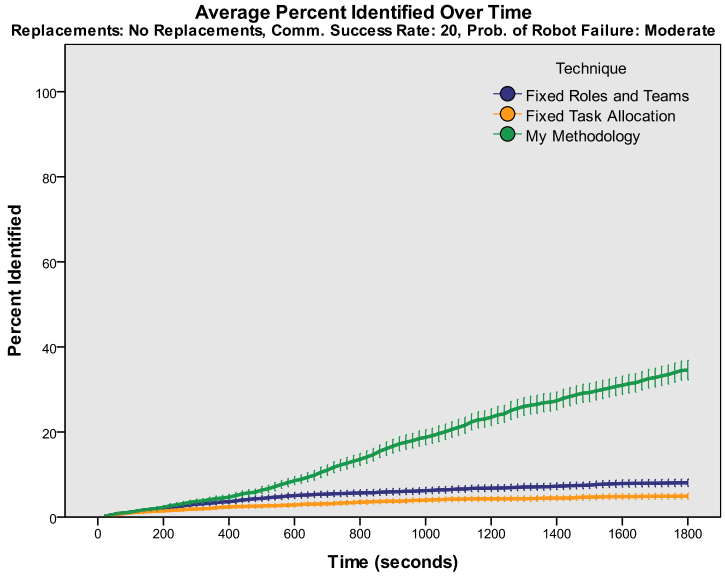


Figure 5.6: Main experiment, victims identified over time where: no replacements are available, comm. success rate = 20%, and prob. robot failure = moderate.

assigned.

Figure 5.6 shows the average percentage of victims successfully identified over the duration of each trial. Both base cases performed similarly, leveling off in performance soon after the start of the trial, and achieving only minor gains. The extreme lack of communication resulted in a general failure to assign tasks in both cases. Unlike with the frontier exploration tasks, victim identification tasks cannot be carried out by the team coordinator. An inability to assign tasks due to poor communication means the team is unable to make significant progress identifying victims.

My methodology showed a considerable improvement over the baseline cases, as the teams were able to break up into smaller teams composed of the geographically close robots, which were occasionally assign some victim identification tasks, despite the poor communication conditions. Further, when a MidBot fills the *team coordi-*

nator role, it can carry out some of the victim identification tasks by itself without relying on wireless communication. The severe communication failures still resulted in a general lack of coordination amongst team members, however. The team splits tended to occur near the middle of the trials I observed, and had not broken apart further by the end of the 30 minute trials. Given the general lack of coordination at the 20% communication level, I suspect the smaller teams would also end up breaking apart if the trials were extended beyond 30 minutes.

Ultimately, a 20% communication success rate does not appear sufficient to support meaningful coordination of the exploration and victim confirmation activities. This is not surprising, as a majority of messages do not get through, leading to robots falling back to their idle tasks. In effect, the teammates are operating autonomously, with very little supervision by the team coordinator.

As shown in Figure 5.7, at the 60% communication success rate, my methodology performed similarly to the base case, in terms of the percentage of the environment covered, where roles and team membership (but not the allocation of tasks) is fixed, up until around the 200 second (approx. 3 minute) mark. At that point, the performance of my methodology began trending up from the fixed roles and team membership (but not the allocation of tasks) baseline, and continued to diverge through to the end of the trials.

The baseline where roles and team membership (but not the allocation of tasks) are fixed showed a significant performance gain, compared to the fixed task allocation baseline, throughout the trials (Figure 5.7). This is to be expected, as fixing the task mapping to specific robot types considerably limits the pool of agents available to

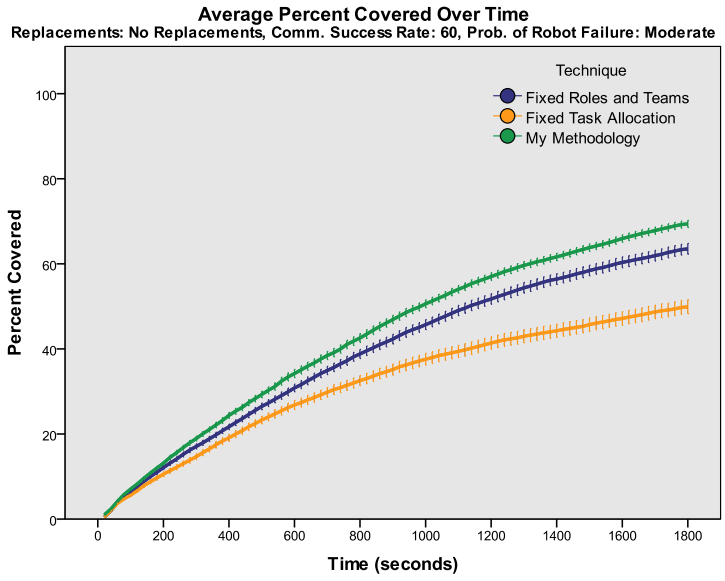


Figure 5.7: Main experiment, coverage over time where: no replacements are available, comm. success rate = 60%, and prob. robot failure = moderate.

carry out any one task.

Figure 5.8 shows the average percentage of victims successfully identified during the duration of each trial, where the communication success rate is 60%. My methodology shows a considerable improvement over the baseline, beginning at the 600 second (5 minute) mark in the trials. Using my methodology, the percentage of victims successfully grows at an increased rate compared to the baseline. This means that victims are being found more quickly using my approach.

Finally, when the communication success rate is 100%, in terms of the percentage of the environment covered (Figure 5.9), my methodology performs similar to the base case where roles and team membership (but not the allocation of tasks) are fixed. This makes sense, as there is no communication failure for my methodology to compensate for, and neither case can benefit from being able to flexibly adapt to the

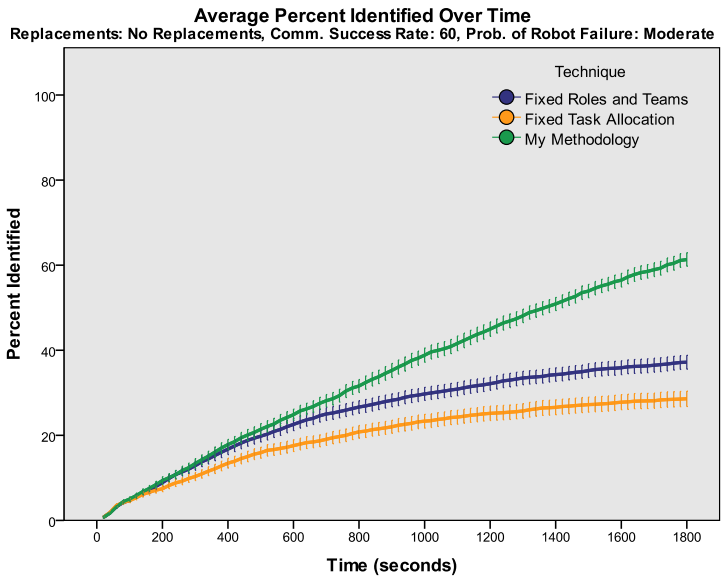


Figure 5.8: Main experiment, victims identified over time where: no replacements are available, comm. success rate = 60%, and prob. robot failure = moderate.

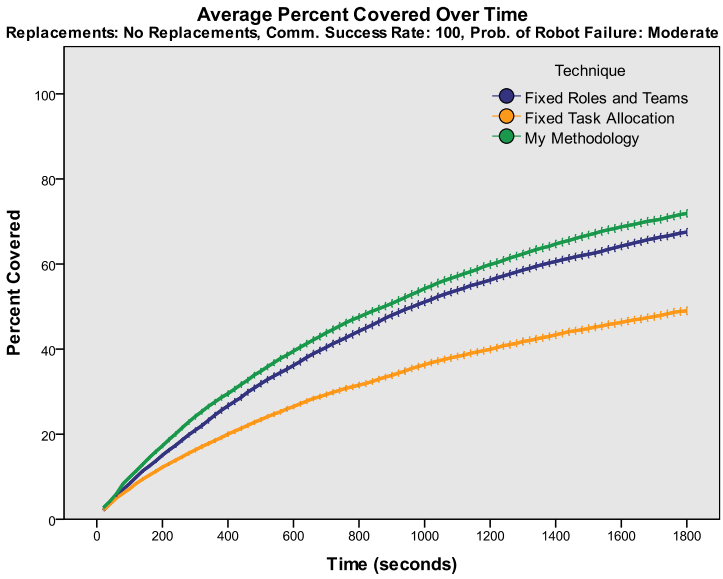


Figure 5.9: Main experiment, coverage over time where: no replacements are available, comm. success rate = 100%, and prob. robot failure = moderate.

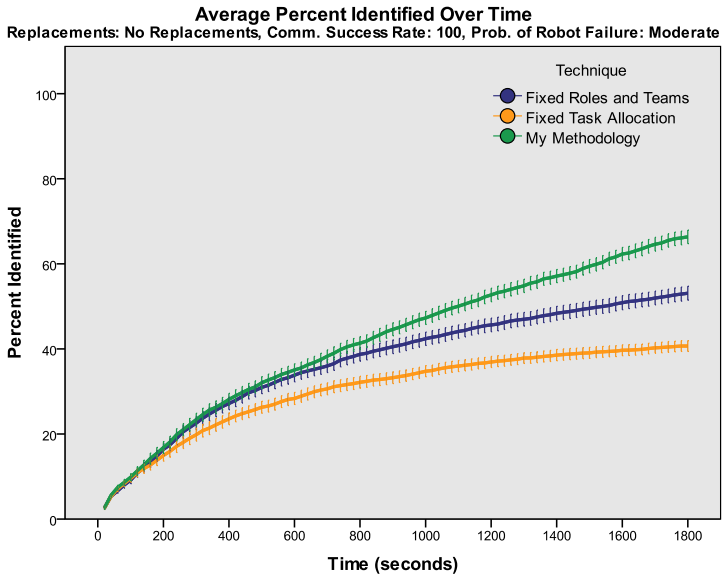


Figure 5.10: Main experiment, victims identified over time where: no replacements are available, comm. success rate = 100%, and prob. robot failure = moderate.

presence of new team members. The performance improvements seen are reflective of the fact robots can still fail or become separated from their team, despite ideal communication conditions.

The percentage of victims successfully identified, however, does show an improvement over the baseline where roles and team membership (but not the allocation of tasks) are fixed, and the communication success rate is 100% (Figure 5.10). The divergence in performance from the baseline, despite perfect communication is likely due to teams adapting to a MidBot robot becoming separated from the team. In such a case, the MidBot would form a team of its own, or join another team. In either case, it is able to keep contributing its victim identification skills to a team, where in the base cases the loss of this important skill significantly hampers the team victim identification efforts.

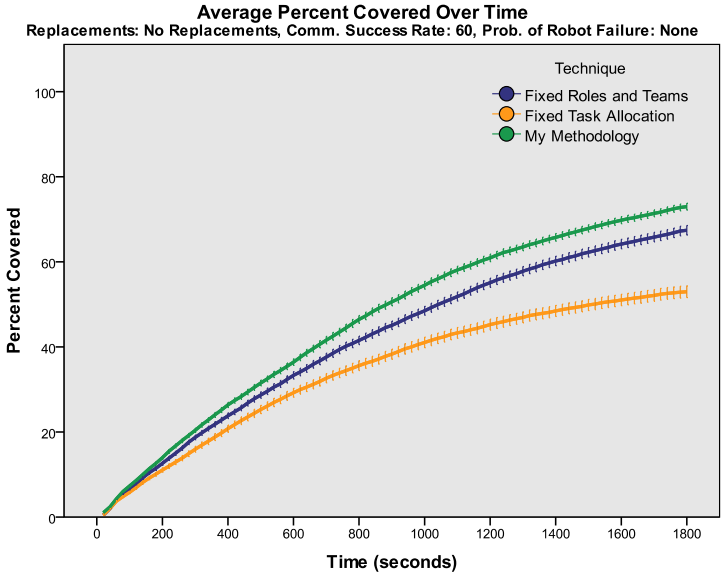


Figure 5.11: Main experiment, coverage over time where: no replacements are available, comm. success rate = 60%, and prob. robot failure = none.

5.7.1.4 Impact of Robot Failures over Time

This section investigates the impact of the probability of robot failure (Section 5.5.2.3) on the percentage of the environment covered and percentage of victims successfully identified, over the duration of the trails. Similar to Section 5.7.1.3, a fixed level of the communication success rate variable is used. Graphs showing the results for the other levels of the communication success rate variable are cross referenced in Table B.1 in Appendix B. I chose to analyze the impact of robot failure where the communication success rate is 60%, as I assume this is a reasonable representation of the conditions my methodology is expected to operate in.

Figures 5.11 and 5.12 show the percentage of the environment covered and percentage of victims successfully identified over the duration of the trials, where there is

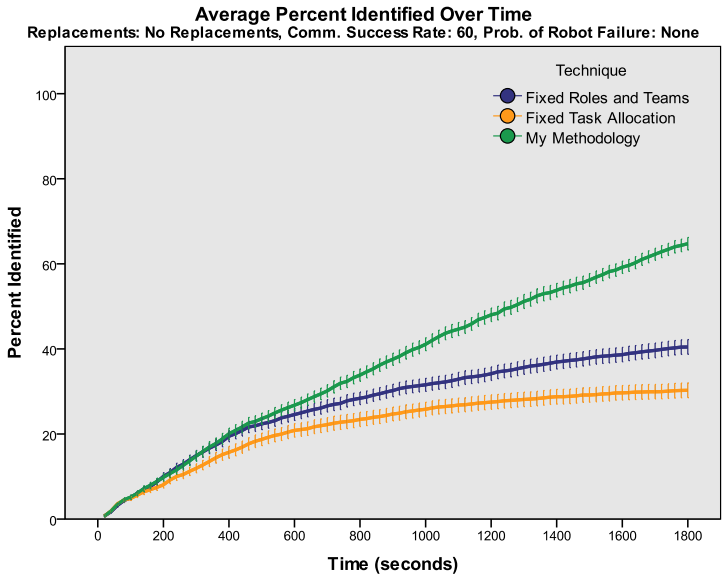


Figure 5.12: Main experiment, victims identified over time where: no replacements are available, comm. success rate = 60%, and prob. robot failure = none.

no probability of robot failure. As expected, where robots do not suffer from failures, the differences seen between the techniques can be traced to other sources. As previously mentioned in Section 5.7.1.3, when robots become separated from their team, my methodology provides the team with the opportunity to adapt to the loss of the robot, and for the lost robots to form a new team or their own or join another team. The small difference in the percentage of the environment covered, seen in Figure 5.11, is partly due to the fact all members of a team area able to contribute to the exploration efforts. This means if a single member becomes separated from the team, the remaining team will still be able to perform well without it. In contrast, when a MidBot, the only robot capable of identifying victims, becomes separated from the team, my methodology results in a large improvement in performance, compared to the baselines. This is seen in Figure 5.12, where my methodology achieves a consid-

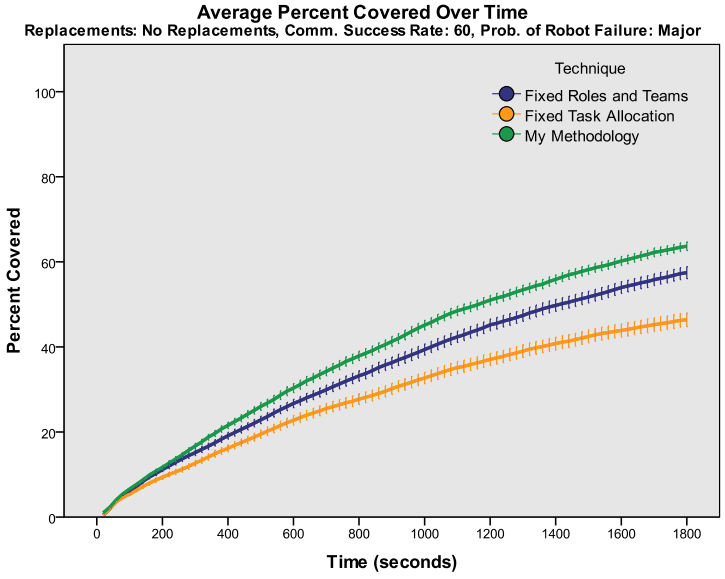


Figure 5.13: Main experiment, percentage of environment covered over time where: no replacements are available, comm. success rate = 60%, and prob. robot failure = major.

erable gain in performance over time, compared to the baseline cases. Observations of trial runs revealed this was due to a MidBot becoming lost. My methodology enables the MidBot to form its own team, or join another team, allowing it to continue performing useful victim identification work. In the baseline cases, a loss of a MidBot has a large impact on the team's ability to quickly identify victims.

The percentage of the environment covered over time for the three probabilities of robot failure (none (Figure 5.11), moderate (Figure 5.7), and major (Figure 5.13)) exhibit the same general trend. As previously explained in Section 5.7.1.1, my methodology achieved approximately a 5% improvement over the baseline where roles and team membership (but not the allocation of tasks) is fixed. This improvement is realized over the course of each trial, building gradually. Increasing the probability

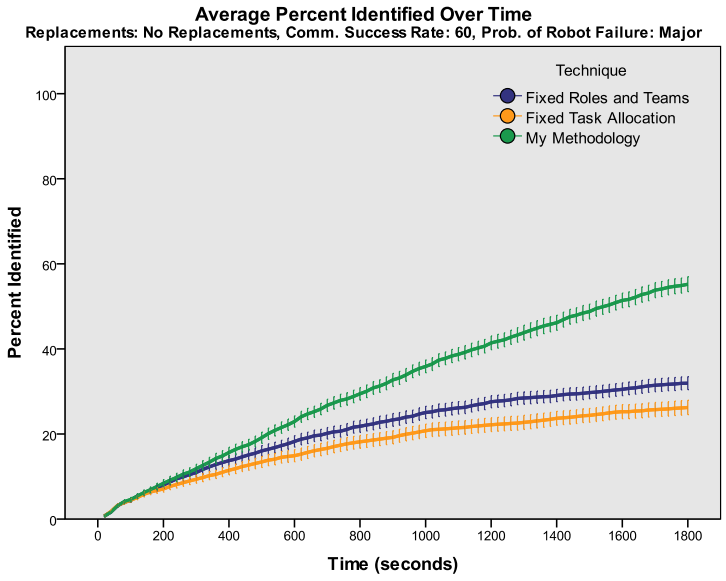
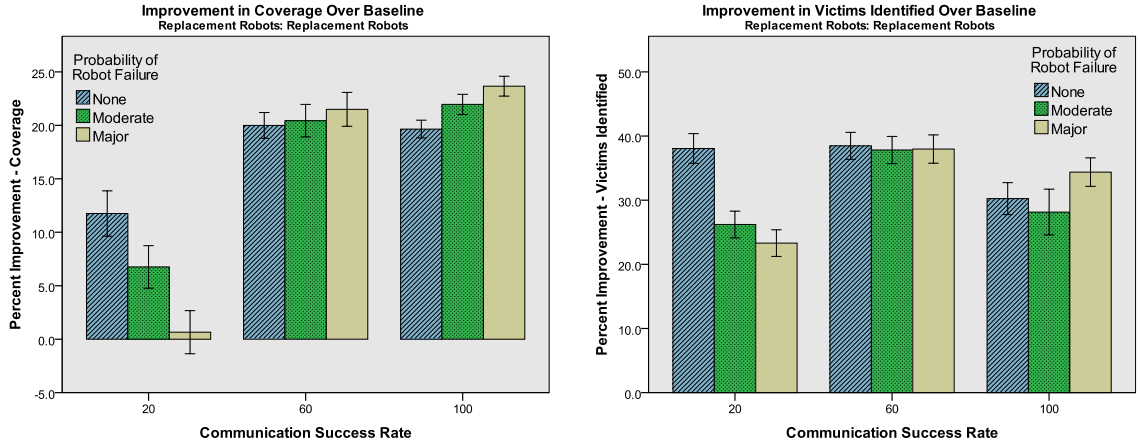


Figure 5.14: Main experiment, victims identified over time where: no replacements are available, comm. success rate = 60%, and prob. robot failure = major.

of robot failure results in the performance graphs shifting down due to the impact of the increasing failures.

The percentage of victims successfully identified over time for the three probabilities of robot failures (none (Figure 5.12), moderate (Figure 5.8), and major (Figure 5.14)) also exhibit the same general trend. The improvement realized over the course of the trials averages approximately 20% (Section 5.7.1.1). The improvement is realized over the course of the trials, beginning to build around the 500 second (8 minute) mark, growing steadily until the end of the trials. During the initial 500 seconds (8 minutes) of operation, robots were still clustered in the same general geographic area, reducing the likelihood of a robot becoming separated from its team.



(a) (b)

Figure 5.15: Performance improvement in terms of coverage and victims successfully identified, where replacement robots are available, compared to the *fixed roles and tasks* base case.

5.7.2 Replacement Robots

Figure 5.15 illustrates the performance improvements realized using my methodology, when replacement robots are available, compared to the base case where roles and team membership (but not the allocation of tasks) are fixed (Section 5.5.1.1). The charts show the performance improvement realized, for each level of the communication success rate and probability of robot failure independent variables.

5.7.2.1 Improvement in Coverage over Baseline

Figure 5.15a shows the improvements in coverage realized using my methodology, compared to the base case where roles and team membership (but not the allocation of tasks) are fixed (Section 5.5.1.1). Similar to the situation where no replacements are

available (Section 5.7.1.1), trials using my methodology show a considerable improvement over the baseline, with the exception of the case where both communication is extremely unreliable (20% success rate), and robots experience high levels of failures (major probability of robot failure).

At the 20% communication success rate, a rapid decrease in performance was observed as the probability of robot failure increased (Figure 5.15a), similar to where no replacement robots are available (Figure 5.4a). When the probability of robot failure is *none* or *moderate*, the availability of replacement robots results in a further 4% improvement in coverage when compared to trials where no replacement robots are available (Figure 5.4a).

At the 20% communication success rate, when the probability of robot failure is *major*, however, my methodology can result in a decrease in performance compared to the baseline. Similar to when no replacement robots are available (Figure 5.4a), when the probability of robot failure is *major*, the combination of poor communication and robot failures results in the team coordinator performing a poor job at coordinating the exploration efforts. Further, the poor communication results in the encounters involving replacement robots having a high chance of failing to complete. This makes the replacements less likely to join a team, or form a team of its own.

Where the communication success rate is 60%, and 100%, my methodology achieves approximately a 20% improvement in the percentage of the environment covered, over the baseline (Figure 5.15a). Recall, where no replacements are available, my methodology resulted in approximately a 5% improvement in the percentage of the environment covered, over the baseline (Figure 5.4a). This makes sense, as using my

methodology, teams are able to adjust their structure to accommodate the arrival of the replacement robots.

When comparing the 60% and 100% communication success rates improvements (Figure 5.15a), my methodology performs slightly better when the communication success rate is 100%. The reverse is true where no replacement robots are available (Figure 5.4a). This is likely the result of better communication conditions making it easier for robots to complete encounters, without being interrupted by communication failures. Successful encounters make it more likely replacement robots will join a team, or form a team of their own.

The improvement in the percentage of victims successfully identified when replacement robots are available, observed in Figure 5.15a, closely resemble those seen when no replacement robots are available (Figure 5.4a). When the communication success rate is 60% or 100%, the improvement seen in the percentage of victims successfully identified is approximately 15% higher than those when no replacements were available. This makes sense, as the replacement robots augment the capability of the teams using my methodology, enabling them to perform more effectively.

When the communication success rate is 20%, the improvements seen in the percentage of victims successfully identified are at most 4% better than those when no replacements are available. Observing some experimental trials revealed that poor communication made the encounters taking place unlikely to succeed, resulting in the replacement robots infrequently joining or forming a new team of their own.

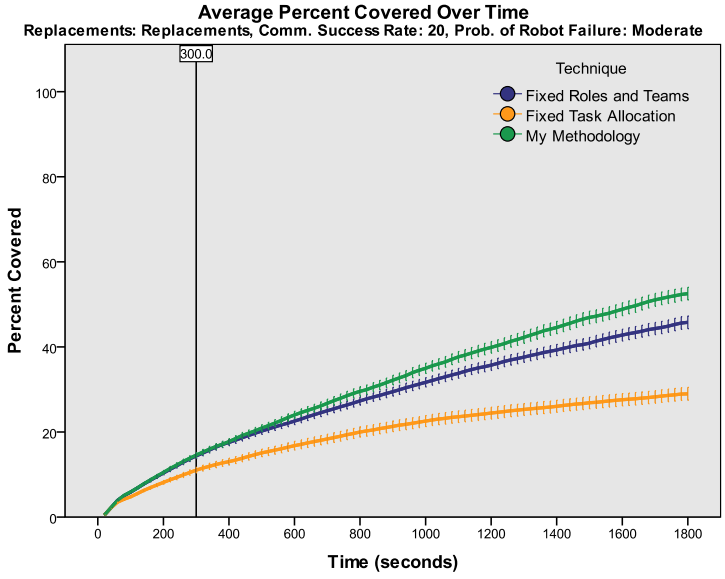


Figure 5.16: Main experiment, coverage over time where: replacements are available, comm. success rate = 20%, and prob. robot failure = moderate.

5.7.2.2 Impact of Communication Failures

This section explores the impact of communication failures on the percentage of the environment covered and percentage of victims successfully identified, over the duration of trials where replacement robots are available. My methodology is compared against both the baseline where roles and team membership (but not the allocation of tasks) are fixed, and the baseline where the allocation of tasks is fixed. Here, I show the results where the probability of robot failure is moderate. Graphs showing results for the other levels of the probability of robot failure variable are cross referenced in Table B.1 in Appendix B.

Figure 5.16 shows the average percentage of the environment covered over the duration of trials, where the communication success rate is 20%, and the probability of

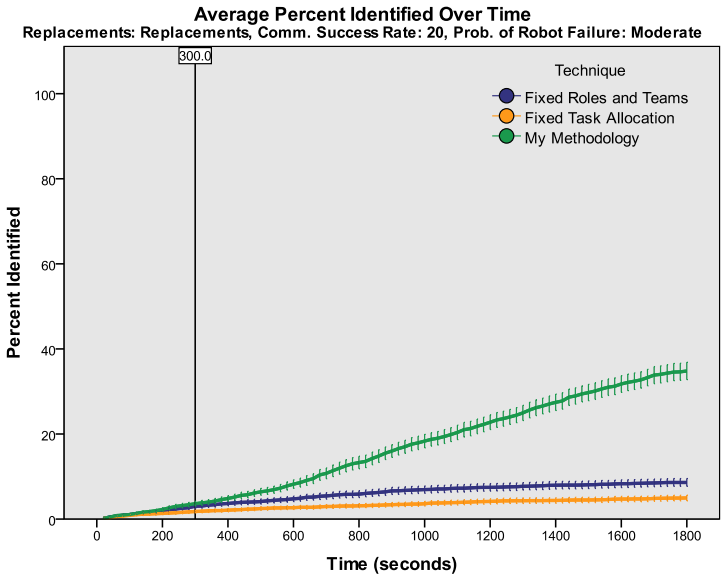


Figure 5.17: Main experiment, victims identified over time where: replacements are available, comm. success rate = 20%, and prob. robot failure = moderate.

robot failure is moderate. The vertical line at the 300 second mark indicates the time at which the replacement robots entered the trial. Similar to when no replacement robots are available (Figure 5.5), my methodology results in a small improvement in the percentage of the environment covered, over the baseline where roles and team membership (but not the allocation of tasks) is fixed. Observing experimental trials at this communication success rate reveals that in most cases, the replacement robots failed to join an existing team or form a new team, due to the communication failures.

The improvement in the average percentage of victims successfully identified over the duration of trials with replacement robots available (Figure 5.17) also closely resembles the improvement where no replacement robots are available (Figure 5.6). This indicates a 20% communication success rate is not sufficient enough for my methodology to be able to effectively make use of replacement robots.

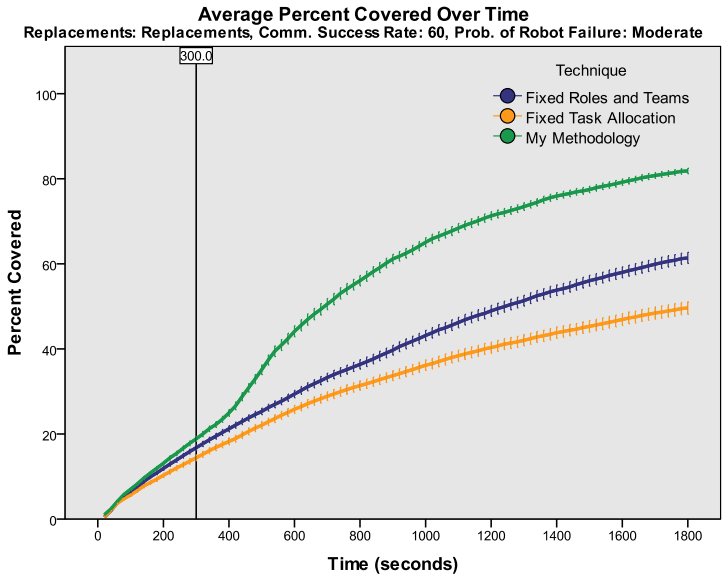


Figure 5.18: Main experiment, coverage over time where: replacements are available, comm. success rate = 60%, and prob. robot failure = moderate.

Where the communication success rate is 60%, as shown in Figure 5.18, my methodology was able to make use of the replacement robots soon after they became available. Approximately 100 seconds after the replacement robots become available, a sharp jump in percent coverage is seen, allowing the teams to establish an even larger lead over the baseline where roles and team membership (but not the allocation of tasks) are fixed, compared to when no replacement robots are available (Figure 5.7). This makes sense, as the teams are able to make use of the replacement robots to augment their capabilities.

A similar jump in the percentage of victims successfully identified in response to the availability of the replacement robots is seen in Figure 5.19.

Where the communication success rate is 100%, the percentage of the environment covered (Figure 5.20) and percentage of victims successfully identified (Figure 5.21)

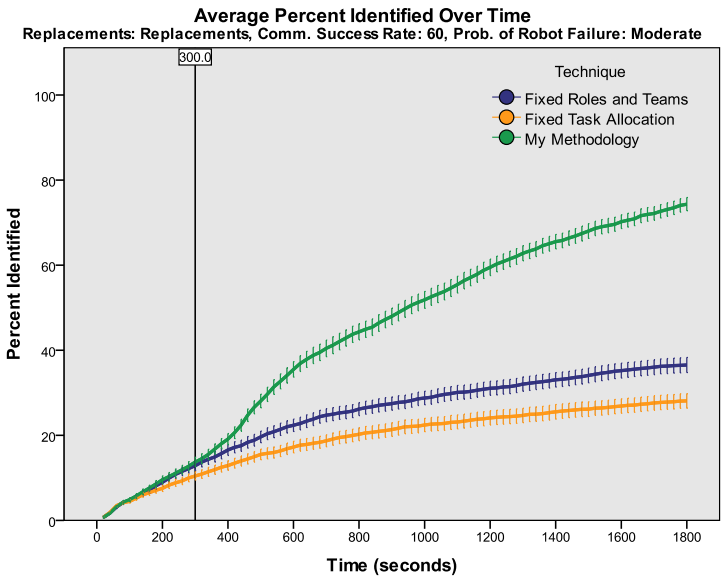


Figure 5.19: Main experiment, victims identified over time where: replacements are available, comm. success rate = 60%, and prob. robot failure = moderate.

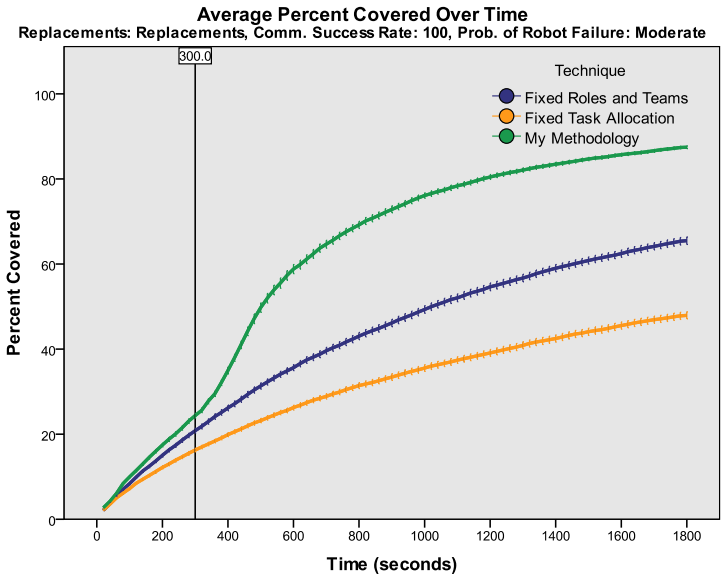


Figure 5.20: Main experiment, coverage over time where: replacements are available, comm. success rate = 100%, and prob. robot failure = moderate.

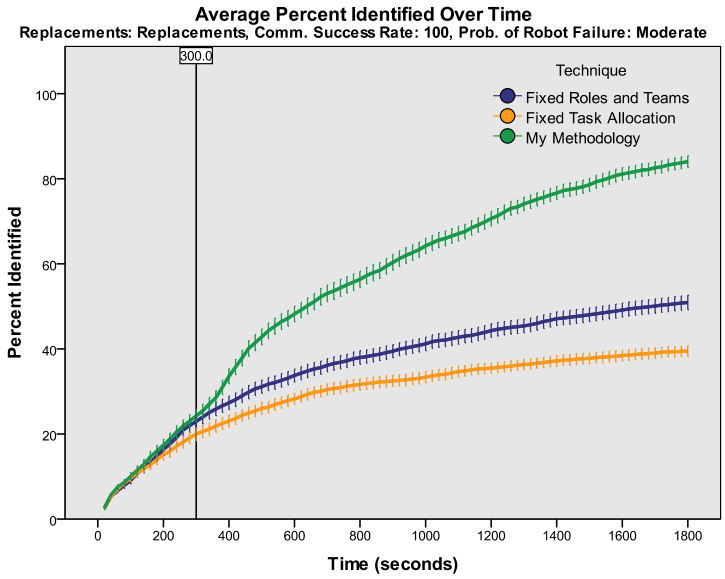


Figure 5.21: Main experiment, victims identified over time where: replacements are available, comm. success rate = 100%, and prob. robot failure = moderate.

show increases in performance for all techniques, due to the better communication conditions. The same general trend is seen, where the replacement robots result in my methodology causing a spike in performance beginning around the 400 second (6.5 minute) mark. The results show a general improvement over when the communication success rate is 60%. This is not surprising as higher levels of communication support more effective coordination, ensuring operations such as the exchange of knowledge, and encounters between robots are able to complete successfully more often. The availability of replacements allows the team to cover a larger area and identify victims earlier in the trial than when no replacements are available.

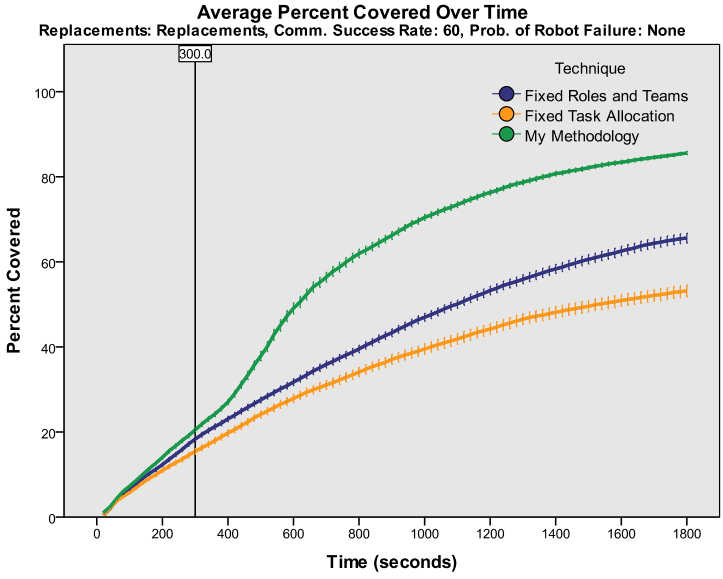


Figure 5.22: Main experiment, coverage over time where: replacements are available, comm. success rate = 60%, and prob. robot failure = none.

5.7.2.3 Impact of Robot Failures

This section investigates the impact of varying the probability of robot failure (Section 5.5.2.3) on the percentage of the environment covered and the percentage of victims successfully identified, over the duration of trials, where replacement robots are available. This section investigates the impact of probabilistic robot failure where the communication success rate is 60%. Graphs showing results for the other levels of the communication success rate variable are cross referenced in Table B.1 in Appendix B.

Figure 5.22 shows the performance of my methodology over the duration of trials, compared to both baselines, where there are no robot failures. The availability of replacement robots at the 300 second (5 minute) mark results in my methodol-

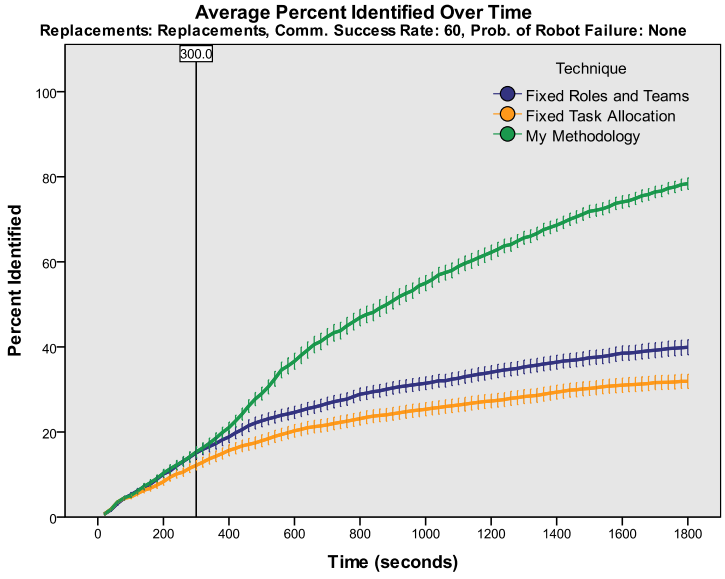


Figure 5.23: Main experiment, victims identified over time where: replacements are available, comm. success rate = 60%, and prob. robot failure = none.

ogy achieving a significant performance improvement over the baseline cases. As explained in Section 5.7.2.2, my methodology enables the replacement robots to join existing teams, or form their own new teams, resulting in the significant performance improvements observed over the baseline.

The same observations apply to the percentage of victims successfully identified when robots do not suffer probabilistic failures, as shown in Figure 5.23. The large jump in performance observed with my methodology is due to the availability of the replacement robots.

The results when the probability of robot failure is *moderate* (Figured 5.18 and 5.24) or *major* (Figures 5.19 and 5.25) are similar in appearance. Increasing the probability of robot failure results in all techniques suffering a similar performance hit, shifting the graph curves down.

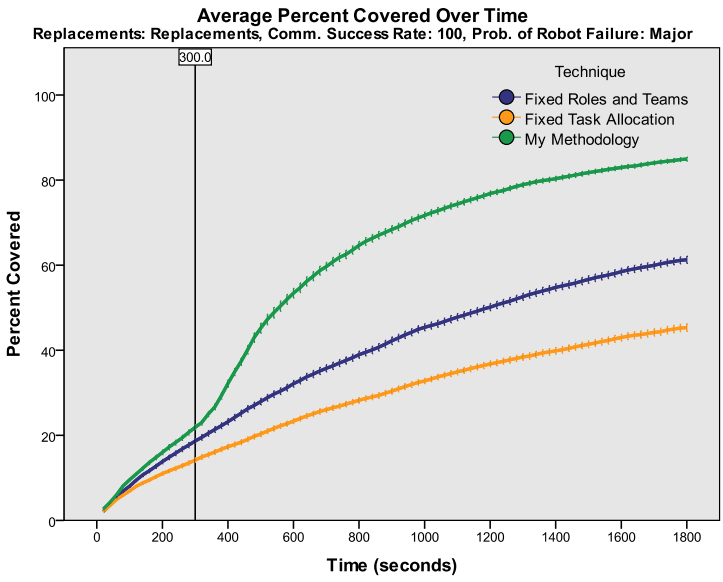


Figure 5.24: Main experiment, coverage over time where: replacements are available, comm. success rate = 60%, and prob. robot failure = major.

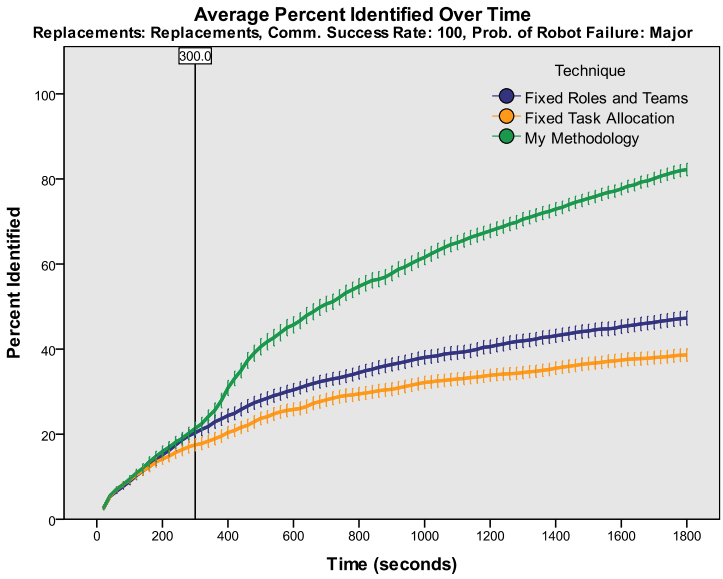


Figure 5.25: Main experiment, victims identified over time where: replacements are available, comm. success rate = 60%, and prob. robot failure = major.

As mentioned in Section 5.7.1.1, my methodology did not appear to result in major differences in performance as the probability of robot failure increased. Since robots fail for a random period of time from 3 to 4 minutes, and failures are only detected once a robot has been absent for 3 minutes, shorter term failures were not likely to be identified. A slight increase can be observed in some of the results, which can be attributed to a team wandering away from a failed robot, and it becoming lost as a result. In such instances, my methodology provides the lost robot with the opportunity to rejoin its team, or a new team it encounters.

5.8 Leadership Failure Experiment Results

This section investigates the results of the leadership failure experiment (Section 5.6) I performed. The leadership failure experiment demonstrates the effectiveness of my methodology in coping with the failure of robots occupying the *team coordinator* role, compared to the baseline where roles and team membership (but not the allocation of tasks) are fixed (Section 5.5.1.1). For the purpose of this analysis, I present results were the communication success rate is 20%, and 60%. Graphs showing results for the remaining 100% level are cross referenced in Table B.1 in Appendix B (the results parallel those observed when the communication success rate is 60%). On all charts in the sections that follow, error bars represent 95% confidence intervals.

The graphs in Figures 5.26 and 5.27 show the percentage of the environment covered and percentage of victims successfully identified, respectively, when the communication success rate is 60%. The vertical line at the 600 second (5 minute) mark indicates the point at which the MaxBot robot filling the team coordinator role failed.

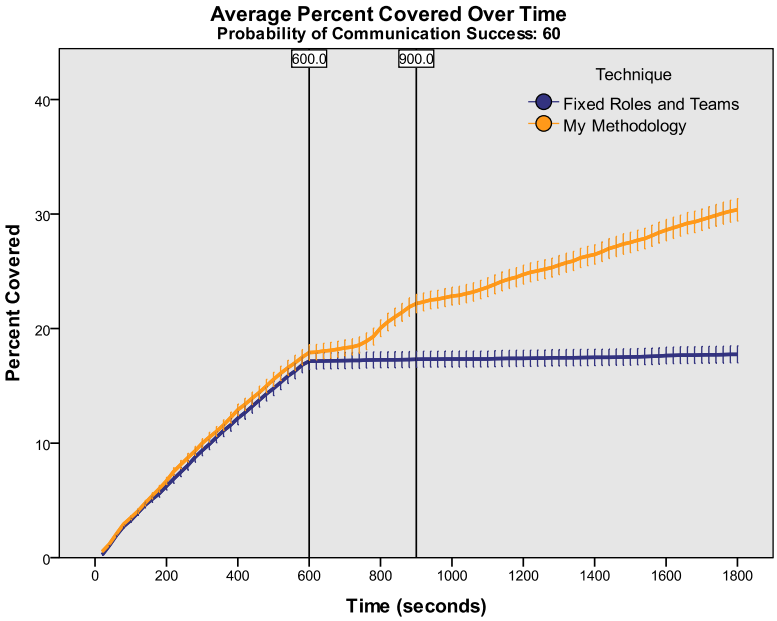


Figure 5.26: Leadership failure experiment, percentage of the environment covered for communication success rate 60%.

The vertical line at the 900 second (15 minute) mark indicates the point at which one of the MidBot robots on the team failed. When the MaxBot filling the role of team coordinator fails, the percentage of the environment covered (Figure 5.26) and percentage of victims successfully identified (Figure 5.27) in the baseline case immediately stops growing. Without the ability to adapt to the loss of the MaxBot, the team is no longer able to benefit from the coordination it provides. Further, the knowledge agents report to their team coordinator as a result of carrying out any backlogged tasks fails to be integrated into the team's consolidated view maintained by the team coordinator.

Using my methodology, the team recognizes the failure of the MaxBot approximately 3 minutes after it fails. In Figures 5.26 and 5.27, my methodology records no

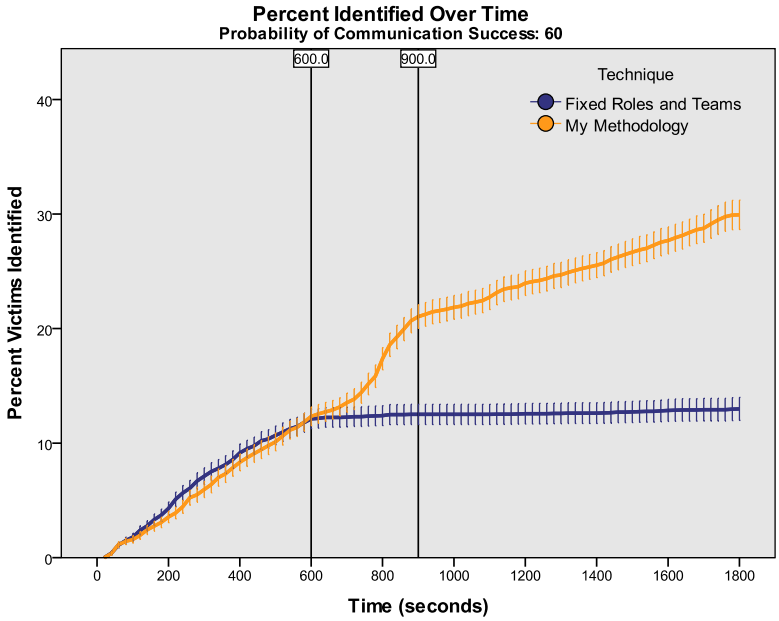


Figure 5.27: Leadership failure experiment, percentage of victims successfully identified for communication success rate 60%.

progress, until one of the MidBot robots takes over the *team coordinator* role (between the vertical 600 and 900 second lines). The team continues to make progress, until the 900 second mark, at which point one of the MidBot robots fails. A slight leveling off in performance can be observed for 3 minutes, after which progress continues. The second failure is not as evident in the graphs, as in some trials the MidBot suffering a permanent failure at 900 seconds may not be the MidBot filling the *team coordinator* role. This ultimately depends on which MidBot recognized the failure of the original team coordinator first.

The graphs in Figures 5.28 and 5.29 show the percentage of the environment covered and percentage of victims successfully identified, respectively, when the communication success rate is 20%. Similar to the results in the main experiment, where

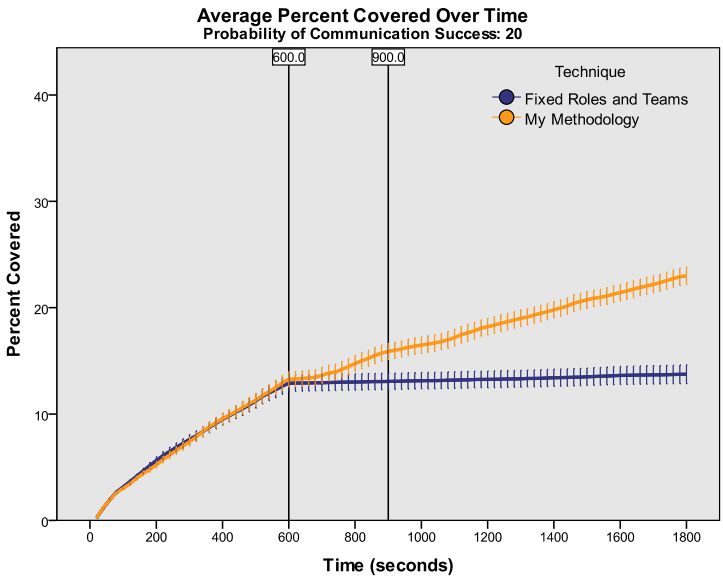


Figure 5.28: Leadership failure experiment, percentage of the environment covered for communication success rate 20%.

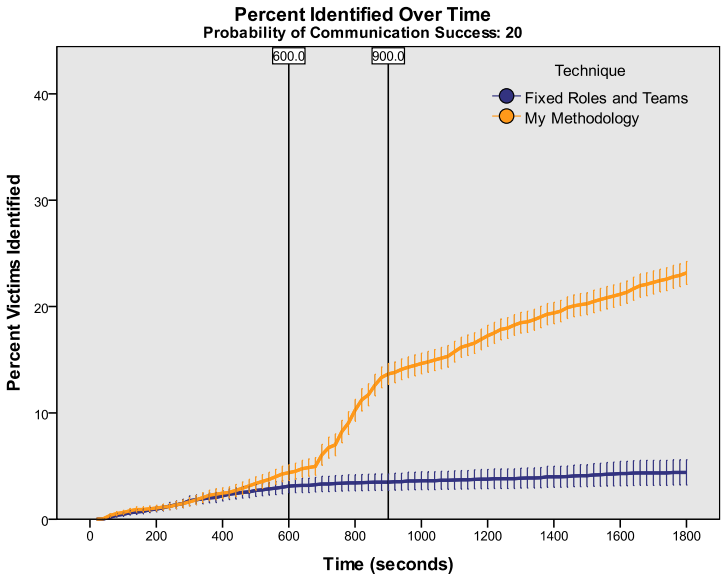


Figure 5.29: Leadership failure experiment, percentage of victims successfully identified for communication success rate 20%.

the communication success rate is 20%, the team is not able to coordinate its efforts well. Reviewing experimental runs reveals task assignment often fails, resulting in team members falling back to their *wander* idle tasks. As the team spreads out, team members move out of radio range of one another. Despite the fact the team coordinator role is assumed by a MidBot at 600 and 900 seconds, the MidBot filling that role is not able to perform any meaningful coordination of the team's efforts.

5.9 Analysis

Having discussed the results of the main experiment (Section 5.7) and the leadership failure experiment (Section 5.8), this section makes some observations about my methodology as a whole, identifying its strengths, and areas where it could be improved through future research.

When considering the reliability of communication in an environment, my results clearly show the need for a bare minimum level of communication, regardless of the technique used. A 20% communication success rate is insufficient to support effective coordination of teams, and results in robots often falling back to their idle work. Given a communication success rate of 60%, my methodology results in performance improvements, when compared to the baseline where roles and team membership (but not the allocation of tasks) are fixed. Reduced levels of communication success tend to result in robots becoming separated from their team more often, and my methodology provides these lost robots with the ability to form a new team of their own, or join another team they encounter. It would be interesting to perform a future experiment to determine the communication success rate necessary for an improvement in

performance to be observed.

My methodology shows significant utility in enabling a team to cope with the complete failure of critical team members carrying out leadership duties. By allowing a less capable robot to shift into the *team coordinator* role, a team can continue operating, despite the failure of a critical team member. Where members of a team cannot change their roles and team membership, the performance of a team is severely impacted by the failure of the team coordinator robot.

Although my methodology results in improvements over a baseline where roles and team membership (but not the allocation of tasks) is fixed and robots suffer from random temporary failures, my implementation does not recognize short duration robot failures well. This is due to the manner in which failed robots become forgotten by their teammates. Further, since each robot maintains a backlog of tasks to complete, a short duration failure of the team coordinator is not likely to impact a robot's ability to continue performing useful work. Once the team coordinator restores operation, it could continue assigning new work to robots. Future work is necessary to determine if my implementation could be made more responsive to short duration failures, and to investigate whether techniques could be applied to allow a team to adjust its response to short duration failures as the mission progresses. Simply forgetting about failed agents after a short period of time is not ideal, as agents experiencing legitimate delays are likely to be written off prematurely. This also means the structure of teams will fluctuate more frequently, as agents attempt to cope with the increased number of perceived failures. Future work is necessary to determine the impact of varying the memory duration. It would also be interesting to investigate the impact of varying

the size of the task backlog each robot maintains. I suspect using a smaller task backlog would more clearly show the improvements realized with my methodology during short term failures. It is possible maintaining a backlog of tasks is a sufficient mitigation to cope with short term failures of the team coordinator.

My example implementation includes two main task types: frontier exploration, and victim identification. The largest gains in performance over the baseline cases were seen with the victim identification tasks. This suggests tasks requiring very specialized skills, available on only a small subset of the robots benefit most from my methodology. This is further supported by the fact the improvements seen in the percentage of the environment covered over the baseline are smaller in comparison. The ability to explore is a capability all team members possess, providing a high level of redundancy for this skill. Future work could investigate the impact of specialization versus generalization with respect to my methodology.

5.10 Summary

Having reviewed my research questions, this chapter provided an overview of the experimental environments I used to evaluate my example implementation and methodology. An explanation of how these experimental environments were generated was provided, along with a description of the controls put in place to ensure consistency among the generated environments, and control over the experimental parameters. The base case conditions against which I compare my methodology were described, along with the experiments I performed to answer my research questions. Evaluation criteria were presented, providing a means of numerically describing the

performance of my methodology versus the baseline cases. Finally, the results from each experiment were discussed, and an analysis of the results seen was presented.

Chapter 6

Conclusion

6.1 Overview

This chapter begins by discussing the results obtained in Chapter 5, and how these results answer my main research questions (Section 6.2). A review of the main contributions of my thesis is presented (Section 6.3), and a discussion of future research directions for my work follows (Section 6.4.2).

6.2 Answers to Research Questions

Section 1.6 presented the research questions around which the work in this thesis is centered. This section reviews those questions, and discusses how they are answered by the results in Chapter 5.

1. **Can my framework provide teams operating in dynamic environments with the ability to adequately cope with changes in team structure**

and composition (i.e. due to loss and failure of team members, and encountering other teams and teammates in the environment)?

My methodology shows clear benefits when helping a team cope with team members getting lost due to unreliable communication and the difficult nature of the environment. Lost team members are able to either form their own team, or join another team they encounter. Where an agent has especially important skills (i.e. victim identification), my methodology helps ensure a lost robot is able to continue providing useful work towards the overall mission goal. Where critical members of the team suffer a failure, my methodology also shows a significant improvement over the baseline cases, allowing a less suited robot to take on the team coordinator role. Where replacement robots are available, my methodology shows a clear benefit in allowing the replacements to form new teams, and be integrated into existing teams.

2. **Can my framework help mitigate the negative affect of unreliable communication on coordination efforts between agents?**

   My main experiment results show that my methodology is able to help cope with the negative affect of unreliable communication on the coordination efforts between robots. With a communication success rate of 60%, my methodology was able to compensate for the coordination issues associated with agents becoming separated from their team, by allowing them to join another team, or form a new team in response. Although my methodology was hampered at the 20% communication success rate, future experimentation might show more robustness of my approach between the 20% and 60% communication success

rates.

3. **Is my framework able to cope with failure of a team's leadership structure?**

   The leadership failure experiment clearly demonstrated my methodology's ability to enable a team to continue operation, despite the failure of a robot filling the critical team coordinator role (Section 5.8). Where roles and team membership are fixed, the failure of the team coordinator resulted in the team ceasing to make further progress. Using my methodology, the team adjusted to the failure of the team coordinator, and was able to continue making progress despite the loss of the better suited robot.

6.3 Contributions

My research provides contributions in a number of key areas, including mobile robotics, distributed artificial intelligence, and multiagent systems. My major contributions include:

1. A methodology supporting the formation, and maintenance of teams of heterogeneous agents operating in dynamic and complex environments.

2. A methodology supporting the use of roles as a heuristic description of the types of work teams can be expected to complete in an environment, and supporting the effective assignment of tasks to team members filling these roles.

3. A means of compensating for the loss of team members, either due to equipment

failures or unreliable communication.

4. A framework to describe agents in terms of the capabilities required to carry out tasks, and to reason about the best suited agent to carry out these tasks.

5. A demonstration of the efficacy of using heterogeneous robots to complete a complex mission in a dynamic and complex domain.

6. Enhancements to the Stage simulator, promoting future research in mobile robotics where communication is limited in range and unreliable in nature.

7. An example implementation demonstrating my methodology in a repeatable manner, in randomly generated environments resembling a disaster environment.

8. A flexibly designed example implementation, which provides a test-bed in which future teamwork research can be conducted (Section 6.4).

My methodology demonstrates several advantages over approaches where the roles agents fill and team membership is fixed. Where no replacement agents are available, my methodology enables agents to change the role they occupy or their team membership, compensating for situations where they become separated from their team (Section 5.7.1). Further, teams using my methodology are able to restructure themselves to compensate for the failure of agents performing critical team leadership duties (Section 5.8). Where replacement agents are available, my methodology enables these agents to either form new teams of their own, or augment the capabilities of existing teams operating in the environment (Section 5.7.2).

6.4 Future Work

Although my methodology showed significant utility, my example implementation revealed a number of areas where improvements could be made, or other interesting research could be performed. The most obvious avenue of future work is an implementation in a physical environment using real robots. If a simulation revealed areas for improvement, the real world will no doubt reveal more, since no simulation can consider all the complexities the real world has to offer [Balch, 1998]. Section 6.4.1 begins by describing improvements which can be made to my implementation, in order to evaluate its performance in a real world environment. Section 6.4.2 discusses elements of my core methodology where improvements could be made, or other research performed.

6.4.1 Future Implementation Work

My example implementation made a number of simplifications in peripheral components to facilitate my thesis research (and implementation in a reasonable timeframe). Although these simplifications are reasonable within the context of a simulated environment, operating in the physical world would require complete, implemented solutions for all peripheral problems, as well as a physical environment to properly evaluate USAR research.

Since disasters occur at unpredictable times and involve conditions truly hazardous to robots and potentially even to human operators, a real disaster zone is not a suitable environment in which to evaluate my methodology next. As discussed in Section 2.2, the National Institute of Standards and Technology (NIST) has developed a series

of USAR test arenas which emulate the conditions found in a real world disaster zone [Jacoff et al., 2003]. Murphy et al. [2000b] evaluated the NIST test arenas, and found them to be a useful research tool, providing a step towards operation in a real disaster zone. One of the USAR test arena configurations would provide a useful real world environment in which an implementation of my methodology on real robots could be studied, and would provide a number of operational challenges not present in simulation.

As discussed in Section 4.3.1, my implementation makes use of the Stage simulator API to provide robots with perfect localization. Although this simplification allowed me to focus on the coordination aspects of my work, an implementation on real robots would need to use a more realistic localization technique. This could be accomplished by using one of the multi-robot localization approaches discussed in Section 2.3.3 (e.g. [Martinelli et al., 2005]). Moving away from perfect localization would necessitate a review of the mapping and map merging techniques used, as they would need to take into account the fact that localization in the real world cannot be considered to be perfect. Further, the occupancy grid map I used in my example implementation provides a 2.5 dimensional view of the world, where the real world is three dimensional. A different map representation, such as Wurm et al. [2011]'s OctoMap representation, might be suitable. The path planning algorithm in my implementation (Section 4.8.4) would also need to be modified (or replaced) to take into account the different map representation. Depending on the localization and mapping techniques employed, the computation required may necessitate the MinBot robots relying on more capable robots to build maps on their behalf, using communicated sensor data.

Although the wireless communication model, described in Section 4.9.1, provides a reasonable simulation of short range, unreliable communication, it simulates communication connectedness based only on the distance between robots. In the real world, the presence of obstacles blocking line of sight communication would serve to attenuate signals. Further, the message delivery provided by my simulated wireless communication does not consider the size of messages, or the time it would take to send these messages. Adapting my implementation for use on real robots would require the use of real wireless radios. As mentioned in Section 4.9.1, a suitable radio might use the IEEE 802.15.4 wireless standard, which is intended for low power, short range communication. Using real radios would require a review of the data communicated between robots, in order to optimize the quantity of data transmitted. Map sharing, for example, should ideally send only the relevant portions of a robot's map, and should break down larger maps into smaller pieces more suitable for transmission. Using real radios would provide the opportunity to develop appropriate communication protocols that take into account the challenges inherent with real wireless communication (e.g. error checking, channel access).

An implementation on real robots would also require an appropriate implementation of robot perception, replacing the abstracted *robot detector* (Section 4.3.3) used in my example implementation. As explained in Section 4.9.3, a solution might be to outfit robots with coded fiducial markers, similar to those used by Howard et al. [2006a] in their work, so robots could visually locate and identify one another in the environment. This would make the encounter process more difficult, as robots could not precisely determine their relative locations to establish a translation between their

respective coordinate systems.

Similarly, a real world implementation of the abstracted *victim detector* (Section 4.3.2) used in my example implementation would be necessary. According to Jacoff et al. [2003], the NIST test arenas include simulated victims which can be detected visually. This would require a vision processing component to recognize the shapes and colors normally associated with victims.

Assuming physical robots that are similar to the models used in my simulated implementation are used, there is an opportunity for more diversity in the skill sets offered by each robot type. A tracked MaxBot, for example, might be modified so that it could transport the MidBot and MinBot robots to areas of the environment which their limited mobility otherwise precludes them from accessing.

Finally, using real robots would necessitate operating under the constraints of the computational hardware provided by the robot types employed. Where a MaxBot could be expected to use a full powered computer (e.g. a laptop computer), the MidBot and MinBots would likely use smaller, embedded computing platforms. These platforms introduce memory and computational constraints not directly present in my simulation (my work did assume weaker robot types would have fewer abilities in these areas, but only in very general ways), and would require algorithms to be adjusted for the different computing hardware. This would result in a greater degree of heterogeneity in the robots' computational capabilities than what was possible to emulate given the granularity of my simulation.

In addition to improvements required to adapt my implementation for use on real robots, improvements are also possible to my control software, making it better able

to perform the USAR mission.

Kratzke et al. [2010] studied how real-world search and rescue operations are planned and executed. It would be interesting to study ways in which their findings could be used to help a team better coordinate its team members. Further, it would be interesting to investigate the addition of a higher level planning mechanism to provide general coordination to the teams operating in the environment. A human operator could also be used to help coordinate the overall efforts of the team. A blended approach, such as the one developed by Wegner and Anderson [2004], could be used to allow the expertise of a human operator to be blended with those of an automated planning mechanism.

When replacement robots begin operation, my implementation uses the *find team* task type (Section 4.5.2.3) to guide these robots along the bearing in which they were introduced into the environment, with the intention of encouraging them to progress deeper into the environment. This behaviour assumes the teams already operating in the environment will have explored the perimeter of the environment first, and that a replacement robot is more likely to encounter other robots deeper in the environment. An alternative possibility would be for the replacement robot to use the wireless signal strengths of other robots it overhears in the environment to attempt to locate existing teams operating in the environment. Wireless signal strength has been used previously (e.g. [Ocana et al., 2005]) to aid robot localization, and similar techniques could be used to help guide a replacement robot towards stronger radio signals, with the goal of encouraging it toward other robots.

In addition to guiding replacement robots toward teams, members of an existing

team could use the signal strength of robots they overhear to help react to a loss of communication with their team (e.g. Ulam and Arkin [2004] studied behaviours which help a team recover from the loss of communication between robots). In such a scenario, a robot which has become separated from its team could attempt to regain communication with its team, by homing in on the radio signals of its teammates. Alternatively, the robot could attempt to home in on the radio signals of any other robot it overhears, increasing the possibility of the robot switching teams, resulting in a transfer of knowledge from one team to another.

Areas for improvement exist within the autonomous control module in my example implementation (Section 4.6). Robots use the *detect lost* perceptual schema to identify scenarios where they have traveled too great a distance to reach a destination (Section 4.6.1.8). Ideally, a robot should attempt to automatically adjust the maximum distance it travels to reach a destination based on its experiences so far. This could help cut down wasted effort resulting from traveling to an unreachable destination. In such a situation, the robot could either ask its team coordinator for a new path plan (where the team coordinator has a planner module) to reach the destination from its new location. Alternatively, the robot could indicate to the team coordinator that the destination could not be reached, and request the task be assigned to another robot for completion.

6.4.2 Future Methodology Work

This section discusses potential areas of future work related to my methodology. Although these are discussed in relation to my example implementation, they

represent general improvements to my methodology which would be useful in other domains as well.

Although my methodology assumes agents can suffer from failures at any point, it does not incorporate any form of comprehensive failure model. The reliability of robotic platforms has been studied previously (e.g. [Carlson and Murphy, 2003]); knowledge of the expected reliability of robots and their components could be programmed into robots at design time. This information could then be used as part of the task assignment process (e.g. Stancliff et al. [2009] found exploration performance could be improved by anticipating failures based on a robot's reliability), ensuring critical tasks as carried out by more reliable agents, for example. Further, agents could actively monitor their own performance (or the team coordinator could monitor the performance of others) in order to adjust knowledge pertaining to their reliability, based on actual experiences. Further, when agents detect failures of their components, they could update their self-knowledge, helping to ensure they do not take on tasks which they may no longer be suited to carry out.

My example implementation uses the *expendability* robot attribute (Section 4.5.1.1) to provide a simple heuristic description of the relative expendability of one robot in relation to another. This value is fixed at design time, and is the same for all robots of the same type. Ideally, the expendability should be adjusted dynamically based on the current team composition, and the knowledge agents possess. A lone MidBot, for example, would be less expendable than if there was another MidBot on the team. Similarly, a value could be assigned to the knowledge a robot possesses, and the expendability adjusted accordingly. A MinBot which has accumulated informa-

tion about a large area of the environment, for example, could be considered as a valuable backup of the team's knowledge, and might be an advantageous robot to send to another team to ensure the mission knowledge is spread further.

My main experiment revealed short term robot failures were not likely to be detected by members of the team (Section 5.7). The fact each robot maintains a backlog of tasks requiring completion meant short-term failures of the team coordinator did not have a large impact on the team's performance, as most team members had a large enough backlog of work to last until the team coordinator returned to service. It would be useful to study this phenomenon further, to determine whether actively detecting failures sooner is beneficial, and to what degree the backlog of tasks provides robots with the ability to continue operating effectively despite the failure of a team coordinator. Further, it might be possible for robots to actively adjust the threshold for forgetting teammates as the mission progresses, possibly using some type of machine learning technique.

Agents could also use learning techniques to adjust the desired team definition (Section 3.4.4) as the mission progresses. For example, a team could recognize that it frequently requires a skill set which is not available in abundance due to the current desired team definition. In such a scenario, it might be advantageous to adjust the desired team definition to accommodate more agents with this skill set. Conversely, a team could recognize its desired team definition causes is to retain more agents possessing a particular skill set than are required. The desired team definition could be adjusted down, providing the opportunity for superfluous agents to switch to another team where they could be better used. Other factors could impact the decision to

adjust the desired team definition. A team could, for example, adjust the desired number of agents filling a role if it determines that a skill set is important enough to warrant an increased level of redundancy.

The team merge and redistribution algorithm (Section 3.7) in my methodology is handled through representative agents who encounter one another in the environment. Although I make the argument for performing the team merge and redistribution using these intermediary agents, it would be useful to study the impact of other variations of this process. An example is having the team coordinator agents move to the point of encounter in order to complete the merge and redistribution operation directly, rather than through the representative agents. Another possibility would be to bring a team coordinator to the encounter point, only if is nearby. Experimentation would be necessary to determine at what point it is advantageous to incur the penalty of moving the team coordinator to the encounter point.

Aside from the knowledge sharing which occurs when teams encounter one another and complete the team merge and redistribution operation, my methodology does not explicitly attempt to coordinate efforts between teams operating in the environment. In my example implementation, for example, the encountering teams could attempt to negotiate general areas of responsibility within the environment in which each team would attempt to constrain its efforts.

The team merge and redistribution algorithm (Section 3.7) attempts to maintain team stability by reducing the number of agents which change between teams. It would be useful to study the impact of team stability on team performance further, to determine if there is a utility to attempting to keep larger teams together, or

whether it is advantageous to encourage teams to split into smaller teams more often than my methodology encourages them to now. Further, it would be interesting to determine whether there is a utility in having a team detect a deficiency in its team structure, and purposely seek out an encounter with another team, rather than relying on chance encounters.

Finally, my methodology does not actively track the progress of tasks after assignment. It would be useful to investigate the utility of tracking the progress of tasks as they are carried out by agents. This would help the team coordinator ensure critical tasks are completed in a timely manner, and could also serve as another means of detecting the failure of task assignees.

6.5 Conclusion

It is my hope that the success of this research will encourage future research into the issues involved with the effective coordination of agents operating in difficult and challenging domains. Robots operating in dynamic and complex domains, such as disaster zones, must cope with the difficult conditions inherent with these environments, and cannot make assumptions about the structure of teams, or the continuing availability of members of a team. This makes investigation into approaches which allow a team to reassign responsibilities to compensate even more important. The high cost inherent with robotic systems also makes it desirable to develop techniques allowing heterogeneous robots to cooperate effectively in these environments. Effective coordination of heterogeneous robots is beneficial as it allows a team to augment the capabilities provided by a larger number of inexpensive general purpose robots

with the capabilities provided by a smaller number of specialized robots.

This research has demonstrated the utility of describing the composition of a team in terms of roles describing the types of work normally expected of its members, and using this as a means of reasoning about the changes which can be implemented to the structure of these teams in response to the loss or failure of team members, or the discovery of new potential team members. This research has also identified a number of interesting areas in which future work could productively be carried out (Section 6.4), based on the techniques which I have developed.

Appendix A

Experimental Environments

Figures A.1, A.2, and A.3 show the three environment configurations in which I ran my experiments (Chapter 5).

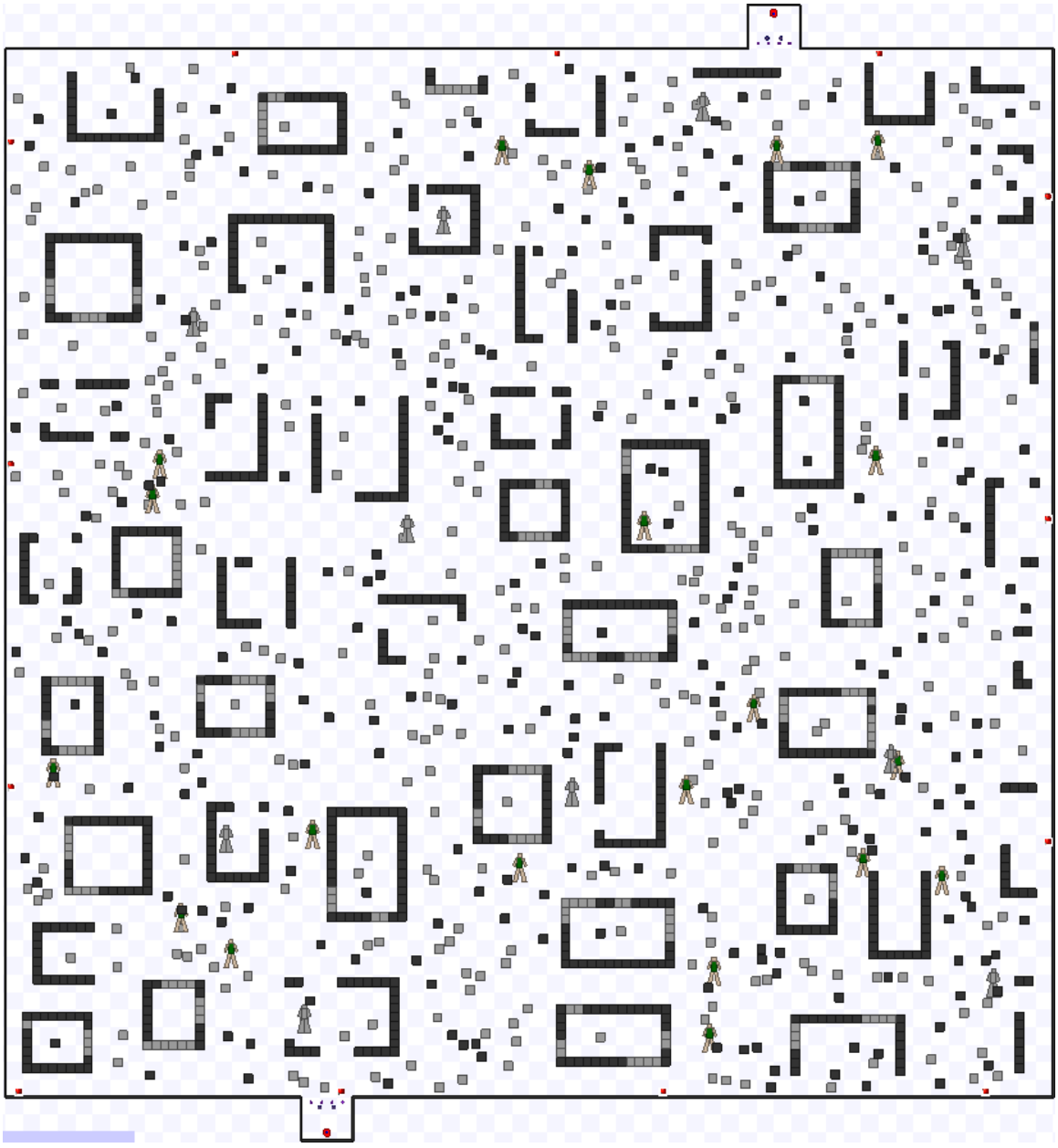


Figure A.1: Experimental environment configuration 1.

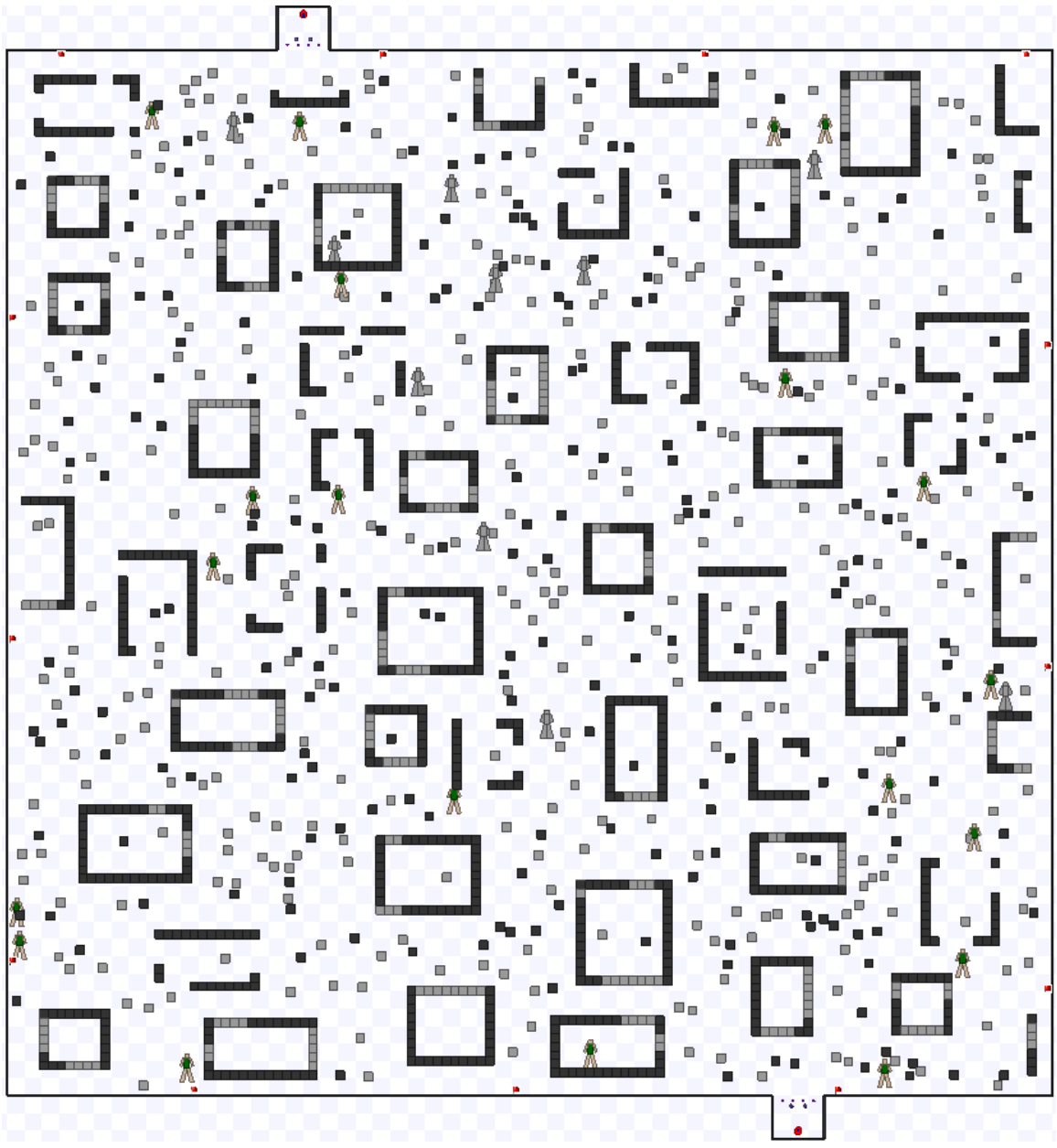


Figure A.2: Experimental environment configuration 2.

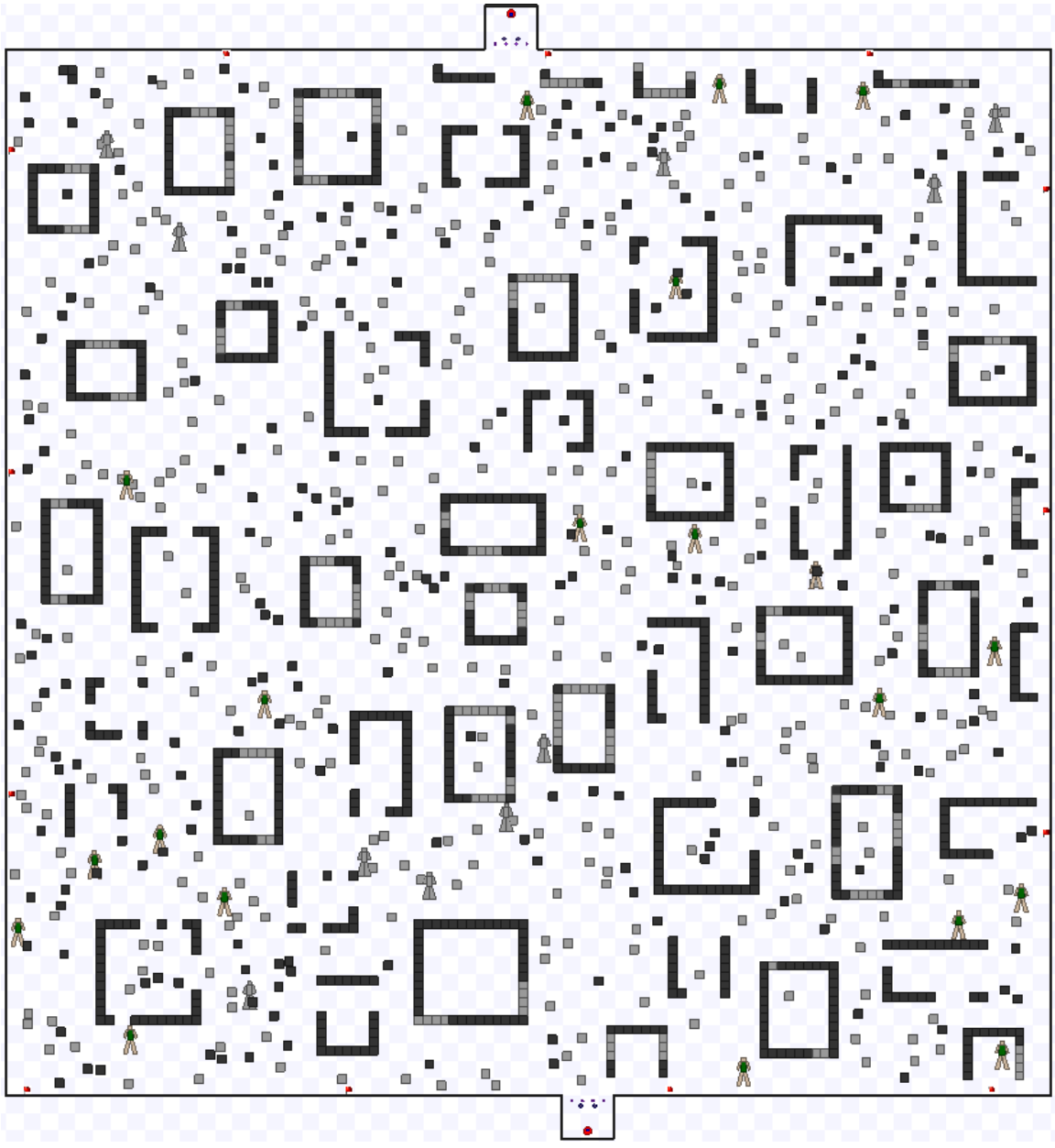


Figure A.3: Experimental environment configuration 3.

Appendix B

Experiment Results

B.1 Main Experiment

Replacements	**Comm Success Rate**	**Probability of Failure**	**Percent Environment Covered**	**Percent Victims Identified**
No	20	None	Fig. B.1	Fig. B.2
	20	Moderate	Fig. 5.5	Fig. 5.6
	20	Major	Fig. B.3	Fig. B.4
	60	None	Fig. 5.11	Fig. 5.12
	60	Moderate	Fig. 5.7	Fig. 5.8
	60	Major	Fig. 5.13	Fig. 5.14
	100	None	Fig. B.5	Fig. B.6
	100	Moderate	Fig. 5.9	Fig. 5.10
	100	Major	Fig. B.7	Fig. B.8
Yes	20	None	Fig. B.9	Fig. B.10
	20	Moderate	Fig. 5.16	Fig. 5.17
	20	Major	Fig. B.11	Fig. B.12
	60	None	Fig. 5.22	Fig. 5.23
	60	Moderate	Fig. 5.18	Fig. 5.19
	60	Major	Fig. 5.24	Fig. 5.25
	100	None	Fig. B.13	Fig. B.14
	100	Moderate	Fig. 5.20	Fig. 5.21
	100	Major	Fig. B.15	Fig. B.16

Table B.1: Main experiment results cross reference.

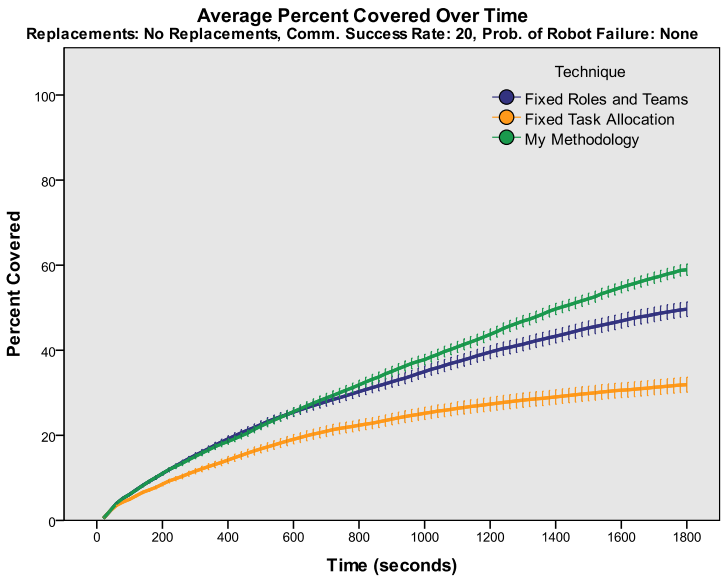


Figure B.1: Main experiment, coverage over time where: no replacements available, comm. success rate = 20%, and prob. robot failure = none.

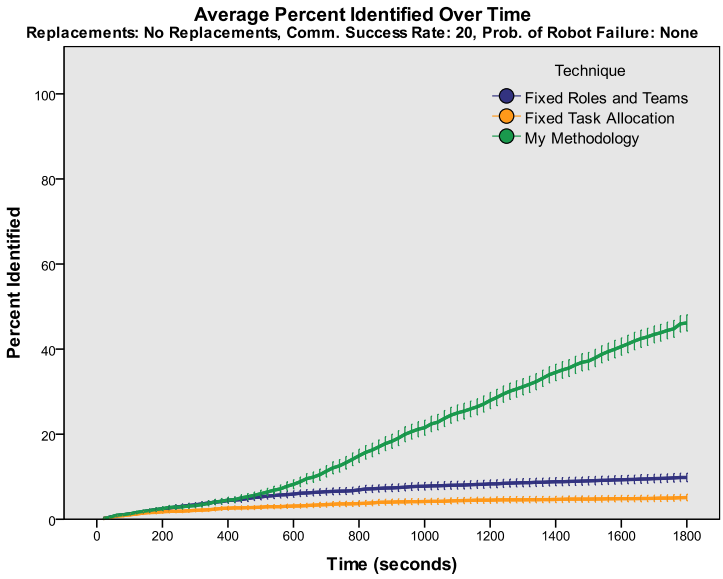


Figure B.2: Main experiment, victims identified over time where: no replacements available, comm. success rate = 20%, and prob. robot failure = none.

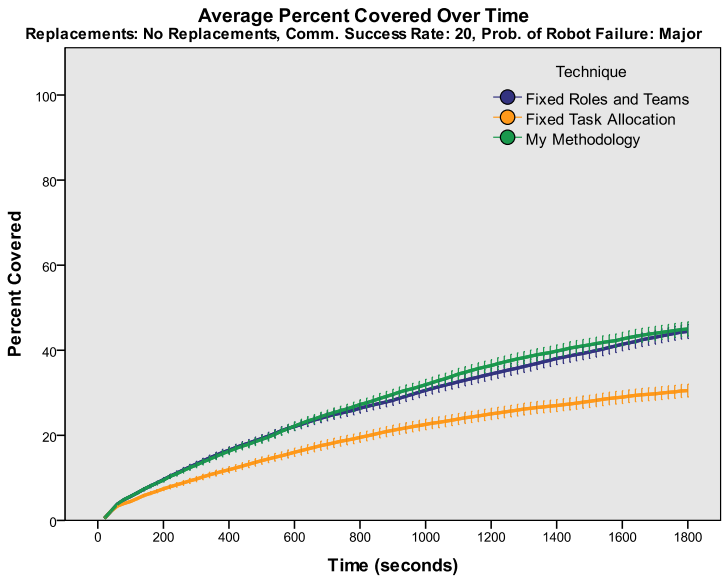


Figure B.3: Main experiment, coverage over time where: no replacements available, comm. success rate = 20%, and prob. robot failure = major.

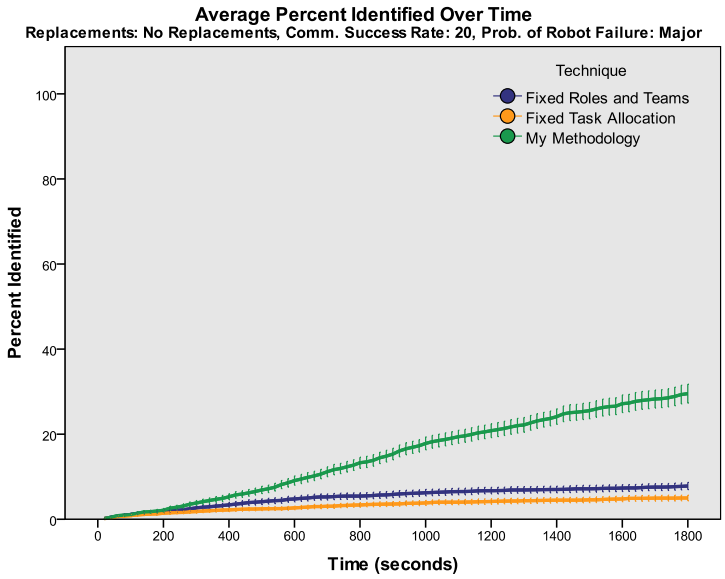


Figure B.4: Main experiment, victims identified over time where: no replacements available, comm. success rate = 20%, and prob. robot failure = major.

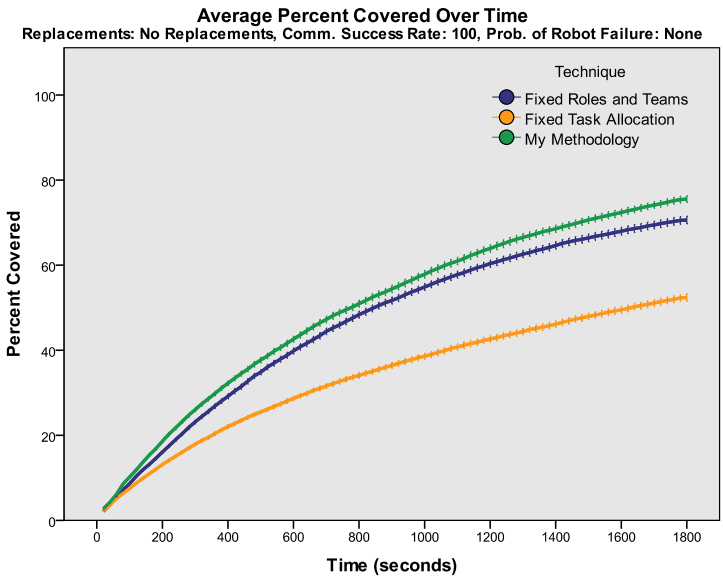


Figure B.5: Main experiment, coverage over time where: no replacements available, comm. success rate = 100%, and prob. robot failure = none.

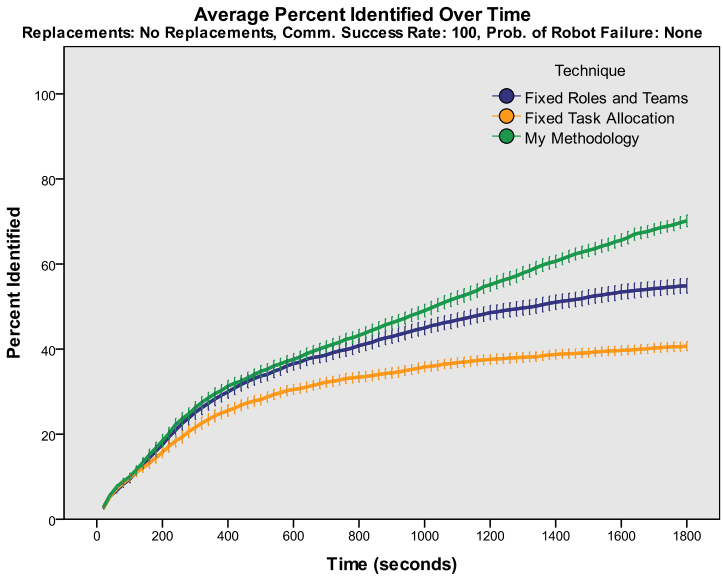


Figure B.6: Main experiment, victims identified over time where: no replacements available, comm. success rate = 100%, and prob. robot failure = none.

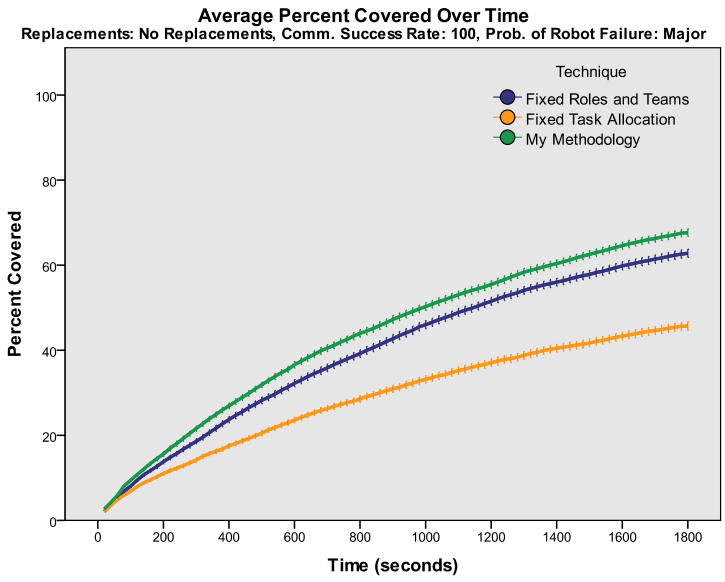


Figure B.7: Main experiment, coverage over time where: no replacements available, comm. success rate = 100%, and prob. robot failure = major.

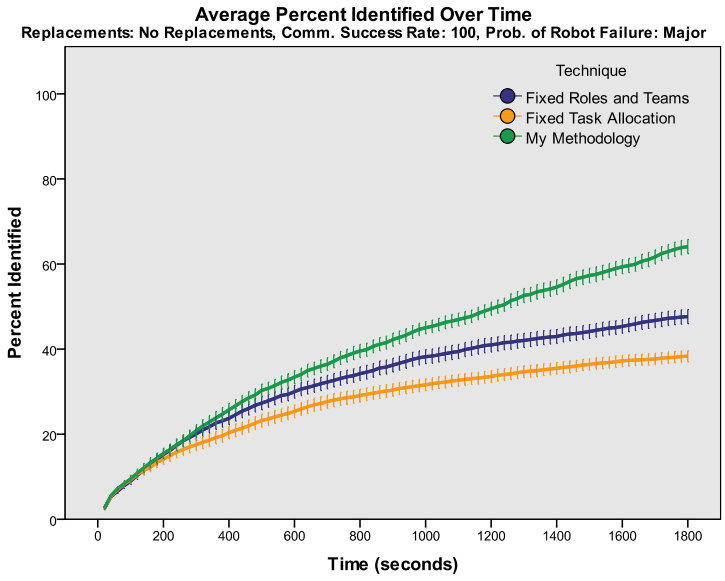


Figure B.8: Main experiment, victims identified over time where: no replacements available, comm. success rate = 100%, and prob. robot failure = major.

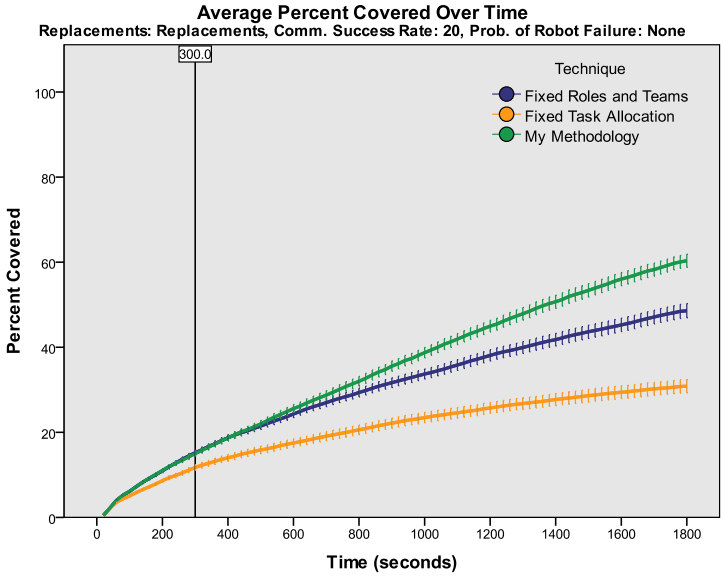


Figure B.9: Main experiment, coverage over time where: replacements available, comm. success rate = 20%, and prob. robot failure = none.

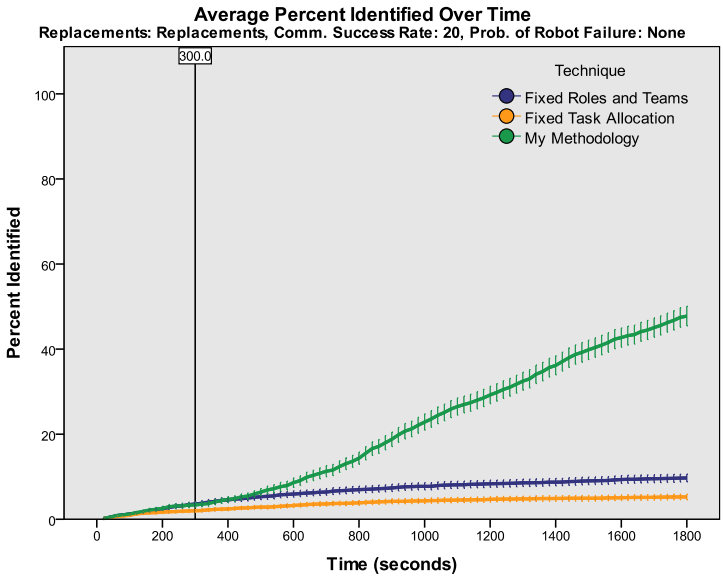


Figure B.10: Main experiment, victims identified over time where: replacements available, comm. success rate = 20%, and prob. robot failure = none.

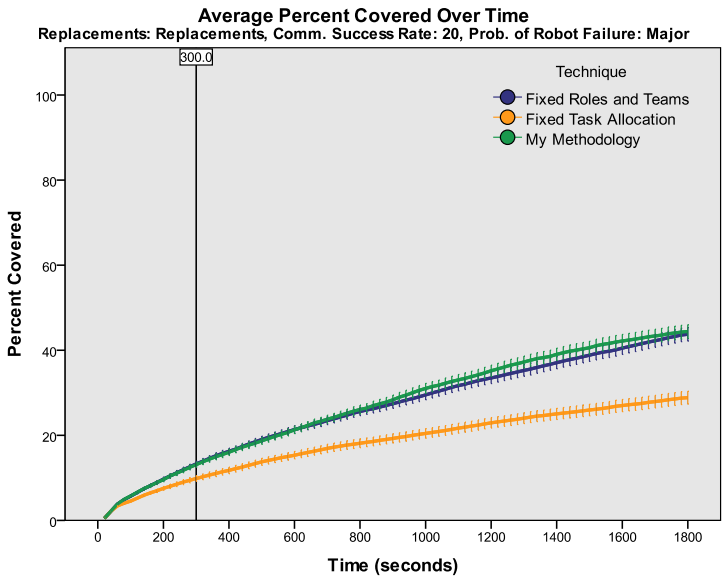


Figure B.11: Main experiment, coverage over time where: replacements available, comm. success rate = 20%, and prob. robot failure = major.

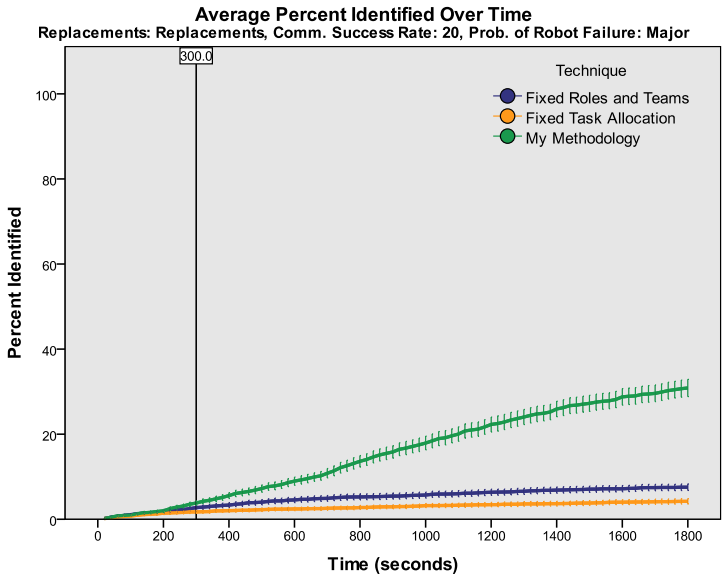


Figure B.12: Main experiment, victims identified over time where: replacements available, comm. success rate = 20%, and prob. robot failure = major.

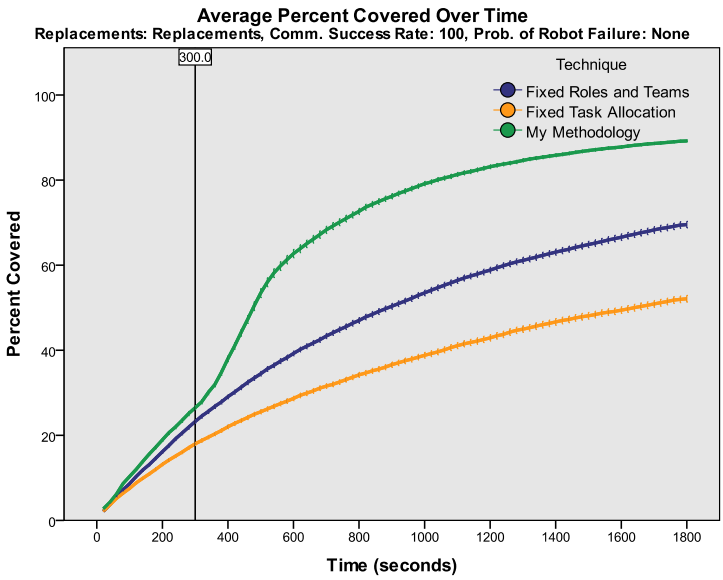


Figure B.13: Main experiment, coverage over time where: replacements available, comm. success rate = 100%, and prob. robot failure = none.

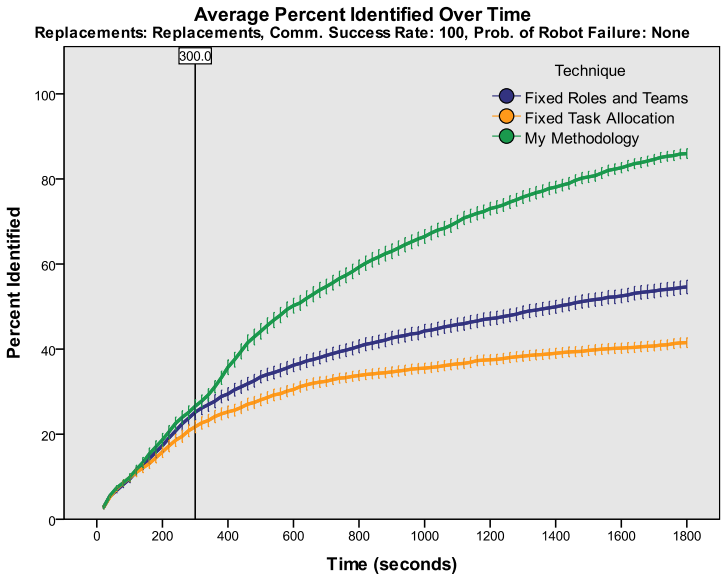


Figure B.14: Main experiment, victims identified over time where: replacements available, comm. success rate = 100%, and prob. robot failure = none.

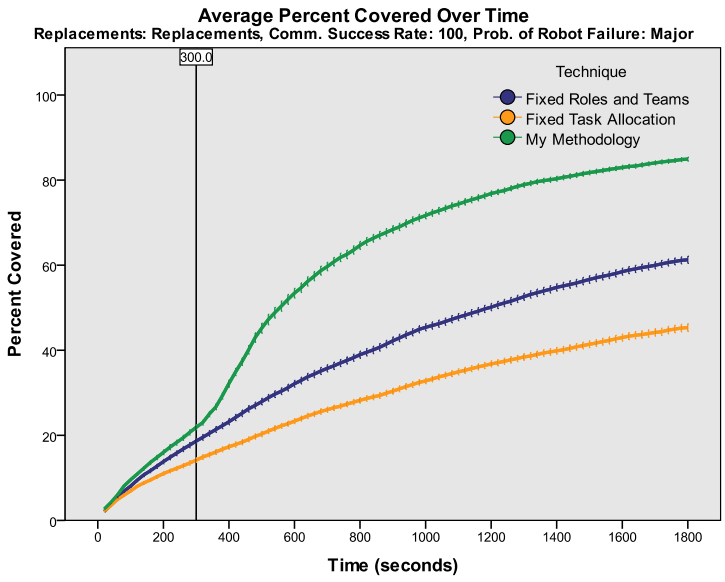


Figure B.15: Main experiment, coverage over time where: replacements available, comm. success rate = 100%, and prob. robot failure = major.

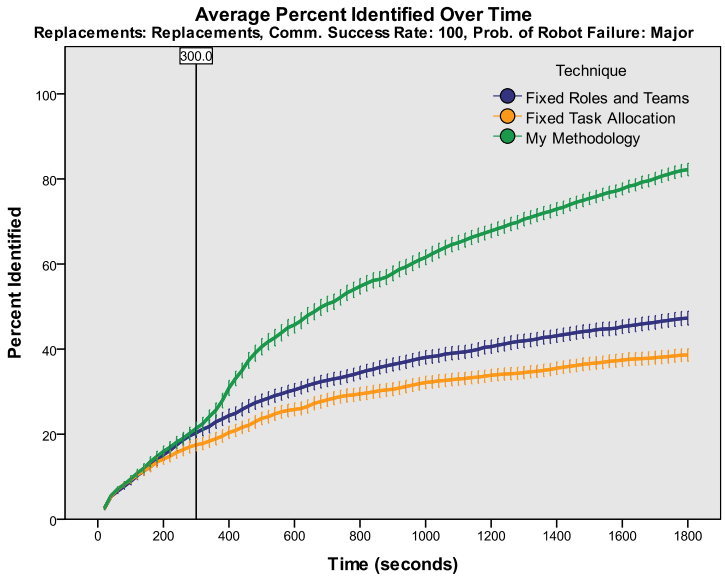


Figure B.16: Main experiment, victims identified over time where: replacements available, comm. success rate = 100%, and prob. robot failure = major.

B.2 Leadership Failure Experiment

Comm Success Rate	Percent Environment Covered	Percent Victims Identified
20	Fig. 5.28	Fig. 5.29
60	Fig. 5.26 (p.254)	Fig. 5.27 (p.255)
100	Fig. B.18	Fig. B.17

Table B.2: Leadership failure experiment results cross reference.

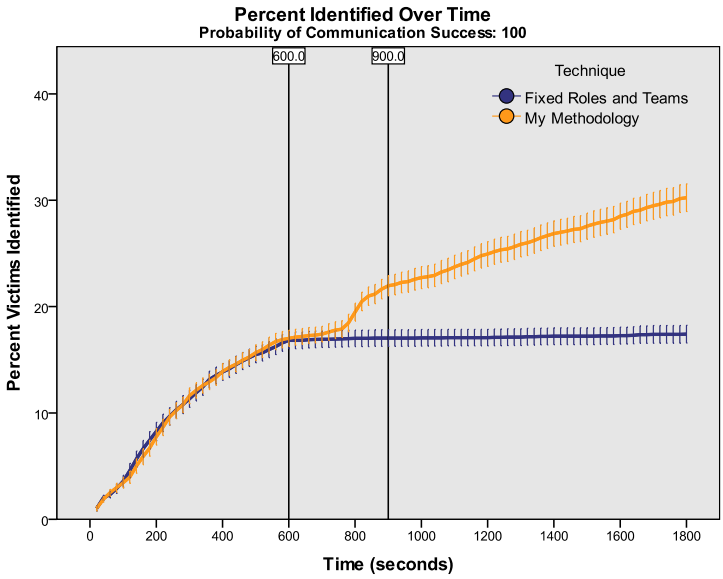


Figure B.17: Leadership failure experiment, percent victims identified for communication success rate 100%.

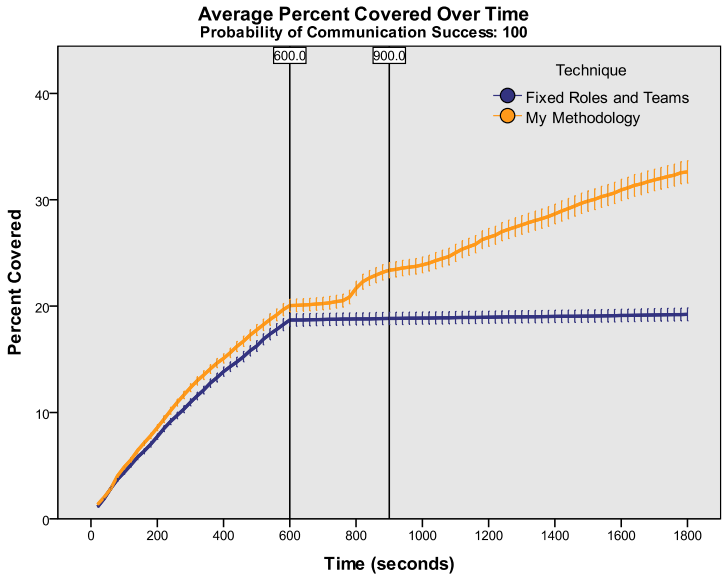


Figure B.18: Leadership failure experiment, percent environment covered for communication success rate 100%.

Bibliography

J. Anderson and J. Baltes. An agent-based approach to introductory robotics using robotic soccer. *Intl. Journal of Robotics and Automation*, 21(2), February 2006.

M. Anderson and N. Papanikolopoulos. Improving multirobot, cooperative search via local target queues. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2590–2595, November 2007. doi: 10.1109/IROS.2007.4399338.

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

L. Andersson and J. Nygards. C-SAM: Multi-robot SLAM using square root information smoothing. In *Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2798–2805, Pasadena, CA, USA, 2008. doi: 10.1109/ROBOT.2008.4543634.

R. C. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 264–271, Raleigh, NC, USA, 1987. IEEE.

R. C. Arkin and T. Balch. Aura: principles and practice in review. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2–3):175–189, 1997. doi: doi: 10.1080/095281397147068.

T. Balch. Robots move: Position paper on simulation. In *Working Notes of AAAI Spring Symposium*, 1998. URL `http://www.cs.cmu.edu/~trb/papers/aaai_spring_position/`.

P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications*, 30(7):1655–1695, 2007. ISSN 0140-3664. doi: 10.1016/j.comcom.2006.12.020.

N. Boonpinon and A. Sudsang. Constrained coverage for heterogeneous multi-robot team. In *Proc. of IEEE Intl. Conf. on Robotics and Biomimetics (ROBIO)*, pages 799–804, December 2007. doi: 10.1109/ROBIO.2007.4522265.

J. Borenstein and Y. Koren. Histogramic in-motion mapping for mobile robot obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 7(4):535–539, August 1991. ISSN 1042-296X. doi: 10.1109/70.86083.

M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3):349–355, 1988. ISSN 1467-8640. doi: 10.1111/j.1467-8640.1988.tb00284.x.

J. Bresenham. Pixel-processing fundamentals. *IEEE Computer Graphics and Applications*, 16(1):74–82, Jan 1996. ISSN 0272-1716. doi: 10.1109/38.481626.

C. H. Brooks and E. H. Durfee. Congregation formation in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 7:145–170, 2003. ISSN 1387-2532.

J. Bruce, M. Bowling, B. Browning, and M. Veloso. Multi-robot team response to a multi-robot opponent team. In *Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 2, pages 2281–2286, September 2003. doi: 10.1109/ROBOT.2003.1241933.

W. Burgard, D. Fox, and S. Thrun. Markov localization for mobile robots in dynamic environments. *CoRR*, abs/1106.0222, 2011.

S. Burion. Human detection for robotic urban search and rescue. Technical report, Swiss Federal Institute of Technology, 2004. Swiss Federal Institute of Technology.

J. Carlson and R. Murphy. Reliability analysis of mobile robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation ICRA*, volume 1, pages 274–281, September 2003. doi: 10.1109/ROBOT.2003.1241608.

D. Carnegie. A three-tier hierarchical robotic system for urban search and rescue applications. In *Proc. of IEEE Intl. Workshop on Safety, Security and Rescue Robotics (SSRR)*, pages 1–6, September 2007. doi: 10.1109/SSRR.2007.4381268.

S. Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, 10 2008. doi: 10.1007/s10514-008-9097-4.

J. Casper and R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Sys-*

tems, Man, and Cybernetics, Part B: Cybernetics, 33(3):367–385, June 2003. ISSN 1083-4419. doi: 10.1109/TSMCB.2003.811794.

K. Cheng and P. Dasgupta. Multi-agent coalition formation for distributed area coverage: Analysis and evaluation. In *Proc. of IEEE Intl. Conf. on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 3, pages 334–337, September 2010. doi: 10.1109/WI-IAT.2010.277.

P. R. Cohen and C. R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212, 1979. ISSN 1551-6709. doi: 10.1207/s15516709cog0303_1.

P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3):32–48, 1989.

Computing Research Association. CRA grand research challenges in information systems final report. In *CRA Conf. on Grand Research Challenges in Computer Science and Engineering*, page 9, Warrenton, Virginia, USA, June 2002. URL `http://www.cra.org/reports/gc.systems.pdf`.

R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, 1983. ISSN 0004-3702. doi: 10.1016/0004-3702(83)90015-2.

A. Diosi and L. Kleeman. Advanced sonar and laser range finder fusion for simultaneous localization and mapping. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent*

Robots and Systems (IROS), volume 2, pages 1854–1859, September 2004. doi: 10.1109/IROS.2004.1389667.

G. Dissanayake, J. Paxman, J. V. Miro, O. Thane, and H. Thi. Robotics for urban search and rescue. In *Proc. of 1st Intl. Conf. on Industrial and Information Systems*, pages 294–298, August 2006. doi: 10.1109/ICIIS.2006.365740.

M. Dorigo, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. Christensen, A. Decugniire, G. Di Caro, F. Ducatelle, E. Ferrante, A. Frster, J. Martinez Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. Montes de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rtornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Sttzle, V. Trianni, E. Tuci, T. A.E., and V. F. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. Technical report, Institut de Recherches Interdisciplinaires et de Dveloppements en Intelligence Artificielle (IRIDIA), Universit Libre de Bruxelles, 2011.

P. S. Dutta and S. Sen. Forming stable partnerships. *Cognitive Systems Research*, 4(3):211–221, 2003. ISSN 1389-0417. doi: DOI:10.1016/S1389-0417(03)00005-6. Cognitive Agents and Multiagent Interaction.

M. Eghbali and M. Sharbafi. Multi agent routing to multi targets via ant colony. In *Proc of the 2nd Intl. Conf on Computer and Automation Engineering (ICCAE)*, volume 1, pages 587–591, February 2010. doi: 10.1109/ICCAE.2010.5451346.

A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, June 1989. ISSN 0018-9162. doi: 10.1109/2.30720.

O. Etzioni and R. Segal. Softbots as testbeds for machine learning. In *Working Notes of the AAAI Spring Symposium on Knowledge Assimilation*, pages 1–8, Menlo Park, CA, 1992. AAAI Press.

FEMA. *Urban Search and Rescue Response System In Federal Disaster Operations*. Washington, DC, January 2000.

FIPA. FIPA contract net interaction protocol & specification, December 2002. URL `http://www.fipa.org/specs/fipa00029/SC00029H.pdf`.

D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8:325–344, 2000. ISSN 0929-5593. doi: 10.1023/A:1008937911390.

M. Gauthier and J. Anderson. Peer instruction for a teleautonomous USAR system. In *Proc. of the 3rd Intl. Conf. on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS)*, Singapore, December 2005.

J. George, P. Sujit, J. Sousa, and F. Pereira. Coalition formation with communication ranges and moving targets. In *Proc. of American Control Conference (ACC)*, pages 1605–1610, July 2010.

B. Gerkey and M. Mataric. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, October 2002. ISSN 1042-296X. doi: 10.1109/TRA.2002.803462.

B. Gerkey, R. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-

robot and distributed sensor systems. In *Proc. of the 11th Intl. Conf. on Advanced Robotics*, pages 317–323, Coimbra, Portugal, June 2003.

L. Giannetti and P. Valigi. Collaboration among members of a team: a heuristic strategy for multi-robot exploration. In *Proc. of 14th Mediterranean Conf. on Control and Automation (MED)*, pages 1–6, June 2006. doi: 10.1109/MED.2006.328718.

T. Goedemé, T. Tuytelaars, and L. Gool. Visual topological map building in self-similar environments. *Informatics in Control Automation and Robotics*, pages 195–205, 2008.

C. Guanghui, H. Nakamoto, N. Matsuhira, and I. Hagiwara. Effective application of monte carlo localization for service robot. In *Proc. of Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1914–1919, San Diego, CA, USA, October 2007. doi: 10.1109/IROS.2007.4399409.

J.-S. Gutmann, M. Fukuchi, and M. Fujita. A floor and obstacle height map for 3d navigation of a humanoid robot. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1066–1071, April 2005. doi: 10.1109/ROBOT.2005.1570257.

A. Howard, L. Parker, and G. Sukhatme. Experiments with a large heterogeneous mobile robot team: exploration, mapping, deployment and detection. *Intl. Journal of Robotics Research*, 25(5–6):431–447, May 2006a. ISSN 0278-3649. doi: 10.1177/0278364906065378.

A. Howard, G. Sukhatme, and M. Mataric. Multirobot simultaneous localization and mapping using manifold representations. *Proc. of the IEEE*, 94(7):1360–1369, July 2006b. ISSN 0018-9219. doi: 10.1109/JPROC.2006.876922.

ITU-R. Recommendations, propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 900mhz to 100ghz. Technical report, International Telecommunication Union (ITU), Geneva, 2003.

A. Jacoff, E. Messina, B. Weiss, S. Tadokoro, and Y. Nakagawa. Test arenas and performance metrics for urban search and rescue robots. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 4, pages 3396–3403, October 2003. doi: 10.1109/IROS.2003.1249681.

J. Kiener and O. von Stryk. Cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 959–964, November 2007. doi: 10.1109/IROS.2007.4399291.

T. Kratzke, L. Stone, and J. Frost. Search and rescue optimal planning system. In *Proc. of the 13th Conf. on Information Fusion (FUSION)*, pages 1–8, July 2010.

N. Lau, L. Lopes, G. Corrente, and N. Filipe. Multi-robot team coordination through roles, positionings and coordinated procedures. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 5841–5848, October 2009. doi: 10.1109/IROS.2009.5354286.

F. Legras and C. Tessier. *Advances in Agent Communication*, chapter LOTTO: Group Formation by Overhearing in Large Teams, page 1956. Springer Berlin, 2004. doi: 10.1007/978-3-540-24608-4_15.

T. Li, C.-Y. Chen, Y.-C. Yeh, C.-C. Yang, H.-K. Huang, H.-L. Hung, C.-H. Chu, S.-H. Hsu, D.-Y. Huang, B.-R. Tsai, M.-C. Gau, and R.-J. Jang. An autonomous surveillance and security robot team. In *Proc. of IEEE Workshop on Advanced Robotics and Its Social Impacts (ARSO)*, pages 1–6, December 2007. doi: 10.1109/ARSO.2007.4531425.

X. Ma, F. Meng, Y. Li, W. Chen, and Y. Xi. Multi-agent-based auctions for multi-robot exploration. In *The Sixth World Congress on Intelligent Control and Automation*, volume 2, pages 9262–9266, June 2006. doi: 10.1109/WCICA.2006.1713793.

A. Martinelli, F. Pont, and R. Siegwart. Multi-robot localization using relative observations. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2797–2802, April 2005. doi: 10.1109/ROBOT.2005.1570537.

M. Mataric, M. Nilsson, and K. Simsarin. Cooperative multi-robot box-pushing. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 556–561, August 1995. doi: 10.1109/IROS.1995.525940.

V. Matelln and D. Borrajo. ABC2 an agenda based multi-agent model for robots control and cooperation. *Journal of Intelligent & Robotic Systems*, 32:93–114, 2001. ISSN 0921-0296. 10.1023/A:1012009429991.

C. McMillen and M. Veloso. *Distributed Autonomous Robotic Systems 7*, chapter Dis-

tributed, Play-Based Role Assignment for Robot Teams in Dynamic Environments, pages 145–154. Springer, 2006. doi: 10.1007/4-431-35881-1_15.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. of the Sixteenth Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

R. Murai, K. Ito, and K. Nakamichi. Proposal of a snake-like rescue robot designed for ease of use. In *Proc. of IEEE Conf. on Industrial Electronics, Control and Instrumentation (IECON)*, pages 1662–1667, November 2008. doi: 10.1109/IECON.2008.4758203.

R. Murphy, J. Casper, J. Hyams, M. Micire, and B. Minten. Mobility and sensing demands in USAR. In *Proc. of IEEE Conf. on Industrial Electronics, Control and Instrumentation (IECON)*, volume 1, pages 138–142, Nagoya, Japan, 2000a. doi: 10.1109/IECON.2000.973139.

R. Murphy, J. Casper, M. Micire, and J. Hyams. Assessment of the NIST standard test bed for urban search and rescue. Technical Report WS-00-09, Association for the Advancement of Artificial Intelligence (AAAI), 2000b.

R. R. Murphy. *Introduction to AI Robotics*, chapter 4, 11, pages 105–152. The MIT Press, Cambridge, MA, USA, 2000.

M. Ocana, L. Bergasa, M. Sotelo, J. Nuevo, and R. Flores. Indoor robot localization system using wifi signal measure and minimizing calibration effort. In *Proc. of the*

IEEE Intl. Symposium on Industrial Electronics (ISIE), volume 4, pages 1545–1550, June 2005. doi: 10.1109/ISIE.2005.1529162.

J. Odell, H. Van Dyke Parunak, and M. Fleischer. The role of roles in designing effective agent organizations. In A. Garcia, C. Lucena, F. Zambonelli, A. Omicini, and J. Castro, editors, *Software Engineering for Large-Scale Multi-Agent Systems*, volume 2603 of *Lecture Notes in Computer Science*, pages 27–38. Springer Berlin / Heidelberg, 2003. doi: 10.1007/3-540-35828-5_2.

L. Parker, B. Kannan, F. Xiaoquan, and T. Yifan. Heterogeneous mobile sensor net deployment using robot herding and line-of-sight formations. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 2488–2493, October 2003. doi: 10.1109/IROS.2003.1249243.

M. Pfingsthorn, B. Slamet, and A. Visser. A scalable hybrid multi-robot SLAM method for highly detailed maps. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*, pages 457–464. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-68846-4.

A. Poernomo and H. Ying. New cost function for multi-robot exploration. In *Proc. of 9th Intl. Conf. on Control, Automation, Robotics and Vision*, pages 1–6, December 2006. doi: 10.1109/ICARCV.2006.345220.

A. Pokahr, L. Braubach, and W. Lamersdorf. A BDI architecture for goal deliberation. In *Proc. of the fourth Intl. Joint Conf. on Autonomous Agents and Multiagent*

Systems, AAMAS '05, pages 1295–1296, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0. doi: 10.1145/1082473.1082740.

J. Reich and B. Sklar. Toward automatic reconfiguration of robot-sensor networks for urban search and rescue. In *Proc. of First Intl. Workshop on Agent Technology for Disaster Management (ATDM): Fifth Intl. Joint Conf. on Autonomous Agents and Multiagent Systems*, Hakodate, Japan, May 2006a. ACM.

J. Reich and B. Sklar. Robot-sensor networks for search and rescue. In *Proc. of IEEE Intl. Workshop on Safety, Security and Rescue Robotics*, Gaithersburg, MD, USA, August 2006b.

R. Reid and T. Braunl. Large-scale multi-robot mapping in MAGIC 2010. In *Proc. of IEEE Conf. on Robotics, Automation and Mechatronics (RAM)*, pages 239–244, September 2011. doi: 10.1109/RAMECH.2011.6070489.

I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited communication, multi-robot team based coverage. In *Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 3462–3468, May 2004. doi: 10.1109/ROBOT.2004.1308789.

M. N. Rooker and A. Birk. Multi-robot exploration under the constraints of wireless networking. *Control Engineering Practice*, 15(4):435–445, 2007. doi: 10.1016/j.conengprac.2006.08.007.

A. Rosenfeld, G. Kaminka, and S. Kraus. *Coordination of Large-Scale Multiagent*

Systems, chapter A Study of Scalability Properties in Robotic Teams, pages 27–51. Springer US, 2006. doi: 10.1007/0-387-27972-5_2.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter 2, page 54. Prentice-Hall, 3 edition, 2003.

C. Schlenoff. A robot ontology for urban search and rescue. In *Proc. of the Workshop on Research in Knowledge Representation for Autonomous Systems, part of the ACM Conf. on Information and Knowledge Management*, 2005.

S. Scone and I. Phillips. Trade-off between exploration and reporting victim locations in USAR. In *Proc. of IEEE Intl. Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, June 2010. doi: 10.1109/WOWMOM.2010.5534926.

J. S. Seybold. *Introduction to RF Propagation*, chapter Indoor Propagation Modeling, pages 210–214. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2005. doi: 10.1002/0471743690.fmatter.

D. Shell and M. Mataric. On foraging strategies for large-scale multi-robot systems. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2717–2723, October 2006. doi: 10.1109/IROS.2006.281996.

S. Stancliff, J. Dolan, and A. Trebi-Ollennu. Planning to fail – reliability needs to be considered a priori in multirobot task allocation. In *Proc. of IEEE Intl. Conf. on Systems, Man and Cybernetics (SMC)*, pages 2362–2367, October 2009. doi: 10.1109/ICSMC.2009.5346359.

P. Stone and M. Veloso. *Intelligent Agents V: Agents Theories, Architectures, and Languages*, volume 1555/2000, chapter Task Decomposition and Dynamic Role Assignment for Real-Time Strategic Teamwork, pages 293–308. Springer Berlin, 1999.

P. Ulam and R. Arkin. When good communication go bad: communications recovery for multi-robot teams. In *Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 3727–3734, May 2004. doi: 10.1109/ROBOT.2004.1308844.

M. van de Vijsel and J. Anderson. Increasing realism in coalition formation. In *Proc. of the 3rd Intl. Conf. on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS)*, Singapore, December 2005.

R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2): 189–208, 12 2008. doi: 10.1007/s11721-008-0014-4.

V. Verma, G. Gordon, R. Simmons, and S. Thrun. Real-time fault diagnosis. *IEEE Robotics Automation Magazine*, 11(2):56–66, June 2004. ISSN 1070-9932. doi: 10.1109/MRA.2004.1310942.

L. Vig and J. Adams. Issues in multi-robot coalition formation. In L. E. Parker, F. E. Schneider, and A. C. Schultz, editors, *Multi-Robot Systems*, volume 3, pages 15–26. Springer Netherlands, 2005. doi: 10.1007/1-4020-3389-3_2.

R. Wegner. Balancing robotic teleoperation and autonomy in a complex and dy-

namic environment. Master's thesis, Department of Computer Science, University of Manitoba, July 2003.

R. Wegner and J. Anderson. Balancing robotic teleoperation and autonomy for urban search and rescue environments. In *Proc. of the 17th Conf. of the Canadian Society for Computational Studies of Intelligence*, volume 3060, pages 16–30, London, ON, CA, 2004.

N. Wiebe and J. Anderson. Local methods for supporting grounded communication in robot teams. In D. Liu, L. Wang, and K. C. Tan, editors, *Design and Control of Intelligent Robotic Systems*, chapter 14, pages 279–301. Springer-Verlag, Heidelberg, 2009. ISBN 978-3-540-89932-7. doi: 10.1007/978-3-540-89933-4_14.

J. Wong, C. Robinson, and M. Worrell. Urban search and rescue technology needs: identification of needs. Technical report, Savannah River National Laboratory, November 2004.

K. M. Wurm, D. Hennes, D. Holz, R. B. Rusu, C. Stachniss, K. Konolige, and W. Burgard. Hierarchies of octrees for efficient 3d mapping. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4249–4255, September 2011. doi: 10.1109/IROS.2011.6094571.

D. Xu and K. Xia. Role assignment, non-communicative multi-agent coordination in dynamic environments based on the situation calculus. In *Proc. of WRI Global Congress on Intelligent Systems (GCIS)*, volume 1, pages 89–93, May 2009. doi: 10.1109/GCIS.2009.255.

B. Yamauchi. Packbot: A versatile platform for military robotics. In G. Gerhart, C. Shoemaker, and D. Gage, editors, *Proc. of SPIE*, volume 5422 of *Unmanned Ground Vehicle Technology VI*, pages 228–237, 2004.

B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proc. of IEEE Intl. Conf. on Computational Intelligence in Robotics and Automation (CIRA)*, pages 146–151, July 1997. doi: 10.1109/CIRA.1997.613851.

B. Yamauchi. Frontier-based exploration using multiple robots. In *Proc. of the 2nd Intl. Conf. on Autonomous agents (AGENTS)*, pages 47–53, New York, NY, USA, 1998. ACM. ISBN 0-89791-983-1. doi: 10.1145/280765.280773.

F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12(3):317–370, 2003. doi: 10.1145/958961.958963.

S. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso. Ssl-vision: The shared vision system for the robocup small size league. In J. Baltes, M. Lagoudakis, T. Naruse, and S. Ghidary, editors, *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *Lecture Notes in Computer Science*, pages 425–436. Springer Berlin / Heidelberg, 2010. doi: 10.1007/978-3-642-11876-0_37.