

Agent Recovery and Team Management in Dangerous Domains

by

Humayra Anjum Rafi

A Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
of the University of Manitoba
in partial fulfillment of the requirements of the degree of

Master of Science

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada

© Copyright 2025 by Humayra Anjum Rafi

Thesis advisor

John E. Anderson

Author

Humayra Anjum Rafi

Agent Recovery and Team Management in Dangerous Domains

Abstract

Search and rescue missions in hazardous environments are dangerous and challenging for robots. Poor communication, unpredictable events, robot failures, etc., are common adversities in such missions. Robot failures can greatly hamper the team's performance if not managed properly. Partial failures or malfunctions in robots can cause task failures or store inaccurate data about the environment, affecting future tasks and team performance. Total failures like robot deaths can lead to information loss and wasting resources. For my thesis research, I developed an agent recovery framework for heterogeneous teams of autonomous robots operating in dangerous domains like Urban Search and Rescue (USAR). It is an extension of an existing framework that addresses team and task management, recruitment strategies, robot satisfaction and rare skill management in heterogeneous teams of robots in USAR. My agent recovery framework addresses detection, diagnosis of robot failures and the recovery of robots from these failures. The recovery process allows attempts at self-recovery, attributes and roles adjustments to reflect the current capabilities and failure rates of a robot's components, and also considers human assistance in repairing robots. I evaluated my framework in a simulated post-disaster environment with

random robot failures and other challenges. My results showed considerable performance gains in terms of gathering critical information about the environment while also providing useful insights into potential improvements in the framework for the future.

Contents

Abstract	ii
Table of Contents	vi
List of Figures	vii
List of Tables	ix
Acknowledgments	x
Dedication	xi
1 Introduction	1
1.1 Chapter Overview	1
1.2 Introduction	2
1.3 Motivation	4
1.4 Terminology	7
1.5 Approach	11
1.6 Problem Setting	14
1.6.1 Urban Search and Rescue	15
1.6.2 Environment	16
1.6.3 Fault Simulator	17
1.7 Research Questions	18
1.8 Thesis Organization	20
2 Related Work	21
2.1 Chapter Overview	21
2.2 Multi-Robot Systems in USAR	22
2.3 Fault Detection and Recovery in Multi-Robot Systems	28
2.4 Existing Frameworks	34
2.4.1 Gunn’s Framework	36
2.4.2 Active Recruitment Controller(ARC) Framework	45
2.4.3 Fiawoo’s Framework	48
2.5 Conclusion	53

3	Methodology	54
3.1	Chapter Overview	54
3.2	Agent Recovery Framework	55
3.2.1	Failure Detection	61
	Evaluate Fail Rate	61
	Detection methods by failure categories	61
3.2.2	Robot Health Diagnosis	65
	Periodic Diagnosis and Triggered Diagnosis	65
	Diagnosis Workflow	66
3.2.3	Recovery Management	67
	Attribute Fail Rate	69
	Recovery Task	69
	Information Retrieval	77
3.3	Conclusion	78
4	Implementation	79
4.1	Chapter Overview	79
4.2	USAR Concept and Goals	80
4.3	Simulated Environment	81
4.3.1	Simulator	81
4.3.2	Environmental Design and Objects	82
4.3.3	Object Detection	86
4.3.4	Energy Allocation and Dissipation	87
4.3.5	Simulated Challenges	89
4.4	Robot Types	93
4.5	Attributes, Roles, and Tasks	97
4.5.1	Attributes	97
4.5.2	Tasks	99
4.5.3	Roles	104
4.5.4	Ideal Team	107
4.6	Autonomous Control	107
4.6.1	Perceptual Schemas	108
4.6.2	Motor Schemas	111
4.7	Framework-Specific Modules	112
4.8	Mission-Specific Modules	115
4.9	Conclusion	116
5	Evaluation	117
5.1	Chapter Overview	117
5.2	Review of Research Questions	118
5.3	Evaluation Metrics	119
5.3.1	Cumulative vs. Instantaneous data	119

5.3.2	Victims Identified	120
5.3.3	Area Covered	121
5.3.4	Unique Tasks	121
5.4	Environment Generation	121
5.5	Experimental Design	123
5.5.1	Independent variables	125
5.6	Main Experiment Results	129
5.6.1	Victims Confirmed	129
5.6.2	Area Covered	135
5.6.3	Specialized Task Statistics	139
5.7	Analysis	143
5.8	Conclusion	149
6	Conclusion and Future Work	150
6.1	Chapter Overview	150
6.2	Answers to Research Questions	150
6.3	Contribution	153
6.4	Future Work	154
6.4.1	Implementation Improvements	154
6.4.2	Future direction of research	157
6.5	Conclusion	160
A	Experimental Worlds	162
B	Additional Experiment Reports	166
	Bibliography	180

List of Figures

2.1	Overview of how attributes make up the knowledge necessary to define expressions, tasks, and roles in Gunn’s framework. Adapted from Gunn [2011].	37
2.2	Role-based task assignment and exhaustive task assignment in Gunn’s framework. Adapted from Nagy [2016].	44
2.3	The Recruitment Modes Spectrum in ARC Framework. Adapted from Nagy [2016].	46
2.4	A robot’s decision-making process based on its current satisfaction. Adapted from Fiawoo [2019].	50
3.1	An overview of the Agent Recovery Framework. Each component will be explained in detail in the following sections.	59
3.2	An overview of existing work around Autonomous Heterogeneous Teams of Robots in USAR, in the AALAB at University of Manitoba, in addition to my Agent Recovery Framework.	60
3.3	FOV of sensors in a MaxBot.	64
3.4	Overview of recovery task.	70
4.1	One of the experimental worlds that I used for the evaluation of my framework.	83
4.2	Objects in the environment.	84
4.3	Figure (a) shows sonar sensors in DebrisBot for detecting low-lying debris and walls. Figure (b) shows a MidBot detecting a victim using its victim sensor. All fiducial sensor data is visualized with magenta dotted lines, like here.	85
4.4	Energy consumption data shown in Stage for DebrisBot.	89
4.5	This figure provides a close look at the physical structure of each robot type in the team, as well as the total energy stored by the team at the beginning of the mission.	95
4.6	This figure shows a bird’s eye view of the robot team structure and robot roles at the start point.	108

5.1	Breakdown of Experimental trials.	125
5.2	Instantaneous percentage of victims confirmed with replacements. . .	131
5.3	Instantaneous percentage of victims confirmed without replacements. .	132
5.4	Cumulative percentage of victims confirmed with replacements. . . .	132
5.5	Cumulative percentage of victims confirmed without replacements. . .	133
5.6	Instantaneous percentage of area covered with replacements.	135
5.7	Instantaneous percentage of area covered without replacements. . . .	136
5.8	Cumulative percentage of area covered with replacements.	136
5.9	Cumulative percentage of area covered without replacements.	137
5.10	Average number of debris units removed with replacements.	140
5.11	Average units of debris units removed without replacements.	141
5.12	Average number of fires extinguished with replacements.	142
5.13	Average number of fires extinguished without replacements.	143
5.14	Instantaneous percentage of victims identified.	146
5.15	Cumulative percentage of victims identified.	147
5.16	Instantaneous percentage of area covered.	147
5.17	Cumulative percentage of area covered.	148
A.1	Experimental World 1.	163
A.2	Experimental World 2.	164
A.3	Experimental World 3.	165
B.1	Cumulative percentage of victims confirmed with replacements. Data collected from dead AND undead leaders.	167
B.2	Cumulative percentage of victims confirmed without replacements. Data collected from dead AND undead leaders.	167
B.3	Cumulative percentage of area covered with replacements. Data col- lected from dead AND undead leaders.	168
B.4	Cumulative percentage of area covered without replacements. Data collected from dead AND undead leaders.	168
B.5	Overall percentage of victims identified.	169
B.6	Overall percentage of area covered.	169

List of Tables

4.1	Values of physical attributes of robots used.	98
4.2	Computational capabilities of robots used.	99
4.3	Sensor attributes of robots used.	100
4.4	Maximum role suitability of robots used.	105
5.1	Failure probabilities by robot types.	127

Acknowledgments

To begin with, I would like to express my heartfelt gratitude to my advisor, Dr. John Anderson. Your exceptional guidance, support, and patience kept me focused and motivated until the end of this journey. Your meticulous review and insightful feedback on my work helped me build a thesis of which I am truly proud.

In addition, I would like to extend my thanks to my committee members who, despite their busy schedules, dedicated time to carefully review and provide useful feedback on my work that further strengthened my thesis.

I would also like to thank my family and friends who supported me through all the ups and downs of this journey. Your encouragement and emotional support helped me persevere through one of the most challenging chapters of my life.

Finally, I am grateful for the resilience I found within myself for pressing forward despite moments of doubt. I am proud of the strength it took to complete this work and I am glad I did not give up.

This thesis is dedicated to my parents, whose love, support, and prayers have shaped who I am today. Your patience, encouragement, and countless hours on calls made it feel like you were right beside me even though you were miles away. Thank you for believing in me more than I did myself. Words are not enough to express how grateful I am.

Chapter 1

Introduction

1.1 Chapter Overview

This research aims to develop a recovery framework for heterogeneous teams of robots working in dangerous domains. In this research, I will use Urban Search and Rescue (USAR) as a representative of such domains. Robots are more prone to damage in these complex and dynamic environments [Liu and Nejat, 2013]. Given the challenging conditions and limited time and resources, a well-planned recovery framework is vital to help minimize the effect of robot failures on the team's overall performance.

In this chapter, I briefly provide the background and motivation for my research. Next, I will define some technical terms that will be used throughout this thesis. Then, I will present a high-level description of my approach. Following this, I will describe some research questions that this thesis poses, and finally, I will provide an outline for the remainder of the thesis.

1.2 Introduction

Urban search and rescue (USAR) refers to searching for and rescuing victims trapped in urban disaster sites like collapsed buildings, factory accidents, etc., and partially or completely securing collapsed structures [Yang and Parasuraman, 2020]. USAR missions in disaster sites come with challenges that make missions difficult and dangerous, and often pose life risks to human rescuers [Ventura and Lima, 2012; Sharmin et al., 2019]. Moreover, there are tasks that humans cannot perform. For example, humans cannot access narrow spaces or remove large debris [Ventura and Lima, 2012]. Robots can vary in size, strength, and skills. They are also more expendable compared to humans [Sharmin et al., 2019]. Substituting human rescuers with robot teams can not only help minimize human life risk but also increase the mission's success rate [Gunn, 2011; Ventura and Lima, 2012; Sharmin et al., 2019].

USAR is a very challenging domain for robots as well. There is an active domain of research dedicated to facilitating the functioning of robots in this domain [Ventura and Lima, 2012]. Real-life scenarios like collapsed buildings are very unpredictable. It is hard to know when or where a concrete structure is falling or breaking down or where something will catch fire. The environment is unknown and dynamic. The cost of robots is another challenge in this domain. Due to the high level of danger, robot failures are frequent. For example, sensors can get damaged, robots can get stuck under debris, system failures can occur, and so on. Surely, we can send in replacements to make up for the damaged robots or use more expensive and damage-resistant robots, but resources are limited and costly [Sharmin et al., 2019]. One of the main reasons for using robots in USAR is that robots are expendable [Gunn and

Anderson, 2015; Sharmin et al., 2019]. The higher cost of robots will make robots less and less expendable. This defeats some of the purposes of having robots as substitutes for humans.

In addition to the high risk of robot failures, there are challenges like poor communication, time constraints, and information loss. Robots use wireless communication to communicate among themselves and with human rescuers. However, such communication is often not reliable in this environment [Gunn and Anderson, 2013]. Time is another important factor in search and rescue missions. Every second counts as injured victims are waiting in the rubble to get rescued. It is a life-or-death situation. The 7.8 magnitude earthquake that hit Turkey and Syria on Feb 6, 2023, shows us how crucial it is to have proper time and resource management during search and rescue missions [Atay Alam et al., 2023]. Last but not least, there are chances of information loss during USAR missions. The main purpose of a USAR mission is to gather data on victims and the environment. This data is crucial to track down victims in dangerous environments. Poor communication often reduces information sharing among team members, resulting in information loss. Robots can also run out of power or suddenly die because of damage before sharing all information with the team.

Heterogeneous robot teams operating in complex environments such as USAR require a recovery framework to manage teams, tasks, and roles when team members suffer damage. A fire extinguisher robot can still manage to explore the environment even if it has damaged fire sensors. It can still help in other tasks that do not need fire sensors. A robot can take measures to share important information like victim

locations or environment maps with neighbouring robots if it reaches a critical power level. It can also stop working to conserve energy by avoiding energy-intensive tasks. It can make other robots aware of its location so that it can be tracked and rescued later. A recovery framework can help partially damaged or dysfunctional robots recover and help us use the robots to their full potential without wasting resources. It can reduce information loss and also maximize the success rate of the robot team.

1.3 Motivation

The main goal of a USAR mission is to locate victims and map the environment. However, many other tasks can require specific skills in robots. For example, there is a need for fire-extinguishing robots to put out fires, debris-cleaning robots to clear pathways, and so on. Equipping a single robot with all capabilities decreases the expendability of the robot and increases the cost. Moreover, robots are subjected to greater risks in dangerous environments, and there is a higher chance of robot failure. The loss of a robot means the loss of skills that came with that robot. Multiple low-cost robots are much better than having a single high-cost robot in a dangerous environment, such as USAR, where robots are exposed to higher risks [Husain et al., 2013]. In addition, equipping different robots with different tasks enables parallelism that allows robots to perform multiple tasks at the same time [Nagy, 2016]. The diversity of achievable tasks by a heterogeneous team of robots is higher than a homogeneous team of robots [Kiener and von Stryk, 2010].

A heterogeneous team of robots require well-planned team and task management strategies to ensure the best performance from the team. It is important to determine

what role each robot plays in the mission, how tasks are prioritized, etc. Previously, in our lab, a heterogeneous team formation framework was developed that supported the maintenance of heterogeneous teams of autonomous robots operating in disaster zones [Gunn and Anderson, 2013]. This framework was later expanded to include different recruitment strategies to address the team’s lack of robots or skills. Robot failures or losses result in the loss of skills in a team. Different role-based and task-based recruitment strategies were introduced to Gunn’s framework that allowed the recruitment of robots from other teams [Nagy and Anderson, 2016]. However, robot failures can increase the rarity of skills across a team. In such cases, multiple teams can be competing for the same few agents for recruitment. To address this issue, the idea of robot satisfaction was added to allow robots to accept or reject recruitment or task requests based on how satisfied the robot would be accepting or rejecting the task [Fiawoo, 2019].

Existing frameworks in our lab investigate mitigating the effect of robot failures in the team, but they do not consider recovering the failed robots. There are fault detection and isolation frameworks like ReFrESH [Cui et al., 2014] and other frameworks that will be discussed later in Chapter 2. However, most of the current approaches to agent recovery focus more on the fault detection and isolation of failures within robots, not the recovery of failed robots. In addition, most other work abstracts away the team formation and management and the different challenges in dangerous environments and may not be adaptable to dynamic and unpredictable environments. Gunn and Anderson [2013]; Gunn [2011] presented a more realistic approach to solving team and task maintenance challenges in hazardous domains. They developed a framework

to facilitate the study of the formation and maintenance of robot teams in complex and dynamic environments like USAR. It addresses challenges like uncertainty, poor communication, task prioritization, teamwork, etc.

While Gunn’s framework for heterogeneous robot team management [Gunn and Anderson, 2013; Gunn, 2011], Nagy’s Active Recruitment Controller (ARC) framework [Nagy, 2016; Nagy and Anderson, 2017], and Fiawoo’s satisfaction management [Fiawoo, 2019; Fiawoo and Anderson, 2021] address some of the challenges associated with robot failures, they lack a specific plan for the failed robots. There are team, task, and role management strategies in place to minimize the effect of robot failures on the team’s performance. However, there is no recovery plan for partially or fully damaged robots. Partially damaged robots can still be useful to the mission. For instance, a robot with damaged wheels or no mobility can still work as a communication relay. A fully damaged robot can still have parts that can be salvaged and used for building other robots. It could also have critical information about the environment that can be retrieved by human rescuers. Furthermore, having damaged robots operate in USAR missions without proper recovery management can have a cascading effect on the team’s performance. A robot with damaged victim sensors can potentially share the wrong location of victims with other robots. Damaged robots can also pose a hazard to other robots navigating the environment. For example, a robot with malfunctioning motor systems or sensors can keep colliding with other robots. Dead robots can distract team members by causing the members to investigate the dead robots over and over again. A recovery framework can ensure proper management of failed robots to minimize their effect on the mission’s success.

My work specifically addresses the recovery management of failed robots. I designed an Agent Recovery framework leveraging the existing team, task, and role management strategies in Gunn’s, Nagy’s and Fiawoo’s works [Gunn and Anderson, 2013; Nagy and Anderson, 2017; Fiawoo and Anderson, 2021]. The purpose of my research is to come up with strategies to recover and manage damaged robots, minimize waste of resources and cost, and improve the overall performance of the robot team in the USAR domain. I added a health manager in agents that will monitor robot health by running periodic or triggered self-diagnosis workflows and create recovery tasks as needed. The recovery task primarily comprises attempting to self-recover by restarting or recalibrating as primary recovery. As a secondary recovery, the failed robot will adjust its attribute map, and eventually, the role and tasks of the robot will be adjusted accordingly with the help of existing role and task management strategies [Gunn and Anderson, 2013; Nagy and Anderson, 2016; Fiawoo and Anderson, 2021]. Finally, if there is no way to recover the robot using primary or secondary recovery, the robot resorts to tertiary recovery, which requires human intervention to fix the robot. My recovery framework also accounts for information loss and the retrieval of dead or immobile robots.

1.4 Terminology

Here I will briefly explain some of the common terminologies that I will use throughout my thesis.

- **Heterogeneous Robots:** It is costly to equip all robots in the team with expensive motor or sensor capabilities. Not all robots will need to use all skills.

It is a waste of resources to equip all robots in the same manner. Heterogeneous teams increase the efficiency of the team by allowing better division of labour [Gunn, 2011]. For my work, robots in a robot team can differ in locomotion, sensing ability, strength, computational capability, physical structures, etc. In this work, I will be using the term "agent" or "robot" interchangeably.

- **Attributes:** Each agent or robot possesses a set of mission-specific attributes that describe the properties and capabilities of a robot [Gunn, 2011]. Robot attributes are shared knowledge among all robots, allowing other robots to perform identification, task assignment, and recruitment according to robot capabilities. For example, a victim identification task will require a robot equipped with sensors that can identify victims. In Gunn's Framework, robots are pre-defined and static. The attributes do not change throughout the mission. However, in my framework, even though the attributes are pre-defined, they are dynamic. Failures in robots can lead to changes in their attributes. A robot with victim sensing capability can lose this ability due to sensor failure. In such cases, attributes will be adjusted according to the current capabilities of the robot.
- **Task:** A task is a single instance of work that a robot can complete [Gunn, 2011]. For instance, putting out fires, removing debris, identifying victims, and so on. For Gunn's framework, it is assumed that one task must be performed by one robot alone; it does not require multiple robots to perform a single task. The collective effort from all tasks contributes to the overall goal of the mission.

- **Role:** A role defines a set of tasks that the robot occupying the role is expected to be able to perform [Gunn, 2011]. For instance, a robot filling a "debris remover" role must be able to differentiate between debris and a victim, navigate uneven terrain and have the strength and physique to remove debris.
- **Team:** A team is a distributed group of heterogeneous robots filling up different roles and working towards a shared common purpose. They may or may not have perfect knowledge of the team's membership [Gunn, 2011]. There is one team leader, and one of its responsibilities is assigning tasks to the team members. Team members report their task status, location, etc., to the team leader. There can be multiple non-competitive robot teams in the USAR environment working towards a common goal.
- **Minimum Requirements:** Minimum Requirements for a task type are the absolute minimum capabilities required by agents to carry out that task [Gunn, 2011]. For example, an agent must have fire-detecting sensors and fire-extinguishing capabilities to be able to put out fires. These are the minimum requirements for fire-quenching tasks.
- **Suitability:** Suitability in Gunn's Framework is a numeric value that defines the degree to which an agent is suitable to do a task based on its capabilities [Gunn, 2011]. It helps in deciding which agent is best suited for a task.
- **Recruitment:** There can be many cases when a robot or a team might need assistance. For instance, the team might need help with a role that lost its robot, or a robot might identify tasks it cannot complete due to a lack of skills

or due to a full task queue. In such cases, robots can send requests for assistance with tasks. These requests can be referred to as recruitment. New robots are released into the environment periodically to make up for losses. Teams can also recruit new robots or robots from other teams to accomplish tasks [Gunn, 2011]. Recruitment can be permanent (e.g., permanently recruiting to fill a role in the team) or temporary e.g., to just assist with a task). Robots are also allowed to reject recruitment requests in certain circumstances [Fiawoo, 2019].

- **Satisfaction:** The Satisfaction of an agent is the measure of how much utility an agent derives from performing a task. For instance, a fire extinguishing robot or *Firebot* will gain more satisfaction by doing fire quenching tasks than victim detection tasks.
- **Willingness:** In Fiawoo’s work, willingness is defined as an agent’s tendency to accept or reject a recruitment request based on its current satisfaction. A robot is said to be willing if its current satisfaction is low and it needs opportunities for tasks to increase its satisfaction. It is said to be somewhat willing if it has moderate satisfaction and is open to accepting recruitment requests. Lastly, it is said to be not willing if its current satisfaction is high, meaning it is more needed in its current role and is not interested in other recruitment requests [Fiawoo, 2019].

All these terminologies will be explained in greater detail in Section 2.4.

1.5 Approach

For my research, I am going to build my ideas on some existing frameworks of heterogeneous robots in the USAR domain: Gunn’s Framework [Gunn, 2011], the Active Recruitment Controller Framework(ARC) [Nagy, 2016], and Fiawoo’s framework [Fiawoo, 2019]. Gunn developed a team management and task allocation framework for a heterogeneous team of robots operating in complex and dynamic environments such as disaster zones. The team formation changes through recruitment or role swapping to maintain the desired optimal team structure. The framework also facilitates the discovery and subsequent assignment of tasks to members of a team in a distributed fashion. Task and team management also addresses robot failures to some extent. Dead or lost robots create vacant roles in teams, which are then filled by the next best robot in the team or through recruitment. Expanding on Gunn’s framework, Nagy introduced improved recruitment strategies that allowed robots to search for robots actively or concurrently while doing their tasks and not just rely on chance encounters for recruitment [Nagy, 2016]. This increased the chances of finding more suitable roles and task recruits in adverse situations and poor communication. Later, Fiawoo added the concept of tracking the utility and satisfaction of a robot with respect to its location and team. This allowed robots to accept or reject recruitment requests based on their current satisfaction. In addition, robots tend to move to locations where they have better use of their skills or higher satisfaction [Fiawoo, 2019].

Gunn [2011], Nagy [2016], and Fiawoo [2019] developed different strategies to improve the existing framework’s performance and robustness to robot failures. Their goal was to maintain or improve the performance of a robot team even with the

missing robots (either because they got lost in the environment or they died) in the team, or because of the lack of unique robot skills in the environment. Their work, however, does not do anything about the dead or damaged robots themselves. Information gathered by those robots is lost forever, which is a waste of time and resources. It also does not address partial failures in robots. In a challenging and dangerous environment such as USAR, robot failures are inevitable. There can be cases where a robot damages its victim sensor and can no longer detect victims. Or maybe a robot lost its ability to move by damaging its motor or wheels. Robots can also get erroneous readings from sensors due to system malfunction. The current framework does not have any way to deal with such situations where robots are partially malfunctioning. Letting malfunctioning robots operate as usual without taking any measures will deteriorate a team's overall performance. They will keep on failing at tasks they are expected to do. Faulty readings will disrupt the team's progress. For example, if a robot has a faulty victim sensor, the victim locations that it recorded may be incorrect and will share that incorrect location with all members of the team. The robot or the team cannot recognize this faulty condition unless there is an active mechanism to identify such errors. A proper recovery management framework is necessary to manage faulty or dead robots and to address the issues that may arise due to their failure.

Since my recovery framework is mainly intended for agent recovery in dangerous and challenging environments, it assumes that any agent could suffer from partial failure or total failure at any point. Partial failures only affect particular components in the robot, but the robot can still function and perform tasks using other unaffected

components. For example, a robot with damaged wheels or failed mobility can still act as a communication relay and help in task assignment, recruitment, etc., if its WiFi is still working. Total failure can occur when all components fail or if the robot dies. For predictable deaths, like running out of battery power, my framework provides energy-saving fallback modes that buy the robots with critical power levels some time to let other robots know about their location so that they can be rescued later if they die. Information like victim locations and maps based on the area explored so far is also shared, so that the information is not lost with the robot's death. Energy consumption is a concept that was not previously discussed in Gunn's [2011], Nagy's [2016], and Fiawoo's [2019] work. I incorporate this concept in my work since it is one of the important factors that can cause robot failures. Future work can expand on this to see how tasks can be optimally assigned and prioritized while minimizing energy use and maximizing the number of tasks completed.

My agent recovery framework manages the partially damaged and fully damaged or dead robots while leveraging the existing team-role management strategies and recruitment strategies by Gunn [2011], Nagy [2016], and Fiawoo [2019]. There are 2 main phases in the agent recovery framework: The detection and Diagnosis phase and the Recovery phase. In the existing frameworks [Gunn, 2011; Nagy, 2016; Fiawoo, 2019], robots have framework-specific modules or managers that support the functionality necessary for them to manage roles, teams, recruitment, satisfaction, and so on. For my agent recovery framework, I added a health manager that monitors robot health, runs health diagnosis, and creates recovery tasks for the robots if it detects any failure in the robot during the diagnosis. In the recovery phase, the

robot first attempts self-recovery by restarting or recalibrating. If self-recovery fails to fix the problem, the robot then adjusts its attributes map such that it reflects the latest changes in its capabilities and lets its team members know of the changes. This attribute adjustment has a cascading effect on future roles and task assignments because of how the existing framework is designed. Robots are assigned to roles or tasks and recruited based on their suitability to the roles or tasks, and this suitability expression is mainly built based on the robot attributes. Any change in the attributes eventually changes the roles and the types of tasks the robot accepts going forward. Lastly, there can be cases where self-recovery fails and attribute adjustment does not improve the performance of the robot. Such cases call for human intervention to fix the robot. For example, if a robot's communication fails and it can't be self-repaired, adjusting its attributes does not help at all since it can not communicate the changes to other robots and cannot ask for help. The best course of action in such a case is to return to the base to be fixed by humans.

1.6 Problem Setting

This section briefly describes the problem domain, which is Urban Search and Rescue, and the environment for which I am developing the Agent Recovery Framework. I will also describe the fault simulator that I built into the environment to help me evaluate my framework.

1.6.1 Urban Search and Rescue

The Urban Search and Rescue (USAR) domain has seen dramatic progress in research over the years in the robotics community [Jacoff et al., 2001]. The level of danger and complexity of this domain makes it a common representative domain used for the research of mobile multi-robot systems. Human rescuers can use rescue robots to assist them in exploring disaster zones and locating casualties. The aftermath of natural or man-made disasters can be dangerous and highly challenging for both humans and robots. However, the expendable nature of robots makes them better choices for sending them to locations that can be life-threatening to human rescuers. For example, it is safer to send robots to explore a room with dangerous fumes instead of humans. In addition, robots can often easily get into locations which are inaccessible to humans. Rescue robots also reduce the workload and improve the situational awareness of human rescuers [Niroui et al., 2019].

Robots also face great challenges in hazardous environments, such as USAR missions, due to the complexity and uncertainty of the missions. Integrating autonomous teams of rescue robots in USAR missions requires addressing many technical, operational, and regulatory challenges [Solmaz et al., 2024]. Unstructured and often unstable rubble makes robot navigation a challenge. Smoke, dust, fire, etc., can cause sensory difficulties [Jacoff et al., 2001]. Robots have to locate victims and explore and map the environment while making careful decisions, coordinating with teammates, planning their path and tasks, and taking into account the time constraints. Robots are also at a high risk of failures or damage. Heavy objects can fall on robots, there can be fire that can cause damage, robots can have system failures or malfunctions,

etc. To make the robot teams more robust to these failures, it is important to incorporate a recovery framework for the partially or fully damaged robots. The real-world utility and the complexities in the USAR domain make it an ideal candidate domain to develop and evaluate my agent recovery framework.

1.6.2 Environment

Real-world experiments to validate robot performance are critical but can be expensive. In fact, it is not feasible to run real-world experiments for research based on a dangerous and complex domain such as USAR, especially in the primary stages of research, where we need reasonable flexibility in terms of time and resources. Additionally, my research work is around robot failures and recovery. It will be very difficult to experiment given the cost of robots and the time taken to fix/break real-life robots. There are also limitations in how much we can customize and replicate real-life disasters in the real world. For example, placing debris, setting fire, and redoing scenarios after every experiment. For my research work, it is more reasonable for me to use a simulation for my experiments. It is better to start with simulations in the preliminary stages of research, where we will not be limited by time and resources as we would be in real-world experiments [Magid, 2023]. Moreover, we can randomize the map and robot failures, customize robots and capabilities, replicate scenarios, and run them as many times as we need.

In our lab, Gunn [Gunn, 2011], Nagy [Nagy, 2016], and Fiawoo [Fiawoo, 2019] used a multiple-robot simulation tool known as Stage [Gerkey et al., 2003]. It simulates multiple robots, sensors, different environmental objects, etc. The simulation

can be customized according to experimental needs. The Stage project also comes with a robot device server known as Player. It facilitates the rapid development of robot controllers that can eventually drive real-life robots. My Agent Recovery framework leverages the team formation, task-role management, and recruitment strategies implemented by Gunn [2011], Nagy [2016], and Fiawoo [2019] using the Stage simulation tool. I use this tool for my implementation and experimentation as well. The simulation maps are randomly generated. There are debris, walls, and fire as obstacles. True and false victims are randomly placed on the map for the robots to find. Robot teams are formed with multiple robots with different capabilities and specialties. More about the environment will be discussed in detail in Section 4.3.

1.6.3 Fault Simulator

The evaluation of my Agent Recovery Framework requires simulation of robot failures, which Stage does not provide. Gunn [2011] simulated robot failures at a very high level as a part of his framework’s evaluation. In his failure simulation, robots are randomly shut down either temporarily or permanently. His goal was to see how the team adapted in response to the absence of a team member or a lack of skill in the team. However, my work focuses more on how the affected or failed robots will respond to failures, how they will adapt their roles and tasks to contribute to the mission. Besides, in real-world USAR missions, robot failures are not just limited to the permanency of shutdowns (total robot failures). Robots can have unresponsive components(sensors, wheels, etc.), error readings, power failures, and so on. Qiao et al. categorized faults and failures into 3 principal categories:

faults, soft failures, and hard failures [Qiao and Weiss, 2016]. They defined fault as defects, such as incorrect readings or decisions caused by inherent flaws in design or implementation. Soft failures refer to degradation or external wear and tear changes that may have damaged something within the system. Finally, hard failure indicates more permanent or total damage to the robot or robot component.

I used a similar concept to build a Fault Simulator and add it to the Stage simulation tool. Each robot has its own fault simulator that will simulate random failures at random intervals. Failures can be partial or total based on whether the robot can still operate to some extent. There can also be different kinds of malfunctions, like erroneous or faulty readings from components and unresponsiveness. In Gunn’s implementation, robots suffered temporary failures that they recovered from automatically after some timeout [Gunn, 2011]. But this is unrealistic. We need real measures to help recover the robots, which is the purpose of my work. This will help me see how my agent recovery framework improves robot and team performance in the presence of different types of robot failures. The fault simulator will be described in detail in Section 4.3.5.

1.7 Research Questions

In this section, I will briefly discuss the questions I will answer using the approach (Section 1.5) and evaluation (Section 1.6) discussed in the previous sections.

1. **What effect does my agent recovery framework have on the overall performance of teams of heterogeneous teams of robots operating in dangerous domains?**

I will determine the overall performance of the team in terms of the number of victims located and the area explored, in the presence versus in the absence of my recovery framework. I will compare these against a baseline approach, which represents an ideal world without robot failures. This will allow me to learn how well the robots perform ideally without robot failures and how we can expect them to perform in more realistic scenarios where robot failures are inevitable.

2. When does my framework work best with respect to different factors such as type of recruitment strategies, quality of communication, presence or absence of replacements, etc. ?

I will experiment with various combinations of recruitment strategies by Nagy [2016], poor versus good communication, the presence or absence of replacements, difficulty of mission or different levels of probabilistic robot failures. I will determine when my framework performs best and when it performs worst. This will also help determine which recruitment strategy works best with my recovery framework under different circumstances. Moreover, one of the purposes of sending in replacement robots is to make up for failed robots. Through my experiments, we will see if it is really necessary to get replacement robots, if robots can recover or if there is a chance for better planning and use of limited resources.

1.8 Thesis Organization

The remainder of my thesis is outlined as follows:

- **Chapter 2 - Related Work** - This chapter reviews literature related to research on multi-robot systems operating in dangerous domains, with a focus on robot attributes, roles, task allocation, team, role, and task management, failure detection and diagnosis, and agent recovery. In addition, I provide a detailed description of the existing frameworks by Gunn [2011], Nagy [2016], and Fiawoo [2019] that I am extending work on.
- **Chapter 3 - Methodology** - This chapter describes my agent recovery framework in detail.
- **Chapter 4 - Implementation** - This chapter describes how I implemented my framework in the USAR domain (robot types, environment, tasks, etc.), including the detailed description of the fault simulator I implemented to evaluate my framework.
- **Chapter 5 - Evaluation** - In this chapter, I provide the experimental details for the evaluation of my framework. I present the results that I obtained from my experiments. Lastly, I analyze these results in detail.
- **Chapter 6 - Conclusion** - In this chapter, I conclude my thesis while answering the research questions in Section 1.7 based on my framework's evaluation. In addition, I discuss the challenges and limitations that I faced in my research work. Lastly, I discuss the future scope of my work.

Chapter 2

Related Work

2.1 Chapter Overview

My work primarily focuses on the recovery of failed robots in dangerous domains. However, my agent recovery framework involves fault detection and diagnosis in robots, adjusting robot attributes, which further requires roles and tasks adjustment and results in an overall change in the team structure. Fault detection and diagnosis include close component-level monitoring of robot health. Agent recovery includes taking necessary measures like role and task adjustments to address and permanently or temporarily resolve the robot failures to allow the failed robots to contribute to the team's mission as much as possible and have minimal effect on the robot team's overall performance.

In this chapter, I discuss some important research done around robot recovery and multi-robot teams operating in dangerous domains and how these works relate to my research work. I discuss the works related to mine in 3 sections. In Section

2.2, I discuss some research around multi-robot systems like robot team formation, task and role management, etc. and how robust these existing strategies are to robot failures, which is the basis of my research. In Section 2.3, I discuss some research done around fault tolerance, detection, diagnosis, and recovery in robots and robot teams. This discussion is not necessarily limited to rescue missions but also includes recovery from failures in robots in general (e.g., industrial robots). Finally, in Section 2.4, I discuss the existing frameworks developed for the USAR domain by Gunn [2011], Nagy [2016], and Fiawoo [2019]. My framework is an extension of these existing frameworks. I extended the task and role management by Gunn [2011], recruitment strategies by Nagy [2016], and satisfaction management by Fiawoo [2019] such that these now address failed robots as well. So, in that section, I will describe the existing frameworks, how they differ from my work, and how my work can make robot teams more resilient to robot failures.

2.2 Multi-Robot Systems in USAR

The coordination and control of multi-robot systems, especially in dangerous domains like urban search and rescue, have gained significant attention in the research community in the past decade. The complexity and challenges of controlling multi-robot teams for rescue missions in dangerous domains made it an interesting problem domain for research with promising potential in USAR applications [Liu and Nejat, 2013]. In this section, I will briefly review the work done so far regarding the challenges addressed in the study of rescue robotics.

While reviewing the current status of multi-robot systems in the context of search

and rescue, Drew [2021] sketched out 4 broad challenge areas based on the reliability, autonomy, and mobility of multi-robot systems: perception and planning, communication, human-robot interaction, and Cost/Function trade-off. Perception and planning contribute to cooperative mapping, team management, task allocation, and so on. In a search and rescue operation, robots are required to explore and map out an unknown environment while navigating obstacles and locating casualties. Several simultaneous localization and mapping (SLAM) techniques have been developed to allow robots to autonomously map USAR environments and locate and track them using the generated map [Cadena et al., 2016; Danping et al., 2019]. For example, Dayoub et al. [2011] proposed a method based on short-term and long-term memory concepts to improve the robustness of localization and mapping in dynamic environments. Information about the map is moved between different layers of memory while keeping the reference views of a robot’s map up-to-date. To reduce factor graph optimization complexity, Ni and Dellaert [2010] recursively partitioned the SLAM graph into multi-level sub-maps using the nested dissection algorithm to generate a cluster tree. Despite the progress made on the SLAM algorithms, there are still novel research questions arising around developing fail-safe SLAM, recovery, SLAM’s robustness to hardware failure, etc. [Cadena et al., 2016]. SLAM solvers are vulnerable to outliers, which degrade the quality of estimates and ultimately lead to an incorrect linearization point, which is hard to recover from [Cadena et al., 2016]. Hardware failures can also impact SLAM systems. Malfunctioning sensors with low accuracy can lead to poor estimates. One way to improve SLAM estimates is to be able to detect and recover degraded hardware operations, which is one of the main goals of

my research.

Robot teams in rescue missions can be teleoperated or semi-autonomous, or fully autonomous. Studies have shown that human operators can effectively teleoperate only up to eight robots at a time before performance starts degrading [Hong et al., 2018]. This led to studies into semi-autonomous or autonomous robot teams, which require efficient autonomous team and task management strategies, especially in complex and challenging environments like USAR. Liu et al. [2012, 2013] introduced a MAXQ hierarchical reinforcement learning (HRL) based semi-autonomous control architecture to address exploration and victim identification problems through optimal task allocation and execution in a robot team. The fundamental principle of MAXQ is to decompose a root task (e.g., USAR mission goal) into smaller tasks (e.g., navigate, identify victim) to achieve a hierarchical optimal policy for the overall task. It allows robots to learn from each other’s previous experience to make informed decisions about who should perform what task at any given time to achieve optimal performance. Human control is requested only when necessary. Later, Hong et al. [2018] developed a semi-autonomous controller based on the MAXQ hierarchical reinforcement learning model for USAR missions, which showed significant improvements in task performance and reduced human-robot interaction effort. However, these studies don’t say how robust this controller is to robot failures.

A lot of studies focused on improving human-robot collaboration in USAR missions. Sharmin et al. [2019] developed a working prototype of a team of low-cost autonomous robots that sends continuous thermal and video feed and sensory data from the environment to humans via a web application. The data can help humans

plan their mission accordingly. While efficient team and task management is necessary for control and coordination of semi-autonomous or autonomous multi-agent teams, it is also important to account for robot failures. Dead or malfunctioning robots can degrade a team's performance (e.g., a malfunctioning robot being assigned specialized tasks, dead robots resulting in a lower task completion rate). Chandra and Rocha [2016] introduced a knowledge-based framework for human-robot collaboration in USAR that enables efficient context-based knowledge sharing and reasoning among team members. They used an ontology that represented and correlated different contextual concepts like scenario-based, agent-based, mission-based, and team-based contexts. Agents obtain information from the environment, which is later processed and updated to the knowledge base. This framework is designed to detect any inclusion or reduction of teammates. When a robot newly joins or rejoins a team, all events stored in the local knowledge base are shared with the team to synchronize the knowledge of all teammates. This makes the system resilient against communication failures but does not directly address robot failures.

There are also studies that have taken a more autonomous approach in handling complex problems like resource management, poor communication, etc. in USAR domain. Yang and Parasuraman [2020] proposed a need-driven multi-robot cooperation mechanism. They encoded the robot's needs hierarchy in a behaviour tree structure. Different needs of the robots include safety needs(avoid collision), basic needs(energy, mobility, etc.), capability needs(supply resources), teaming needs(increasing utility), and learning needs(upgrade). Robots are able to reorganize to adapt to the current situation as needed to minimize cost and loss of agents. Banfi et al. [2018] addressed

the poor communication problem in mobile robot teams with the help of Integer Linear Programming (ILP) to compute optimal ways to redeploy robots as communication relays incrementally. In my work, I will use a similar idea of using partially damaged robots as communication relays if the robot is still able to communicate.

Robot teams deployed in search and rescue missions are often heterogeneous. A homogeneous team of robots may seem ideal for rescue missions, where any robot in the team can accomplish any task they encounter, without asking for help. However, this is not at all cost-effective or realistic [Gunn, 2011]. In addition to the increased cost of redundant equipment for all robots, there is also a higher cost incurred in case of robot failures [Liu et al., 2013]. The dynamic nature of USAR environments may require different robot morphologies (legged robots, tracked robots, etc.), which cannot all practically be equipped into one robot [Delmerico et al., 2019]. In a study, the Department of Homeland Security (DHS), Federal Emergency Management Agency (FEMA) and the National Institute of Justice (NIJ) identified and defined functional requirements in rescue robots that meet the needs of both USAR teams and law enforcement agencies. The result of the study was a list of 13 robot categories and 103 performance requirements by the responders [Schlenoff and Messina, 2005]. There has also been research studying the division of labour, roles, teams, and task management in a heterogeneous team of robots. Some role-based approaches include the explore vs. relay roles, where explorers explore the environment and relayers relay the information from explorers to the base station [de Hoog et al., 2010]. Gunn’s Framework has a more diverse team structure with roles primarily being sorted into leader, coordinator, and explorer types [Gunn, 2011]. It allows role-swaps and recruitment,

which makes the team more dynamic and robust to robot failures. If any robot in the team dies, the vacant role is filled by the next best-suited robot in the team.

Sivaraman et al. [2025] introduced a pack hunting strategy for USAR operations, where robots can switch between the roles of 'Chaser' and 'Flanker' based on current data, improving adaptability. Chasers pursue targets actively while flankers assist chasers when necessary. Targets or tasks are dynamically reallocated if a robot's status changes, such as low energy or communication failure. Brutschy et al. [2012] employed a simple task allocation strategy that allows behavioural specialization in a robot swarm. Robots learn, adapt, and specialize in tasks through repetition. They select tasks with a higher probability of improved performance. They also employ a form of forgetting, where if a robot has improved on one task and then starts working on another task, it loses part of its performance gain for the former task if it has not performed it for a while. This adaptive behavioural specialization can potentially help in role adjustment needed for robots that are partially damaged or malfunctioning and have performance degradation. However, behavioural specialization is not advantageous in environments like USAR, as specialists may not be fast enough to adapt to the changes in task distribution, team structure, and environment [Brutschy et al., 2012].

Chitikena et al. [2023] discussed ethical and design issues that should be considered at the development stage of robotic and cognitive technology in search and rescue missions [Chitikena et al., 2023]. One of the issues included in their ethical framework is accountability. It is challenging to determine who is liable for harm (e.g., injuring victims, failing to detect victims) caused due to hardware failures that are not always

in the control of the operator or the developer. There is a higher risk of robots getting damaged in dangerous and unpredictable environments. A lot of studies around multi-robot teams in dangerous domains address the potential robot failures and incorporate strategies such that these failures have minimal effect on the team's performance (e.g., role-swap in Gunn's framework [Gunn, 2011]), but very few of these studies address the management and recovery of damaged robots in dangerous environments. In my agent recovery framework, I aim to incorporate recovery for damaged robots into team, task and role management strategies such that even partially damaged robots can contribute to the mission's success. Robots will adapt their roles and tasks according to their current health status. In addition, recovery steps like recording robot location, asking for rescue, critical knowledge transfer, etc., are considered so that all robots are retrieved and accounted for in the mission.

2.3 Fault Detection and Recovery in Multi-Robot Systems

The existing research in rescue robotics strives to improve performance while making the robots and teams robust to challenges faced in search and rescue operations. Some studies focus on making the multi-robot teams fault-tolerant, while others specifically design fault detection, diagnosis, and recovery mechanisms to directly address the failures. The approaches taken to recover from robot failures are diverse. In this section, I will discuss the research around fault tolerance, detection, diagnosis, and recovery while also making some comparisons with my work.

Over the years, researchers have taken different approaches to fault detection in multi-robot systems. Arrichiello et al. [2014] presented a combination of distributed and local approaches to solve the fault detection and recovery problem for teams of networked robots. Every agent has a fault observer module that uses local information and a vector received from the neighbours to make an estimate of the overall team. A set of fault-sensitive residual vectors is compared with a set of adaptive thresholds to detect faulty robots. In case of non-disruptive faults, a fault-tolerant approach is pursued by adding an adaptive compensation term. In case of disruptive or unrecoverable faults, the faulty robots are completely removed, and teams are re-configured. In my work, I will also use different thresholds to detect faults in robots. For instance, there can be a threshold for the maximum and minimum distance for identifying an object based on the field of view (FOV) of a sensor in the robot. If the sensor's readings show objects located outside the usual FOV or at a distance outside the minimum or maximum range, then we can suspect that there is something wrong with the sensor. However, this is a very simplified way of detecting errors in distance sensors. In the real world, there are many kinds of sensors and they are often more complex. Distance sensors can output incorrect readings due to calibration errors, physical damage, etc. Sensor failure detection is a vast area of study that, which lies outside the scope of my thesis. My thesis focuses more on the recovery of failed robots.

Marino et al. [2017] later proposed a distributed Fault Detection and Isolation (FDI) schema for a homogeneous team of robots based on linear time-discrete dynamics. Once a fault is detected, the faulty agents are removed from the team,

carefully following a recovery strategy based on maximum detection time to avoid abnormalities that could arise from inconsistent team knowledge among team members. Tousi et al. [2008] introduced a hybrid fault detection, isolation and recovery (FDIR) methodology for a team of unmanned vehicles. It consists of a low-level or agent-level FDIR and a high-level or team-level FDIR. The low-level FDIR is based on a quantitative model-based approach that uses a priori knowledge to detect inconsistencies or residuals in the system. The high-level FDIR uses a discrete-event system (DES) supervisory framework. The DES supervisor maintains a team's best performance by disabling team structures where faulty robots have a large impact on the team. Tousi and Khorasani [2009] further extended this methodology to build an optimal hybrid FDIR methodology where the DES supervisor can analyze available solutions and find the optimum path instead of using just feasible solutions based on given specifications and limitations. The Hybrid FDIR framework is designed to detect failures, but it cannot completely recover agents from failures [Tousi et al., 2008]. So, the recovery includes the supervisor removing team structures with faulty agents.

Parker and Kannan [2006] presented an adaptive causal model method (CMM) called LeaF (Learning based Fault Diagnosis), which uses a pre-specified partial causal model for fault detection and recovery. When a known fault is encountered, the robots can take recovery actions as instructed in the nodes of the causal model. In case of new, unknown faults, the robots use case-based reasoning (CBR) to extract relevant recovery actions from prior experience. The causal model is then updated with the new fault experience. If the system fails to classify a fault, the human operator

is requested to update the causal model with recovery measures necessary for the unidentifiable fault. Recovery measures include re-attempts, resets, notifying human operators, etc. However, in fast-changing and time-critical domains like USAR, learning and adapting the causal model with appropriate recovery actions may be complex and time-consuming. Moreover, we have to consider cases where immediate recovery robots are not possible. The recovery of robots in such cases must include management of malfunctioning robots that can not be fixed right away, such that they can still help the team in other ways with minimal disruption to the team's efforts.

Fault tolerance and recovery in multi-agent systems is also a popular topic of research in robotics. Parker [1998] proposed a fully distributed behaviour-based architecture that models motivations. They modelled two types of internal motivations. The robot's *impatience* motivation increases as a robot waits for its team member to complete a task, and at a certain point, the robot assumes its team member has failed in doing the task and does the task itself. The robot's *acquiescence* motivation helps the robot handle situations where the robot itself fails to perform a task. Parker further added a learning mechanism to allow the rate of impatience and acquiescence to be context sensitive. However, this architecture does not have any mechanism for agent recovery from failures. Liu et al. [2015] developed a supervisory controlled architecture where a human oversees a large heterogeneous team of robots in USAR. It helps determine a team's behaviour in response to robot failures. It also addresses task recovery in the event of robot repair. When a robot fails and needs to be fixed, the tasks allocated to the robot are returned to the task buffer so that other robots can complete them. Then the control of the failed robot is given to humans until it is

repaired. Humans can also retrieve and reallocate tasks from failed robots. A priority is maintained in agent recovery when there are multiple robot failures in the team. Liu et al. [2015] touch on the topic of agent recovery, but they do not consider the communication challenges in USAR and require supervisory control. My framework considers different real-life challenges of USAR, including communication failures during agent recovery. It will allow autonomous agent recovery and management to some extent.

Hofbaur et al. [2007] presented a detailed model-based fault diagnosis and re-configuration framework for a single autonomous robot. It ensures that the robot completes its tasks as much as possible, even if it has a hardware failure. A robot can switch its operation mode according to the type of failures. In my agent recovery framework, robots can switch to energy-saving or immobile modes when health is critical, but can still do other tasks like recruiting team members, managing the team, etc. Long et al. [2003] presented their approach in sensor diagnosis and recovery for single and multiple robots. In their work, for fault diagnosis, robots generate a list of hypotheses about the fault and then perform tests for the listed hypotheses. Robots can also query peers for sensor information to correlate with local test results and diagnose faults. Once the fault is confirmed, corrective actions are taken accordingly, like re-configuring sensors, re-calibrating the camera and so on. My work will also include diagnostic tests, but the results will be compared to pre-defined expected functions (e.g., a sensor is expected to detect objects only within its field of view with respect to the robot's position). Surely, there are more sophisticated methods of failure detection in robots; failure detection in my framework is only a part of it,

not the only focus. Fault diagnosis and recovery in my framework are more complicated because of the heterogeneity of robots, and so I used a simpler approach to detect errors in robots. Parker and Tang [2006] proposed using sensor sharing through dynamically formed coalitions to increase fault tolerance in multi-robot systems. This allows multiple robots to collaborate to perform a single task through sharing information between schemas. Gunn’s framework allows coalitions to some extent, where robots with high-accuracy sensors help confirm victims identified by robots with low-accuracy sensors [Gunn, 2011]. It can be extended to allow coalitions for other tasks as well in the future. However, breaking up tasks into smaller subtasks to allow efficient coalitions can make the mission planning more complicated.

Several health monitors, diagnostics, prognostics, and maintenance frameworks (collectively known as Prognostics and Health Management (PHM)) have been developed over the years for direct monitoring and management of robot health and failures [Qiao and Weiss, 2016]. HSU et al. [2021] developed an intelligent health diagnostic system for industrial robots to prevent failure and performance degradation through constant monitoring, prediction and estimation of equipment and production performance. The health diagnostic system is composed of 3 subsystems: the fault detection subsystem, the fault diagnosis subsystem, and the health evaluation subsystem. The fault detection subsystem continuously monitors tasks and the operational status of robots. The fault diagnosis subsystem is triggered when the operation status reaches a warning threshold. The historical normal data of the current task is analyzed and triggers the health evaluation subsystem if the warning conditions are not met. The health evaluation subsystem evaluates robot health and generates a

Robot Health Index to quantify the health status of the robot. Similarly, I added a health manager to the robots in Gunn's, Nagy's, and Fiawoo's framework [Gunn, 2011; Nagy, 2016; Fiawoo, 2019] that monitors robot health, runs diagnostic tests on robot components, and creates appropriate recovery tasks to address the failure in the robot. A lot of research related to robot failures focuses on fault detection and diagnosis, such that it allows for supervisory control of recovery by human operators. Very few studies have examined the autonomous recovery of damaged robots. While full-autonomous recovery of robots is not always possible, measures can be taken to mitigate some effects of the damage to the robot and allow the robots to operate until they are repaired by humans. My work deals with this type of recovery management of partially or totally damaged robots operating in teams in dangerous domains where resources are scarce and immediate repair of robots is not always possible.

2.4 Existing Frameworks

In our lab, Gunn and Anderson [Gunn, 2011; Gunn and Anderson, 2013, 2015] developed the first step in a framework for the formation of dynamic heterogeneous teams of robots for USAR. After any loss in a team, other robots switch roles to fill the gap, even if they are not the perfect fit. The team can recruit new members through redistribution with other teams in the environment via encounters or passive recruitment. Building on Gunn's framework, Nagy and Anderson [Nagy, 2016; Nagy and Anderson, 2016, 2017] proposed the Active Recruitment Controller (ARC) framework that allows permanent and temporary recruitment of skills. It also allows concurrent recruitment where robots broadcast recruitment requests while doing other

tasks. There is also active recruitment where robots can stop doing tasks and actively search for other robots for recruitment. Later, Fiawoo and Anderson [Fiawoo, 2019; Fiawoo and Anderson, 2021] considered robot satisfaction and sense of locality to maximize the utility of rare or unique skills in the environment. The rarity of skills is usually caused by the loss of robots. In Fiawoo’s framework, rare robots tend to stay in locations or teams where they can achieve the highest utility, even if it means rejecting other task requests or leaving their own team.

Even though the existing frameworks in our lab have strategies in place to make up for lost robots in the environment through recruitment or better management of rare skills, they do not directly address the problem of robot failures or what happens to the failed robots. There is no way to diagnose failures in robots, nor are there any agent recovery strategies. If a team loses communication with a robot for a certain amount of time, it is considered lost. However, there can be different levels and types of damage a robot can have. Different damaged robots will need different kinds of recovery. In my work, I detect and diagnose the type of damage in a robot and robot recovery is done accordingly. Robot recovery included reconfiguration of robots, reassessment of skills and reassignment of roles in case of partially damaged robots, retrieval and re-use of intact parts like sensors, camera, etc. in non-functional robots, or robots are sent back to base to be repaired or salvaged by humans.

My work expands on Gunn’s, Nagy’s, and Fiawoo’s work [Gunn, 2011; Nagy, 2016; Fiawoo, 2019] to include a recovery framework that manages partially or fully damaged robots. A part of my recovery strategy revolves around how the damaged robot’s attributes, roles, and tasks are adjusted to account for the failure in the robot.

As my framework is an extension of the existing work in our lab, it is necessary to understand this work before I proceed with discussing my framework later. In this section, I will discuss the main concepts of Gunn's, Nagy's, and Fiawoo's work [Gunn, 2011; Nagy, 2016; Fiawoo, 2019].

2.4.1 Gunn's Framework

In this section, I will briefly explain Gunn's framework, which is central to my work. This framework is a typical representation of the USAR domain with challenges we see in real life, like poor communication, a dynamic environment, limited resources, and so on. It supports the maintenance of teams of heterogeneous robots in complex disaster zones. Collapsed buildings, minefields, etc., are some of the many real-life scenarios that can be represented with the help of this framework. Using Gunn's framework as my reference allowed me to make important implementation and design decisions for my research.

- **Attributes:** The agents in this framework differ in physical attributes, skills, and capabilities. Each robot is specialized for a certain set of tasks. To identify the type of tasks a robot is suited to perform, Gunn [2011] used attributes. Attributes refer to a set of properties used to define the types of tasks a robot can perform. Each attribute is assigned a value at the agent design time. For example, *CanDetectVictims* can be a boolean attribute that determines if the robot has the ability to detect victims, or *Expendability* can be a numeric attribute to determine how unique the robot is or how expendable it is.

Even though Gunn based his strategies on pre-defined attribute-value pairs for

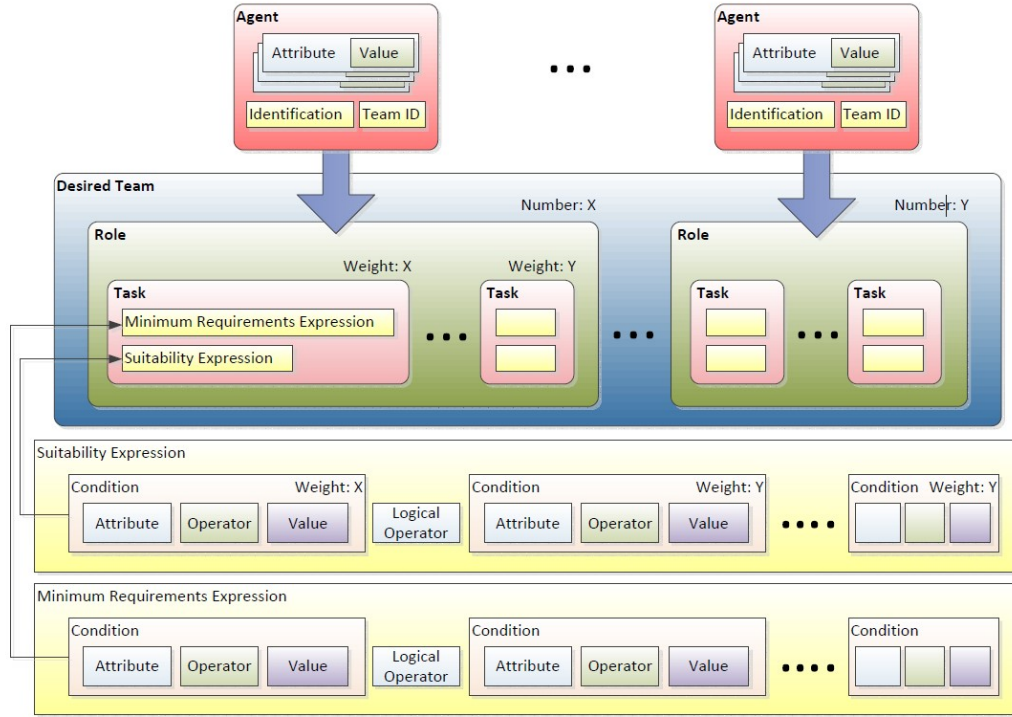


Figure 2.1: Overview of how attributes make up the knowledge necessary to define expressions, tasks, and roles in Gunn's framework. Adapted from Gunn [2011].

the robots, for my framework, these values will be determined dynamically. A robot with attribute *CanDetectVictims* = *True* can be changed to *CanDetectVictims* = *False* if it loses or damages its victim sensors. However, it may still have the ability to move around on flat terrains and do other tasks. I also add a *Attribute Fail Rate* to each attribute that determines how often the component related to the attribute might fail or how well it works. This will help in assigning roles and tasks to the best-suited robots based on attribute state. Robots will run periodic self-diagnostic tests to determine any damage or dysfunction in their system. Any change in attribute-value will be reported to team leaders so that they can re-assign roles and tasks accordingly. The

attribute and role reassessment will be discussed in detail in Chapter 3.

- **Task:** As discussed in Section 1.4, a task is the basic unit of work that contributes to the overall goal of the mission. A task can be performed by a single robot in isolation. It doesn't allow or require shared effort. Extinguishing fire, exploring the frontier, and identifying victims are some of the common tasks in USAR. In this framework, a task is also defined as a set of *minimum requirements* and *suitability expression*. *Minimum requirements* refers to the attributes that are mandatory in a robot for it to be able to perform the task. It is represented by an expression formed with a set of conditions joined by logical operators (See Figure 2.1). Each condition applies to an attribute. Equation 2.1 is the minimum requirement expression for a fire-quenching task. To quench fire, a robot must be able to move to the location of the fire, have the ability (e.g., having an extinguisher) to put out the fire, and be able to detect the fire. *Suitability*, or more specifically task suitability, is a numeric value that determines how suitable a robot is for carrying out a task, used for choosing the best-suited robot. The suitability expression is formed by a set of weighted conditions joined by some logical operators (See Figure 2.1). The numeric task suitability is calculated by summing the weights of the conditions in case of the *AND* operator or the maximum weight in case of the *OR* operator. In Equation 2.2, the task suitability can be a maximum of $70 + 30 = 100$. Here, 70 and 30 are the weights of the conditions $CanQuenchFire = true$ and $HasFireDetector = true$ respectively. These weights reflect how important the condition is to perform the task. In general, the maximum value of task

suitability is 100. For my work, I will modify the task suitability expression to factor in the *Attribute Fail Rate* per attribute or condition to be able to better distinguish how well the robots are satisfying each condition. More will be discussed in Chapter 3.

$$M = (Mobility > 0) \text{ AND } (CanQuenchFire = true) \text{ AND } (HasFireDetector = true) \quad (2.1)$$

$$S = (CanQuenchFire[70] = true) \text{ AND } (HasFireDetector[30] = true) \quad (2.2)$$

Each task also has a *priority* attribute that indicates its importance with respect to other tasks. If a robot is assigned a high-priority task, it will drop the current tasks it was doing if its priority is lower than the newly assigned task. It will go back to doing the old tasks once it completes the high-priority task.

- **Role:** A team is formed of different roles. A role is defined as a set of tasks that a robot occupying that role is normally expected to be capable of performing. If $M = T_1, T_2, \dots, T_i$ is a set of all tasks a mission can have, then role R_i is a subset of M or the mission tasks that are normally expected of an agent occupying a role. Each expected task in the subset has an importance weighting associated with it to distinguish how important the task is to the role. It acts as a weight in the overall role suitability calculation. Every role also has an associated ideal task that the robot defaults to when it has no task in its task queue. An agent

must meet the suitability requirements to be qualified to occupy a role. The role suitability is calculated by summing the weighted task suitability for each expected task (See Algorithm 1). This allows less qualified agents to fulfill the role until a better candidate is found. For example, a debris removing agent could assume the role of a leader even if it lacks the skill of managing the team map, which is one of the expected tasks of a team leader role, in case no better candidate for the leader role is found.

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

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

Require: $W = \{W_1, W_2, \dots, W_n\}$, The weights of tasks T expected of role R

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

```

1:  $S = 0$ 
2: for all  $T_i \in \{T_1, T_2, \dots, T_n\}$  do
3:    $E_i \leftarrow \text{Evaluate suitability expression of } T_i \text{ against } A$ 
4:    $S = S + E_i W_i$ 
5: end for
6: return  $S$ , suitability of agent to fill role  $R$ 

```

- **Team Management and Maintenance:** Gunn and Anderson [2015] introduced different strategies to maintain teams in the USAR domain. Every team will have a leader who is responsible for the overall team management. A team will always try to maintain an optimal team structure to contribute to the mission efficiently. This optimal team structure can otherwise be called the *desired team*. It is a team with a particular set of roles, and each role is filled with

an optimal number of suitable agents. Each role will have a maximum and a minimum number of agents defined. The minimum number is the minimum number of agents needed in a role to accomplish a mission. The maximum number is the threshold beyond which the performance of the team starts degrading. This is because having too many agents filling a role will cause too much communication and interference.

Every agent in the team regularly broadcasts its status to its team members. If a robot is not heard from for a certain period, it is considered lost. Robots can be lost for many reasons, like failing to communicate their presence to the team due to network issues, getting stuck somewhere, straying too far away from the team, etc. Such a loss of team members can result in a lack of skills in the team. Replacements are sent into the environment to make up for lost skills. It is important to note that not all members of the team will have the same view of the team membership at any point in time. Poor communication can cause agents to miss team updates. To address this, there are periodic ***role switch checks*** that are performed to determine if the robots in the team are currently occupying the roles they are best suited for. During a role switch check, each team member creates a list of underfilled roles in the team and ranks them by their *role score value*, which is the sum of suitability to fill those roles and the importance of the role to the team. If they find a role with a higher role score value than the current role, they will switch to that role and notify the other team members of the change. If a role has fewer than the minimum agents, the importance weight of that role will increase, and during the role switch check,

more priority will be given to fill that role.

It becomes challenging to efficiently rebalance and fill roles when there are too many losses in the team. Gunn's framework [Gunn, 2011] relies on chance encounters with other teams or stray single robots to rebalance team membership and recoup lost skills. When two agents from two teams encounter each other in the environment, they first see which of them(if any) is computationally capable of performing a merge and distribution between the two teams. The most capable agent takes all members out of both teams and then iterates through each agent and reassigns them to their most suitable roles. Hence, two new teams are formed or a single new team is formed by integrating members of both teams (e.g., in case of encountering single robot teams). The agents then communicate the new structure to their respective teams.

In Gunn's framework, robot failure can mean being stuck or damaged or dysfunctional or losing communication with the team. These do not necessarily mean that the robots are completely useless. While replacements, periodic role switch checks, and team redistribution address the loss of team members by rebalancing team structure, Gunn's framework does not address the management of the lost robots. The lost robots could still be fully functioning robots that just lost communication or got stuck. In my research, I develop a recovery framework to recover agents from failures. If full autonomous recovery is not possible, I reassess the skills, update attribute values and roles of the damaged robots and then perform the redistribution of members in the team. If any role exceeds the maximum number of agents due to this redistribution, the damaged

robot can form a team of its own and look to recruit members or be recruited by other teams in the environment. More will be discussed in Chapter 3.

- **Task Management:** Every robot in the environment can identify tasks. If they are suitable to perform the task and if their workload permits, they can add the identified task to their task queue. Otherwise, they send the team leader a message about the newly identified task. The team leader is then responsible for assigning the task to another agent. There are two ways the leader can assign tasks: *role-based assignment* and *exhaustive assignment*. In the role-based assignment, the leader will send requests to agents who are filling roles that can perform the task. If a member accepts the request, the task is assigned to that agent; otherwise, the leader resorts to *exhaustive assignment*. The task assignment request is broadcast so that any agent in the environment can receive the request. If this assignment does not succeed either, then the task is re-queued in the leader's list and retried for assignment later.

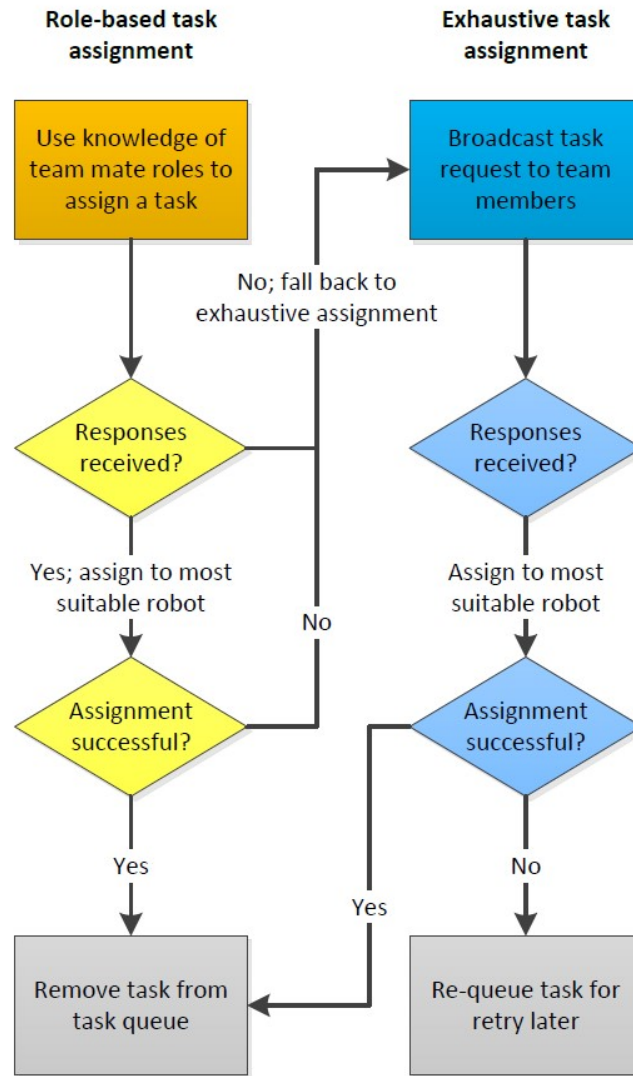


Figure 2.2: Role-based task assignment and exhaustive task assignment in Gunn's framework. Adapted from Nagy [2016].

The whole process of task assignment is completed in a series of messages exchanged between the leader and the potential assignees. At first, the leader sends the task request to the team members (in case of role-based task assignment) or all agents (in case of exhaustive task assignment). Assignees respond with their suitability if they meet the minimum requirements to perform the

task and have room in their workload, otherwise, they reject the task. The leader then sends a task assignment confirmation message to the agent with the highest task suitability. The selected assignee then adds the task to their queue and sends a task acceptance message back to the leader. The leader then removes the task from its queue. After completion of a task, the agent reports the results back to the team coordinator. If, in any case, the robot is damaged or lost before completing a task, the task is left to be rediscovered.

2.4.2 Active Recruitment Controller(ARC) Framework

Extending Gunn’s framework [Gunn, 2011], Nagy [2016] introduced the idea of recruiting new skills to the team. In Gunn’s framework, the only way of adding new members to the team was through mergers and redistributions during encounters. Nagy [Nagy and Anderson, 2017, 2016; Nagy, 2016] introduced other ways of adding new members to the team. These new members could be from other teams or new replacements released into the environment.

- **Recruitment Modes:** The recruitment modes characterize the effort by an agent to recruit a robot. It can range from *passive* to *concurrent* to *active* recruitment.

In *passive* recruitment, there is no effort spent on the recruitment. The recruitment depends on chance encounters. This is already implemented in Gunn’s framework [Gunn, 2011; Gunn and Anderson, 2013, 2015] where agents can encounter, merge and redistribute team members while doing other tasks in the environment (Details in Section 2.4.1). A little more active recruitment is

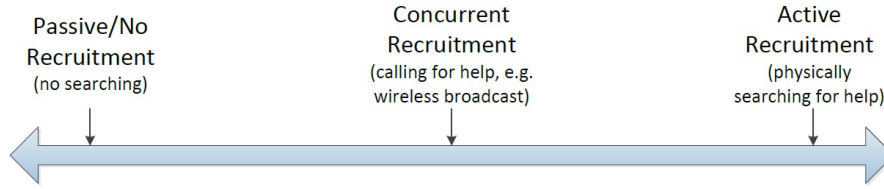


Figure 2.3: The Recruitment Modes Spectrum in ARC Framework. Adapted from Nagy [2016].

concurrent recruitment. In concurrent recruitment, an agent can broadcast requests for recruitment concurrently while doing its own tasks. This increases the chances of recruiting robots while completing the tasks at hand. Lastly, there is *active* recruitment where an agent is assigned a recruitment task to actively recruit agents. The priority of the recruitment task depends on the type of recruitment: role-level or task-level (Details in Section 2.4.2). Role-level tasks are of the highest priority, and so an agent assigned to this task drops other tasks to actively perform role-level recruitment. The priority of task-level recruitment tasks is the same as the priority of the task being recruited for. The assignee drops whatever it was doing and focuses on physically exploring the environment, looking for recruitment. This increases the chances of finding a suitable recruit as the agent is actively searching; however, the agent is leaving important tasks incomplete to perform recruitment.

Currently, the mode of recruitment used by agents in the mission is pre-specified at the beginning of the mission. The performance of the teams varies based on different factors like communication quality, replacement availability, failure rate, etc. Moreover, the robot failures simulated to evaluate the ARC framework were limited to temporary or permanent robot losses. I evaluate my agent

recovery framework under different conditions, in addition to more diverse robot failures. One of the conditions is the different methods of recruitment. For evaluation, I aim to see how well my recovery framework performs with different recruitment strategies in the presence of more realistic robot failures. In future, this may help develop a more dynamic recruitment strategy where agents can switch recruitment modes according to their needs.

- **Recruitment Strategies:** There can be situations where an agent needs assistance with some task due to work overload or inability to perform the task. The team can add new members, but there is a limit to how many agents can be added to the team. There may be a need to acquire a skill temporarily without incurring the cost of managing an additional team member. There may also be cases where a high-importance role in the team is underfilled or vacant and requires it to be filled permanently. The ARC framework addresses this idea of temporary or permanent recruitments through task-level recruitment and role-level recruitment, respectively.

In task-level recruitment, agents can temporarily recruit agents from inside or outside the team to perform a specific task. During the discovery of tasks or getting assigned tasks by the leader, if the agent's task queue is full or if it does not have the ability to perform the task, it will offer to recruit someone else for the task and add the task to its task recruitment list if the leader cannot find anyone else that accepted to perform the task themselves. No intervention from the leaders is required to accept or recruit for task-level recruitments. The external agent helps with the task and goes back to doing tasks for its own

team.

In role-level recruitment, an agent is permanently added to fill a role in the team. A team will always strive to maintain its ideal or desired team structure (Section 2.4.1). Similar to *role switch checks* in Gunn’s framework (described in Section 2.4.1), Nagy [Nagy, 2016] introduced a ***Missing role check***. The differences are that the role switch check is done more frequently (every 30 seconds) and by all members of the team. However, the missing role check is only done by the team leader and less frequently (every 6 minutes). During this check, the leader identifies if any role is underfilled or under-skilled. After the check, the leader assigns a member with a task to do role-level recruitment for the most important under-filled role.

The ARC framework’s recruitment strategies allow temporary or permanent recruitment to make up for lost robots. However, without a recovery mechanism in place, it is possible that malfunctioning robots are being recruited for tasks or roles. For instance, a team could be recruiting a firefighting agent with damaged fire sensors. My framework updates the current attributes based on its health status, which accounts for any damage or change in its system. Eventually, the changes in task and role-suitability are synced with the team’s knowledge during subsequent self-announcements, role checks, and so on. This allows for more efficient recruitment of skills. I will discuss this in detail in Chapter 3.

2.4.3 Fiawoo’s Framework

Fiawoo [2019] introduced the concept of robot satisfaction, robots’ sense of locality

and robots' knowledge of rare skills to Gunn's framework [Gunn, 2011] and the ARC framework [Nagy, 2016] to improve the performance of the robots.

- **Robot Satisfaction:** Robot satisfaction refers to the utility of the robot in the team or a location in the environment. For instance, suppose a robot team is exploring an area with no debris. A debris-removing robot on the team will not find a debris-removing task in the location and must resort to performing tasks trivial to its role, like exploring the environment. This will result in low satisfaction in the robot. This is because the special skill of removing heavy objects is not being used in that team or in that location. This skill could be better used in other locations or teams in the environment. This satisfaction level eventually leads to the willingness of the robot to be recruited by other teams or become independent. In terms of location, the robot will tend to stay at the location where it has high satisfaction, meaning its skills are being put to good use. Fiwoo's framework supports two kinds of satisfaction tracking: 1) a robot tracking its own satisfaction, and 2) a leader tracking its team members' satisfaction.

For individual satisfaction tracking, calculation of satisfaction involves the weighted number of completed tasks over time, where a heavier weight is given to tasks that require rare skills, future satisfaction from upcoming tasks in the task queue, and the number of tasks rejected to continue with the current team. A satisfaction threshold is used to determine whether a robot should accept or reject recruitment requests. Three willingness states are used to discretize the amount of effort a robot will expend to improve its satisfaction value: *not*

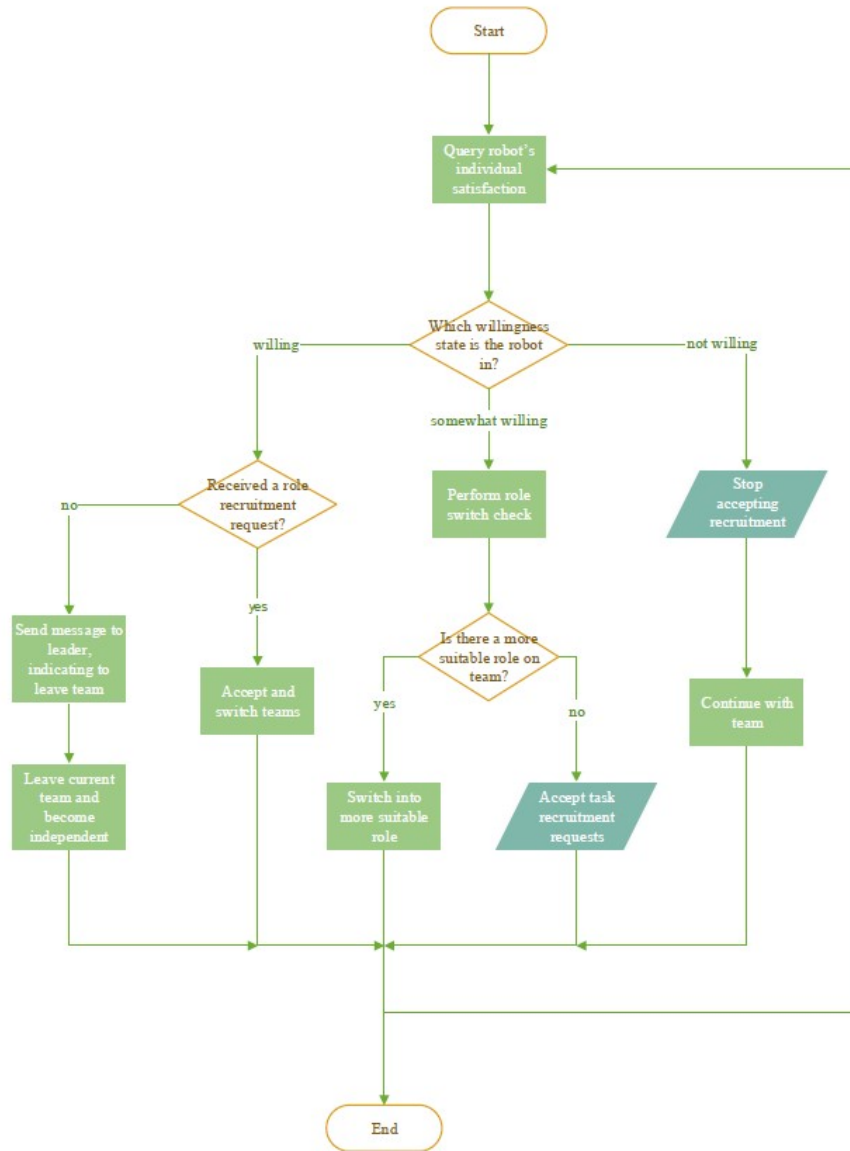


Figure 2.4: A robot's decision-making process based on its current satisfaction. Adapted from Fiawoo [2019].

willing, somewhat willing, and willing.

If the robot's skill is rare or if its satisfaction value is well above the satisfaction threshold, its willingness state is set to *not willing*. If its skill is rare, it will be available for temporary or task-based recruitment for tasks that require its rare

skill. Otherwise, it will only perform tasks that use its rare skill within its own team. The willingness state is set to somewhat willing when the satisfaction value lingers around the satisfaction threshold. In this state, the robot will be more open to task-level recruitments or occasional role switch checks to improve its satisfaction. When the satisfaction falls below the satisfaction threshold, the robot starts taking a more aggressive approach in improving its satisfaction, which includes accepting role-based recruitment by other teams. While accepting a role-based recruitment request, the robot first lets its team leader know that it is leaving the team. The leader determines if it wants to let go of the team member by determining how useful it is to the team. If it is not useful anymore, it lets go of the team member, otherwise, it tells the team member to reject the recruitment request and set its willingness to *somewhat willing*. This helps maintain the ideal team structure since the leader has a more complete picture of the team than the individual member, and would know better if the robot is still useful to the team. Figure 2.4 shows the decision-making process of an agent based on its satisfaction.

The leader of a team tracks the satisfaction of the team members based on the number of tasks the member has completed within the team. If a team member repeatedly rejects tasks from the team leader, the leader will deem the member not useful to the team and will eventually let them go. Then the leader can recruit a new member for the vacant role.

- **Rare Skills:** Loss or damage of robots can lead to a lack of specific skills in the environment. With the high demand for rare skills, the agents should be

able to choose to accept or reject tasks depending on the rarity of their skills. The rarity of a skill is determined by the number of tasks an agent receives that require a particular skill. In addition, the number of teams requesting the skill is also tracked. If multiple teams are requesting the same skill, then it may indicate a global shortage of that skill. For example, suppose there is a lack of firefighter robots in the environment. Once an agent knows that its skill is rare, they use it along with their satisfaction to decide whether they want to accept task-level recruitments (as discussed in Section 2.4.3). The rare robots will keep checking if the skill is still rare. If it is no longer rare (maybe because of enough new replacements being released into the environment), the rare robot can then join a new team or its old team.

- **Robots' sense of locality:** Having a sense of locality will help the robots to identify where their skills are being put to good use. It will help with decisions like whether to move with the team or stay with the current location. Staying at a location for too long can result in the agent being detached from its team. It can also cause the satisfaction level to decrease as it completes more tasks in that location. When no more tasks are left for it to do, it can move around and even join new teams or rejoin old teams. A sense of locality will also allow the robots to identify locations they have already visited and move to unexplored areas.

2.5 Conclusion

In this chapter, I have discussed some research done on Multi-Robot Systems in search and rescue operations, how robust the existing strategies are to robot failures, studies on fault detection, diagnosis, and recovery of robots, and finally, the existing frameworks for heterogeneous teams of robots in dangerous environments that are pivotal to the development of my agent recovery framework. I will now discuss the concepts of the agent recovery framework in Chapter 3.

Chapter 3

Methodology

3.1 Chapter Overview

In this chapter, I discuss the main concepts of the Agent Recovery Framework that I developed, which is the main contribution of my thesis. My framework is an extension of prior research work done at the Autonomous Agents Lab at the University of Manitoba, on different concepts around Multi-Robot Teams in USAR. It includes Gunn’s Framework that addresses team and task management in heterogeneous teams of robots in USAR [Gunn, 2011], Nagy’s Active Recruitment Controller (ARC) framework that supports different recruitment modes and strategies [Nagy, 2016], and finally Fiwoo’s Framework that supports management of rare skills based on robot satisfaction [Fiawoo, 2019]. The core concepts of the existing work are discussed in the Related Work Section 2.4.

The Agent Recovery Framework addresses detection and diagnosis of robot failures in heterogeneous teams of robots, and appropriate recovery strategies to recover from

failure while minimizing the effect of failure on the overall performance of the team. I describe my framework in 3 sections: Detection, Diagnosis, and Recovery. In Section 3.2.1, I discuss how failures are detected in robots, the different types of failures that are detected, and how these failures can affect the mission. In Section 3.2.2, I discuss different kinds of diagnostic tests or workflows that are run to monitor robot health. Finally, in Section 3.2.3, I discuss a 3-phase recovery task that helps a robot and its team recover from robot failures.

3.2 Agent Recovery Framework

Robot failures are unwanted but frequently unavoidable events. Software or hardware failures in robots can be caused by human error during robot development, external force, etc. Robots can also output incorrect sensor readings due to smoky environments, lack of light, uneven terrains, etc. Ideally, we want our robots to function perfectly all the time. However, this is not something we can expect in the real world, especially in an unpredictable, dynamic, and dangerous domain like USAR. Robots are at a higher risk of failures, such as sensor failures due to fire or water, getting stuck on debris, getting hit by falling debris, and so on. In the case of teleoperated robots, it is much easier for humans to assess the situation and take control. But, for autonomous teams of robots, it is more challenging for robots to navigate, plan, and perform tasks safely. It is also very important for robots to function with as much efficiency and accuracy as possible due to the critical nature of rescue missions. Studies show that the human survival rate without food and water is 90% in the first 24 hours after the disaster. It drops to only 5%-10% after 72 hours [Hakami et al.,

2013]. Robot failures can greatly affect the mission progress. Even if robots seem to be operating and performing tasks, malfunctioning sensors, actuators, servos, etc., can not only delay the tasks of the robot itself, but also those of its team. For instance, a robot that has faulty victim sensors will incorrectly identify victims and record the incorrect location of victims. This can later delay the rescue process. Hence, the recovery of robots holds high importance in multi-robot rescue missions in USAR. It is hard to guarantee full autonomous recovery of robots, but measures can be taken so that the malfunctioning robots do not hamper the team's progress any further. In fact, it is possible to have the malfunctioning robots continue to contribute to the missions using the robot components that are still functioning. For example, a robot that lost its mobility can still help in relaying messages, recruiting robots, etc.

Existing studies (as discussed in Chapter 2) show the development of frameworks and strategies to facilitate rescue missions in disaster sites using teams of robots. These studies take different approaches in addressing the various challenges, like control and coordination of robots, task allocation, and so on [Drew, 2021]. As discussed in Chapter 2, many of these works address the robustness of this development against robot failures and other challenges in the environment (e.g., poor communication). Gunn's framework facilitates team and task management of heterogeneous teams of robots [Gunn, 2011]. It has periodic role checks to check for missing robots on the team, where missing robots can be dead robots or robots that have strayed away from the team. To minimize the effect of the missing robots, important but vacant roles are temporarily filled with the next best-suited team member. This framework also allows team merges and redistribution to bring in new members to the team to meet

the desired team structure. Nagy's ARC framework extended Gunn's framework to allow different modes (active, concurrent, and passive) of temporary and permanent recruitments to bring in new members to address vacant roles, skill or resource shortage on the team [Nagy, 2016]. Fiawoo further added robot satisfaction metrics to better manage rare and unique skills in the environment, which ensures efficient use of rare skills in case of skill shortage due to robot failures [Fiawoo, 2019]. There is also a number of studies on building failure detection, diagnosis, and isolation methods that alleviate the effect of robot failures to some extent by detecting and isolating failed robots. However, very few consider the various kinds of robot failures that may be approached differently. They lack elaborate failure-specific recovery of failed robots. In addition, very few studies address how the failed robots should be managed until they are retrieved and repaired by human rescuers.

There can be various kinds of robot failures: partial or total, permanent or temporary, software or hardware, etc. Totally damaged robots often suffer from critical failures that make it impossible to perform any task. These robots can be considered dead, and there is no way to recover themselves without external help. It may be possible for the human rescuers to retrieve information and fix or salvage unaffected parts from the robots. This requires proper detection and tracking of these damaged robots. Partially damaged robots suffer from failures that only affect certain components. The robots may still be able to perform other tasks that do not require affected components. In fact, in any mission, it is normal for robot parts or components to wear out over time. The condition of robot wheels at the end of a mission may not be as good as when the mission started. Similarly, some partial failures may not have

a significant effect on robot tasks. Robots can still use the affected components to perform tasks, but with slightly lower efficiency. The permanency of failures depends on how long the failure persists. There can be sudden one-time malfunctions or system errors that may not affect future tasks. Monitoring the affected component can confirm if the error is persisting and needs recovery. In some cases, a simple restart, recalibration, or repositioning can fix the error. For instance, robot sensors can give incorrect readings in a smoky room. Moving away from that smoky room can confirm that the sensors are working. Failures can also occur due to software issues or hardware issues. Hardware issues are often permanent in nature and may require human assistance to get fixed. Careful management of malfunctioning robots is of immense importance in rescue missions where time is of the essence and resources are limited. Incorrect decisions or outputs from malfunctioning robots can degrade a team's performance and hamper the rescue mission. While detecting and isolating these robots will reduce their effect on the team, it will also lead to a resource shortage. Dead or dying robots can have critical information that can be lost, which is a waste of time and resources.

In my Agent Recovery Framework (see Figure 3.1), I address various kinds of robot failures, the detection, diagnosis, and recovery management of failed robots and their teams. Failures are detected when some component becomes unresponsive, or readings cross some threshold (e.g., becoming overheated), or through the periodic diagnosis of robot health. The latter helps in tracking a robot's health status. If any issue is detected during the diagnosis, a recovery task is created for the affected component accordingly. Different types of failures can require different recov-

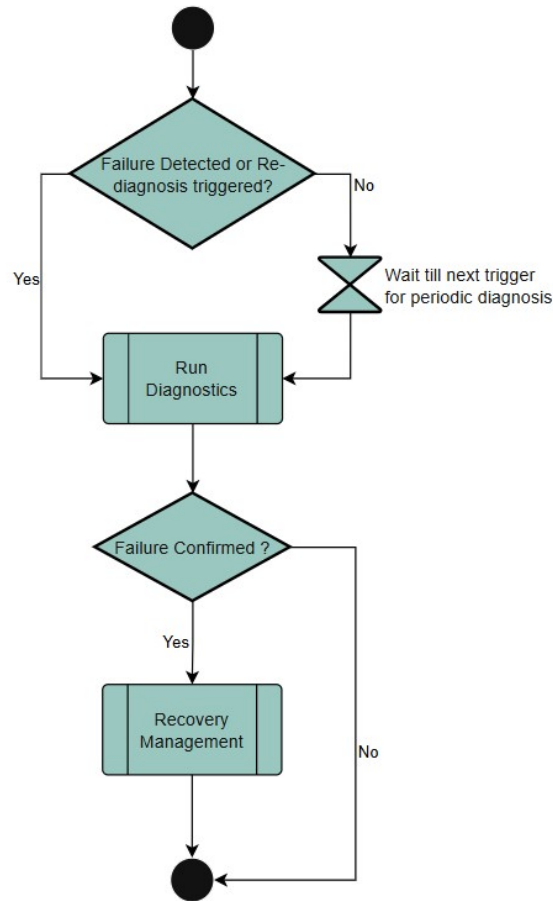


Figure 3.1: An overview of the Agent Recovery Framework. Each component will be explained in detail in the following sections.

ery strategies. Some failures may be self-recoverable, and robots can continue doing tasks as usual after recovery. Some failures may be more permanent but isolated to only a part of the robot. The attributes, roles, and tasks of such robots are readjusted such that the robot only performs tasks that are not affected by its failures. Finally, there are failures that are not recoverable at all and have a significant effect on the robot's performance. Such failures require human assistance. The recovery task in my framework supports all these scenarios. My framework is only a pre-designed guide or template that aims to address the challenges and concerns in agent recov-

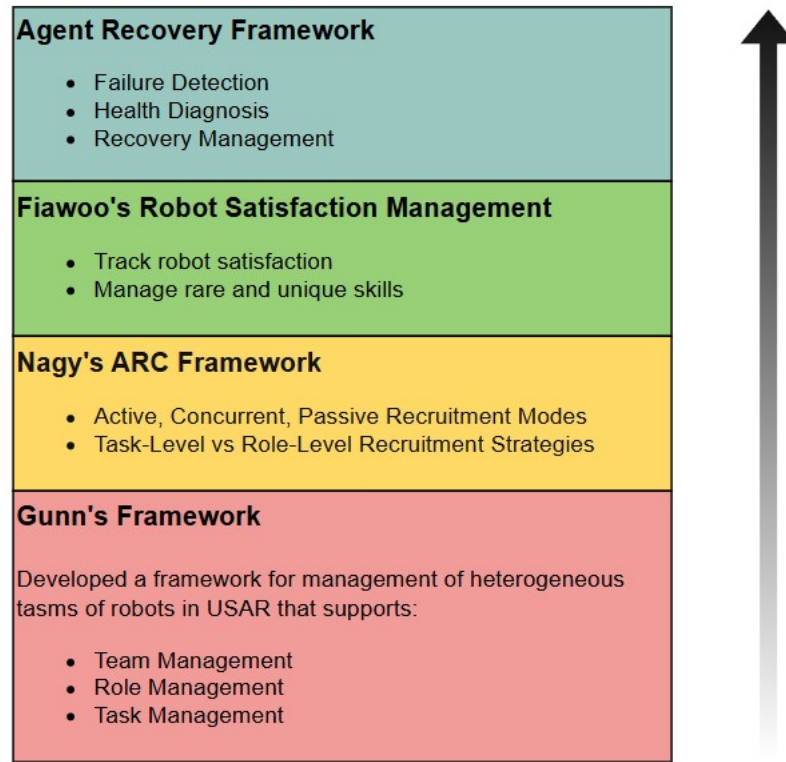


Figure 3.2: An overview of existing work around Autonomous Heterogeneous Teams of Robots in USAR, in the AALAB at University of Manitoba, in addition to my Agent Recovery Framework.

ery and team management in dangerous domains like USAR. I have generalized and abstracted away the specifics of detection, diagnosis, and recovery, which can be tailored to fit different use-cases in future. For the purpose of my thesis, I am expanding on Gunn's [2011], Nagy's [2016], and Fiawoo's [2019] work on autonomous teams of robots in USAR (see Figure 3.2).

In the following sections, I will discuss the concepts of failure detection, diagnosis, and recovery management of failed robots in detail.

3.2.1 Failure Detection

The prerequisite to robot recovery is, firstly, being able to detect the failure in the robot. Failure detection is a part of the overall recovery management. Failures can be detected based on pre-defined conditions while monitoring robot health, looking out for deviations of actual behaviour from expected behaviour, etc. In this section, I discuss the strategies and ideas I use for failure detection in my framework.

Evaluate Fail Rate

I used a *fail rate* metric to quantify the severity of failures detected in robot components. The robot's health is monitored over a certain period of time during health diagnosis to increase the accuracy of failure detection. The number of failures or faulty readings detected over this time period is the *fail rate* of the component. For instance, if a robot runs a diagnosis on its sensors for 2 minutes, and detects incorrect readings 50% of the time, then we can say the fail rate of the sensors is 50%. The incorrect sensor readings in my implementation are detected by determining whether the readings are within the expected field of view of the sensor (See 3.3). This is a very simplified method of sensor failure detection. Real-life sensor failure detection can be far more complex (more in section 3.2.1).

Detection methods by failure categories

To determine the steps required for recovery, it is important to know what kind of failures are being detected and how or what conditions are being monitored for the detection of failures. Robots are complex machines with numerous possibilities

of failure. The risk of failures is higher in unpredictable domains like USAR, where robots are exposed to danger. The robots used in rescue missions are often built at low cost to increase their expendability in the missions [Sharmin et al., 2019]. Heavy-built robots are not suitable for sensitive disaster zones. So naturally, robots in USAR missions have a higher risk of failure. Heterogeneity in teams of robots increases the number of possible failures that need to be considered while designing a recovery framework. With challenges like poor communication, uneven terrain, lack of time and resources, limited energy, etc., it becomes difficult to keep track of all robot failures by robot components. So, I used two broad categories of failures that my framework supports: *faulty readings* and *unresponsive* failures.

Faulty readings are often difficult to detect if the failure rate is too small. These may go unnoticed as they do not always trigger any alarm or hamper tasks, but harm the robot's performance in the long run. Robots still perform tasks, but with lower accuracy, which is hard to detect if not actively monitored. This is why these kinds of errors require routine or periodic diagnosis. Faulty readings can be expected from robot components that provide readings or outputs, like sensors. For example, a sensor can incorrectly detect an object at 2 meters away when the object is actually 3 meters away. The failure detection of sensors can differ based on the type of sensors and use case. Some common sensor failure detection methods include analyzing trends and patterns in sensor data over time to detect deviation from normal operating conditions [Sabry and Ungku Amirulddin, 2024], using multiple sensors to cross-check and confirm errors [Solórzano et al., 2017], using AI algorithms to analyze historical data and detect anomalies [Seçilmiş et al., 2023], etc. Sensor failure detection is a

broad topic of research and out of scope for my thesis. My focus is on managing the recovery of failing robots rather than the specifics of the failing components. For simplicity, I use a mix of rule-based and threshold-based detection methods to detect failures. I use the field of view (FOV) of a sensor to get the extreme points of the FOV with respect to the position of the sensor on the robot's body. If a sensor provides the position of objects outside the expected FOV, it satisfies the condition for a faulty sensor reading. Faulty sensor readings in real-world applications can arise from physical damage that alters the sensor's position or orientation, from particles obstructing its field of view, from thermal expansion or contraction of sensor materials due to temperature changes, or from various other environmental factors [Gao et al., 2015]. In my implementation, the area of FOV depends on the range of sensors and the position of the sensor on the robot's body. Figure 3.3 shows the FOVs of sensors in a MaxBot(a robot that I used in my implementation). Here, sensor B can detect Wall A but not Wall B. If it provides any object location outside this FOV, there may be something wrong with that sensor. The physical attributes, including the types of sensors for the robots used for my implementation, will be described in Chapter 4.

Some failures can be easily detected when a robot component becomes unresponsive, or provides no output, or the readings cross a certain threshold. When such failures are detected, it triggers a health diagnostic test that tests all components of the robot and confirms the type and source of error in the system. Unresponsive failures stem from the components that stopped working or are not providing any output. These failures are often easier to detect as they are noticeable right away when a robot fails on a task while trying to use the unresponsive component. For ex-

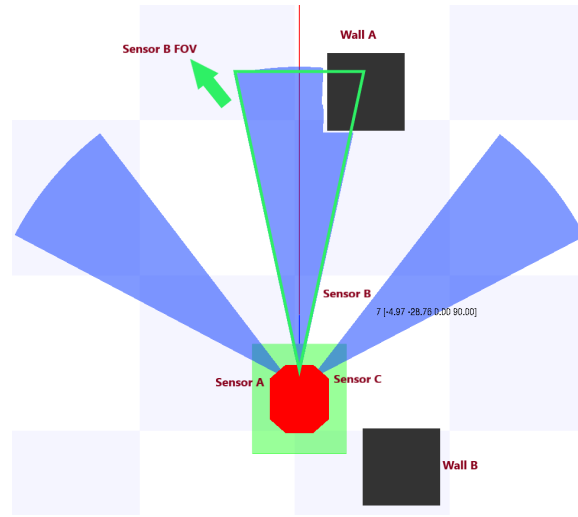


Figure 3.3: FOV of sensors in a MaxBot.

ample, if a robot's sensors become unresponsive, it will get system errors while trying to access the sensor, and the failure can be detected right away. In such a case, the failure rate of the sensor is 100%. A system diagnosis is then triggered to confirm the error and to initiate recovery if necessary. Another example of unresponsive failures includes loss of mobility. This may or may not be detected right away. However, I added diagnostic tests to monitor the location of the robots. If the robot position doesn't change over a certain period of time and if it is not at its expected destination location, we can assume that it is either stuck on debris or has lost mobility. Other examples of unresponsive failures include unresponsive network drivers or WiFi, low power pack power level, sudden death (total damage), etc. A low power level is not exactly an unresponsive failure, but it is an indication of future unresponsive failure if the power pack is exhausted.

3.2.2 Robot Health Diagnosis

Failure detection and diagnosis go hand in hand in my framework. The process of diagnosis involves running targeted diagnostic tests on the main robot components, like sensors, wheels, WiFi, etc. While detection is the process of detecting failures in robots, diagnosis of the affected component gives us a better idea about the failure, such as the affected components, tasks, the severity of the failure, etc. This helps us make better, more informed decisions during recovery.

Periodic Diagnosis and Triggered Diagnosis

My framework supports 2 types of diagnosis based on how frequently the robot's health is diagnosed: *Periodic Diagnosis* and *Triggered Diagnosis*.

Periodic diagnosis of overall robot health is scheduled to run at certain intervals. For my experiments, periodic diagnoses are run every 8 minutes and each trial is 30 minutes long. A future direction of this research can include experimenting with different intervals to see what works best with different scenarios. Ideally, it may be preferable to monitor robot health at all times. However, in a search and rescue scenario, the robot should spend more time and computing power on the mission, rather than on diagnosing itself all the time. In my diagnostic test, every essential robot component is tested separately. Even though the diagnostic tests in my implementation seem simple, in the real world, these can be more complex. Periodic diagnosis is suitable for detecting failures that do not trigger alarms or thresholds. For example, inaccurate readings from sensors may go unnoticed without direct monitoring of the sensors.

Triggered diagnosis is run when any component becomes unresponsive (e.g, WiFi not working) or when any pre-defined condition (e.g., a threshold) of warning becomes true (e.g, power pack power level below 5%). The diagnostic tests can also be triggered by components as re-diagnosis for recovery, to check if the attempted self-recovery solved the problem or failure. If re-diagnosis shows that the failure still persists, alternative recovery measures are taken to address the issue.

Diagnosis Workflow

There is a diagnosis workflow (See Algorithm 2) dedicated to every component in a robot. These diagnosis workflows are executed and managed by the health manager of the robot. The same workflows are run for both periodic and triggered diagnosis. The main logic for the workflows is similar for all components. These workflows are usually run in the background while doing other tasks. However, for re-diagnosis, it runs as part of the recovery task when the task is being performed.

The diagnostic test is a part of the diagnostic workflow. It is run if the component is still working or active. Reading or feedback is monitored for a certain period of time to get the *fail rate* of the component. This test is different for different components. For example, to test sensors, the readings of the sensors are evaluated. Sensor testing in my framework requires the presence of objects in the surroundings, which helps with the testing. To check mobility, the change of position of the robot is tracked. A workflow state is maintained in the diagnosis workflow to preserve the ongoing diagnosis state and avoid redundant invocation of the workflow. The diagnostic tests can be substituted with more complex and better ones in real life. If the fail rate

Algorithm 2 Diagnosis workflow

Require: RT = The recovery task that triggered this workflow. Indicates re-diagnosis if not null.

if Robot has component C **then**

if ((Periodic timer triggered OR Threshold triggered OR Re-diagnosis triggered)

 AND $WorkflowState == IDLE$) **then**

$WorkflowState = ACTIVE$ ▷ Initiate diagnosis

else if $WorkflowState == ACTIVE$ **then**

if Component is Unresponsive **then**

$WorkflowState = RUN$

else if Ready for Diagnostic Test OR Re-diagnosis Triggered **then**

$WorkflowState = RUN$

 Initiate Diagnostic Test

end if

observed from the diagnostic test is worse than the related *Attribute Fail Rate*, a recovery task is created for the robot to recover from this failure. This makes sure that the robot's health status is up to date if the failure occurs multiple times. This also gives the robot a chance to re-attempt recovery if the condition gets worse.

3.2.3 Recovery Management

Once the initial diagnosis is completed, the agent uses the result from the diagnosis to decide whether recovery is necessary. In my framework, the diagnosis result mainly refers to the average failure rate of the robot component. If the average failure rate

Algorithm 2 Diagnosis workflow; *Continued*

```

else if WorkflowState == RUN then

    if Diagnostic Test Not Complete then

         $FailRate \leftarrow RunDiagnosticTest()$ 

         $TotalFailRate+ = FailRate$ 

         $NumberOfTestInvocation+ = 1$ 

    else

        if Component Responsive then

             $AvgFailRate = TotalFailRate / NumberOfTestInvocation$ 

        else

             $AvgFailRate = 100.0$ 

        end if

    end if

    if  $AvgFailRate > Attribute\ Fail\ Rate$  AND Dedicated recovery task
    doesn't already Exist then

        Create Recovery Task for Component

    end if

end if

end if

```

is greater than the related *Attribute Fail Rate*, then the robot creates a recovery task for the affected component.

Attribute Fail Rate

Gunn [2011] used robot attributes to identify the types of tasks a robot can perform (see Section 2.4.1). Each attribute has an associated value which is pre-defined. In my framework, the attributes are more dynamic and allow tracking the health of the component that relates to the attribute. I use a metric, *Attribute Fail Rate*, which refers to how often a robot component related to an attribute fails or how well it works. It is associated with the attributes related to each robot component. For example, a robot with the attribute *HasFireSensor* can have a *value* true and a *Attribute Fail Rate* of 0.5 if its fire sensors are expected to fail 50% of the time. The attribute fail rate is calculated from the average failure rate calculated during the diagnosis of a robot part or component. Initially, all robot components start with a attribute fail rate of 0, meaning they are in a perfect state. The attribute fail rate gets updated if the average failure rate identified during the component diagnosis is greater than the attribute fail rate.

Recovery Task

The recovery task (see figure 3.4) can be self-executed by any robot. It is not transferable, and it is a high-priority task. This is because it is preferable to resolve any issue with the robot as soon as possible rather than having to deal with potential aftereffects from the failure. The longer the robot is allowed to operate with faulty components, the higher the chances of failing tasks. Moreover, incorrect readings or feedback from the malfunctioning robot may be shared with other agents in the environment. This incorrectly shared knowledge or misinformation becomes hard to

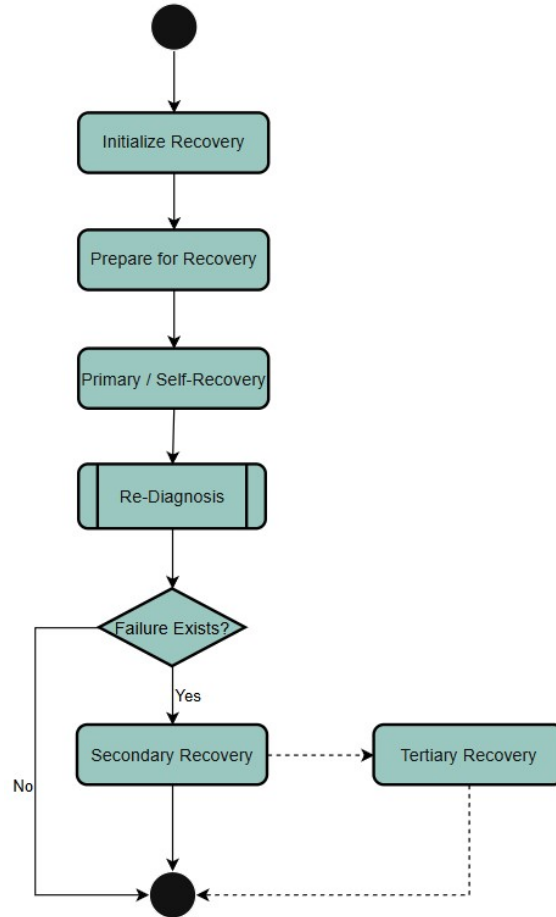


Figure 3.4: Overview of recovery task.

track and will hamper the progress of the mission. The recovery task is executed in 3 phases: primary recovery, secondary recovery, and tertiary recovery. The execution of these 3 phases depends on whether the preceding recovery steps (if any) have resolved the failure. Each of the 3 phases is described in detail as follows:

- **Primary Recovery:** In the primary recovery phase, the agent attempts to resolve the failure by itself. Robots will try to self-repair by restarting, recalibrating, etc. These steps may resolve any corrupted files, drivers or system

issues, readjust sensors or wheels, and so on. Self-repairing may also include moving out of the current location, changing positions, etc. This helps in ruling out cases where the environment might be the cause of the failure. For example, sensors might give incorrect readings in a smoky room. Moving out of the smoky room might help confirm that there is nothing wrong with the robot, but the environment. Due to the nature of recovery, the robot may pause its ongoing tasks during primary recovery and resume when primary recovery is complete. The primary recovery phase acts as a first-aid step that aims to rectify the robots' situation, without making any permanent changes to the robots' capabilities or requesting human assistance for repair. While there is no guarantee that the primary recovery will be successful, it is still worth taking the chance since it may fix the issue, and the robot can keep doing its expected tasks as usual. The completion of primary recovery invokes the diagnosis workflow for the affected component. This step can also be referred to as re-diagnosis. The purpose of re-diagnosis is to run the diagnostic test again to confirm if the primary recovery has resolved the issue. If the failure is resolved, then the recovery task is complete. Otherwise, the secondary recovery phase starts.

- **Secondary Recovery:** In the secondary recovery phase, the agent reassesses its current attributes, roles, and the fail rate of the malfunctioning component. The attribute fail rate is updated with the average fail rate that we get from diagnosing the related failing component. Consequently, the current role suitability is reassessed and updated. In Gunn's framework [Gunn, 2011], a role is a set of tasks expected to be done by an agent occupying that role. The role suit-

ability of an agent is calculated by summing the weighted task suitabilities for that role (see Section 2.4.1 and Algorithm 1). Task suitability is calculated from the suitability expression associated with the task (see Section 2.4.1). Originally, this calculation was based mainly on the attribute values. In my framework, I factor in the fail rate of the attributes as well to calculate the task suitability. Let us revisit the fire quenching task suitability example from Section 2.4.1. Let conditions *CanQuenchFire* = *true* and *HasFireDetector* = *true* be represented by C_1 and C_2 , and their weights be W_1 (70) and W_2 (30), respectively. Let the condition *Mobility* > 0 be C_0 . Then, the expressions can be rewritten as:

$$M = C_0 \text{ AND } C_1 \text{ AND } C_2 \quad (3.1)$$

$$S = C_1[W_1] \text{ AND } C_2[W_2] \quad (3.2)$$

Here, M is the minimum requirements expression, and S is the suitability expression. According to Gunn's framework, the suitability would be calculated as:

$$S = W_1 + W_2 \quad (3.3)$$

If the logical operator between conditions was *OR*,

$$S = MAX(W_1, W_2) \quad (3.4)$$

I modified these equations by factoring in the attribute fail rates associated with the conditions. Let the attribute fail rate for the condition *CanQuenchFire = true* be F_1 . This refers to how often the fire quenching component (e.g., water gun) fails. Similarly, let the attribute fail rate for the condition *HasFireDetector = true* be F_2 (i.e., fail rate of fire sensors). Then, the suitability is calculated as:

$$S = (W_1 * (1 - F_1)) + (W_2 * (1 - F_2)) \quad (3.5)$$

If the logical operator between conditions was *OR*,

$$S = MAX(W_1 * (1 - F_1), W_2 * (1 - F_2)) \quad (3.6)$$

The expression $(1 - F)$ tells us how well we can expect the component to perform. The lower the attribute fail rate, the higher the suitability to perform the task. The agent calculates the task suitability of each task expected from its current role and then calculates the role suitability using Algorithm 1. In Gunn's framework, team members announce themselves periodically (e.g., every 15 seconds) to keep the team's knowledge up to date. This announcement includes the team members' role suitability. During the role switch checks, the

team members evaluate their current role and suitability and switch roles if they can find roles that they are better suited for (Details in Section 2.4.1). Eventually, the attribute fail rates or the new suitabilities will be factored in for all future decisions, like role and task assignments and recruitments. For example, a firebot with damaged water guns will be more suitable for explosion tasks than fire-quenching tasks. As another example, if firebot A has fire sensors with attribute fail rate 0.5 and firebot B has fire sensors with attribute fail rate 0.1, and if both firebots are bidding for the same fire-quenching task during task-level recruitment, firebot B will be assigned the task since it has higher suitability(or lower attribute fail rate). This way, malfunctioning robots can keep contributing to the team's goal with minimal effect on the team's performance.

Tertiary Recovery Preparation: Not all failures are recoverable using attribute, role, and task reassessments. These failures affect a robot's overall functionality. For instance, with damaged WiFi, a robot cannot share information with other robots and cannot receive or send role or task requests. Again, a robot becomes completely useless if its power level depletes or it suddenly dies. Such critical failures are more difficult to recover from. They often require human intervention to get them fixed. In case of such failures, the secondary recovery phase prepares the agent for tertiary recovery. For example, for WiFi failure, a task is created for the robot to go back to its starting point for repair by humans. For loss of mobility, initially, the robot asks for help from other robots to help it get unstuck from debris. But if it is still stuck in one place

after a long time, it is assumed that there is something wrong with its motors or wheels. In such cases, the immobilized robot records its current location and announces this location with the periodic announcement. This lets the other teammates know about its location, which can be used by humans later for recovery. While being stuck in a position, it continues to perform tasks that do not require it to move, like sharing knowledge, recruiting members, etc. For a depleted energy level, the agent can't attempt any recovery if it is already dead. So, a threshold is set that triggers recovery if the energy drops to that threshold level (e.g., threshold 5% energy). This gives the agent enough time and energy to prepare for tertiary recovery. In my framework, when the power level of an agent reaches a critical point or threshold, the agent switches to power-saving mode.

- **Tertiary Recovery:** Tertiary recovery is mainly intended for the human rescuers. The goal of the tertiary recovery phase is to address and resolve issues that cannot be fixed without human intervention. The secondary recovery phase includes some preparatory steps for the tertiary recovery phase to enable the human rescuers to track and retrieve these malfunctioning robots. These preparatory steps also include information retrieval from malfunctioning robots to reduce information loss.

Once the preparation for the tertiary recovery phase is complete in the secondary recovery phase, the malfunctioning robots are then ready for repair or retrieval by humans. In some cases, the malfunctioning robots can go to the human rescuers by themselves for recovery. For example, when the WiFi of a robot is

malfunctioning, a task is created in the secondary recovery phase to send it to the starting point where the humans are. The robot finds its way back to the humans, gets repaired, and gets released back into the environment. In other cases, where robots cannot move (e.g., damaged wheels, low power level, etc.), the location of the robot is shared with other agents. So, even if humans cannot directly access the damaged robots right away, they can get informed about the location of the damaged robots from other robots in the environment. The humans can then retrieve the robots from those locations just as they would rescue victims in the environment. After retrieval, humans can fix and re-release or release substitutes in the environment. Even if humans do not retrieve these robots physically, there is a lower chance of information loss since these damaged robots may have already shared critical information with other robots in the environment. So, even if the robots get lost or are dead, their efforts so far in the mission are not in vain.

Tertiary recovery is not available right away in USAR missions. There may be communication challenges and other environmental challenges that might make it difficult for humans to teleoperate and control the robot. Moreover, human rescuers might need to retrieve the robots physically for repair. It will take time for the robots to reach the robot's location and fix the robot. They also need information on the damaged robot's location and status to be able to track the robot. The secondary recovery phase prepares for and enables tertiary recovery by getting that information. The best-case scenario is probably a successful primary recovery phase where the robot repairs itself and resumes doing its expected tasks as usual, with

minimal to no effect on the mission. Secondary recovery preserves the usefulness of the damaged robot to the mission. However, there can still be a loss of skills in the environment. For example, a robot which damaged its victim sensors can no longer do victim confirmation tasks. This means the number of robots specialized in conforming victims has reduced. Tertiary recovery ensures that damaged robots are accounted for by allowing tracking and repair by humans. However, it may take time for the humans to correctly track down the robot, retrieve it, repair it, and release it or a replacement back to the environment.

Information Retrieval

Information retrieval is an important aspect of my recovery framework. The robot attributes, role suitabilities, robot operation modes, etc., are adjusted as a part of the secondary recovery phase. Power saving mode in a robot usually indicates critical health of the robot, where the robot may die at any time. In this mode, the robot turns off its mobility to save power. Instead, it uses its remaining power to periodically communicate its location and knowledge with neighbouring robots. This increases the chances of someone noticing its broadcasts in case it is stranded in some isolated area where there are fewer robots. The knowledge shared includes the location of the victims it tracked and the area it has mapped so far. This information is critical to the success of the mission. Without proper management, this information may be lost with the death of the robot. This information is shared less frequently than the typical robot announcements due to the larger size of the information, which may potentially increase network traffic. Another instance of critical information

sharing in my framework is the sharing of the location of immobile robots. Immobile robots periodically broadcast their location to other robots so that the robots can pass this information to humans, and the immobile robot can be retrieved from the environment.

3.3 Conclusion

This marks the end of the main concept description or methodology for my framework. In the following chapter, I will describe the details of my implementation for the purposes of evaluating my framework in Chapter 5. It will include the core implementation from the previous works from my lab, in addition to the components I added to the existing work to help with my evaluation.

Chapter 4

Implementation

4.1 Chapter Overview

This chapter describes the implementation of a simulated USAR environment that will help me to evaluate my framework. Since my framework is an extension of the existing works by Gunn [2011], Nagy [2016], and Fiawoo [2019], I use their framework implementation as a starting point for my implementation. This facilitates comparison between my work and the existing work in our lab. In addition, it gives us an overview of the gradual development of a well-defined framework for multi-robot systems operating in dangerous environments. Each of the existing works by Gunn, Nagy, and Fiawoo, including my research, addresses different challenges, gradually building towards a more complete framework for robot rescue missions in USAR. In this chapter, I will describe the components that I use from the existing framework and the modifications and new components that I added to evaluate my framework.

Section 4.2 provides a high-level description of the concept of the problem domain,

USAR, and the operations my robots are expected to perform to meet the goals of the USAR mission. Section 4.3 gives an overview of the simulated environment that I use for my implementation. I provide the implementation details for the simulation challenges (e.g., poor communication, robot failures). In Sections 4.4, 4.5, and 4.6, I describe the robot models, their attributes, roles, tasks, and autonomous control. In Section 4.7, I provide a brief description of the modules implemented for previous frameworks, as well as a detailed description of the module implemented for my framework.

4.2 USAR Concept and Goals

The USAR domain is one of the most commonly used problem domains when researching robotics in dangerous environments. Disaster environments like USAR are often dynamic and unpredictable [Chitikena et al., 2023]. Poor communication, risk of failures, limited time and resources, etc., make the mission even more challenging. The main goal of robots operating in USAR missions is to map out the disaster site and track down victims. The information gathered by the robots is used by human rescuers to learn about the environment and carefully rescue victims. This improves the chances of finding victims within a shorter time and lowers risk to human life. However, robots are also at risk of failures during the mission. For example, robots getting damaged due to being hit by falling debris, or unexpected system failures, etc. Lack of proper management of robots during failures may jeopardize the mission, delaying rescues or even harming victims.

I have developed a recovery framework for robot teams in dangerous environments.

The ultimate goals of the robots in my work are to map out as much of the disaster zone and find as many victims as possible within a given time, for given resources (e.g, robots), all while minimizing the effect of robot failures on the mission's success. The agent recovery framework allows management of malfunctioning, partially damaged, or totally damaged robots, such that they disrupt the mission as little as possible. I have explored ways to allow the partially damaged robots to continue contributing to the mission as much as possible. Moreover, this framework facilitates the retrieval and recovery of robots by humans if necessary.

It would have been interesting to evaluate my framework using real-life robots in disaster zones, but this is not feasible. Setting up a realistic USAR environment with real robot teams is difficult and expensive. Moreover, my experiments require simulation of robot failures to evaluate the agent recovery framework. Having to trigger robot failures and reset them repeatedly does not seem feasible in real life. At this point in the research, where I have only begun developing the agent recovery framework, it is more suitable to implement my work in a simulated environment. I will have more control over my experiments, and I can repeat scenarios as many times as I want at minimal to no cost compared to real-life experiments.

4.3 Simulated Environment

4.3.1 Simulator

There are quite a few multi-robot simulators that are commonly used in the study of multi-agent systems. Koenig and Howard [2004] is a 3D high-fidelity simulator

based on Open Dynamics Engine. It is designed to simulate complex multi-robot environments with fine-grained control. However, it runs slowly with large populations. USARSim [Carpin et al., 2007] is another multi-robot simulator developed using the "Unreal" game engine, which is similar in scope to Gazebo. Gunn [2011], Nagy [2016], and Fiawoo [2019] implemented their work using the Stage API [Gerkey et al., 2003]. Stage simulates virtual worlds populated with robots, sensors, and environmental objects, enabling robot experiments without access to real hardware and environments. Vaughan [2008] proposed a benchmark for multi-robot simulator performance, and the results showed that Stage's runtime scales linearly with the population size up to 100 complex robots or 1000 simple robots simulated in real-time. I chose the Stage API for implementing my work, firstly because my work is an extension of Gunn's [2011], Nagy's [2016], and Fiawoo's [2019] works, and they used the Stage API to implement their work. Secondly, due to the complex logic behind the existing works, in addition to my work, my implementation can benefit from the scalability and performance of the Stage API. Finally, the Stage API provides various sensors and actuators, including rangefinders, fiducial detectors, and a versatile robot base, making robots easily configurable and controllable [Gerkey et al., 2003]. This facilitates the simulation of a variety of robot failures for my experiments.

4.3.2 Environmental Design and Objects

Gunn's framework was implemented using Stage API version 3.2. The experimental environment was a simulation of the USAR domain with obstacles like walls and debris, victims and false victims (debris that may look like victims), and teams

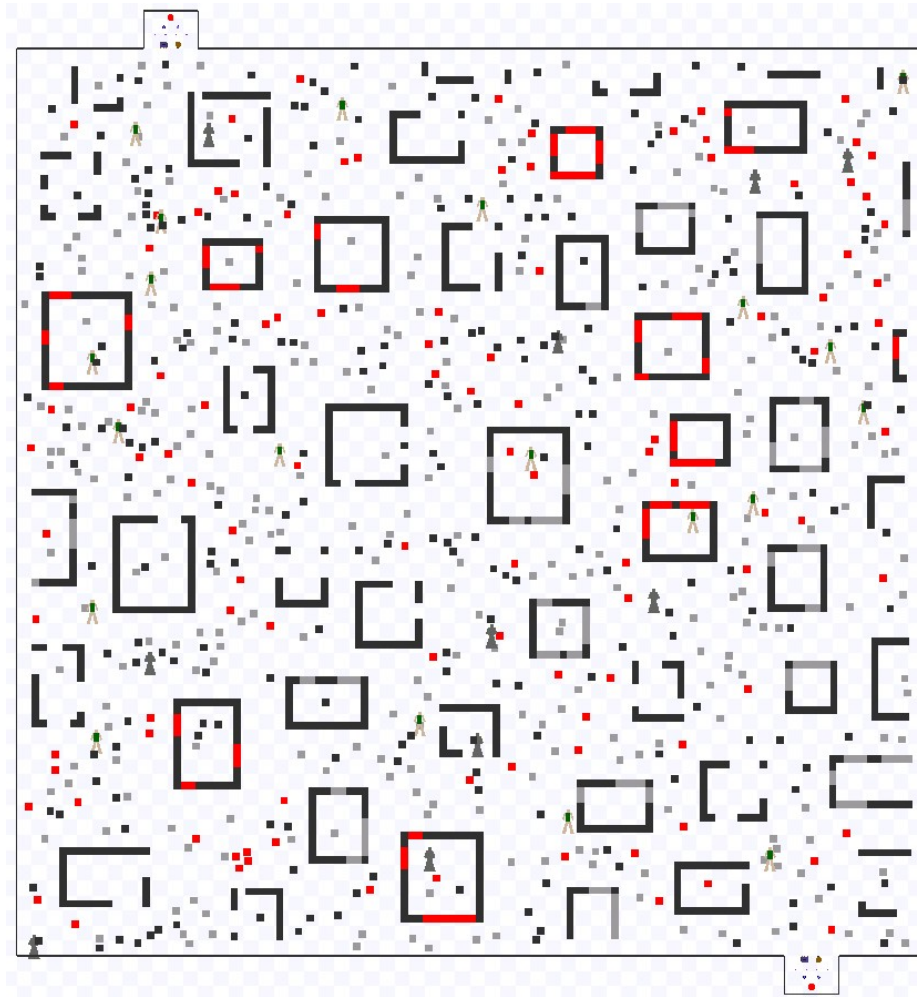


Figure 4.1: One of the experimental worlds that I used for the evaluation of my framework.

of heterogeneous robots. 3 types of robots were used: MaxBots, MidBots, and MinBots (More on robot types in Section 4.4). Nagy later ported Gunn’s framework to Stage version 4.1 and added a new robot type: DebrisBot, a new object: a marker to mark victims or to represent a first aid kit, in addition to the implementation of the ARC framework. Fiawoo added 2 new object types: fire and stairs, and 2 new robot types: FireBot and StairBot. He simulated multi-storeyed disaster sites for some of his experiments to evaluate how often StairBots helped other robots move between

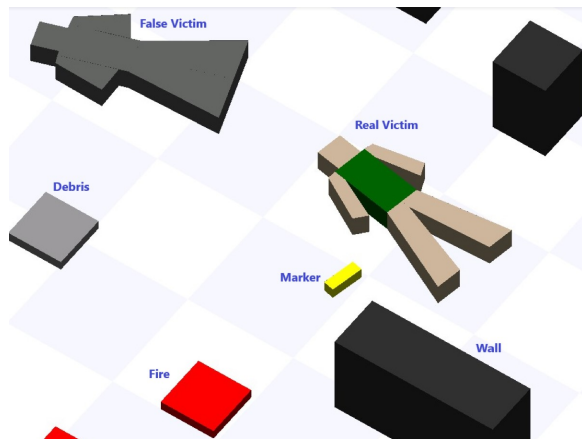


Figure 4.2: Objects in the environment.

different levels to improve their location-based satisfaction. My work focuses more on the robot failures and recovery management, which includes role, task, and team management during robot failures. It is sufficient for me to evaluate my framework on single-storeyed sites. Moreover, MaxBot, MidBot, MinBot, DebrisBot, and FireBot provide enough heterogeneity to test the recovery of heterogeneous teams of robots. As such, I do not use Stairbot and stairs in my experimental worlds.

Figure 4.1 shows an example of my experimental world. Objects in this world include *walls*, *debris*, *fire*, *markers*, and *robots*. Walls are represented by the dark grey blocks, usually taller than the debris, and not passable by robots. Debris is represented by grey blocks, usually smaller in height, and passable by robots with tracked wheels. These are also removable by DebrisBot to help clear the way or help robots stuck in debris. Fire is represented by red blocks and is removable by FireBots. Victims are represented by human-like figures scattered randomly in the environment. There are 2 types of victims simulated. The grey human-like figures represent false victims. These are debris that resemble humans. The rest of the human-like figures

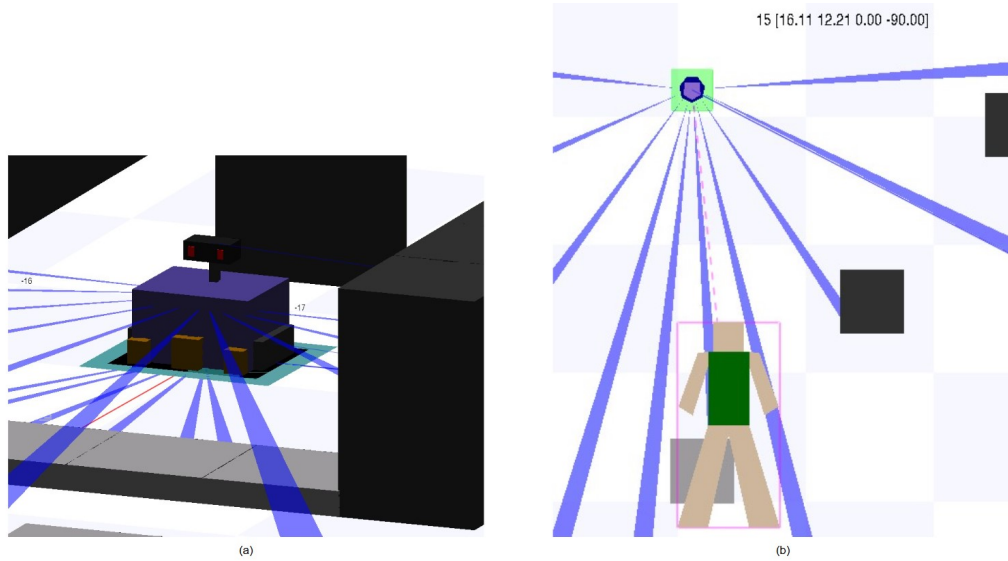


Figure 4.3: Figure (a) shows sonar sensors in DebrisBot for detecting low-lying debris and walls. Figure (b) shows a MidBot detecting a victim using its victim sensor. All fiducial sensor data is visualized with magenta dotted lines, like here.

in green represent the real victims that need to be rescued. Only specialized robots with enhanced sensors can correctly identify real from false victims. Other robots are equipped with victim sensors with lower accuracy and cannot distinguish between real and false victims. *Markers* are represented by small yellow objects. These are carried by robots and dropped near potential victims. Each robot carries a maximum of 2 markers and can request markers from other robots if necessary. At the beginning of a mission, two teams of robots are released from the ends of the environment. Details on the team formation are discussed in Section 4.5. Replacement robots are also released into the environment 5 minutes after the mission starts. These are not part of any team at the beginning. They are intended to be recruited into existing teams or form new teams.

4.3.3 Object Detection

Stage 4.1 provides a "ranger" model to simulate range finding sensors like laser sensors, sonar sensors, etc., for object detection in the environment. In this implementation, these types of sensors are mainly used for debris and wall detection. All robots have some type of sensors, which can vary in numbers, detection ranges, FOVs, etc. Stage also provides another abstract class of sensors known as fiducial sensors (see Figure 4.3 (b)). It can be used to simulate RFID technology, robotic vision, or other sensors based on object identification. In this implementation, this class of sensors is used to simulate specialized object detectors or sensors like *fire sensors*, *victim sensors*, *robot sensors*, and *marker sensors*. All of the fiducial sensors used in this implementation have a FOV of 180°. *Marker sensors* can only detect markers within 8 meters of distance. *Robot sensors* and *fire sensors* can detect robots and fires within 6 meters, respectively. There are 2 types of victim sensors to simulate victim sensors with different capacities. The *basic victim sensor* can identify victims within 4 meters with very low accuracy. It cannot distinguish between real and false victims. However, these basic sensors help robots to navigate around potential victims, reducing the possibility of driving into victims. A high-accuracy *victim sensor* can correctly identify and confirm victims within a 6-meter range. More about how these sensors are distributed among different types of robots is discussed in Section 4.4.

4.3.4 Energy Allocation and Dissipation

Stage provides a simple energy storage power pack model that can be used as an energy source for robots in the environment. The previous implementations by Gunn, Nagy, and Fiawoo did not include energy consumption by robots and did not use the power pack model. However, proper energy management is also another important aspect of missions like USAR. Robots are powered by a limited source of energy, which may run out at any point in the mission. Throughout the mission, robots are constantly using different parts of their bodies to perform tasks. The rate of energy usage may depend on the complexity of the robot's structure, weight, number of tasks done, etc. Once a robot dies, it will not be able to communicate and ask for help. Critical information gathered from the environment by a robot may be lost if it dies. Like any battery-powered device, like computers, cellphones, etc., it is possible to set a threshold power level that marks a critical power level in a robot. Once the power level reaches the threshold, the robot can switch to energy-saving mode. This will give it enough time to take recovery measures. For instance, letting neighbours know of its location so that someone can retrieve it if it dies, asking for help, charging from a charging station or other robots, etc. One of the failure recovery approaches addressed by my agent recovery framework is the depletion of the power level of robots. To evaluate my recovery framework, I simulated power depletion or failure by including power packs for the robots in the environment.

The Stage model uses a very simplistic model of power consumption. The base power consumption rate for each device (e.g., robot base, actuators, sensors, etc.) is predefined and the consumption rate changes with the velocity, weight, and the type

of the device. For example, a base robot model in Stage has a base cost of 1.0 watts and has a cost of 10 watts per kg per meter per second while moving in any direction, each sensor has a base cost of 2.0 watts, etc. Power consumption is then calculated as:

$$P = W_{base} + abs(V_x) * W_{kgms} * M + abs(V_y) * W_{kgms} * M + abs(V_z) * W_{kgms} * M \quad (4.1)$$

Where P is the power consumed by the device, M is the mass of the device, V is the velocity in x , y , or z direction, W_{base} is the base cost, and W_{kgms} is the cost per kg per meter per second. This equation was adapted from the source code of Stage API version 4.1.

Real-life robots are more complex and, as such, have more sophisticated power consumption models. For my implementation, I focus more on the possible preparations and recoveries from power failures, hence, this simplistic model of power consumption is sufficient for me. For my experiments, I allocated each robot enough energy to last them approximately 27-30 minutes of the mission. Each trial runs up to 30 minutes. So, most robots are bound to deplete 100% energy within this period, triggering the recovery mechanism in my framework. This will help me evaluate the team's performance given the limited energy with versus without a recovery framework. I calculated the energy allocated for each robot based on its energy dissipation rate. For example, I observed that the DebrisBot consumed energy at an average rate of about 0.23-0.27 KW or KJ/s. I estimated that with this consumption rate, a DebrisBot would need about 450 KJ of energy for 30 minutes. It can use up all the



Figure 4.4: Energy consumption data shown in Stage for DebrisBot.

energy long before 30 minutes if it is doing a lot of energy-consuming tasks. It may also have surplus energy left even after 30 minutes if it does not perform too much energy-intensive work.

4.3.5 Simulated Challenges

USAR is a challenging domain. In this section, I will discuss some of the challenges simulated for the purpose of evaluating the agent recovery framework, and comparing it to prior work.

- **Complex Map:**

One of the main tasks of robots in a USAR mission is mapping the environment to help human rescuers in navigating the disaster zone and rescuing victims. A disaster zone like USAR is dangerous and unpredictable. There can be debris,

fire, flooding, smoke, etc., which makes it difficult to navigate. Moreover, the structures are often unstable, which is why maps need to be updated from time to time to account for changes in the environment. Robots need to navigate around obstacles and track down victims in such a complex environment. To simulate such a complex environment, Gunn [2011], Nagy [2016], Fiawoo [2019] used a software tool developed by Wegner and Anderson [2004] to generate simulated environments with random maps with obstacles like debris, walls, and fire. In my experiments, I use 60m x 60m environments, with approximately 13% of the environment covered by obstacles (e.g., Figure 4.1). There are also voids or rooms placed randomly in the environment. These rooms have one or more openings, often covered by debris or fire. These simulate situations like a debris cleaning robot clearing the way for other robots, and a victim detector robot confirming if there are victims in the room, or requiring robots with tracked drive systems to navigate, etc. Real and false victims are placed randomly within this environment.

- **Unreliable Communication:**

Stage provides an ITU indoor radio model to simulate wireless radio transmission among robots. As in the previous implementations [Gunn, 2011; Nagy, 2016; Fiawoo, 2019], all robots in my implementation are equipped with short-range (approximately 20m) wireless communication modules. It is true that wireless transmitters nowadays can achieve greater distance in open areas. However, the environment configuration (e.g., debris, walls, flood, fire, etc.) in disaster zones can obstruct communication. Gunn [2011] extended the WiFi

module in Stage to simulate random communication failures. A communication success parameter is used to determine the percentage of message failures for every robot.

- **Robot Failures:**

The previous works [Gunn, 2011; Nagy, 2016; Fiawoo, 2019] implemented random temporary and permanent robot disablements to simulate robot failures. During temporary failures, robots stopped functioning completely for about 3-4 minutes before resuming operation. Permanent failures lasted for the remaining duration of the mission. A *failure probability* parameter is defined for every robot, which specifies the probability of a robot experiencing temporary or permanent failure. Real-world robot failures are not so simple. Robots do not automatically resume operation without any recovery measures. Moreover, robot failures are not always total. It can affect individual robot components as well. Some failures can be detected right away by monitoring system errors or thresholds, while some are not so apparent and require more complex detection methods. For example, a malfunctioning sensor in the robot may not be detected right away, but by monitoring the sensor readings, comparing them with other sensor readings or expected ranges, it is possible to deduce that there is something wrong with the sensor.

To help me evaluate my agent recovery framework, I added a *fault simulator* module to Stage and previous implementations, which manages and simulates robot failures by individual components. The fault simulator simulates mainly 2 kinds of failures: *faulty readings* and *unresponsiveness*. Faulty readings refer

to the erroneous readings originating from malfunctioning components such as sensors. It is simulated by adding small random offsets to the actual readings. Unresponsiveness is simulated by simply turning off the component so that the component stops functioning. The duration of these failures cannot be defined in terms of permanency. The robots continue in fail states until and unless they are recovered successfully. During the primary recovery phase of my framework, I simply turn off the fail state to resume normal behaviour in robots. I set a downtime for the robot to simulate the time taken to restart or recalibrate a robot in real life. I use the same *failure probability* parameter to control how likely a robot is to fail at any given time.

Physically, the robots in my implementation are not as complex as the real-world robots they represent. However, they have enough components to allow a variety of failures. The main components include sonar sensors, robot sensors, victim sensors, fire sensors, marker sensors, fire extinguishers, damage rays for debris removal, motors or wheels, and WiFi. I also simulate an overall failure or sudden death of robots, where the robots stop working completely. This phenomenon can be accidental, e.g., a robot completely damaged by fire or falling debris etc. It can also be predictable, like a robot battery power running out. Stage already provides a power pack module and power consumption model that I used to simulate robot deaths due to power failure (described in Section 4.3.4). The fault simulator is invoked randomly based on the failure probability parameter. At every invocation, the fault simulator randomly selects one component to fail in a robot. The type of failure (faulty reading or unresponsiveness) is also

randomized. At any time, a robot can suffer from multiple failures.

4.4 Robot Types

In this section, I will describe the characteristics and the capabilities of the 5 robot types or models that I use in my implementation: MaxBot, MidBot, MinBot, DebrisBot, and FireBot.

- **MaxBot:**

The MaxBot is the 3rd largest robot in my implementation. It is equipped with a tracked drive system that allows it to traverse not only through open areas but also over low-lying debris. It has an array of 3 sonar infra-red (IR) range sensors, each with a 25° FOV and 2 meters range, positioned on the lower body of the robot that help it detect low-lying debris in the environment. It also has a laser range finder with a FOV of 180° to identify all other obstacles. Stage stopped supporting the laser model starting version 4.1. Instead, now it considers all rangefinder models to be part of the "ranger" model. So, even though the implementation represents laser and sonar sensors separately, they derive from the same "ranger" model. It also has a basic victim sensor that can identify victims with very low accuracy and allows navigating around victims. It also has robot sensors to help with robot interaction and coordination. Finally, it has marker sensors to detect markers in the environment. MaxBot is most suitable for leadership roles due to its computational capabilities. It can detect frontiers, maintain a shared map and assign tasks. It is also equipped with a

path planner that allows it to communicate path information for other robots when assigning tasks, if a path can be found using its map knowledge. The power pack in MaxBot is charged with 342 kilojoules of energy, enough to last 27-30 minutes of the mission (more in Section 4.3.4).

- **MidBot:**

MidBot is smaller than MaxBot but larger than MinBot. It has a wheeled drive system. Unlike MaxBot or robots with a tracked drive system, it cannot drive over low-lying debris and has to navigate around it. A MidBot has an array of 10 sonar sensors positioned on its body in such a way that it has a larger FOV in total. Each sonar sensor has a FOV of 5° and a range of 6 meters. These sensors help it in obstacle avoidance and mapping the environment. MidBot is the only robot type in this implementation that is equipped with a high-accuracy victim sensor, making it most suitable for victim confirmation tasks. It also has robot sensors and marker sensors. Its computational facilities include frontier detection, generating frontier tasks, maintaining a combined team environment map, and task assignment. These capabilities make it somewhat suitable for leadership roles. However, it lacks a path planner, meaning robots may become lost in the environment trying to perform tasks assigned by MidBots. The MidBot power pack is allocated 162 kilojoules of energy based on dissipation rate, such that it lasts approximately 27-30 minutes.

- **MinBot:**

MinBot is the smallest and simplest robot model in this implementation. It has

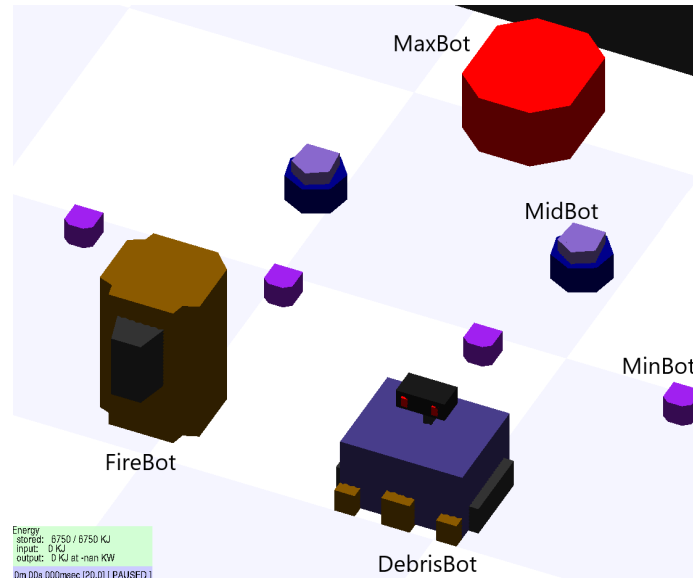


Figure 4.5: This figure provides a close look at the physical structure of each robot type in the team, as well as the total energy stored by the team at the beginning of the mission.

the least sensory and computational capabilities. Its inexpensive and expendable nature makes it more suitable for exploration tasks. It has a wheeled drive system. There is an array of 5 sonar sensors spread across the front of its body to help it navigate around obstacles. It lacks robot sensors and hence, cannot initiate rendezvous or encounter tasks with other teams. It has a basic victim sensor to help it detect potential victims and navigate around them during exploration. They also have marker sensors to mark potential victims if necessary. MinBot is the most poorly suited for leadership roles due to its limited physical and computational abilities. It cannot detect frontiers, generate frontier exploration tasks, assign tasks, combine shared maps, or plan paths. MinBot in my implementation are allocated the least amount of energy (90 kilojoules) due to its simple structure, enough to last 27-30 minutes.

- **DebrisBot:**

DebrisBot is a robot specialized in debris removal tasks. It is larger than MaxBots. It is equipped with damage rays that help it break down low-lying debris. It has a tracked drive system. It has 2 sonar arrays: one with 6 sonar sensors placed at ground level on the robot's body and the other with 10 sonar sensors mounted 20 cm above the ground (see Figure 4.3 (a)). Each sensor has a 5° FOV. The ground-level sonar array has a narrower FOV overall, used in detecting low-lying debris. The top-level sonar array has a wider FOV overall, suitable for detecting walls. The two separate sonar arrays distinguish between objects that are removable (e.g., debris) and that are not removable (e.g., walls). DebrisBots are also equipped with basic victim sensors and robot sensors to avoid mistakenly identifying potential victims and robots as debris. It also has a marker sensor. Computationally, it is similar to MinBots, making it one of the least suitable robots for leadership roles. I allocated it 450 kilojoules of energy that will last it approximately 27-30 minutes in the mission.

- **FireBot:**

FireBot is a robot specialized in detecting and extinguishing fires. It is equipped with fire extinguishers that it uses to put out fires in the environment. It has a tracked drive system that allows it to traverse through low-lying debris in the environment. It has the same sonar sensor arrangement as DebrisBot. In addition, it has a fire sensor, a basic victim sensor, a robot sensor, and a marker sensor. Computationally, it is similar to MinBots and DebrisBots, lacking the ability to detect frontiers, maintain shared map knowledge, plan paths, nor

assign tasks. It is more suitable for exploration tasks where it looks for fires in the environment and extinguishes them. It is one of the least suitable robots for a leadership role. FireBot is the most complex robot with the highest number of sensors. Based on its energy dissipation rate, I allocated it 594 kilojoules of energy that will last it approximately 27-30 minutes in the mission.

4.5 Attributes, Roles, and Tasks

4.5.1 Attributes

As discussed in Section 2.4.1, each robot has a set of characteristics and capabilities known as attributes that help it evaluate its attributes and those of others during role and task assignments. Each attribute has a value assigned, like the value for the fire extinguisher attribute in FireBot is "yes" or "true", but "no" or "false" for a DebrisBot. Again, all robots have an attribute of marker count with a value of 2 at start, referring to the number of markers a robot has. In my framework, I added a measure fail rate to each attribute in a robot. This represents how often the related component related to the attribute fails. For example, fire sensors with 0.2 attribute fail rate may fail 20% of the time (more in Section 3.2.3). At the beginning of an experiment, the fail rates of all attributes in all robots are 0.0, representing perfect health. There are 3 categories of attributes: physical, computational, and sensory attributes.

- **Physical Attributes:**

Physical attributes comprise physical features in robots, like size, weight, speed,

Table 4.1: Values of physical attributes of robots used.

	MinBot	MidBot	MaxBot	DebrisBot	FireBot
Locomotion	wheeled	wheeled	tracked	tracked	tracked
Length x Width (cm)	10x10	20x20	44x38	40x50	45x35
Expendability	1.0	0.25	0.05	0.1	0.1
Debris Remover	no	no	no	yes	no
Fire Extinguisher	no	no	no	no	yes
Marker Count	2	2	2	2	2
Power level at start	90 KJ	162 KJ	342 KJ	450 KJ	594 KJ

type of drive system for navigation, and any type of special equipment or hardware they possess. Table 4.1 shows the physical attributes and their respective values for the robots used in my framework.

- **Computational Attributes:**

Computational attributes represent the computational capabilities or components that exist in a robot's control software. Not all robots are equipped with the same computational capabilities. Some have limited CPU power and memory, and hence are not so good with computational tasks like maintaining maps and teams. On the other hand, there are some robots with high CPU power and memory, but limited sensory and navigation capabilities, which makes them less

Table 4.2: Computational capabilities of robots used.

	MinBot	MidBot	MaxBot	DebrisBot	FireBot
Victim Tracker	yes	yes	yes	yes	yes
Frontier Finder	no	yes	yes	no	no
Manage Team Map	no	yes	yes	no	no
Assign Tasks	no	yes	yes	no	no
Planner	no	no	yes	no	no

suitable for exploration tasks and more suitable for leadership roles that require efficient team management. Table 4.2 shows the computational capabilities of the robots in my framework.

- **Sensory Attributes:**

Sensory attributes refer to the types of sensors used by different robots in my implementation. The simulation logic behind object detection in my implementation is discussed in Section 4.3.3. The different sensors simulated and the sensory capabilities in different robots are presented in Table 4.3

4.5.2 Tasks

The robots performed various tasks during my implementation. Every task has a set of minimum requirements, suitability requirements, self-execution suitability requirements, and a priority metric. Before accepting and adding a task to its queue for execution, a robot evaluates itself against these requirements to see if it is fit for

Table 4.3: Sensor attributes of robots used.

	MinBot	MidBot	MaxBot	DebrisBot	FireBot
Sonar Sensor Count	5	10	3	14	8
Sonar range	4m	6m	2m	6m	6m
Victim Sensor	basic	advanced	basic	basic	basic
Robot Sensor	no	yes	yes	yes	yes
Fire Sensor	none	basic	basic	basic	advanced
Marker Sensor	yes	yes	yes	yes	yes
Laser rangefinder	yes	yes	yes	yes	yes

the task. The evaluation process to determine the robot's task suitability is described in detail in Sections 2.4.1 (Gunn's equations) and 3.2.3 (modified equations for my framework). After calculating the robot's task suitability, it is compared to the task's self-execution suitability metric. The robot's suitability to perform the task must be greater than or equal to the pre-defined self-execution suitability metric for the task. Otherwise, the robot is deemed unfit for the task. If multiple robots are fit to perform the task, the robot with the highest suitability is assigned to do it. The priority of the task helps in prioritizing tasks in the robot's task queue.

Most of the tasks in my implementation were a part of the previous implementations by Gunn [2011], Nagy [2016], and Fiawoo [2019]. I modified some of the tasks to adapt to my framework's needs and added a new task for the recovery of robots.

I briefly describe all the tasks as follows:

- *Explore*: This is the lowest priority task where an agent randomly roams about in the environment. Most robots do this task in their idle state. However, the minimum requirement of this task is that a robot can move and can see obstacles. If a robot has damaged its wheels or sensors or both, and does not meet the minimum requirement, it will sit idly doing nothing, unless it has other tasks in its queue that it can do (e.g., recruitment, rendezvous, team management, etc).
- *Explore Frontier*: In this task, robot is provided a location coordinate to explore, and it reports any findings to the team leader. This task requires a robot to be able to move and have sonar sensors to navigate around obstacles. It is more suitable for robots with higher expendability and smaller size, like MinBots since the location can be dangerous and too narrow for other robots.
- *Find Team*: This task is assigned to robots when they are first released into the environment, so that they can encounter other robots and get added to or form new teams.
- *Find Victim*: This task is not explicitly assigned to any robot. Rather, it is used to determine a robot's suitability to track victims. It requires at least a basic victim detector.
- *Confirm Victim*: In this task, robots are expected to confirm victims at a given location. It requires advanced victim sensors to correctly identify real versus false victims.

- *Manage Team*: This task defines the capabilities required for a robot to be able to manage a team, like a leader.
- *Encounter*: Encounter or rendezvous refers to a task where a robot performs team merge and redistribution when they see another robot (see Section 2.4.1 for details).
- *Begin Role Recruitment*: This task is assigned to initialize and activate the role-recruitment (see Section 2.4.2) process in a robot in an active recruitment setting.
- *Find Robot*: In an active recruitment setting, when the role-recruitment process is active, the robot actively looks for a suitable robot for role-level or task-level recruitment.
- *Explore Debris*: This is the idle task of a DebrisBot when it has no other tasks to do. It randomly roams around in the environment and removes debris.
- *Remove debris at location*: This task is a guided debris removal task, where a robot requests a DebrisBot to remove debris at a specified location. This task is mainly requested when a robot is stuck in debris and it needs help from a DebrisBot to remove the debris.
- *Donate Marker*: This is a task where a robot carries out the marker donation process. Each robot in the environment is provided with 2 markers to drop by potential victims. If the robot runs out of markers, it can request a marker from another robot. A robot can donate a marker as long as it can move, has

at least one marker (even if donating it leaves it with none), and has a robot sensor. The marker donation task is also subject to a robot's willingness to do the task based on its current satisfaction (see Section 2.4.3) and workload [Fiawoo, 2019].

- *Wait for Marker:* During this task, a robot maintains its position while waiting for the robot that accepted its marker request. If the robot waits too long for the marker, it discards this task and looks for another robot that can donate a marker.
- *Extinguish fire:* This task is usually executed by FireBots to extinguish fires at a given location. If the FireBot detects any fire in the environment, it creates a fire extinguishing task for that location. It returns to that location to put out the fire once it is done with the current or other higher priority tasks.
- *Move To Goal:* This task is used to guide a robot to a given location or goal. It may or may not be provided with the waypoints or a path to the location. For example, a MaxBot can provide a location and a path to guide a newly recruited robot into its team. In my implementation, I also used this task to guide robots back to humans or the starting point for repair. Each robot records its starting location, so that it can get back if needed. The priority of this task is as high as the recovery task if it is assigned as part of recovery.
- *Recover from failure:* This task refers to the recovery process that a robot executes to recover from a failure. Any robot can perform this task as long as it is not dead. This is the highest priority task in this environment. This is

because the longer a robot is allowed to operate with failures, the higher the chances of incorrect readings and decisions to affect the team's performance. It is preferable to address any failures as early as possible. This task is only self-executable and not transferable, meaning it cannot be assigned to others. It can only be assigned by the robot's health manager module to the robot itself if a failure is detected. Each of these tasks is intended for only one component recovery. For instance, if the health manager diagnoses the robot's health and detects a victim sensor failure, it creates and assigns a recovery task for the victim sensor. The recovery task is a 3-phase task with primary, secondary, and tertiary recovery phases. It is described in greater detail in 3.2.3 (also see figure 3.4). If the self-repair is successful during primary recovery, the robot ends the task and resumes other tasks as usual. If self-repair is unsuccessful, the robot attempts secondary recovery to adjust its attributes and role suitability accordingly and prepares for tertiary recovery if necessary (e.g., creating a task to go to humans for repair). Once the task ends, the robot re-evaluates its role and suitability during the periodic role switch check and adjusts its role in the team if needed. It also broadcasts any change in its role and attributes during periodic announcements.

4.5.3 Roles

In this section, I will describe the different roles used in my implementation. As described in Section 2.4.1, a role is a set of tasks expected of a robot that will occupy that role. A robot's role suitability metric defines how well a robot suits that role. It

Table 4.4: Maximum role suitability of robots used.

	MinBot	MidBot	MaxBot	DebrisBot	FireBot
Leader	11	83	106	26	26
Coordinator	24	96	52	32	32
Explorer	124	56	52	52	52
Debris Remover	0	10	10	110	10
Fire Extinguisher	5	15	15	15	115

is calculated by evaluating its attributes against the expected tasks for the role (see Algorithm 1). Every task expected by a role has a hand-tuned relative importance metric that determines how important the task is to that role. For example, the coordinator role puts high emphasis on the *confirm victim* task. This metric is used as a weight when calculating a robot's role suitability. A robot's suitability to fill a role can change due to robot failures. Section 3.2.3 describes the details of how the fail rate of a component can affect a robot's task suitability. For instance, a FireBot's suitability to fill the fire extinguisher role will decrease if its fire extinguisher equipment is damaged and no longer works. Table 4.4 shows the maximum suitability of robots to fill a role in perfect health.

All roles in my implementation are briefly described below:

- *Leader*: The most important task expected from a leader is the management of their team. Team management includes tasks like task assignment, coordinating the efforts of the team members, detecting and assigning frontiers, and main-

taining an overall map. In my implementation, MaxBot is the most suitable to fill this role as it comes with high computational capabilities (path planning module, high CPU, etc.) that this role requires. MidBot is also somewhat suitable for the role, but it lacks the ability to plan paths. The other robots in the environment specialize in exploration tasks and lack the computational capabilities required by a leader role.

- *Coordinator*: Robots in the coordinator role are primarily responsible for exploration tasks, victim verification, team merges and re-distributions during encounters, and map management. MidBots are most suitable for this role. It is the only robot equipped with advanced victim sensors that can be used for confirming victims. Other robots can fill the role; however, it would not be the most perfect fit since they may be lacking the ability to do all the tasks expected of this role.
- *Explorer*: In this role, robots are mainly expected to perform exploration tasks like exploring frontiers at given locations and randomly exploring with the aim of finding potential victims, etc. During exploration of unknown locations, there is a chance of the robots running into danger, potentially being harmed. As such, low-cost robots with high expendability are best suited for this role (e.g., MinBots). Other robots are more specialized, higher cost, and lower in number. They are suited for more advanced and specialized tasks.
- *Debris Remover*: The main responsibility of a robot in this scenario is clearing debris in the environment. The debris removal process can be guided or un-

guided. In guided debris removal, the robot in this role is requested to remove debris at a specified location. Whereas, in unguided debris removal, the debris remover is expected to randomly explore the environment and clear debris. Only DebrisBots have debris-removing equipment (e.g., damage ray). So, these robots are most suitable to fill the role.

- *Fire Extinguisher*: This role requires the robot to find and extinguish fire. Only FireBots have advanced fire sensors to detect fire and fire extinguishing equipment to put out fires. Other robots are poorly suited for this role as they lack these special components.

4.5.4 Ideal Team

As discussed in Section 2.4.1, every team in the environment strives to maintain an optimal team structure, which is easy to manage and gives the best performance. In this implementation, an ideal team is formed with 1 leader, 1-2 coordinators, 3-10 explorers, 1-2 debris removers, and 1-2 fire extinguishers. These numbers were set through experiments in previous implementations [Gunn, 2011; Nagy, 2016; Fiawoo, 2019]. Figure 4.6 shows the team structure at the beginning of the environment. The structure changes due to recruitments, robot losses, etc., as the mission progresses.

4.6 Autonomous Control

Autonomous control refers to the low-level control mechanism in my implementation. All of the control schemas were already implemented in a previous implementa-

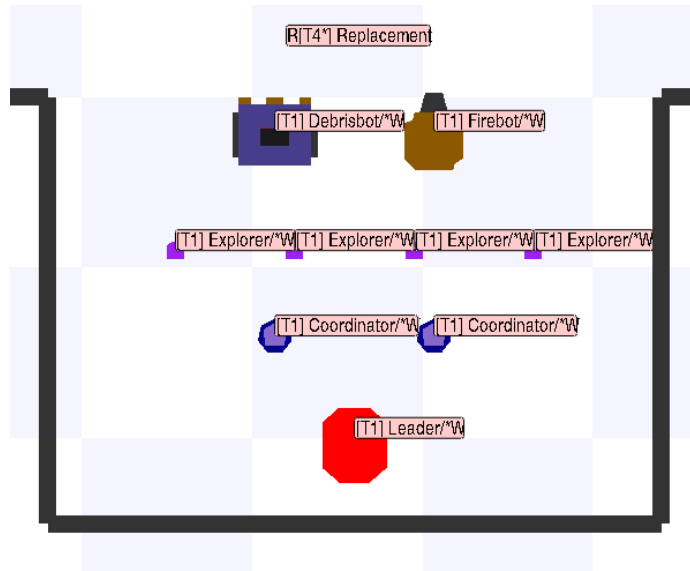


Figure 4.6: This figure shows a bird's eye view of the robot team structure and robot roles at the start point.

tion. However, I modified some to simulate failures, failure detections, and recoveries. Two categories of schemas are used in this implementation: perceptual schemas and motor schemas.

4.6.1 Perceptual Schemas

Perceptual schemas process the sensory inputs from sensors that help the robot to get a meaningful representation of the environment. The perceptual schemas in my implementation are as follows:

- *Localization*: This schema helps in tracking a robot's position and orientation in the environment. While all team members share the same coordinate system (centred at the starting point of the team), different teams may have different translational origins. This schema helps in translating coordinates into the re-

ceiver's or sender's coordinate system during the sharing of locations of victims, frontiers, etc. It also detects when a robot is stuck in debris, not moving for a long time. It creates a debris removal request to help get itself unstuck from debris. This is also a form of recovery implemented by Nagy [2016].

However, it is not always the case that a robot is stuck in debris that it cannot move. It is also possible that the robot's wheels are not working, or there may be something wrong with the motor schema that may be causing the problem. In my framework, the robot's position is tracked for a while during diagnosis, and if there was a debris removal request is made, the robot waits for help from the debris bot. If the debris robot successfully removes the debris and gets it unstuck, then recovery is not required. But if the debris removal does not work or takes too long, then there may be a failure in the robot's components, and so a recovery task is created for the motor schema.

- *Process Range Data:* This schema processes sonar range data from the sonar sensors that helps in detecting obstacles like debris and walls. In my implementation, during robot failures, the sensors may show erroneous readings or no readings at all (e.g., when sensors are unresponsive). The readings from this schema are monitored during diagnosis to detect failures in sonar sensors.
- *Detect Debris:* This schema uses the localization schema to detect when it is stuck in debris and records the obstacle so that it can avoid it in the future.
- *Detect Obstacles:* This schema uses the process range data, detect debris, detect robots, and detect victims schema to generate a list of obstacles. It distinguishes

between removable (debris) and unremovable (robot, wall, victims) objects in the environment.

- *Detect Robots:* This schema processes data received from the robot sensors in a robot. It gives a list of positions of robots detected at any time. Similar to sonar sensors, robot sensors can also give erroneous readings or no readings at all in case of unresponsive robot sensors.
- *Detect Victims:* This schema processes data from victim sensors to generate a list of locations of potential or confirmed victims. This list is fed into the victim tracker module to create victim confirmation tasks as required. Inaccurate readings can lead to the creation of victim confirmation tasks for inaccurate location, resulting in misleading another robot to an incorrect location.
- *Detect Lost:* This schema helps a robot determine if it is lost. It tracks the distance traversed by the robot while heading to a certain location. If the distance traversed is greater than five times the actual distance [Nagy, 2016], it assumes that the robot is lost. It discards or requeues the current task for later and moves on to the next task.
- *Detect Markers:* This schema helps in detecting markers around a potential victim. Similar to other sensory perceptual schemas, this schema may process inaccurate sensor readings in case of damage to the marker sensor.
- *Detect Fire:* This schema processes fire sensor data in robots with fire sensors. It generates a list of observed fire locations that helps in creating quenching

tasks for those locations. Inaccurate fire locations from damaged fire sensors can lead to wasted efforts in locating and quenching the fire.

4.6.2 Motor Schemas

The motor schemas guide a robot's motion using the data from the perceptual schemas. I only simulated one kind of motor schema failure, where the wheels stop functioning, and the robot cannot move at all. The motor schema failure is detected by monitoring the position of the robot. If it has not moved in a long time, there is a possibility of failure. The motor schemas used in my implementation are as follows:

- *Avoid obstacle:* This schema helps the robot in steering clear of detected objects by generating a repulsive force.
- *Move to location:* This schema guides the robot to a destination using a specified path or a reactive approach if a path is not available.
- *Turn in place:* This schema turns the robot in a clockwise motion while at its place, until it finds the robot for encounter or marker donation.
- *Random:* This schema prevents a robot from getting stuck by guiding the robot to a new random direction.
- *Recover Stuck:* This schema helps a robot get unstuck from debris or corners by backing up.
- *Seek debris:* This schema uses data from the detect obstacle schema to generate attraction force that guides DebrisBots towards short removable debris, and

repels away from unremovable objects like walls, victims, and robots.

- *Move to robot:* This schema uses data from the detect robot schema to move a robot towards another robot, enabling a physical encounter.

4.7 Framework-Specific Modules

In this section, I will describe the robot software that supports my framework. All except the health manager module were implemented by Gunn [2011], Nagy [2016], and Fiawoo [2019] to support their frameworks.

- *Encounter manager:* The encounter manager module manages the team merge and re-distribution process when two robots encounter each other.
- *Knowledge manager:* This manager manages any information known to the robot, like self-attributes, map, tracked victims, seen robots, current structure of the team, etc. It makes periodic announcements to let its team know that it is still on the team. If it does not hear from any teammates in 3 minutes or vice versa, the team considers it to be lost. In my framework, the knowledge manager is also responsible for sharing critical information like an immobile robot's location or a dying robot's knowledge about the environment, etc. When a robot becomes immobile due to damage, it announces its location with periodic announcements, so that the team and neighbouring robots know of its location, which can be used to rescue it later. When a robot has a critical power level and power-saving mode, it periodically broadcasts its map and victim knowledge.

Neighbouring robots update their knowledge with this shared information. This is done so that the progress of the mission is not lost with a robot's death.

- *Communication manager*: This module is responsible for managing the processing of wireless messages shared among robots in the environment. Messages can be of different kinds, like robot announcements, task assignments, recruitment, etc.
- *Role manager*: The role manager manages role and team-related processes like role-switch checks, desired team structure, missing role checks, etc.
- *Task manager*: This module manages the processes related to tasks such as task assignment, task selection, task preemptions, task execution, etc.
- *Recruitment manager*: The recruitment manager manages the recruitment processes, especially the active recruitment process where the robot actively searches for a suitable robot for recruitment. Concurrent recruitment does not require the robot to travel since the recruitment is done while it is doing its regular work.
- *Satisfaction manager*: This module manages rare skills in the environment by monitoring a robot's location and team-based satisfaction. A robot may choose to leave a team or location if its satisfaction is below a threshold. It may be more willing to accept recruitment requests. In addition to the robot tracking its own satisfaction, the leader also tracks the satisfaction and productivity of the team members and may let go of team members if they are not productive

enough. The satisfaction management process is described in detail in Section 2.4.3.

- *Health manager:* I implemented the health manager module to support my framework. It monitors robot health through periodic and triggered diagnosis of components. The diagnosis process runs in the background as it is doing other tasks. The periodic diagnosis is triggered every 8 minutes for all components, and the triggered diagnosis is triggered when a pre-defined condition is met, like power reaching threshold level, sensors becoming unresponsive, etc. Diagnosis can also be triggered for a re-diagnosis after attempting self-repair. This helps in confirming if the self-repair was successful. During diagnosis, some diagnostic tests are run on the components to determine the failure rate of the component. If a failure is detected and if the fail rate is greater than the component's current attribute fail rate (indicating the component's state has worsened), then a recovery task is created for the recovery of the component. If failure is detected during re-diagnosis, then the secondary recovery phase is triggered in the existing recovery task for the component. The health manager also manages the self-repair process, where the robot restarts, recalibrates, etc., to repair itself. It also indirectly manages the attribute, role, and mode adjustments in the knowledge manager through the recovery task. During the secondary recovery phase, the attributes are updated with new attribute fail rates obtained from the diagnosis. The knowledge manager is made aware of the robot's health state through modes (like immobile mode, power saving mode), as preparation for tertiary recovery by humans. For instance, sharing an immobile robot's location

to facilitate retrieval later. The failure detection process, health diagnosis, and recovery management process are discussed in greater detail in Chapter 3.

4.8 Mission-Specific Modules

The mission-specific modules refer to the modules specific to the USAR implementation. All these modules were implemented by Gunn [2011], Nagy [2016], and Fiawoo [2019]. I use the same mission-specific modules to support the USAR aspect of my framework.

- *Environment mapper*: This module maintains the shared team map of the environment by integrating smaller maps from other robots. It required good computational capability, which is not present in all robots.
- *Frontier finder*: The frontier finder module creates *explore frontier* tasks to explore unknown locations in the map. This also requires considerable computational capabilities.
- *Victim tracker*: This module keeps track of potential, false, or actual victims currently known to a robot.
- *Planner*: The planner is a computation-intensive module that enables a robot to find the shortest paths to goals. These paths can be shared with other robots with no planner, to help them find their path to their task's location.
- *Marker manager*: The marker manager manages the marker release process when a victim is detected in the environment. It also manages the donation or

borrowing of markers when a robot runs out of markers.

4.9 Conclusion

In this chapter, I described the implementation details of the simulated environment and my framework, which will help me evaluate my framework. In the next chapter, I will describe my experimental design, evaluation of my framework, and analysis of the results.

Chapter 5

Evaluation

5.1 Chapter Overview

With the goal of evaluating the effectiveness of my methodology (Chapter 3), I conducted some experiments in a simulated USAR environment. I compared the performance of robot teams using my methodology against 2 baseline approaches. The first baseline case represents an ideal world, where there are no robot failures at all. In the second baseline case, the experimental worlds have robot failures but no recovery measures for the failures. I compare how well the teams perform with and without my agent recovery framework. Additionally, I evaluate the effectiveness of the secondary recovery phase of my framework through separate experiments. In my evaluation, I consider factors like communication quality, recruitment modes, use of replacement robots, and the probability or extent of robot failures.

I begin this chapter by revisiting the research questions that I stated at the outset of my thesis (Section 1.7), and describe the metrics I used to evaluate my work. I also

describe the process of environment generation and selection, and the experimental design. Finally, I present the results from my experiments and discuss my findings.

5.2 Review of Research Questions

Recall my questions from Section 1.7:

1. **What effect does my agent recovery framework have on the overall performance of teams of heterogeneous teams of robots operating in dangerous domains?**
2. **When does my framework work best with respect to different factors such as type of recruitment strategies, quality of communication, presence or absence of replacements, etc. ?**

To answer my first question, I first conducted experiments to evaluate the performance of robots in an ideal world where there are no robot failures. This gives an idea of what an ideal performance might look like. I then conducted experiments to evaluate the performance of robots in environments with different levels of failures, with and without the use of the agent recovery framework. For the second question, I incorporated independent variables like communication quality, recruitment modes, and the use of replacement robots to see how well my framework performed under various combinations of these variables.

5.3 Evaluation Metrics

I mainly used 4 metrics to evaluate the performance of my framework: the *percentage of victims successfully identified to leaders*, the *percentage of area coverage*, the *number of debris items removed*, and the *number of fires quenched*. Most of these metrics were used by Gunn [2011], Nagy [2016], and Fiawoo [2019] as well. However, the previous works only focused on statistical cumulative data that included results from all robots, active or inactive. In my evaluation, the collected data can also be separated into 2 categories based on the method I used to collect it: *cumulative* and *instantaneous*. Only the final data collected after the end of each trial will be discussed for the evaluation.

5.3.1 Cumulative vs. Instantaneous data

The cumulative data refers to the data that was merged and recorded every 1 minute of an experiment, for the total 30 minutes of the trial. I only used the final cumulative value collected at the end (30th minute) of each experiment for my evaluations. The instantaneous data is only a snapshot of the data at the 30th minute of the trial. The reason behind collecting data in these two different ways is that while the cumulative data helps us observe the performance of robots throughout the experiment, the instantaneous data gives us a more realistic perspective of the data. In real-world missions, humans may not be able to collect data as frequently (e.g., every 1 minute). It is more likely that the human rescuers will retrieve instantaneous or snapshot data from the robots at longer intervals. Moreover, robot deaths also affect the data collected. In my experiments, I assume that data cannot be collected

from dead robots that are in the environment. Realistically, the dead robots can be recovered by humans tracking their locations that are stored by other robots. However, for my experiments, I only focus on autonomous recovery. So, the dead robots in my experiments remain in the environment. A robot that was alive at 5 minutes into the mission may die and not provide data at the 30th minute. This may affect the cumulative and instantaneous results. I also collect cumulative data regardless of the robot's status (dead or alive) to get an experimental point of view. All these statistics will help me in a more thorough evaluation of my framework.

5.3.2 Victims Identified

My implementation tracks the number of actual or confirmed victims reported or known to the leader of each team in the environment. Not every robot has complete knowledge of the team and the environment. In my framework and the ones before me [Gunn, 2011], [Nagy, 2016], [Fiawoo, 2019], a leader is considered to have the most complete knowledge of the team and the environment due to their role and computational capabilities. Data from MinBots is ignored due to their inability to merge and maintain team knowledge. Two types of cumulative data are collected: statistical data that includes data from both dead and active leaders, and realistic data that only includes data from active leaders. For instantaneous data, only active leaders' data is collected to simulate real-life scenarios where mission data is collected at longer intervals (e.g., every 30 minutes).

5.3.3 Area Covered

Similar to the victims' data, the area coverage data is also collected by merging maps from the team leaders. Team members report results from their exploration tasks to the team leaders. Leaders like MaxBots or MidBots are capable of maintaining and merging maps. They have the most complete version of the map among all team members. We do not consider knowledge from MinBots at all due to their inability to maintain and merge maps. Once again, statistical cumulative data (including dead leaders), actual cumulative data (excluding dead leaders), and actual instantaneous data are collected for area coverage.

5.3.4 Unique Tasks

Unique tasks refer to the tasks that can be performed by special types of robot [Fiawoo, 2019]. For example, debris removal and extinguishing fire are unique tasks that can only be done by DebrisBots and FireBots, respectively. While the information on unique tasks is not crucial to the USAR mission, I collect it for statistical purposes. Evaluating the number of debris pieces removed, or the number of fires put out, can give us an idea of how robot failures affect unique tasks and how these special robots perform with my agent recovery framework.

5.4 Environment Generation

The first stage of my experimental setup is the generation of the environment where I will run my experiments. I used multiple maps with similar configurations

(in terms of difficulty, obstacle and void coverage) to simulate the experiments instead of just one. This will help limit environmental bias to some extent. As described in Section 4.3.5, I used a world generator tool developed by Wegner and Anderson [2004] and later modified by Gunn [2011], Nagy [2016], and Fiawoo [2019] to generate worlds for my experiment. Each world has an area of 60m x 60m or 3600 sq. meters. 13% of the map is covered with debris, walls, and fire. The probability of an obstacle being debris is 60%, and for it to be fire is 15%. There are 50 random voids or rooms in the environment. 60% of these rooms have their openings covered with debris, and 15% are covered with fire.

There are 10 false victims and 20 true victims randomly placed in the environment (these are further discussed in Section 4.3.3). They are placed at least 0.8 m away from the walls and do not overlap with each other.

There are two starting positions for the two robot teams to enter at the opposite ends of the environment. Equally spaced entrance points for replacement robots are placed along the perimeter of the environment.

Initially, I generate 40 random worlds with the specified configuration (e.g., obstacle ratio, victim ratio, etc). I then hand-picked 3 worlds that fit my selection criteria the most. Similar to Gunn [2011], Nagy [2016], and Fiawoo [2019], I used the following selection criteria:

- I discarded worlds where entrances are covered by too much removable debris and fire. Otherwise, robots will spend too much time clearing out the debris and fire at the entrance instead of contributing to the main mission.
- I also discarded worlds where the entrances are covered by large walls. In these

worlds, robots have to spend a lot of time navigating the walls to enter the environment. This affects the overall performance due to limited mission time.

- Worlds with highly clustered obstacles were discarded. I only chose worlds where obstacles are evenly distributed. The goal was to select environments with roughly consistent navigational difficulties rather than having domains where difficulties were more concentrated in some areas than others.
- Lastly, I discarded worlds where victims were too close to each other. This is to remove the bias of cases where a robot may ignore a close victim if it notices a marker nearby left for another victim and assumes it is marked. This ensures that all the selected worlds would be roughly equally challenging.

At the end of the selection process, I had 3 worlds ready for my experiments. All 3 worlds are presented in Appendix A.

5.5 Experimental Design

The second stage of my experimental setup is the configuration of the independent variables, like recruitment modes, communication quality, etc. Stage requires the configurations, including the world map, to be pre-defined in a world file. The world file (**.world*) is passed as an argument to the stage run command. Stage then runs on the provided world file that includes the configurations. I used a factorial experiment design to evaluate my agent recovery framework. I used different combinations of the independent variables and the 3 world maps to generate the worlds for my experiments. The independent variables are discussed in Section 5.5.1.

An advantage of Stage is that it allows repeatability of experiments through the use of random seed integers embedded in the *.world* configuration file. These seeds are pre-defined and static. As a result, we get the same results no matter how many times we run the same configuration file. This is useful when we need to re-run trials for observation. However, to cover a sufficient breadth of events in my experiments, I intend to run each configuration multiple times. For each environmental configuration, I used 50 identical sets of random seeds to generate 50 more worlds. The main configurations, like maps, independent variables, are the same in these world configurations, only the random seeds are different. This allows me to run the experiment for each configuration 50 times. This helps me eliminate any bias that may arise from random numbers that may favour or disfavour a group of experimental trials.

I generated 2 sets of world files for my experiments. The first set is for the base case scenario, which represents a perfect world where there are no robot failures. Robots never run out of power or die. Since there are no failures, there is no need for recovery. So, these world files are configured with no robot failures and no recovery. I generated 1800 such world files. The experiments on these worlds will give me an idea of the highest performance of robots that I can expect, which is something I use to compare and evaluate the experiments with robot failures and the agent recovery framework. The second set of world files includes the robot failures and the use of the agent recovery framework. I generated 10800 of these world files. These worlds are used to evaluate the effectiveness of my framework in the presence and absence of my recovery framework. Figure 5.1 depicts my experimental breakdown.

I run all these on trials on an Amazon Web Services c5.9xlarge EC2 instance with

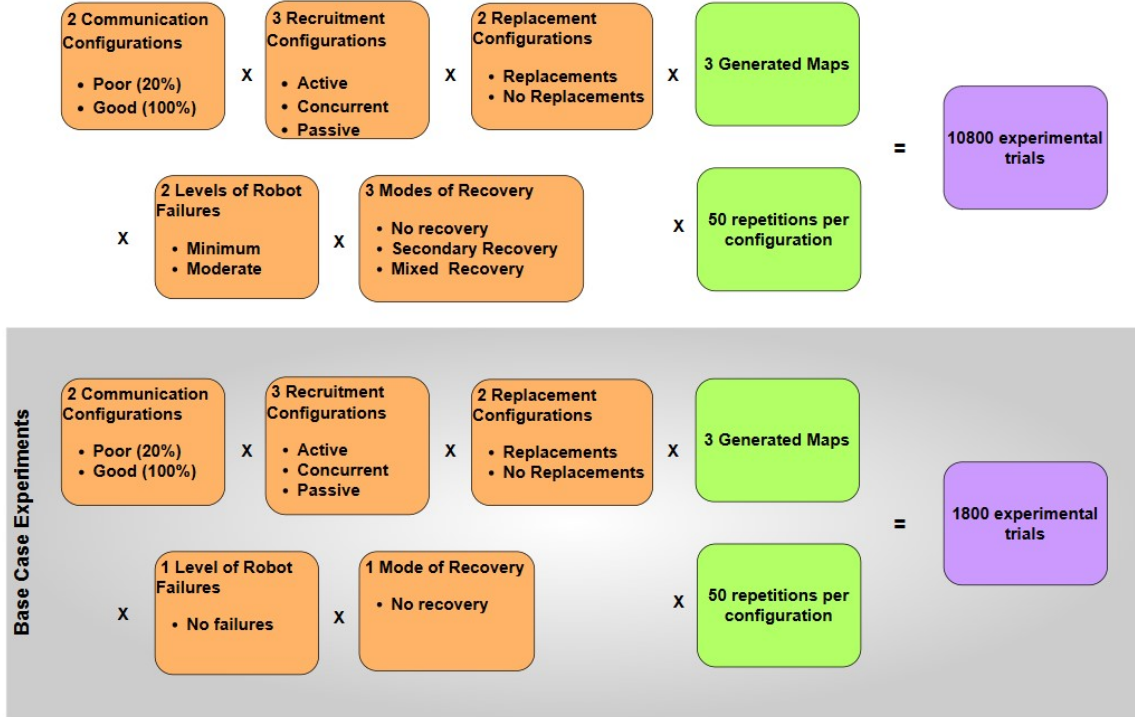


Figure 5.1: Breakdown of Experimental trials.

36 cores and 72GB RAM. I ran approximately 30 trials concurrently at a time. Robots and other models in Stage execute as worker threads. Multiple threads multiplied by multiple concurrent trials consume a lot of CPU resources. This leads to the AWS compute running out of resources if I run 36 trials concurrently over 36 cores. Which is why I maintained approximately 83% CPU Utilization rate by running fewer trials concurrently.

5.5.1 Independent variables

- **Communication Success Rate:** This refers to the probability of successful wireless message transmission between a sender and a receiver robot. For my

experiments, I use 2 success rates: 20% or low, meaning poor communication and 100% or high, referring to excellent communication quality.

- **Recruitment Mode:** This refers to the recruitment mode used in the experimental trial. I use the ARC framework’s [Nagy, 2016] 3 recruitment modes: active, concurrent, and passive.
- **Replacement:** This refers to whether replacement robots are used in the experiment. In experimental trials with replacements, I deploy an additional 10 MinBots, 2 MidBots, 1 MaxBot, 1 DebrisBot, and 1 FireBot as replacements. These are released 5 minutes into the experiment.
- **Robot failure level:** The robot failure level refers to the likelihood of robot failures in the environment. The higher the robot failure level, the more the robots are prone to failures. I use the same level of failures as Gunn [2011], Nagy [2016], and Fiawoo [2019]: none, minimum, and moderate. However, I do not simulate temporary or permanent failures. My failures are more specific to robot components and failure types (See Section 4.3.5). Every failure in my implementation has an approximately equal chance of happening. However, in reality, robot components may fail at different rates depending on how frequently they are used, their physical design and materials, and the degree and type of environmental exposure they experience. My thesis does not delve into the specifics of component failures. A future direction of my research could include experimenting with more realistic component failure probabilities.

As shown in Table 5.1, the failure probabilities are set in such a way that the

Table 5.1: Failure probabilities by robot types.

Level	Model	Average % total time failed
None	MinBot	0.000%
	MidBot	0.000%
	MaxBot	0.000%
	DebrisBot	0.000%
	FireBot	0.000%
Minimal	MinBot	14.9%
	MidBot	11.9%
	MaxBot	9.0%
	DebrisBot	14.9%
	FireBot	14.9%
Moderate	MinBot	25.1%
	MidBot	20.9%
	MaxBot	18.5%
	DebrisBot	25.1%
	FireBot	25.1%

explorer-type robots are most prone to robot failures as they have a higher chance of running into danger during exploration tasks. Robots filling coordinator roles like MidBots are slightly less exposed to danger as they stay closer to the team, helping coordinate the team. Maxbots have the lowest fail probability

due to their simple yet strong physical build. They stay closer to the team base, coordinating the team, tasks, and the map. So, they are less prone to failures. All these failure probabilities were used by the previous implementations by Gunn [2011], Nagy [2016], and Fiawoo [2019].

- **Recovery Mode:** This refers to the different configurations of robot recovery I use to evaluate my framework. There are 3 recovery modes I use in my experiments: none, secondary, and mixed. In the first mode, there is no robot recovery. This is to see how the robots perform with or without robot failures, without a recovery framework. In the second mode, I only use the secondary phase of my agent recovery framework. The primary and tertiary recovery phases have a chance of completely recovering the failing robots. Naturally, experiments with these recovery phases will show slight improvement compared to trials with no recovery management. On the other hand, the secondary recovery phase adjusts the robot attributes, roles, and tasks in such a way that it can continue the mission even with failing components. I ran an additional set of trials, with just the secondary recovery phase, to see how the attribute, role, and task adjustments affect the robot's performance. Finally, the mixed recovery mode refers to the overall recovery task, including all 3 phases of recovery as seen in Figure 3.4.

5.6 Main Experiment Results

In this section, I describe my experimental results using the evaluation metrics defined in Section 5.3. As mentioned before, results were cumulatively collected and merged or collected at once instantaneously at the end of each trial. I mainly focus on data collected from live robots only. This is because I assume dead robots cannot provide data. I also briefly discuss data collected from all leaders, regardless of being dead or alive, to get a statistical insight into the data. All results show the impact of communication, the presence of replacement robots, failure levels, recruitment modes, and recovery modes.

Initially, I ran a set of trials with no failures and with no recovery framework. The results from these trials are used as a reference point to contrast and compare with the results from trials with failures. It shows the highest performance of robots in my environmental setup under ideal conditions, where there are no robot failures. Only the base case shows results for no failures. All other trials run with minimal or moderate failure levels.

5.6.1 Victims Confirmed

The data on the number of victims confirmed is collected from the leaders only, as leaders are expected to have the most complete list of victims. Team members report the victims they find to their leaders.

Figure 5.2 and 5.3 show the instantaneous percentage of victims discovered with replacements versus without replacements under varying conditions of recruitment mode, robot failures, communication, and recovery mode. Data was collected from

live leaders at the 30th minute or the end of the trial. Figure 5.4 (with replacements) and 5.5 (without replacements) show data collected and merged every minute for the duration of the trials or 30 minutes. In general, results show that the percentage of victims confirmed is lower for trials with minimal or moderate failures compared to trials without failures or baseline. Naturally, trials with moderate failure levels showed results worse than minimal failure levels. Again, for the majority of the trials, the mixed recovery mode trials show the highest percentage of victims identified among the trials with robot failures. The trials with only the secondary phase perform a bit worse than the mixed recovery mode, which is expected since the primary and tertiary recovery phase in mixed mode gives a chance for full recovery with no requirement for attribute adjustments. The secondary mode results are still better than the trials with no recovery at all.

With 100% communication success rate, the instantaneous percentage of victims confirmed in the trials with failures but with no recovery is significantly lower than the baseline trials. In the baseline trials, approximately 80% of victims were identified when replacements are available, while 61-65% of victims were identified when no replacements are provided. Only 32-39% (with replacements) and 17-19 % (without replacements) of the victims were identified for the trials with failures and no recovery. However, a significantly higher percentage of victims were identified with recovery (mixed or secondary). The cumulative results do not show such significant differences, but still follow a similar trend where robots perform better with the recovery framework versus without the recovery framework. The instantaneous data is collected from only the live leaders at the 30th minute. On the other hand, cumu-

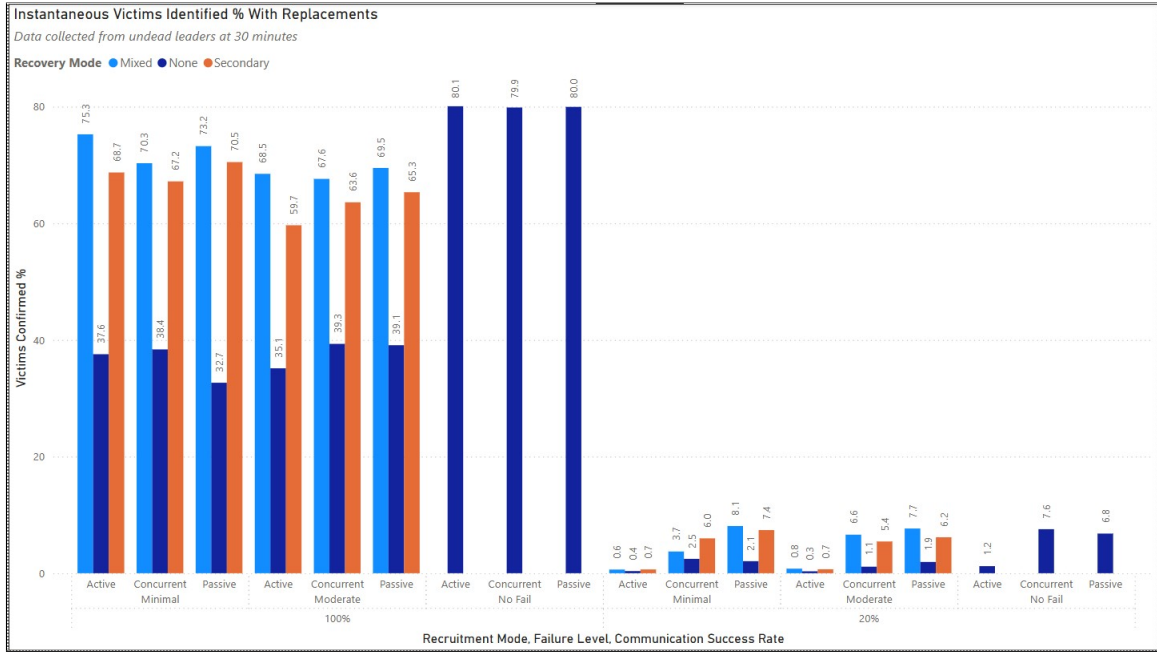


Figure 5.2: Instantaneous percentage of victims confirmed with replacements.

lative data is collected from leaders every 1 minute. A robot that may have been a leader and alive at 5 minutes, may not be a leader or may be dead at 30th minute. So, the instantaneous results show a smaller percentage of victims reported than the cumulative reports. A lot of information may be lost without proper recovery management.

Again, with 20% communication success rate, the difference between baseline results and main results without recovery is not as significant as seen in trials with 100% communication success rates. In fact, with my recovery framework, the percentage of victims identified is very close to the percentage of victims identified under ideal circumstances (no failures). This is because poor communication significantly affects the performance of teams. Robots cannot share information as well as they would with good communication, resulting in everyone having more incomplete information

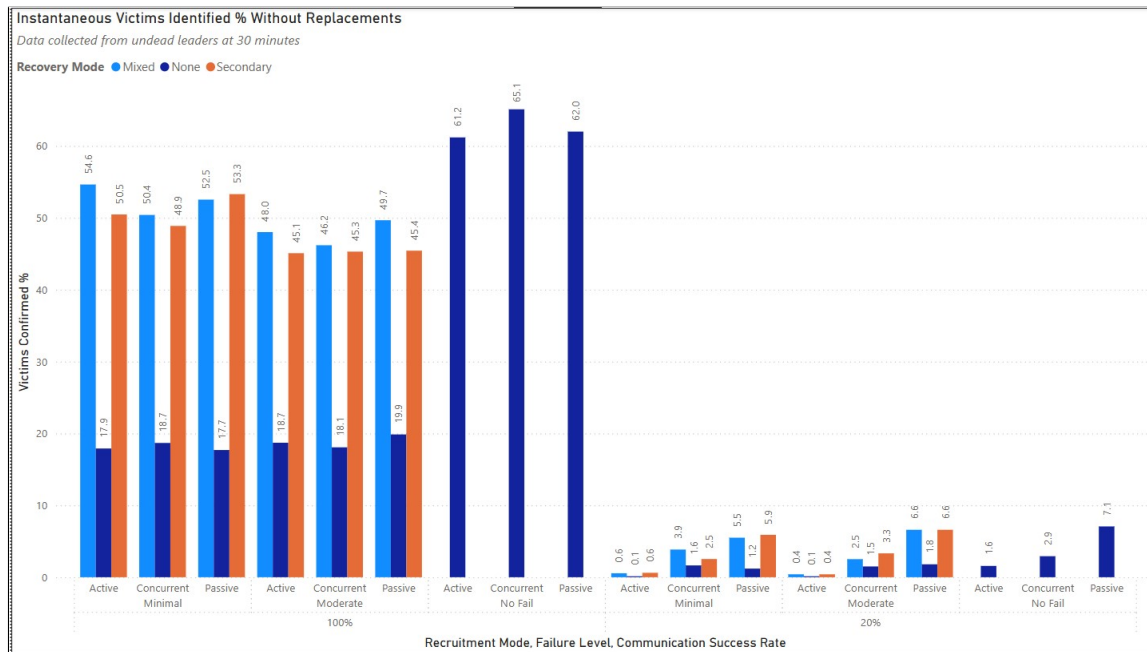


Figure 5.3: Instantaneous percentage of victims confirmed without replacements.

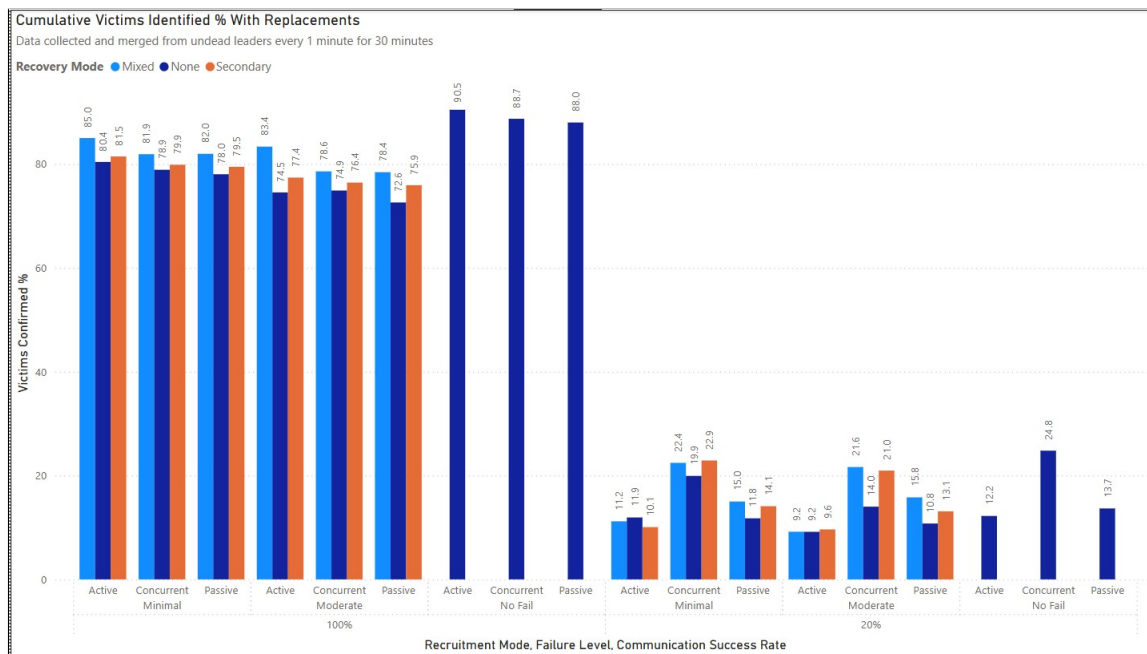


Figure 5.4: Cumulative percentage of victims confirmed with replacements.

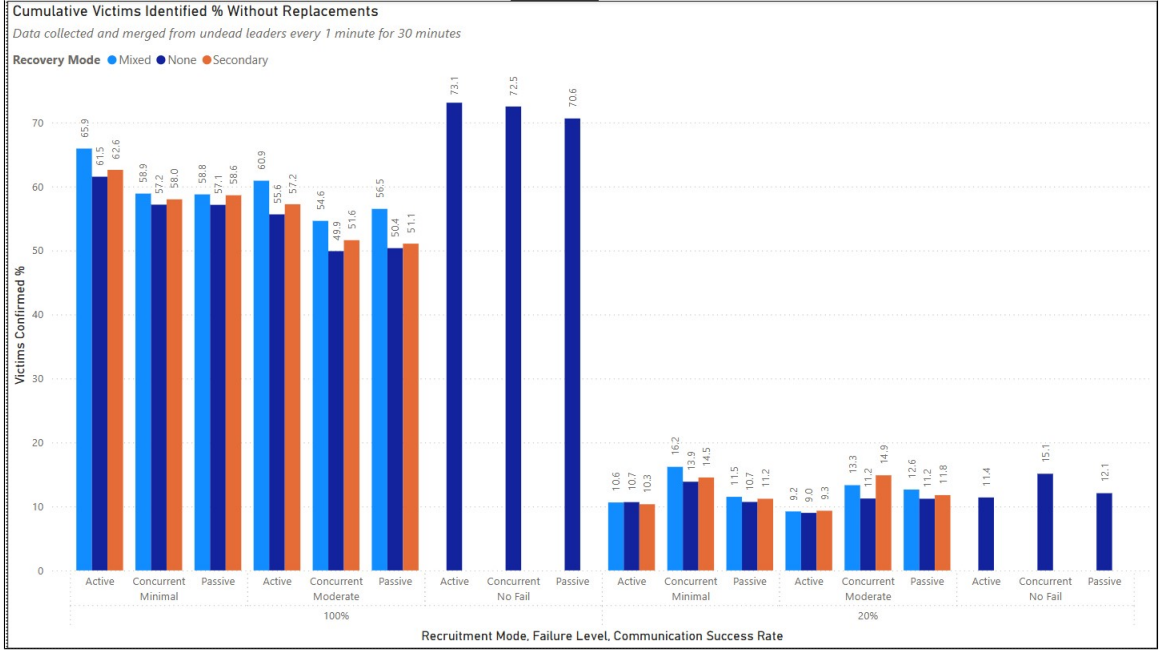


Figure 5.5: Cumulative percentage of victims confirmed without replacements.

about the other team members. With my recovery framework, robots can attempt self-recovery, manage attributes and roles independently and instantly. So, whenever robots can establish a connection, they are sharing the most up-to-date information.

While in most cases the percentage of victims identified is noticeably higher with recovery, there are some situations where this is not the case. The cumulative results show that the recovery framework has almost no effect for trials with poor communication and an active recruitment mode. In fact, the secondary recovery phase performs even worse for minimal failures. This may be because, on one hand, the poor communication hampers the information exchange between robots, which includes the attribute and role adjustment information. Leaders may keep sending task requests according to their own knowledge of team members and roles, and not the most up-to-date information. In active recruitment mode, robots stop their current

tasks to recruit for important tasks like victim confirmation. Robots may suffer failure while actively searching for robots, but due to poor communication, fail to report to leaders or to delegate the task to someone else. As a result, the robot fails to recruit someone and has wasted efforts in searching in vain for robots when it could be doing its own tasks. Moreover, the secondary recovery phase may create a lack of skills and vacant roles in the team, resulting in a higher need for new recruits. The failure to recruit new robots can affect the team's performance. Passive or concurrent recruitment modes do not require robots to abandon their own tasks to search for new recruits, so efforts are not wasted. Moreover, even robots with lost mobility can recruit other robots, while this is not possible in active recruitment mode, since robots have to move to look for robots. Results also show a 2-4% improvement for concurrent recruitment mode and only $\leq 1\%$ improvement for passive recruitment mode under adverse conditions like poor communication, no replacements, and moderate failure level (see Figure 5.5). These levels of improvements suggest that my recovery framework performs better with concurrent recruitment mode, where robots have a better chance at conveying necessary information since it enables consistent periodic recruitment requests.

I also collected combined cumulative data from all dead and live leaders for statistical purposes. This data is similar to the cumulative data collected from only live leaders. This may be because the number of dead leaders is so low that the difference between the two cumulative data sets is unnoticeable. The figures are presented in Appendix B.

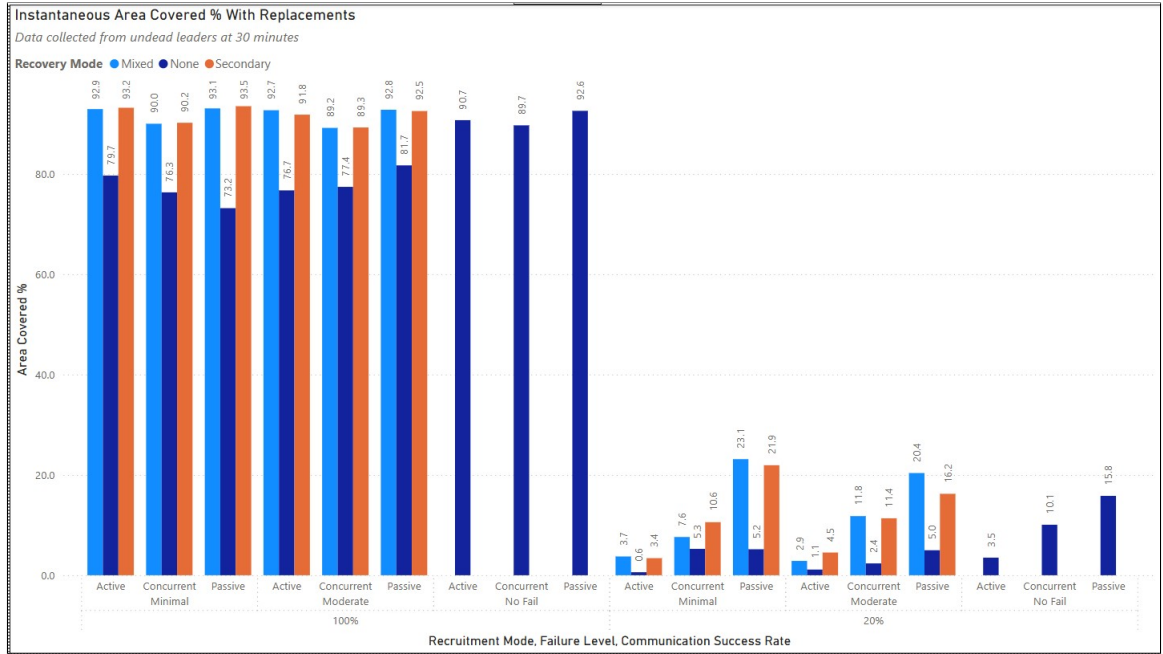


Figure 5.6: Instantaneous percentage of area covered with replacements.

5.6.2 Area Covered

Similar to the victims' data, the area coverage data is collected only from leaders (details about metrics can be found in Section 5.3). Figure 5.6 and 5.7 show the instantaneous data on the percentage of area covered for trials with and without replacements, respectively. The data was collected once at the end of each trial from all active leaders. Figure 5.8 and 5.9 show the cumulative data on the percentage of area covered for trials with and without replacements, respectively. This data was collected and merged every 1 minute for 30 minutes or the duration of the trial.

The instantaneous data shows a higher percentage of area covered in trials with recovery compared to the trials without recovery and with failures. For trials without replacements, there is a 40-45% drop in the percentage of area covered in trials with

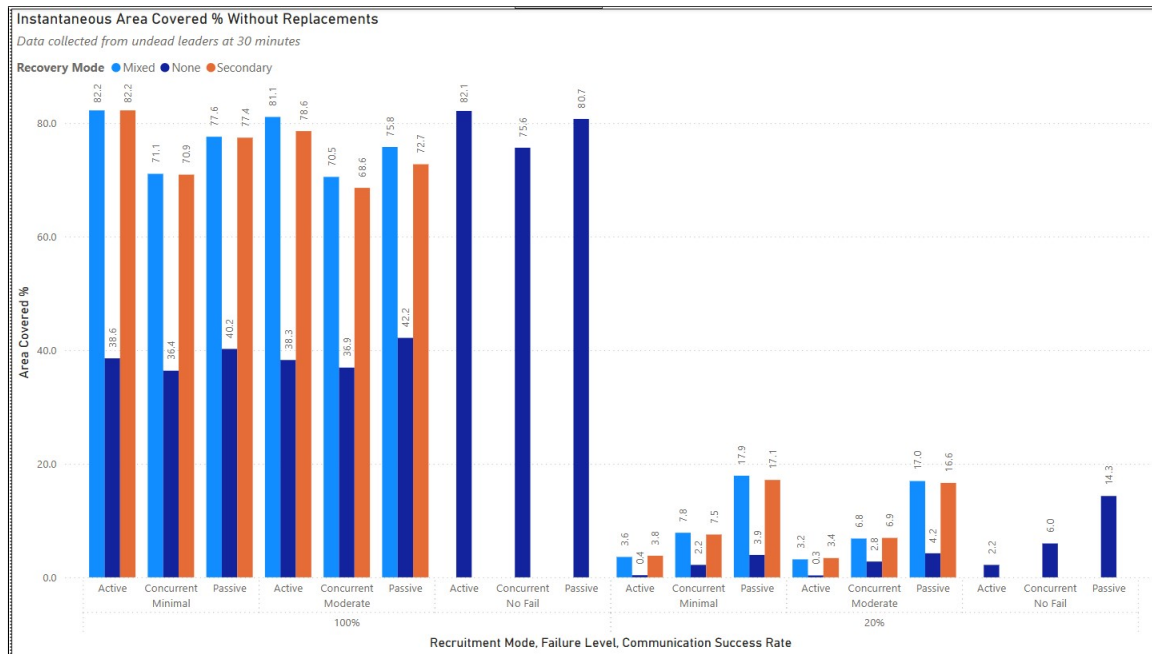


Figure 5.7: Instantaneous percentage of area covered without replacements.

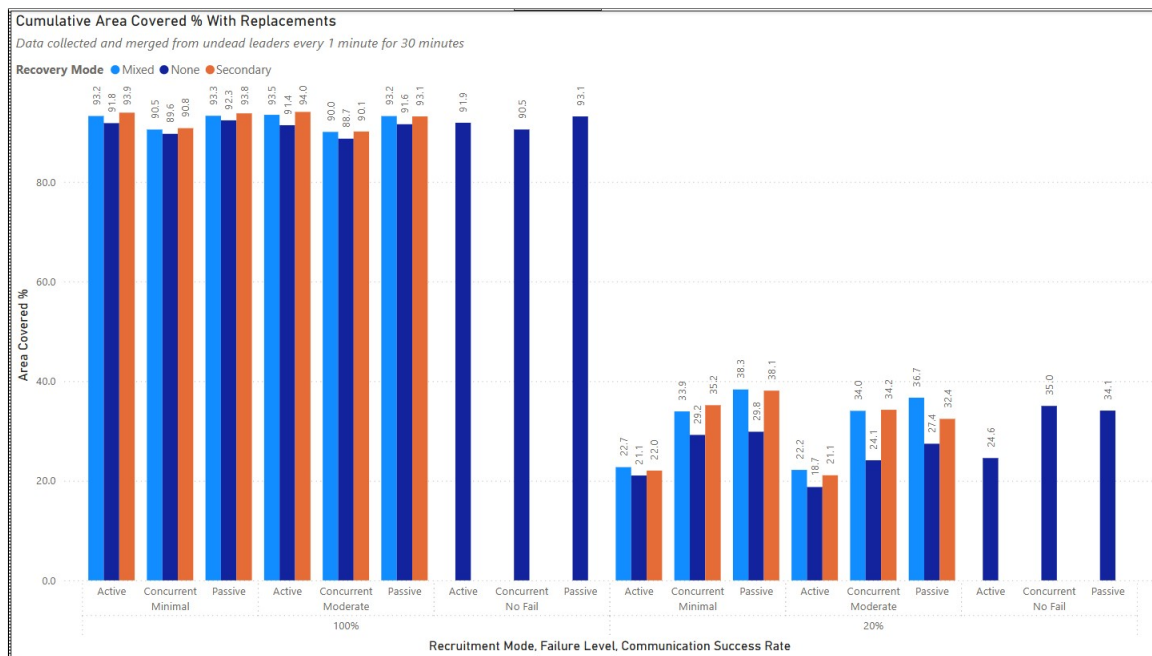


Figure 5.8: Cumulative percentage of area covered with replacements.

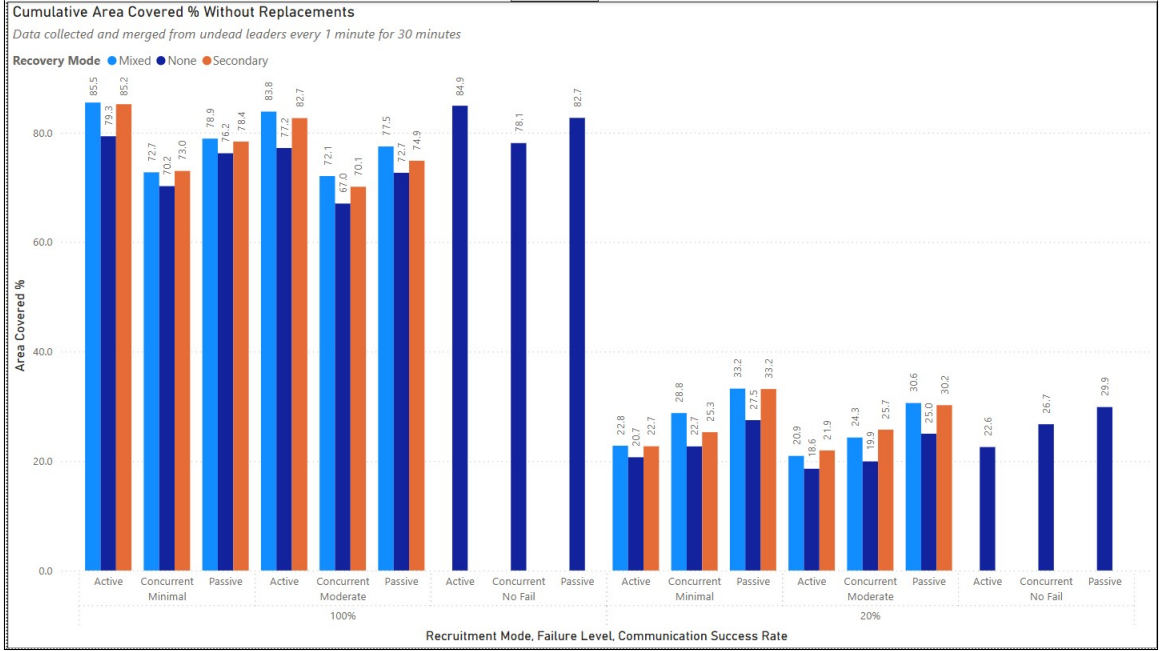


Figure 5.9: Cumulative percentage of area covered without replacements.

failures without recovery compared to the baseline. The drop is not as significant in trials with replacements. This is because almost all robots have basic area exploration skills. So, additional replacement robots, no matter what kind, add to the number of robots able to explore the environment. This results in more area covered. In both cases (with and without replacements), the trials with robot failures and recovery management strategies often covered equal or more area than the base case trials, where there are no robot failures at all. This is because of how common the exploration skill is among robots. Whenever there is a robot component failure, in the secondary recovery phase, the robot re-adjusts its attributes and roles so that it does not have to perform tasks that require the failed component. After readjustment, robots are often left with the capability of performing only exploration-type tasks. For example, if a fire robot loses its fire-sensing capabilities, it can now only do

exploration-type tasks like exploring the frontier, or finding robots for recruitment, etc. With additional replacement robots, there are more robots capable of doing exploration tasks, resulting in a higher percentage of area covered.

Both mixed and secondary recovery mode trials showed close area coverage, with secondary recovery mode sometimes showing even better results. This is also because of how common the exploration capability is among the robots in my implementation. After attribute re-adjustments, robots are only left with exploration capabilities, resulting in more area coverage. So, the more the robots resort to the secondary recovery phase, the more there are robots skilled only in exploration tasks. This is true for both instantaneous and cumulative data. The percentage of area covered cumulatively and instantaneously is almost the same for trials with agent recovery.

While observing the improvements in trials with recovery compared to trials with failures and without recovery, it can be seen that the difference in area coverage is the greatest for the active recruitment mode in trials with 100% communication. For example, in Figure 5.9, robots covered about 6-7% more area with recovery than without recovery (with failures). Contrarily, with poor communication, the improvement was the least for the active recruitment mode. The more there are attribute and role adjustments due to robot failures, the more likely it is for the team to have vacant roles or a lack of skills. As a result, there will be more recruitment tasks. Active recruitment mode requires robots to actively look for other robots for recruitment, covering a larger area. However, with poor communication, fewer recruitment tasks are successfully assigned. Even if they are, robots may lose communication with their team while wandering in the environment looking for new recruits. So, the newly

acquired information about the environment does not often get reported back to the leaders successfully.

Although the recovery framework shows improvement in most scenarios, the success of the recovery framework is also more noticeable in poor communication environments (for example, Figure 5.8). Under poor communication, it becomes harder for the leader to manage the team members. Fewer tasks are assigned. Robots often stray away from their teams and form their own teams. So, there are more leaders in the environment. More data is collected from more leaders. Even more area is explored when robots lose their special skills and are left with only exploration tasks. All these factors contribute to the noticeable increase in area coverage with recovery mode and poor communication.

The area coverage was lowest for moderate failures. However, robots in trials with the recovery framework still covered more area than trials without recovery. I also analyzed the area coverage data collected from all leaders, dead or alive. The results are similar to the cumulative data collected from live leaders. The figures are presented in Appendix B.

5.6.3 Specialized Task Statistics

To evaluate how robot failures and the agent recovery framework affect specialized tasks, I collected and analyzed some additional statistics on debris removal tasks and fire extinguishing tasks. Figure 5.10 and 5.11 portray the average number of debris units removed by debris bots while exploring the environment, with or without replacements. Figure 5.12 and 5.13 portray the average number of fires extinguished

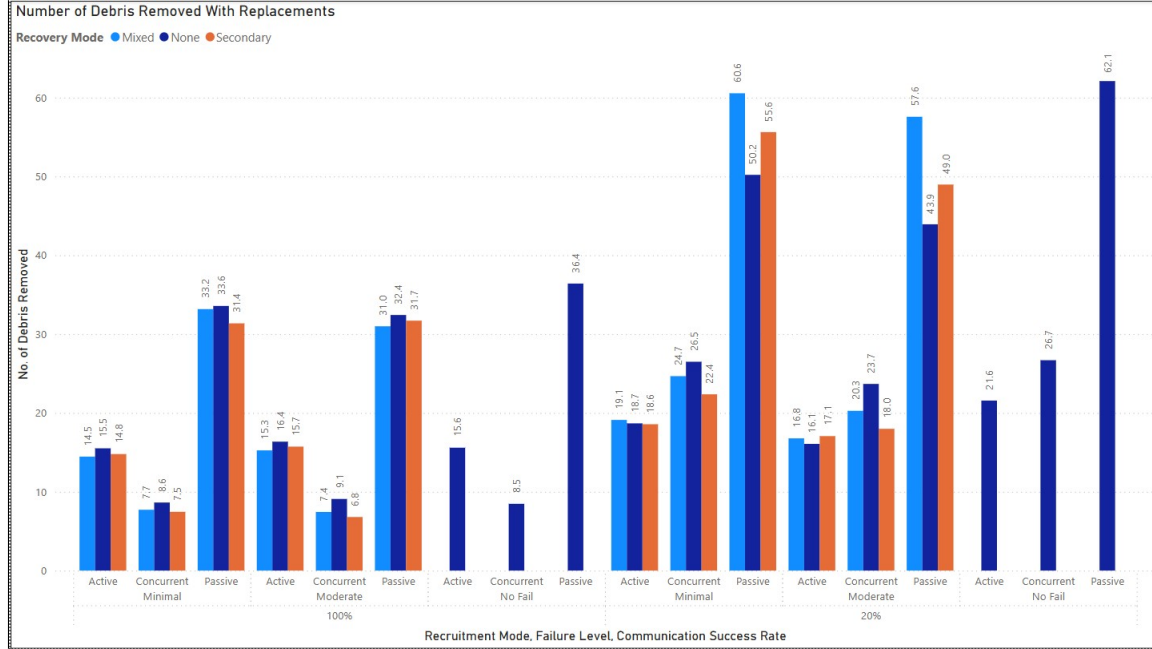


Figure 5.10: Average number of debris units removed with replacements.

by fire bots while exploring the environment. This data is collected for statistical purposes only. So, the data is collected from all debris bots and fire bots in the environment. In the real world, human rescuers are more interested in the victims and the area covered, for which we have to consider the status of robots: dead or alive.

From the debris removal data, the agent recovery framework, especially the secondary recovery phase, performs slightly worse than the trials without recovery for most cases. This is more noticeable for trials conducted with replacements or with good communication. Damaged debris bots lose their debris-clearing skill due to failures in components used in this task, like sonar sensors, damage rays, etc. While they can still do other tasks like exploration, recruitment, etc., they cannot do debris cleaning tasks. So, there is a lack of robots that can clean debris. The debris cleaning

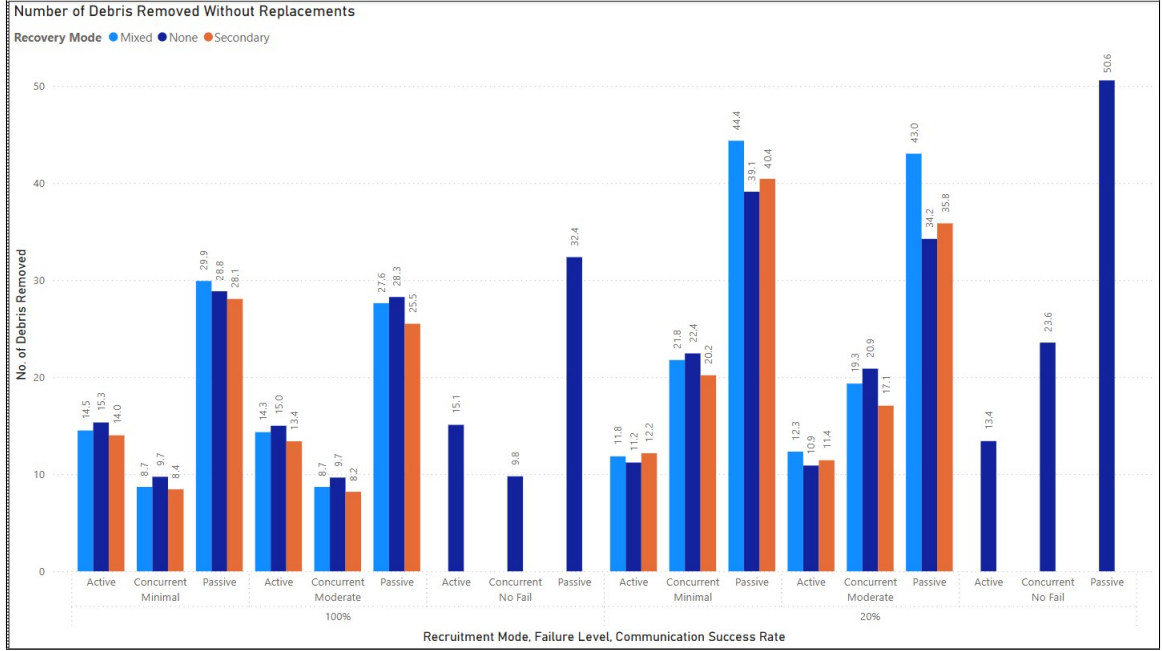


Figure 5.11: Average units of debris units removed without replacements.

task in my implementation requires at least a 95% task suitability. The framework tolerates about 5-10% attribute fail rate in robot components. A higher fail rate of the sensors or debris removal equipment can trigger the recovery mechanism and potentially, a change of attributes and roles. This is because I did not want the robots doing tasks with low accuracy and damaged equipment. These robots can potentially harm victims in the environment by mistaking them for debris. However, in trials without a recovery framework, robots keep operating with faulty sensors and debris cleaning equipment. So, the average number of debris units cleaned in these trials is slightly less than the base case (no failures), but slightly better than trials with the recovery framework. Another observation is that equal or greater debris is removed in trials with recovery, under poor communication for active and passive recruitment modes. Poor communication makes it challenging to send task request

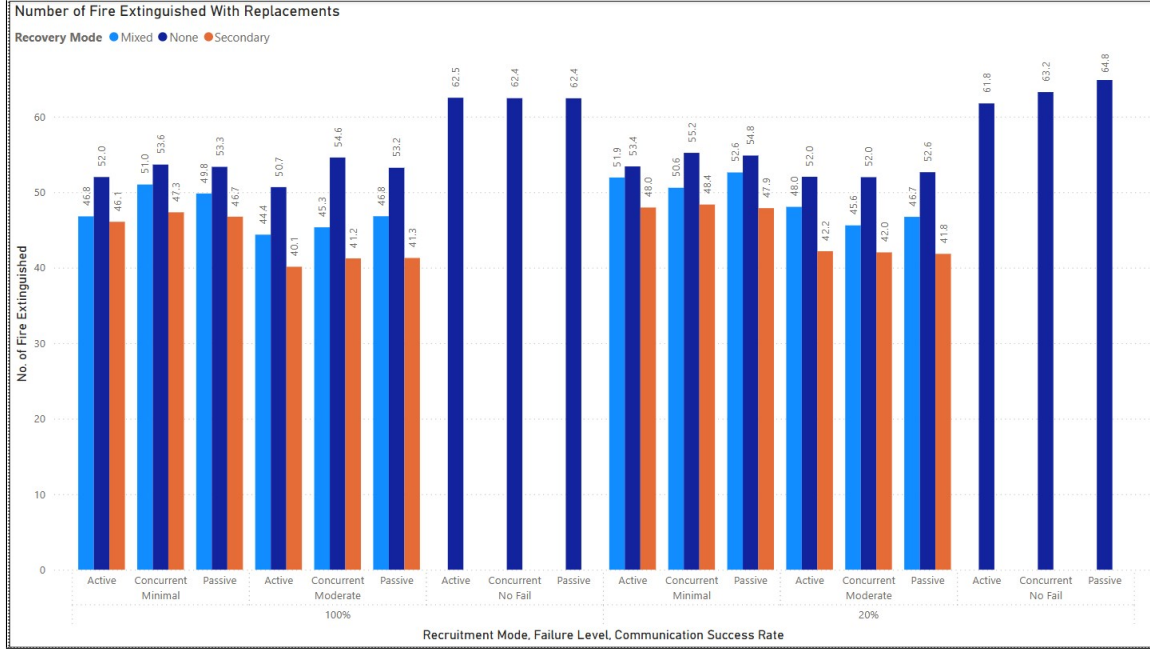


Figure 5.12: Average number of fires extinguished with replacements.

messages. Moreover, passive recruitment mode makes it even more challenging as it relies on chance encounters for task or role recruitments. So, debris bots keep doing exploration and debris removal tasks by themselves. The primary or tertiary recovery improves the chances of full recovery, such that the debris bots can keep doing the tasks with better efficiency.

We can make a similar observation from the fire extinguishing data (Figure 5.12 and 5.13). The trials with recovery mechanisms have a slightly lower number of fires extinguished than trials without a recovery strategy. This is for the same reason that was described previously for debris removal tasks. In trials without a recovery mechanism, FireBots continue finding and extinguishing fires, even with malfunctioning fire sensors or fire extinguishers. For example, they can be operating with even 50% fail rate. However, in my implementation, FireBots will stop doing fire extinguishing

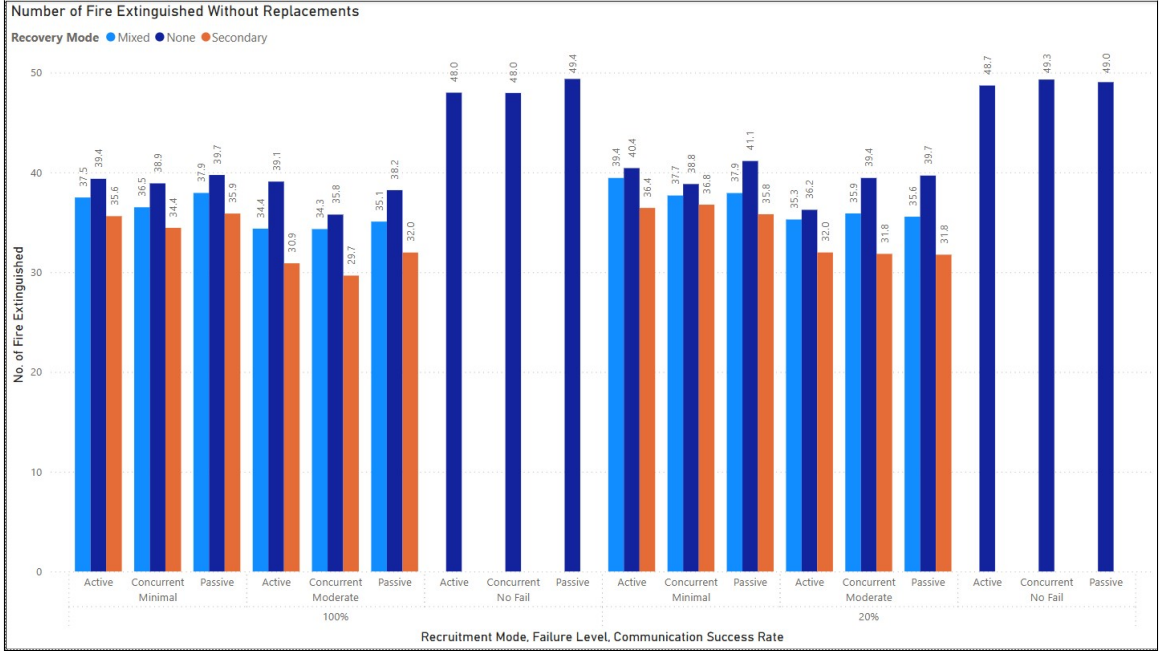


Figure 5.13: Average number of fires extinguished without replacements.

tasks if there is a attribute fail rate greater than 5-10%. For this reason, slightly more fires are extinguished in trials with no recovery mechanism. Another observation is that the passive recruitment mode and poor communication show a similar trend in fire extinguishing data, unlike the debris removal data, where the recovery framework performed better. This is probably because of how the environment is generated. An obstacle has a 60% chance of being debris and only a 15% chance of being a fire. There is less fire to extinguish in the environments used in my experiments.

5.7 Analysis

Now that I have described the experimental results individually, in this section, I will do an overall analysis of my framework based on the results while discussing

some strengths and weaknesses of my framework.

The main goal of deploying robots in USAR is to gather information about the victims and the environment that can later help human rescuers rescue the victims. The victim identification and area coverage results from my experiments clearly showed that the robots identified more victims and covered more area with my agent recovery framework, compared to cases where there were robot failures but no recovery management. The improvement is more apparent in instantaneous results than in cumulative or statistical results. The percentage of instantaneous victim data and area data is significantly lower in cases without a recovery framework. This is because the instantaneous data is collected from live leaders only at the 30th minute or the end of each trial, whereas the cumulative data is collected from live leaders and merged cumulatively every minute for 30 minutes. The robot that may have been a leader or alive at the 5th minute, may change role or die at the 30th minute. So, if the data is only collected at the 30th minute, we lose that data. This suggests the importance of frequent data sharing or data collection. To ensure we do not lose important information, robots share critical information with neighbours when they have a critical health state (e.g.: power level below 5%). The cumulative results in trials without recovery are pretty close to trials with recovery. This suggests that, without a recovery mechanism, it is possible to reduce information loss if human rescuers get data from the robots frequently. But it may not be ideal to get data every minute in the real world.

In terms of recruitment mode, robots performed consistently well with the concurrent mode of recruitment for both victim identification and area coverage. They

also performed quite well with passive recruitment. However, they did not perform as well with active recruitment mode, especially for victim identification under adverse conditions (poor communication or without replacements or moderate failure). This is probably because active recruitments require robots to physically look for robots to recruit. Robots stop whatever task they are doing, even if it is a victim-marking task, to look for a new recruit. With poor communication, it is challenging for robots to communicate recruitment messages even if they find a robot. Robot failures like loss of mobility or unresponsive WiFi can make active recruitments even more challenging. Moreover, victims can only be confirmed by robots with advanced victim sensors, which are limited (a total of 4 MidBots in initial teams, 2 replacement MidBots). Whereas, almost every robot can explore the environment and contribute to area mapping. For this reason, only a slight improvement is seen for victim identification in trials with an active recruitment mode.

My recovery framework performed well in identifying victims and exploring the environment, even with just the secondary recovery phase. I ran a set of experimental trials to evaluate the efficiency of the secondary recovery phase individually. There is no doubt, the robots will perform better with primary and tertiary recovery phases, as there is a chance of full recovery. However, the secondary recovery phase relies on adjusting robot attributes, enabling them to continue contributing to the mission, even with partial robot failures. The majority of the time, robots using just the secondary recovery phase performed better than robots with no recovery at all, but slightly worse or the same as the robots with the overall framework (primary, secondary, and tertiary recovery phases). This suggests that the robots may perform

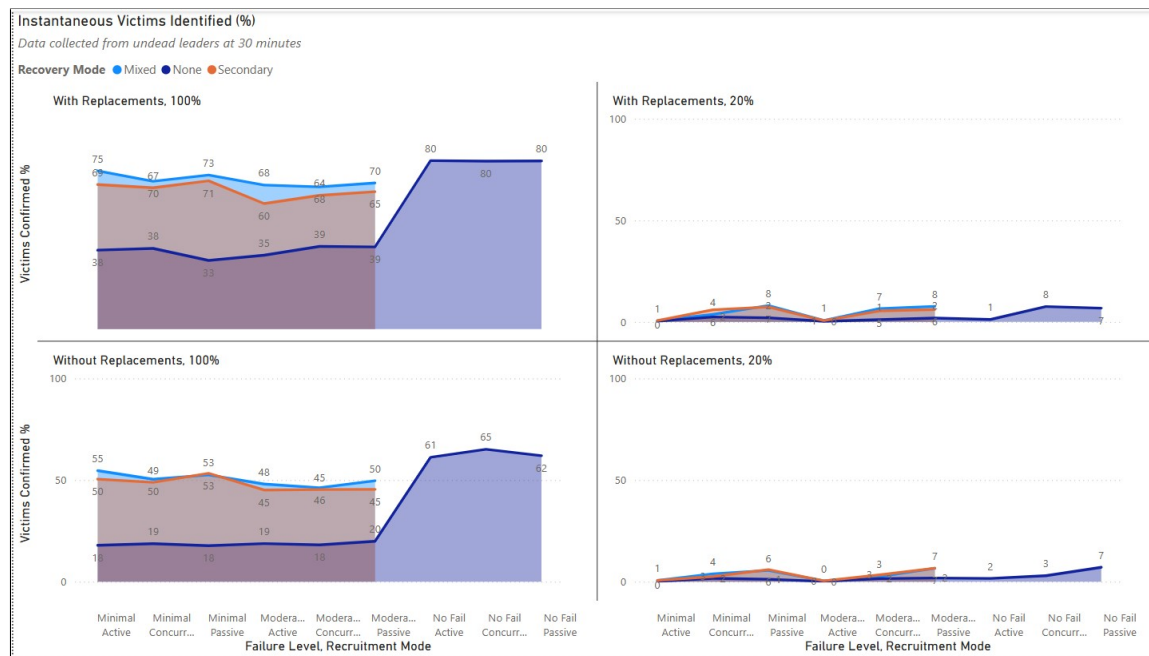


Figure 5.14: Instantaneous percentage of victims identified.

better with just proper attribute, role, and task adjustment without the immediate interference from human rescuers for recovery.

A weakness of my framework is the imbalance of skills that is created in the environment due to the secondary recovery phase. The area charts in Figures 5.14 and 5.15 show that for victim identification, the robots in trials with the secondary recovery perform only slightly better than the trials with failures but no recovery. However, they still perform slightly worse than the mixed recovery. Mixed recovery includes primary, secondary, and tertiary recovery phases, giving the robots a better chance at full recovery. But with secondary recovery, robots can only adjust their attributes and roles to reduce the impact of the robot failure on their future performance. So, when a robot loses a skill due to robot failure, there is a lack of that skill created in the environment. This lack of skills cannot be fulfilled with only random replacements

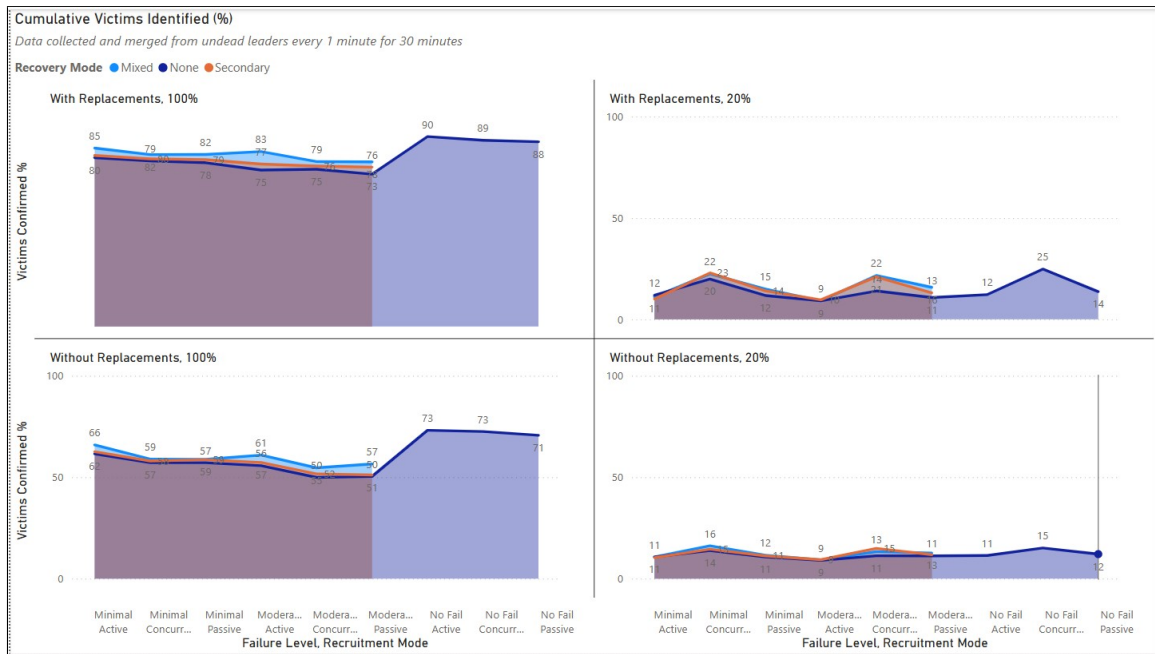


Figure 5.15: Cumulative percentage of victims identified.

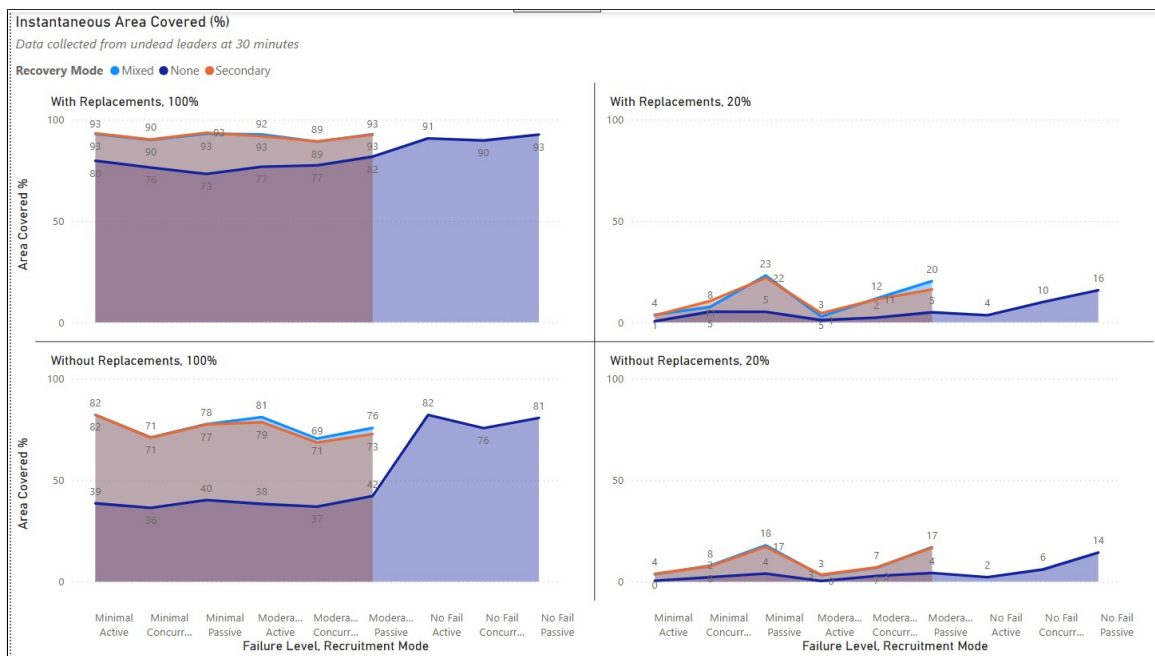


Figure 5.16: Instantaneous percentage of area covered.

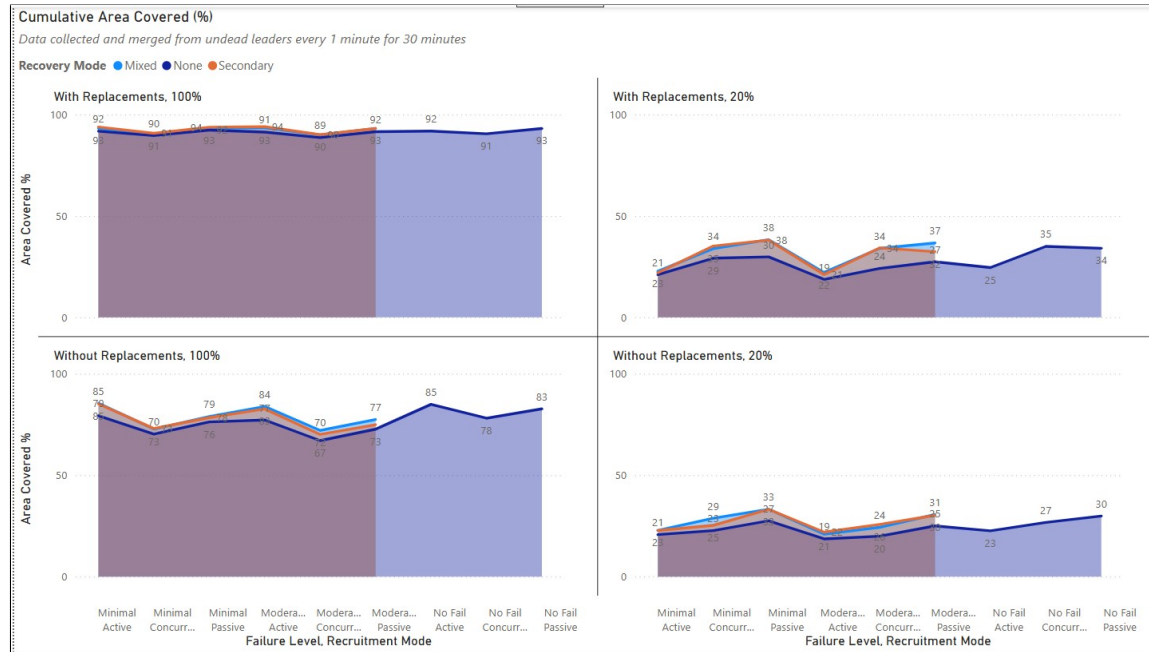


Figure 5.17: Cumulative percentage of area covered.

being sent into the environment. There should be targeted replacements sent in to address the skill shortage. For area coverage (Figures 5.16 and 5.17), robots with only secondary recovery performed almost as well as the robots with mixed recovery. This indicates that there is little to no skill shortage when it comes to area exploration tasks. In my implementation, area exploration is the most basic task that almost any robot can do. If a robot loses its special abilities, like victim confirmation, debris removal, etc., it is often left with the exploration skill after secondary recovery. This leads to a higher number of explorers in the environment. Moreover, the deployment of more replacements adds to the number of robots with exploration skills.

The imbalance of skills is also apparent in the debris removal and fire extinguishing data discussed in Section 5.6.3. Another observation is that the recovery framework does not seem to work as well in the debris removal and fire extinguishing tasks as it

did in the victim confirmation tasks. This is probably because of the robot-to-object ratio. There are only 20 victims in the environment, and the majority of the robots have at least a basic victim detector. There are 2 MidBots per team with advanced victim detector sensors. Besides, victim confirmation can be a team task where a robot with a basic detector marks a potential victim, and a MidBot can confirm the victim. On the other hand, there is only one DebrisBot and one FireBot per team, and a much higher quantity of debris and fire scattered around in the environment. This is why the lack of skill is more apparent in the debris cleaning and fire extinguishing data.

5.8 Conclusion

In this chapter, I first revisited my research questions. Then I described the evaluation metrics, the environment generation and design. After that, I presented and discussed my experimental results. Finally, I provided an overall analysis of my framework while discussing its strengths and weaknesses.

Chapter 6

Conclusion and Future Work

6.1 Chapter Overview

I begin this chapter with a review of my research questions and discuss how my experimental results from Chapter 5 answer them. I then present the main contributions of my thesis. Finally, I discuss the future scope of my work.

6.2 Answers to Research Questions

In this section, I review the research questions from section 1.7 and discuss how my experiments from Chapter 5 answer these questions.

- **What effect does my agent recovery framework have on the overall performance of teams of heterogeneous teams of robots operating in dangerous domains?**

The main goal of robots operating in a USAR mission is to locate victims and

map the disaster site. This information from the robots can help the human rescuers navigate the hazardous environment with greater efficiency. So, mapping the environment and locating victims are some of the high-priority tasks in robot missions in USAR. Results from 5 show that robots identify more victims and cover more area with the agent recovery framework than without the agent recovery framework. Even with just secondary recovery or attribute and role adjustments, robots can continue to contribute to the mission when they have partial component failures. For example, a robot with a damaged victim sensor can remove the victim detector attribute from its skill set and still do area exploration tasks using its sonar sensors. Moreover, instantaneous vs. cumulative data shows how important it is to manage important knowledge, such as victim location and area map, to reduce information loss. The difference between data collected with and without the recovery framework is much higher in the instantaneous data than in the cumulative data. The instantaneous data reflects a more realistic scenario, where the human rescuers may retrieve data from robots at long intervals (e.g., 30 minutes or end of trial). On the other hand, cumulative data is collected and merged every minute. It may be unrealistic to expect human rescuers to retrieve information from robots at such small intervals, especially in a challenging domain such as USAR. The difference between instantaneous and cumulative data suggests that the more frequently data is retrieved from robots, the less likely it is to lose data. Another aspect of my recovery is the enablement of tertiary recovery by human rescuers. The secondary recovery phase prepares robots for tertiary recovery by storing and

sharing the location of immobile or low-battery robots with neighbours. Humans can use this information to rescue the robots just as they use the victim's location to rescue victims.

- **When does my framework work best with respect to different factors such as type of recruitment strategies, quality of communication, presence or absence of replacements, etc. ?**

For victim detection and area coverage tasks, experiments showed that the robots performed very well with a mixed recovery mode that included all 3 recovery phases: primary, secondary, and tertiary. With primary and tertiary recovery, robots have a chance at full recovery, while secondary recovery allows a robot to continue doing tasks even if it is partially damaged. For example, an immobile robot can still communicate about recruitments if its WiFi is still working. In terms of recruitment, robots with the agent recovery framework performed consistently well with the concurrent and passive recruitment modes. The performance was only slightly worse or similar to the trials without the recovery framework, which is mostly noticeable in the cumulative data for victim identification under poor communication. The results also show that with replacements, the improvement in area coverage is higher than the improvement in victim identification. This is because of the imbalance of skills created due to the secondary recovery phase, in addition to the unplanned deployment of replacements. There are too many robots with exploration skills and very few with victim identification skills. The robots with the agent recovery framework performed much better with good communication than with poor communi-

cation. This may be because the secondary recovery requires communicating a robot's changed attributes and roles, and also critical information like victim data, robot location, etc. However, robots also perform better with poor communication compared to robots without the recovery framework.

6.3 Contribution

In this section, I summarize the main contributions of my research:

1. A methodology for the detection and diagnosis of failures in robots operating in dynamic and heterogeneous teams. The detection process is generalized to address several categories of failures, like faulty readings vs unresponsiveness or partial vs total, etc. A diagnosis workflow is added that can be triggered periodically or conditionally to monitor robot health.
2. A methodology to recover robots from failures. This methodology allows 3 forms of recovery. The primary recovery gives the robot a chance at attempting self-recovery through robot restarts, recalibration, etc. The secondary recovery adjusts the attributes and roles of failing robots so that their failures have the least effect on the team's performance. Finally, a tertiary recovery phase that requires human assistance to repair or recover the robot.
3. An example implementation of my methodology while demonstrating the effectiveness of my framework for varying recruitment strategies, communication quality, etc.

4. An experimental domain including a robot fault simulator and my framework, that can be used for other future work.
5. An agent recovery framework that can be further improved by future research.

6.4 Future Work

The experimental results from Chapter 5 have shown that the agent recovery framework has helped with an overall improvement in achieving the main goals of a USAR mission, where robot failures are common. While developing and evaluating this framework, I have also come across areas that can be further improved, such as implementation improvement, and other future aspects of the agent recovery framework. In this section, I will discuss some of these potential improvements.

6.4.1 Implementation Improvements

Firstly, it would be much more beneficial to be able to implement and experiment with the help of real-life robots. In the simulation, a lot of details like a robot's physical structure, component failures, failure detection, and other real-life challenges were abstracted away for the ease of implementation and experimentation. The more complex the implementation, the greater the runtime, and the harder it is to repeat scenarios. The Stage simulator was designed as a lightweight and highly configurable robot simulator supporting a large population of robots [Gerkey et al., 2003]. Simulations can be sped up such that even a 30-minute mission can be run in 10-15 minutes. It is expensive and time-consuming to have real-time robots fail and fix them, setting

up the environment with debris and fire, etc. While working with a simulation is ideal in the first stages of USAR research, a real-life implementation can bring to light complexities and issues that we can not learn from simplified simulations. Especially for the evaluation of my framework, working with real-life robots would give me a better idea about how well my agent recovery framework works in handling real robot failures.

Improving the realism in the simulation would help me confirm my findings more convincingly. The current simulation of robot failures and detection is not realistic enough, as the number and type of failures are very limited due to the simplified nature of robots in the simulation. For example, real-world fire detection uses 3 types of sensory inputs: heat, smoke, and visual. However, in my simulation, only visual sensors were used in fire detection. In the future, new abstracted heat and smoke sensor models can be implemented to facilitate the simulation of partial and complex component failures. Recovery of such failures could include fallback mechanisms where the robot can switch to using the functioning sensors only. It would be interesting to see how my agent recovery framework evolves and performs in such cases. Primary recovery is also simplified to facilitate simulation. In my implementation, primary recovery is simulated as a 20-second time-out. The robot is expected to restart or recalibrate, or reset during this time period. But real-life self-recovery may be more complicated. We need to consider the robot's memory state before and after a restart, task and knowledge, etc. Tertiary recovery requires human intervention, which is also slightly difficult to simulate, especially when I am running thousands of simulations. I simulated tertiary recovery for communication failures by sending

the robot back to base and assuming it gets repaired. I also stored and shared the locations of immobile robots to facilitate tertiary recovery.

Originally, the Stage simulator could simulate 1 simple robot at 1000 times faster than real time and about 1000 real robots simultaneously at real time [Vaughan, 2008]. With gradual modification of the Stage simulator by Gunn [2011], Nagy [2016], Fiawoo [2019], and me, for implementation and evaluation of different frameworks, the simulation logic became more and more complicated. Even though the robots are able to perform these complex computations in a single simulation time-step, a real robot may take a longer time to perform these computations. In fact, in some complicated scenarios, the simulation slows down so much that it takes a few hours to complete a 30-minute mission. Stage uses multi-threading that allows non-interacting sections of the world to run concurrently (e.g., robot models, WiFi model, etc.). But there is an additional overhead in maintaining a causal ordering of events to ensure that the simulations are repeatable [Vaughan, 2008]. To reduce the simulation run time, I limited the number of tasks each robot can manage at a time by task types (e.g., self-execution tasks, recruitment tasks, frontier tasks, etc.). This reduced the simulation run time a lot without compromising the performance of robots in the experiments. In fact, it helped in ensuring that the robot had a chance at attempting different types of tasks instead of being too busy doing the same type of tasks. For example, initially, frontier finders used to loop through all frontiers and create frontier exploration tasks at every simulation time step. This led to hundreds of frontier exploration tasks in the frontier finder's task queue. With frontier exploration being a high-priority task, the robot had fewer chances at attempting other task types like

victim confirmation, recruitments, etc. Limiting the number of frontier exploration tasks that can exist in the queue not only improved the simulation run time but also improved task management. There may be other ways to improve the Stage run time and performance. However, the current version of the Stage that includes my and other frameworks (by Gunn [2011], Nagy [2016], and Fiawoo [2019]) is already complicated to debug and manage. It may be beneficial to explore other multi-robot simulators like Koenig and Howard [2004], USARSim [Carpin et al., 2007], etc., for further development and evaluation of my framework.

6.4.2 Future direction of research

Robot fault detection, diagnosis, and recovery are three broad avenues of research which I simplified and generalized into a single framework: the agent recovery framework for heterogeneous teams of robots in dangerous domains. This framework can be further extended to explore the numerous important questions emerging from my research. I will discuss some of these potential topics of research in this section.

The main focus of my work is to develop an overall recovery framework for robots in heterogeneous teams operating in dangerous domains. My framework supports the idea of failure detection, diagnosis, and recovery. It especially provides an elaborate recovery methodology to help recover robots from robot failure. However, to be able to recover from failure, a malfunctioning robot needs to be able to detect failures in itself. For that, I added a naive fault detection process that validates sensor readings by comparing expected and actual sensor readings, or component working state (e.g., if the component is unresponsive). However, real-life fault detection in robots is

much more complex. Fault Detection and Isolation is a popular domain of research in robotics (more discussed in Section 2.3), which dives into the details of robot failure detection in robot teams. One future direction of my work would be developing the detection process to be more efficient and realistic for each robot's components. For example, reinforcement learning can be used to detect anomalies in sensor readings. However, it may be challenging to find adequate training data to train such a machine learning model. It will be interesting to see if it is possible to use simulation software to generate such training data.

Another concern raised during the evaluation of my framework is the imbalance of skills created in the environment due to the secondary recovery phase. When a malfunctioning robot cannot be fixed in the primary recovery phase, the robot's attributes are adjusted as a part of the secondary recovery phase to avoid using the failing components in its future tasks. The robot adjusts its role during the role swaps or checks so that it does not receive any task that may require it to use its failing component. For example, a FireBot with a damaged fire extinguisher cannot put out fires. So, it changes its role from FireBot to the next most suitable role based on its existing capabilities and task suitability. In my implementation, such a FireBot would change roles to being just an explorer due to its exploration capabilities. But this means there is one less fire extinguishing robot and one more explorer in the environment. FireBots are expensive due to their expensive equipment and are very limited in the environment. Only 1 replacement FireBot is sent into the environment in my implementation. On the other hand, almost all robots have exploration skills, and even then, about 10 MinBots are released in the environment as replacements.

The mission may not need that many additional explorers. 1 or 2 more FireBot replacements can be released instead of sending in 10 MinBots. An improvement of the framework would be to send in replacements in a planned manner according to the demand for skills in the environment.

Currently, the task queue is not reassessed after a role change. If a role change occurs due to robot failure, it is highly likely that the robot will fail in doing tasks that were not addressed after the role adjustment. For example, if a robot has malfunctioning victim sensors and has changed role to an explorer as secondary recovery, there may still be victim confirmation tasks left in its queue that the robot will fail to accomplish. Moreover, there is a chance that the robot will incorrectly locate victims and that the incorrect location will be shared among its team members. Eventually, someone will stumble upon the actual victim and store the location, but the incorrect location will also be in the victim list. A future development of my framework can include the reassignment of faulty component-related tasks in the task queue to other capable robots. A robot should check whether it has the ability to perform the task before it executes a task from its queue.

Information exchange during role switches is another potential improvement for my framework. In all previous frameworks and my framework, it is assumed that the leaders have the most complete knowledge of the team, the victims, and the environment map. The difference between cumulative and instantaneous results from Chapter 5 suggests that there is some information loss due to leader role changes and robot deaths. A robot that may be a leader or alive in the 10th minute, might change role or die in the 15th minute. So, if data is collected instantaneously at the 15th

minute, we may lose information from the 10th minute. My framework addresses the information loss due to predictable robot deaths, like (e.g., the robot may die any minute due to low power level), by sharing critical information with neighbours in the last moments. Another approach to reduce information loss is to retrieve information from robots as frequently as possible. But this may not always be feasible in real life. One other approach to reduce information loss due to role changes is the exchange of critical information. A leader who is switching out of the leader role can share their knowledge as a leader with the new leader. This way, even if the information is not collected cumulatively and frequently, there will be less information loss due to role switches.

6.5 Conclusion

Development of autonomous and heterogeneous teams of robots for dangerous missions like USAR is an interesting but challenging domain of research. Adversities like unpredictable events and environment, poor communication, and limited resources make it difficult for robots to operate. Robot failures are also one of the challenges of operating in such a dangerous domain. I have developed an agent recovery framework to manage robot failures and recoveries in autonomous and heterogeneous teams of robots. My framework takes a generalized approach by combining detection, diagnosis, and recovery of robots, such that it can be implemented in use cases similar to USAR. It includes an elaborate recovery process that allows robots to attempt self-recovery, or attribute and role adjustments, so that they can continue contributing to the mission while minimizing the effects of robot failures. Tertiary recovery or hu-

man assistance in recovery is considered a last resort. Experimentation results have shown considerable performance gains in robots operating with the recovery framework. I have also uncovered potential areas for improvement that can be considered for further research.

Appendix A

Experimental Worlds

Figure A.1, A.2, and A.3 show the worlds on which I conducted my experiments (Chapter 5).

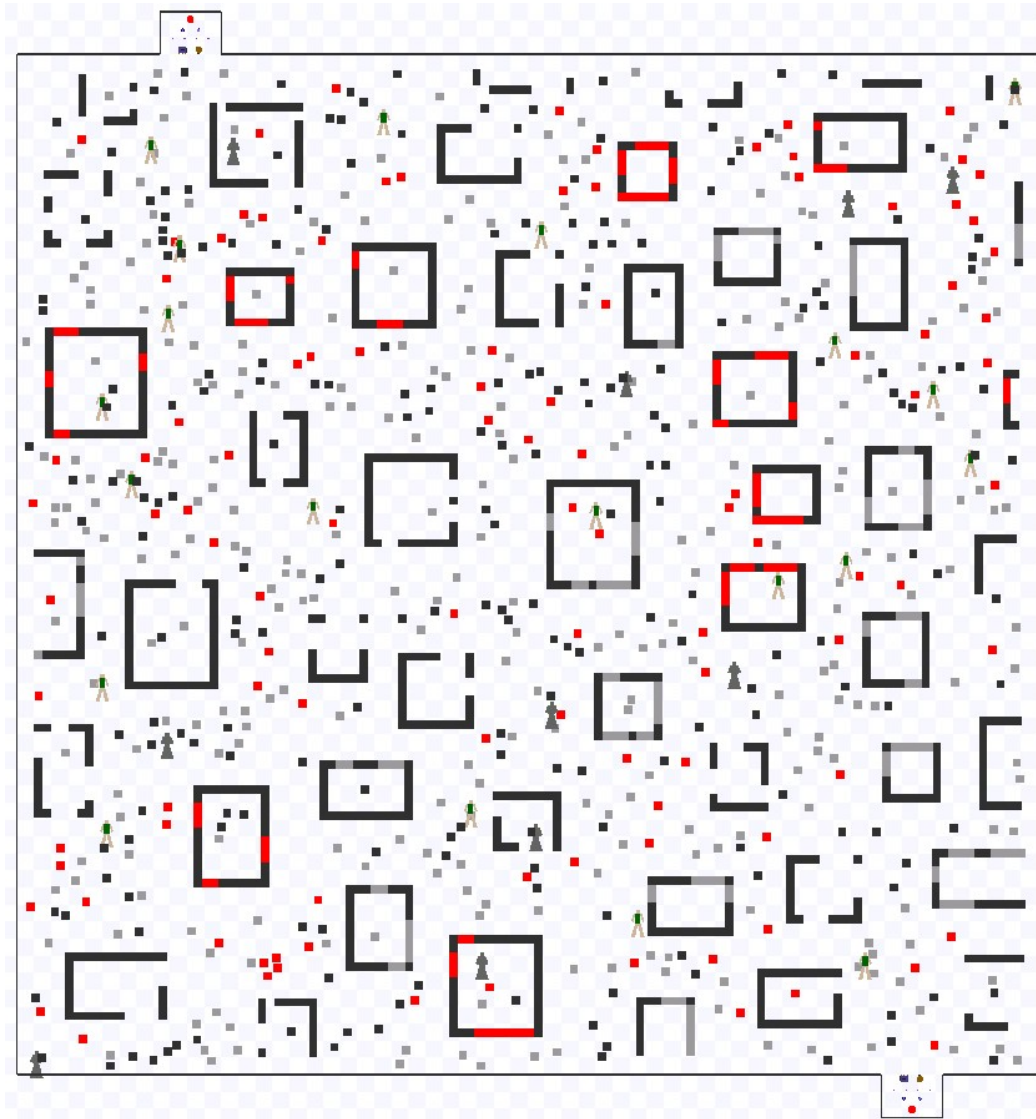


Figure A.1: Experimental World 1.

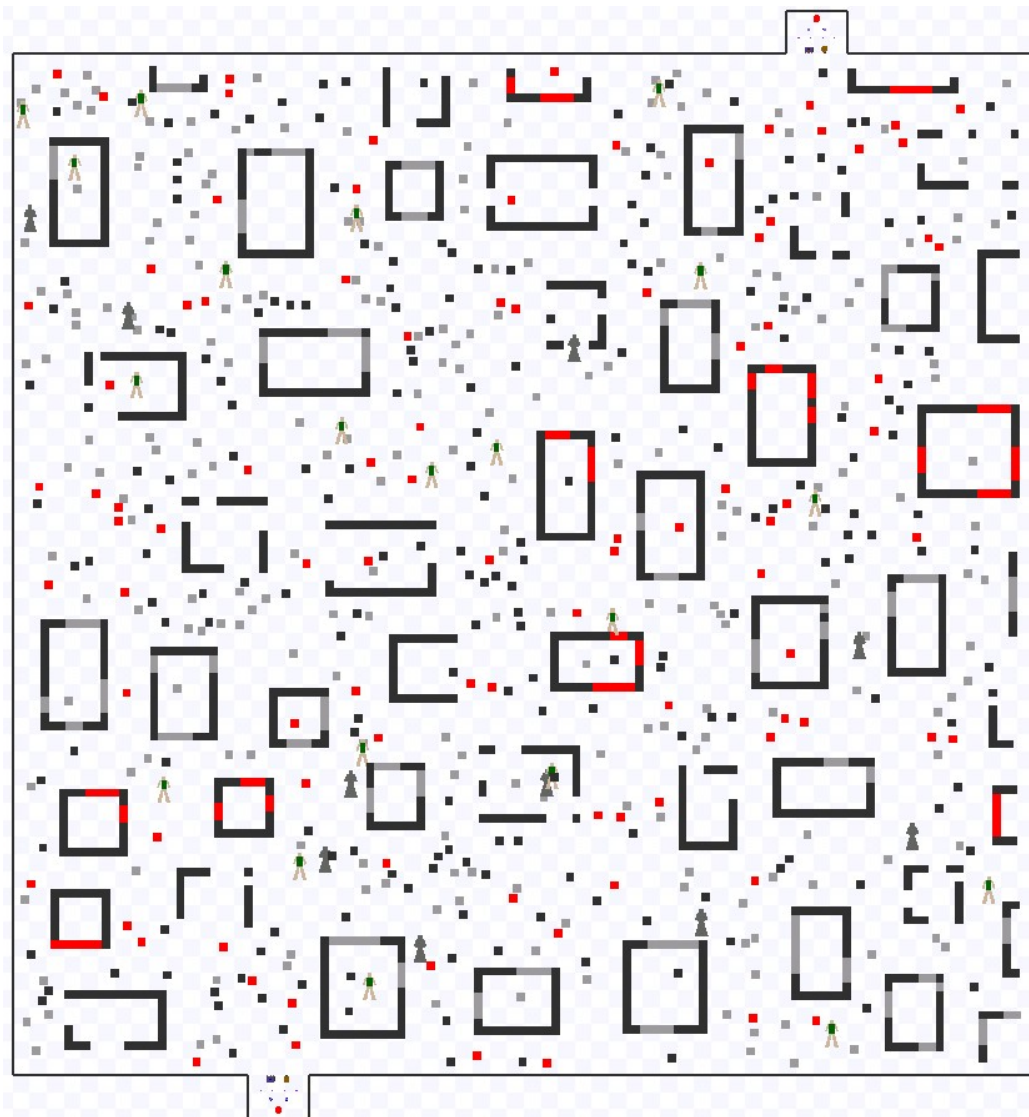


Figure A.2: Experimental World 2.

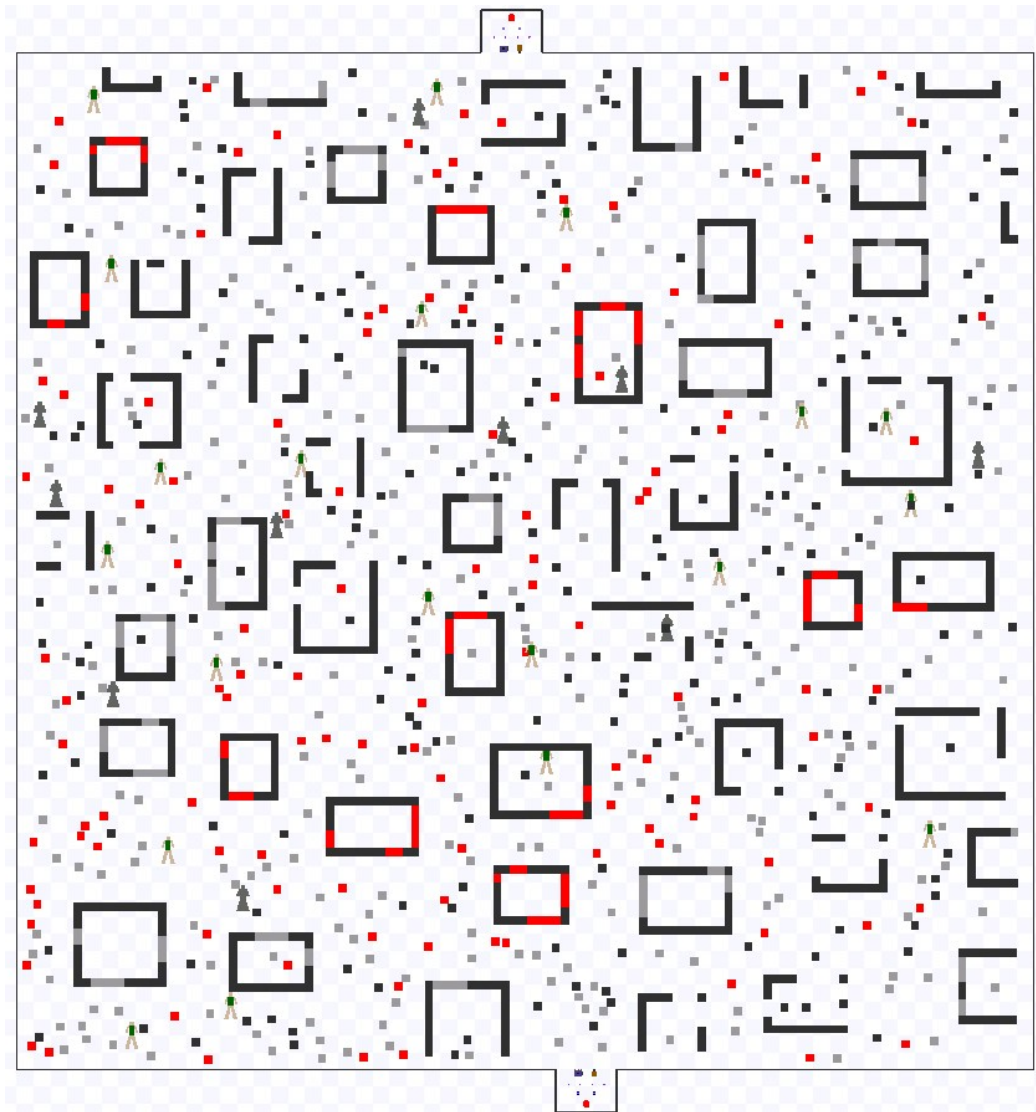


Figure A.3: Experimental World 3.

Appendix B

Additional Experiment Reports

In this chapter, I have included the additional overall statistics that I collected as part of the evaluation of my framework in Chapters 5. These data were cumulatively collected from all dead and undead robots.

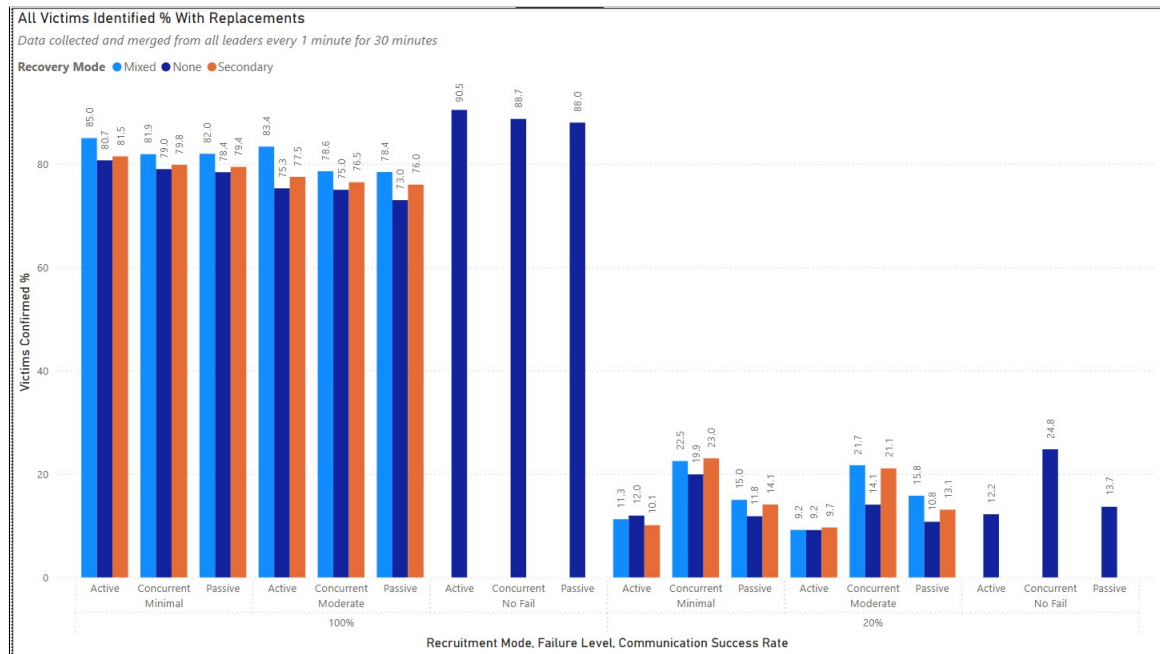


Figure B.1: Cumulative percentage of victims confirmed with replacements. Data collected from dead AND undead leaders.

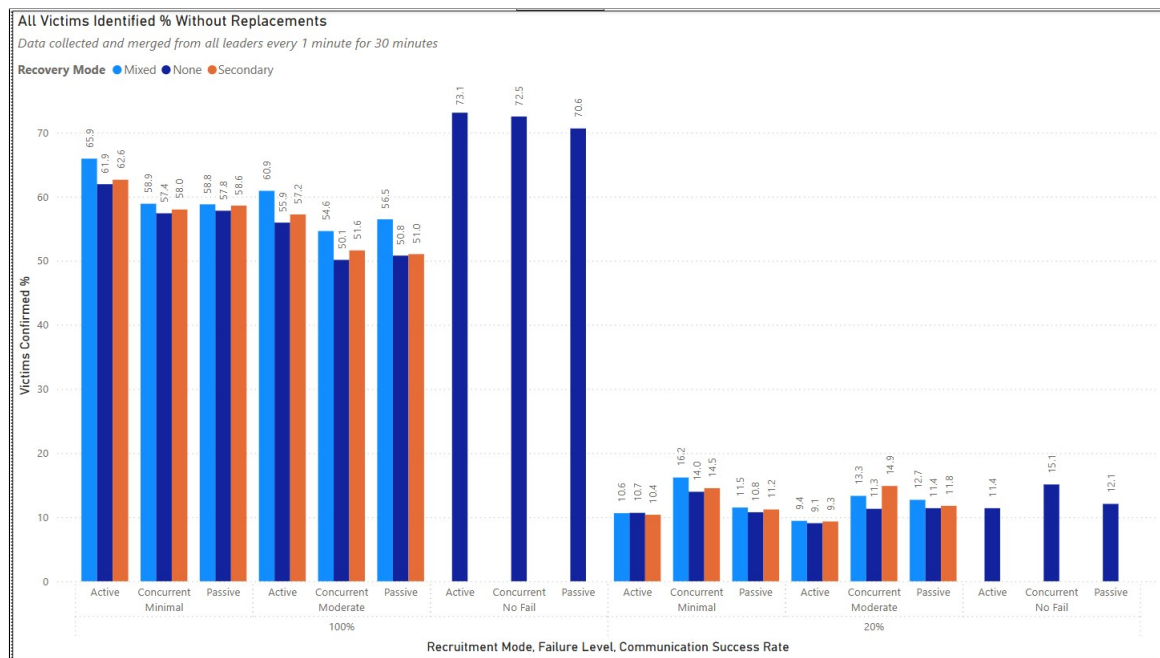


Figure B.2: Cumulative percentage of victims confirmed without replacements. Data collected from dead AND undead leaders.

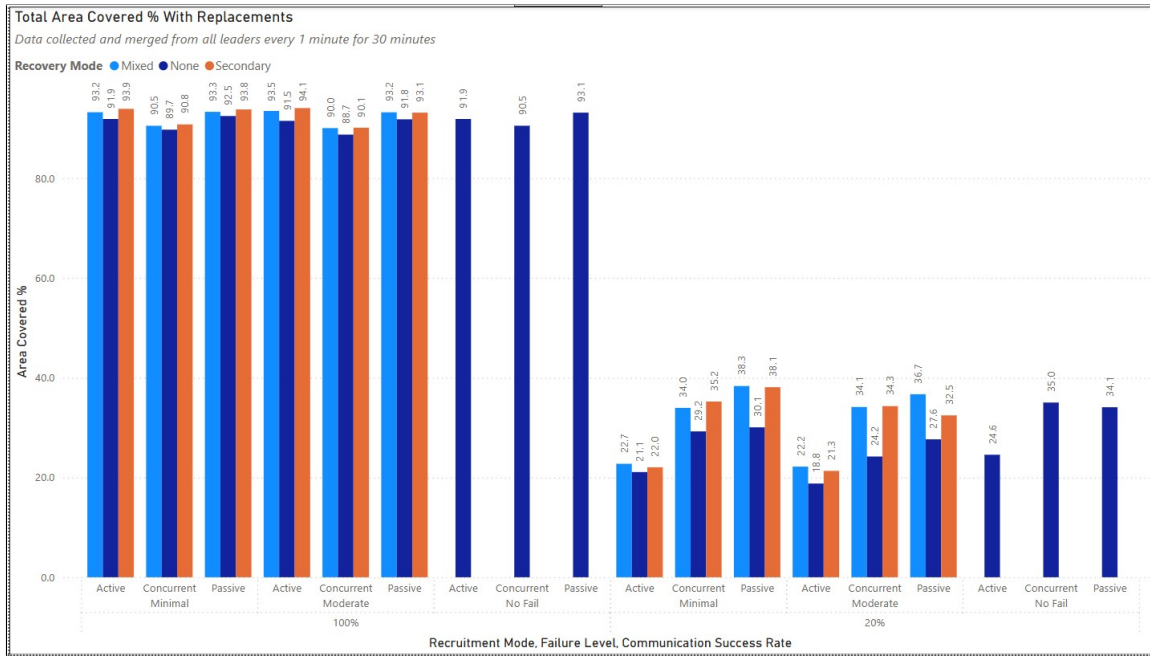


Figure B.3: Cumulative percentage of area covered with replacements. Data collected from dead AND undead leaders.

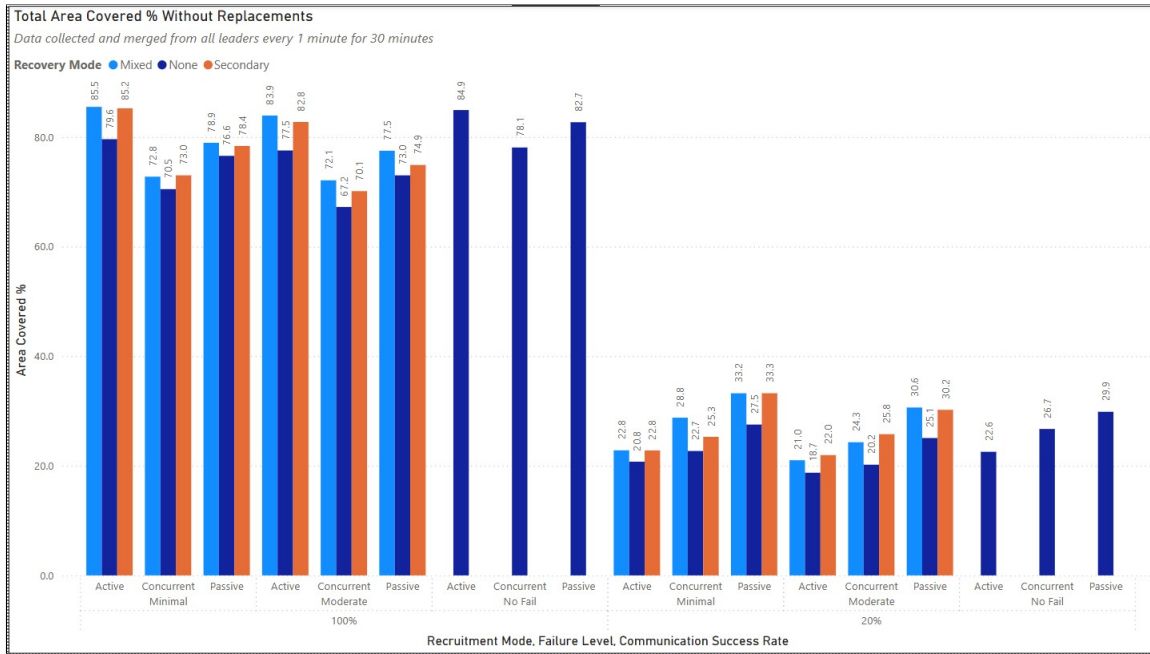


Figure B.4: Cumulative percentage of area covered without replacements. Data collected from dead AND undead leaders.

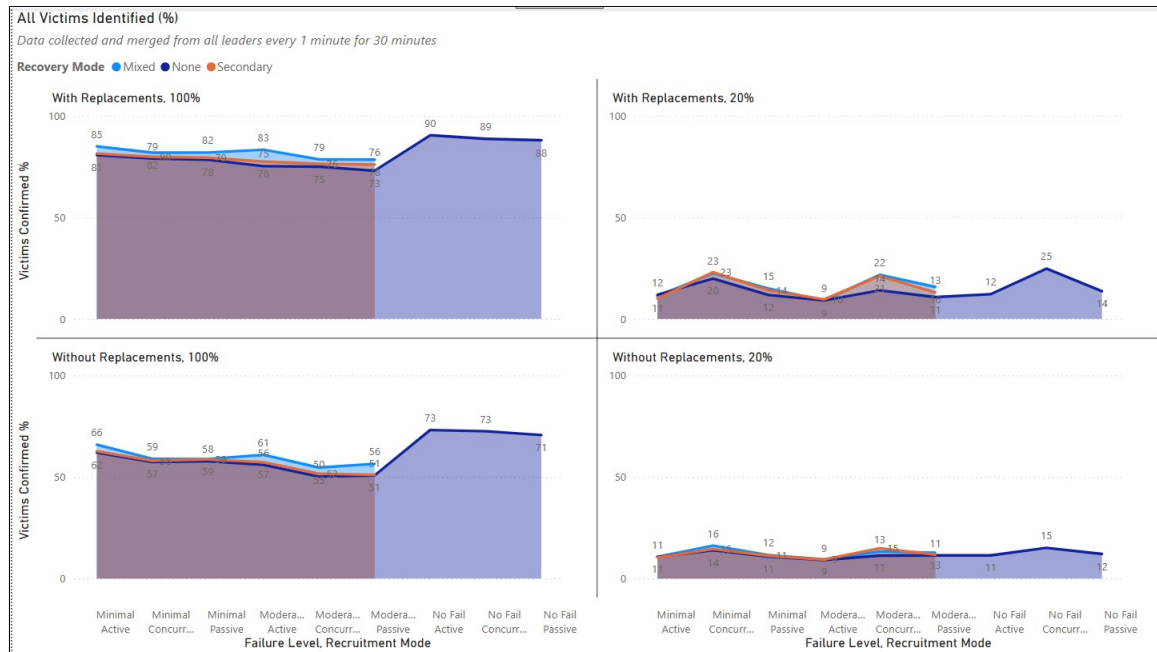


Figure B.5: Overall percentage of victims identified.

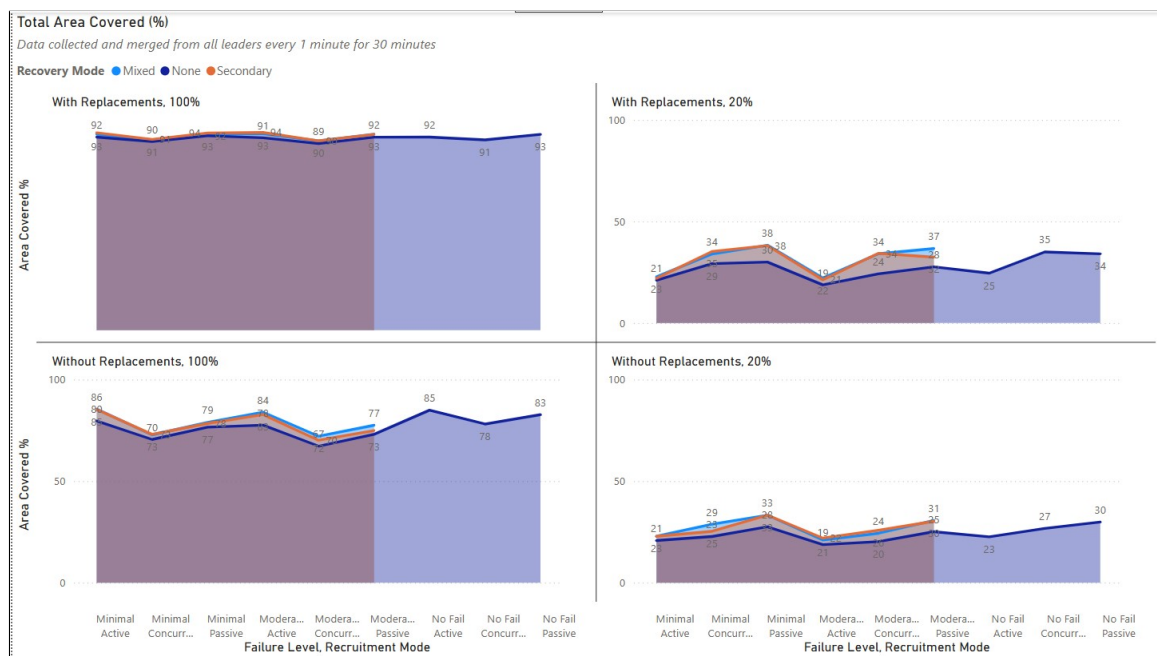


Figure B.6: Overall percentage of area covered.

Bibliography

- F. Arrichiello, A. Marino, and F. Pierri. Distributed fault detection and recovery for networked robots. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3734–3739, 2014. doi: 10.1109/IROS.2014.6943086.
- H. Atay Alam, I. Sariyuce, and S. Noor Haq. Three survivors pulled alive from earthquake rubble in turkey, more than 248 hours after quake. *CNN*, 2023. URL <https://www.cnn.com/2023/02/16/europe/turkey-syria-earthquake-rescue-efforts-intl/index.html>.
- J. Banfi, N. Basilico, and S. Carpin. Optimal redeployment of multirobot teams for communication maintenance. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3757–3764, 2018. doi: 10.1109/IROS.2018.8593532.
- A. Brutschy, N.-L. Tran, N. Baiboun, M. Frison, G. Pini, A. Roli, M. Dorigo, and M. Birattari. Costs and benefits of behavioral specialization. *Robotics and Autonomous Systems*, 60(11):1408–1420, 2012. ISSN 0921-8890. doi: <https://doi.org/10.1016/j.robot.2012.05.018>. URL <https://www.sciencedirect.com/>

- science/article/pii/S0921889012000747. Towards Autonomous Robotic Systems 2011.
- C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. doi: 10.1109/TRO.2016.2624754.
- S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. Usarsim: a robot simulator for research and education. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405, 2007. doi: 10.1109/ROBOT.2007.363180.
- R. Chandra and R. P. Rocha. Knowledge-based framework for human-robots collaborative context awareness in usar missions. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 335–340, 2016. doi: 10.1109/ICARSC.2016.50.
- H. Chitikena, F. Sanfilippo, and S. Ma. Robotics in search and rescue (sar) operations: An ethical and design perspective framework for response phase. *Applied Sciences*, 13:1800, 01 2023. doi: 10.3390/app13031800.
- Y. Cui, J. Lane, R. Voyles, and A. Krishnamoorthy. A new fault tolerance method for field robotics through a self-adaptation architecture. In *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, pages 1–6. IEEE, 2014. ISBN 9781479941995.

- Z. Danping, P. Tan, and W. Yu. Collaborative visual slam for multiple agents:a brief survey. *Virtual Reality Intelligent Hardware*, 1:461–482, 10 2019. doi: 10.1016/j.vrih.2019.09.002.
- F. Dayoub, G. Cielniak, and T. Duckett. Long-term experiments with an adaptive spherical view representation for navigation in changing environments. *Robotics and Autonomous Systems*, 59(5):285–295, 2011. ISSN 0921-8890. doi: <https://doi.org/10.1016/j.robot.2011.02.013>. URL <https://www.sciencedirect.com/science/article/pii/S0921889011000339>. Special Issue ECMR 2009.
- J. de Hoog, S. Cameron, and A. Visser. Dynamic team hierarchies in communication-limited multi-robot exploration. In *2010 IEEE Safety Security and Rescue Robotics*, pages 1–7, 2010. doi: 10.1109/SSRR.2010.5981573.
- J. Delmerico, S. Mintchev, A. Giusti, B. Gromov, K. Melo, T. Horvat, C. Cadena, M. Hutter, A. Ijspeert, D. Floreano, L. M. Gambardella, R. Siegwart, and D. Scaramuzza. The current state and future outlook of rescue robotics. *Journal of Field Robotics*, 36, 08 2019. doi: 10.1002/rob.21887.
- D. Drew. Multi-agent systems for search and rescue applications. *Current Robotics Reports*, 2, 03 2021. doi: 10.1007/s43154-021-00048-3.
- S. Fiawoo. Independent activity and local opportunity for dynamic robot team management in dangerous domains. Master’s thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, July 2019.
- S. Fiawoo and J. Anderson. Balancing team membership and independent activity

- for robots in dangerous domains. pages 414–421, 12 2021. doi: 10.1145/3486622.3493964.
- Z. Gao, C. Cecati, and S. X. Ding. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. *IEEE Transactions on Industrial Electronics*, 62(6):3757–3767, 2015. doi: 10.1109/TIE.2015.2417501.
- B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the International Conference on Advanced Robotics*, 08 2003.
- T. Gunn. Dynamic heterogeneous team formation for robotic urban search and rescue. Master’s thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, December 2011.
- T. Gunn and J. Anderson. Dynamic heterogeneous team formation for robotic urban search and rescue. *Journal of Computer and System Sciences*, 81(3):553–567, May 2015.
- T. Gunn and J. Anderson. Dynamic heterogeneous team formation for robotic urban search and rescue. In *Proceedings of the Fourth International Conference on Ambient Systems, Networks and Technologies (ANT-2013)*, Halifax, Canada, June 2013. Best Paper Award.
- A. Hakami, A. Kumar, S. Shim, and Y. Nahleh. Application of soft systems methodology in solving disaster emergency logistics problems. *World Academy of Science*,

Engineering and Technology International Journal of Industrial Science and Engineering, 7:786–790, 01 2013.

M. Hofbaur, J. Köb, G. Steinbauer, and F. Wotawa. Improving robustness of mobile robots using model-based reasoning. *Journal of intelligent and robotic systems*, 48(1):37–54, 2007. ISSN 0921-0296.

A. Hong, O. Igharoro, Y. Liu, F. Niroui, G. Nejat, and B. Benhabib. Investigating human-robot teams for learning-based semi-autonomous control in urban search and rescue environments. *Journal of Intelligent & Robotic Systems*, 94:669 – 686, 2018. URL <https://api.semanticscholar.org/CorpusID:115585704>.

H.-K. HSU, H.-Y. TING, M.-B. HUANG, and h. Huang. Intelligent fault detection, diagnosis and health evaluation for industrial robots. *Mechanics*, 27:70–79, 02 2021. doi: 10.5755/j02.mech.24401.

A. Husain, H. Jones, B. Kannan, U. Wong, T. Pimentel, S. Tang, S. Daftry, S. Huber, and W. L. Whittaker. Mapping planetary caves with an autonomous, heterogeneous robot team. In *2013 IEEE Aerospace Conference*, pages 1–13. IEEE, 2013. ISBN 9781467318129.

A. Jacoff, E. Messina, and J. M. Evans. A reference test course for urban search and rescue robots. In *The Florida AI Research Society*, 2001. URL <https://api.semanticscholar.org/CorpusID:758506>.

J. Kiener and O. von Stryk. Towards cooperation of heterogeneous, autonomous

- robots: A case study of humanoid and wheeled robots. *Robotics and autonomous systems*, 58(7):921–929, 2010. ISSN 0921-8890.
- N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. doi: 10.1109/IROS.2004.1389727.
- Y. Liu and G. Nejat. Robotic urban search and rescue: A survey from the control perspective. *Journal of Intelligent & Robotic Systems*, 72:147–165, 2013. URL <https://api.semanticscholar.org/CorpusID:20879107>.
- Y. Liu, G. Nejat, and B. Doroodgar. Learning based semi-autonomous control for robots in urban search and rescue. In *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE, 2012. ISBN 9781479901647.
- Y. Liu, G. Nejat, and J. Vilela. Learning to cooperate together: A semi-autonomous control architecture for multi-robot teams in urban search and rescue. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE, 2013. ISBN 1479908797.
- Y. Liu, M. Ficocelli, and G. Nejat. A supervisory control method for multi-robot task allocation in urban search and rescue. In *2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6, 2015. doi: 10.1109/SSRR.2015.7443000.

- M. Long, R. Murphy, and L. Parker. Distributed multi-agent diagnosis and recovery from sensor failures. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2506–2513 vol.3, 2003. doi: 10.1109/IROS.2003.1249246.
- E. Magid. Simulation tools for urban search and rescue robotics. *Proceedings of International Conference on Artificial Life and Robotics*, 2023. URL <https://api.semanticscholar.org/CorpusID:259895291>.
- A. Marino, F. Pierri, and F. Arrichiello. Distributed fault detection isolation and accommodation for homogeneous networked discrete-time linear systems. *IEEE Transactions on Automatic Control*, 62(9):4840–4847, 2017. doi: 10.1109/TAC.2017.2694556.
- G. Nagy. Active recruitment in dynamic teams of heterogeneous robots. Master’s thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, October 2016.
- G. Nagy and J. Anderson. Active team management strategies for multi-robot teams in dangerous environments. In *Advances in Artificial Intelligence: 30th Canadian Conference on Artificial Intelligence*, pages 385–396, Edmonton, AB, May 2017. doi: 10.1007/978-3-319-57351-943. Best Paper Award.
- G. Nagy and J. Anderson. Active recruitment mechanisms for heterogeneous robot teams in dangerous environments. In *Advances in Artificial Intelligence: 29th Canadian Conference on Artificial Intelligence*, pages 276–281, Victoria, British Columbia, June 2016.

- K. Ni and F. Dellaert. Multi-level submap based slam using nested dissection. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2558–2565, 2010. doi: 10.1109/IROS.2010.5650197.
- F. Niroui, K. Zhang, Z. Kashino, and G. Nejat. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019. doi: 10.1109/LRA.2019.2891991.
- L. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998. doi: 10.1109/70.681242.
- L. Parker and F. Tang. Building multirobot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 94(7):1289–1305, 2006. doi: 10.1109/JPROC.2006.876933.
- L. E. Parker and B. Kannan. Adaptive causal models for fault diagnosis and recovery in multi-robot teams. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2703–2710, 2006. doi: 10.1109/IROS.2006.281993.
- G. Qiao and B. Weiss. Advancing measurement science to assess monitoring, diagnostics, and prognostics for manufacturing robotics. *International journal of prognostics and health management*, 7, 12 2016. doi: 10.36001/ijphm.2016.v7i3.2410.
- A. H. Sabry and U. A. B. Ungku Amirulddin. A review on fault detection and diagnosis of industrial robots and multi-axis machines. *Results in Engineering*, 23:102397,

2024. ISSN 2590-1230. doi: <https://doi.org/10.1016/j.rineng.2024.102397>. URL <https://www.sciencedirect.com/science/article/pii/S2590123024006522>.
- C. Schlenoff and E. Messina. A robot ontology for urban search and rescue. In *Proceedings of the 2005 ACM Workshop on Research in Knowledge Representation for Autonomous Systems*, KRAS '05, page 27–34, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 159593202X. doi: 10.1145/1096961.1096965. URL <https://doi.org/10.1145/1096961.1096965>.
- A. Seçilmiş, N. Aksu, F. Dael, I. Shayea, and A. El-Saleh. Machine learning-based fire detection: A comprehensive review and evaluation of classification models. *JOIV : International Journal on Informatics Visualization*, 7:1982, 11 2023. doi: 10.30630/joiv.7.3-2.2332.
- S. Sharmin, S. I. Salim, and K. R. I. Sanim. A low-cost urban search and rescue robot for developing countries. In *2019 IEEE International Conference on Robotics, Automation, Artificial-intelligence and Internet-of-Things (RAAICON)*, pages 60–64, 2019. doi: 10.1109/RAAICON48939.2019.27.
- D. Sivaraman, S. Ongwattanakul, C. Moonjaita, J. Suthakorn, and B. Madhavan Pillai. A pack hunting strategy for heterogeneous robots in rescue operations. *Bioinspiration Biomimetics*, 20, 01 2025. doi: 10.1088/1748-3190/ad9f01.
- S. Solmaz, P. Innerwinkler, M. Wójcik, K. Tong, E. Politi, G. Dimitrakopoulos, P. Purrucker, A. Höß, B. W. Schuller, and R. John. Robust robotic search and rescue in harsh environments: An example and open challenges. In *2024 IEEE International*

- Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–8, 2024. doi: 10.1109/ROSE62198.2024.10591144.
- A. Solórzano, J. Fonollosa, L. Fernández, J. Eichmann, and S. Marco. Fire detection using a gas sensor array with sensor fusion algorithms. In *2017 ISOCS/IEEE International Symposium on Olfaction and Electronic Nose (ISOEN)*, pages 1–3, 2017. doi: 10.1109/ISOEN.2017.7968889.
- M. M. Tousi and K. Khorasani. An optimal hybrid fault recovery for a team of unmanned vehicles. In *2009 3rd Annual IEEE Systems Conference*, pages 241–246, 2009. doi: 10.1109/SYSTEMS.2009.4815805.
- M. M. Tousi, A. G. Aghdam, and K. Khorasani. A hybrid fault diagnosis and recovery for a team of unmanned vehicles. In *2008 IEEE International Conference on System of Systems Engineering*, pages 1–6, 2008. doi: 10.1109/SYBOSE.2008.4724196.
- R. Vaughan. Massively multi-robot simulation in stage. *j swarm intell* 2(2-4):189-208. *Swarm Intelligence*, 2:189–208, 12 2008. doi: 10.1007/s11721-008-0014-4.
- R. Ventura and P. U. Lima. Search and rescue robots: The civil protection teams of the future. In *2012 Third International Conference on Emerging Security Technologies*, pages 12–19. IEEE, 2012. ISBN 9781467324489.
- R. Wegner and J. Anderson. Balancing robotic teleoperation and autonomy for urban search and rescue environments. In A. Y. Tawfik and S. D. Goodwin, editors, *Advances in Artificial Intelligence*, pages 16–30, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24840-8.

Q. Yang and R. Parasuraman. Needs-driven heterogeneous multi-robot cooperation in rescue missions. In *2020 IEEE INTERNATIONAL SYMPOSIUM ON SAFETY, SECURITY, AND RESCUE ROBOTICS (SSRR 2020)*, IEEE International Symposium on Safety Security and Rescue Robotics, pages 252–259, NEW YORK, 2020. IEEE. ISBN 9781665403900.