2016

A New Frontier Based Approach for Multi-Robot Coverage in Unknown Environments

Sushil Parti
University of Windsor

A New Frontier Based Approach for Multi-Robot Coverage in Unknown Environments

by

Sushil Parti

A Thesis

Submitted to the Faculty of Graduate Studies

through the School of Computer Science

in Partial Fulfillment of the Requirements for

the Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

2016

A New Frontier Based Approach for Multi-Robot Coverage in Unknown Environments

by

Sushil Parti

May 24, 2016

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

With the emergence of technology in our daily lives, robots are being increasingly used for coverage tasks which were earlier considered too dangerous or monotonous to be performed by humans such as interplanetary exploration and search & rescue. Out of all the multi-robot coverage approaches, the frontier based approach is one of the most widely used. Most of the coverage approaches developed so far, face the issue of frontier duplication and require access to the maps of the environment prior to coverage. In this work, we have developed a new frontier based approach for multi-robot coverage in unknown environments. This new approach is scalable to multiple robots and does not require prior access to the maps. This approach also uses a new frontier allocation and robot coordination algorithm, which reduces the frontier duplication in the robots and improves the efficiency of robot coverage.

DEDICATION

To my loving family:

Mother: Rama Parti

Father: Raj Kumar Parti

Siblings: Mukesh Parti, Roshika, and Raj Chopra

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the guidance, help and support of several individuals who contributed and extended their valuable assistance in the completion of this work.

First and foremost, I would like to thank my supervisor Dr. Dan Wu for his guidance and support throughout my graduate studies. I would like to thank him for being so friendly and for motivating me through our conversations. One simply could not wish for a better supervisor.

I would like to thank members of my thesis committee, Dr. Imran Ahmad, and Dr. Huapeng Wu for their valuable time, suggestions and constructive comments. I would also like to thank Mrs. Karen Bourdeau, for helping me with all the administrative duties with this thesis.

I would like to thank the open source community, especially the ROS and Stage communities for volunteering their valuable time to create open source tools and tutorials. Without their support, this work could not have completed in a timely and effective manner.

Finally, I would like to thank my family and friends for all their support and motivation in these last two years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS/SYMBOLS

AMCL	Adaptive Monte-Carlo Localization
CFOD	Closest Frontier Obstacle Distance
CMU	Carnegie Mellon University
EKF	Extended Kalman Filters
GUI	Graphical User Interface
LOS	Line of Sight
MRDS	Microsoft Robotics Developer Studio
NASA	National Aeronautics and Space Administration
OpenGL	Open Graphics Library
ROS	Robot Operating System
R.U.R	Rossum´s Universal Robots
RVIZ	Robot Visualization Tool
SLAM	Simultaneous Localization and Mapping
SAIL	Stanford Artificial Intelligence Laboratory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WFD	Wavefront Frontier Detection
YARP	Yet Another Robotics Platform

Chapter 1

INTRODUCTION

The field of robotics has very much evolved far since 1921 when the term robot was introduced in the play R.U.R by the Czech writer, Karel Capek [1]. Since then, robots have been slowly but steadily moved from the realm of science fiction to reality. One of the major fields of robotics research is using robots for exploration of areas which are inaccessible to or dangerous for humans or for activities which are considered too monotonous for humans [6].

After the Fukushima disaster in 2011 in Japan, robots were used to explore the nuclear plant as any operation by humans would have been dangerous due to high radiation levels. The robots provided useful data about the temperature and radiation levels inside the reactor buildings. NASA also uses robots such as Curiosity and Opportunity to explore the surface of Mars. Other applications include search and rescue [2], floor cleaning [3], ocean floor mapping [4], and battlefield reconnaissance [5].

Robots can be categorized into two major categories: remote controlled robots and autonomous robots. Remote controlled robots do not have the ability to analyze their surroundings and then take decisions on their own. They always require humans to provide them with continuous instructions. In contrast to them, autonomous robots are capable of performing their tasks without human intervention. Autonomous robots sense their environments, analyze the data and then respond back. The major advantage of autonomous robots over remote controlled ones is that they do not have to spend a lot of time and energy asking humans what to do and then wait for the response, and this results in benefits in terms of cost and time [7].

The main focus of this work would be on using autonomous robots. Most of the applications discussed so far require the robot to maximize the area covered in the environment. All of these applications are derivatives of the coverage problem, which is a subset of robot exploration [12]. The coverage problem is related to the "covering salesman problem," where the goal is to find the shortest length path covering all the given nodes [8]. Robotics literature is filled with numerous exploration approaches [9, 10, 11, 12, 13], with a focus on using multiple robots to explore known environments.

Using multiple robots instead of one provides multiple benefits, such as the following [14]:

- Improved Performance: multiple robots working in parallel are usually able to complete a task faster than a single robot.
- Increased Fault tolerance: using multiple robots increases the redundancy of the overall system, so that even if one robot fails, others are still able to finish the task.
- Efficient Localization: multiple robots exchange information about their position leading to more efficient localization.

But, these advantages come with several issues which arise due to the use of multiple robots. To overcome these problems, the system requires the following [15, 16]:

- A dedicated communication network for inter-robot coordination.
- Coordination strategy, so that robots cooperate, rather than compete with each other.
- Increased budget due to extra hardware costs.

The environments covered by the robots can be categorized into two categories: known environments and unknown environments. Known environments are the environments for which maps are already available to the robots, but unfortunately, it is not practical to have precise and updated maps for every case. This is especially true in the case of applications to search and rescue, where every second of delay can cause the loss of human lives, and time taken to create a map can be fateful. Robots working in such applications should have the ability to work in unknown environments and perform coverage in them. Coverage of unknown environments requires [6]:

- Simultaneous Localization and Mapping (SLAM) modules for individual robots.
- Map Merging module algorithms to create a global map from individual maps of each robot.

Due to the nature of applications in which robot coverage is employed, time taken to cover the whole environment is the most important evaluation criterion. Another key evaluation criterion needed to measure the performance of robot coverage is the amount of overlap in the robots' coverage. In an ideal scenario, there should be no overlap in the area covered by all the individual robots. But in real life, while performing their individual tasks, robots can overlap areas already explored by other robots. In our proposed approach, the main aim is to minimize the time taken by the robots as well as their overlap during area coverage.

1.1 Motivation

As robots are being increasingly used for a variety of applications, the robotic hardware is changing at a rapid pace. Robots are now equipped with a wide array of sensors such as laser, temperature, pressure, light, sound, altitude and radar [17]. Moreover, as the application and the environment changes, the challenge for software developers becomes to design a system which works on a wide cluster of robotic hardware. To solve this problem, robotic frameworks have been introduced, to create shared functionalities in order to allow code to be reused rather than writing core functionalities from scratch for every new platform [18].

For example, Robot Operating System (ROS) is an open source framework consisting of a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [19]. ROS already has implementations for algorithms such as SLAM, Map Merging, robot localization and robot navigation. However, not many algorithms exist for multi-robot coverage in unknown environments.

Between the year 1994 and 2013, an average of 68000 people died globally every year in natural disasters [20]. Many of these lives could have been saved if search and rescue had been promptly delivered. In many of these cases, the delay was caused because it was still unsafe to send humans for rescue. Autonomous robots present an opportunity to deliver help promptly in these cases.

All of these aspects have motivated us to develop a new frontier based approach for coverage of unknown environments using multiple robots. The robots can start at any random point in the environment and then build their own maps by updating them at each step. These individual maps are then passed to a map merging module which creates a global map. The approach will be tested in a simulator developed through the Stage software platform.

1.2 Problem Statement

The problem of making search and rescue more efficient can be improved by new robotic systems which are faster, more realistic and able to work in unknown environments. Here, realistic systems mean systems which operate in a way which is closer to how a human would do the same work in real circumstances. Our main goal is to develop a new multi-robot coverage strategy for unknown environments which is faster and realistic than earlier approaches. Robots will be equipped with laser sensors to gain information about the environment and simulations will be developed in Stage for multiple 2D environments.

In our proposed approach, individual robots will localize themselves using the Monte Carlo Localization algorithm and then they will create a map of the area in their field of range using SLAM. Robots will then use the frontier technique to choose which direction they should proceed without colliding with the obstacles or the environment boundary. The performance of the approach will depend upon the number of robots and the nature of the environment used for the simulation.

1.3 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 provides an overview of basic terminology and concepts related to this work. It also includes a review of some of the related work which has been already done in this field. Chapter 3 presents an introduction to robotics software frameworks ROS and Stage. In Chapter 4, the proposed multi-robot frontier approach for coverage in unknown environments is introduced. Chapter 5 presents the performed experiments and showcases the results of comparison between our proposed algorithm and of the existing exploration algorithms: Threshold Based and Rank Based. This thesis will be concluded in Chapter 6 with few ideas for future work.

Chapter 2

LITERATURE REVIEW

In order to make this document self-contained, this chapter provides a review of basic concepts, terminology, and related work which has been already done in this field. However, these explanations are not exhaustive and only serve as a guide for the reader.

2.1 Exploration and Coverage

One of the major qualities that distinguishes humans from all other species on this planet is our curiosity to discover new areas, information, and resources. From the ancient times, civilizations have been exploring the universe with most notable periods such as Age of Discovery and Space Race. With the advent of technology, exploration has evolved, with the machines now playing an increasing and more cost effective role in exploring areas which were inaccessible until now or are considered too dangerous for humans and at a much lower cost than humans.

In robotics, exploration is a fundamental problem where the main goal is to maximize robots' knowledge about the environment [6]. Coverage is a subset of this problem where the goal is to completely cover the whole environment. Coverage can be of various types based on the type of robots, nature of environment and the purpose of the application. For example a floor cleaning robot like iRobot Roomba 980 [21] uses a wide variety of sensors like infrared sensor, wall sensor, cliff sensor and a low-resolution camera to clean the whole environment [22]. The environment, in this case, is usually static, which means it does not change while an agent is deliberating [23]. Another type of coverage can be using a cleaning robot to clean a park which is a dynamic environment as people in the park are constantly moving. The main problem in these environments is that paths which were believed to be free earlier can become suddenly blocked and vice versa [24]. This causes the robot to change the coverage strategy and the path planning frequently as the environment changes.

2.2 Navigation and Path Planning

While performing coverage of an unknown environment, we face three major questions related to navigation and path planning that need to be solved first in order to perform coverage [25]:

- Where am I?
- What does my world look like?
- How should I get to my destination?

The first question deals with Localization. Before a robot can accomplish any task, it needs to know its own location in the environment. Localization is the process of estimating a robot's current position and orientation in a given environment. The second question concerns with Mapping. In the case of unknown environments, the robot does not possess the map of its environment and, thus, needs to generate a map. Mapping is the process of constructing a map of the environment using sensor data from the robots. The third question addresses Path Planning. Path Planning is the procedure of finding an optimal path between source and destination.

These three challenges are related to each other and cannot be solved independently [26]. While performing localization through the sensor measurements of the robot, in order to successfully estimate the position of the robot, we need points of reference to link the sensor observations and a map of the environment to estimate the position of the robot relative to the map. Simultaneous Localization and Mapping (SLAM) is the process of performing localization and mapping at the same time and is required in unknown environments where no map is available and it is difficult to estimate the pose of robot accurately. Path planning from the source to the destination requires that the estimation for the robot's location (source) and the information about the environment (destination) are accurate. Active localization is a technique of controlling the robot in such a way as to improve the pose estimation. Mapping the environment uses sensor observations to map the robot's local area at each step and thus requires accurate path planning techniques and pose estimation. The integrated approach to robot coverage works in tandem with all of these three approaches simultaneously to present the best results.

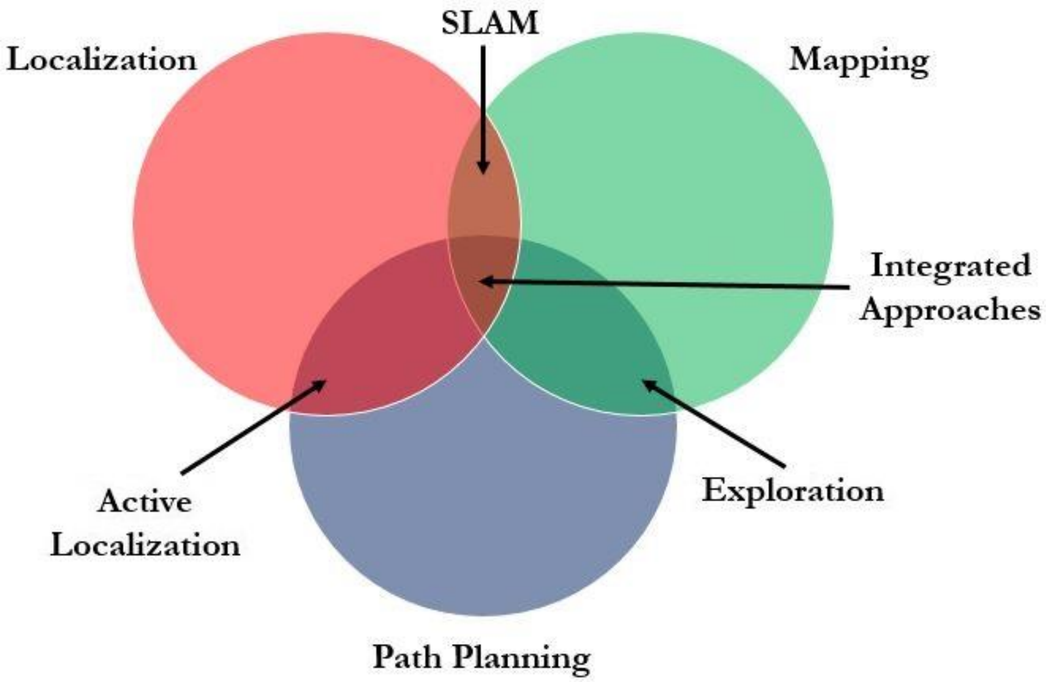


Figure 2.1 Processes required to solve challenges associated with robotic coverage [26].

Fig 2.1 depicts the relation between these three challenges of robotic coverage, how they depend on each other and the processes which lie at the intersections of these processes. The three circles represent the three major processes required for coverage and each circle's intersection with another represents a technique to perform both processes at the same time.

All of these major processes will be discussed in detail in the next sections.

2.2.1 Mapping

In order for a robot to explore, the first critical requirement is that the robot should be able to estimate its position with respect to a fixed frame of reference. Maps allow us to read sensor measurements from the robot and then plot their positions on the map based on various landmarks.

Formally, a map m can be expressed as a list of objects in the environment with their respective properties as given by the equation 2.1 [6]:

$$m = \{m_1, m_2, m_3, \dots, m_N\} \tag{2.1}$$

Here, N denotes the total number of objects in the environment and each m_n with $1 \leq n \leq$ N represents a property of an object. There are three main categories of maps: Feature-based maps, location-based maps, and Occupancy grid maps.

2.2.1.1 Feature-based maps

Feature-based maps, as their name suggests, only represent the features of an environment. They specify the location of objects contained in the map but provide no information about the free space in the environment [6]. Here, n represents the feature index and m_n contains the feature's Cartesian location. These maps are suitable for static environments but do not perform well in unstructured environments, where distinguishing individual obstacles is difficult [27].

Fig 2.2 displays a sample feature based map of an environment. As can be seen in the figure, the map only displays the features of the environment using parametric features such as points and lines and leaves the rest of the environment blank.

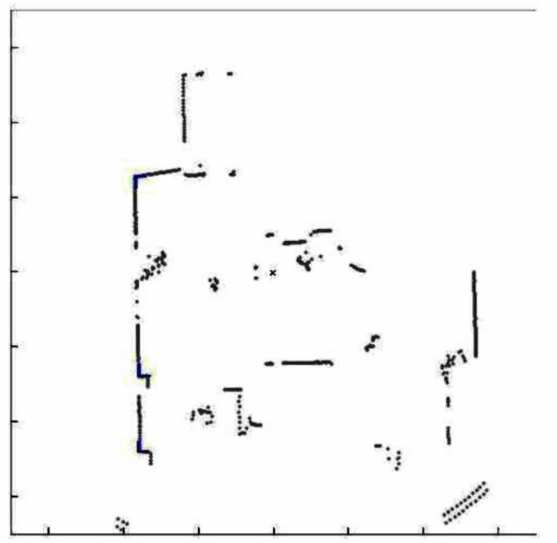


Figure 2.2 Feature-based map of a sample environment [45].

2.2.1.2 Location-based maps

Location-based maps, also known as topographical maps, are volumetric in nature and represent every location on the map. They contain information about both objects and the free space on the map [6]. Here, n specifies a particular location in the map. These maps are depicted by graphs where a node represents landmarks and edges represent the connecting paths. Location-based maps are compact, permit efficient planning and have a lower space complexity, but recognition of places can be ambiguous [27].

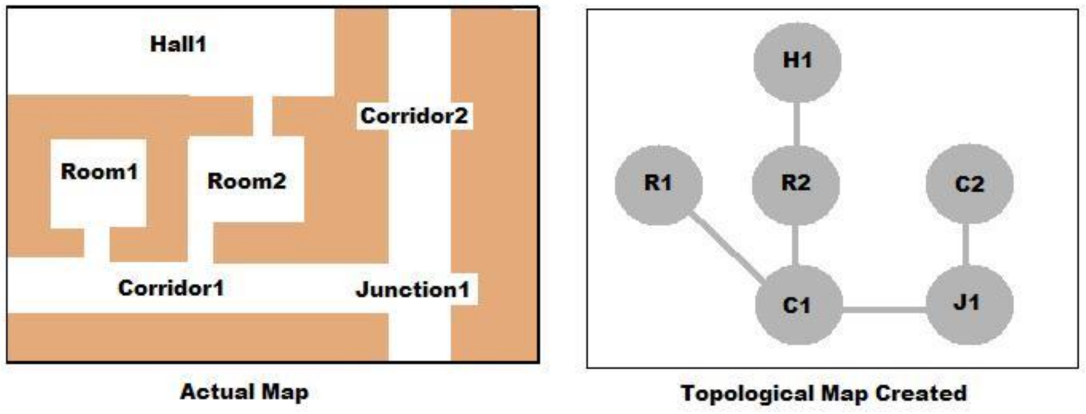


Figure 2.3 Construction of a location-based map [46].

Fig 2.3 depicts the comparison between an actual map of an environment and a topological map of an environment. As it can be seen in the figure, the topological map is a higher level representation of an environment where the nodes represent the information about the features, while edges represent the pathways connecting two features.

2.2.1.3 Occupancy grid maps

Occupancy grid maps are a classical map representation developed in the mid-eighties by Moravec and Elfes at CMU [28]. These maps are suitable for mobile robot navigation and work best with range sensors like sonar and laser. They also allow easier path planning [12] but are not scalable to large environments, taking map construction time into consideration [15]. Occupancy grid maps are represented by cells of the same shape and size. Each of these cells has an occupancy value associated with it based on the probability of the occupancy of that grid cell. Occupancy value can have one of three values:

- **Free**: The grid cell has been explored and has no obstacles
- **Occupied:** The grid cell has obstacles
- **Unknown:** The grid cell has not yet been explored

Fig 2.4 depicts an occupancy grid map for a sample environment. As we can see in the figure, the whole environment is divided into cells of same size and shape. This decomposition of the whole environment allows the algorithm to count which areas are already covered by the robot and thus can provide a measure of the coverage.

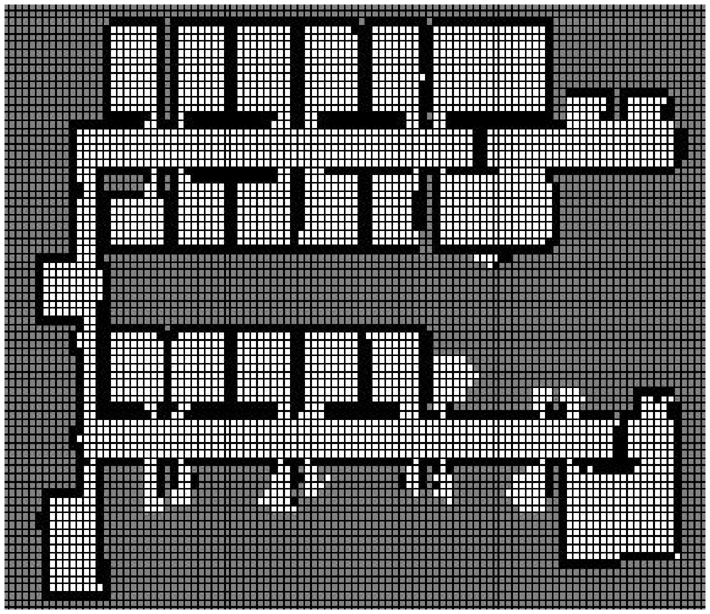


Figure 2.4 Occupancy grid map of an environment [29].

Another major category of maps is hybrid maps which are created by combining other maps. We will not be looking into this kind of maps in detail in this work. In the end, the choice of the map depends on the task to be performed and the nature of the environment.

2.2.1.4 Map Merging

When using multiple robots to perform coverage in an unknown environment, one problem is that each robot performs its own mapping and a need arises to combine these individual maps to create a global map shared by all the robots. The process of creating a global map from multiple individual maps is known as map merging [47]. One of the most fundamental problems in map merging is to combine duplicate regions – regions which are present in multiple maps. The map merging algorithm identifies these regions and then creates a global map without duplicating these regions.

2.2.2 Localization

Localization is the process of estimating the position and orientation of a robot relative to a fixed frame of reference in the given environment. Localization is a fundamental problem in mobile robotics, and in order to perform any major task, the robot must be localized first. As per [6], the problem of localization can be deduced as a problem of coordinate transformation. In any environment, there are two major coordinate systems – the global coordinate system and the robot's local coordinate system. Localization is the process of making a connection between the two coordinate systems.

To establish this connection, the sensor measurements of the robot are analyzed with the environment map repeatedly and over a period of time; the probability of the robot's position increases at some points on the map. Fig 2.5 displays the general idea of how localization is performed in a sample environment. It plots a graph between the robot's beliefs of its position on the x-axis. The peaks show the estimated position of the robot; the higher the peak, the greater the probability of the robot's position.

As we can see in the beginning, the robot's belief is equally distributed, because the robot has no idea of his location. But, as the robot moves forward and finds a door, now his belief has three small peaks as he can be near any of the doors. As he moves forwards and finds the next doors, there is a large peak in his belief that he is near door 2. Thus, the robot has now performed localization and has a strong belief about his location on the map.

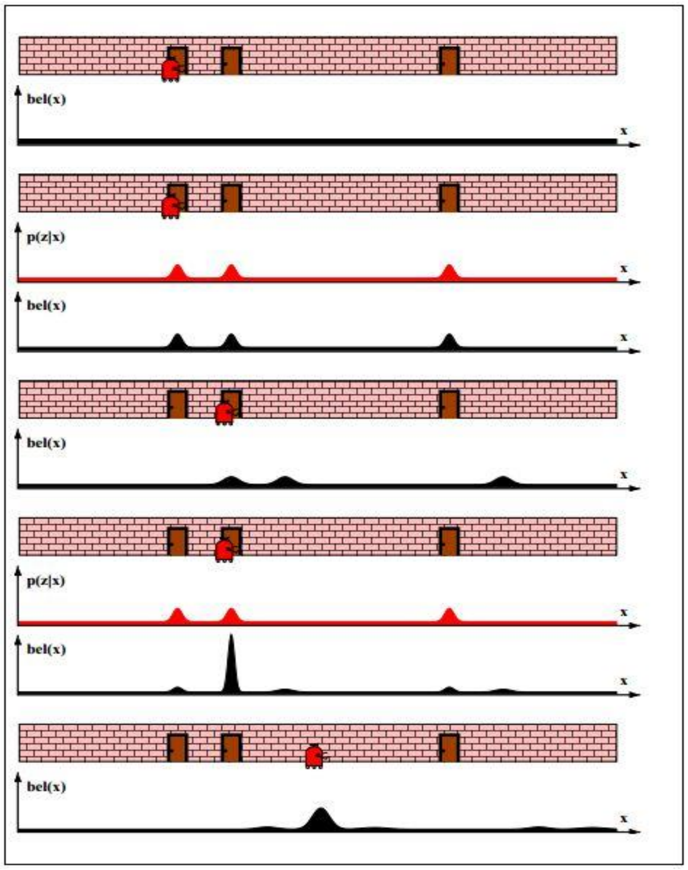


Figure 2.5 Basic idea of robot localization in a sample environment [6].

The problem of localization can be categorized into various categories:

1) **Local vs Global Localization:** Local localization techniques require an approximate estimate of the initial position of the robot. While Global localization can localize a robot without any prior knowledge about the position of the robot [30].
2) **Static vs Dynamic Environment:** Static environments are environments where the robot is the only object which is moving and thus changing its pose. While in dynamic environments, there are other objects than robots which are changing their position and pose. Dynamic environments are more difficult to perform coverage on than static environments.

3) **Passive vs Active Approaches:** In Passive approach, the localization module is limited to only observation of robot's motion. While in Active approach, the localization module has the control over robot's motion and thus can drive the robot in such a way to minimize localization error [6].

4) **Single Robot vs Multi-Robot Approaches:** In single robot systems, localization involves taking sensor measurements into account and then estimating robot's pose and position. While in multi-robot systems, we can use robot's estimates with each other to reduce localization error and time required to localize robots'.

As discussed before, in order to perform localization, some kind of environment mapping should be available. There are two types of environments – known and unknown. Known environments are environments for which maps are available. Unknown environments are those environments for which the robot does not possess any maps. For unknown maps there is two ways localization can be performed:

- First, robots can learn the map in advance in a pre-exploration phase, but is time-consuming
- The second option is to use SLAM, where robots perform both localization and mapping of the environment simultaneously.

2.2.2.1 Simultaneous Localization and Mapping (SLAM)

SLAM is often compared with the chicken-or-egg problem because a map is needed for localization and an estimate of robot's pose is required in order to perform the mapping. SLAM can be categorized into two forms: First is online SLAM which estimates the posterior of robot's pose over the momentary pose along with the map only for time t as depicted in equation 2.2 [6]

$$p(x_t, m \mid z_{1:t}, u_{1:t}) \tag{2.2}$$

Here, x_t denotes the pose of the robot at time t, m is the map and $z_{1:t}$ and $u_{1:t}$ are the measurement and control readings for time 1 to t. Whereas Full SLAM calculates the estimate of robot's pose posterior over the entire path $x_{1:t}$ along with the map instead of just the current pose x_t as depicted in equation 2.3 [6]

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{2.3}$$

There are three main paradigms of SLAM:

- Extended Kalman Filters (EKF) SLAM
- Graph-Based SLAM
- Particle-Based SLAM

In this thesis, the particle-based SLAM is being used to localize the robots and map the unknown environment.

2.2.3 Path Planning

Path Planning is the process of finding a path for the robots to travel from the current position to the goal without any collisions. Path Planning can be divided into two major categories: local path planning and global path planning. As per [31], in the case of global path planning, robots have the map of environment available to them and thus, have prior knowledge about the environment and its obstacles. Another requirement for global path planning is that the environment is static. Due to all these constraints, the path planning algorithm is able to plan the path from source to destination even before the robot starts motion.

In the case of the local path planning, the environment is unknown to the robots in the beginning and thus, there are no maps available. Due to this, the robot is not able to plan the path in advance and the path planning gets updated at every step as the mapping and localization information changes through SLAM [31].

A number of path planning algorithms exist in the literature and the choice of algorithm will depend upon the nature of the environment, mapping and the application of the system. One of the most heavily used algorithms for path planning is the A* search algorithm. Equation 2.4 shows the A* algorithm where $\mathrm{g(n)}$ denotes the total cost from the start node to node n and $\mathrm{h(n)}$ is the heuristic cost from node n to goal node [32].

$$f(n) = g(n) + h(n) \tag{2.4}$$

A* algorithm maintains two lists – the open list and the closed list. At the beginning of the algorithm, the open list contains the start node and the closed list is empty. The algorithm starts expanding the nodes from the open list and then selects the one with minimum cost. The closed list keeps track of nodes which are already visited. The algorithm stops when the goal node is expanded. The A* algorithm is used extensively in the research community because it provides fast results with a low memory cost.

2.3 Multi-Robot Coverage

The next logical step after performing coverage with a robot is to scale the problem to multiple robots. As discussed before, using multiple robots for coverage provides us with advantages like increased performance and robustness; but these benefits come at a price with increased costs, required dedicated communication network and a coordination strategy. In addition to these new modules, using multiple robots also requires changes in existing modules like path planning, localization, and mapping. In this section, some of these additional changes will be discussed.

2.3.1 Communication Network

Using multiple robots in a project requires a communication network over which the robots can communicate and follow a coordination strategy. The choice of the network depends on the developer's choice, but the most common ones are 802.11g Wi-Fi network, Bluetooth, and infrared systems. Out of these, the infrared systems works only on line of sight (LOS) paths; while other two works even outside LOS paths but in a specified range.

2.3.2 Coordination Strategy

A coordination strategy is required because it provides a guidance to each and every robot for what actions it should take. Without a coordination strategy, it may happen that rather than working cooperatively with each other, robots may start competing with each other leading to sub-optimal resource utilization. There are two major coordination strategies that can be employed by multiple robots: centralized approach and distributed approach.

2.3.2.1 Centralized Coordination Approach

This coordination approach is a kind of master-slave configuration. One robot is termed as master and acts like the central controller. The master robot interacts with all of the robots and assigns them the navigational goals [15]. Master robot is responsible for making sure there are minimum overlaps and no collisions between robots. This type of systems are easier to implement and perform well for smaller systems. But as the system size increases, it becomes difficult to scale this approach due to increased load on the master system and induced latency.

Another major problem with this approach is a single point of failure. As the master robot is responsible for all the coordination and communication between all the robots; once the master robot fails, the whole system stops working.

2.3.2.2 Distributed Coordination Approach

In distributed coordination approach, every robot is self-sustained. Each robot performs its own calculations and decides its navigational goal independently [15]. In this type of systems, all the robots communicate with each other rather than just a master robot as is the case with a centralized approach. This system is a bit more complex than the centralized approach but it is more robust as the system will keep on working even if all but one robot fail. Another disadvantage of distributed systems is that they are less secure than centralized systems.

2.3.3 Path Planning

In the case of multiple robots, path planning becomes more complex as now the module has to plan the paths for multiple robots such that the robots should not collide with obstacles or other robots. As the number of robots increase in an environment, centralized approaches become impractical and a decentralized path planning approach is preferred [33].

2.4 Coverage Strategies

Even after performing localization and mapping, a key question that still remains in robot coverage is that where should the robot move next in order to perform coverage of environment in minimum time? This problem corresponds to the travelling salesman problem for known graph like environments and is an NP-hard problem [14]. In this section, some strategies which have been used to cover the environment effectively will be discussed.

2.4.1 Traditional Coverage Strategies

Traditional coverage strategies include randomized and heuristic approaches. These approaches had the advantage that they were easy to implement and had lower computational cost. But they do not provide any guarantee whether the coverage will be successful or not.

2.4.1.1 Randomized Approach

In the randomized approach, robots select any random points as their next goal. This algorithm is easy to implement and does not requires costly sensors, but also does not provides the guarantee that complete coverage will be successful [12]. Another major problem with this approach is that in the worst case the robot can keep selecting points that are in an already explored area or in obstacles. Due to these disadvantages, a smarter version of this approach has been created in which the robot will randomly pick a point from an unexplored area in each iteration.

Heuristics Approach

The randomized approach provides a simple and easy to implement a strategy for coverage. But, using randomized approach can lead to situations where the robot never finishes coverage or can take a long time; in such situations, heuristics comes to our rescue. The main idea behind heuristics is to use practical techniques, which often provide us a good solution, but cannot guarantee an optimal solution. In heuristics approach, robots follow simple rules of thumbs like follow the wall and repel from each other [12].

But, these approaches also do not provide us a guarantee that the coverage will be completed successfully.

2.4.2 Cellular Decomposition

In order to provide some form of guarantee, cellular decomposition was introduced. Cellular decomposition is the process of breaking a large area into smaller parts. Then, by taking into account how many smaller parts have been covered, we can get an estimate whether the complete coverage has been achieved or not.

There are three types of cellular decomposition:

- Approximate Cellular Decomposition
- Semi-Approximate Cellular Decomposition
- Exact Cellular Decomposition

2.4.2.1 Approximate Cellular Decomposition

Approximate Cellular Decomposition is a type of cellular decomposition in which the area is divided in such a way that all cells are of same size and shape whose union only approximates the total area of the region [12]. In this decomposition, typically the size of a cell is equal to the footprint of the robot and coverage is assumed to be completed, when a robot visits a cell [12]. Thus, the complete coverage is achieved when all the cells are visited once by the robot.

Occupancy Grids as displayed in Fig 2.6 is one of the most common forms of approximate cellular decomposition used in research community due to its ease of use. Here, the whole map is divided into cells of same size and shape.

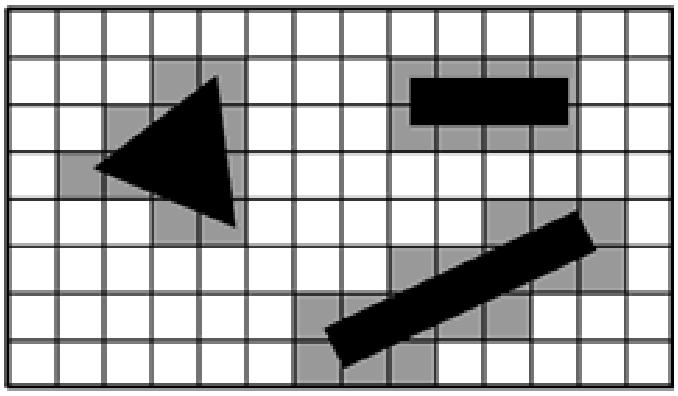


Fig 2.6 Occupancy Grid [35].

2.4.2.2 Semi-Approximate Cellular Decomposition

In semi-approximate cellular decomposition, the area is divided into smaller cells in such a way that the width of cells is fixed, but the length of each cell varies. This technique allows us to explore the map recursively. A robot using this technique can start at any arbitrary point and then completely explore the environment by zigzagging along parallel grid lines [12]. This technique is usually used for environments which are of irregular shapes. Fig 2.7 displays a typical environment with semi-approximate cellular decomposition.

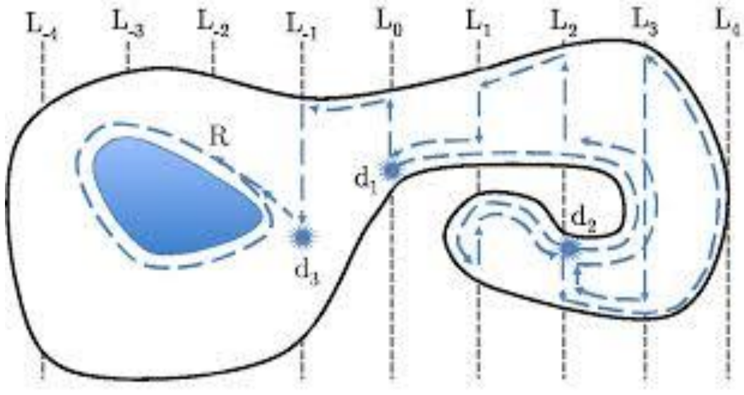


Figure 2.7 Path of a robot in an environment with semi-approximate cellular decomposition [48].

As we can see in the above figure, the environment is divided into cells equal width. The zig-zag line shows the path taken by a robot to cover the environment, while d_1 and d_2 represents the inlet points of the environment.

2.4.2.3 Exact Cellular Decomposition

Exact cellular decomposition is a type of cellular decomposition in which the given environment is divided into a set of non-intersecting cells, which can be covered by the robot using simple back and forth motions and whose union forms the entire environment [12]. One of the most widely used types of exact cellular decomposition is trapezoidal decomposition [12], in which the given environment is divided into cells shaped like trapezoids and triangles. In this scenario, the whole area can be easily covered using back and forth motions through each cell. Fig 2.8 shows an environment with start and goal points and non-intersecting cells. The robot can completely cover this environment by covering all of these non-intersecting cells with simple back and forth motions.

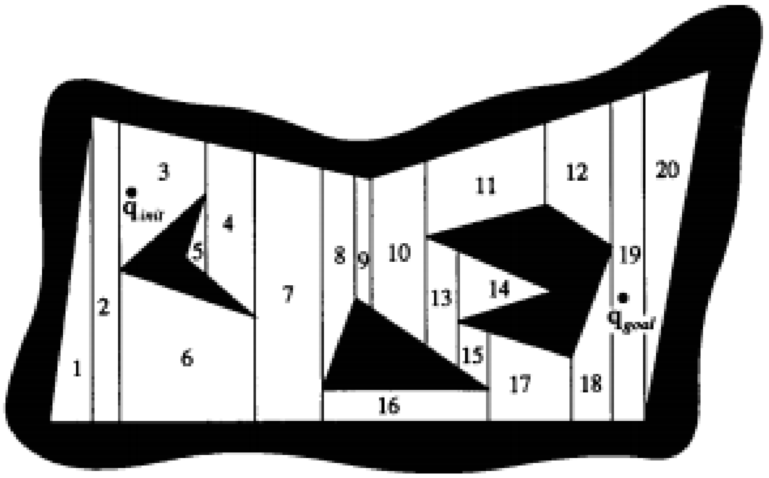


Figure 2.8 Trapezoidal Exact Cellular Decomposition [36].

2.4.3 Multi-Robot Coverage Strategies

The traditional coverage strategies discussed in previous sections were also applicable to multiple robot systems, but they do not utilize the full potential of all the robots. Some of the most popular coverage strategies discussed below [9], are designed for with multiple robots in mind and thus provide better performance than the traditional strategies.

2.4.3.1 Potential Fields

The Potential field's strategy of multi-robot coverage combines electrostatics, a fundamental concept of physics with robotic coverage. As we know from high school physics, like charges repel each other and opposite charges attract each other. This strategy uses this concept to create fields in the environment in such a way that robots and obstacles repel robots away; ultimately forcing the robots to spread out throughout the given environment [37]. The force with which robots' and obstacles repel the robots away is analogous to the inverse square law of electrostatic potentials.

The advantages of this approach are that there is no need for a centralized control and coordination strategy, localization or even inter-robot communication, leading to easy scaling to large environments [37]. But the major disadvantage of this approach is that it does not guarantee that complete coverage of the environment will be successful. It can happen that the robots reach a stage of equilibrium before the environment is fully covered. This issue can be solved with a slight variation in strategy where the particles in unexplored space attract the robots towards them. But in this case, robots can get trapped in local minima [15].

2.4.3.2 Graph Methods

As with most of the problems in computer science, graphs can be used to transform this problem from abstract form to graphs and then find new approaches to solving them. In our case, this environment can be transformed to a graph where the edges represent the paths and the nodes represent the intersection of these paths. Once the problem is transformed to a graph, any classic graph traversal problems like travelling salesman can be used to perform coverage [9].

The main advantage of this approach is that it allows us to compute the path of each robot before the program actually starts. But, this approach does not work as efficiently in cases where the environment in unknown or dynamic and SLAM is being used to update the map in each turn. In these cases, as the map changes, the path needs to be computed again. Moreover, in cases where one of the robots fails, or an unknown obstacle appears in the system, this approach does not behave in a robust way [9].

2.4.3.3 Frontier Based

In his landmark paper published in 1997 [38], Yamauchi introduced the concept of frontiers. Since then frontiers have become one of the most widely used coverage strategies. The map is represented by an occupancy grid where each cell can have one out of three values – free, occupied and unknown. Frontiers are the cells which lie at the boundary of unexplored and explored areas. When a robot moves to a frontier cell, it gains new information about the unexplored space [38]. Thus, the problem of coverage can be stated as the problem of selecting successive frontiers in such a way that the robot increases its knowledge about the environment at each step.

There are three main components of frontier based coverage:

- Frontier Detection
- Frontier Selection
- Frontier Navigation

Frontier Detection is the technique of detecting which cells are frontiers out of all the cells in an environment. To detect the boundaries between explored and unexplored space, techniques of edge detection and region extraction are used [38]. Once this boundary has been detected, free cells lying adjacent are marked as frontier cells. Fig 2.9 below shows the process of frontier detection.

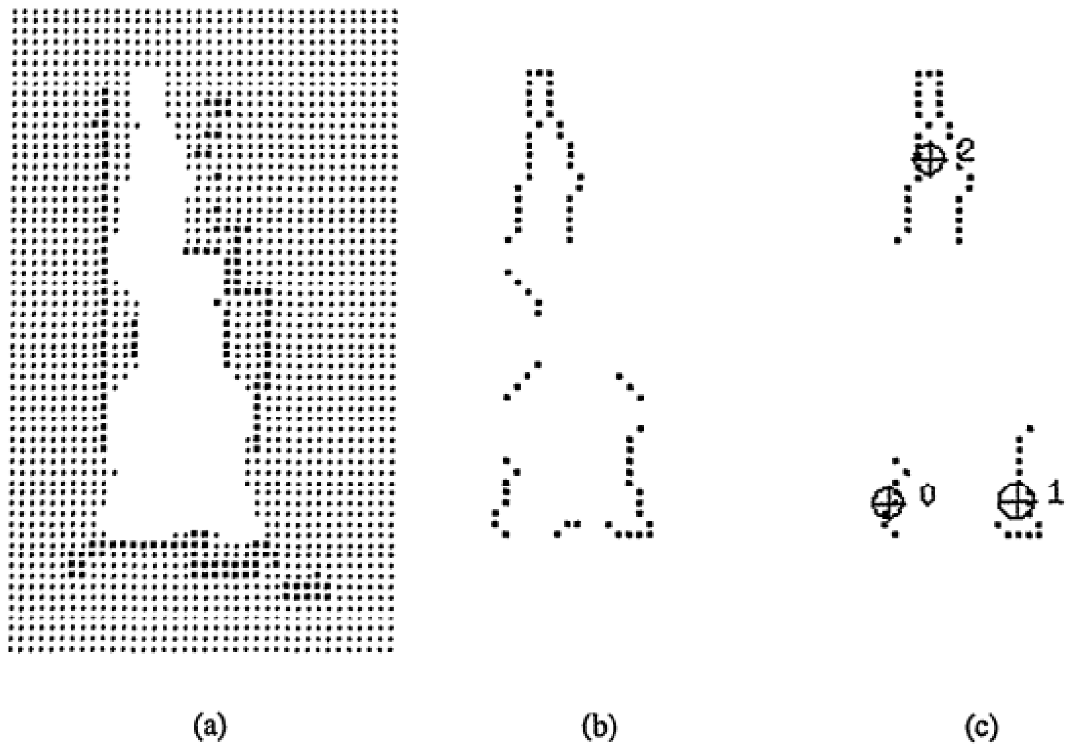


Figure 2.9 Frontier Detection Process (a) evidence grid (b) frontier edge segments (c) frontier regions [38].

Another common technique for frontier detection is Wavefront Frontier Detection (WFD) technique which uses a variant of breadth-first search algorithm to create a wave which starts from the current position of the robot and grows until it reaches the goal position [9]. The major disadvantage of WFD technique is that it can become costly as for each iteration of frontier detection, the full map has to be scanned.

In the case of multiple robots, each robot maintains its own version of the environment and perform frontier detection and frontier selection in that local version. After this process, this local map is merged with the global map and broadcasted to every other robot.

Once the frontiers have been detected, the key question arises which frontiers should be chosen in order to maximize the coverage in minimum time; this process is called Frontier Selection. There are multiple techniques like nearest based, greedy based, rank based, threshold based which are developed to solve this problem. These techniques will be discussed in detail in Chapter 4.

The last component of frontier-based exploration is Frontier Navigation which deals with how to navigate the robot to the chosen frontier. This part is taken care by the path planning module.

2.5 Summary

In this chapter, various techniques and processes required to perform multi-robot coverage in an unknown environment were discussed. Each technique has its own advantages/disadvantages and the decision to choose a strategy depends upon the nature of the environment, application and robot design. The user should keep these in mind while choosing the strategy for their system.

Chapter 3

ROBOT OPERATING SYSTEM (ROS)

As robotics is becoming more important in our lives, the scale and size of robots are increasing rapidly. Robot hardware and designs change according to the tasks for which they are employed. This causes a major problem for code reusability. ROS is an open-source framework consisting of tools, libraries, and conventions which simplify the creation of complex robotic applications [19], by allowing developers to reuse the existing implementations of algorithms such as map merging, vision, and navigation. ROS can be used with both physical and simulated robots. Stag is another open source 2D robot simulation software. Stage provides a virtual environment where we can create various robots with a variety of sensors to test our algorithms [39]. This chapter provides a brief overview about ROS, Stage, and their architecture.

3.1 Overview

ROS was developed in 2007 at Stanford Artificial Intelligence Laboratory (SAIL). It started as a service for inter-module messaging, but later after continuous upgrades became a framework. ROS is upgraded through distributions which are a set of ROS packages similar to Linux distributions. In this document wherever we are using ROS, we are referring to ROS Indigo released in 2014.

ROS acts as a meta-operating system and provides users with a range of services such as inter-process communication, hardware abstraction, multi-lingual development, rapid testing, and distributed computing [40]. ROS provides novice users with the capability to use the existing libraries and develop robotic systems at a much faster rate, without worrying about various hardware. ROS divides the major functionalities of a system into a distributed system of modules and then uses messaging to pass information between them.

Other major alternatives to ROS are Player, Microsoft Robotics Developer Studio (MRDS), Orocos, Open RTM and YARP. But out of all these ROS has achieved major support and is by far the most popular robot development platform as of now with the biggest developer community.

3.2 Architecture

At architectural level, ROS can be divided into three major categories [40]:

- Filesystem Level
- Computation Graph Level
- Community Level

3.2.1 Filesystem Level

The Filesystem level provides the details about the internal structure of the software and the core functionalities without which it cannot work. Some major functionalities of filesystem level are packages, stacks, services and messaging. Fig 3.1 displays the components of the Filesystem level, their hierarchy and how they interact with each other.

3.2.1.1 Packages

Packages are the most basic unit of ROS, which provides functionalities in easy to use modules. A package is a module with some functionality in it; it may contain runtime processes, library or configuration files. Packages are the smallest individual thing we can build in ROS and are channel for software release [41]. ROS provides the command line tool “roscreate-pkg” to create a new package.

3.2.1.2 Stacks

Stacks are combinations of several packages which together provide a functionality. For example, navigation stack takes information from robots' sensors, processes them and then sends commands to robot's actuators to perform navigational tasks. Navigation stack is composed of packages like amcl, costmap_2d, move_base and many more. The main advantage of using stacks is that they make code sharing much easier. ROS even provides "roscreate-stack" which is a command line tool to create stacks manually.

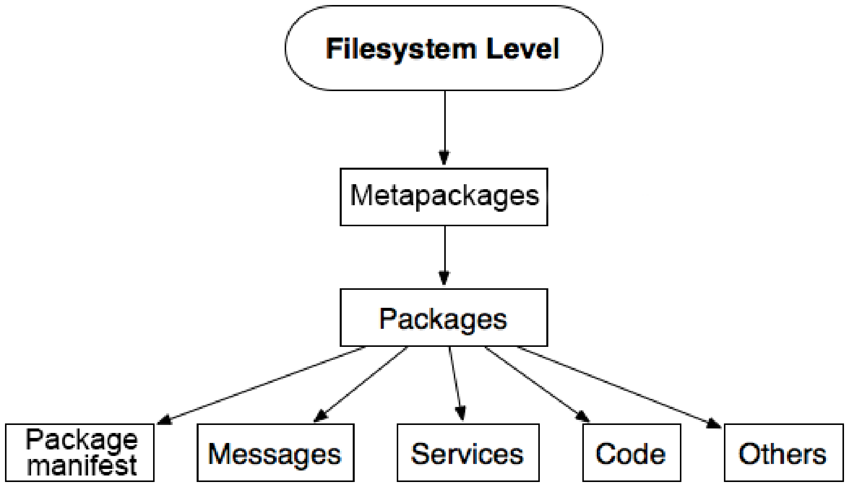
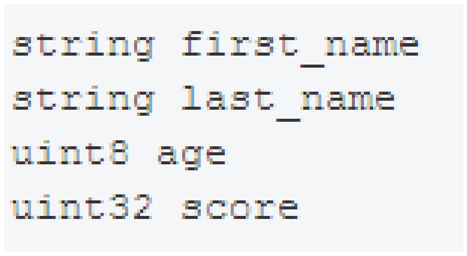


Figure 3.1 Architecture of ROS Filesystem Level [40].

3.2.1.3 Messages

Messages are tools in ROS to define data values which are exchanged between various processes. A Message in ROS is composed of two fields – fields and constants. Fields define the data type and constants define the field name. ROS allows many standard types like int, bool, float, time and string. In addition to these, there is a special type called Header. Header allows us adding frameID's and timestamp. ROS provides a command tool "rosmsg" which can print out information about message definition and files using a specific message type.

Fig 3.2 provides an example of a message in ROS, where the first field is of type String with field name first_name.



```
string first_name
string last_name
uint8 age
uint32 score
```

Figure 3.2 StudentGrades.msg [49].

3.2.1.4 Services

Messages provide many-to-many communication in ROS, where processes publish specific messages and other processes can access them on specific channels. But this kind of communication is not efficient in cases, where we want reply interactions, which are often required in distributed systems [42]. Services are based on a client/server model and allow client nodes to request information from other nodes, which then provide a response back to these nodes. ROS provides us two major command line tools to work with services:

- rossrv – provides information about services and the source files using a specific service
- rosservice – can list and query specific ROS services

3.2.2 Computation Graph Level

Computation Graph Level creates a network where all processes connect with each other; which is accessible to all the processes in the system [40]. This allows processes to communicate and exchange information with each other. The major functionalities provided by ROS at this level are nodes, master, messages, services, and topics. Fig 3.3 displays the components of computation graph level and how they interact with each other.

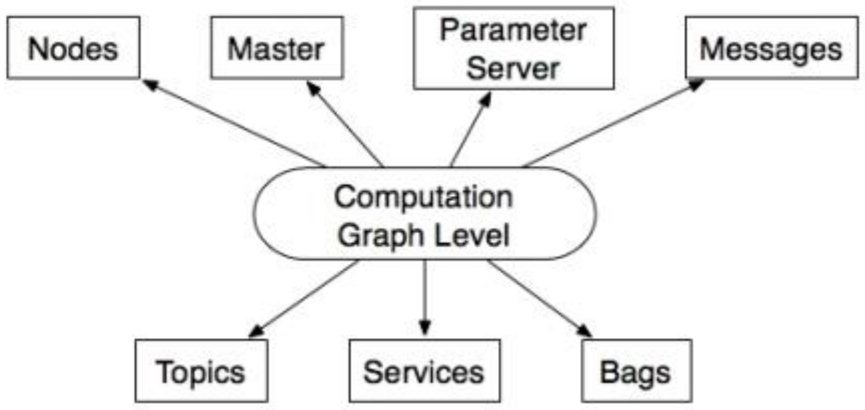


Figure 3.3 Architecture of ROS Computation Graph Level [40].

3.2.2.1 Nodes

In ROS, a node is a process where computation is performed [50]. Nodes are designed in such a way that each node is responsible for a small task. A simple robotic system usually contains a number of nodes. For example, in a simple system for robot navigation, one node will be responsible for laser sensor, one for robot localization, one for robot motion and so on. The usage of multiple nodes increases the robustness of the system and reduces code complexity in comparison to a system where a single node performs all the functions. ROS nodes are written in client libraries like roscpp (for C++) and rospy (for Python). ROS provides us with various command tools to interact with nodes such as rosnode info node, rosnode kill node, and rosnode list.

3.2.2.2 Master

ROS Master is the central core module of a ROS system which provides naming and registration services to the rest of the nodes in ROS system [51]. ROS Master allows nodes to find other nodes and then communicate with them. The master is run using the roscore command tool provided by ROS. ROS master is the first node which has to be executed when an ROS system is brought up.

3.2.2.3 Topics

Topics are the buses on which ROS nodes transmit data. Topics allow decoupling of data production and consumption as data can be transmitted between nodes, even if there is no connection between them [40]. Each topic has a data type and only that specific type of data can be transmitted on that specific topic. Any node which wants to listen to a specific type of data can subscribe to its equivalent topic. ROS allows both TCP and UDP based transmission on topics. It also provides us command tools such as rostopic echo /topic, rostopic list and rostopic info /topic to work with topics.

3.2.2.4 Parameter Server

A Parameter Server is a multivariable dictionary that is accessible to all the nodes where they can store and retrieve parameters during runtime [40]. Parameters server is a component of ROS Master and is widely used for configuration parameters so that system configuration can be viewed by any node.

3.2.3 Community Level

Community Level is the last level of ROS resources that allow various communities of people to exchange information, resources and functionalities [40]. These include resources like Distributions and repositories. Distributions are collections of stacks. A new distribution for ROS is released every year in May. The latest distribution of ROS is Kinetic Kame.

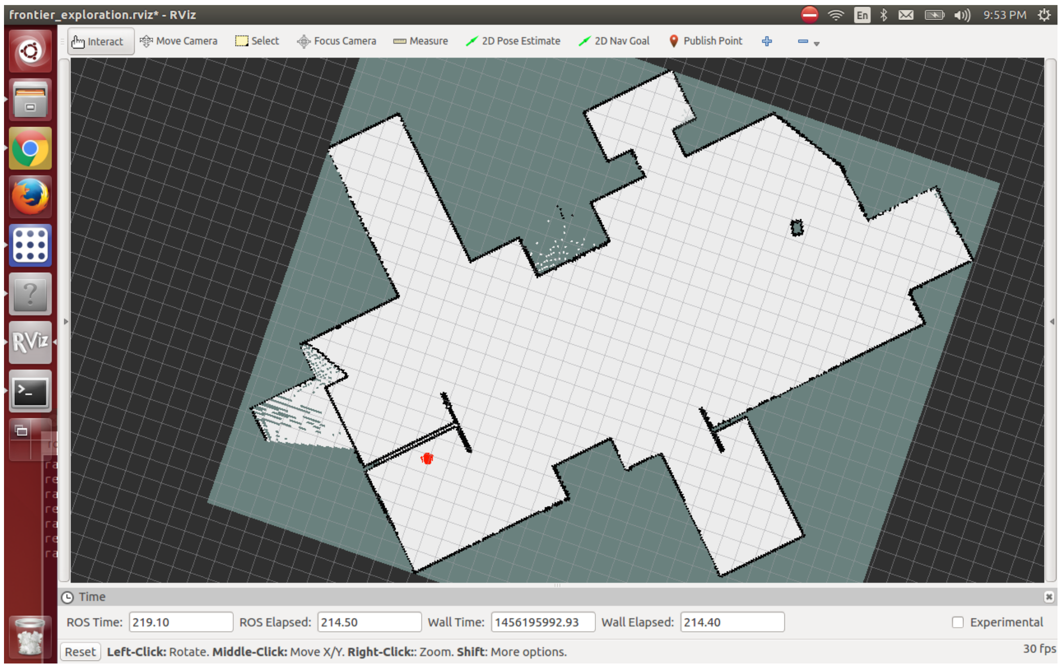
ROS repositories are a network of online code repositories which allow different communities to develop and publish their software packages.

3.3 ROS Development Tools

ROS provides a number of command tools that help developers in debugging their applications and visualize critical information about the system. In this section, we will look at the three major tools: rviz, rqt_console, and rqt_graph.

3.3.1 rviz

rviz is a visualization tool that integrates an OpenGL interface and represents the data collected from the sensors in a modeled environment. ROS has a lot of options for the kind of data to be displayed such as Grid, Laser-Scan, Maps, Markers and many more. The user can choose the display type and then perform various configurations on it such as color, size, decay time etc. Fig 3.4 displays rviz tool configured for multi-robot coverage. Here, the red dot represents the robot and the white area represents the area explored by the robot.



ROS Time: 219.10 ROS Elapsed: 214.50 Wall Time: 1456195992.93 Wall Elapsed: 214.40 Experimental

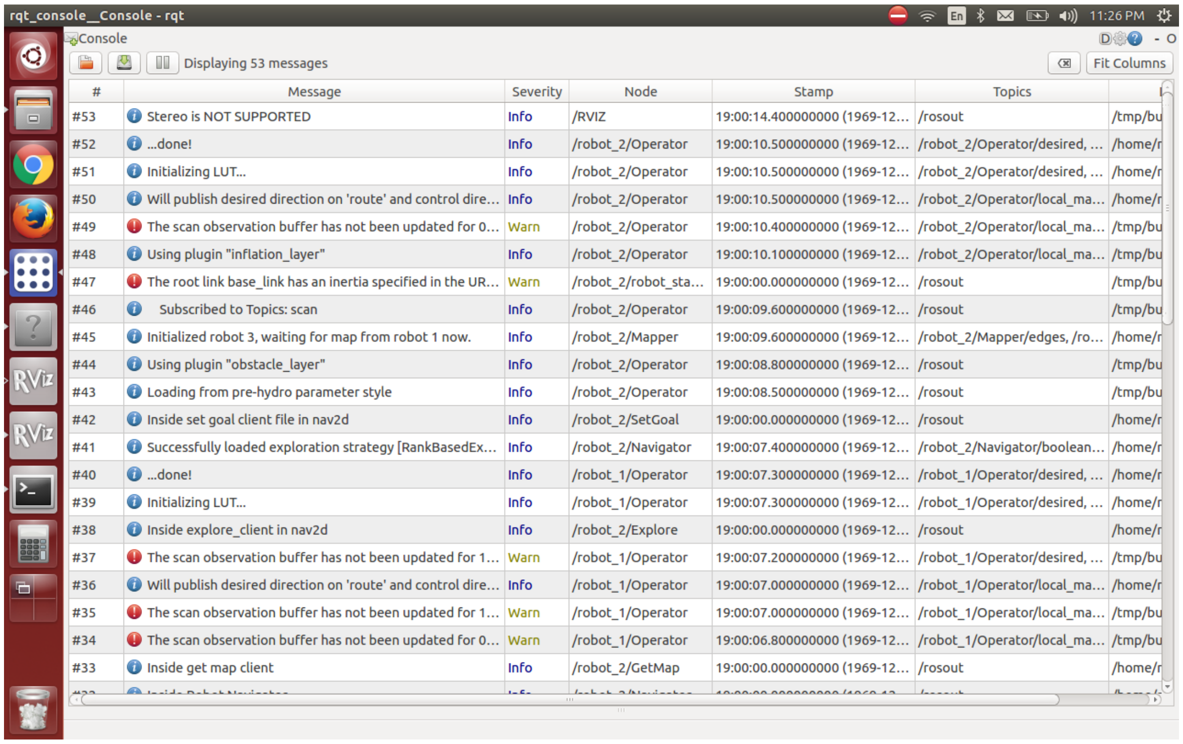
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options.

30 fps

Figure 3.4 Environment visualization through rviz.

3.3.2 rqt_console

'rqt_console' is a message viewer in ROS that allows users to see the messages published to the topic rosout in real time. ROS allows the user to filter out messages, so if the user only wants to see the most critical error, he/she can apply a filter on message severity to 'Error'. Fig 3.5 depicts a sample 'rqt_console' depicting various messages generated during operation. In addition, to the messages, the tool also displays the name of the node which published the message, timestamp and the topic where the message was published.



rqt_console__Console - rqt

Console

Displaying 53 messages

#	Message	Severity	Node	Stamp	Topics	
#53	Stereo is NOT SUPPORTED	Info	/RVIZ	19:00:14.400000000 (1969-12...	/rosout	/tmp/bu
#52	...done!	Info	/robot_2/Operator	19:00:10.500000000 (1969-12...	/robot_2/Operator/desired, ...	/home/r
#51	Initializing LUT...	Info	/robot_2/Operator	19:00:10.500000000 (1969-12...	/robot_2/Operator/desired, ...	/home/r
#50	Will publish desired direction on 'route' and control dire...	Info	/robot_2/Operator	19:00:10.500000000 (1969-12...	/robot_2/Operator/local_ma...	/home/r
#49	The scan observation buffer has not been updated for 0...	Warn	/robot_2/Operator	19:00:10.400000000 (1969-12...	/robot_2/Operator/local_ma...	/tmp/bu
#48	Using plugin "inflation_layer"	Info	/robot_2/Operator	19:00:10.100000000 (1969-12...	/robot_2/Operator/local_ma...	/tmp/bu
#47	The root link base_link has an inertia specified in the UR...	Warn	/robot_2/robot_sta...	19:00:00.000000000 (1969-12...	/rosout	/tmp/bu
#46	Subscribed to Topics: scan	Info	/robot_2/Operator	19:00:09.600000000 (1969-12...	/rosout	/tmp/bu
#45	Initialized robot 3, waiting for map from robot 1 now.	Info	/robot_2/Mapper	19:00:09.600000000 (1969-12...	/robot_2/Mapper/edges, /ro...	/home/r
#44	Using plugin "obstacle_layer"	Info	/robot_2/Operator	19:00:08.800000000 (1969-12...	/rosout	/tmp/bu
#43	Loading from pre-hydro parameter style	Info	/robot_2/Operator	19:00:08.500000000 (1969-12...	/rosout	/tmp/bu
#42	Inside set goal client file in nav2d	Info	/robot_2/SetGoal	19:00:00.000000000 (1969-12...	/rosout	/home/r
#41	Successfully loaded exploration strategy [RankBasedEx...	Info	/robot_2/Navigator	19:00:07.400000000 (1969-12...	/robot_2/Navigator/boolean...	/home/r
#40	...done!	Info	/robot_1/Operator	19:00:07.300000000 (1969-12...	/robot_1/Operator/desired, ...	/home/r
#39	Initializing LUT...	Info	/robot_1/Operator	19:00:07.300000000 (1969-12...	/robot_1/Operator/desired, ...	/home/r
#38	Inside explore_client in nav2d	Info	/robot_2/Explore	19:00:00.000000000 (1969-12...	/rosout	/home/r
#37	The scan observation buffer has not been updated for 1...	Warn	/robot_1/Operator	19:00:07.200000000 (1969-12...	/robot_1/Operator/desired, ...	/tmp/bu
#36	Will publish desired direction on 'route' and control dire...	Info	/robot_1/Operator	19:00:07.000000000 (1969-12...	/robot_1/Operator/local_ma...	/home/r
#35	The scan observation buffer has not been updated for 1...	Warn	/robot_1/Operator	19:00:07.000000000 (1969-12...	/robot_1/Operator/local_ma...	/tmp/bu
#34	The scan observation buffer has not been updated for 0...	Warn	/robot_1/Operator	19:00:06.800000000 (1969-12...	/robot_1/Operator/local_ma...	/tmp/bu
#33	Inside get map client	Info	/robot_2/GetMap	19:00:00.000000000 (1969-12...	/rosout	/home/r

Figure 3.5 rqt_console displaying messages.

3.3.3 rqt_graph

rqt_graph provides a GUI plugin in order to visualize the ROS computation graph [52]. This tool allows us to create a graph with all the running nodes and the publisher-subscriber connections between them [52]. Fig 3.6 displays the nodes in the system and how they communicate with each other.

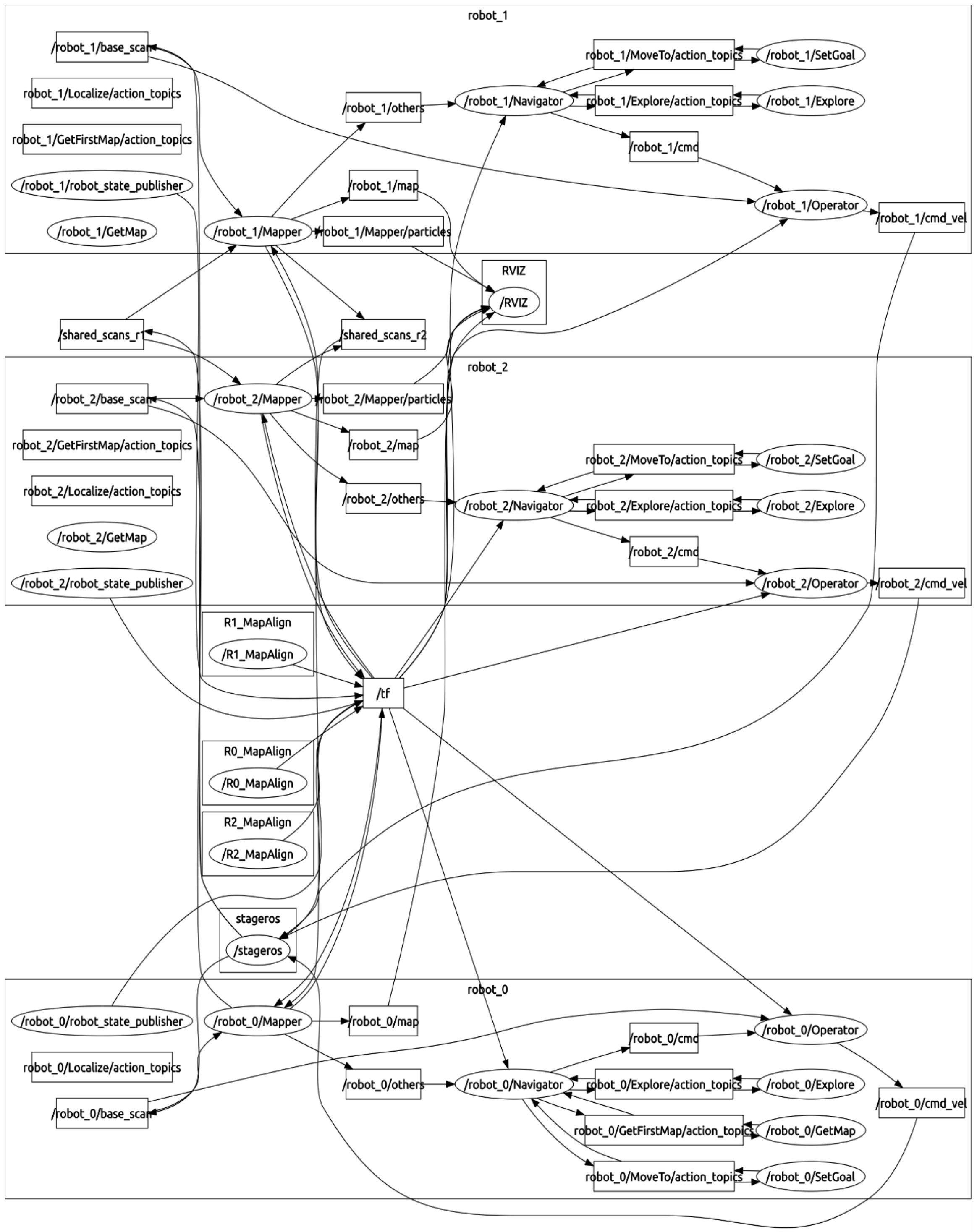


Figure 3.6 ROS computation graph.

3.4 Important ROS Stacks

As we discussed earlier, stacks are a collection of packages which together provide a major functionality. Each stack has an associated version and can have dependencies on other stacks. In this section, we will go through an overview of some of the major stacks used in our work.

3.4.1 Navigation Stack

The navigation stack of ROS takes input from the odometry, sensor streams, and a goal pose and provides velocity commands to the mobile base of the robot as output [53]. The velocity commands make sure that the navigation is collision free.

Navigation stack has some specific requirements that a robot must satisfy in order to use navigation stack [40]:

- The robot should have differential drive and holonomic wheels
- The robot's shape should be either square or rectangle
- The robot should publish all the information about the relationship between all the joints and sensors' position
- The robot should send messages with both linear and angular velocities
- The robot should possess a planar laser in order to proceed with map and localization

Fig 3.7 depicts the typical organization of the navigation stack. This diagram is depicting three types of nodes. The white nodes are the packages/stacks which are provided by default by ROS. While gray nodes are the ones that are not necessary, but provide some extra functionality. The nodes with dotted paths are the one which needs to be developed based on the platform on which we are using navigation stack.

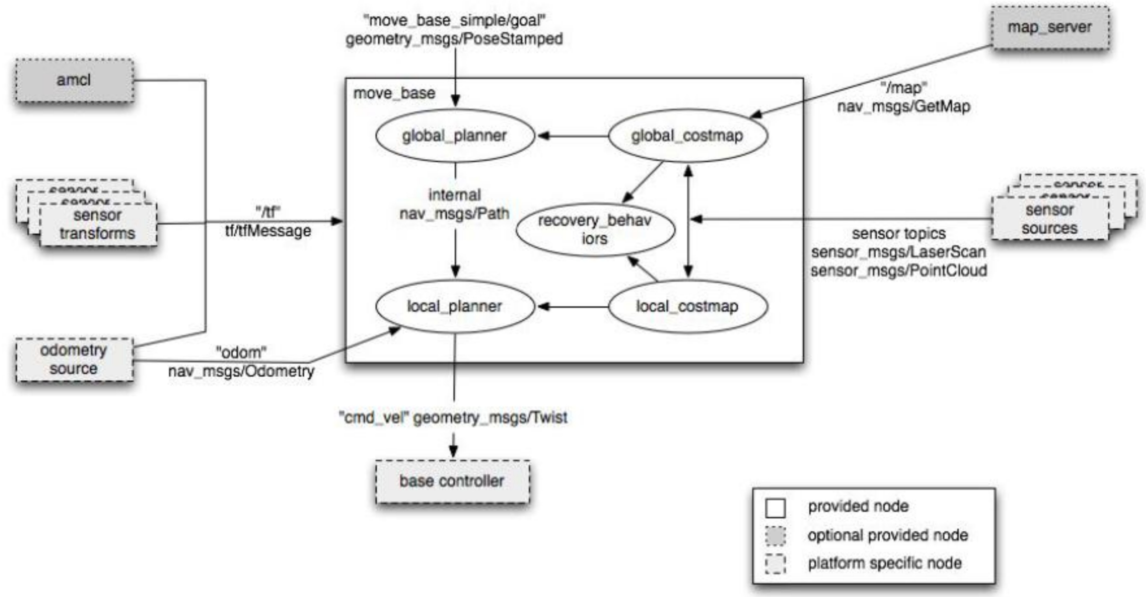


Figure 3.7 Organization of Navigation Stacks [40].

In order to work successfully, the navigation stack requires the map of the environment. In order generate a map, we need to perform SLAM. We can use the package gmapping for this. gmapping creates a node 'slam_gmapping' which takes input from robots' laser sensor and creates a 2-d occupancy grid map from laser data and pose information.

Other important packages in navigation stack are:

- **amcl** – amcl is a probabilistic localization system which performs localization for a 2D robot using Monte-Carlo localization approach [54]
- **costmap_2d** – this package creates a 2D costmap which takes sensor data as input as produces an occupancy grid map with each cell having values free, occupied or unknown.

- **robot_pose_ekf** – this package is used to estimate the 3D pose of a robot using extended kalman filters while taking measurements from multiple sources.

3.4.2 Actionlib Stack

Actionlib stack provides a standardized interface which allows us to interface with preemptive tasks [40]. Here, preemptive tasks are tasks which may take a long time to execute and the user may want in such cases to get a feedback about the status of the task or cancel the tasks in between. Some examples of such tasks are using robot's sensor to get environment data, object detection, robot motion from a source to destination. Actionlib works on a client/server model. Fig 3.8 displays the client-server interaction in the actionlib stack. The ROS is responsible for communication between client and server application.

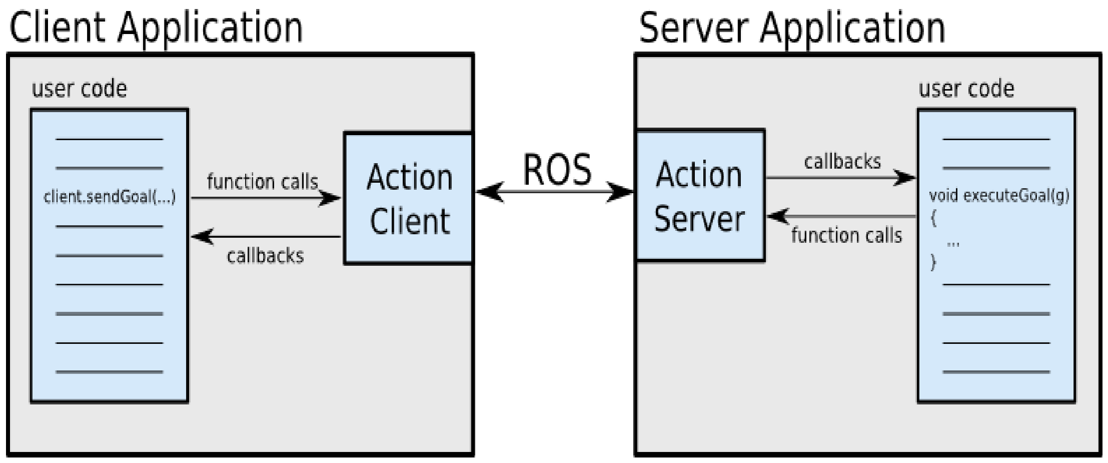


Figure 3.8 Client Server Model in actionlib Stack [61].

3.5 Stage

Stage is an open source simulation software which simulates a set of robots with their sensors in a 2D environment. Stage interfaces with the robots and receives sensor data from the ROS and then moves the robots in the simulated environment in accordance with real robots' behavior. Fig 3.9 shows Stage software being used for multi-robot coverage simulation. The two red dots here represents the robots in the system.

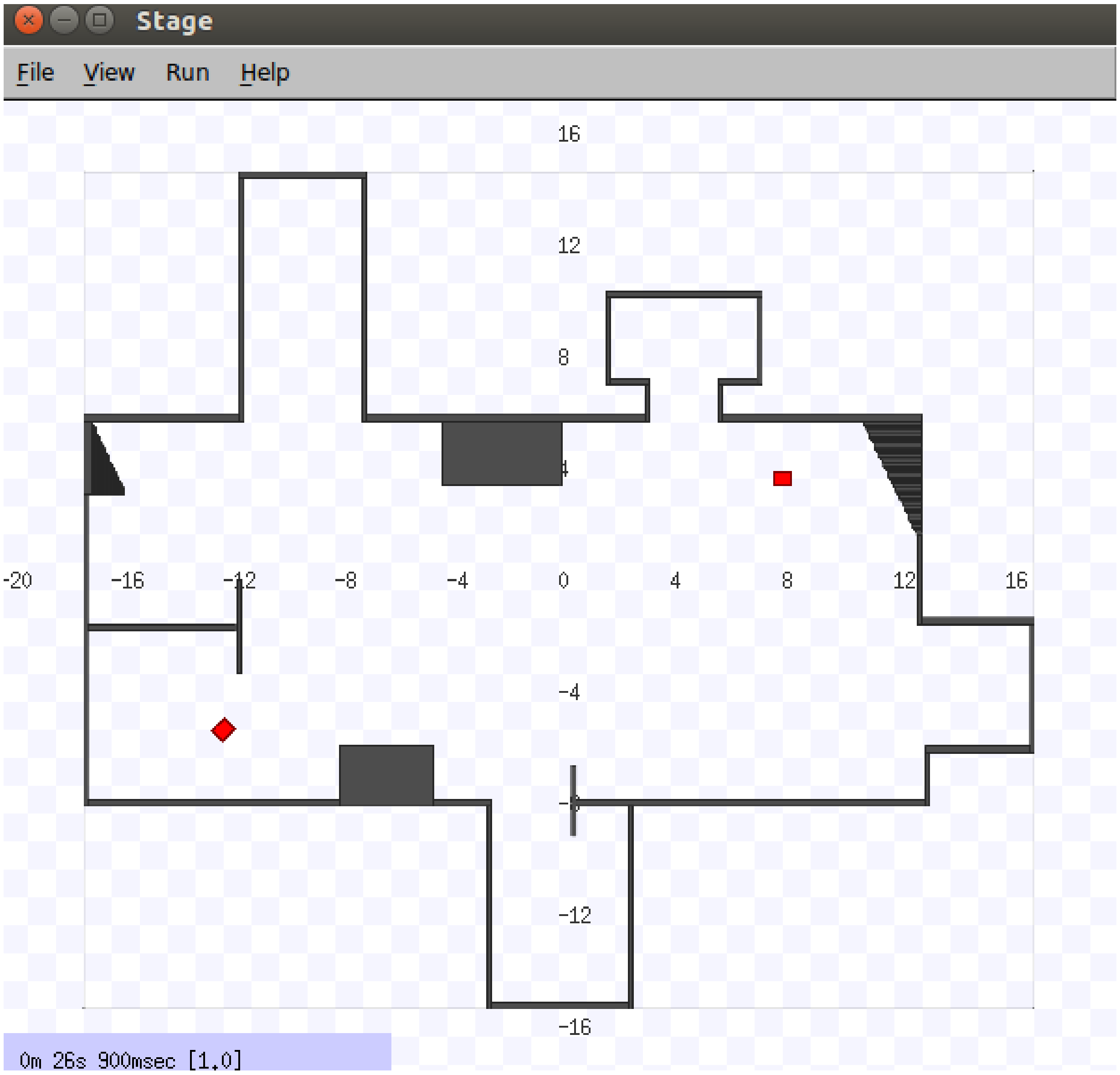
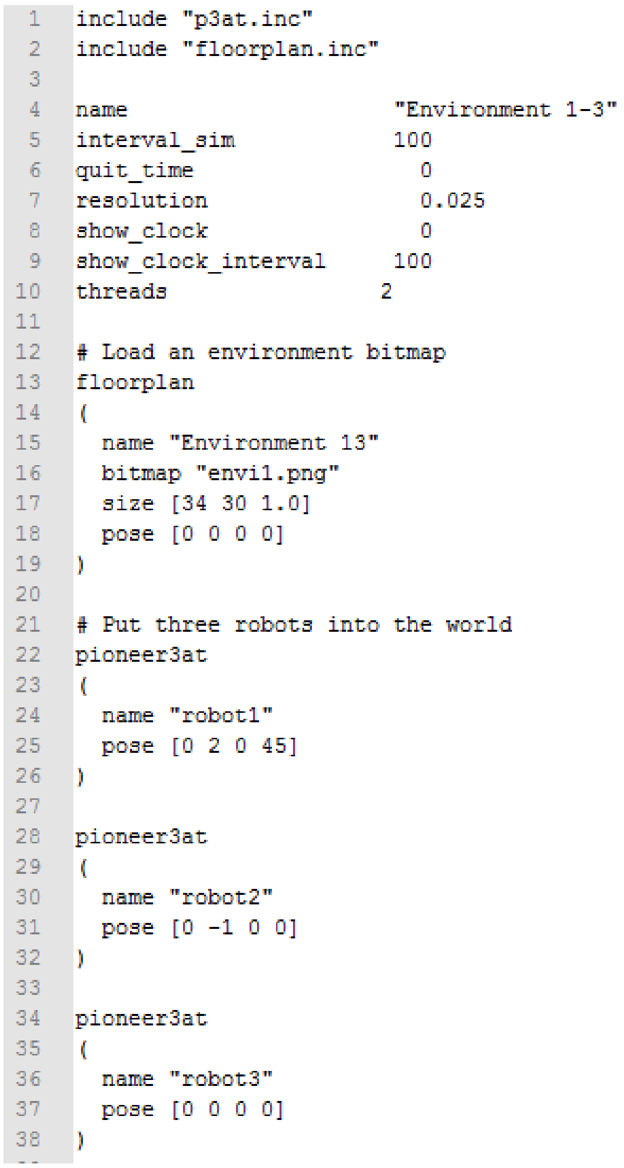


Figure 3.9 Multi-robot simulation in Stage.

Stage creates the simulated environment as per the configurations in a ".world" file. In this .world file, we provide all the details about the environment such as resolution, dimensions, number of robots and many more. Fig 3.10 shows a sample .world file with various parameters such as starting location of all the robots in the environment, the number of robots etc.



```
include "p3at.inc"
include "floorplan.inc"

name                    "Environment 1-3"
interval_sim            100
quit_time                 0
resolution                0.025
show_clock                0
show_clock_interval     100
threads                2

# Load an environment bitmap
floorplan
(
  name "Environment 13"
  bitmap "envi1.png"
  size [34 30 1.0]
  pose [0 0 0 0]
)

# Put three robots into the world
pioneer3at
(
  name "robot1"
  pose [0 2 0 45]
)

pioneer3at
(
  name "robot2"
  pose [0 -1 0 0]
)

pioneer3at
(
  name "robot3"
  pose [0 0 0 0]
)
```

Figure 3.10 Sample world file

Chapter 4

PROPOSED REMEMBER-ALL FRONTIER BASED MULTI-ROBOT COVERAGE APPROACH

In Chapter 2, the problem of multi-robot coverage and various approaches that can be employed to solve that problem were discussed. Out of all the coverage approaches discussed so far, the frontier based approach is one of the most popular in the research community. This chapter will focus on the existing frontier based multi-robot coverage approaches and the proposed Remember-All multi-robot coverage approach in detail.

4.1 Background

One of the major problems in multi-robot coverage is to decide where the robots should move next, in order to complete the coverage of the environment in minimum time. The frontier based coverage approach attempts to provide a solution to this problem. In the frontier based approach, the map is decomposed into cells of same size and shape using an occupancy grid. Each cell in this grid is then provided a state value – free, occupied, and unknown. The cells which lie at the boundary of explored and unexplored areas are known as frontier cells. Once the robots have identified frontier cells in the grid, frontier allocation strategy is used to allocate one frontier to each robot. The robots then move towards their respective frontiers. This process is repeated again until no new frontiers can be found.

In this chapter a new Remember-All frontier coverage approach, which is an extension of the Rank Based frontier coverage approach is proposed in order to improve the performance of robot coverage in terms of runtime and overlap. This proposed coverage approach has the following major features:

- Multi-Robot coverage of unknown environments
- A new frontier allocation strategy – "Remember-All" frontier allocation strategy
- Robots can start coverage from random points within environment
- New communication policy
- Improved coordination strategy

4.2 Assumptions

The proposed approach for multi-robot coverage is easily scalable to multiple robots and could be deployed on various types of environments irrespective of their shape and size. But, in order to narrow the scope of the implemented system, few assumptions have been made. First, it is assumed that a communication network for inter-robot communication is available over the entire environment. In any case, if a robot loses its connection with the network and is not able to communicate with the other robots, it is assumed that the robot has failed.

Second, robots are equipped with a fixed laser sensor, which does not move during the process, leading to easier mapping and localization. Third, it is assumed that the environments are static and there will be no change in them while the robots are deliberating. This assumption constraints our scope to non-dynamic environments only. Fourth, in this work, experiments are conducted using two and three robot systems. But, the proposed approach can be easily extended to systems with more than three robots.

The above assumptions are similar to the assumptions made in previous robotics literature [9, 15] and do not diminish the challenges of multi-robot coverage in unknown environments in comparison to previous works.

4.3 Proposed Coverage Approach

The process of multi-robot coverage starts with multiple robots placed at random locations in an unknown environment. Unlike some other approaches [59, 60], where the starting locations of the robot remain same for every case, the proposed coverage approach is able to work in situations where the robots start at random locations on the map.

The proposed multi-robot coverage approach can be divided into six major steps to be completed by each robot:

1) Localize the robot and map the environment using SLAM
2) Create a global map through merging individual robot maps
3) Update the occupancy grid map using individual sensor measurements
4) Identify frontiers
5) Allocate frontier to the robot
6) Navigate the robot towards the selected frontier

Fig 4.1 displays the steps of the proposed Remember-All approach for multi-robot coverage in unknown environments. The steps 1 to 6 are executed continuously and concurrently for each robot in a serial manner till the robot is unable to find any new frontiers. Each of these modules will be discussed in detail in the following sections. The above-mentioned steps are similar to the robot coverage approaches followed in [9, 13, 15]. But in the proposed approach there is a major change in step 5, which demonstrates a new frontier allocation strategy.

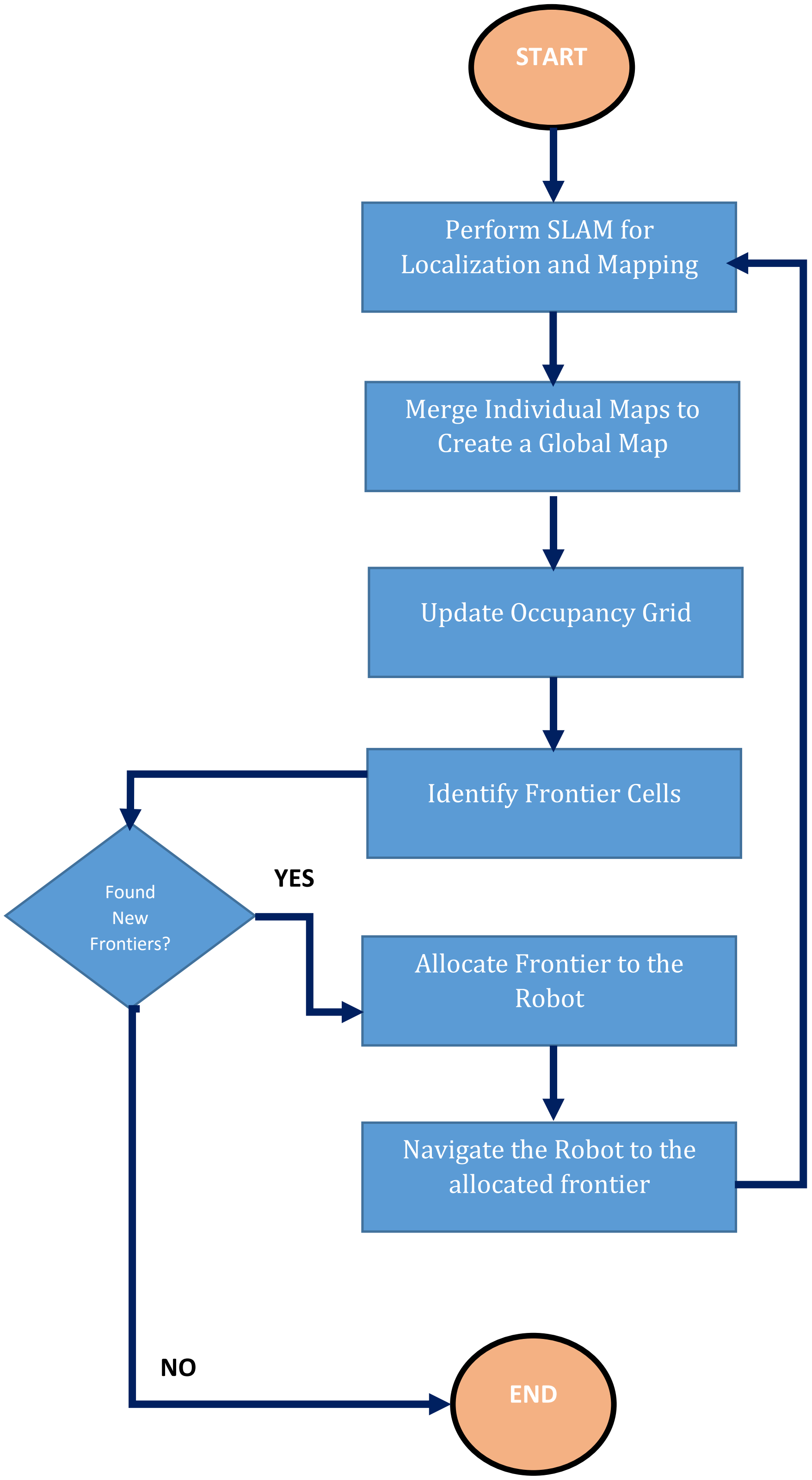

START
Perform SLAM for Localization and Mapping
Merge Individual Maps to Create a Global Map
Update Occupancy Grid
Identify Frontier Cells
Found New Frontiers?
YES
Allocate Frontier to the Robot
Navigate the Robot to the allocated frontier
NO
END

Fig 4.1 Steps for Multi-Robot Coverage in Unknown Environments using the proposed approach

4.3.1 Localization and Mapping

The 'Localization and Mapping' module in step 1, uses the ROS package 'gmapping' to localize the robot and create a map of environment simultaneously using SLAM. This package creates a node 'slam_gmapping' which takes sensor data from robot's laser sensor and pose data from the 'amcl' package [53]. It then uses this set of data to create a 2-D occupancy grid map of the explored environment. 'amcl' is an ROS package which implements a particle filter-based localization technique.

In this technique, the whole map of the environment is filled with a large number of particles (in our experiments, we used 10,000 particles). Each of these particles is then simulated to behave as an actual robot. The particles are given the same 'move' commands which are provided to the robot and then their laser measurements are compared with the actual robot's measurements. The particles with similar measurements as the robot are retained while others are replaced with new particles. These steps are repeated over and over until we have a concrete estimate of the location of the robot in the environment.

4.3.2 Map Merging

The second step in the proposed approach is to take individual maps of each robot and merge them together to create a global map of the explored environment. The map merging module uses an ROS package 'map_merger' which keeps track of the changes to the local maps of each robot and then automatically distributes them to the other robots [54]. Once the maps are distributed to the other robots, the robots then merge their own maps with the maps of other robots to create a global map of the explored environment.

In order to merge maps from the multiple robots, the package performs coordinate transformations between the multiple local coordinate systems of each robot and the global coordinate system using the ROS package 'tf'. The 'map_merger' package solves this problem by identifying the overlapped regions in the maps, which then provides it with the transformation points [54]. Once the coordinate transformations are completed, the algorithm performs sanity checks to remove any merging errors. It is observed that the environments having repetitive structures are usually more prone to merge errors.

It may also happen in few cases that, no transformation can be performed due to lack of sufficient overlapping regions. In such cases, the 'map_merger' package stores the map and checks it in the next iteration. If the transformation is possible in the next iteration the map is merged with other maps otherwise, it is stored for another iteration. The map merging algorithm is scalable to multiple robots, with no theoretical limit to the number of robots; but it is observed that the quality of global map generally decreases with the increase in the number of robots being used [54]. In the author's view, this proposed approach will work with a practical number of robots. But in cases where a swarm of robots is being used, a new map-merging algorithm is required to maintain the desired quality of the global map.

4.3.3 Update Occupancy Grid

Once a map of the environment has been created through map merging of individual maps, the occupancy grid is updated to associate occupancy values (free, occupied and unknown) with the cells in step 3. The module uses ROS package 'costmap_2d' which takes sensor readings from the robot sensors and then performs coordinate transformations to convert them as per the global map coordinates [55]. In order to perform the transformations from local coordinate frames to the global coordinate frame ROS package 'tf' is used. After this, the sensor readings are used to associate values with all the cells.

4.3.4 Identify Frontiers

Once we have updated the cell values in occupancy grid based on the sensor measurements, the next step (step 4) is to identify the frontier cells. Frontier cells are the cells which lie at the boundary of unknown and known areas. In our approach, we have used the wavefront propagation technique from the ROS package 'nav2d_exploration' to identify frontiers, which is quite popular in the robotics community [56]. In this technique, we create a wave which traverses all the cells from the robot's current location, until it hits the unexplored area.

The underlying algorithm for this technique uses breadth-first search to propagate the wave forward. In order to improve the performance of the overall approach, the algorithm creates clusters from the adjacent frontiers. The 'nav2d_exploration' package creates a 'nav2d' node which gets the information about the map by subscribing to the topic /map and then publishes the cells identified as frontiers over the topic /frontiers [57].

4.3.5 Allocate Frontiers

A number of strategies exist for allocating frontiers to the robots. In this work, we propose a new frontier allocation strategy, which is an extension of Rank based frontier allocation strategy [9]. The proposed allocation strategy is compared with some of the existing frontier allocation strategies: Nearest [38], Greedy [14], Rank [9] and Threshold [15]. This section will provide an overview of the existing and the proposed allocation strategies. All of these strategies have underlying algorithms which are used for their implementation. In this work, the words 'strategy' and 'algorithm' are used interchangeably whenever no confusion arise.

4.3.5.1 Nearest Based Frontier Allocation

In this strategy, the algorithm first calculates the Euclidean distance between each of the frontiers identified in the frontier identification step and the corresponding robot. The robot is then allocated the nearest frontier to that robot. This strategy has the advantage that it is simple and is easy to implement. But, one of the major disadvantages of this technique is that one frontier may get allocated to multiple robots, leading to increase in robot overlap and under-utilization of resources, inducing a lower runtime performance.

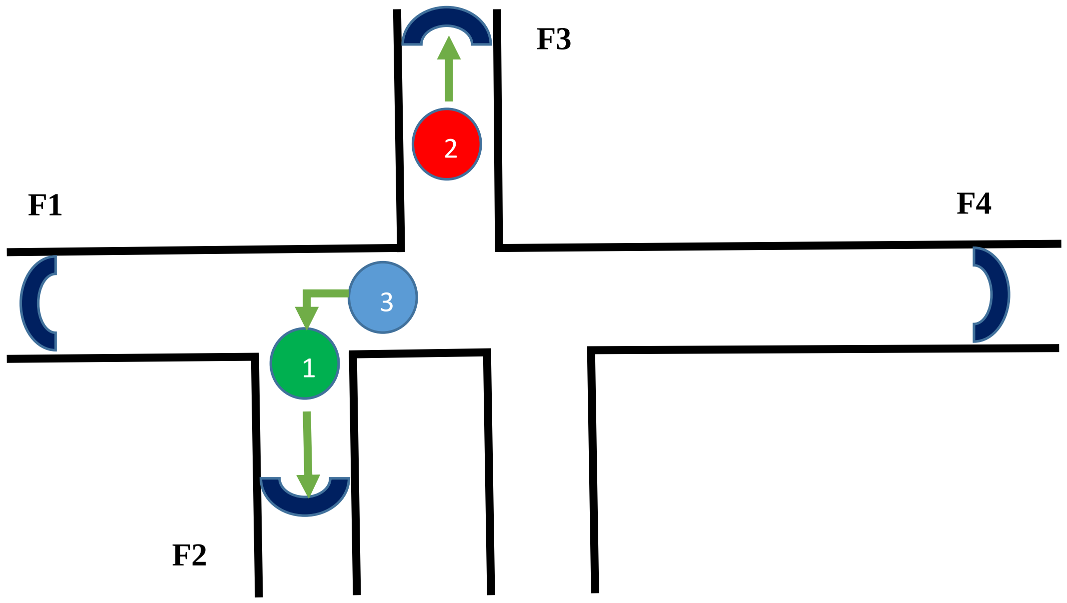


Fig 4.2 Nearest Based frontier allocation.

Example 1: Consider Fig 4.2, which displays an example of the Nearest Based frontier allocation strategy. The figure shows a sample environment with four frontiers (F1, F2, F3, and F4) and three robots (1, 2 and 3). The algorithm first calculates the Euclidean distance between the robot 1 and the frontiers identified by the robot 1 in the current iteration. In this example, let us suppose robot 1 identifies two frontiers – F1 and F2.

The algorithm then calculates the Euclidean distance of the frontiers to robot 1 and allocates the frontier F2 to robot 1 as it is closest to the robot. Similarly, the robot 2 is allocated the frontier F3. In the case of robot 3, the algorithms calculate the distance between the robot and the frontiers F1, F2, F3, and F4.

After this, the algorithm allocates the frontier F2 to robot 3, even though it was previously allocated to robot 1 because it is the closest frontier to the robot. ▲

4.3.5.2 Greedy Based Frontier Allocation

The Greedy Based frontier allocation strategy is similar to Nearest Based frontier allocation strategy, as it also calculates the Euclidean distance between the frontiers and uses it as a deciding factor for allocating frontiers. But rather than allocating the nearest frontier to the robot, as was the case in Nearest Based Allocation, the Greedy Based frontier allocation allocates the nearest unassigned frontier to the robot.

This makes sure that a single frontier is not allocated to multiple robots and thus provides better performance than the Nearest Based frontier allocation strategy. But the major disadvantage of this strategy is that it may lead to situations where robots do not spread evenly in an environment and keep on getting allocated with the frontiers of the same region. This leads to reduced runtime performance of the overall coverage strategy.

Example 2: Consider Fig 4.3, which displays an example of the Greedy Based frontier allocation. The figure shows a sample environment with four frontiers (F1, F2, F3, and F4) and three robots (1, 2 and 3). The algorithm first calculates the Euclidean distance between the robot 1 and frontiers F1 and F2. It then allocates the frontier F2 to the robot 1, as it is the nearest frontier to the robot. Similarly, the robot 2 is allocated the frontier F3. In the case of robot 3, the algorithms calculate the distance between the robot and the frontiers F1, F2, F3, and F4. The nearest frontier to the robot 3, in this case, is frontier F2, but it is already allocated to robot 1. Therefore, the next nearest frontier F1 is allocated to the robot 3. ▲

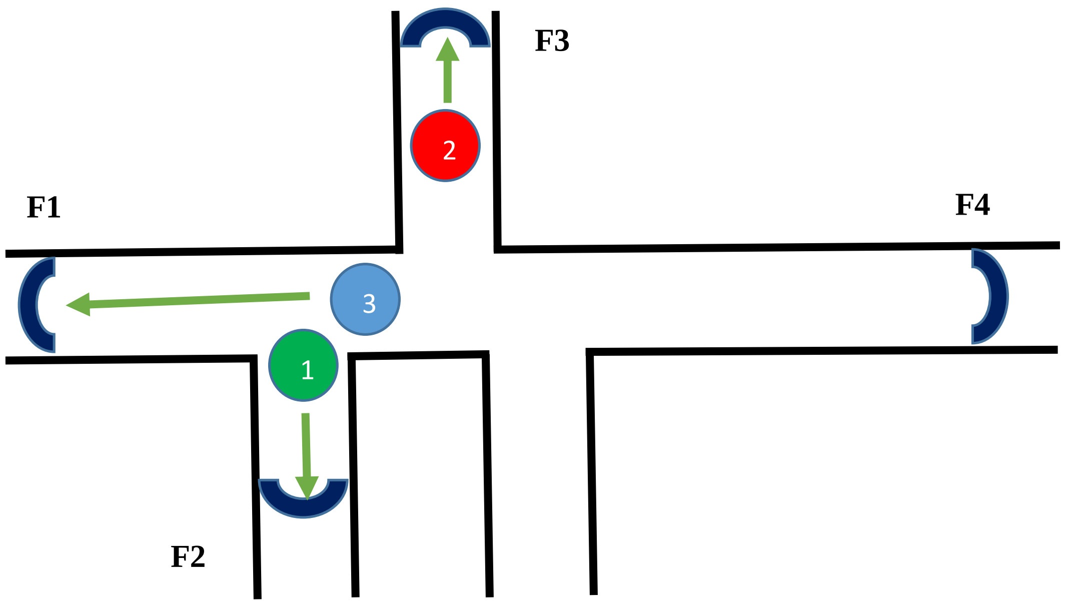


Fig 4.3 Greedy Based Frontier Allocation.

4.3.5.3 Rank Based Frontier Allocation

The rank based frontier allocation strategy, first proposed in [9], focuses mainly on the idea that better performance in coverage could be achieved if each robot is allocated to a frontier with fewer robots in its direction [13]. This strategy allocates frontiers based on two main criteria – distance to the robots and number of robots near a frontier. Each frontier is then assigned a rank by all the robots based on the number of robots near to that frontier; higher the number of robots near the frontier, the higher its rank.

Each frontier is then allocated to the robot with the lowest rank for that frontier. In an ideal scenario, there would be only one robot which is best suited for a specific frontier. But, in reality, it may happen that there is a tie between multiple robots for a frontier. In that case, the tie is resolved by taking into account the distance between the robot and the frontier.

The Rank Based algorithm first creates a cost matrix C, where each element C_{ij} represents the distance between the robot R_i and the frontier F_j . Then, a position matrix P is created, where each element P_{ij} represents the rank of the robot R_i for the frontier F_j and is calculated through the following equation [9]:

$$\sum_{\forall R_k \in R,\ k \neq i,\ C_{kj} < C_{ij}} 1 \tag{4.1}$$

Example 3: Consider Fig 4.4, which displays an example of the Rank Based frontier allocation strategy. The figure shows a sample environment with four frontiers (F1, F2, F3, and F4) and three robots (1, 2 and 3). The Rank Based algorithm first calculates the Euclidean distance between each robot and its frontiers to create a cost matrix C.

Table 4.1 represents the cost matrix for this example. In table 4.1 below, it can be seen that the Euclidean distance between the frontier F1 and robot 1 is 20. A value of 0 in the cost matrix denotes that the particular frontier is not identified by the robot; as is the case for the frontier F1 and robot 2.

Table 4.2 represents the position matrix P for this example, which represents the rank of the frontiers for each robot. This matrix is created using the Cost Matrix C in Table 4.1 and the equation 4.1. As it can be observed from the Table 4.2, the frontier F3 has rank 1(lowest rank) for robot 2 as in the cost matrix it is the only frontier identified by the robot. But, in the case of robot 1, both frontiers F1 and F2 have rank 1. This is due to the fact that the robot 1 has identified both frontiers F1 and F2, and both of these frontiers have two robots near them. As there is a tie, the frontier with the least distance to the robot will be allocated (in this case F2).

Table 4.1: Cost Matrix

Frontiers	Robot 1	Robot 2	Robot 3
F1	20	0	25
F2	10	0	15
F3	0	5	20
F4	0	0	45

Table 4.2 Position Matrix

Frontiers	Robot 1	Robot 2	Robot 3
F1	1	0	2
F2	1	0	2
F3	0	1	2
F4	0	0	1

Similarly, in the case of robot 3, the frontier F4 is allocated to the robot 3 as it has the lowest rank for that robot. ▲

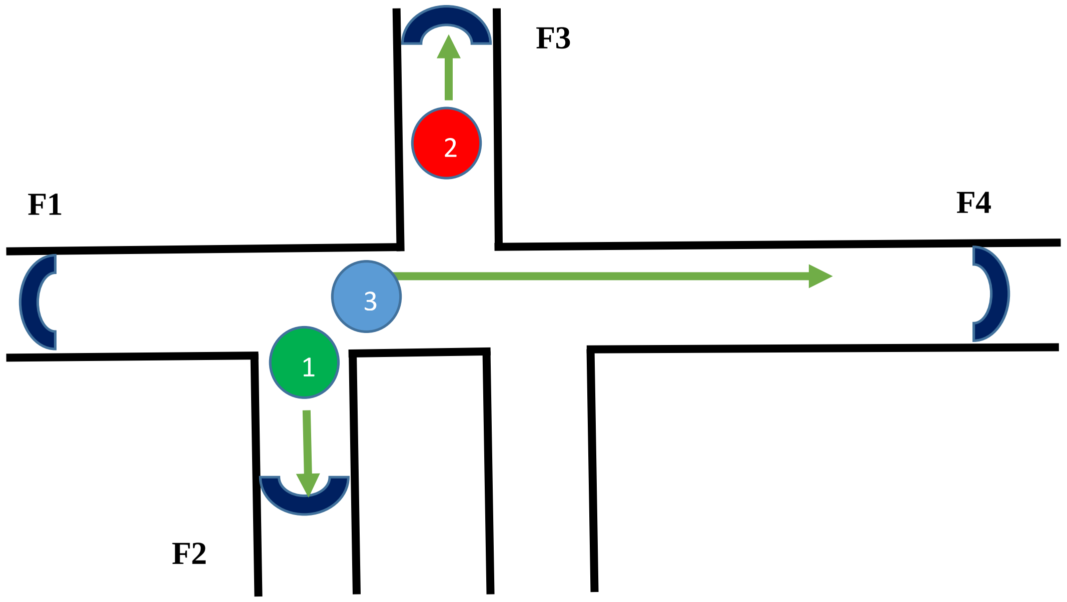


Fig 4.4 Rank Based Frontier Allocation.

4.3.5.4 Threshold Based Frontier Allocation

Threshold Based frontier allocation strategy is an extension of the Rank Based Strategy [15]. In this strategy, the frontiers are allocated using Rank Based approach, but unlike the Rank Based strategy, all of the frontiers which lie within a pre-defined threshold limit of a robot are marked for allocation. All of these marked frontiers are then allocated to the robot in a serial manner of their ranks – from lowest to the highest. Once, all the marked frontiers are allocated to the robot, the algorithm moves on to the next step.

Example 4: Consider Fig 4.5, which displays an example of the Threshold Based frontier allocation. The figure shows a sample environment with four frontiers (F1, F2, F3, and F4) and three robots (1, 2 and 3). The dashed circle around the robot 1 represents its threshold radius (assumed for the simplicity of the example); all the frontiers in this circle (F1 and F2) would be marked and explored by robot 1 (in order of their ranks) first, before moving on new frontiers. In this case, the robot 1 first goes to frontier F1 and then navigates back to frontier F2. ▲

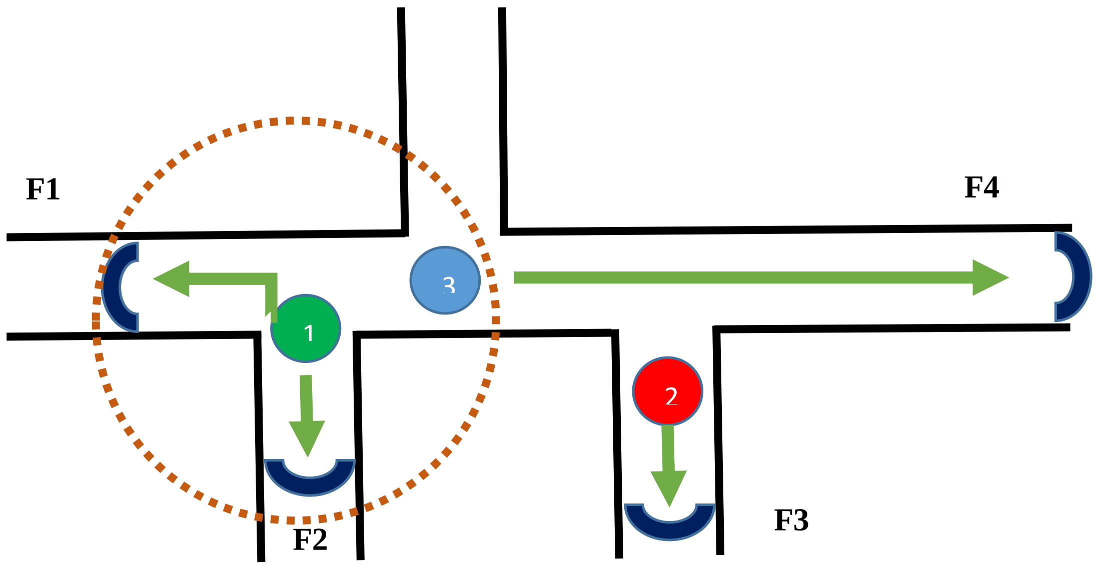


Fig 4.5 Threshold Based Frontier Allocation.

One of the major advantages of this approach is that it keeps track of the frontiers within the threshold range of a robot and thus makes sure no identified frontiers are neglected by the robot. But, this approach often leads to large amounts of overlap in situations where a large number of frontiers are present in the threshold range of a robot, leading to increased run time.

4.3.6 Navigate Robots

Once the robot has been assigned a frontier, ROS navigation stack is used to plan a path from the robot's current position to the destination. The navigation stack takes as input sensor data and the goal position; it then calculates the optimum path for the robot [41]. This information is then sent to ROS 'move_base' package which interacts with the robot's mobile base to actually move the robot.

4.4 Proposed "Remember-All" Frontier Allocation Strategy

All of the frontier allocation strategies discussed so far only store information of the allocated frontiers [15]. Each robot is allocated one frontier based on the frontier allocation strategy being used and the rest of the frontiers are discarded. Moreover, these unallocated frontiers may very well be rediscovered in future iterations. This action of discarding unallocated frontiers can lead to loss of valuable information about the prospective frontiers for future frontier allocation iterations.

The threshold based frontier allocation strategy tried to tackle this problem by marking all the frontiers in a predefined threshold range. But, it forces the robots to cover all of these marked frontiers first before moving on to new frontiers in the next iteration [15]. This compulsion to cover all the marked robots first often leads to a large amount of overlap as the robot moves through the same area repeatedly while covering these marked frontiers.

The proposed Remember-All frontier allocation strategy aims to maintain all the information about the unassigned frontiers and use this information in the future frontier allocation iterations. This measure will reduce the robot overlap during coverage, thereby leading to improved performance of the overall strategy, as to be demonstrated in the next Chapter.

Another major advantage of the proposed Remember-All frontier allocation strategy is that, as the robots are storing the information of unallocated frontiers, it ensures that the robots do not miss any unallocated frontiers that were identified in previous iterations.

In the proposed strategy, each robot maintains three individual lists over the course of the program:

- 'identified_frontiers'
- 'explored_frontiers'
- 'unallocated_frontiers'

The 'identified_frontiers' list is reset at the beginning of each iteration while the other two lists retain information till the end of the program. At the beginning of the program, all three lists are empty.

The proposed Remember-All frontier allocation strategy can be summarized through the following set of steps:

1) Before the process of frontier allocation takes place, each robot uses frontier identification technique to identify frontier cells in the explored map. The identified frontier cells are stored in the robot's individual 'identified_frontiers' list. The 'identified_frontiers' list is then compared with the robot's 'explored_frontiers' list to remove any identified frontiers which are already explored.

Consider Fig 4.6 which shows a sample environment with three robots – 1, 2 and 3 and six frontiers – F1 to F6. The gray areas represent the obstacles in the environment. Before the frontier allocation process begins, each robot identifies frontier cells and stores them in its individual 'identified_frontiers' list.

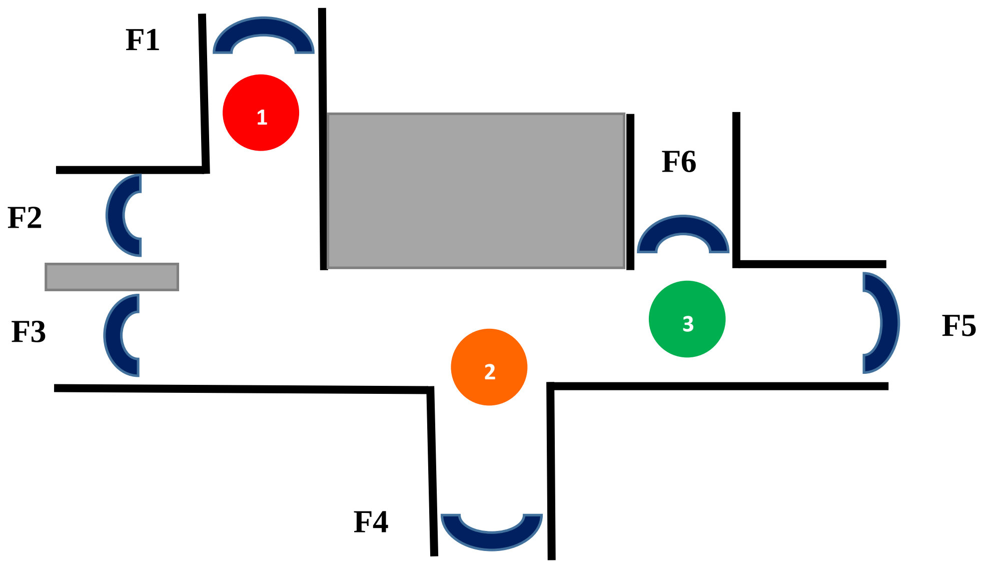


Fig 4.6 Sample environment for proposed Remember-All Based Frontier Allocation

Robot 1 ('identified_frontiers')	**F1**	**F2**	**F3**		
Robot 2 ('identified_frontiers')	**F2**	**F3**	**F4**	**F5**	**F6**
Robot 3 ('identified_frontiers')	**F5**	**F6**			

Fig 4.7 'identified_frontiers' list for all the three robots

Fig 4.7 displays 'identified_frontiers' lists for each of the three robots at this step of the process. Both of the other lists – 'unallocated_frontiers' and 'explored_frontiers' are empty for all the three robots at this moment, as the process of frontier allocation has not yet started. But in the future iterations, these two lists may have frontiers.

2) Frontiers are allocated to each robot using the Rank based frontier allocation strategy

This step of the proposed frontier allocation strategy ranks the frontiers, identified in the previous step, based on the Rank Based strategy. As the 'unallocated_frontiers' list is empty at this point, the frontiers used for ranking are only taken from the robot's 'identified_frontiers' list. But in the future iterations, the algorithm will use a number of frontiers (configurable) from the 'unallocated_frontiers' list, in addition to the 'identified_frontiers' list. This will be explained in detail in the below steps.

As it can be observed from Fig 4.6, if the robot 1 identifies three frontiers – $\mathrm{F1, F2, and\ F3}$ in the frontier identification step, the Rank Based strategy will rank all these frontiers. In this case, the frontier $\mathrm{F1}$ will be ranked lowest for robot 1 and thus, gets allocated to robot 1. Similarly, frontier $\mathrm{F4}$ will be allocated to robot 2 and frontier $\mathrm{F5}$ will be allocated to the robot 3. The frontiers allocated to the robots in this step are highlighted in Fig 4.7

3) All of the unallocated frontiers from the 'identified_frontiers' list of a robot are copied in the robot's 'unallocated_frontiers' list

Before each robot navigates to its allocated frontier, all the unallocated frontiers from the 'identified_frontiers' list of the robot are copied to the robot's 'unallocated_frontiers' list. Fig 4.8 displays the 'unallocated_frontiers' list for all the three robots at this step of the frontier allocation process.

Robot 1 (‘unallocated_frontiers’)	**F2**	**F3**		
Robot 2 (‘unallocated_frontiers’)	**F2**	**F3**	**F5**	**F6**
Robot 3 (‘unallocated_frontiers’)	**F6**			

Fig 4.8 ‘unallocated_frontiers’ list for all the three robots

As robot 1 was allocated frontier F1 in the first step, frontiers F2 and F3 are added to robot 1’s ‘unallocated_frontiers’ list. Similarly, robot 2 adds frontiers – F2, F3, F5, and F6; while robot 3 adds frontier F6 to its list.

4) Each robot then broadcasts the frontier allocated to it, to all other robots. All the robots then add this frontier to their ‘explored_frontiers’ list and remove it from their ‘unallocated_frontiers’ list, in case this frontier is already present in it.

In the case of the example discussed in Fig 4.6, robot 1 will broadcast to all other robots that frontier F1 is explored. All the other robots will then add frontier F1 to their ‘explored_frontiers’ list and will check if F1 exists in their ‘unallocated_frontiers’ list. If F1 does exists in any robots’ ‘unallocated_frontiers’ list, then F1 will be removed from the list. This is done to make sure that robots do not rank an explored frontier in the future frontier allocation iterations.

Similarly, when robot 3 is allocated frontier F5, it is broadcasted to all other robots that frontier F5 is already explored. It can be observed from the Fig 4.8 that robot 2’s ‘unallocated_frontiers’ list contains frontier F5. Therefore, robot 2 will remove F5 from its ‘unallocated_frontiers’ list.

Fig 4.9 represents the 'unallocated_frontiers' list maintained by each robot at the end of this step. As it can be observed from the below figure, at the end of this step, the 'unallocated_frontiers' list only contains the frontiers which were identified by the respective robot but were not allocated to any of the robots till this iteration.

Robot 1 ('unallocated_frontiers')	**F2**	**F3**	
Robot 2 ('unallocated_frontiers')	**F2**	**F3**	**F6**
Robot 3 ('unallocated_frontiers')	**F6**		

Fig 4.9 'unallocated_frontiers' list for all the three robots

Fig 4.10 represents the 'explored_frontiers' list maintained by each robot at the end of this step. As it can be observed from the below figure, the 'explored_frontiers' list of each robot will have the exactly same values.

Robot 1 ('explored_frontiers')	**F1**	**F4**	**F5**
Robot 2 ('explored_frontiers')	**F1**	**F4**	**F5**
Robot 3 ('explored_frontiers')	**F1**	**F4**	**F5**

Fig 4.10 'explored_frontiers' list for all the three robots

This measure makes sure that every robot has the information about which frontiers have been explored till any point of time and thus, allowing lower frontier duplication among the robots.

5) In the next iterations of frontier allocation using the 'Remember-All' frontier allocation strategy, once the frontier identification step has been completed, a number (configurable) of frontiers from the 'unallocated_frontiers' list of the robot will be added to the robot's 'identified_frontiers' list.

The usage of this combination of frontiers (consisting of the frontiers identified in the current frontier identification iteration and a fixed number of the frontiers from the list of unallocated frontiers maintained by the robot) allows the algorithm to allocate a frontier from a wider pool of prospective frontiers. In comparison to this, the Threshold Based frontier allocation strategy coerces the robot to cover all the marked robots first, even if they are not the best frontier at that step.

The number of frontiers added in the 'identified_frontiers' list in this step is configurable at the beginning of the program. The main reason for using only a fixed number of frontiers rather than all of the unallocated frontiers stored in the list is that, as the robot covers an environment, the list of unallocated frontiers grows rapidly. Using all these unallocated frontiers can cause a performance overhead to the algorithm. Therefore, in order to maintain the good performance of the algorithm only a limited number of unallocated frontiers are used in each frontier allocation iteration.

The frontiers to be added to the 'identified_frontiers' list of a robot are selected from the robot's 'unallocated_frontiers' list in LIFO (Last in First Out) order. Thus, only most recent frontiers are used first. Fig 4.11 displays the changes in the sample environment discussed in Fig 4.6 after the proposed frontier allocation strategy has reached the current step. As it can be observed from the below figure, all the three robots have navigated to the respective frontiers allocated to them.

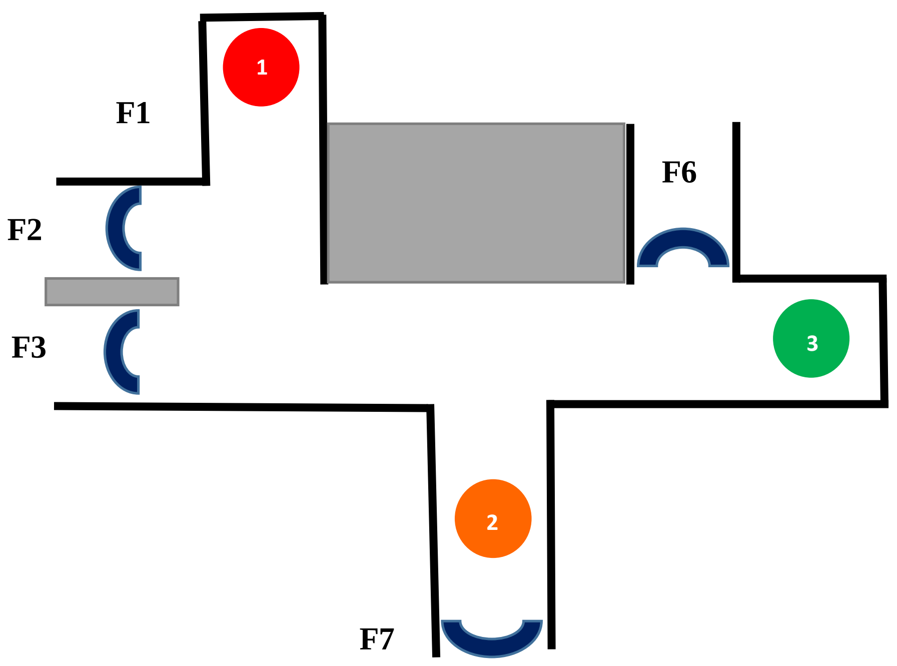


Fig 4.11 Sample environment for proposed Remember-All Based Frontier Allocation

For the example in Fig 4.11, let us suppose that the number of unallocated frontiers which can be added to the 'identified_frontiers' list is configured at two. In the next frontier allocation iteration, if there is a dead end after frontier F1, robot 1 will find no more new frontiers in step 0. The 'identified_frontiers' list, in this case, will consist of only the unallocated frontiers from the previous frontier allocation for robot 1 – F2 and F3.

In the case of robot 2, if it finds a new frontier F7, its 'identified_frontiers' list will consist of F7, F6, and F3. Similarly, in the case of robot 3, its 'identified_frontiers' list will consist of only frontier F6. Once the algorithm has added frontiers to the 'identified_frontiers' list of a robot from its 'unallocated_frontiers' list, it will check the 'explored_frontiers' list to remove any frontiers which are already explored. Fig 4.12

displays the 'identified_frontiers' list for all the three robots at this step of the proposed frontier allocation strategy.

Robot 1 ('identified_frontiers')	**F2**	**F3**	
Robot 2 ('identified_frontiers')	**F7**	**F6**	**F3**
Robot 3 ('identified_frontiers')	**F6**		

Fig 4.12 'identified _frontiers' list for all the three robots

6) The algorithm now uses the Rank Based frontier allocation strategy to allocate one frontier for each robot from the robot's individual 'identified_frontiers' list.

The above-mentioned steps are repeated over and over again until the robots are not able to find any more new frontiers in the frontier identification step and there are no more frontiers in the 'unallocated_frontiers' of the robots.

There is also a special case in the above strategy where, if there are no frontiers identified in step 0 and there are also no frontiers available in the robot's 'unallocated_frontiers' list. In this particular case, the robot will contact another robot which is nearest to its current position (let's say Robot X). Then, a number of recent frontiers (configurable) will be copied from the robot X's 'unallocated_frontiers' list to the robot's 'identified_frontiers' list.

4.5 Proposed Communication Policy

The proposed Remember-All frontier based multi-robot coverage approach is implemented as a distributed system, where all robots perform their tasks individually. All robots localize themselves and create a map of their environment independently. The robots then communicate and exchange information in step 2 as displayed in Fig 4.1, when they create a global map of the explored environment by combining the individual maps of the robots [54].

After each robot is allocated a frontier using the proposed frontier allocation strategy, each robot broadcasts the allocated frontier to all the other robots. The robots then add this broadcasted frontier to their individual lists of explored frontiers – 'explored_frontiers' and check if the broadcasted frontier exists in their individual lists of unallocated frontiers – 'unallocated_frontiers'. If the broadcasted frontier exists in the latter list, it is removed from the list.

This measure aims to avoid re-allocation of explored frontiers to the robots. This type of distributed communication model, where each robot maintains its own individual lists of explored and unallocated frontiers, leads to increased robustness. In this model, failure of one robot would not lead to failure of the overall system due to the presence of additional redundancies and will make sure that duplication of frontiers is minimum.

4.6 Proposed Coordination Strategy

The coordination strategy of a robotic system guides the robots to work with each other cooperatively. In the proposed 'Remember-All' frontier based coverage approach, each robot maintains its own list of unexplored frontiers, allowing other robots to use this information when they don't have any frontiers of their own. As the proposed frontier allocation strategy uses a combination of frontiers (consisting of frontiers identified in the current step and few unassigned frontiers from the robot's 'unallocated_frontiers' list), even if a robot is unable to identify any new frontiers in the current iteration, the 'identified_frontiers' list will contain few frontiers from the robot's 'unallocated_frontiers' list.

It may also happen that a situation arises, where a robot finds no new frontiers and the 'unallocated_frontiers' list of the robot is also empty. In this case, the robot will be allocated the nearest frontier from the n recent frontiers in the nearest robot's 'unallocated_frontiers' list; where n is the configurable limit of frontiers allowed to be added from 'unallocated_frontiers' list to the 'identified_frontiers' list.

4.6.1 Proposed Buddy Approach

In [44], the authors proposed a unique bidding-based approach for frontier allocation, where the robots bid for the frontiers based on the cost of navigating to a frontier, and the robot with the best bid is allocated that frontier. They also introduced a problem that in cases where the number of frontiers becomes fewer than the number of robots, the coverage can come to a halt as robots may start believing that coverage is already completed.

But, it may happen that this decrease in the number of frontiers is temporary, and once the robots move into the next region, the frontiers increase again. This scenario usually takes places in maps which contain narrow corridors linking rooms. If multiple robots enter a narrow corridor, only the robot at the front is able to identify new frontiers. All other robots believe that as there are no more frontiers, the coverage is completed and stop their operation. This scenario can cause a huge performance drop as few robots will stop their operation, leading to under-utilization of resources.

In this proposed approach, this problem is tackled through the Buddy Approach (name inspired from [62]). When the number of frontiers becomes fewer than the number of robots, the robots with no new frontiers will start following a robot with a frontier at a safe distance. This process will be repeated until any new frontiers are found or the leading robot has no more frontiers. If the leading robot finds new frontiers, they will be shared with the buddy robots; however, if no new frontiers are found by the leading robot, coverage will be stopped. The Buddy Approach is especially useful in the environments where narrow corridors connect various regions of the map.

4.7 Comparison of Frontier Allocation Strategies

In this section, various frontier allocation strategies are compared with the proposed Remember-All approach.

a) **Nearest Based Frontier Allocation Strategy**

Advantages:

- Easy to understand and easy to implement

Disadvantages:

- Multiple robots may get allocated to a single frontier, leading to under-utilization of resources

b) **Greedy Based Frontier Allocation Strategy**

Advantages:

- Easy to understand and easy to implement
- Improved performance in comparison to Nearest Based strategy

Disadvantages:

- Robots usually get confined to the same region

c) **Rank Based Frontier Allocation Strategy**

Advantages:

- Improved performance in comparison to the above approaches

- Robots are dispersed in a larger region as compared to previous approaches, leading to a better area coverage

Disadvantages:

- Works only in known environments
- No storage of unallocated frontiers often causes frontier duplication and increased robot overlap

d) Threshold Based Frontier Allocation Strategy

Advantages:

- Improved performance in comparison to the above approaches
- All the frontiers in the robot's threshold radius are marked and explored first, ensuring that the robots do not miss any unallocated frontiers that were identified in previous iterations.

Disadvantages:

- May lead to excessive overlap of the same area.
- Robot coverage may stop prematurely in cases where number of frontiers become fewer than the number of robots in the environment

e) Remember-All Based Frontier Allocation Strategy

Advantages:

- Improved performance in comparison to the above approaches
- Reduced frontier overlap
- Able to work in situations where number of frontiers are fewer than number of robots

- Storage of unallocated frontiers ensures that the robots do not miss any unallocated frontiers that were identified in previous iterations.

Disadvantages:

- Extra computation is required to avoid frontier duplication

4.7.1 Comparison with the Enhanced Frontier Based Approach

The enhanced frontier based approach in [43] also tried to solve the problem of robot coverage through the storage of unassigned frontiers. This approach also stored the unassigned frontiers for each robot individually and then, used this storage pool of frontiers to get frontiers in case no new frontiers are found. But, as it can be observed this approach has several shortcomings, which the proposed 'Remember-All' approach endeavors to resolve. Some of the major shortcomings of the Enhanced Frontier Based approach are as following [43]:

- The Enhanced Frontier Based approach uses the unallocated frontiers in only those cases where no more frontiers are found and thus does not take full advantage of the stored unassigned frontiers. In the proposed 'Remember-All' approach, a combination of identified and unallocated frontiers is used to take full advantage of the stored frontiers
- The Enhanced Frontier Based approach uses a combination of Closest Frontier Obstacle Distance (CFOD) [58] and Euclidean distance as a parameter to choose frontiers. This may lead to extra overhead for the algorithm and may not be efficient in environments with wide corridors. In the proposed approach, the Euclidean distance is used

- Usage of CFOD also confines the robots starting at a similar location to a similar region, leading to uneven diffusion of the robots in an environment. The proposed approach uses the number of robots near a frontier in addition to the Euclidean distance for frontier allocation. This makes sure that the robots are allocated frontiers with fewer robots in the same direction, leading to improved diffusion.
- This approach may provide reduced performance in obstacle-rich environments as the strategy needs to perform extra calculations for CFOD
- Does not performs well in cases where the number of frontiers in an environment may become fewer than the number of robots, as the robots may stop prematurely. The proposed approach uses Buddy Approach to tackle this scenario

Chapter 5

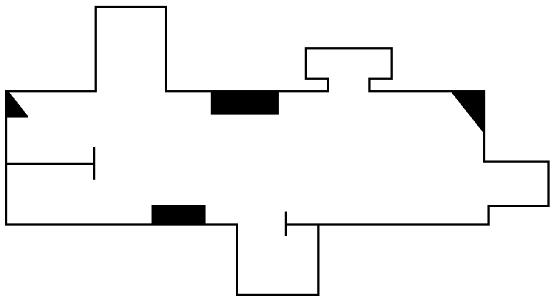
EXPERIMENTS AND RESULTS

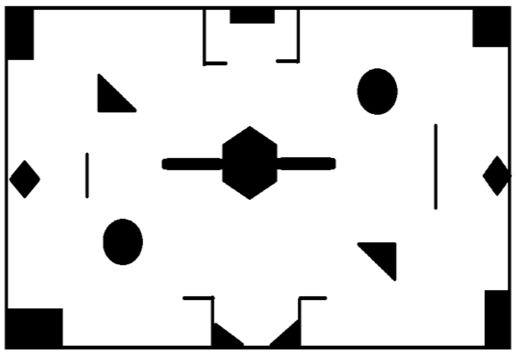
The proposed multi-robot coverage approach was tested extensively through a number of experiments performed in several environments and in various conditions. This chapter presents the results of these experiments and shows how the proposed approach stands in comparison to other existing approaches.

5.1 Experimental Setup

The experiments were performed on simulations of robot environments created through Stage 2D [39]. The simulated robots were configured with a hokuyo laser [63] scanner with an 180-degree field of view and a range of 6 meters. The experiments were performed on a personal computer with 4GB of RAM and an Intel Core i3 processor. The robots were tested in three different types of environments, which were unknown to the robots at the beginning of the experiments. Fig 5.1 displays the three different types of custom-built maps used to create the environments and conduct experiments. In the maps, the white color represents free space, while black color represents obstacles or boundaries. All three maps are rectangular, with dimensions of 34 x 30 meters.

Fig 5.1(a) depicts a map of the hallway with an open area in the center and small linked spaces on the sides. This map is an introductory map to test the basic capabilities of the multi-robot system. Fig 5.1(b) represents a map of an asymmetric room containing numerous obstacles. This map is much more complex and tests the behavior of the system in an obstacle-rich environment. Fig 5.1(c) depicts the map of an office environment with long and narrow corridors and multiple cubicles. This map tests the behavior of the system in an environment where the number of frontiers may become fewer than the number of robots frequently.


(a) Hallway map


(b) Asymmetric room map

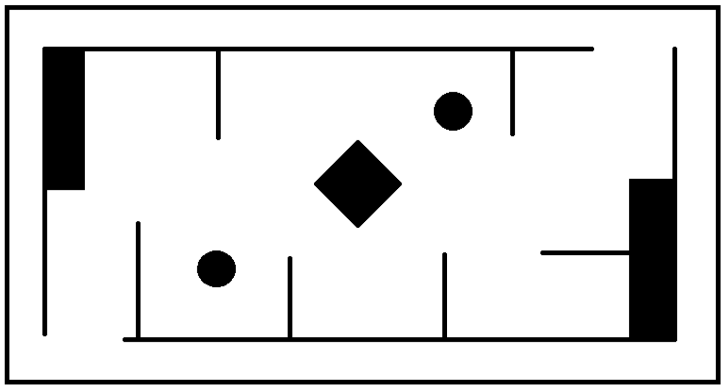

(c) Office Map

Fig 5.1 Custom-built maps for the coverage experiments.

5.2 Coverage Results

This section presents the coverage results of the experiments performed in three environments using the maps in Fig 5.1. It is demonstrated from the experiments conducted in [9], [13] and [15], that the Rank Based and the Threshold Based coverage approaches are superior to the Nearest Based and the Greedy Based approaches in most of the situations. In addition to this, the implementation details of Enhanced Frontier Based coverage approach were obscure and required extra time and effort to implement from the start. Therefore, the experiments were performed for three coverage strategies only, i.e., the Rank Based approach, the Threshold Based approach, and the proposed Remember-All Based approach. For each map, ten experiments were conducted with each of the three selected strategies.

The comparison parameters for these results are runtime, the percentage of duplicate frontiers and frontiers coverage percentage among individual robots. Runtime is the amount of time taken by the robots to perform coverage in a given environment. For example, if the coverage process of a robot using Rank Based approach in a map takes 200 seconds, then the runtime of Rank Based approach for that map is 200.

In the occupancy grid, created in ROS each frontier has a unique id associated with it. By comparing the id of an allocated frontier with that of already explored frontiers, the algorithm can detect if an already explored frontier is explored again by any robot. The percentage of duplicate frontiers metric denotes how many explored frontiers were explored again by the robots. The value of duplicate frontiers % is calculated through the formula depicted below in equation 4.2.

$$Duplicate\ Frontiers\ \% = \frac{Number\ of\ Duplicate\ Frontiers}{Total\ Number\ of\ Frontiers} * 100 \tag{4.2}$$

For example, if in a given map the robot has explored 100 frontiers and out of all these frontiers, 20 frontiers are already explored. Then, the percentage of duplicate frontiers will be 20 %. The third comparison metric is the Individual frontier coverage percentage which depicts the percentage of individual contribution by each robot. The formula used to calculate this metric for each robot is depicted below in equation 4.3. In the below sections

Robot X Frontiers % denote the percentage share of robot X out of all the covered frontiers, where the value of X can be 1 , 2 or 3.

$$Frontier\ Coverage\ \%\ of\ Robot\ i = \frac{Number\ of\ frontiers\ explored\ by\ robot\ i * 100}{Number\ of\ frontiers\ explored\ by\ all\ the\ robots} \tag{4.3}$$

In an ideal scenario, both of the robots in a two-robot system should explore 50% each of the total explored frontiers. But, this number varies depending upon the nature of the map and the coverage strategy being used.

5.2.1 Coverage Results for Two-Robot Systems

This section displays the coverage results in all three environments using two robots. The two robots start near the center of the map, within close proximity of each other, and then proceed depending on the coverage strategy being used.

Hallway Map

Table 5.1 displays the comparison of Rank Based, Threshold Based and the proposed Remember-All Based approach for the hallway map. All the results in this table are averages of the set of values calculated over ten runs of experiments performed.

Table 5.1 Comparison of Coverage Strategies for the Hallway map

Frontier Strategy	Runtime (in seconds)	Duplicate Frontiers (%)	Robot 1 Frontiers (%)	Robot 2 Frontiers (%)
Remember-All Based	160.2	0	57.98	42.02
Rank Based	166	25.66	65.30	34.70
Threshold Based	224.7	31.54	40.38	59.62

As it can be observed from the above table, the proposed Remember-All Based approach has an average run time of 160.2 seconds, which is better than that both of the other two coverage approaches. The proposed approach is faster than the Threshold Based approach by 28.5 % and 3.4% faster than the Rank Based approach. Even, in terms of percentage of duplicate frontiers, the Remember-All Based approach gives the best performance, with no duplicate frontiers.

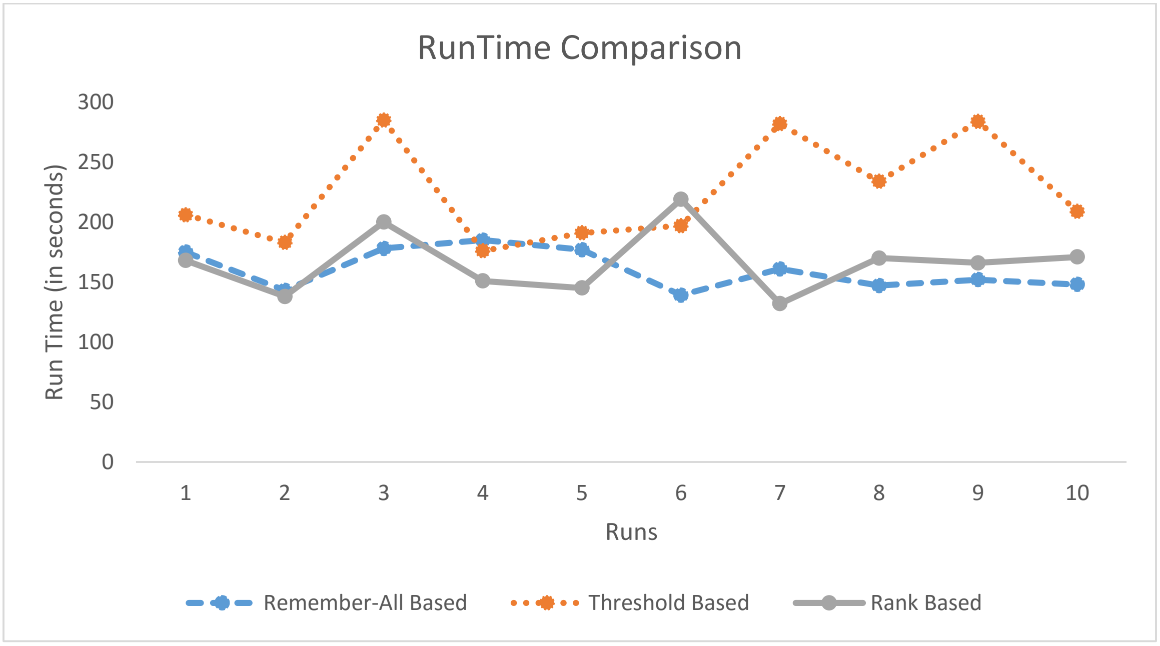


Fig 5.2 Graph of robot coverage run-time in the hallway map.

The Remember-All Based approach performs better than the Threshold Based approach because the lower percentage of duplicate frontiers in the Remember-All Based approach reduces the robot overlap, leading to faster coverage of the environment.

Fig 5.2 represents the runtime of the robots for each coverage strategy over ten runs. As we can see from the graph, the maximum individual time is taken by the Threshold Based approach for run 3, when it covered the hallway map in 285 seconds. In comparison to this, the minimum individual time is 132 seconds for Rank Based approach in run 7.

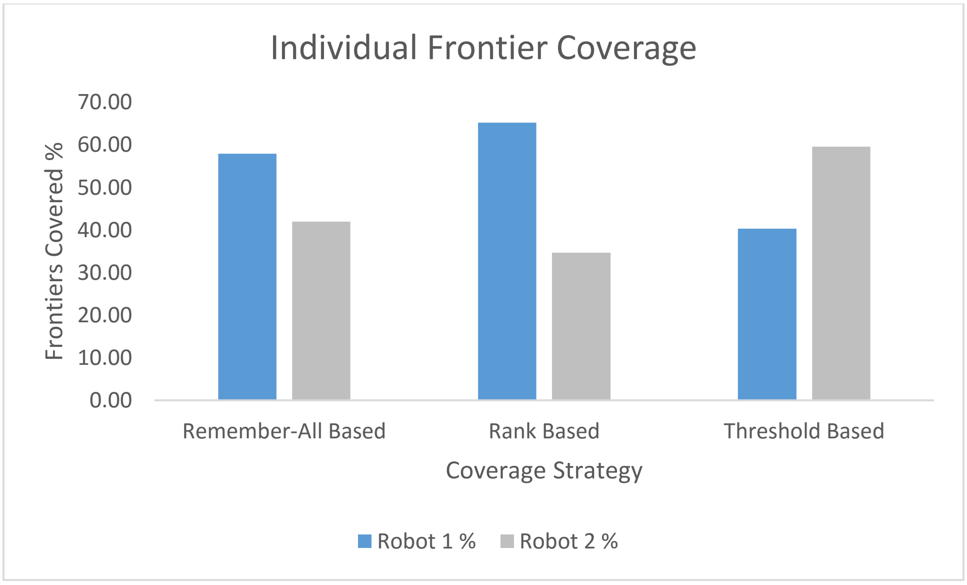


Fig 5.3 Graph of percentage of frontier coverage by individual robots in the hallway map.

Fig 5.3 represents the average frontier coverage percentage by the individual robots in the hallway environment over the course of ten runs. As it can be observed from the below figure, the share of explored frontiers is higher in the case of robot 1 in the proposed Remember-All Based and the Rank Based approach. While, in the case of Threshold Based approach, robot 2 has more share of explored frontiers than robot 1. Out of all the three approaches, Rank Based approach has the highest difference in the individual frontier coverage % (15.30 %), while the proposed Remember-All Based approach has the least difference (7.98%) in the individual frontier coverage %.

Asymmetric Room Map

This map is overall asymmetric in nature, but is obstacle rich and has few symmetric regions which can cause confusion in the robots. Table 5.2 displays the comparison of the Rank Based, the Threshold Based and the proposed Remember-All Based approach for the asymmetric map. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

As it can be observed from the below table, the proposed Remember-All Based approach has the best performance with the average run time of 228.9 seconds and a duplicate frontier percentage of 0%. In comparison, the Threshold Based approach has the worst performance with the average run time of 286.9 seconds (20.2 % higher) and a duplicity percentage of 26.83%.

The threshold based strategy performs worst here as it has a large number of duplicate frontiers (already explored frontiers, which are again explored by the robots) and it forces the robot to first explore all the marked frontiers rather than choosing best frontiers at each step.

Table 5.2 Comparison of Coverage Strategies for the Asymmetric room map

Frontier Strategy	**Runtime (in seconds)**	**Duplicate Frontiers (%)**	**Robot 1 Frontiers (%)**	**Robot 2 Frontiers (%)**
Remember-All Based	228.9	0	54.26	45.74
Rank Based	249.1	27.99	48.17	51.83
Threshold Based	286.9	26.83	42.05	57.95

Fig 5.4 represents the run time of the robots with each coverage strategy over ten runs. As we can see from the graph, the maximum individual time is taken by the Threshold Based approach for run 7, when it covered the asymmetric map in 381 seconds. In comparison to this, the minimum individual time is 164 seconds for Rank Based approach in run 2.

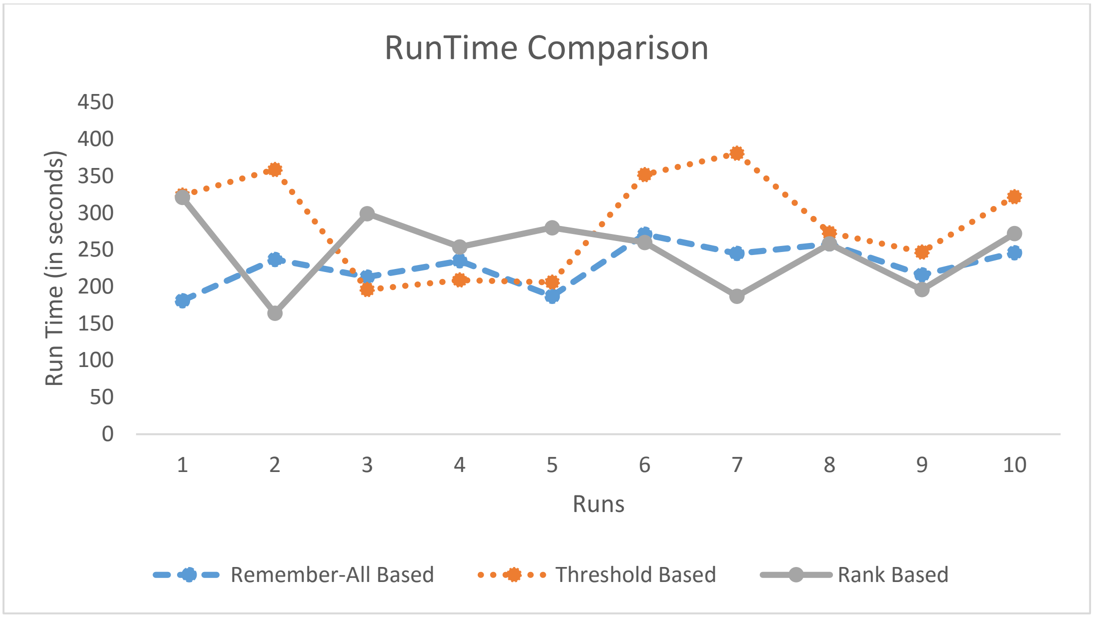


Fig 5.4 Graph of robot coverage run-time in the asymmetric room map

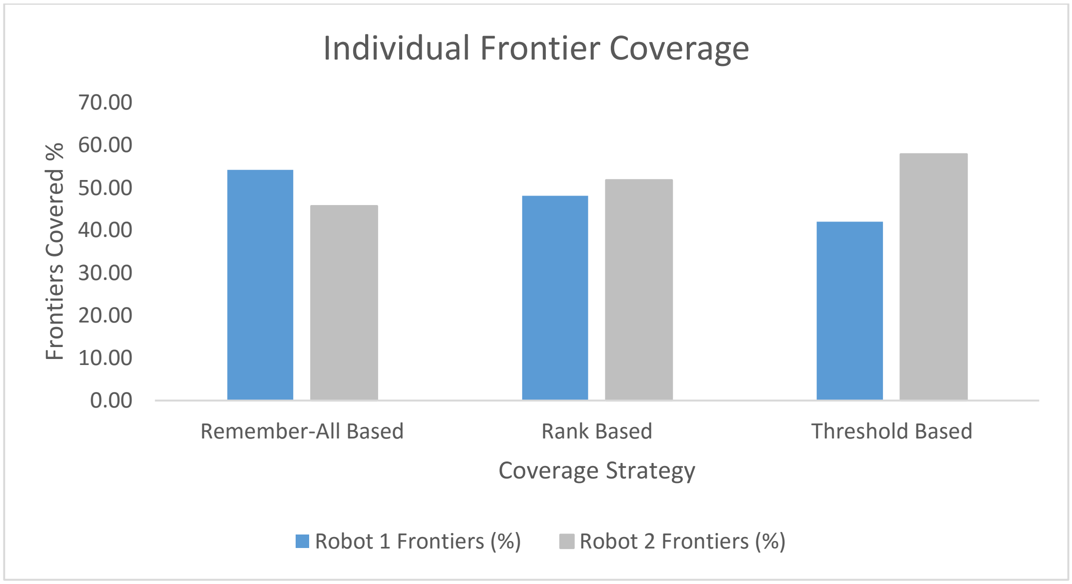


Fig 5.5 Average frontier coverage by individual robots in the asymmetric hall map

Fig 5.5 represents the average of individual frontier coverage of both the robots in the asymmetric room map over the course of ten runs. As it can be observed from the below figure, the share of explored frontiers is higher in the case of robot 1 in the proposed Remember-All Based approach. While, in the case of the other two approaches, robot 2 has more share of explored frontiers than robot 1. Out of all the three approaches, Threshold Based approach has the highest difference in the individual frontier coverage % (7.95 %), while the proposed Rank Based approach has the least difference (1.83 %) in the individual frontier coverage %.

Office Map

This map is a simulation of an office environment with long narrow corridors, obstacles and cubicles. This map tests the behavior of the robots and the coverage strategies in a situation where the number of frontiers may become fewer than the number of robots. Table 5.3 displays the comparison of Rank Based, Threshold Based and proposed Remember-All Based approach for the office map. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

Table 5.3 Comparison of Coverage Strategies for the Office map

Frontier Strategy	**Runtime (in seconds)**	**Duplicate Frontiers (%)**	**Robot 1 Frontiers (%)**	**Robot 2 Frontiers (%)**
Remember-All Based	230.3	0	60.43	39.57
Rank Based	264.1	30.7	56.19	43.81
Threshold Based	284.2	29.91	60.40	39.60

As it can be perceived from the above table, the proposed Remember-All Based approach performs best with the average run time of 230.3 seconds and a duplicate frontier percentage of 0%. The best performance of Remember-All Based approach here, can be related to the robots with no frontiers following the robots with one frontier in cases where the number of frontiers becomes fewer than the number of robots. This case is especially true in the narrow corridors where the lack of adequate frontiers, forces the algorithm to activate buddy approach.

Fig 5.6 represents the run time of the robots with each coverage strategy over ten runs. As we can see from the graph, the maximum individual time is taken by the Threshold Based approach for run 9, when it covered the asymmetric map in 429 seconds. In comparison to this, the minimum individual time is 172 seconds for the proposed Remember-All Based approach in run 7.

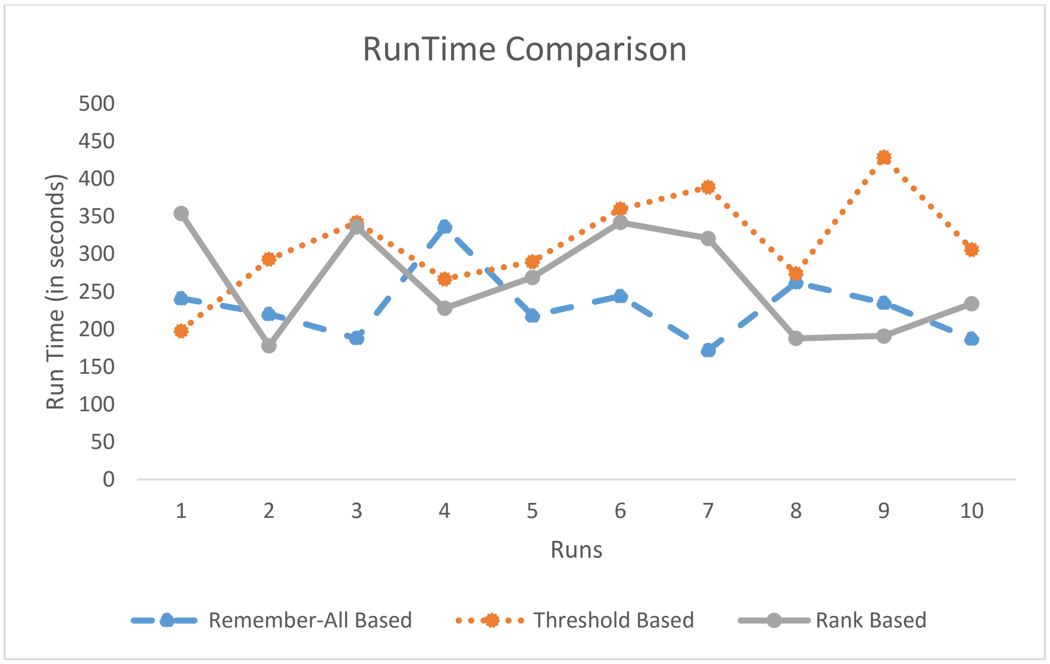


Fig 5.6 Graph of average run-time for robot coverage in the office map

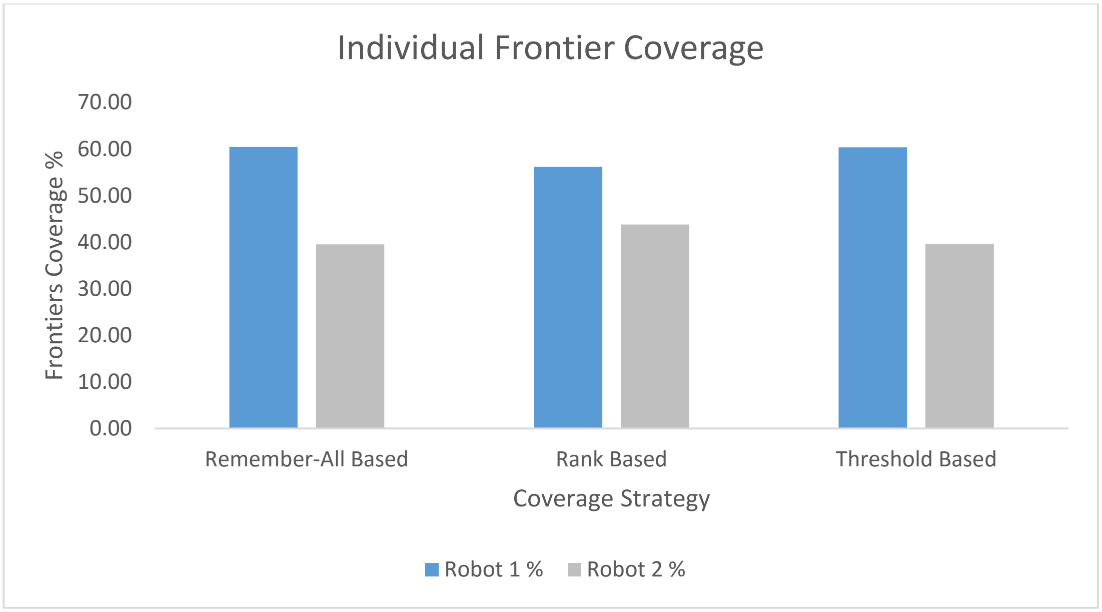


Fig 5.7 Average frontier coverage by individual robots in the office map

Fig 5.7 represents the average of individual frontier coverage of both the robots in the office map over the course of ten runs. As it can be observed from the below figure, robot 1 has higher individual frontier coverage percentage than the second robot in all the three approaches. Out of all the three approaches, Remember-All Based approach has the highest difference in the individual frontier coverage % (10.43 %), while the Rank Based approach has the least difference (6.19 %) in the individual frontier coverage %.

5.2.2 Coverage Results for Three Robot Systems

This section displays the coverage results of the robots in all three environments using three robots. All the three robots start near the center of the map, within close proximity of each other, and then proceed depending upon the coverage strategy being used.

Hallway Map

Table 5.4 displays the comparison of Rank Based, Threshold Based and proposed Remember-All Based approach for the hallway map with a three-robot system. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

As it can be observed from the below table, the proposed Remember-All Based approach has an average run time of 155 seconds which is better than both of the other coverage approaches.

Table 5.4 Comparison of Coverage Strategies for the Hallway map

Frontier Strategy	Runtime (in seconds)	Duplicate Frontiers (%)	Robot 1 Frontiers (%)	Robot 2 Frontiers (%)	Robot 3 Frontiers (%)
Remember-All Based	155	0	59.25	23.13	17.62
Rank Based	159.1	31.45	65.00	15.00	20.00
Threshold Based	189.9	34.29	47.00	20.00	33.00

While, in terms of percentage of duplicate frontiers, the Remember-All Based approach gives the best performance with 0% duplicate frontiers. The table also displays the individual frontier coverage percentage of the three robots in the map.

Fig 5.8 displays the run times of the three robot system for the hallway map over the course of ten runs. As we can see from the graph, the maximum individual time is taken by the Threshold Based approach for run 6, when it covered the hallway map in 286 seconds. In comparison to this, the minimum individual time is 132 seconds for the proposed Remember-All Based approach in run 7.

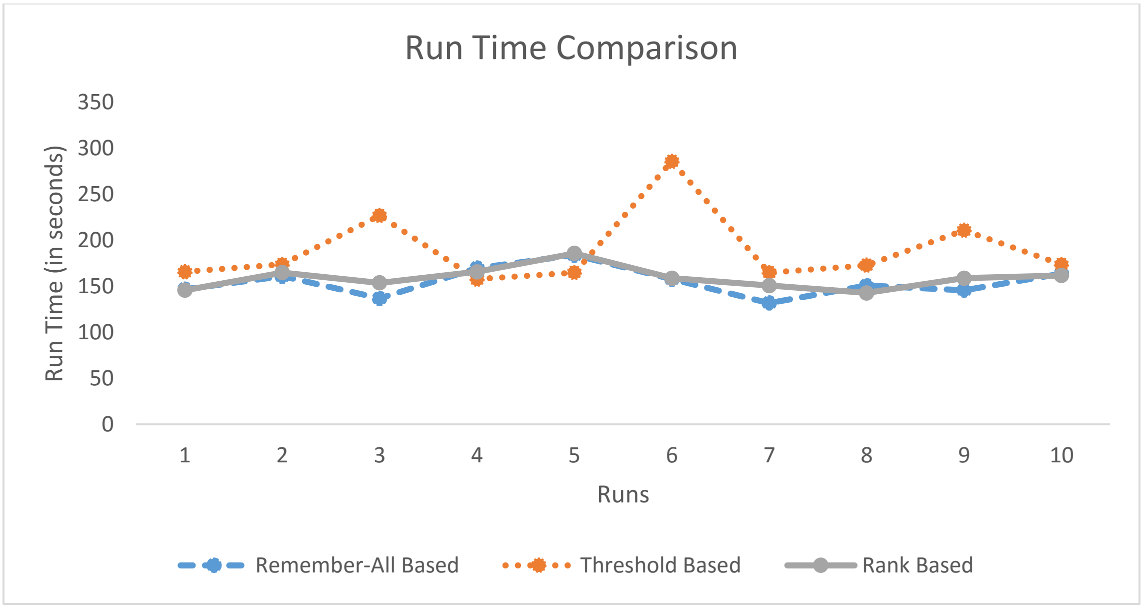


Fig 5.8 Graph of robot coverage run-time in the hallway map

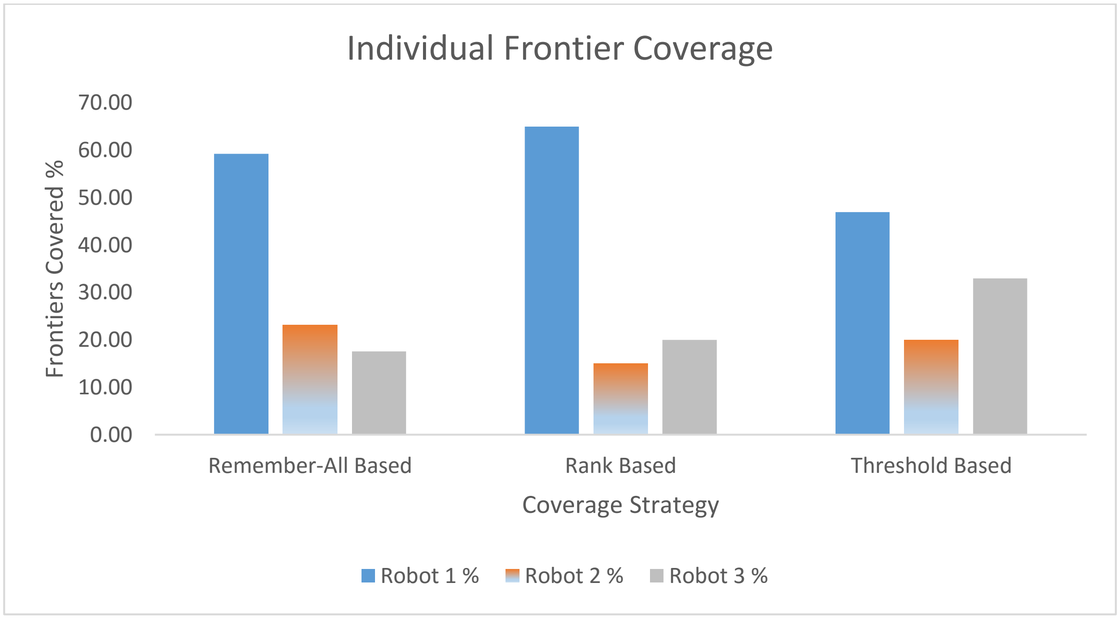


Fig 5.9 Average frontier coverage by individual robots in the hallway map

Fig 5.9 represents the average of individual frontier coverage of both the robots in the hallway map over the course of ten runs. As it can be observed from the below figure, in all the three approaches robot 1 has highest individual frontier coverage percentage than other two robots. Out of all the three approaches, Rank Based approach has the highest difference in the individual frontier coverage with robot 1 covering 65 % of the total frontiers. In comparison, the proposed Threshold Based approach has the least difference with robot 1 covering 47 % of the total frontiers.

Asymmetric Room Map

This map is overall asymmetric in nature, but is obstacle rich and has few symmetric regions which can cause confusion in the robots. Table 5.5 displays the comparison of Rank Based, Threshold Based and proposed Remember-All Based approach for the asymmetric map. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

Table 5.5 Comparison of Coverage Strategies for the Asymmetric map

Frontier Strategy	**Runtime (in seconds)**	**Duplicate Frontiers (%)**	**Robot 1 Frontiers (%)**	**Robot 2 Frontiers (%)**	**Robot 3 Frontiers (%)**
Remember-All Based	215.4	0	44.04	30.96	25
Rank Based	226.6	32.02	48	35	17
Threshold Based	248.7	29.74	44	29	27

As it can be noted from the above table the proposed Remember-All Based coverage approach provides the best performance with the average run time of 215.4 seconds. In comparison to this, the threshold based approach has an average run time of 248.7 seconds. The Remember-All Based approach also has the least percentage of duplicate frontiers of all three coverage strategies with 0 % duplicate frontiers.

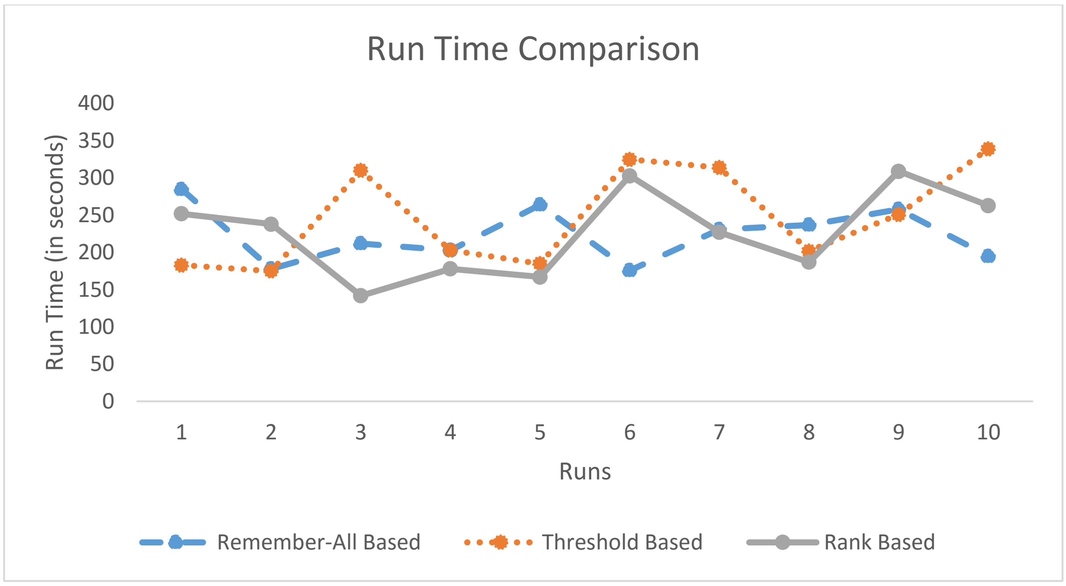


Fig 5.10 Graph of average run-time for robot coverage in the asymmetric room map

Fig 5.10 displays the run times of all the runs performed in the three robot system. As we can see from the above graph, the worst run time performance record is with the Threshold Based approach which took 339 seconds for coverage in the run 10. In comparison to that, the Rank Based approach has the lowest run time of 142 seconds in the run 3.

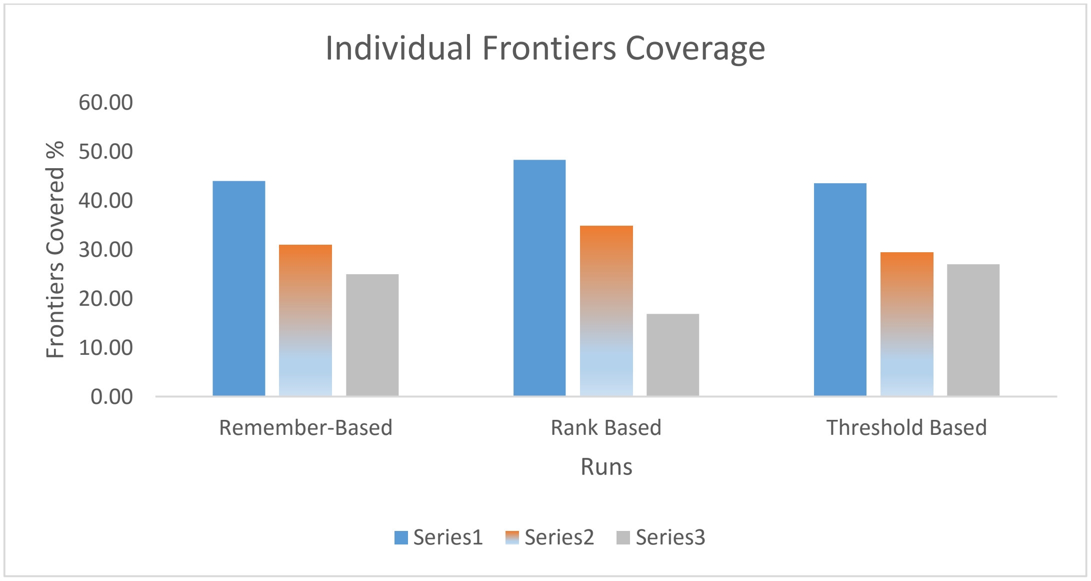


Fig 5.11 Average frontier coverage by individual robots in the asymmetric room map

Fig 5.11 represents the average frontier coverage by the individual robots over the course of ten runs in the asymmetric room map. As it can be observed from the below figure, in all the three approaches robot 1 has highest individual frontier coverage percentage than other two robots. Out of all the three approaches, Rank Based approach has the highest difference in the individual frontier coverage with robot 1 covering 48 % of the total frontiers. In comparison, the proposed Threshold Based approach has the least difference with robot 1 covering 44 % of the total frontiers.

Office Map

Table 5.6 displays the comparison of Rank Based, Threshold Based and proposed Remember-All Based approach for the office map within a three robot system. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

Table 5.6 Comparison of Coverage Strategies for the Office map

Frontier Strategy	**Runtime (in seconds)**	**Duplicate Frontiers (%)**	**Robot 1 Frontiers (%)**	**Robot 2 Frontiers (%)**	**Robot 3 Frontiers (%)**
Remember-All Based	222.5	0	55.51	19.64	24.85
Rank Based	253.8	32.27	36	33	31
Threshold Based	277.6	29.43	25	30	44

As it can be perceived from the above table, the Remember-All Based Approach provides the best performance with the average run time of 222.5 seconds and a frontier duplication of just 0 %. In comparison to this, the Threshold Based approach performs worst with the average run time of 277.6 seconds and frontier duplication of 29.43%.

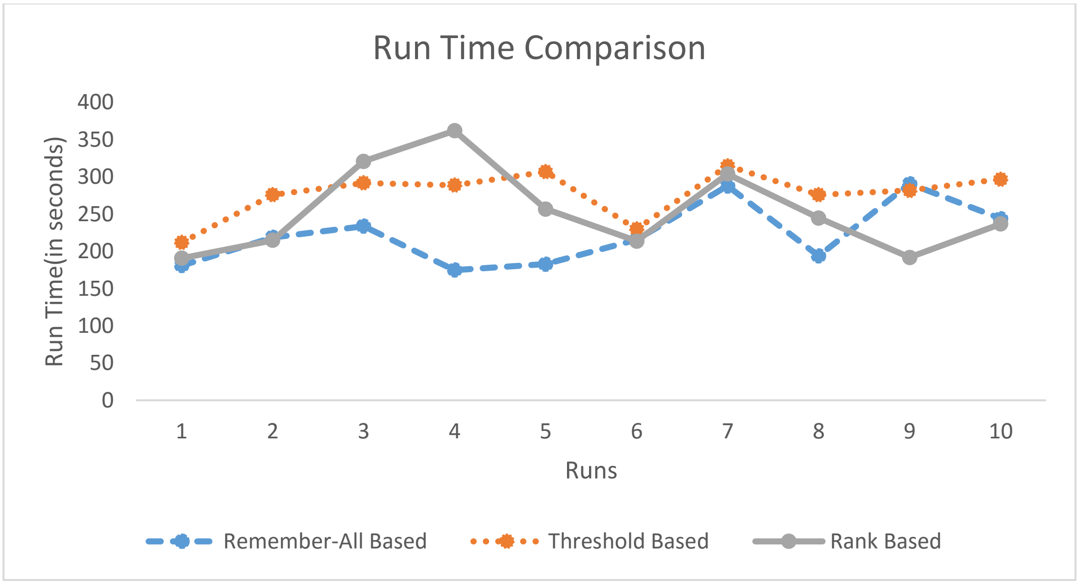


Fig 5.12 Graph of individual robot run-time in the office map

Fig 5.12 represents the run times of all the runs of experiments conducted in the three robot systems. As we can see from the below graph, the worst individual run time performance record is with the Rank Based approach which took 362 seconds for coverage on the run 4. In comparison to that, the Remember-All Based approach has the lowest run time of 172 seconds in the run 2.

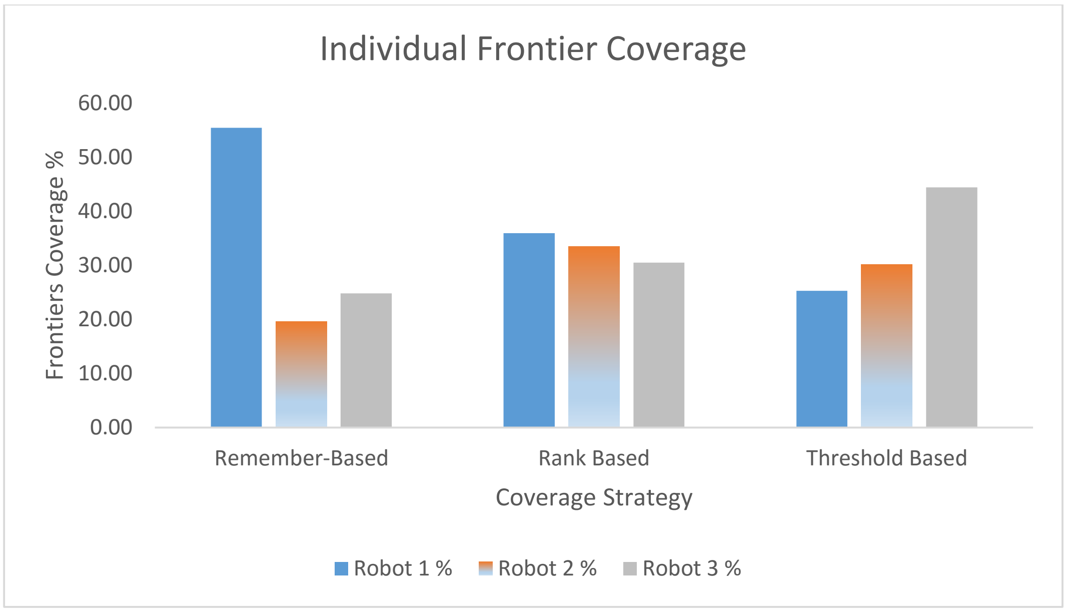


Fig 5.13 Average frontier coverage by individual robots in the office map

Fig 5.13 represents the average frontier coverage by the individual robots over the course of ten runs in the office map. As it can be observed from the below figure, robot 1 has higher individual frontier coverage percentage than other two robots in the proposed Remember-All Based approach and the Rank Based approach.

Out of all the three approaches, Rank Based approach has the least difference in the individual frontier coverage with robot 1 covering 27.11 % of the total frontiers. In comparison, the proposed Threshold Based approach has the highest difference with robot 1 covering 55.11 % of the total frontiers.

5.2.3 Coverage Results for Two Robot System (with Distant Starting Locations)

This section displays the coverage results of the robots in all three environments using two robots. Both the robots start at distant locations on the map and then proceed depending upon the coverage strategy being used.

Hallway Map

Table 5.7 displays the comparison of Rank Based, Threshold Based and proposed Remember-All Based approach for the hallway map within a two-robot system where the robots start at locations distant from each other. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

Table 5.7 Comparison of Coverage Strategies for the Hallway map

Frontier Strategy	Runtime (in seconds)	Duplicate Frontiers (%)	Robot 1 Frontiers (%)	Robot 2 Frontiers (%)
Remember-All Based	161.2	0	69.35	30.64
Rank Based	164.4	24.1	75.36	24.64
Threshold Based	210	28.59	78.42	21.58

As it can be observed from the above table, the Remember-All Based approach performs better than the Rank Based and Threshold Based approaches with an average run time of 161.2 seconds and the frontier duplication of 0 %. The frontier duplications for the other two approaches in this result set are lower than the frontier duplications in Table 5.1, as now the robots are starting at distant locations.

Fig 5.14 represents the individual run times of all the runs of experiments conducted in the two robot systems when the robots are starting at distant locations. As we can see from the graph below the best individual performance is in run 1, when the proposed Remember-All Based approach performed coverage in 137 seconds. In comparison to this, the Rank Based approach has the record for the worst individual performance of 362 seconds in the fourth run.

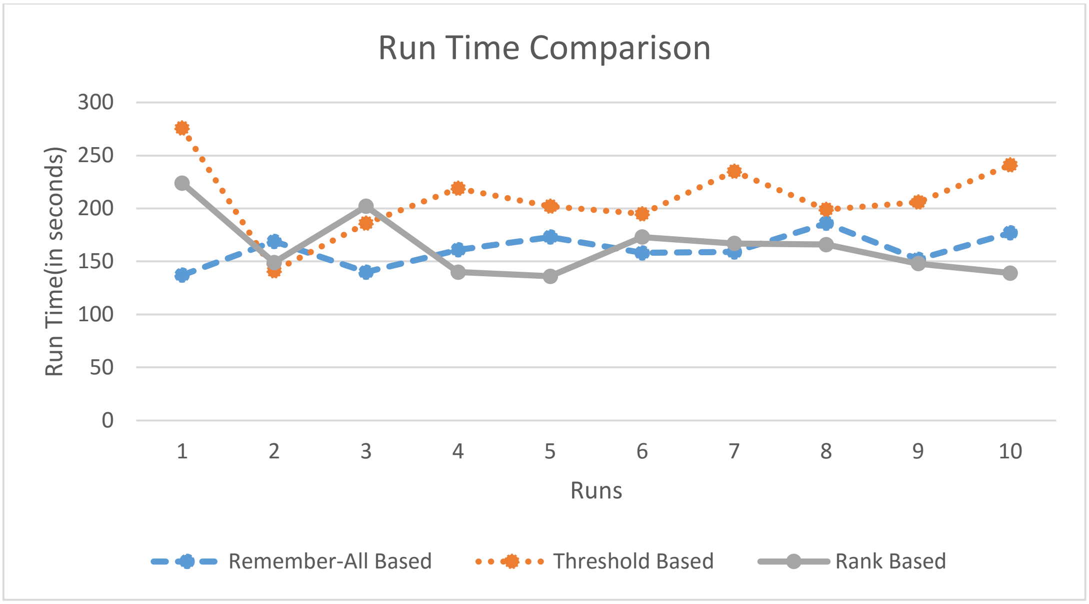


Fig 5.14 Graph of robot run-time comparison in the hallway map

Fig 5.15 represents the average of individual frontier coverage of both the robots in the hallway map over the course of ten runs. As it can be perceived from the below figure, the robot 1 has in all the cases outperformed robot 2 in terms of individual frontier coverage percentage. The main reason for this behavior is that as the robots now start in distant locations, the time taken for localization of robot 2 increases as the presence of another robot helps the robot to get localized faster.

Out of all the three approaches, Threshold Based approach has the highest difference in the individual frontier coverage with robot 1 covering 78.42% of the total frontiers. In comparison, the Rank Based approach has the least difference with robot 1 covering 75% of the total frontiers.

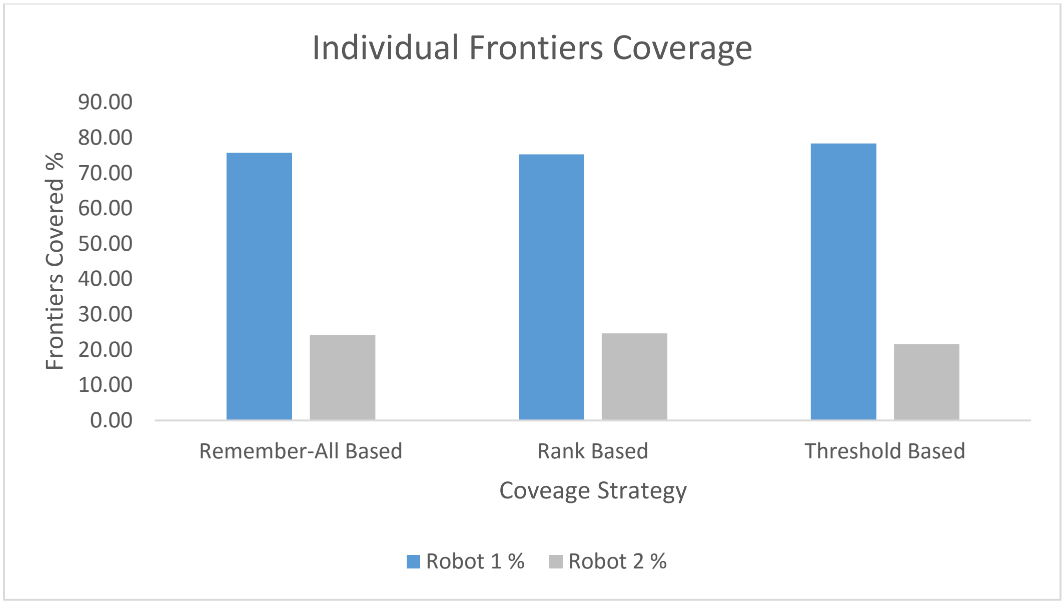


Fig 5.15 Average frontier coverage by individual robots in the hallway map

Asymmetric Room Map

Table 5.8 displays the comparison of the Rank Based, Threshold Based and proposed Remember-All Based approach for the asymmetric map within a two-robot system where the robots start at locations distant from each other. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

Table 5.8 Comparison of Coverage Strategies for the Asymmetric Room map

Frontier Strategy	**Runtime (in seconds)**	**Duplicate Frontiers (%)**	**Robot 1 Frontiers (%)**	**Robot 2 Frontiers (%)**
Remember-All Based	194.7	0	60.8	39.11
Rank Based	206.8	21.19	69.17	30.83
Threshold Based	243.8	25.72	68.86	31.14

As we can see from the above table, the Remember-All Based approach has the best average run time of 194.7 seconds and also the lowest percentage of duplicate frontiers – 0 %. The

Threshold Based approach performs worst with the average run time of 243.8 seconds and 25.72 % duplicate frontiers. The Rank Based performance lies in between the other two approaches. Fig 5.16 represents the individual run times of thirty runs performed in the asymmetric room map with the three coverage strategies.

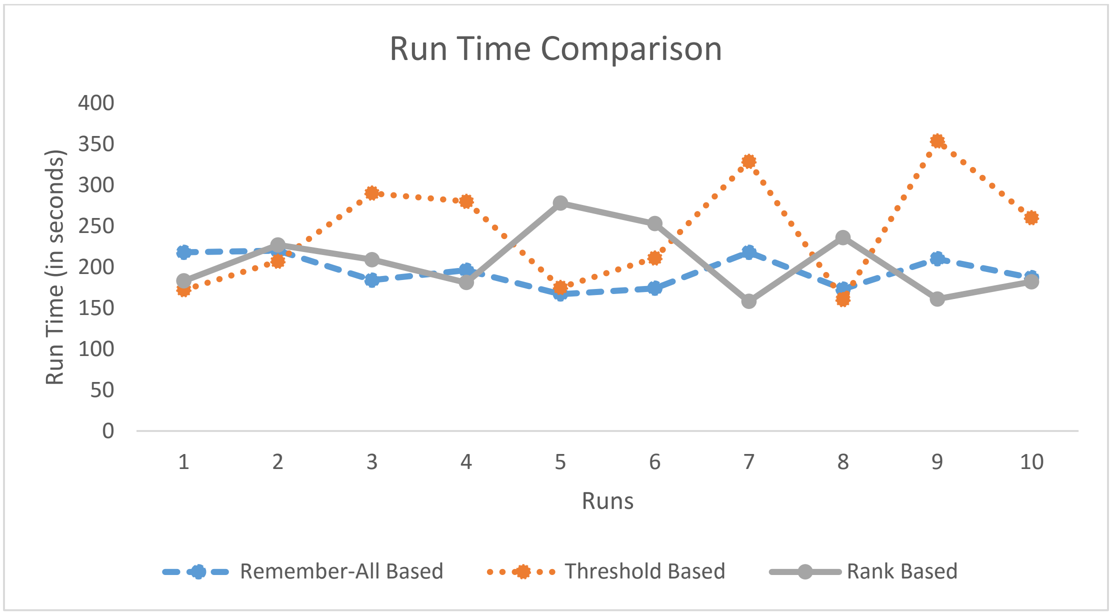


Fig 5.16 Graph of robot run-time comparison in the asymmetric room map

As it is clear from the above graph, the Remember-All Based approach has the best individual performance in run 4 with the run time of 158 seconds. In comparison to that, the worst individual performance is given by the Threshold Based Approach which has the run time of 354 in run 9.

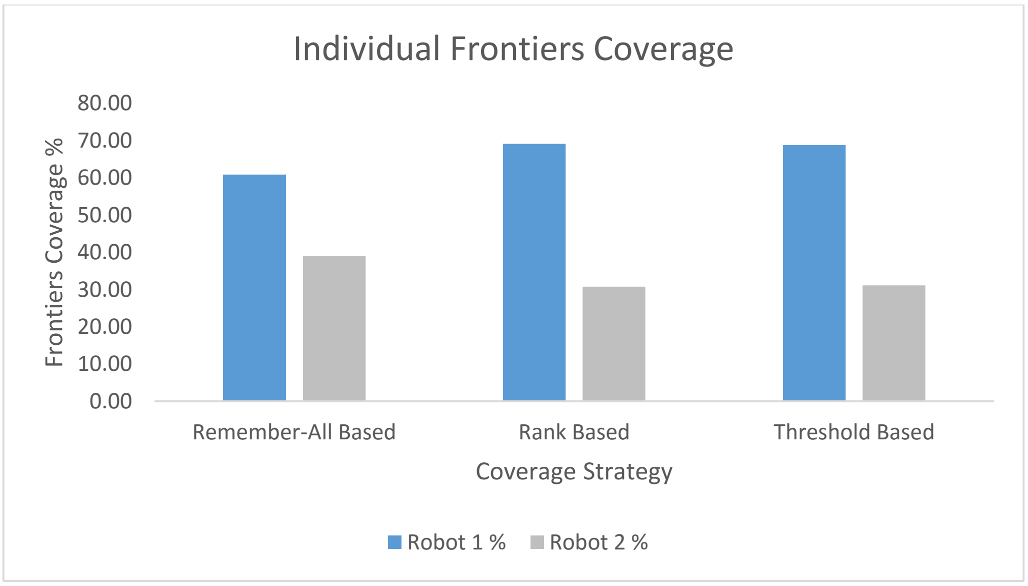


Fig 5.17 Average frontier coverage by individual robots in the asymmetric room map

Fig 5.17 represents the average individual frontier coverage of both the robots over the course of ten runs. As we can observe the frontier coverage % of the robot 1 decreases a bit in comparison to Fig 5.15. The main reason for this is that this environment is highly rich is obstacles, which help the robot 2 to get localized a bit faster. As it can be perceived from the above figure, the robot 1 has in all the cases outperformed robot 2 in terms of individual frontier coverage percentage.

Out of all the three approaches, Rank Based approach has the highest difference in the individual frontier coverage with robot 1 covering 69.17 % of the total frontiers. In comparison, the proposed Remember-All Based approach has the least difference with robot 1 covering 60.89% of the total frontiers.

Office Map

Table 5.9 displays the comparison of the three coverage strategies - Rank Based, Threshold Based and proposed Remember-All Based approach for the office map within a two-robot system. All the results in this table are averages of the set of values calculated over the ten runs of experiments performed.

Table 5.9 Comparison of Coverage Strategies for the Office map

Frontier Strategy	**Runtime (in seconds)**	**Duplicate Frontiers (%)**	**Robot 1 Frontiers (%)**	**Robot 2 Frontiers (%)**
Remember-All Based	239.30	0	62.30	37.69
Rank Based	286.30	28.17	71.43	28.57
Threshold Based	281.90	29.35	65.87	34.13

As it can be observed from the above table, the proposed Remember-All Based approach provides the best performance with an average run time of 239.30 seconds and a frontier duplication percentage of 0 %. The Threshold Based approach provides the worst performance with an average run time of 281.90 seconds and the frontier duplication percentage of 29.35 %. The Remember-All Based approach is having a higher performance as a result of storage of performance and usage of buddy approach in cases where the number of frontiers becomes fewer than the number of robots.

Fig 5.18 represents the individual run times of the robots through all the runs in a graphical manner. As we can see from the graph below, the worst individual performance is given by Rank Based approach in run 10, with the run time of 409 seconds. In contrast to this, the best performance is given by the Remember-All Based approach in run 3, with the run time of 195 seconds.

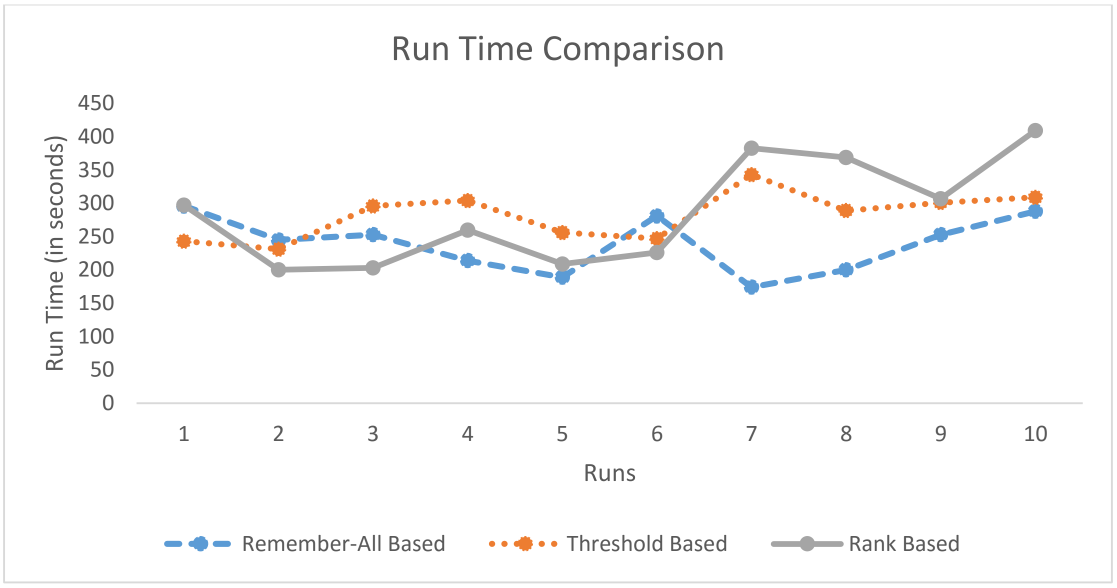


Fig 5.18 Graph of robot run-time comparison in the office map

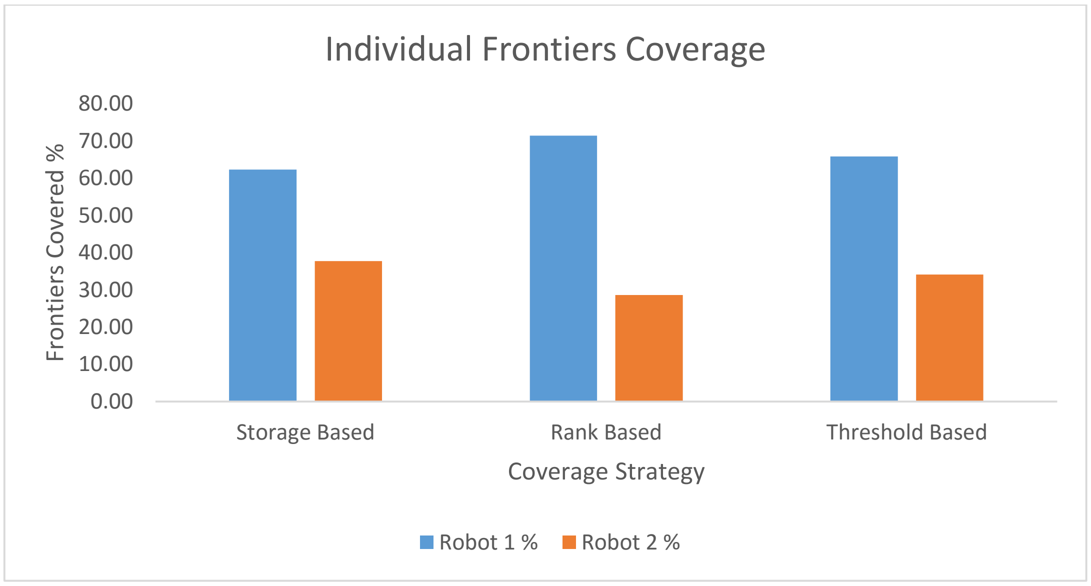


Fig 5.19 Average frontier coverage by individual robots in the office map

Fig 5.19 represents the individual average of frontier coverage of the robots in the office environment in a graphical manner. As it can be perceived from the above figure, the robot 1 has in all the cases outperformed robot 2 in terms of individual frontier coverage percentage. Out of all the three approaches, Rank Based approach has the highest difference in the individual frontier coverage with robot 1 covering 71.43 % of the total frontiers. In comparison, the Remember-All Based approach has the least difference with robot 1 covering 62.30% of the total frontiers.

All of the results data obtained from the experiments is available for reference in Appendix A. The data consists of values of runtime, count of duplicate frontiers and individual frontiers coverage of robots for ten iterations for each experiment set. The iterations here denote a complete run from start to end of the program.

5.3 Summary

This chapter discussed the various experiments and their results using a variety of frontier based coverage approaches. The experiments were performed on three kinds of custom built maps – hallway, asymmetric room, and office. Based on the results of the above experiments, following conclusions can be drawn:

- The proposed Remember-All approach has the zero percentage of duplicate frontiers in all the three environments
- The proposed Remember-All approach performs better than the Threshold Based approach and the Rank Based approach in in all the three environments
- The performance of the coverage approaches improves when the number of robots is increased from two robots to three robots
- The performance of the coverage approaches generally deteriorates if the robots are started at distant locations to each other. This is due to the fact that, as the robots are away from each other, the time for localization increases

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis, concepts of multi-robot exploration are explained and a new frontier-based approach for multi-robot exploration is presented for unknown environments using ROS framework. This implementation is built on already available open source implementations of algorithms in ROS, such as Map Merging, SLAM and Localization. The proposed coverage approach was tested rigorously in a 2D open source robot simulator called Stage.

6.1 Conclusion

The proposed Remember-All Based approach was tested in three custom-built environments – hallway, asymmetric room and office. The performance of the approach was compared with already existing coverage approaches – Rank Based and Threshold Based – on the parameters of runtime, the percentage of duplicate frontiers and individual coverage percentage by each of the robots.

In a two-robot system, where the two robots started in close proximity to each other, the proposed approach outperformed the Threshold Based approach in all the three maps with an 18-28.5% improvement in runtime. While in comparison to the Rank Based approach, the proposed approach provides an improvement of 3.3 – 12.7 % in runtime.

In the next set of experiments called three-robot systems, one additional robot was added to the environment to see how the coverage was affected. In all three environments, the runtime for coverage was reduced in comparison to the two-robot system. However, the percentage of duplicate frontiers increased in each case due to the presence of more robots. In this set of experiments, the proposed Remember-All Based approach outperformed the

Threshold Based approach in all three maps with a runtime improvement of 13.3 – 19.8 %. The proposed approach also improved its runtime in comparison to the Rank Based approach by 2.5 – 12.3%.

In the last set of experiments, the coverage strategy was tested in a two-robot environment where the robots started in locations far away from each other, making localization difficult. In this scenario, the proposed Remember-All Based approach outperformed the Threshold Based approach in all the three maps with a runtime improvement of 14.8 – 23.2 %. The proposed approach also outperformed the Rank Based approach with a runtime improvement of 1.8 – 16.2%.

In this particular scenario, the individual area coverage of the first robot outpaces that of the second robot by a higher margin than in comparison to other experiment sets, due to the increased time taken for the localization of robot 2. In all the three experiments, the proposed Remember-All Based approach has the lowest frontier duplication percentage.

In the end, we can conclude that the proposed coverage approach provides a good alternative to the existing coverage strategies to perform multi-robot coverage in unknown environments.

6.2 Future Work

One of the major areas for the future work can be the implementation of the proposed approach for the physical robots. The use of robot simulators allows the researchers to test their algorithms/strategies at a much rapid pace and at a relatively low cost. A simulator allows us to quickly make changes in the code and see how the simulated robots react to the change. But, testing this strategy on physical robots presents a good case for future work as the simulation is a controlled environment and does not take into account external factors which are not thought by the developer. Implementing this coverage strategy for the physical robots such as 'Turtlebot' will allow us to see how the proposed approach behaves in the real world scenarios.

It was discovered from the analysis of the results that the performance of proposed approach is lower in small and relatively open environments such as the hallway map in comparison to the other two maps. This behavior presents a good case for further investigation. In the future, the proposed approach could be tested on special custom built environments to see how the proposed approach behaves in the open environments in comparison to congested areas.

BIBLIOGRAPHY

[1] Long, T. (2016). Jan. 25, 1921: Robots First Czech In. [Online] WIRED. Available at: http://www.wired.com/2011/01/0125robot-cometh-capek-rur-debut/ [Accessed 25 May 2016].

[2] Forumonenergy.com. (2016). Rescue Robots: The Future of Nuclear Cleanup: Forum on Energy. [Online] Available at: http://forumonenergy.com/2013/12/30/rescue-robots-the-future-of-nuclear-cleanup [Accessed 25 May 2016].

[3] Ranger, S. (2014). After space exploration and war, robots are ready for their biggest challenge: Conquering the home - TechRepublic. [Online] TechRepublic. Available at: http://www.techrepublic.com/blog/european-technology/after-space-exploration-and-war-robots-are-ready-for-their-biggest-challenge-conquering-the-home/ [Accessed 25 May 2016].

[4] ScienceDaily. (2016). Underwater robots on course to the deep sea. [Online] Available at: http://www.sciencedaily.com/releases/2010/11/101123121105.htm [Accessed 25 May 2016].

[5] Hougen, Dean F., et al. "A miniature robotic system for reconnaissance and surveillance." Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on. Vol. 1. IEEE, 2000.

[6] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT press, 2005.

[7] Frc.ri.cmu.edu. (2016). Nomad's Autonomy System. [online] Available at: http://www.frc.ri.cmu.edu/projects/bigsignal/2000/background/telerobotics/autonomy/new_autonomy.html [Accessed 25 May 2016].

[8] Golden, Bruce, et al. "The generalized covering salesman problem." INFORMS Journal on Computing 24.4 (2012): 534-553.

[9] Pappas, Brian. Multi-Robot Frontier Based Map Coverage Using the ROS Environment. Diss. Auburn University, 2014.

[10] Mannadiar, Raphael, and Ioannis Rekleitis. "Optimal coverage of a known arbitrary environment." Robotics and Automation (ICRA), 2010 IEEE International Conference on. pp. 5525-5530. IEEE, 2010.

[11] Choset, Howie. "Coverage of known spaces: The boustrophedon cellular decomposition." Autonomous Robots 9.3 (2000): 247-253.

[12] Choset, Howie. "Coverage for robotics–A survey of recent results." Annals of mathematics and artificial intelligence 31.1-4 (2001): 113-126.

[13] Bautin, Antoine, Olivier Simonin, and François Charpillet. "Minpos: A novel frontier allocation algorithm for multi-robot exploration." Intelligent Robotics and Applications. Springer Berlin Heidelberg, 2012. 496-508.

[14] Burgard, Wolfram, et al. "Collaborative multi-robot exploration." Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on. Vol. 1. IEEE, 2000.

[15] Muddu, Raja Sankar Dileep. "A Frontier Based Multi-Robot Approach for Coverage of Unknown Environments." MSc thesis, University of Windsor (2015).

[16] Y. Tan, Handbook of research on design, control, and modeling of swarm robotics. IGI Global, 2015.

[17] Electronicsteacher.com. (2016). Sensors - Robotics Technology. [Online] Available at: http://www.electronicsteacher.com/robotics/robotics-technology/sensors.php [Accessed 25 May 2016].

[18] Martinez, Aaron, and Enrique Fernández. Learning ROS for robotics programming. Packt Publishing Ltd, 2013.

[19] ROS Wiki: ROS. http://wiki.ros.org/ROS (Accessed on 06/02/2016).

[20] ReliefWeb. (2016). The human cost of natural disasters 2015: a global perspective. [Online] Available at: http://reliefweb.int/report/world/human-cost-natural-disasters-2015-global-perspective [Accessed 25 May 2016].

[21] Cordless Vacuum Guide. (2016). Neato vs Roomba (with Infographic). [Online] Available at: http://www.bestcordlessvacuumguide.com/neato-vs-roomba [Accessed 25 May 2016].

[22] Brown, M. and Brown, M. (2016). iRobot's new Roomba 980 vacuum features room-mapping technology and Wi-Fi. [Online] TechHive. Available at: http://www.techhive.com/article/2984337/smart-appliance/irobots-new-roomba-980-vacuum-features-room-mapping-technology-and-wi-fi.html [Accessed 25 May 2016].

[23] M. Welling, "Intelligent Agents", 1st ed. 2009.

[24] Cameron, Stephen, and Julian de Hoog. "Selection of Rendezvous Points for Multi−Robot Exploration in Dynamic Environments." (2010).

[25] Leonard, John J., and Hugh F. Durrant-Whyte. "Mobile robot localization by tracking geometric beacons." Robotics and Automation, IEEE Transactions on 7.3 (1991): 376-382.

[26] Stachniss, Cyrill. Exploration and mapping with mobile robots. Diss. University of Freiburg, 2006.

[27] Chen, Chen, and Yinhang Cheng. "Research on map building by mobile robots." Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on. Vol. 2. IEEE, 2008.

[28] Thrun, Sebastian. "Robotic mapping: A survey." Exploring artificial intelligence in the new millennium 1 (2002): 1-35.

[29] J. Röning, J. Riekki and T. Seppänen, "Intelligent Systems Group (ISG)", Infotech Oulu, 2008. Available: http://www.infotech.oulu.fi/Annual/2008/isg.html. [Accessed: 20-Feb- 2016].

[30] Se, Stephen, David Lowe, and Jim Little. "Local and global localization for mobile robots using visual landmarks." Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on. Vol. 1. IEEE, 2001.

[31] N Leena and K.K. Saju, "A survey on path planning techniques for autonomous mobile robots", IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE), pp. 76-79

[32] Sariff, N., and Norlida Buniyamin. "An overview of autonomous mobile robot path planning algorithms." Research and Development, 2006. SCOReD 2006. 4th Student Conference on. IEEE, 2006.

[33] Lynne E. Parker, "Multi Robot Path Planning and Motion Coordination", 2011.

[34] Gage, Douglas W. "Randomized search strategies with imperfect sensors." Optical Tools for Manufacturing and Advanced Automation. International Society for Optics and Photonics, 1994.

[35] Mcs.alma.edu. (2016). GridWalker. [Online] Available at: http://www.mcs.alma.edu/LMICSE/LabMaterials/AlgoComp/Lab3/AlgCoL3.htm [Accessed 25 May 2016].

[36] M. Ribeiro and P. Lima, Motion Planning, 1st ed. Lisbon, Portugal, 2002.

[37] Howard, Andrew, Maja J. Matarić, and Gaurav S. Sukhatme. "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem." Distributed Autonomous Robotic Systems 5. Springer Japan, 2002. 299-308.

[38] Yamauchi, Brian. "Frontier-based exploration using multiple robots." Proceedings of the second international conference on Autonomous agents. ACM, 1998.

[39] Richard Vaughan. "Massively Multiple Robot Simulations in Stage", Swarm Intelligence 2(2-4):189-208, 2008. Springer.

[40] O'Kane, Jason M. "A Gentle Introduction to ROS." (2014): 1564.

[41] ROSWiki: packages. http://wiki.ros.org/Packages (Accessed on 11/04/2016).

[42] ROSWiki: Services. http://wiki.ros.org/Services (Accessed on 11/04/2016).

[43] Al Khawaldah, Mohammad, and Andreas Nüchter. "Enhanced frontier-based exploration for indoor environment with multiple robots." Advanced Robotics 29.10 (2015): 657-669.

[44] Elizondo-Leal, Juan C., Gabriel Ramírez-Torres, and Gregorio Toscano Pulido. "Multi-robot Exploration and Mapping Using Self Biddings and Stop Signals." MICAI 2008: Advances in Artificial Intelligence. Springer Berlin Heidelberg, 2008. 615-625.

[45] Siegwart, Roland, Illah R. Nourbakhsh, and Davide Scaramuzza. "Autonomous mobile robots." Massachusetts Institute of Technology (2004).

[46] Cse17-iiith.virtual-labs.ac.in. (2016). Virtual Labs. [Online] Available at: http://cse17-iiith.virtual-labs.ac.in/mapping/index.php [Accessed 25 May 2016].

[47] Adluru, Nagesh, et al. "Merging maps of multiple robots." Pattern Recognition, 2008. ICPR 2008. 19th International Conference on. IEEE, 2008.

[48] Galceran, Enric, and Marc Carreras. "A survey on coverage path planning for robotics." Robotics and Autonomous Systems 61.12 (2013): 1258-1276.

[49] ROSWiki: Messages http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv (Accessed on 11/04/2016).

[50] ROSWiki: Nodes. http://wiki.ros.org/Nodes (Accessed on 11/04/2016).

[51] ROSWiki: Master. http://wiki.ros.org/Master (Accessed on 11/04/2016).

[52] ROSWiki: Rqt_graph. http://wiki.ros.org/rqt_graph (Accessed on 11/04/2016).

[53] ROSWiki: Navigation. http://wiki.ros.org/navigation (Accessed on 11/04/2016).

[53] ROSWiki: AMCL. http://wiki.ros.org/amcl (Accessed on 11/04/2016).

[54] ROSWiki: Map_Merger. http://wiki.ros.org/map_merger (Accessed on 11/04/2016).

[55] ROSWiki: CostMap_2D. http://wiki.ros.org/costmap_2d (Accessed on 11/04/2016).

[56] ROSWiki: Nav2d_exploration. http://wiki.ros.org/nav2d_exploration (Accessed on 11/04/2016).

[57] ROSWiki: Nav2d. http://wiki.ros.org/nav2d (Accessed on 11/04/2016).

[58] Friedman, Stephen, Hanna Pasula, and Dieter Fox. "Voronoi Random Fields: Extracting Topological Structure of Indoor Environments via Place Labeling." IJCAI. Vol. 7. 2007.

[59] Min, Hyeun Jeong, and Nikolaos Papanikolopoulos. "The multi-robot coverage problem for optimal coordinated search with an unknown number of robots." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.

[60] Morlok, Ryan, and Maria Gini. "Dispersing robots in an unknown environment." Distributed Autonomous Robotic Systems 6. Springer Japan, 2007. 253-262.

[61] ROSWiki: Actionlib. http://wiki.ros.org/actionlib (Accessed on 11/04/2016).

[62] Nieto-Granda, Carlos, John G. Rogers, and Henrik I. Christensen. "Coordination strategies for multi-robot exploration and mapping." The International Journal of Robotics Research (2014): 0278364913515309.

[63] ROSWiki: Hokuyo. http://wiki.ros.org/hokuyo_node (Accessed on 11/04/2016).

APPENDICES

Appendix A: RESULT DATA

This appendix section contains the results data obtained from the experiments performed in a tabular format.

Two-Robot System (Hallway)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Remember Based	Hallway	1	175	54	0	54	35	19
Remember Based	Hallway	2	143	39	0	39	27	12
Remember Based	Hallway	3	178	48	0	48	22	26
Remember Based	Hallway	4	185	59	0	59	36	23
Remember Based	Hallway	5	177	47	0	47	21	26
Remember Based	Hallway	6	139	41	0	41	26	15
Remember Based	Hallway	7	161	53	0	53	34	19
Remember Based	Hallway	8	147	36	0	36	23	13
Remember Based	Hallway	9	152	49	0	49	30	19
Remember Based	Hallway	10	148	50	0	50	22	28

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Rank Based	Hallway	1	168	52	18	31	36	16
Rank Based	Hallway	2	138	39	8	26	26	13
Rank Based	Hallway	3	200	58	17	33	27	31
Rank Based	Hallway	4	151	41	9	24	23	18
Rank Based	Hallway	5	145	37	8	24	26	11
Rank Based	Hallway	6	219	46	14	27	36	10
Rank Based	Hallway	7	132	35	6	24	23	11
Rank Based	Hallway	8	170	57	16	31	38	19
Rank Based	Hallway	9	166	49	9	29	34	15
Rank Based	Hallway	10	171	50	14	31	34	16

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Threshold Based	Hallway	1	206	27	11	24	20	7
Threshold Based	Hallway	2	183	23	3	20	10	13
Threshold Based	Hallway	3	285	42	15	22	22	22
Threshold Based	Hallway	4	176	23	8	15	5	18
Threshold Based	Hallway	5	191	27	7	20	14	13
Threshold Based	Hallway	6	197	18	6	12	6	12
Threshold Based	Hallway	7	282	41	12	24	7	34
Threshold Based	Hallway	8	234	19	7	12	8	11
Threshold Based	Hallway	9	284	24	8	18	10	14
Threshold Based	Hallway	10	209	16	5	11	3	13

Two-Robot System (Asymmetric Room)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Remember Based	Room	1	181	54	0	54	41	13
Remember Based	Room	2	237	69	0	69	46	23
Remember Based	Room	3	213	53	0	53	23	30
Remember Based	Room	4	235	57	0	57	9	48
Remember Based	Room	5	187	49	0	49	25	24
Remember Based	Room	6	271	61	0	61	38	23
Remember Based	Room	7	245	55	0	55	37	18
Remember Based	Room	8	258	58	0	58	28	30
Remember Based	Room	9	216	42	0	42	23	19
Remember Based	Room	10	246	66	0	66	36	30

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Rank Based	Room	1	321	76	16	60	32	44
Rank Based	Room	2	164	41	14	27	28	13
Rank Based	Room	3	299	68	19	49	14	54
Rank Based	Room	4	254	59	12	47	42	17
Rank Based	Room	5	280	61	19	42	16	45
Rank Based	Room	6	260	60	14	46	27	33
Rank Based	Room	7	187	53	16	37	18	35
Rank Based	Room	8	258	67	22	45	48	19
Rank Based	Room	9	196	54	15	39	22	32
Rank Based	Room	10	272	61	19	42	42	19

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Threshold Based	Room	1	324	44	7	37	11	33
Threshold Based	Room	2	359	50	14	36	13	37
Threshold Based	Room	3	196	32	8	24	17	15
Threshold Based	Room	4	209	34	7	27	10	24
Threshold Based	Room	5	206	28	8	20	21	7
Threshold Based	Room	6	352	43	11	32	15	28
Threshold Based	Room	7	381	55	19	36	13	42
Threshold Based	Room	8	273	38	10	28	9	29
Threshold Based	Room	9	247	33	9	24	26	7
Threshold Based	Room	10	322	52	19	33	37	15

Two-Robot System (Office)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Remember Based	Office	1	241	60	0	60	38	22
Remember Based	Office	2	220	48	0	48	29	19
Remember Based	Office	3	188	53	0	53	36	17
Remember Based	Office	4	336	41	0	41	23	18
Remember Based	Office	5	218	46	0	46	27	19
Remember Based	Office	6	244	64	0	64	34	30
Remember Based	Office	7	172	43	0	43	16	27
Remember Based	Office	8	262	46	0	46	25	21
Remember Based	Office	9	235	66	0	66	53	13
Remember Based	Office	10	187	46	0	46	29	17

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Rank Based	Office	1	354	77	20	57	49	28
Rank Based	Office	2	178	40	11	29	22	18
Rank Based	Office	3	336	71	28	43	43	28
Rank Based	Office	4	228	62	15	47	31	31
Rank Based	Office	5	269	68	16	52	43	25
Rank Based	Office	6	342	75	34	41	37	38
Rank Based	Office	7	321	78	28	50	55	23
Rank Based	Office	8	188	56	17	39	25	31
Rank Based	Office	9	191	30	8	22	19	11
Rank Based	Office	10	234	57	16	41	21	36

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Threshold Based	Office	1	198	28	4	24	9	19
Threshold Based	Office	2	293	41	10	31	24	17
Threshold Based	Office	3	342	53	21	32	37	16
Threshold Based	Office	4	267	34	9	25	12	22
Threshold Based	Office	5	290	33	10	23	20	13
Threshold Based	Office	6	360	50	15	35	43	7
Threshold Based	Office	7	389	59	18	41	19	40
Threshold Based	Office	8	274	38	11	27	26	12
Threshold Based	Office	9	429	68	26	42	31	37
Threshold Based	Office	10	306	44	16	28	23	21

Three-Robot System (Hallway)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Remember Based	Hallway	1	147	59	0	59	32	16	11
Remember Based	Hallway	2	161	55	0	55	22	14	19
Remember Based	Hallway	3	147	40	0	40	29	7	4
Remember Based	Hallway	4	170	58	0	58	27	18	13
Remember Based	Hallway	5	184	43	0	43	31	6	6
Remember Based	Hallway	6	158	41	0	41	30	7	4
Remember Based	Hallway	7	132	35	0	35	23	5	7
Remember Based	Hallway	8	141	38	0	38	25	7	6
Remember Based	Hallway	9	146	37	0	37	23	7	7
Remember Based	Hallway	10	164	48	0	48	27	18	3

Algorithm	Environment	Iteration	Run Time (seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Rank Based	Hallway	1	146	37	11	26	25	5	7
Rank Based	Hallway	2	165	47	15	32	34	10	3
Rank Based	Hallway	3	154	40	15	25	29	4	7
Rank Based	Hallway	4	166	56	16	40	36	11	9
Rank Based	Hallway	5	186	60	19	41	25	7	28
Rank Based	Hallway	6	159	46	12	34	26	6	13
Rank Based	Hallway	7	151	39	15	24	33	4	2
Rank Based	Hallway	8	143	33	12	21	23	3	7
Rank Based	Hallway	9	159	39	10	29	26	8	5
Rank Based	Hallway	10	162	35	10	25	23	5	7

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Threshold Based	Hallway	1	166	19	3	16	14	2	3
Threshold Based	Hallway	2	174	23	9	14	24	3	2
Threshold Based	Hallway	3	227	30	14	16	10	1	19
Threshold Based	Hallway	4	158	24	8	16	9	11	4
Threshold Based	Hallway	5	165	25	5	20	12	6	7
Threshold Based	Hallway	6	286	55	25	30	8	14	23
Threshold Based	Hallway	7	165	20	6	14	10	4	6
Threshold Based	Hallway	8	173	21	9	12	10	5	6
Threshold Based	Hallway	9	211	34	12	22	12	3	19
Threshold Based	Hallway	10	174	32	11	21	24	6	2

Three-Robot System (Room)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Remember Based	Room	1	196	53	0	53	26	16	11
Remember Based	Room	2	246	65	0	65	28	23	14
Remember Based	Room	3	188	52	0	52	21	12	19
Remember Based	Room	4	203	43	0	43	19	11	13
Remember Based	Room	5	214	56	0	56	25	18	13
Remember Based	Room	6	184	49	0	49	16	21	12
Remember Based	Room	7	206	57	0	57	21	19	17
Remember Based	Room	8	196	49	0	49	26	14	9
Remember Based	Room	9	287	61	0	61	31	17	13
Remember Based	Room	10	234	35	0	35	16	10	9

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Rank Based	Room	1	252	38	17	21	21	7	10
Rank Based	Room	2	238	57	15	42	21	30	6
Rank Based	Room	3	142	42	14	28	23	12	7
Rank Based	Room	4	178	37	12	25	24	7	6
Rank Based	Room	5	167	42	9	33	22	12	8
Rank Based	Room	6	303	57	21	36	42	8	7
Rank Based	Room	7	227	46	14	32	31	10	5
Rank Based	Room	8	187	35	8	27	16	12	7
Rank Based	Room	9	309	59	24	35	10	41	8
Rank Based	Room	10	263	61	19	42	19	26	16

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Threshold Based	Room	1	183	15	3	12	4	8	3
Threshold Based	Room	2	175	20	4	16	4	12	4
Threshold Based	Room	3	310	46	15	31	32	11	3
Threshold Based	Room	4	203	23	9	14	6	8	9
Threshold Based	Room	5	185	24	3	21	2	7	15
Threshold Based	Room	6	325	43	19	24	33	8	2
Threshold Based	Room	7	314	52	22	30	7	13	32
Threshold Based	Room	8	202	23	6	17	2	16	5
Threshold Based	Room	9	251	39	9	30	18	9	12
Threshold Based	Room	10	339	48	18	30	37	6	5

Three-Robot System (Office)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Remember Based	Office	1	181	49	0	49	27	13	9
Remember Based	Office	2	219	34	0	34	24	5	5
Remember Based	Office	3	234	70	0	70	25	20	25
Remember Based	Office	4	175	46	0	46	40	3	3
Remember Based	Office	5	183	37	0	37	31	2	4
Remember Based	Office	6	216	41	0	41	31	5	5
Remember Based	Office	7	288	48	0	48	24	11	13
Remember Based	Office	8	194	64	0	64	25	23	16
Remember Based	Office	9	291	40	0	40	31	6	3
Remember Based	Office	10	244	70	0	70	19	10	41

Algorithm	Environment	Iteration	Run Time (seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Rank Based	Office	1	191	21	7	14	17	2	2
Rank Based	Office	2	215	39	8	31	13	16	10
Rank Based	Office	3	321	52	23	29	12	38	2
Rank Based	Office	4	362	96	37	59	14	21	61
Rank Based	Office	5	257	19	5	14	12	2	5
Rank Based	Office	6	214	32	10	22	12	10	10
Rank Based	Office	7	304	47	13	34	21	4	22
Rank Based	Office	8	245	44	14	30	15	28	1
Rank Based	Office	9	192	28	7	21	17	5	6
Rank Based	Office	10	237	25	11	14	12	9	4

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count	Robot 3 Count
Threshold Based	Office	1	212	17	2	15	11	5	1
Threshold Based	Office	2	276	50	19	31	5	43	2
Threshold Based	Office	3	292	59	17	42	8	10	41
Threshold Based	Office	4	289	52	21	31	10	8	34
Threshold Based	Office	5	307	78	32	46	23	20	35
Threshold Based	Office	6	230	22	5	17	15	4	3
Threshold Based	Office	7	315	80	29	51	15	23	42
Threshold Based	Office	8	276	17	3	14	7	5	5
Threshold Based	Office	9	282	57	16	41	15	28	14
Threshold Based	Office	10	297	54	16	38	14	1	39

Two-Robot System with Distant Starts (Hallway)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Case Based Allocation	Hallway	1	137	32	0	32	19	13
Case Based Allocation	Hallway	2	169	41	0	41	31	10
Case Based Allocation	Hallway	3	140	38	0	38	29	9
Case Based Allocation	Hallway	4	161	43	0	43	38	5
Case Based Allocation	Hallway	5	173	47	0	47	27	20
Case Based Allocation	Hallway	6	158	46	0	46	31	15
Case Based Allocation	Hallway	7	159	38	0	38	29	9
Case Based Allocation	Hallway	8	186	51	0	51	33	18
Case Based Allocation	Hallway	9	152	36	0	36	24	12
Case Based Allocation	Hallway	10	177	49	0	49	31	18

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Rank Based	Hallway	1	224	51	13	33	41	10
Rank Based	Hallway	2	149	40	9	27	32	8
Rank Based	Hallway	3	202	59	18	36	42	17
Rank Based	Hallway	4	140	40	6	30	27	13
Rank Based	Hallway	5	136	25	5	20	21	4
Rank Based	Hallway	6	173	46	12	29	36	10
Rank Based	Hallway	7	167	44	11	30	34	10
Rank Based	Hallway	8	166	38	13	25	31	7
Rank Based	Hallway	9	148	43	8	32	29	14
Rank Based	Hallway	10	139	28	7	21	19	9

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Threshold Based	Hallway	1	276	39	19	24	26	13
Threshold Based	Hallway	2	141	22	4	18	20	2
Threshold Based	Hallway	3	186	21	5	16	17	4
Threshold Based	Hallway	4	219	34	8	25	25	9
Threshold Based	Hallway	5	202	31	10	21	24	7
Threshold Based	Hallway	6	195	31	12	19	27	4
Threshold Based	Hallway	7	235	33	11	26	27	6
Threshold Based	Hallway	8	199	28	8	20	19	9
Threshold Based	Hallway	9	206	24	6	18	19	5
Threshold Based	Hallway	10	241	29	4	23	25	4

Two-Robot System with Distant Starts (Room)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Remember Based	Room	1	218	43	0	43	34	9
Remember Based	Room	2	220	42	0	42	27	15
Remember Based	Room	3	184	49	0	49	33	16
Remember Based	Room	4	196	32	0	32	15	17
Remember Based	Room	5	167	46	0	46	31	15
Remember Based	Room	6	174	69	0	69	44	25
Remember Based	Room	7	218	51	0	51	18	33
Remember Based	Room	8	173	41	0	41	26	15
Remember Based	Room	9	210	57	0	57	37	20
Remember Based	Room	10	187	66	0	66	37	29

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Rank Based	Room	1	183	45	6	39	39	12
Rank Based	Room	2	227	57	9	48	44	13
Rank Based	Room	3	209	71	13	58	32	26
Rank Based	Room	4	181	40	8	32	26	14
Rank Based	Room	5	278	63	21	42	47	16
Rank Based	Room	6	253	60	18	42	44	16
Rank Based	Room	7	158	34	5	29	26	8
Rank Based	Room	8	236	62	15	47	41	21
Rank Based	Room	9	161	29	5	24	23	6
Rank Based	Room	10	182	32	8	24	19	13

Algorithm	Environment	Iteration	Run Time (seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Threshold Based	Room	1	172	23	7	16	16	7
Threshold Based	Room	2	207	38	6	32	29	9
Threshold Based	Room	3	290	37	11	26	24	13
Threshold Based	Room	4	280	33	8	25	21	12
Threshold Based	Room	5	175	26	6	20	16	10
Threshold Based	Room	6	211	37	9	28	25	12
Threshold Based	Room	7	329	40	8	32	29	11
Threshold Based	Room	8	160	20	5	15	16	4
Threshold Based	Room	9	354	50	16	34	38	12
Threshold Based	Room	10	260	46	15	31	27	19

Two-Robot System with Distant Starts (Office)

Algorithm	Map	Iteration	Run Time (in seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Remember Based	Office	1	296	72	0	72	48	24
Remember Based	Office	2	245	41	0	41	19	22
Remember Based	Office	3	253	55	0	55	32	23
Remember Based	Office	4	214	57	0	57	34	23
Remember Based	Office	5	189	76	0	76	48	28
Remember Based	Office	6	281	77	0	77	43	34
Remember Based	Office	7	174	47	0	47	34	13
Remember Based	Office	8	200	36	0	36	21	15
Remember Based	Office	9	253	67	0	67	48	19
Remember Based	Office	10	288	37	0	37	25	12

Algorithm	Environment	Iteration	Run Time (seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Rank Based	Office	1	297	61	20	41	48	13
Rank Based	Office	2	200	39	12	27	23	16
Rank Based	Office	3	203	59	16	43	38	21
Rank Based	Office	4	260	46	8	38	40	6
Rank Based	Office	5	209	35	7	28	27	7
Rank Based	Office	6	226	41	10	31	32	9
Rank Based	Office	7	383	75	20	55	65	10
Rank Based	Office	8	369	77	21	56	35	42
Rank Based	Office	9	307	46	16	30	29	17
Rank Based	Office	10	409	74	30	44	58	16

Algorithm	Environment	Iteration	Run Time (seconds)	Explored Frontiers	Duplicate Frontiers	Unique Frontiers	Robot 1 Count	Robot 2 Count
Threshold Based	Office	1	243	28	6	22	16	6
Threshold Based	Office	2	231	25	8	17	20	5
Threshold Based	Office	3	296	35	10	25	19	16
Threshold Based	Office	4	304	37	13	24	26	11
Threshold Based	Office	5	256	41	11	30	29	12
Threshold Based	Office	6	247	39	8	31	25	14
Threshold Based	Office	7	343	54	18	36	31	23
Threshold Based	Office	8	289	45	12	33	32	13
Threshold Based	Office	9	301	40	16	24	28	12
Threshold Based	Office	10	309	31	9	22	21	9

VITA AUCTORIS

NAME:	Sushil Parti
PLACE OF BIRTH:	Pathankot, India
YEAR OF BIRTH:	1990
EDUCATION:	Lovely Professional University, Punjab, India Bachelor of Technology, Computer Science 2007-2011
	University of Windsor, Windsor ON, Canada Master of Science, Computer Science 2014-2016