NAVIGATION PLANNING FOR A MULTI ROBOT SYSTEM EXPLORING AN UNKNOWN ENVIRONMENT SUPPORTED BY VOLUMETRIC DATA

R.M.K.V. Ratnayake

208005L

Degree of Master of Science

Department of Computer Science & Engineering

University of Moratuwa
Sri Lanka

October 2021

NAVIGATION PLANNING FOR A MULTI ROBOT SYSTEM EXPLORING AN UNKNOWN ENVIRONMENT SUPPORTED BY VOLUMETRIC DATA

R.M.K.V. Ratnayake

208005L

Thesis submitted in partial fulfillment of the requirements for the degree

Master of Science in Computer Science and Engineering

Department of Computer Science & Engineering

University of Moratuwa

Sri Lanka

October 2021

DECLARATION

I, Kalana Ratnayake, declare that this is my own work and this dissertation does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning, and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: [illegible] Date: 28/01/2022

The above candidate has carried out research for the Masters thesis/dissertation under my supervision.

Name of Supervisor: Dr. Chandana Gamage

Signature of the Supervisor: [illegible] Date: 28/01/2022

Name of Supervisor: Dr. Sulochana Sooriyaarachchi

Signature of the Supervisor: [illegible] Date: 29/01/2022

ABSTRACT

Exploration and navigation in unknown environments can be done individually or as a group of robots. The current state-of-the-art systems mainly use frontier detection-based exploration approaches based on occupancy grids and are available as either single robot systems or multi-robot systems.

In this research, we propose a two-stage octomap-based exploration system for multi-robot systems that improve multi-robot coordinated exploration. We also present a prototype robotic system capable of exploring an unmapped area individually or while coordinating with other robots to complete the exploration fast and efficiently. During single robot exploration, the proposed system only uses the first stage of the two-stage system to evaluate the octomap of the environment. This stage utilizes the state of voxels to calculate target locations for navigation using a distance-based cost function. During multi-robot exploration, the proposed system uses both stages of the two-stage system to explore the given area. The second stage uses maps created by individual robots to create a merged map. The merged map can be used to evaluate the environment using octomaps to identify target locations for exploration and navigation.

We have also proposed a performance evaluation criterion for exploration systems considering the robot's operation time, power consumption, and stability. This criterion was used to evaluate the system and compare the performance of the individual robot system against the multi-robot system as well as against the state-of-the-art Explore-Lite system. Results of experiments show that the individual robot system proposed in this paper is about 38% faster than the Explore-Lite system, the multi-robot system using two robots is 48% faster than the individual robot system, and the multi-robot system using three robots is 38% faster than the individual robot system.

Keywords: multi-robot system; exploration; path planning; navigation; octomap based exploration; unstructured environment;

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisors Dr. Chandana Gamage and Dr. Sulochana Sooriyaarachchi for the support given for the success of this research. I'm incredibly grateful for their guidance throughout this research.

I want to thank both the academic and non-academic staff of the Department of Computer Science and Engineering for their help during this research in terms of administrative matters and in providing necessary resources.

I would also like to thank my family for all the love and support given throughout this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SLAM	Simultaneous Localization And Mapping
ROS	Robot Operating System
ORB	Oriented FAST and rotated BRIEF
PRM	Place Recognition Module
ICP	Iterative Closest Point
SD	Standard Deviation
2D	Two Dimensional
3D	Three Dimensional
BFS	Breadth-First Search
BGS	Basic Goal Seeking
MGS	Modified Goal Seeking
GSI	Goal Seeking Index
LIDAR	Light Detection and Ranging
EKF	Extended Kalman Filter
RGB-D	Red Green Blue - Depth

Chapter 1

INTRODUCTION

1.1 Background

Robots are more suited to navigate, explore and map regions such as disaster-struck regions, underground tunnel systems, underground cave systems, and underwater trenches than humans due to their robustness, teleoperability, and reproducibility compared to humans. Figure 1.1a shows a quadruped robot exploring an underground cave system and Figure 1.1b shows a swarm of drones entering a collapsed building for exploration. Usually, a single robot is operated by a single operator or a team of operators [1] when navigating, exploring, or mapping in the above applications. This operation approach usually limits the parallel operations to one or two robots in this kind of situation. However, we can improve the speed and efficiency of task completion of robots by introducing autonomous navigability and coordination with each other.

(a) A single robot exploring a cave; Source :[2]

(b) A swarm of drones exploring a collapsed building; Source:[3]

Figure 1.1: Robot systems used for exploration

There are various approaches for autonomous navigation, such as map-less direct navigation [4], creation of navigability maps [5], and creation of occupancy grids [6] [7] [8]. Among these, occupancy grid-based navigation is one of the most practiced approaches.

Exploration also has been developed through various approaches, and frontier-based exploration is one of the popular methods. This approach uses occupancy grids created for navigation to identify unexplored regions. Frontier-based algorithms are geared towards free exploration and fail when the requirement is to explore and map a predefined open space area such as an outdoor disaster site.

This research proposes a multi-robot system that can be extended freely by adding more robots without modifying the internal structure of the server. This system utilizes 3D representations of the environment for navigation and coordinated exploration. The system has been developed as the second phase of a disaster mapping robotic system currently being designed and developed in the IntelliSense Lab of the Department of Computer Science and Engineering. We developed a single robot system as the first stage [9] to explore an unknown environment specified by boundaries. This research builds on top of the single robot [9] to create the multi-robot system focusing on controlling multiple robots.

The proposed system in this thesis comprises two major system components. The server system implements the map merging and coordination, whereas the robot system implements a standalone exploration system that accepts exploration goals from the server system.

1.2 Research Problem

The purpose of this research is to create a coordinated exploration and mapping mechanism capable of guiding the robots such that the robots can explore and map an unstructured unknown area with minimum overlap while increasing the speed of exploration and efficiency compared to a single robot system. Accordingly, the research problem is to,

"Creating a multi-robot system capable of fast coordinated exploration and mapping with minimum region overlapping between robots."

1.3 Research Objectives

Objectives of this research are as follows:

1. Design and development of a single robot system that can accept exploration goals from outside and configure its internal parameters to support the new task.

2. Design and development of a server system that can accept a variable number of robots and guide them to explore a given area.

3. Develop, test, and evaluate several instances of multi-robot systems with a different number of robots.

1.4 Research Contributions

The research contribution can be identified as five outcomes:

- A two-stage octomap-based exploration system for multi-robot systems. The first stage resides on the robot and guides the robot to explore the given area. The second stage resides on the server and coordinates the robots to explore a given area as a multi-robot system

- An exploration module that uses volumetric data to evaluate and explore a given target area

- An server-based exploration system that uses volumetric data to evaluate and guide robots to explore a given area.

- An autonomous multi-robot exploration system prototype that can explore a given area as a robot cluster. We have implemented this system using ROS and have shown that it performs as expected using Gazebo Physics Simulator.

- A performance evaluation criteria for single and multi-robot systems, which considers the time, energy, and stability of the robot.

1.5 Publications

K. Ratnayake, S. Sooriyaarachchi and C. Gamage, "OENS: An Octomap Based Exploration and Navigation System," 2021 5th International Conference on Robotics and Automation Sciences (ICRAS), 2021, pp. 230-234, doi: 10.1109/ ICRAS-52289.2021.9476592.

Received Excellent Oral Presentation of the session award

Scopus Indexed

1.6 Outline of Thesis

The rest of the thesis is organized as follows. Chapter 2 presents a review of literature on navigation, coordination, exploration, and mapping. Chapter 3 presents the research methodology along with the techniques and systems discovered during the literature review on navigation, coordination, exploration, and mapping. Chapter 4 presents the design of the two-stage octomap based exploration system. Chapter 5 presents the experiments that were performed, the results that were collected, and the evaluation of the results. The last chapter contains the conclusions and planned future work.

Chapter 2

LITERATURE SURVEY

Navigation, coordination, exploration, and mapping are four main branches of research related to robotics. Navigation systems focus on moving the robot from one position to another, and exploration systems identify areas the robot has to explore. Mapping systems focus on creating maps of the environment so that the robot can visualize that environment and utilize them for navigation and exploration. Coordination systems focus on inter-robot interaction in multi-robot systems. In the following sections, we will review literature in these four areas of research.

2.1 Navigation

We can divide robotic systems into three different categories: teleoperated, semi-autonomous, and autonomous, depending on the utilized navigation approach. An operator completely controls teleoperated robots while only setting waypoints for semi-autonomous robots indicating the navigation path it should follow. Fully autonomous robots navigate without the guidance of an operator.

Firefighters have used Semi-autonomous robots to map disaster zones [1] in which the robot utilizes navigation points selected by a firefighter to calculate a path while using a 3D laser scanner to create a map of the environment. This system also supports the continuation of mapping from a previously stopped location.

2.1.1 Environment Perception

Autonomous navigation systems need to be able to perceive and store the state of the environment in order to calculate a path in it. Among the methods available to store the state of the environment, a point cloud is one of the widely used methods that enables the users to view and process the environment in a three-dimensional perspective. Figure 2.1 shows one instance of the point cloud. An autonomous navigation system can utilize a mapping subsystem to create point clouds using depth cameras, stereo cameras, or LIDAR sensors.

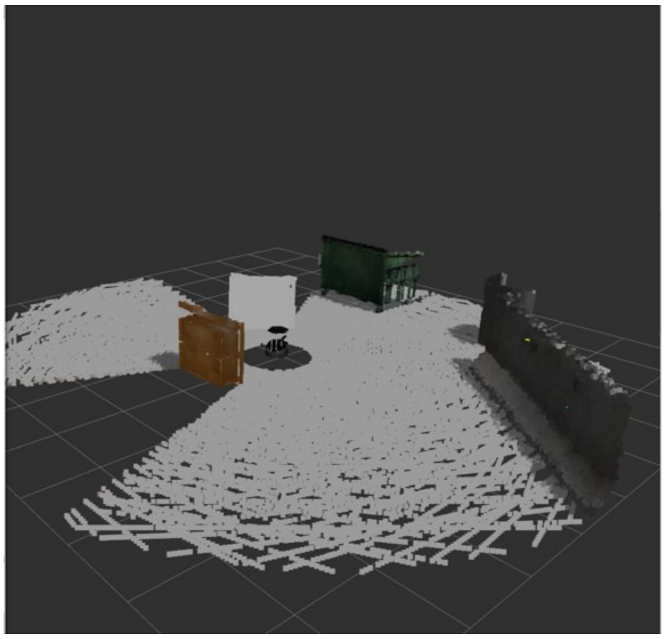


Figure 2.1: Viewing environment through a point cloud

Occupancy grids [10] are a medium to analyze the environment captured as a point cloud probabilistically. These grids discretize the environment into a two-dimensional grid consisting of cells and give each cell a probability of being occupied. The cells can have three primary states as free, occupied, and unknown. Path planning calculations based on occupancy grids generally consider cells with the occupied state as obstacles.

The Octomap framework allows the generation of volumetric 3D environmental models [11] based on octrees and uses probabilistic occupancy estimation. Similar to occupancy grids, octomap voxels also have three occupancy states called free, occupied, and unknown. Figure 2.2 shows the corresponding octomap for the point cloud shown in Figure 2.1.

2.1.2 Path Planning

The artificial potential field concept [12] is an approach that can be used on occupancy grids to calculate a path. This approach calculates virtual repellent

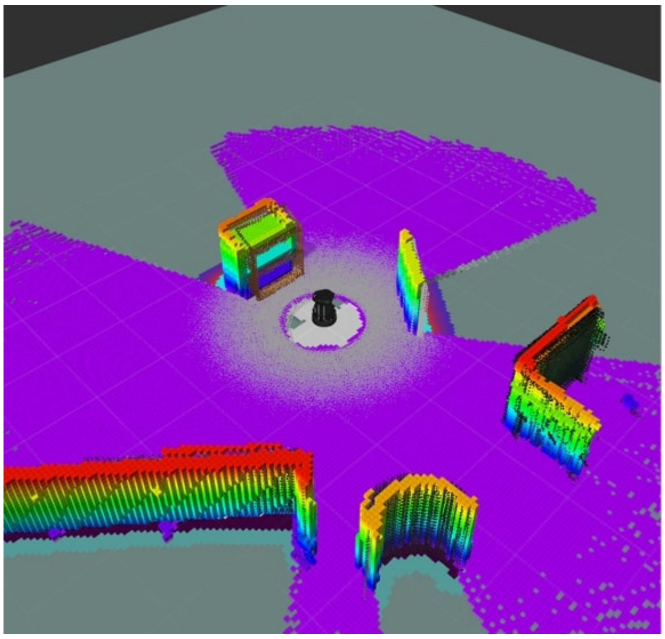


Figure 2.2: Viewing environment through an octomap

forces between the robot and obstacles and virtual attraction force between the robot body and the goal and then uses these attraction and repellent forces to calculate the lowest potential route between the robot and the goal.

As mentioned, a navigation system can use occupancy grids calculated using point clouds to interpret static environments. However, additional processing steps such as comparing occupancy grids taken at different points in time are required to interpret the dynamic environments and identify dynamic obstacles [7]. Identification of dynamic obstacles and prediction of their paths can also be achieved if the kinematic information of the robot and kinematic information of obstacles are known [8]. This information can be inserted into a new occupancy grid and used for navigation planning along with the occupancy grid with static obstacles.

Maier et al. have used octomaps to navigate a biped robot [13] in an enclosed space. First, they have created an octomap of the enclosed space. The octomap was then used to create an occupancy grid, which was used to navigate a bipedal robot without colliding with obstacles in the room. Octomap framework has also been used for motion planning of robot arm manipulators [14]. The motion planner used a multi-layered 2D occupancy grid extracted from an octomap to interpret the environment. The height differences between the occupancy layers were decided according to the shape and movement capabilities of the robot. Octomaps can also be used to identify and overcome slopes and staircases [15] by slicing the octomap at different heights to create several occupancy grids and then using those to create a traversability map that considers slopes as traversable areas. Rapidly exploring Random Tree algorithm can be used to calculate a path on the traversability map.

Schmid et al. have used flying robots to map indoor and outdoor buildings [16] with a 'waypoint following' approach. This system uses octomaps to identify and avoid 3D obstacles in the environment while following the path set using waypoints.

Path planning can also be done without using occupancy grids or octomaps. In such scenarios, one approach to identify traversable floor areas is the use of edge detection [4]. Another approach is to use a depth image to identify obstacles and floors and use a color image to identify roads [17]. The floor area the robot can navigate can then be separated and identified using margin lines, which are the lines that separate obstacles and ground and roads.

2.1.3 Summary

Table 2.1 includes a summary of methods available for environment modelling.

Table 2.1: Summary of environment perception approaches

Approach	Data structure	Primary data	Secondary Data	Ref.
Point Cloud	1D Array	XYZ coordinates	RGB values	-
Occupancy Grid	2D Array	Probability of occupancy	-	[10]
Octomap	Octree	Probability of occupancy	RGB values	[11]

Table 2.2 includes a summary of methods available for general path planning.

Table 2.2: Summary of path planning approaches

Approach	Input Data	Remarks	Ref.
Artificial potential Field	Occupancy grid	Calculate virtual forces between robot, obstacle and goal and follow lowest potential route	[12]
Comparing occupancy grids	Occupancy grid	Identify dynamic obstacles in the map by comparing occupancy grids with a time interval	[7]
Overlay kinematic information on occupancy grid	Occupancy grid	Insert kinematic information of robot and obstacles into a new cost map layer and path plan with obstacle avoidance	[8]
Occupancy grid for biped robot navigation	Octomap	Generate an occupancy grid from the data available in the octomap and biped robot constraints	[13]
Multi-layered occupancy grids for arm manipulation	Octomap	Generate multiple occupancy grids considering the shape and movement capabilities of robot arm	[14]
Traversability Map	Octomap	Slice octomap at different heights and calculate several occupancy grids and combine them to create the traversability map which includes stairs and steps	[15]
Obstacle avoidance	Octomap	Directly use octomaps to identify and avoid obstacles in the path	[16]

Separate traversable floor	Point cloud	Separate floor and use edge detection to identify traversable area.	[4]
Identify floor using sightlines	Color and Depth Images	Use depth image to identify floor and obstacles, color image to identify roads. Utilize sightlines and identified data to identify traversable area	[17]

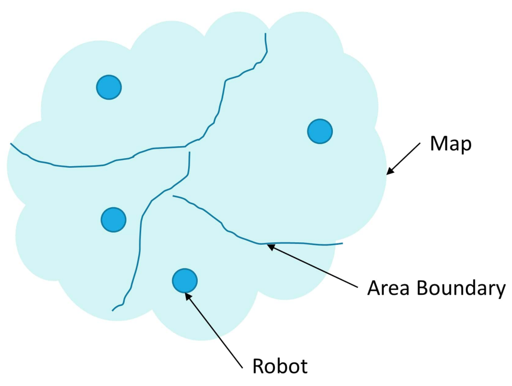
2.2 Coordination

Coordination between robots in a multi-robot system can be achieved in several different ways. In this section, we mainly consider two approaches. One approach is to divide the area into several regions and allocate them to robots in the system. Another approach is to divide robots into teams, explore the area, and assign different exploration targets during operation.

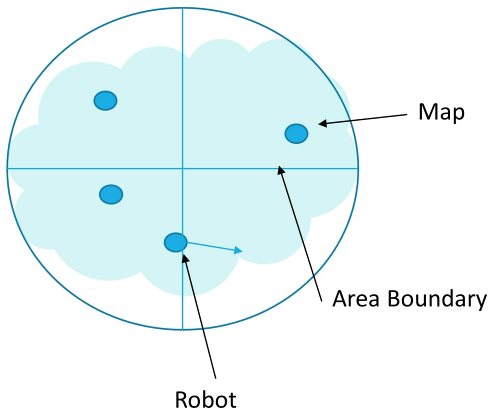
2.2.1 Division of Map

The division of the map approach focuses on reducing the overlap in the maps created by the robots. One implementation named "zone partitioning" shown in Figure 2.3a divides the map into partitions depending on the robot's position and assigns each partition to a robot [18]. The robot can then identify unexplored cells in the assigned zone, cluster them and use the clusters to perform the exploration. Once the map is updated, zones need to be re-calculated and assigned depending on the new positions of the robots. Compared to using zone partitioning, which divides area based on the position, the "circle partitioning" approach shown in Figure 2.3b divides the map into fixed equal-sized segments [19] based on the number of robots. It assumes that the center of the map is the center of a circle and calculates a virtual circle that encompasses the whole map. Then the circle is divided into partitions such that each robot can be assigned a single partition.

The robot can then identify unexplored cells in the assigned zone and explore similarly to the zone partitioning approach.



(a) Zone partition approach



(b) Circle partition approach

Figure 2.3: Division of map approaches

2.2.2 Association of Robots

The association of robots approach focuses on grouping robots into teams and assigning them exploration targets during the operation. Divide and Conquer, Buddy and Reserve are three different implementations of the association of robots approach [20]. In the *Reserve* implementation, initially, only one robot of the fleet conducts the exploration. When that robot detects more than one exploration goal, resulting in a branching point, new robots from the fleet are assigned to the new exploration goals. This implementation initially has a high idling time since only one robot of the fleet is exploring but robots branching out enables a lesser overlap related to exploration and mapping. In the *Divide and Conquer* implementation, all robots start exploration at once. When a branching point is identified, the fleet is divided into separate groups such that each group can explore a single branch. Compared to the reserve system, this has a low idling time but has a high overlap related to exploration and mapping until a large number of branches are discovered such that each robot is assigned a single branch. The *Buddy* implementation divides the fleet into small groups of two robots each. Exploration starts with one group of robots exploring while others are waiting at the starting position. Once the starting group reaches a branch, they divide the group and follow the branching goals individually. If one of the single robots discovers new branching points, those are assigned to a new robot group at the starting position. This implementation reduces the idle time compared to the *Reserve* implementation but still results in more idle time than the *Divide and Conquer* implementation. When considering overlap related to exploration and mapping, the *Buddy* implementation has less overlap than *Divide and Conquer* implementation and more than *Reserve* implementation.

Another implementation is to allow robots to map individually while avoiding overlap as best as possible. In such implementations, map-based calculations that need all the maps from all the robots can occur on a separate system that runs on a robot or a separate computer. One such system is a centralized system where maps from each robot are collected and then merged based on the robot's initial position [21]. All robots share their sensor logs to reduce the overlap between maps even under communication failures in this system. This sharing of sensor logs allows all the robots to identify regions where other robots have visited until the communication failure, enabling them to ignore those regions and reduce re-exploration during a communication failure.

2.2.3 Summary

Table 2.3 includes a summary of methods available for robot coordination.

Table 2.3: Summary of robot coordination approaches

Implementation	Remarks	Ref.
Zone Partition	Divide the map into partitions considering the positions of robots and number of robots and assign each robot a partition	[18]
Circle Partition	Imagine a circle on the map with centers overlapping, divide the circle into segments and assign each segment to a robot	[19]

Reserve	One robot of the group explore until it reaches a branching point, then informs other robots idling at the starting position to explore other branching paths	[20]
Divide and conquer	All robots of the group explore at once and divide into sub groups and explore separate ways at a branching point	[20]
Buddy	Two robots of the group explore as a team until it reaches a branching point, at which point they divide and explore. when the individual robots meet a new branch they inform robots idling at the starting point to explore	[20]
Sensor log sharing	Robots map while sharing the maps with a server that handles map based calculations. Sensor logs are also shared to avoid revisiting a mapped area during a communication failure	[21]

2.3 Exploration

In this section, we consider exploration systems under two categories, exploration systems used in single robot systems and exploration systems used in multi-robot systems.

2.3.1 Single Robot Systems

Occupancy grids used for navigation calculations can also be used for exploration with the concept of frontier-based exploration [22]. Yamauchi first proposed frontier-based exploration for evidence grids that are conceptually similar to occupancy grids. In the frontier-based exploration, the evidence grid cells are categorized as occupied, unoccupied, and unknown in the frontier-based exploration, based on their prior occupancy probability and updated occupancy probability. Then the boundary cells bordering both unoccupied and unknown cells were identified as frontier edge cells. Finally, the adjacent frontier edge cells were grouped, and if the group contained more than a predefined number of cells, it was identified as a frontier. Figure 2.4 contains an example of frontier calculation. The author selected the nearest frontier as the navigation goal for the robot, and the distance was calculated using a depth-first search on the grid. When the robot reached the frontier, it performed a 360-degree sensor sweep and updated the map. Then the new frontiers were calculated using the updated map. This process repeated until the whole accessible area was mapped.

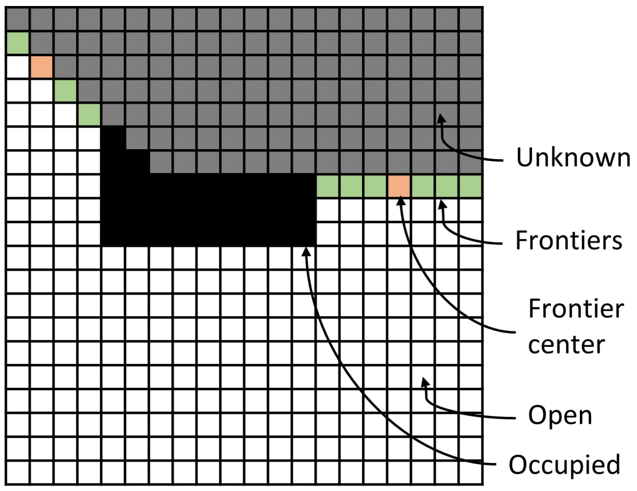


Figure 2.4: Frontiers on an occupancy grid

Basic goal Seeking algorithms [23] and Modified Goal Seeking algorithms [23] focus on using frontier-based exploration to reach a known goal region in an unknown environment as shown in Figure 2.5. In this research, exploration goal was selected from all available frontiers using a cost function named "Goal Seeking Index," which was calculated for each frontier considering the distance to frontier from the robot and the distance to goal from the frontier. Ultimately the frontier with the maximum GSI was selected as the navigational goal. This approach enabled the robot to reach the goal region without exploring all the frontiers. BGS had the issue of falling into trap-like situations when the sensing region was explored entirely, and the path to the goal region was blocked due to obstacles. This issue was solved in MGS by allowing the algorithm to select a frontier outside the sensing region when a trap-like situation was detected.

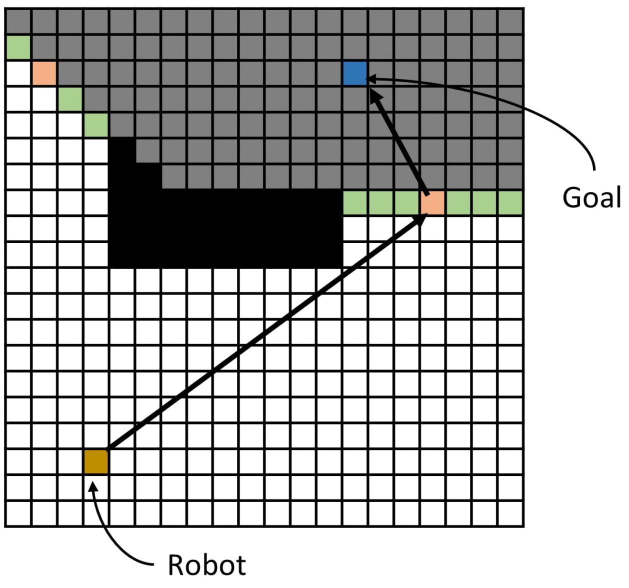


Figure 2.5: Reaching a known position in an unknown area in goal-seeking algorithms

The Wave-front Frontier Detector algorithm [24] used BFS, to search the unoccupied cells of the occupancy grid to identify frontiers as shown in 2.6. Since accessible frontier points are always adjacent to unoccupied grid cells, this enabled much faster grid processing. To further improve the frontier identification, Keidar et al. also proposed to process the raw laser scanner readings in the Fast Frontier Detector algorithm [24]. In this algorithm, they first had to sort the laser scanner readings according to angle, detect contours from the readings, then detect frontiers, and update previous frontiers. They proved that this approach was faster than using the occupancy grids.

Quick Goal Seeking algorithm [25] divides the occupancy grid into four quadrants and focuses only on exploring the quadrant with the goal region. Trap-like situations where frontiers in the selected quadrant became unreachable were handled by enabling the algorithm to select frontiers from other quadrants during such situations.

Octomap framework has been used to model large structures such as monuments and buildings [26] with a ground robot using an approach similar to the wall

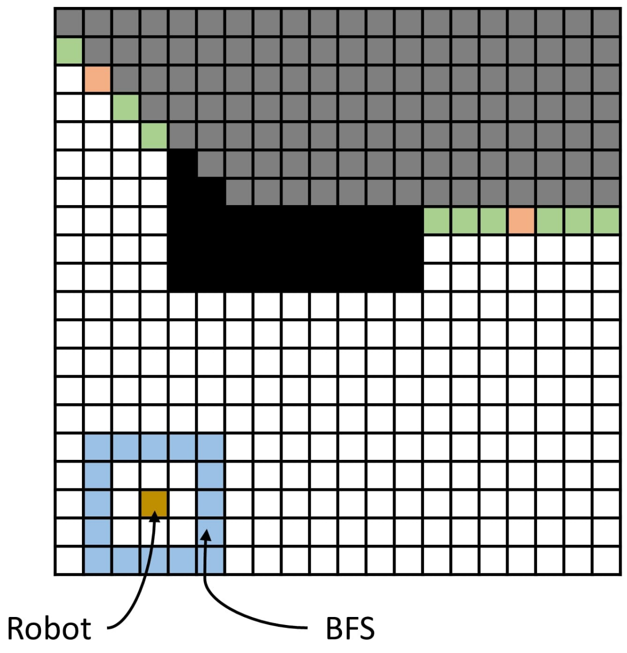


Figure 2.6: Searching the occupancy grid using BFS

following to map the structure's perimeter. If the system detected an unmapped cavity area, the robot proceeded to explore them individually.

Explore-Lite exploration system [27] is one of the most popular greedy frontier-based exploration systems readily available in ROS. This system uses occupancy grids calculated by a separate system such as a *SLAM System* to calculate frontiers and depends on the ROS Navigation Stack developed by Willow Garage for path planning and controlling of the robot.

2.3.2 Multi-Robot Systems

In the multi-robot version of frontier-based exploration, [28] each robot maintains a global evidence grid updated by local evidence grids created and shared by other robots in the system. This global map allows for decentralized implementation, which enables independent exploration and robustness to individual robot failure. Still, it also has the disadvantage of having possibilities for the robots to target the same frontiers and block others during exploration.

Frontiers can be identified from a shared occupancy grid, and they can be assigned to robots in the system using a cost function that considers the distance to each frontier from the robot and the number of other robots moving to that frontier or nearby frontier [21]. This cost function allows the robots to not select frontiers near other robots and effectively cover a large area.

Another way to assign frontiers is to divide the exploration area into zones and assign each zone to robots in the system as shown in the Figure 2.3a. The zone partitioning approach calculates zones depending on the robot's position and assigns each zone to a robot [18]. The robot then can calculate unexplored cells in the zone and perform the exploration. Once the map is updated, zones need to be re-assigned depending on the new positions. Compared to using zone partitioning, which is based on the position, the circle partitioning method shown in Figure 2.3b, divides the map into fixed segments [19] based on the number of robots. The circle is divided into partitions such that each robot can be assigned a single partition. Robots then use frontier exploration to explore assigned partition.

Communication failure during multi-robot coordination can be a significant issue that can cause the robots to explore the whole area individually. One solution is to incorporate communication signal strength into the frontier selection cost function, [29] enabling the robot to traverse areas with good signal strength.

Another approach for multi-robot exploration is to collect enough information such that the rest of the information can be accurately estimated based on that as proposed in [30]. This method is mainly useful for tasks such as temperature variation mapping that allows for data estimation. The proposed system is a centralized system where the central station auctions the exploration tasks among the robots in the network. Robots bid for tasks and, if won, follow the path and explore until enough information is collected.

2.3.3 Summary

Table 2.4 includes a summary of methods available for exploration.

Table 2.4: Summary of approaches for exploration

Approach	Category	Remarks	Ref.

Frontier-based Exploration	Single robot	Introduces the concept of frontier based exploration that uses evidence grids	[22]
Basic Goal Seeking algorithm	Single robot	Designed to reach a known position in an unknown environment. Fails when sensor region has been explored and goal is blocked by obstacles	[23]
Modified Goal Seeking algorithm	Single robot	Similar to Basic Goal Seeking Algorithm but when sensor region has been explored and goal is blocked by obstacles, picks exploration point from outside sensing region	[23]
Wave-front Frontier Detector	Single robot	Use BFS to expand free space. allows to detect frontiers faster	[24]
Fast Frontier Detector	Single robot	Process the sensor data directly without using an occupancy grid	[24]
Quick Goal Seeking algorithm	Single robot	Divide the map into quadrants and explore the quadrant with the goal. In a trap-like scenario, select explore a neighbouring quadrant	[25]
Wall following on Octomap	Single robot	explore around monuments with a wall following like approach	[26]
Explore-Lite system	Single robot	Explore using greedy frontier based exploration on occupancy grid	[27]
Frontier-based Exploration (Multi-robot)	Multi robot	Maintain a global evidence grid in each robot and use local evidence grids shared by all robots to update the global grid	[28]

Shared occupancy grid	Multi robot	Use a shared occupancy grid to identify frontiers and use a cost function to assign frontiers to each robot	[21]
Zone partition	Multi robot	Divide area among robots considering the position and assign them each sub area to explore like in single robot exploration	[18]
Circle Partition	Multi robot	Divide the area into sectors of a circle and assign each to a robot to explore like in single robot exploration	[19]
Incorporate signal strength to cost function	Multi robot	Incorporate communication signal strength to frontier selecting cost function and reduce the risk of communication failure	[29]
Collect the minimum data	Multi robot	Collect minimal data required to infer the data in the other areas	[30]

2.4 Mapping

Mapping of the environment needs to have a perfect position calculation in order to create a perfect map. However, since all mechanical systems tend to have errors or develop errors over time, most mapping systems are bundled with a localization system that uses details in the map itself to calculate the afro mentioned error and support mapping systems to create a good quality map. These two systems together are called Simultaneous Localization And Mapping systems. Our review first looks at simple single robot SLAM systems, then SLAM systems that support multi-session mapping and multi-robot mapping concepts.

2.4.1 Simple Mapping Systems

HectorSLAM [31] is a simple SLAM system that uses occupancy grids for mapping and localization, allowing for three-dimensional movement. It uses LIDAR scans, 3D EKF, and scan matching techniques to estimate the robot's position and correct it.

ORBSLAM [32] is a SLAM system that takes images from monocular cameras as input for localization and mapping, whereas ORBSLAM2 [33] is its second generation SLAM system that supports monocular, stereo, and RGB-D images for localization and mapping. Both systems extract ORB features from images it receives as input and use them to estimate the position error. ORB-SLAM uses motion-only bundle adjustment for tracking and place recognition for relocalization. With stereo and RGB-D cameras introduced in ORBSLAM2, generated depth images can be used for tracking.

2.4.2 Multi-session Mapping Systems

Multi-session mapping runs on the robot itself, and unlike a simple SLAM system, these systems create separate maps when tracking is lost. Multi-session mapping systems use PRM to identify common regions in the maps and to merge them. Figure 2.7 shows a scenario where tracking has been lost. "Current map" in the upper diagram has become the "map 3" in the lower diagram and a new map has been created to hold sensor data.

ORBSLAM-atlas [34] is an extension to the ORB-SLAM system that adds multi-session mapping capabilities. The system has a place recognition database that manages detected key-frames of all the maps. If all the map regions identified by the PRM are in the active map, it is considered a loop closure, and if map regions identified by the PRM are on different maps (apart from on the active map and apart on a static map), it is considered a map merge.

ORBSLAM3 [35] is the third generation of the ORBSLAM system, introducing the ORBSLAM-atlas mentioned above to the ORBSLAM system and further improving the core systems. This system provides monocular and stereo visual-

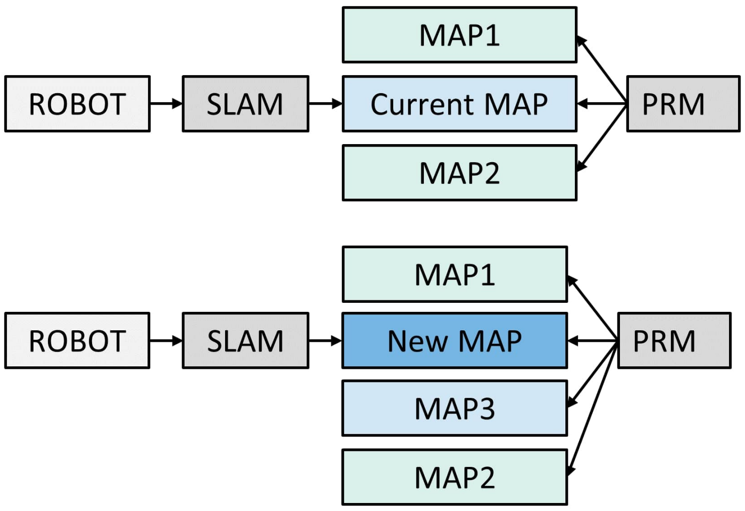


Figure 2.7: Abstract design of multi-session mapping systems

inertial SLAM, multi-session mapping with high recall place recognition, and an abstract camera representation that removes camera dependencies.

RTAB-Map [36] is a multi-session mapping supported SLAM system that takes LIDAR, depth cameras, monocular cameras, and stereo cameras data as inputs for localization and mapping. The system also includes a memory management subsystem that handles the pose-graph loaded onto the system memory to support long-term mapping without an issue. This subsystem also enables the system to store and maintain point clouds, identified features without discarding them. This system uses appearance-based localization and supports visual odometry and ICP odometry when the robot does not contain odometry sensors or has faulty odometry sensors.

2.4.3 Multi-robot Mapping Systems

Multi-robot mapping systems collect data from several robots and create a merged map. This mapping system could be deployed on a separate computer called a server if it used Server-Client system architecture or on the robots itself if it used distributed system architecture. In Server-Client systems, as shown in Figure 2.8, the client contains a tracking system for localization and the server contains maps and PRM for map merging. Maps are created when a new robot joins the system or when an existing robot loses tracking. The systems that follow distributed system architecture uses map sharing and caching techniques to share maps with each other.

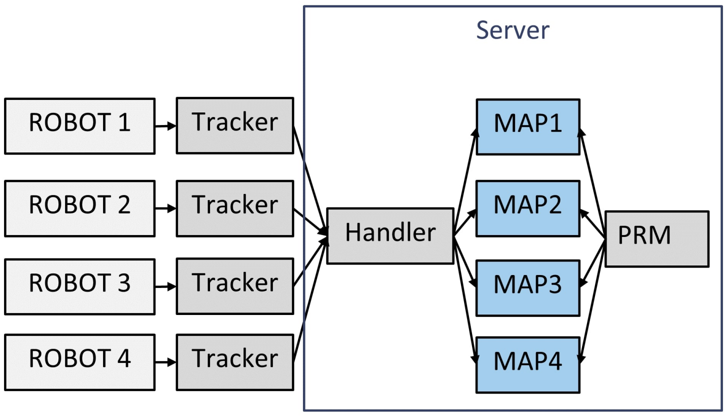


Figure 2.8: Abstract design of multi-robot mapping systems

CCM-SLAM [37] is a monocular image-based SLAM system that supports multi-robot mapping. The system has two components, a server component, and a client component. The client component that runs on the robot has a visual odometry system that maintains a local map with a limited number of key-frames used for real-time localization. The robot offloads heavy computational calculations such as place recognition, global optimization, and redundancy detection of the map to the server component. The server maintains a separate map for each robot until a common overlapping area can be identified, at which point the maps get merged. Since this system has been divided into two parts, network delays and failures can disrupt the mapping process.

C2TAM [38] is another cloud-based multi-robot mapping supported SLAM system that uses monocular images. In this system, tracking occurs on the robot, and mapping and place recognition occurs on the server-side. Relocation occurs both in robot and server because if the localization lost time is high, the robot can be in a completely different location that requires a large amount of processing power to calculate. Since this system has also been divided into two parts, network delays and failures can disrupt the mapping process.

FastSLAM2.0 [39] introduces batch processing to cloud-based SLAM systems. In this approach, each robot creates a local map and shares it with the cloud server. The server collects local maps and inter robot-robot observations. If the overlap between maps is above a threshold, it uses the overlaps to calculate the transformation between the local map and the robot positions. If the overlap is less than the threshold, the server uses inter robot-robot observations to calculate the transformation. Then it executes the map transform using the calculated transformation as a Hadoop Map/Reduce task.

DDF-SAM [40] describes a decentralized SLAM system back-end that implements a distributed mapping system. This system first uses single robot SLAM to create a map of the environment which is then shared among the other robots of the system. All the robots maintain a cached set of local maps of other robots, enabling them to share maps of third-party robots with the robots they connect. This implementation allows map sharing between robots that do not meet directly. A neighborhood map is created using landmarks on each local map using a constrained factor graph during its mapping process.

Map merge [27] is another system that implements multi-robot mapping. This system supports two approached initial position known merging and initial position unknown merging. The former approach creates a global map by simply transforming each map and merging it using the initial positions to calculate the relative transformation. The latter extracts key points and compares features to identify a valid transform between maps. The initial position is known merging is stable, and the initial position unknown merging depends on the overlap between the maps and features recognition accuracy, making it comparatively unstable.

ORBSLAMM [41] is an extension to the ORB-SLAM system, which adds the functionality of multi-session mapping and multi-robot mapping. This system adds a processing thread that handles multi-mapping in addition to the three threads already available with the ORB-SLAM system. This thread accepts different maps created by odometry failures (multi-session mapping) and different maps created by separate robots (multi-robot mapping). The newly introduced thread implements map insertion, matching, loop closing, and optimization.

2.4.4 Summary

Table 2.5 includes a summary of methods available for environment mapping.

Table 2.5: Summary of literature related to mapping

Name	Type	Remarks	Ref.
HectorSLAM	Basic	Uses occupancy grids for mapping and localization and supports three dimensional movement	[31]
ORBSLAM	Basic	Extract ORB features from monocular images, use motion only bundle adjustment for tracking and place recognition	[32]
ORBSLAM2	Basic	Extract ORB features from monocular images, depth data from depth images and use motion only bundle adjustment for tracking and place recognition	[33]
ORBSLAM-Atlas	Multi session	Use a place recognition database to track key frames across several maps and attempt to merge those maps	[34]
ORBSLAM3	Multi session	Integrate ORBSLAM-Atlas with ORBSLAM core system for multi session capabilities and further improve cores functionality	[35]
RTAB-Map	Multi session	Store sensor data in a pose graph and use feature matching, ICP methods to align them. Has memory management for long duration operations	[36]

CCM-SLAM	Multi robot	Only monocular images. Robot runs visual odometry while server runs place recognition, global optimization and map management. Network issues can affect the performance	[37]
C2TAM	Multi robot	Only monocular images. Robot runs tracking while server runs place recognition and map management. Relocation occur on both. Network issues can affect the performance	[38]
FastSLAM2.0	Multi robot	A cloud based SLAM system that uses batch processing for map transform. Robots share their maps with server and transform is calculated using overlap or robot-robot observation	[39]
DDF-SAM	Multi robot	Share own and third party maps with other robots in the system and calculate a neighbourhood map. Only proposes the back-end system for map sharing and merging	[40]
Map-Merge	Multi robot	Only focuses on map merging and no support for localization. supports feature based merging and initial position known merging	[27]
ORBSLAMM	Multi robot	Supports multi robot and multi session mapping. Extension to the ORBSLAM system that allows for new map creation for tracking failures and robot joining	[41]

Chapter 3

Methodology

3.1 Introduction

This section discusses the selected methodologies for the two components proposed in this research, a server system and a robot system. The literature review presented in the previous chapter has been analyzed against the requirements of each system, and selected techniques and data structures are presented in the following sections. The overview of methodology is illustrated in Figure 3.1.

3.2 Robot System

This system is expected to run on the robot and thus needs to be configured according to the robot's specifications. This system needs to communicate with the server system and operate as a part of a multi-robot system. It should also be able to operate individually as a fully functional exploration system without the support of the server system during communication failures and when the area to be explored is too small for a multi-robot system. Because of these requirements, we decided to include a mapping subsystem, an exploration subsystem, and a navigation subsystem as parts of the robot system. Each of the subsystems has been explained in the following sections.

3.2.1 Mapping

The mapping subsystem running on the robot needs to create a map of the environment for exploration and navigation. Since the robot needs localization capability for correct navigation, we decided to use a SLAM system as our mapping subsystem. We elected to use a multi-session mapping system as the mapping subsystem from the systems presented in the section 2.4.2.

- Simple mapping systems discussed in section 2.4.1 lacked the functionality required for our system compared to multi-session mapping systems.
- Multi-robot mapping system discussed in the section 2.4.3 could not be used because it did not support operation as a single robot system when required.

From the multi-session mapping systems available in the section 2.4.2, ORB-SLAM3 and the RTAB-Map were more prominent since ORBSLAM3 had the ORBSLAM-Atlas system integrated into it. The Table 3.1 includes a comparison between the two systems.

Table 3.1: Summary of robot coordination approaches

	RTAB-Map	ORBSLAM3.
Sensor support	Monocular camera, Stereo camera, RGBD camera, 2D LIDAR, 3D LIDAR	Monocular camera, Stereo camera, RGBD camera
Core technique used	Feature matching (cameras), ICP (LIDAR)	Feature matching (cameras)
Core structure used	Pose-graph with sensor data and extracted features	Pose-graph with extracted features
Long term operation support	Dedicated memory management system	Not available
Integrated Octomap support	Available	Not available
Output	Position Correction and Point cloud (All Data)	Position Correction and Point cloud (Keypoints)
Availability	Tested ROS package	3rd party ROS wrappers

Based on the details present in the Table 3.1 we chose to use RTAB-Map since it had an integrated octomap server, detailed point cloud output, and was available as a tested ROS package.

3.2.2 Exploration

The exploration subsystem running on the robot needs to be able to explore independently during single robot instances and under the guidance of the server system during multi-robot instances. To achieve this, we needed to design a complete exploration system on the robot. Since this subsystem acts as the core system for exploration in single robot and multi-robot instances, this would be the first stage of the two-stage exploration system proposed in this research. To achieve independence during operation, we designed the system based on the literature presented in the section 2.3.1. From the environment modeling methods presented in the section 2.1.1, we could use occupancy grids and octomaps to evaluate the environment probabilistically.

Table 3.2: Summary of robot coordination approaches

	Occupancy Grid	Octomap.
States available	Free, Occupied, Unknown	Free, Occupied, Unknown
Dimensions	Two	Three
Z axis compression	If the column of cells contain certain number of occupied cells, corresponding grid cell is considered occupied	None

Based on the comparison present in the Table 3.2 we chose to use octomaps which are calculated based on the point clouds created by the mapping system, for exploration calculation since it prevents the loss of data on the z-axis compared to occupancy grids and allows us to identify unexplored regions throughout the whole volume of space.

3.2.3 Navigation

The navigation subsystem running on the robot needs to be able to navigate independently during single robot instances and under the guidance of the server system during multi-robot instances. Though we selected octomaps for exploration calculations, for navigation, we elected to use occupancy grids extracted from the octomap for simplicity. In this implementation, we filtered the octomap for obstacles while considering the height of the robot to consider only the obstacle within the robot's height limit.

3.3 Server System

This system is expected to communicate with the robots to collect data from them and to guide them for efficient exploration. The system is only used as a part of a multi-robot system either on a stationary computer or on a companion computer on a field robot, along with an instance of a robot system. Because of this, we decided to include a mapping subsystem, an exploration subsystem, and a coordination subsystem as parts of the server system. Each of these subsystems has been explained in the following sections.

3.3.1 Mapping

Based on the literature presented in the section 2.4.3, there were two significant approaches we identified that could be used to design this subsystem, an initial position known mapping where robots knew each others position at least relative to a single robot and initial position unknown mapping, which utilized a place recognition algorithm to estimate transformations between robots. The following Table 3.3 contains the differences we identified through literature and the basic experiments to evaluate both approaches.

Table 3.3: Summary of robot coordination approaches

	Initial position known	Initial position unknown.
Required overlap	Not required	large

Time till merge	None	Considerable
Accuracy	High	Depends on overlap and features
Computational time	Low	High

Our initial position unknown mapping tests were based on feature matching approaches, and they showed that a considerable overlap was required between the maps created by robots. One reason for this could be the degradation of features on the map due to the map creation process in the selected SLAM system. The map was created by merging point cloud sensor data and distributing the error detected during loop closing between the point clouds of the loop. Comparing two maps that were created using this process on two separate computing devices makes feature matching difficult compared to the feature matching of point cloud sensor data taken from the same camera. Merging accuracy was also low, and computational time was high due to the same above reason. Because of that, we selected to use the initial position known approach for designing the mapping subsystem of the server system.

3.3.2 Exploration

The exploration subsystem of the server system needs to function along with the exploration subsystem of the robot system. Since this system assists the exploration subsystem of the robot during multi-robot instances, this subsystem act as the second stage of the two-stage exploration system proposed in this research. This system follows the same design principle as the exploration subsystem of the robot system but uses the merged point cloud map created by the mapping subsystem of the server system instead. We selected to use octomaps to convert the merged point cloud map into a map that can be used for exploration calculations similar to the exploration subsystem implementation of the robot system.

3.3.3 Coordination

Division of map approaches discussed in the section 2.2.1 and Association of Robots methods discussed in the section 2.2.2 are the primary approaches for coordination we identified during the literature review. Following were the conclusions we reached after evaluating each method against our system structure.

- The server system can derive the robot's position from the starting position and odometry at any time because we selected known position mapping for the server-side mapping subsystem. Availability of the robot's position allowed us to use the "Zone partition method" from the beginning of exploration to assign exploration targets and avoid overlapping the exploration area.
- "Circle partition method" causes the robots to move through sectors assigned to other robots causing overlapping and repetitive exploration unless the robots enter the given area from all sides or to their respective sector directly that makes this approach less effective than the "Zone partition method."
- In methods discussed under Association of Robots, all robots take the same path until they reach a branching point and then take the branching paths as described in each respective method. Following the same path means that this approach also causes overlapping of exploration areas and is less effective than the "Zone partition method."

We selected the "Zone partition method" from the approaches mentioned in the section 2.2.1 over Association of Robots methods in section 2.2.2 and "Circle partition method" in section 2.2.1 due to the reasons we presented above.

3.4 Summary

Figure 3.1 illustrates the overview of methodology including technologies selected for each subsystem, and selected input and output data

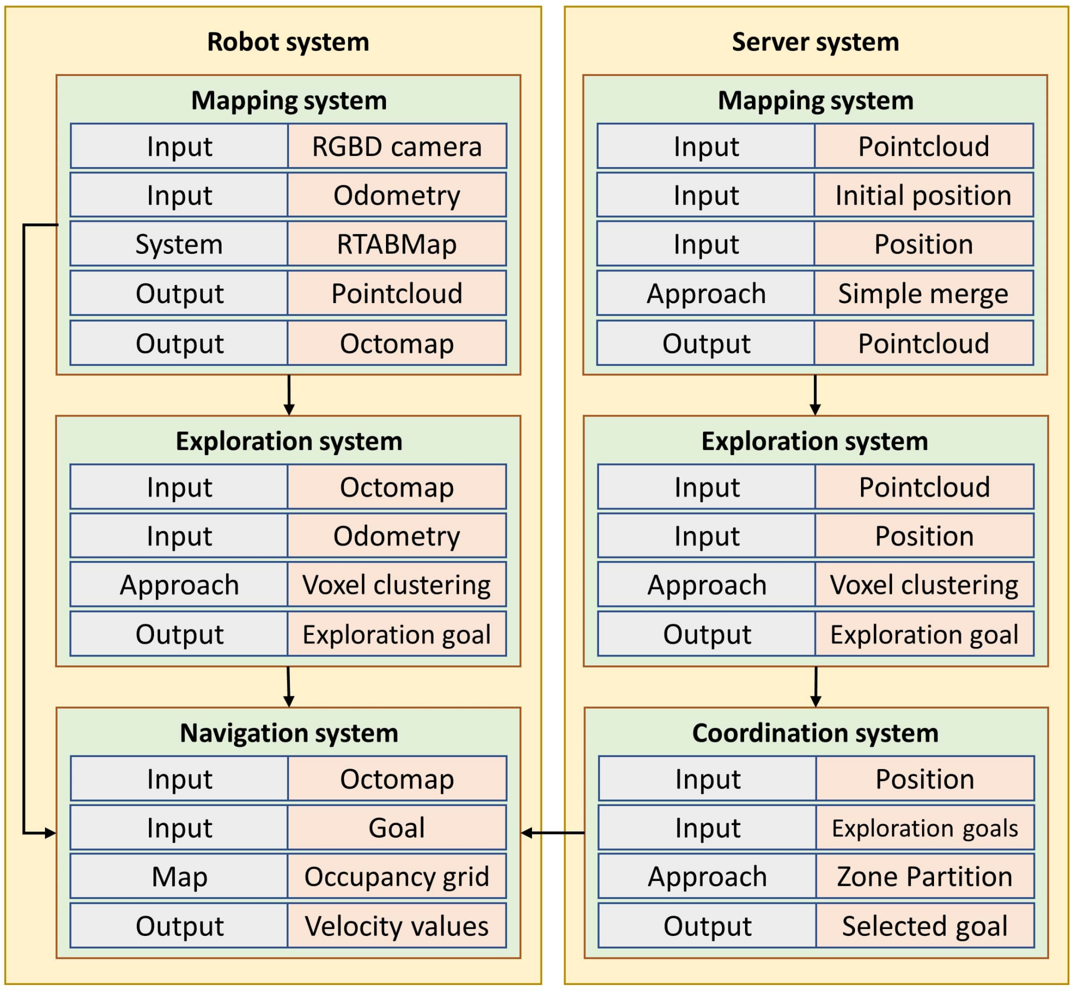


Figure 3.1: Overview of Methodology

Chapter 4

System Design

4.1 Introduction

Figure 4.1 illustrates the expected structure of the system proposed in this research. As shown in Figure 4.1, the robots share the robot's name, point cloud, initial position, and the robot's position with the server while the server shares the goals for each robot for exploration.

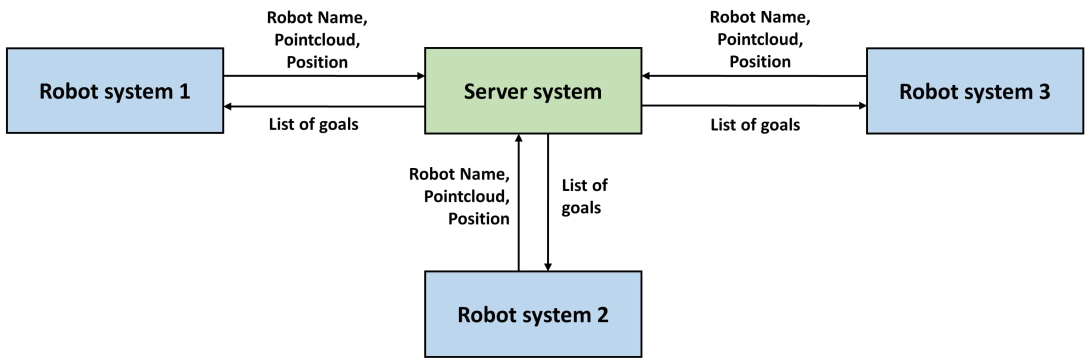

Robot system 1
Robot Name, Pointcloud, Position
List of goals
Server system
Robot Name, Pointcloud, Position
List of goals
Robot system 3
Robot Name, Pointcloud, Position
List of goals
Robot system 2

Figure 4.1: Expected use of the multi-robot system

The proposed system should create a merged map of the intended area using maps created by individual robots. Then it should identify unexplored regions in the merged map and assign exploration tasks to each robot so that the intended area can be explored efficiently. Considering these requirements, we use Server-Client architecture to create the system. We use point clouds as maps since they allow us to visualize and store the environment with a three-dimensional perspective.

The server system calculates the relative position of each robot relative to a selected leading robot named *root* and uses those relative positions to calculate the relative transformations of each robot. Once calculated, the relative transformations transform point clouds and merge them into a single point cloud. The merged point cloud is used to calculate an octomap that enables the statistical evaluation of the three-dimensional space. The octomap is used to identify unexplored regions in the intended area, and these regions are assigned to each robot depending on their positions. An exploration goal is identified for each robot and is assigned to that robot from the assigned area.

Each robot system act as a client component of the architecture. The robot system creates a point cloud of the environment using a SLAM system. This point cloud is used for on-robot calculations as well as transmitted to the server for remote processing. A robot system consists of a *Exploration Module* for robot exploration calculations, a *Planning Module* for path planning and module coordination, a *Control Module* for controlling the robot, and a *ClientModule* for communicating with the server. The *Exploration Module* performs calculations similar to the server, and in multi-robot system instances, it is used by the robot in between server connections when the communication gap becomes more significant and also used as a backup system in case of a communication failure.

4.2 Server System

The server system collects data from all the robots available and processes them to identify unexplored regions. Once identified, it guides the robots to explore the identified unexplored regions. Figure 4.2 illustrates the high-level design of the processing pipeline implemented in the server system.

In order to allow the robots to join at any time and allow a variable number of robots to join, we designed the system with one input port and multiple input processing threads. Figure 4.2 illustrates an instance of the system with n robots and m input processing threads. Input processing threads extract data such as robot name, point cloud, position, initial position, and orientation and store them separately based on the robot's name separately. If a previous entry exists under the robot's name, those are updated, and otherwise, a new entry is created based on the name. The input data processing thread can also be extended to perform additional operations on input data such as filtering, clustering, and

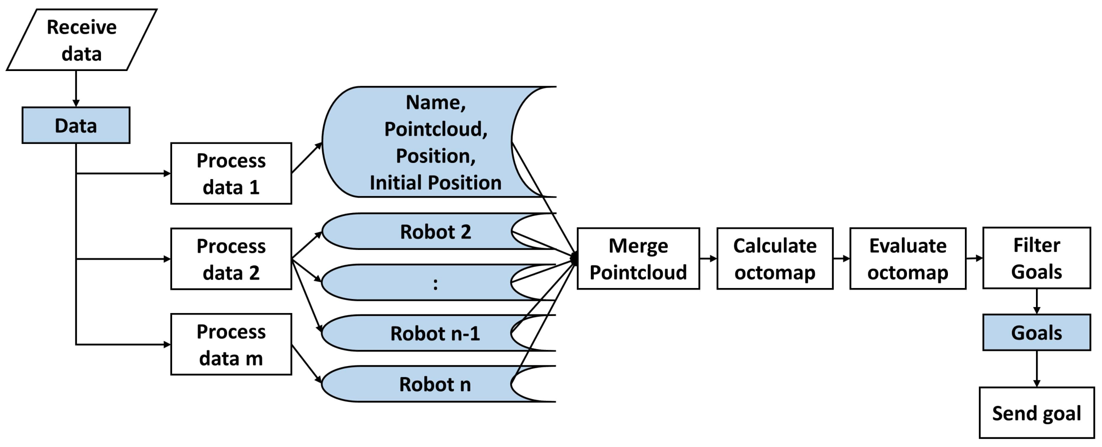


Figure 4.2: Abstract design of the server system with n robots and m input processing threads

plane extraction. The extracted data are used to calculate the merged point cloud, calculate the octomap and ultimately identify the unexplored regions. Figure 4.3 illustrates the pipeline from merging the point clouds to the selection of goals in detail. This pipeline has not been optimized and runs on a single thread. The following sub-sections contain a detailed description of each step from merging the point cloud to filtering goals.

4.2.1 Merging of Point Cloud

First, for each robot, a relative transformation is calculated using the initial position of the robot. This transformation is calculated relative to a robot named *root*, which can be any robot from the robots in the system that needs to be selected at the start of the operation of the server system. This robot is considered the leading robot, and the point cloud created by this robot is used as the base point cloud for map merging. The overall area to be explored by all the robots need to be defined relative to the *root* robot.

All points in the point clouds from all the robots are transformed into the *root* robot's coordinate frame (equation 4.1). The transformed point clouds are merged onto a single point cloud that visualizes the total area explored and mapped by all the robots.

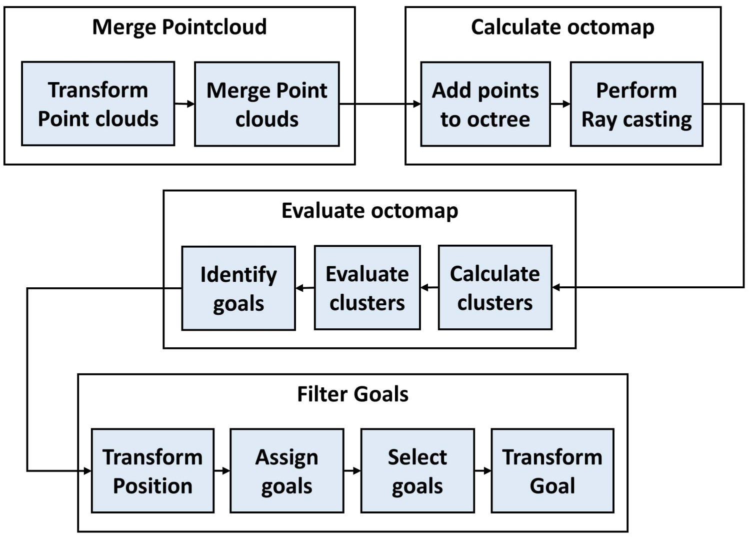


Figure 4.3: Detailed design of processing stages from merging of point clouds to filtering of goals

$$^{root}PointCloud =^{root} T_{robot} \, ^{robot}PointCloud \tag{4.1}$$

4.2.2 Calculation of Octomap

The merged point cloud is used to calculate an octomap of the environment. The point cloud is used to estimate unknown and known cells in the octomap. The initial position of the *root* robot is considered the sensor origin, and based on that, ray casting is performed. Ray casting allows differentiating known cells between occupied and unoccupied cells.

4.2.3 Evaluation of Octomap

The calculated octomap is divided into voxel clusters, and the voxels in each cluster are evaluated for their occupancy state. The evaluation checks for the percentage of unknown cells in each cluster, and if it is larger than a predefined value, the cluster is considered an unexplored cluster, and its center is selected as a candidate goal for exploration. Then the unreachable goals are removed from the candidate goals using a different goal set maintained on the server named *Unreachable Goals*. The Algorithm 1 explains this procedure.

Algorithm 1: Exploration Goal identification using Octomap

```
Input: Octomap, Boundaries of the area to be explored, octomap
       resolution, cluster resolution, percentage, Unreachable Goals
Output: Exploration goal coordinates
1  $N_x, N_y, N_z \leftarrow$ cluster counts along each of X,Y,Z dimensions
2  $Goals \leftarrow \emptyset$
3  for $i \leftarrow 0$ to $N_x$ do
4      for $j \leftarrow 0$ to $N_y$ do
5          for $k \leftarrow 0$ to $N_z$ do
               /* here $i$, $j$, $k$ are not distance measures */
6              $unknownCount, \leftarrow 0$
7              calculate cluster center using $i$, $j$, $k$, octomap resolution,
                 octomap resolution
8              calculate $X_{min}, X_{max}, Y_{min}, Y_{max}, Z_{min}, Z_{max}$ cluster boundaries
                 using cluster center, $i$, $j$, $k$, octomap resolution, cluster
                 resolution
9              for $a \leftarrow X_{min}$ to $X_{max}$ do
10                 for $b \leftarrow Y_{min}$ to $Y_{max}$ do
11                     for $c \leftarrow Z_{min}$ to $Z_{max}$ do
                           /* here $a$, $b$, $c$ are distance measures */
12                         if point (a,b,c) not available in Octomap then
13                             increment $unknownCount$
14             if $unknownCount \geq percentage*totalCellCount$ then
15                 $Goals \leftarrow$ cluster center
16 for rejectedGoal in UnreachableGoals do
17     if rejectedGoal is in Goals then
18         remove rejectedGoal from Goals
19 return Goals
```

4.2.4 Filtering of Goals

Once the candidate goals are calculated, the relative transformations are used to calculate the robot's position (equation 4.2) in the coordinate frame of the *root* robot in which the rest of the calculations occur.

$$^{root}Position = ^{root}T_{robot}\,^{robot}Position \tag{4.2}$$

Two processing stages follow; in the first stage, candidate goals need to be assigned to the robots. In the second stage, the most reasonable goal for each robot needs to be selected from the set of candidate goals assigned to each robot. Two separate cost functions are used in the two stages.

The first stage has been designed based on the *Voronoi Decomposition*. Euclidean distance from the robot to the goal has been used as the cost function to divide the candidate goals among the robots. If the set of robots R, set of goals G, euclidean distance cost function d and A_k, A_j $(j, k \in R)$ are two robots, the goal region Z_k associated with robot A_k is the set of all goals in G whose distance to A_k is less than to other robots T_j where $k \neq j$. Z_k can also be denoted as in equation 4.3

$$Z_k = \{g \in G \mid d(g, T_k) \leq d(g, T_j) \; \forall \; j \neq k \; j, k \in R\} \tag{4.3}$$

The robot-goal pair with the least cost is selected in the first implementation stage, resulting in goals being assigned to its closest robot. The Algorithm 2 explains the implementation.

Algorithm 2: Assigning goals to robots

```
   Input: Goals, Robots
   Output: Goals Divided among robots
1  for goal in Goals do
2      closestRobot ← None
3      minDistance ← ∞
4      for robot in Robots do
5          calculate distance from robot to goal
6          if distance < minDistance then
7              minDistance ← distance
8              closestRobot ← robot
9      select closestRobot from Robots and assign goal to it
10 return Robots
```

In the second stage, euclidean distance has been used as the cost function to select a goal for each robot from the candidate goals assigned to it. The goal with the least cost was selected from the candidate goals assigned to each robot as the selected goal for exploration. The Algorithm 3 explains the implementation.

Algorithm 3: Selecting goals for each robot

```
Input: Robots
Output: Robots with one selected goal each
1  for robot in Robots do
2      selectedGoal ← None
3      minDistance ← ∞
4      for goal in robot do
5          calculate distance from robot position to goal
6          if distance < minDistance then
7              minDistance ← distance
8              selectedGoal ← goal
9      robot ← selectedGoal as a selected goal for exploration
10 return Robots
```

After selecting a goal for each robot, they are transformed back into the coordinate frame of the respective robot and transmitted from a single output port as a single message. This message contains goals assigned to each robot and the robot's name, which allows for a variable number of robots to receive the transmission. *Client Module* in each robot listens to this port and receives the transmission. Once received, it extracts the exploration goal intended for the host robot using the host robot's name as the search key. The extracted goals are then transmitted to *Planner Module* for path planning to reach and explore them. If any of these received goals cannot be reached, those goals are transmitted back to the server by *Client Modules* through a separate port, and the server adds them to the Unreachable goals set maintained in the *Server System*.

4.3 Robot System

This system is based on the motion planner, [9] and the modules proposed in this research are primarily based on several subsystems of the motion planner. *Exploration Module* is based on the Goal Identifier subsystem and focuses on processing the octomaps to identify exploration goals. *Planning Module* is based on the Path Planner subsystem focuses on path calculation required to reach goals calculated or received by other modules, and *Control Module* is based on the Velocity Controller subsystem and focuses on controlling the robot as required.

Combination of *Exploration Module*, *Planning Module* and *Control Module* allows the system to perform as a single robot system. SLAM system manages the localization and mapping process with input from the camera system, and the octomap server system calculated octomaps using the point cloud data from the SLAM system or sensor data from the camera system. Figure 4.4 illustrates the high-level design of the robot system, including data and command flow between modules.

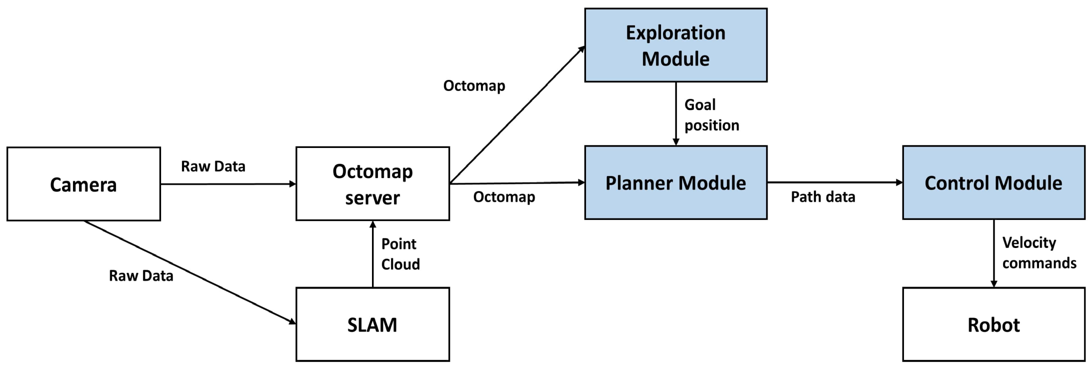


Figure 4.4: Robot system structure when operating as a single robot system

Compared to the subsystems of motion planner [9], each module's process pipeline has been optimized and extended to support server coordinated multi-robot functionality while retaining the ability to operate as a single robot system.

Client Module was newly designed to act as the interface between the modules mentioned above and the server system when the robot system functions as a part of a multi-robot system. It receives data from the server and transmits data to the server. Figure 4.5 illustrates the high-level design of the entire Robot system, including data and command flow between modules.

4.3.1 Simultaneous Localization and Mapping System

In this research, we have selected the RTAB-Map [36] system as the SLAM system of our proposed robot system due to its ability for long-term operation and the readily available point clouds. Internally the system maintains a pose-graph

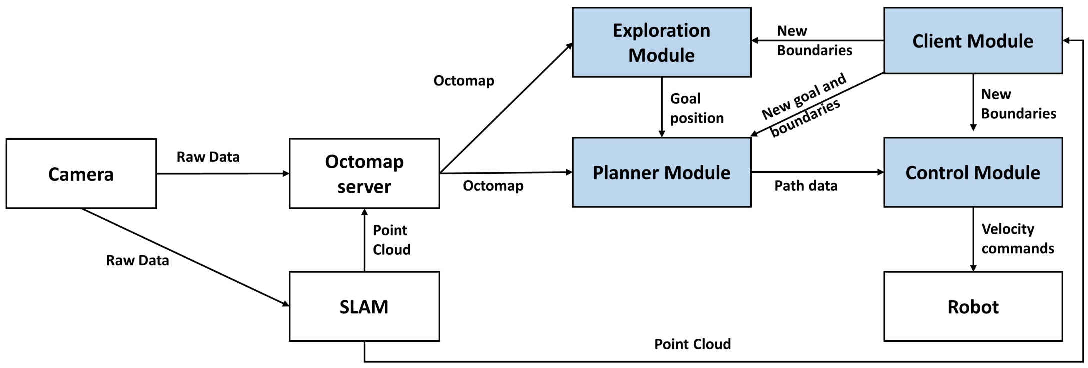


Figure 4.5: Robot system structure when operating as a multi-robot system

with nodes containing a time-based index, bag-of-words based on features, sensor data, and edges containing rigid transformations. The system uses a bag-of-word approach for loop closing. In addition to that, it also has an integrated octomap server that allows us to run a single system instead of separate SLAM and octomap server systems. This octomap uses a point cloud calculated from the SLAM system of octomap calculations. We use an RGB-D camera as an input device and uses odometry data from the robot as input position data.

4.3.2 Exploration Module

Figure 4.6 illustrates the basic *Exploration Module*, which is based on the Goal Identifier subsystem proposed as a part of the motion planner [9]. We have replaced the odometry-based position tracking used in the Goal Identifier subsystem with a native position tracking system implemented in the *Control Module*. We have also further optimized the processing pipeline compared to the Goal Identifier subsystem. Figure 4.6 illustrates the *Exploration module* operation when the robot operates as a single robot system.

In addition to the above modifications, we have extended the *Exploration Module* pipeline to allows it to be used in robot system instances in multi-robot systems. The new functionality allows the *Client Module* to resize the boundaries of the area considered for exploration such that the goal assigned from the server

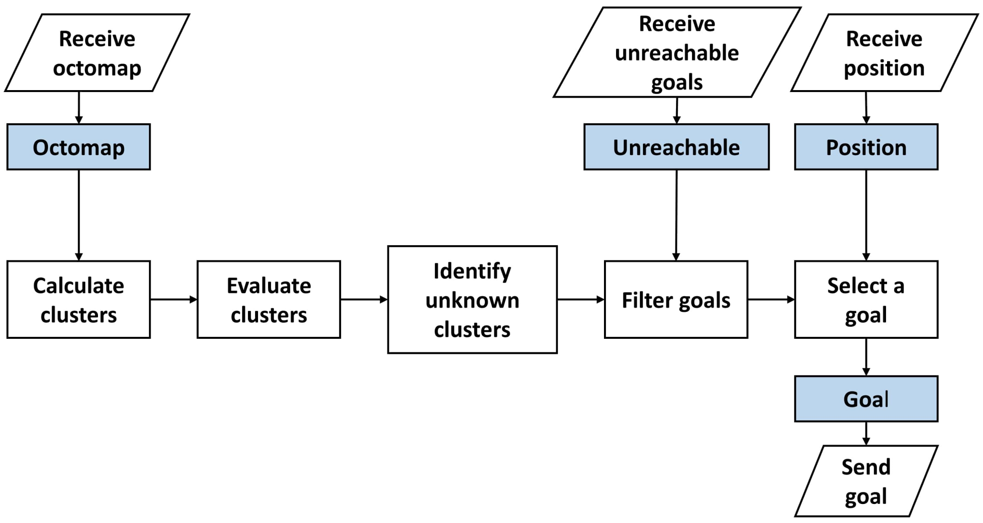


Figure 4.6: Exploration module during single robot operation

can be accepted even if it exists outside the region specified at the start for each robot instance. The ability to update the boundaries allows the robot system instance to continue exploring the surrounding area of the assigned goal in between server transmissions. Figure 4.7 illustrates the resultant system architecture of the *Exploration Module* after adding new functionality.

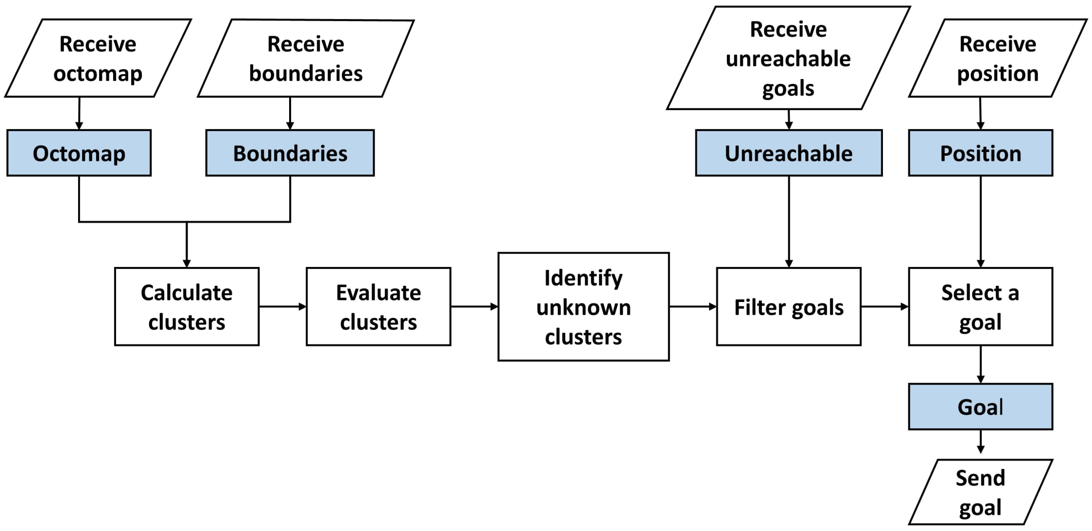


Figure 4.7: Exploration module during multi-robot operation

The *Exploration Module* uses octomaps to evaluate the surroundings to identify unexplored regions within specified boundaries. The octomap is divided into clusters, and clusters are evaluated for their occupancy state. The evaluation checks for the percentage of unknown cells in each cluster, and if it is larger than a predefined value, the cluster is considered an unknown cluster, and its center is selected as a candidate exploration goal. Then the unreachable goals are removed from the candidate goals using a different goal set maintained on the exploration module named *Unreachable Goals*. This is the same procedure used in the server system, and thus The Algorithm 1 can be used to explain this procedure.

Once the candidate goals are filtered, a single goal must be selected as the exploration goal from the set to be sent to the *Planning Module*. Euclidean distance was used as the cost function to select this. The goal with the least cost was selected from among the goals for exploration and is transmitted to the *Planning Module*. If the *Planning Module* cannot calculate a path to reach the selected goal, it transmits the goal back, and the *Exploration Module* adds it to the *Unreachable Goals* set. Algorithm 4 explains the implementation.

Algorithm 4: Selecting a goal for exploration

```
  Input: Goals
  Output: One selected goal
1 selectedGoal ← None
2 minDistance ← $\infty$
3 for goal in Goals do
4     calculate distance from robot position to goal
5     if distance < minDistance then
6         minDistance ← distance
7         selectedGoal ← goal
8 return selectedGoal
```

4.3.3 Planning Module

Figure 4.8 illustrates the basic *Planning Module*, which is based on the Path Planner subsystem proposed as a part of the motion planner [9]. We have replaced the odometry-based position tracking used on the Path Planner subsystem with a native position tracking system implemented in the *Control Module*. We have improved the path conversion functionality to remove intermediate points in the path when the path is straight, which improves the robot's movement by reducing the number of waypoints in the path and increasing the distance between waypoints in the path. We have also further optimized the processing pipeline compared to the Path Planner subsystem. Figure 4.8 illustrates the Planning module operation when the robot operates as a single robot system.

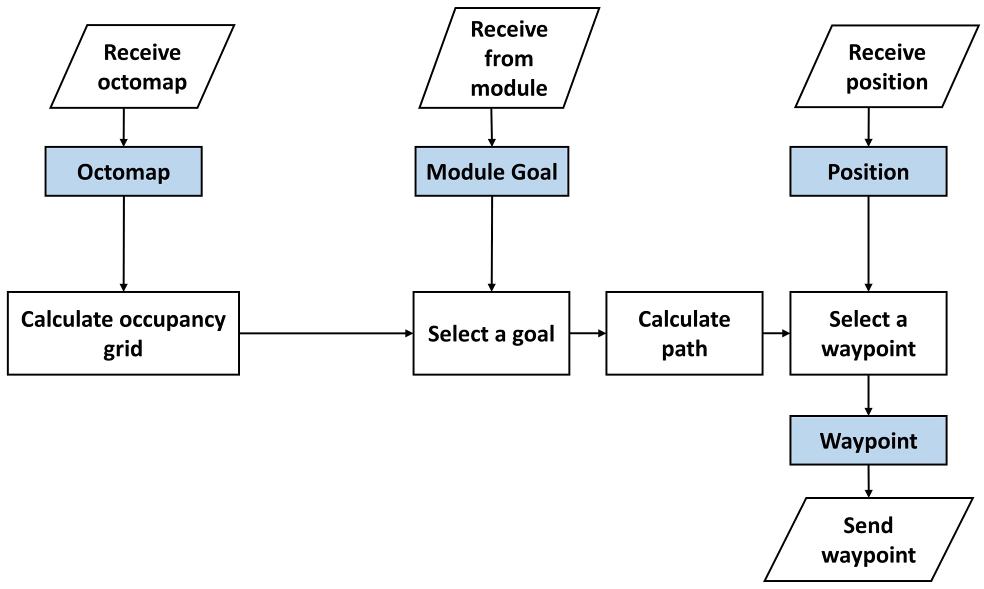


Figure 4.8: Planning module during single robot operation

In addition to the above modifications, we have extended the *Planning Module* pipeline to allow it to be used in robot system instances in multi-robot systems. One of the new functionality allows the *Client Module* to resize the boundaries of the area considered for path planning such that the goal assigned from the server can be accepted even if it exists outside the region specified at the start for each robot instance. This resizing allows the path planning calculations to succeed even if the server assigned goal is outside the region considered in the occupancy grid.

Another new functionality that we introduced was a multiplexer to select between the goals from the server and goals from the *Exploration Module*. Multiplexer assigns a higher priority to goals from the server, which enables the *Planning Module* to reach and explore regions outside the area specified at the start. We also added new functionality to report unreachable goals from the server to the *Client Module* to transmit the unreachable goal back to the server for filtering. Figure 4.9 illustrates the resultant system architecture of the *Planning Module* after adding new functionalities.

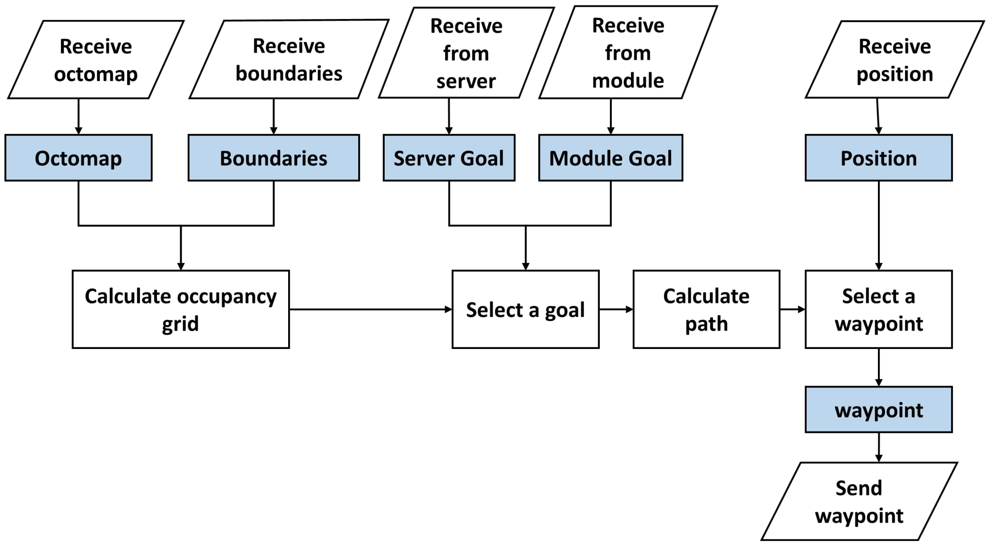


Figure 4.9: Planning module during multi-robot operation

Unlike the Path Planner subsystem proposed as a part of the motion planner [9], *Planning Module* uses both octomap and boundaries to calculate the occupancy grid. The occupancy grid is created by down projecting the octomap within boundaries. As explained earlier, it uses a multiplexer implementation to select between goals from the server and goals from *Exploration module*. Once the occupancy grid is calculated and the goal is selected, the module uses the A* algorithm to calculate a path. It then converts the path to real-world coordinates and transmits waypoints along the path to *Control Module* to move the robot to reach the selected goal. Algorithm 5 contains the explained control sequence. Though *Planning Module* coordinates both *Exploration Module* and *Control Module* similar to Path Planner subsystem, it does not coordinate but only communicates with the new *Client Module*.

Algorithm 5: Path planning to reach the goal

```
Input: Octomap, position
Output: Movement commands to Control Module
1  $IncompleteExploration \leftarrow True$
2  while $IncompleteExploration$ do
3      Update map
4      $goal \leftarrow serverGoal$ or $moduleGoal$ from multiplexer
5      calculate occupancy grid
6      calculate path
7      while path is not valid do
8          if position is blocked then
9              move robot
10         else
11             remove goal
12         $goal \leftarrow serverGoal$ or $moduleGoal$ from multiplexer
13         if $goal = serverGoal$ then
14             recalculate occupancy grid
15         recalculate path
16     convert path
17     while goal not reached do
18         move robot to next way point
```

4.3.4 Control Module

Figure 4.10 illustrates the base common control sequence used in the forward, backward, and rotational movements available in the *Control Module*, which is based on the Velocity Controller subsystem as proposed as a part of the motion planner [9]. We have improved the controlling capability and have extended the control interface to ROSBot 2.0 robot model in addition to the Turtlebot robot model. Like the Velocity controller, *Control Module* also offers three basic movements, Forward, Backward, and Rotation. Figure 4.10 illustrates the Planning module operation when the robot operates as a single robot system.

We have replaced the odometry-based position tracking with position tracking using both odometry and transformation data from *SLAM System*, illustrated in Figure 4.11. Since the transformation data from *SLAM System* has a low frequency compared to odometry, whenever the transformation data is available, we use it to calculate the difference between correct position and odometry whenever

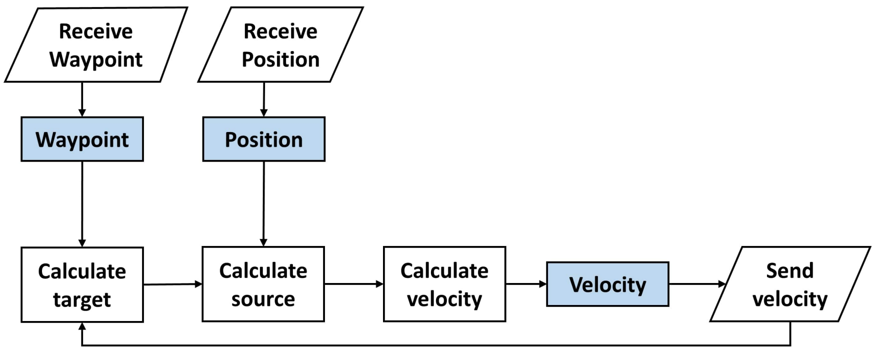


Figure 4.10: Common control sequence offered by control module as a single robot system

the transformation data is available. Until the following transformation data is made available by the *SLAM System*, we use the calculated difference to derive subsequent positions. Algorithm 6 contains the explained process. Though the positions are not highly accurate compared to only using transformation data, this approach enables a high frequency similar to odometry with reduced error. *Control Module* transmits this improved position data to all other modules.

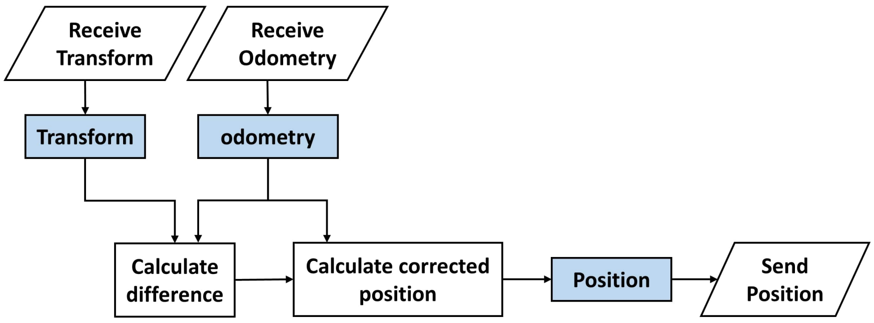


Figure 4.11: Position tracking system

In addition to the above modifications, we have extended the *Control Module* to allow it to be used in robot system instances in multi-robot systems. The new functionality allows the *Client Module* to resize the area's boundaries considered for robot movement. The goal assigned from the server can be accepted even if it exists outside the region specified at the start for each robot instance because the ability to update the boundaries allows the robot system instance

Algorithm 6: Position Tracking with SLAM and odometry

```
   Input: Transformation from SLAM System, Odometry
   Output: Corrected position data
1  $\Delta_x, \Delta_y, \Delta_z, \Delta_{roll}, \Delta_{pitch}, \Delta_{yaw}, \leftarrow 0$
2  while $True$ do
3      if $SLAM\ transformation\ is\ available$ then
4          $\Delta_x \leftarrow x_{SLAM} - x_{odometry}$
5          $\Delta_y \leftarrow y_{SLAM} - y_{odometry}$
6          $\Delta_z \leftarrow z_{SLAM} - z_{odometry}$
7          $\Delta_{roll} \leftarrow roll_{SLAM} - roll_{odometry}$
8          $\Delta_{pitch} \leftarrow pitch_{SLAM} - pitch_{odometry}$
9          $\Delta_{yaw} \leftarrow yaw_{SLAM} - yaw_{odometry}$
10     $x_{correct} \leftarrow x_{odometry} + \Delta_x$
11     $y_{correct} \leftarrow y_{odometry} + \Delta_y$
12     $z_{correct} \leftarrow z_{odometry} + \Delta_z$
13     $roll_{correct} \leftarrow roll_{odometry} + \Delta_{roll}$
14     $pitch_{correct} \leftarrow pitch_{odometry} + \Delta_{pitch}$
15     $yaw_{correct} \leftarrow yaw_{odometry} + \Delta_{yaw}$
16 return $x_{correct}, y_{correct}, z_{correct}, roll_{correct}, pitch_{correct}, yaw_{correct}$
```

to track and reach a position outside the existing boundaries. Figure 4.12 illustrates the resultant system architecture of the *Control Module* after adding new functionality.

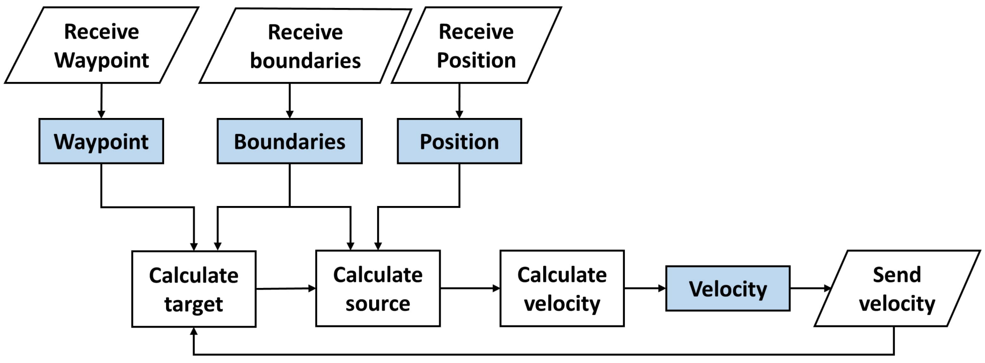


Figure 4.12: Common control sequence offered by control module as a multi-robot system

4.3.5 Client Module

This module is used when the robot system operates as part of the multi-robot system and not as a single robot system. The module is designed to connect with the *Server System* in order to transmit data back and forth between server and robot system. Figure 4.13 illustrates the intended server and robot connectivity through the *Client Module*.

As illustrated in Figure 4.14, *Client Module* collects data such as robot name, initial position, and orientation from the configuration file, point cloud generated by the *SLAM System* and position data from the *Control Module* and transmits them to the single input port in the server to be used for calculations.

It also listens to the transmissions from the server and receives exploration goals from which it selects the appropriate goal using the host robot's name. Then it uses the goal to calculate new area boundaries as shown in Figure 4.15 such that new boundaries encompass the new goal. This allows the other modules to perform calculations without failure. Then it updates other modules with the new goal assigned by the server and new area boundaries calculated based on the goal. When the *Planning Module* notifies it about unreachable goals assigned by the server, *Client Module* transmits the unreachable goals back to the server so that they would be excluded in future calculations.

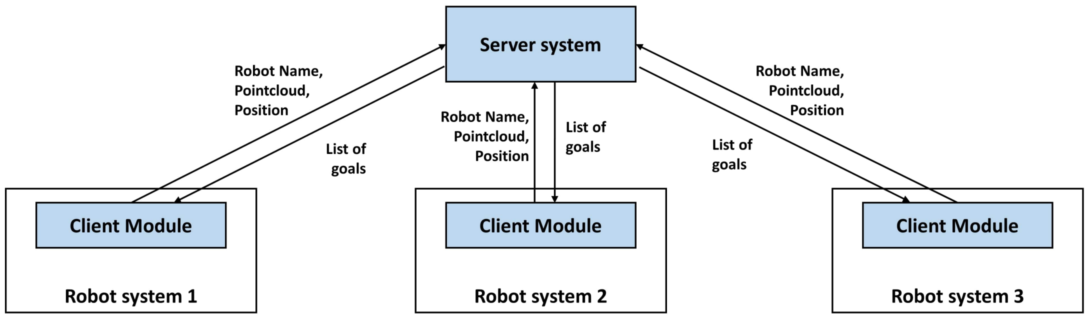


Figure 4.13: Connectivity between the server system and client modules

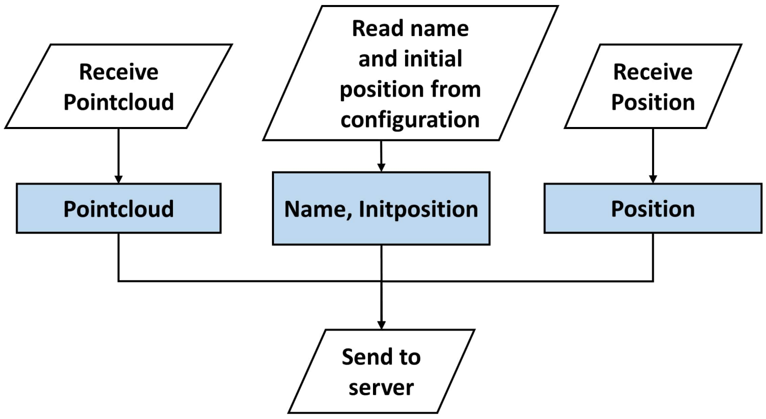


Figure 4.14: Data collection by client module to be sent to the server system

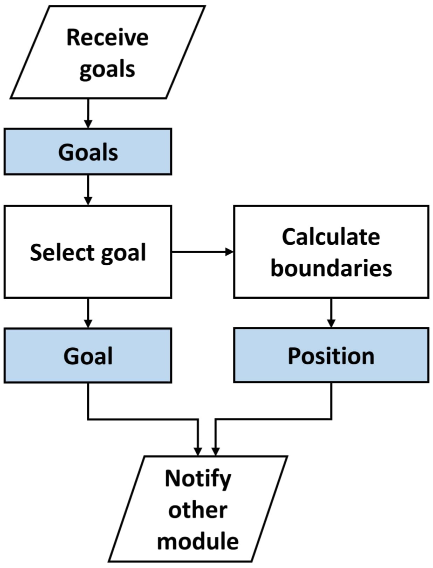


Figure 4.15: Data received by the client module is used to update other modules

.

Chapter 5

EXPERIMENTS AND RESULTS

In this section, we explain the experiments we performed to evaluate our system. First, we present the performance evaluation criteria we designed to assess exploration systems, and then we present the single robot experiments we performed and the collected results. Then we present the multi-robot experiments we performed, including two robot systems and three robot systems on the same simulation environment. Then we present the results we collected during experiments related to two robot and three robot systems and compares them with one robot experiment results collected from the same simulation environment to identify performance gains and losses the multi-robot systems have over one robot system.

5.1 Performance Evaluation Criteria

Due to the lack of proper evaluation criteria for evaluating the performance of exploration systems statistically, we formulated the criteria based on the performance evaluation of pure-motion tasks [42] on the basis that a single *exploration task* could be considered as a collection of *pure motion tasks* and *map evaluation tasks.*

A *map evaluation task* is only related to exploration, and it can either be a task related to processing on the onboard computer or be a simple movement that enables the robot to extend or improve the map. A *pure motion task*, as described in [42], is only concerned with moving from one place to another.

The proposed performance evaluation criteria contain five components: Time to explore, Exploration path length, Lateral Stress, Tangential Stress, and Explored Percentage. We have explained each of the components in detail in the following sub-sections.

5.1.1 Time to Explore

As mentioned earlier, we consider an exploration task to consist of pure motion tasks and map evaluation tasks, and *Time to reach the goal* measure [42] focuses only on measuring the robot's operating time to complete the pure motion tasks. Hence, we propose a new measure named *Time to explore* to measure the total time the robot spends on pure motion and map evaluation tasks instead of *Time to reach the goal* measure.

For single robot systems, we calculate *Time to explore* as the time it takes for the robot to complete the *exploration task.* For multi-robot systems, we calculate *Time to explore* as the time it takes for all the robots in the system to complete the *exploration task* while coordinating with each other.

When considering this measure, exploration systems with a low *Time to explore* value perform better. This measure is important when the robot is required to explore in a time-critical situation (e.g., disaster zone) or with a limited power supply (e.g., battery or fuel).

5.1.2 Exploration Path Length

Path length measure [42] focuses on measuring the distance the robot travels to complete a pure motion task. Depending on the design of the exploration system, the Map evaluation task may or may not include movements intended for extending for the map. In order to accommodate the movement during map evaluation tasks, we define a new distance measure, *Exploration Path Length* as the distance the robot travels during an *exploration task.* This measure contains the distance traveled during both pure motion tasks and map exploration tasks.

For single robot systems, we calculate *Exploration Path Length* directly as the distance the robot travels during an *exploration task.* For multi-robot systems, we calculate *Exploration Path Length* on the system level as an accumulated value and an average distance value a robot traveled to complete the *exploration task* while coordinating with other robots.

This measure is also important when the robot operates with a limited power supply. When considering this measure, exploration systems with a low *Total Path Length* value perform better. When considering multi-robot systems, having a low *Total Path Length* standard deviation indicates having a better coordination strategy because that means all robots have traveled nearly similar distances.

5.1.3 Lateral Stress

The stability of a robot depends on the forces acting upon its body. The forces in the lateral direction to the direction of travel is a primary factor that decides the robot's stability. When the centrifugal force acting on the robot is low, the robot becomes stable. The third measure, *Lateral Stress* [42], is used as proposed. *Lateral Stress* indicates the stress on the robot in the lateral direction to the direction of travel. The *Lateral Stress* can be calculated by integrating the centrifugal forces along the trajectory. If the instantaneous linear speed is *v(t)* and instantaneous angular speed is *w(t)*, considering a unit mass, the *Lateral Stress (LS)* can be calculated as in equation 5.1.

$$LS = \int_0^{t_F} v(t) \cdot w(t)\, dx \tag{5.1}$$

For single robot systems, we calculate *Lateral Stress* as the stress on the robot in the lateral direction to the travel direction during an *exploration task*. For multi-robot systems, we calculate *Lateral Stress* on the system level as an accumulated value and as an average value on a robot in the lateral direction to travel direction during an *exploration task* while coordinating with other robots.

If an exploration system has a lower *Lateral Stress* value, the robot has higher stability. When considering multi-robot systems, having a low *Lateral Stress* standard deviation indicates having a better coordination strategy because that means all robots have traveled on smooth and equally long trajectories.

5.1.4 Tangential Stress

The forces acting on the robot along the direction of travel are the other main factor that decides the robot's stability and power consumption. The fourth measure, *Tangential Stress*, [42] is used as proposed. It measures the stress caused by the inertial forces acting on the robot in the direction of travel. The inertial forces on the robot are caused by acceleration and deceleration that occurs due to variations in trajectory. The *Tangential Stress* can be calculated by integrating the inertial forces along the trajectory. If the instantaneous linear acceleration is *a(t)*, considering a unit mass, the *Tangential Stress* (TS) can be calculated as in the equation 5.2.

$$TS = \int_{0}^{t_F} |a(t)| \, dx \tag{5.2}$$

We calculate *Tangential Stress* directly as the stress on the robot in the direction of travel during an *exploration task* for single robot systems. For multi-robot systems, we calculate *Tangential Stress* on the system level as an accumulated value and average stress on a robot in the direction of travel during an *exploration task* while coordinating with other robots.

If an exploration system has a lower *Tangential Stress* value, the robot has higher stability. When considering multi-robot systems, having a low *Tangential Stress* standard deviation indicates having a better coordination strategy because that means all robots have traveled on smooth and equally long trajectories.

5.1.5 Explored Percentage

The last measure, *Explored Percentage*, indicates the completeness of the autonomous exploration task compared to a manually controlled exploration using the same robot and same sensors. *Explored Percentage* is calculated as the ratio between the *Explored volume* and the *Explorable volume. Explored volume* refers to the volume of space the exploration system managed to explore during an autonomous exploration task, and *Explorable volume* refers to the volume of space that can be discovered by navigating the robot manually. If the *Explored volume* is v and the *Explorable volume* is *V*, the *Explored Percentage (EP)* can be calculated as in the equation 5.3

$$EP = \frac{v*100}{V} \tag{5.3}$$

For single robot systems, we calculate *Explored Percentage* directly on the robot. For multi-robot systems, we calculate *Explored Percentage* on the server system as it allows us to measure the *Explored Percentage* value without being affected by changing dimensions due to server goal updates. When considering this measure, exploration systems with a higher *Explored Percentage* value perform better.

5.2 Single Robot Experiments and Results

5.2.1 Explore-Lite System

Explore-Lite [27] implements greedy frontier-based exploration and is readily available as a ROS package. This system has been used as a system that implements frontier-based exploration [22] by many experiments that include testing reward functions for reinforcement learning-based SLAM systems [43], creating generative models for switching between SLAM map and CAD maps [44], map merging in robot swarms [45].

In addition to the experiments, one team of the 2021 RoboCup Rescue Simulation Virtual Robot Competition has proposed using Explore-Lite to rescue victims in disaster areas [46]. It has also been used by the ATR Kent team from Kent State University, USA, for RoboCup Rescue 2019 TDP Virtual Robot Simulation competition for rescuing victims from disaster areas using a heterogeneous multi-robot system [47] that includes ground robots and drones.

We compare the performance of our single robot system against the Explore-Lite system to understand the capabilities of our system.

5.2.2 Experimental Setup

We conducted experiments using ROS and Gazebo physics simulator on a computer with a 4th generation Intel i7 processor with 2.60 GHz base clock speed and 16 GB RAM. We used a simulated model of a Turtlebot with a mounted Microsoft Kinect with an active sensing range of 3 meters to evaluate the single robot system. Figure 5.1 illustrates the simulation environment, consisting of basic objects such as cubes, spheres, and cylinders in a 6 meter x 6 meter square area. We evaluated the center 5 meter x 5 meter area with up to a 0.5 meter height from the ground. All basic objects present in the evaluation space accounts for about 1.6 square meters or 13% of the evaluation space.

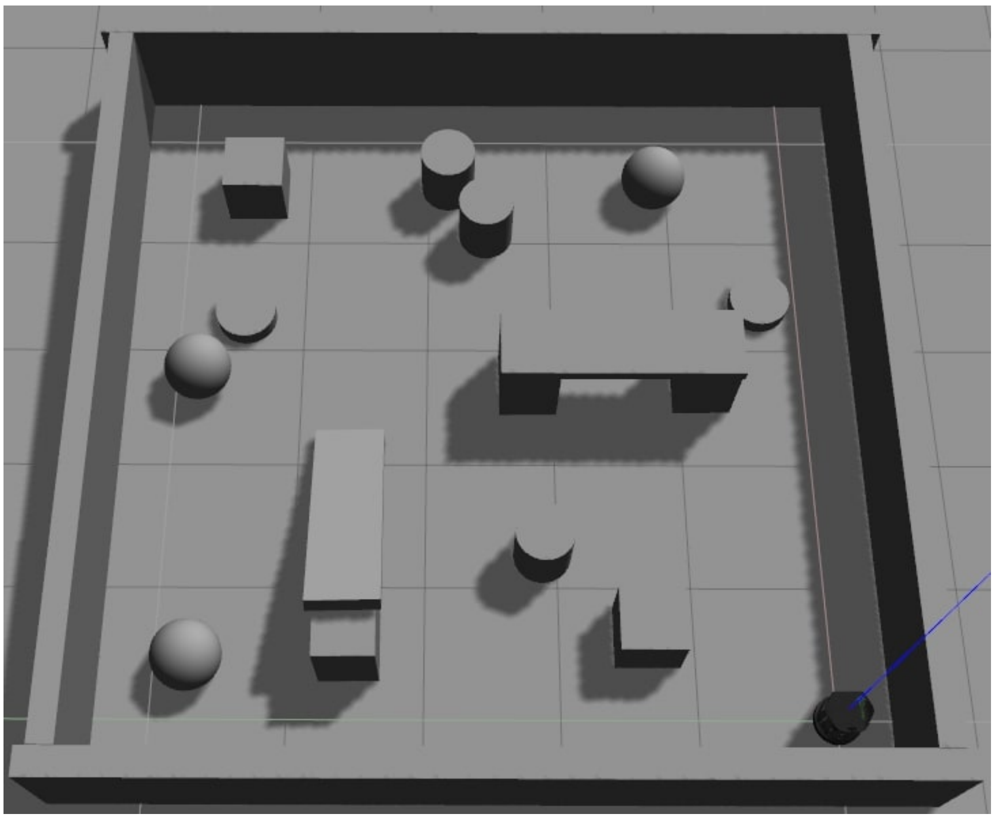


Figure 5.1: The simulated environment used for a single robot experiment

Figure 5.2 shows several instances of a single robot exploration experiment. Figure 5.2a shows the beginning of the exploration and Figure 5.2f shows the completion of exploration.

5.2.3 Experiment

We implemented *Planner Module*, *Exploration Module*, and *Control Module* proposed in this research as separate ROS packages that can be configured and

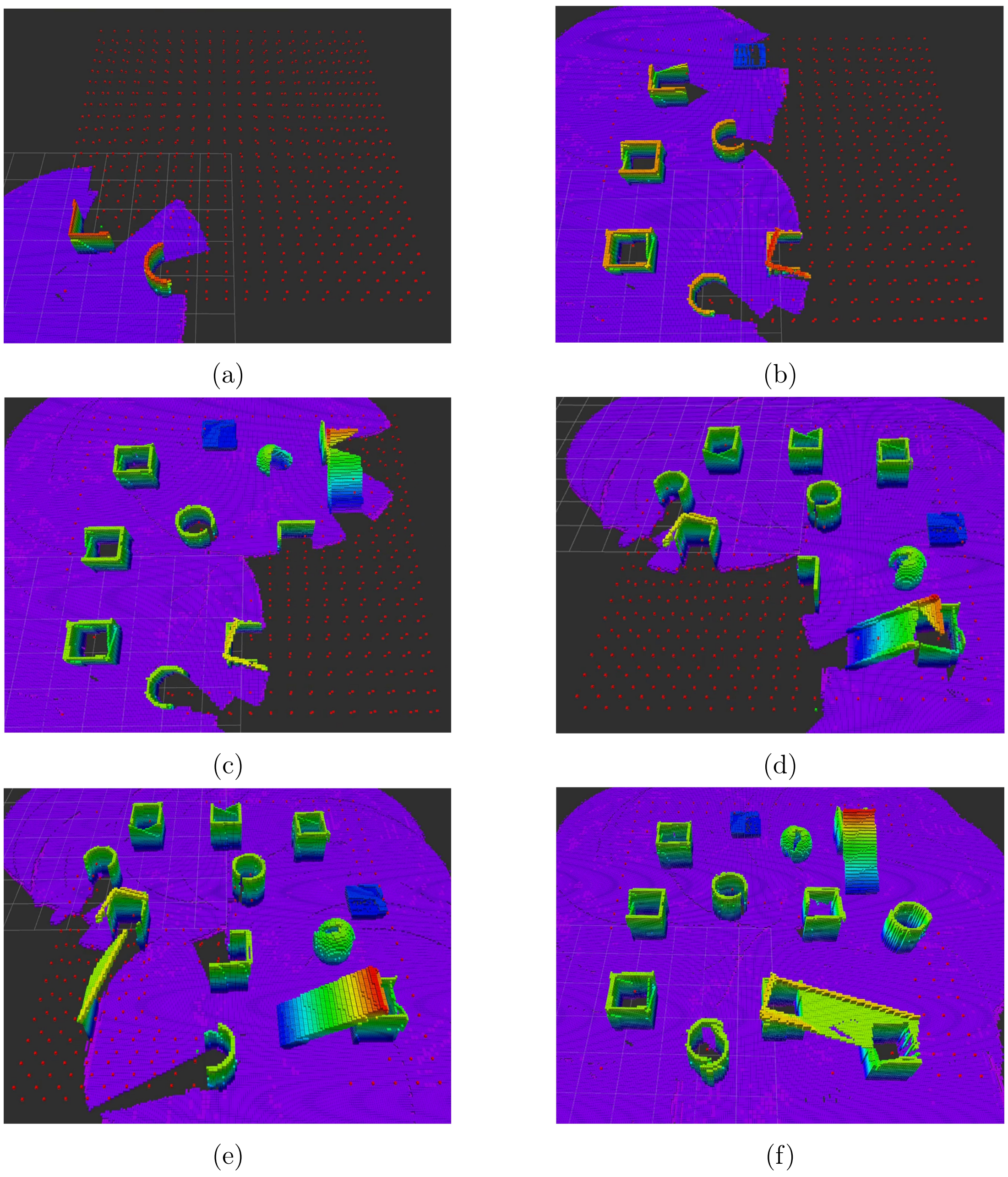


Figure 5.2: Single robot exploration during an experiment

launch via a master ROS package. For the experiments, we use Gazebo Physics Simulator-based simulations. Data required for the performance evaluation, such as octomap, position, velocity, and acceleration, were gathered from sensors on the Turtlebot robot through 7 experiments.

We used the Explore-Lite system [27] combined with ROS navigation stack as the comparison system against which we compared our performance. For the experiments, we again used Gazebo Physics Simulator-based simulations. Data required for the performance evaluation, such as octomap, position, velocity, and acceleration, were gathered from sensors on the Turtlebot robot through 7 experiments.

We used a manually teleoperated robot in the same simulation environment to identify the maximum area that could be mapped. Octomap data required for area calculation was gathered from sensors available on the Turtlebot robot through 3 experiments.

We have summarized the results of the experiments in the form of Mean and Standard Deviation in Table 5.1. *Time to explore*, *Total Path Length*, *Lateral Stress*, *Tangential Stress* values have been calculated from the results of the experiments, while the *Explored Percentage* was calculated using the results from the experiments and results from the manual operation of the robot.

Table 5.1: Single robot system evaluation results

Measure	Explore-Lite		Single robot	
(unit)	Mean	SD	Mean	SD
Explored Percentage	94.28	1.82	94.49	0.94
Time to explore (s)	1377.39	414.60	855.02	51.03
Total Path Length (m)	10.27	2.44	6.45	0.43
Lateral Stress (Ns)	2.54	0.51	0.06	0.01
Tangential Stress (Ns)	394.51	153.27	96.04	6.69

5.2.4 Result Analysis

All the results we collected in these experiments are dependant on the simulation environment variables such as sensor range, the physical size of the region, the density of static obstacles. Our selected physical size of the area is 1.67 times the range of the sensor. According to the results in Table 5.1, both systems have almost similar *Explored Percentage* mean values. Respective low standard deviation values mean that the variation of *Explored Percentage* values in individual experiments is low. These facts indicate that the ability to explore is similar in both systems.

Hence other factors such as efficiency and stability need to be considered when evaluating the systems. *Total Path Length*, *Lateral Stress*, and *Lateral Stress* measures help evaluate the robot's efficiency and stability. The Explore-Lite system has higher mean values for both *Lateral Stress* and *Tangential Stress* than the proposed single robot system, which means the proposed system is more efficient and stable. The respective higher standard deviation shows that even under the same environmental conditions, the Explore-Lite system tends to navigate inconsistently compared to the proposed system when efficiency and stability are concerned. *Total Path Length* value mirrors the efficiency of the system in terms of travel distance. Explore-Lite system has traveled more distance than the proposed systems, which means that the proposed system is capable of more efficient exploration.

Time to explore value shows the time a system takes to explore the map. Explore-Lite system has a high *Time to explore* mean value and a considerably sizeable standard deviation that shows that experiments had varying completion times. The proposed system has a low *Time to explore* mean and a standard deviation compared to the Explore-Lite system, which shows that it is faster and had an almost similar exploration throughout experiments compared to the Explore-Lite system.

Figures 5.3 & 5.4 contain comparisons between the Explore-Lite and proposed systems using *Explored Percentage* and *Time to explore* values.

Figure 5.3 shows the experiments with the fastest explorations of both systems. The proposed system managed to complete the exploration with an *Explored Percentage* of 94% in 783 seconds, and the Explore-Lite system has managed to complete the exploration with an *Explored Percentage* of 91% in 828 seconds. After about 500 seconds from the start, the proposed system has achieved an *Explored Percentage* of about 88%, and Explore-Lite has achieved about 78%.

Figure 5.4 shows the experiments with the highest *Explored Percentage* of both systems. The proposed system managed to complete the exploration with an *Explored Percentage* of 95% in 929 seconds, and the Explore-Lite system has managed to complete the exploration with an *Explored Percentage* of 97% in 2154

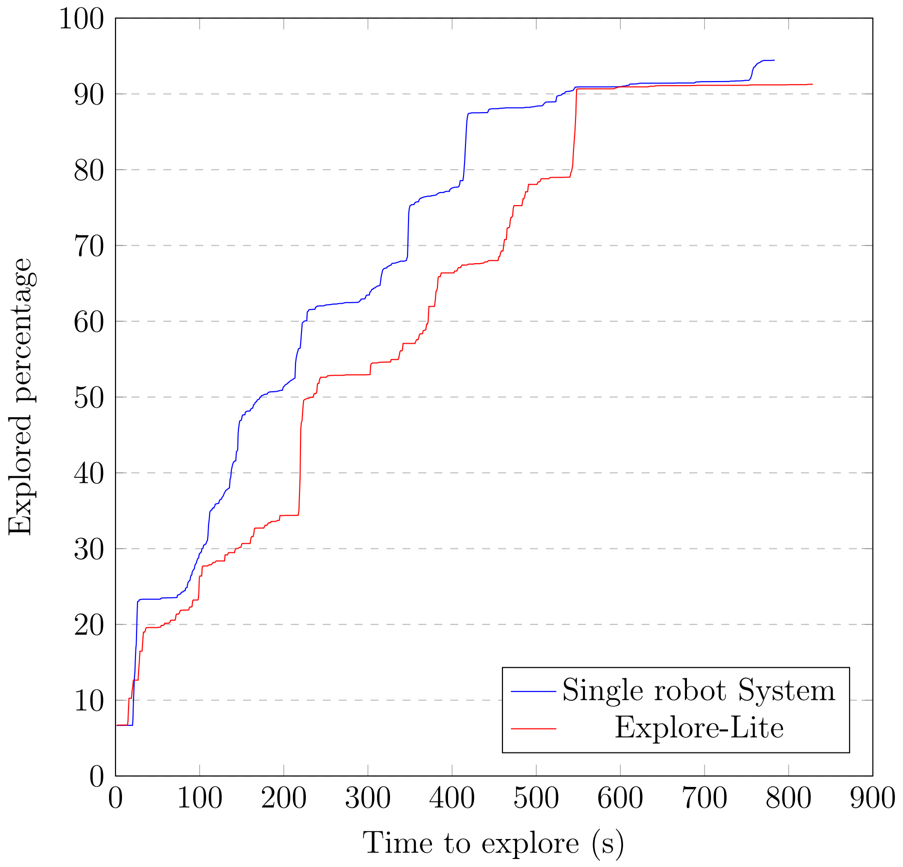


Figure 5.3: Comparison of fastest explorations of the single robot system and Explore-Lite system

seconds. After about 930 seconds from the start (when the proposed systems completed exploration), the Explore-Lite system has achieved an *Explored Percentage* of about 77%, and after about 1900 seconds from the start is has achieved an *Explored Percentage* of about 95% (highest *Explored Percentage* proposed system managed to achieve).

5.3 Multi-Robot Experiments and Results

5.3.1 Experiment Setup

We conducted experiments using ROS and Gazebo physics simulator on a computer with an 8th generation Intel i7 processor with 3.20 GHz base clock speed and 64 GB RAM. Simulated models of Turtlebot, each with a mounted Microsoft Kinect with an active sensor range of 3 meters, were used for evaluations of the

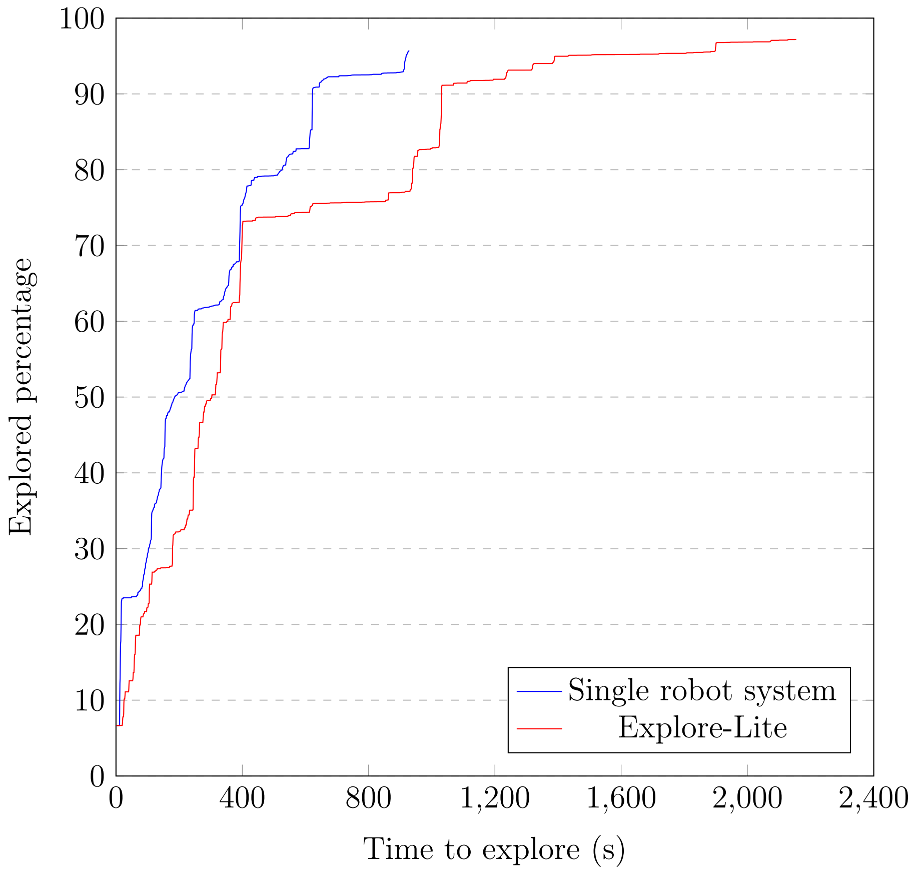


Figure 5.4: Comparison of explorations with highest *Explored Percentage* of the single robot system and Explore-Lite system

multi-robot system. There were three experiments, one robot experiment, two robot experiments, and three robot experiments where all three experiments used the same environment. One robot experiment was conducted as a controlled experiment without the server system, which enabled us to compare the two robot and three robot systems against it to calculate the performance gain. The simulation environment was a 10 meter x 10 meter square area. We evaluated the whole 10 meter x 10 meter area with up to a 0.5 meter height from the ground. The evaluation space was sparsely populated with static obstacles while the robots in the two robots and three robots experiments acted as dynamic obstacles to the other robots present in each experiment.

5.3.2 One Robot Experiment

We used ROS packages containing *Planner Module*, *Exploration Module* and *Control Module* along with the master ROS package used for configuration and launching the system. For this experiment, we use Gazebo Physics Simulator-based simulations. The simulation setup is available in Figure 5.5. Data required for the performance evaluation, such as octomap, position, velocity, and acceleration, were gathered from the Turtlebot robot's sensors through 5 experiments.

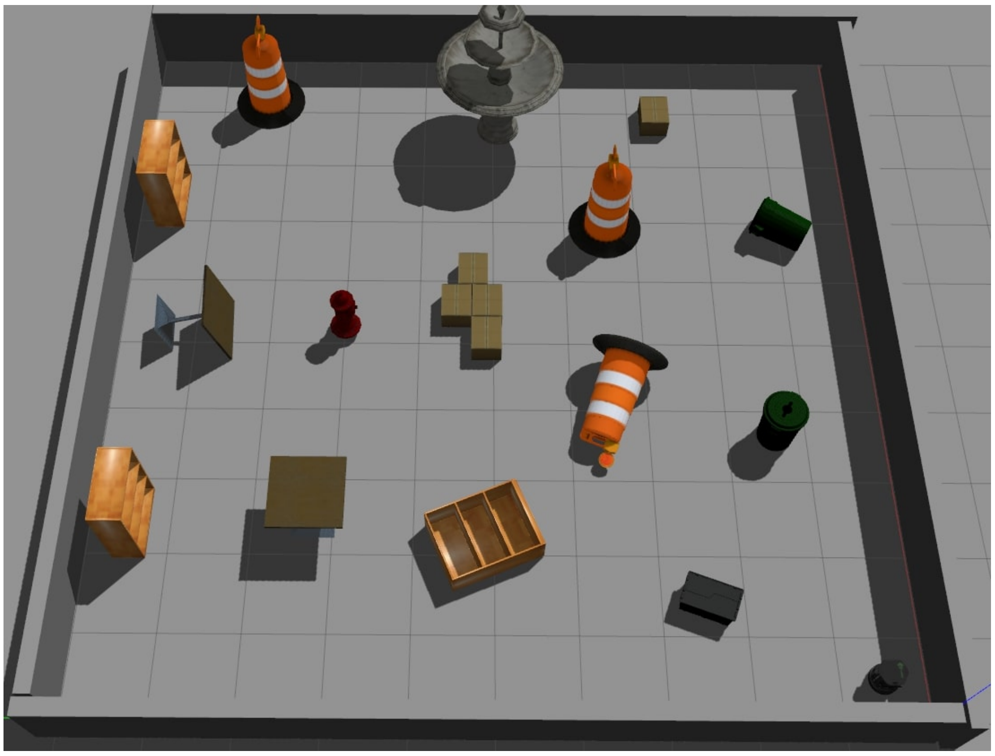


Figure 5.5: The simulated environment used for one robot experiment

We used a manually teleoperated robot in the same simulation environment to identify the maximum area that can be mapped. Octomap data required for the area calculation was gathered from sensors on the Turtlebot robot through 3 experiments. The *Explored Percentage* was calculated using the results from the experiment and the manual operation of the robot.

5.3.3 Two Robot Experiment

Server system and *Client Module* proposed in this research for multi-robot coordination has been implemented as a ROS package for *Coordination*. ROS packages containing *Planner Module*, *Exploration Module* and *Control Module* along with *Client Module* ROS node from *Coordination* ROS package were used along with the master ROS package for configuration and launching two instances of the Robot system. *Server System* ROS node from *Coordination* ROS package was used along with the master ROS package for configuration and launching the Server system. For this experiment, we use Gazebo Physics Simulator-based simulations. The simulation setup is available in Figure 5.6. Data required for the performance evaluation, such as octomap, position, velocity, and acceleration, were gathered from sensors available on the Turtlebot robot through 5 experiments.

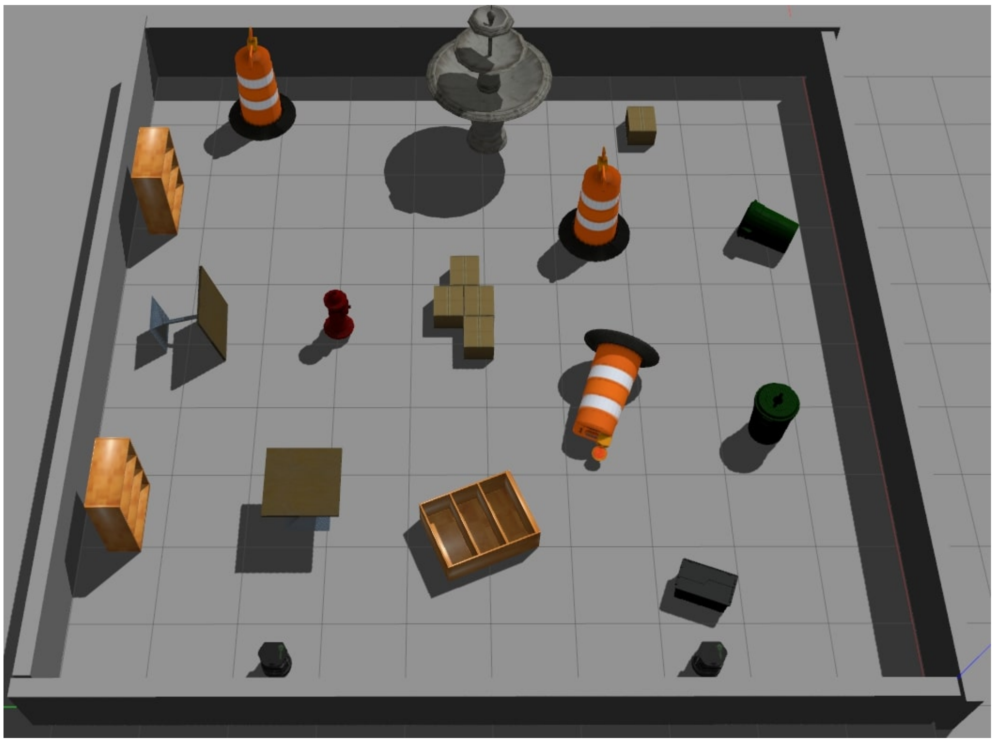


Figure 5.6: The simulated environment used for two robot experiment

Figure 5.7 shows several instances of a two robot exploration experiment. Figure 5.7a shows the beginning of the exploration and Figure 5.7f shows the completion of exploration.

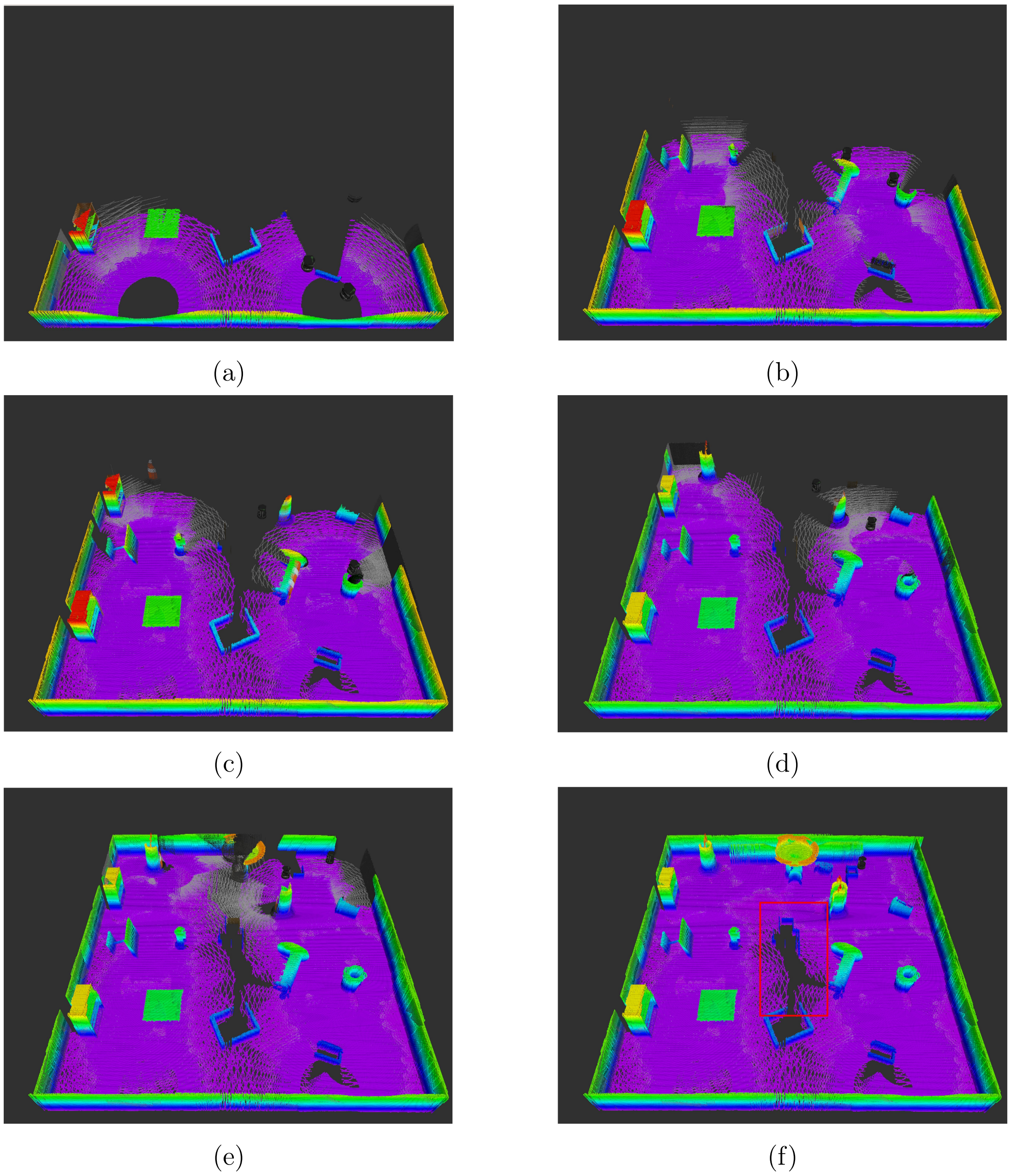


Figure 5.7: Two robot exploration during an experiment

In the shown experiment, we observed an empty area in the middle of the octomap that has been highlighted in a red rectangle in Figure 5.7f. Though the floor has not been identified, the octomap evaluation showed that the area above the floor is free space. This did not occur in all the experiments.

5.3.4 Three Robot Experiment

ROS packages containing *Planner Module*, *Exploration Module* and *Control Module* along with *Client Module* ROS node from *Coordination* ROS package were used along with the master ROS package for configuration and launching three instances of the robot system. *Server system* ROS node from *Coordination* ROS package was used along with the master ROS package for configuration and launching the Server system. For this experiment, we use Gazebo Physics Simulator-based simulations. The simulation setup is available in Figure 5.8. Data required for the performance evaluation, such as octomap, position, velocity, and acceleration, were gathered from sensors available on the Turtlebot robot through 5 experiments.

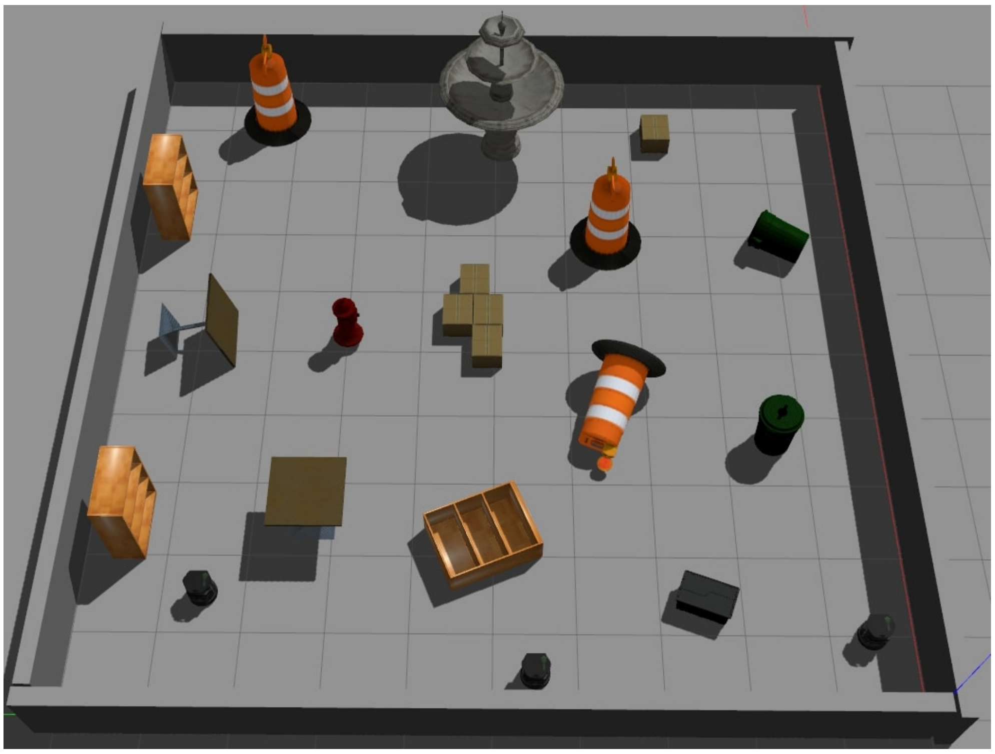


Figure 5.8: The simulated environment used for three robot experiment

Figure 5.9 shows several instances of a three robot exploration experiment. Figure 5.9a shows the beginning of the exploration and Figure 5.9f shows the completion of exploration.

Results of the experiments have been summarized in the form of Mean and

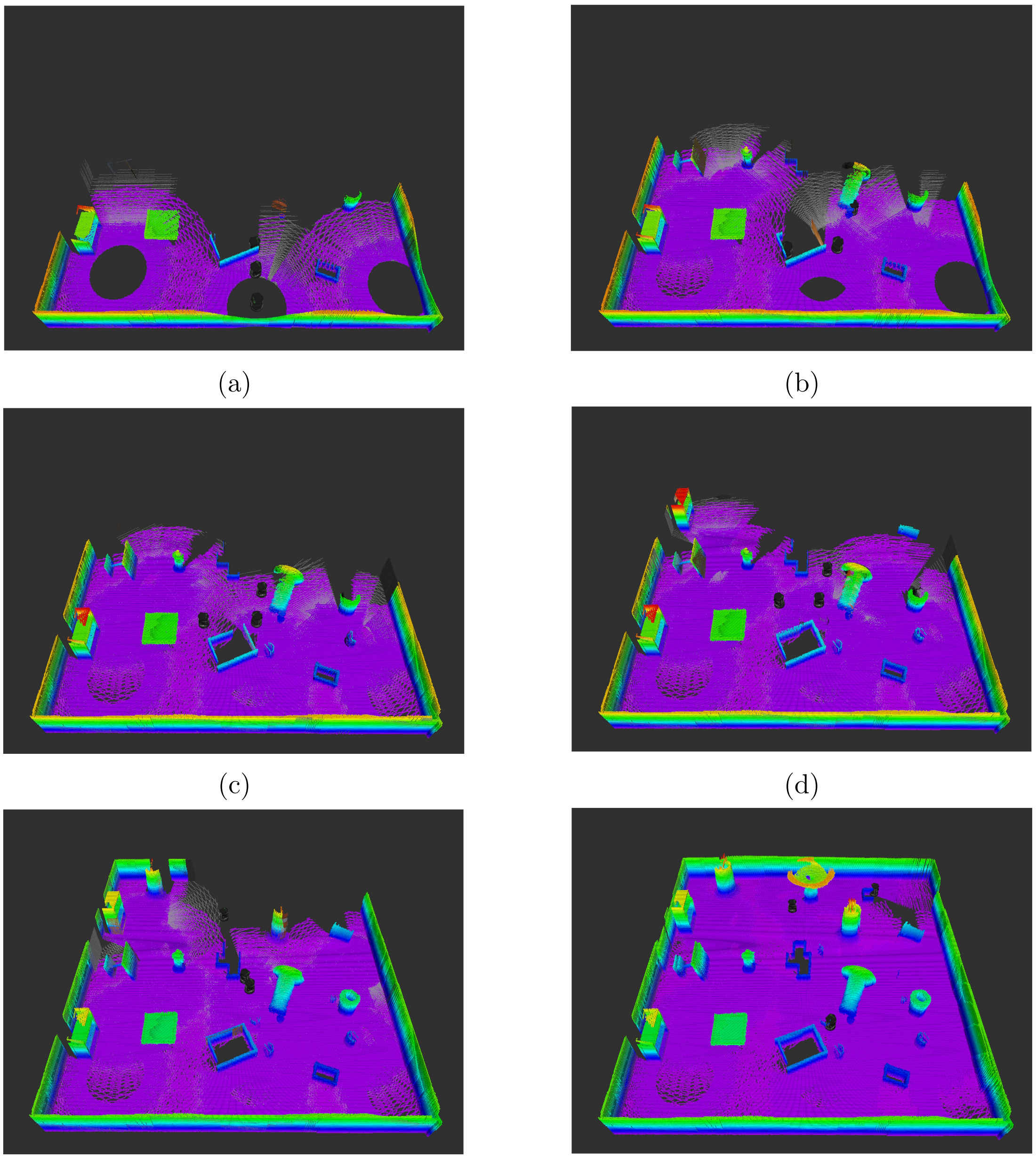

(a) (b) (c) (d) (e) (f)

Figure 5.9: Three robot exploration during an experiment

Standard Deviation in Table 5.2 and Table 5.3.

Explored Percentage and *Time to explore* needs to be measured and evaluated on the system level since they contain aspects that relate to coordination and cannot be measured from individual robots when considering two robot and three robot systems. The Table 5.2 contains comparisons on system-level where *Explored Percentage*, *Time to explore* has been calculated at the robot level in

Table 5.2: Multi robot system - system level evaluation results

Measure	One Robot		Two robots		Three robots	
(unit)	Mean	SD	Mean	SD	Mean	SD
Explored Percentage	98.21	1.70	96.92	2.71	99.17	0.24
Time to explore (s)	1294.84	169.46	672.62	72.49	807.66	91.26
Total Path Length (m)	40.29	4.80	29.97	3.72	36.60	4.49
Lateral Stress (Ns)	0.29	0.03	0.44	0.19	0.82	0.63
Tangential Stress (Ns)	250.39	81.21	209.91	65.30	326.63	70.89

one robot system and server level in two and three robot systems. *Total Path Length*, *Lateral Stress*, and *Tangential Stress* in Table 5.2 are collected from each robot and represented as an accumulated value in system-level comparisons.

Table 5.3: Multi robot system - robot level evaluation results

Measure	One Robot		Two robots		Three robots	
(unit)	Mean	SD	Mean	SD	Mean	SD
Total Path Length (m)	40.29	4.80	14.98	1.86	12.20	1.50
Lateral Stress (Ns)	0.29	0.03	0.22	0.09	0.27	0.21
Tangential Stress (Ns)	250.39	81.20	104.95	32.65	108.88	23.63

Since *Total Path Length*, *Lateral Stress*, *Tangential Stress* values are calculated from robot level, there are variations between values gathered from the robots in the same system. Because of that reason, we cannot make further claims on these using system-level comparisons of *Total Path Length*, *Lateral Stress*, *Tangential Stress* values. Therefore we have presented robot level values for these measures in Table 5.3. We have not included *Explored Percentage* and *Time to explore* values in Table 5.3 because those values are calculated only from the system level and cannot be related to robot level performance. The values presented in Table 5.3 are mean and standard deviation values calculated using averaged values based on the number of robots in each experiment.

5.3.5 Results Analysis

All the results we collected in these experiments are dependant on the simulation environment variables such as sensor range, the physical size of the region, the density of static obstacles, and availability of dynamic obstacles. Our selected physical size of the area is 3.34 times the range of the sensor. According to the results in the Table 5.2, three robot systems got the highest *Explored Percentage* mean and the lowest standard deviation values of all experiments. Two robot systems got the least mean and highest standard deviation among the experiments. Considering that all systems had *Explored Percentage* mean values, which are about 1% apart, we can conclude that all three systems perform well when considering the exploration aspect.

Time to explore value shows the time a system takes to explore the map. one robot system has a high *Time to explore* the mean value and a considerably sizeable standard deviation compared to the other two systems. One robot system has, on average, completed the explorations at about 1295 seconds. Two robot systems have taken 673 seconds approximately, which means it is 48% faster than the one robot system. Three robot systems have taken 808 seconds approximately, which means it is 38% faster than the one robot system. The three robot system was 20% slower than the two robot system.

Total Path Length value mirrors the efficiency of the system in terms of travel distance. One robot system has moved the most distance compared to multi-systems amounting to a mean value of 40.29 meters. Robots of the two robot system have collectively moved about 30 meters which is about 25% less than one robot system. When considering robot level performance, a robot from two robot system has on average moved about 14.98 meters which is approximately a 63% reduction compared to one robot system. Similarly, robots of the three robot system have collectively moved about 36.6 meters which is about 10% less than one robot system. A robot from three robot system has, on average, moved about 12.20 meters which is approximately a 70% reduction compared to one robot system. Even though the robots of the three robot system has collectively traveled 22% more than the robots of the two robot system, a robot from the three robot system has moved approximately 19% less than a robot from the two robot system.

On the system level, three robot system has the highest *Lateral Stress* and *Tangential Stress* values of all systems. However, a more meaningful robot level comparison on Table 5.3 shows that One robot system has the highest value for both *Lateral Stress* and *Tangential Stress* compared to other systems, which means exploring the area alone causes the robot to operate under higher stress than multi-robot systems. A robot in two robot system experiences 25% less lateral stress and 58% less tangential stress than the robot that operates alone. A robot in three robot system experiences 7% less lateral stress and 57% less tangential stress than the robot that operates individually. A robot from three robot systems experiences 24% more lateral stress and 4% more tangential stress than a robot that belongs to two robot system.

Figure 5.10 shows the experiments with the fastest explorations of all systems. One Robot system has completed the exploration with an *Explored Percentage* of 95% in 1026 seconds. Two Robot system has completed the exploration with an *Explored Percentage* of 96% in 577 seconds. Three Robot system has completed the exploration with an *Explored Percentage* of 99% in 715 seconds. After about 580 seconds from the start, when the two robot system completes the exploration, one robot system has achieved an *Explored Percentage* of about 66%, and three robot system has achieved about 90%.

Figure 5.11 shows the experiments with the highest *Explored Percentage* of all systems. one robot system completed the exploration with an *Explored Percentage* of 99% in 1495 seconds. Two robot system also completed the exploration with an *Explored Percentage* of 99% in 690 seconds. Three Robot system completed the exploration with an *Explored Percentage* of 99% in 973 seconds. After about 690 seconds from the start, when the two robot system completed exploration, one robot system achieved an *Explored Percentage* of about 80%, and three robot system achieved an *Explored Percentage* of about 82%.

5.4 Discussion

Comparisons in Table 5.1, Figure 5.3, Figure 5.4 prove that the proposed single robot system is faster, more stable, and more efficient than the Explore-Lite system [27] available as a greedy frontier-based exploration system.

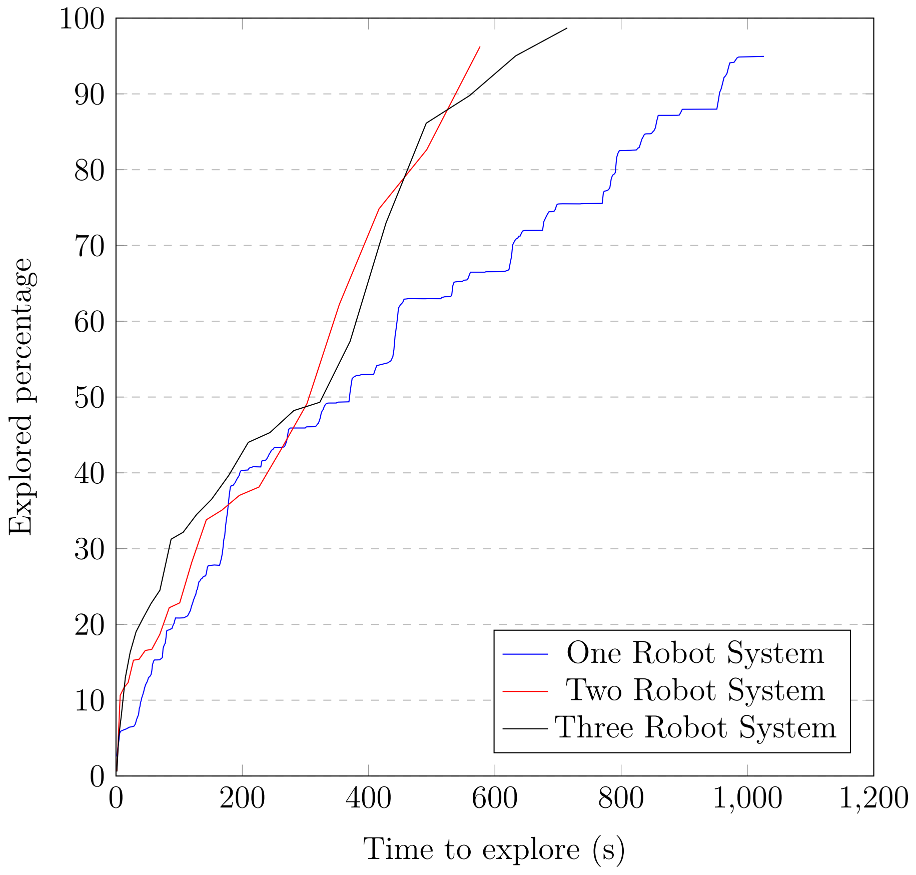


Figure 5.10: Comparison of fastest explorations of one robot system, two robot system, and three robot system

Results analysis and data in Table 5.2, Table 5.3, Figure ,5.10 and Figure 5.11 prove that multi-robot systems are faster, more stable, and more efficient than one robot system during simulations. Reducing trends in values such as *Total Path Length* and *Time to explore* proves that our proposed multi-robot system design is **effective** and **successful**.

When considering system-level and robot levels comparisons between two robot and three robot systems, even though both perform better than one robot system, we can see a performance decrease in the three robot system compared to the two robot system. This performance decrease can either be due to a resource limit in the computer used for simulations as shown in Figure 5.12 or due to two robot system being more effective against the selected environment type and size than three robot system.

Figure 5.12 contains three screen captures taken during each system simula-

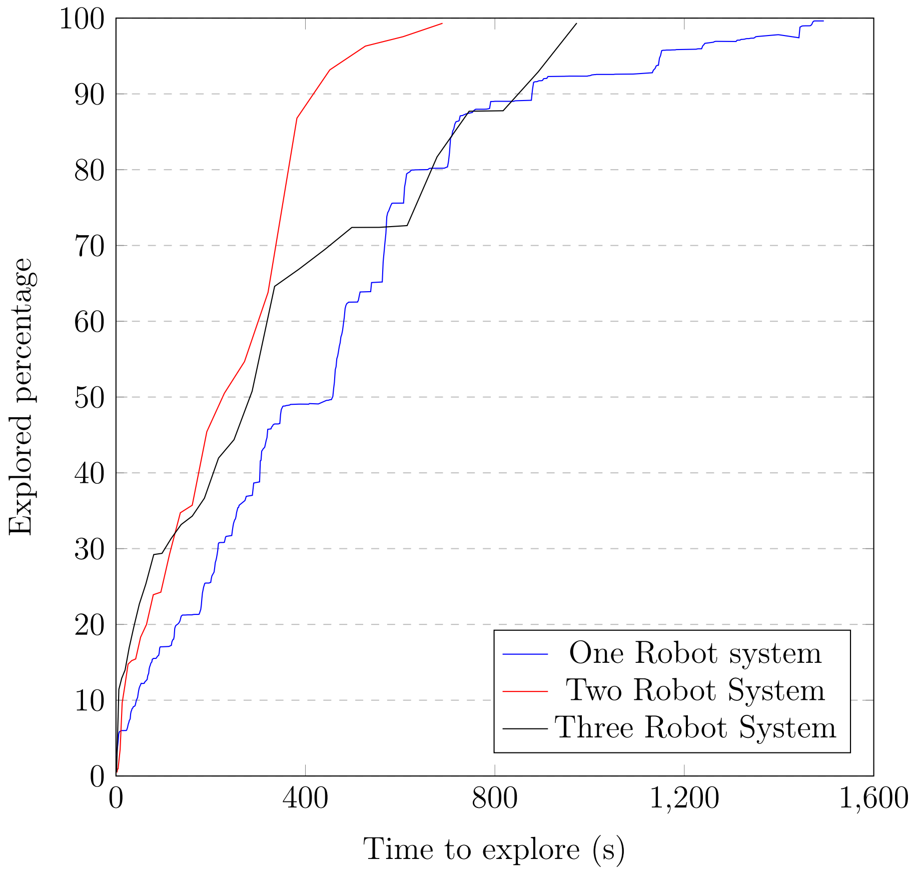
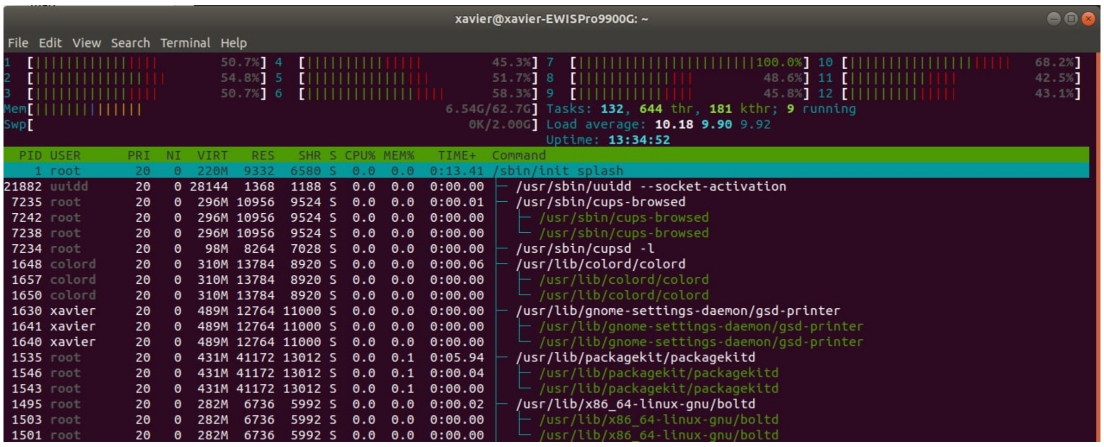


Figure 5.11: Comparison of explorations with highest *Explored Percentage* of one robot system, two robot system, and three robot system

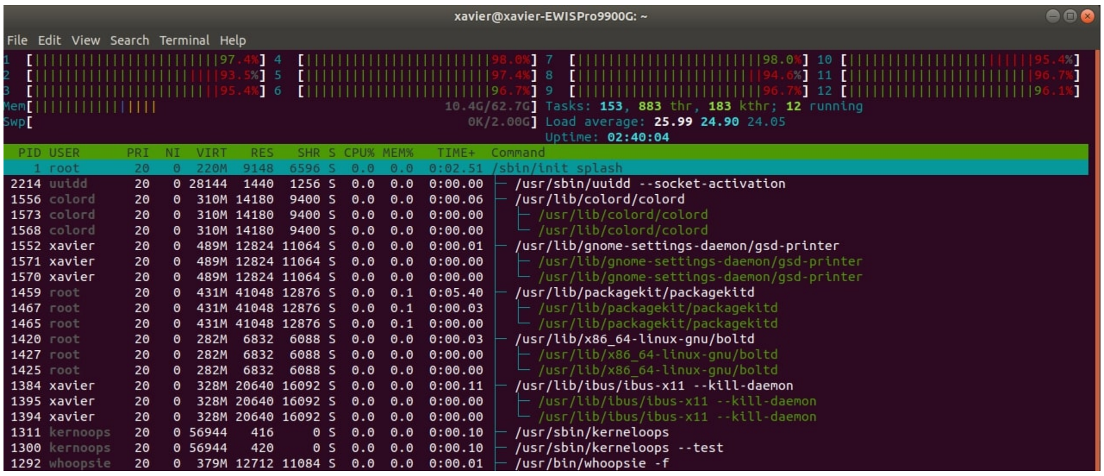
tion, and it shows resource utilization by ROS and Gazebo simulator during each simulation. Compared to one robot system utilization in Figure 5.12a, two and three robot systems utilize much more resources, as illustrated in Figure 5.12b and Figure 5.12c. Comparison between two and three robot systems reveals that three robot system might lack the resources it needs to perform at total capacity. Such lack of resources can prolong the exploration calculations and cause inconsistent coordination of 3 robots, reducing the *Total Path Length* improvement and increasing *Lateral Stress* and *Tangential Stress* on robots and *Time to explore* value of the system.

The reason two robot system being effective than three robot system can be supported by the fact that our simulated environment having a physical area size to sensor range ratio of 3.34 : 1, which means that the whole area can be covered with two robots due to the rotational action available in our system with


xavier@xavier-EWISPro9900G: ~
File Edit View Search Terminal Help

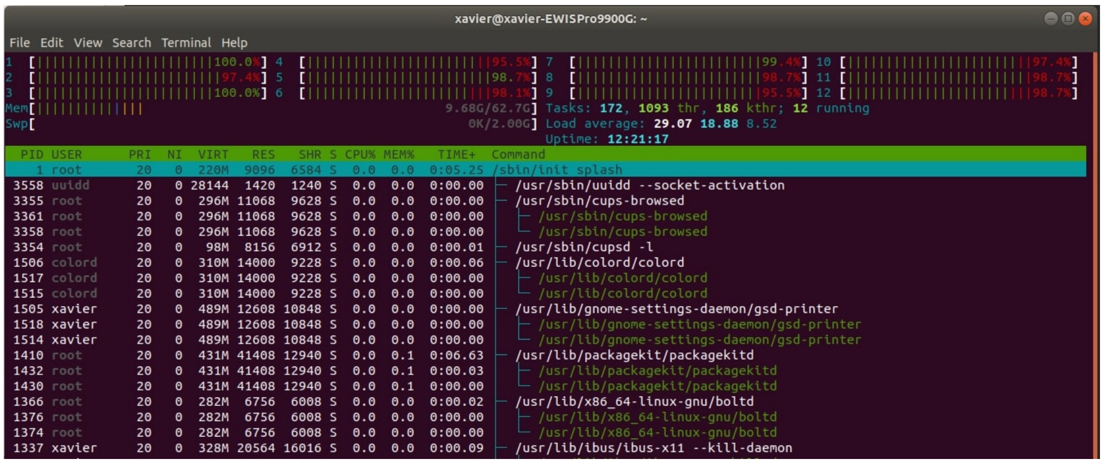
```
1  [||||||||||||||||       50.7%] 4  [|||||||||||||||          45.3%] 7  [||||||||||||||||||||||||100.0%] 10 [|||||||||||||||||||||      68.2%]
2  [|||||||||||||||||      54.8%] 5  [||||||||||||||||         51.7%] 8  [||||||||||||||||          48.6%] 11 [|||||||||||||||            42.5%]
3  [||||||||||||||||       50.7%] 6  [|||||||||||||||||||      58.3%] 9  [||||||||||||||||          45.8%] 12 [|||||||||||||||            43.1%]
Mem[||||||||||||||                                  6.54G/62.7G] Tasks: 132, 644 thr, 181 kthr; 9 running
Swp[                                                  0K/2.00G] Load average: 10.18 9.90 9.92
                                                                Uptime: 13:34:52
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
    1 root       20   0  220M  9332  6580 S  0.0  0.0  0:13.41 /sbin/init splash
21882 uuidd      20   0 28144  1368  1188 S  0.0  0.0  0:00.00 ├─ /usr/sbin/uuidd --socket-activation
 7235 root       20   0  296M 10956  9524 S  0.0  0.0  0:00.01 ├─ /usr/sbin/cups-browsed
 7242 root       20   0  296M 10956  9524 S  0.0  0.0  0:00.00 │  ├─ /usr/sbin/cups-browsed
 7238 root       20   0  296M 10956  9524 S  0.0  0.0  0:00.00 │  └─ /usr/sbin/cups-browsed
 7234 root       20   0   98M  8264  7028 S  0.0  0.0  0:00.00 ├─ /usr/sbin/cupsd -l
 1648 colord     20   0  310M 13784  8920 S  0.0  0.0  0:00.06 ├─ /usr/lib/colord/colord
 1657 colord     20   0  310M 13784  8920 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/colord/colord
 1650 colord     20   0  310M 13784  8920 S  0.0  0.0  0:00.00 │  └─ /usr/lib/colord/colord
 1630 xavier     20   0  489M 12764 11000 S  0.0  0.0  0:00.00 ├─ /usr/lib/gnome-settings-daemon/gsd-printer
 1641 xavier     20   0  489M 12764 11000 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/gnome-settings-daemon/gsd-printer
 1640 xavier     20   0  489M 12764 11000 S  0.0  0.0  0:00.00 │  └─ /usr/lib/gnome-settings-daemon/gsd-printer
 1535 root       20   0  431M 41172 13012 S  0.0  0.1  0:05.94 ├─ /usr/lib/packagekit/packagekitd
 1546 root       20   0  431M 41172 13012 S  0.0  0.1  0:00.04 │  ├─ /usr/lib/packagekit/packagekitd
 1543 root       20   0  431M 41172 13012 S  0.0  0.1  0:00.00 │  └─ /usr/lib/packagekit/packagekitd
 1495 root       20   0  282M  6736  5992 S  0.0  0.0  0:00.02 ├─ /usr/lib/x86_64-linux-gnu/boltd
 1503 root       20   0  282M  6736  5992 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/x86_64-linux-gnu/boltd
 1501 root       20   0  282M  6736  5992 S  0.0  0.0  0:00.00 │  └─ /usr/lib/x86_64-linux-gnu/boltd
```

(a)


xavier@xavier-EWISPro9900G: ~
File Edit View Search Terminal Help

```
1  [||||||||||||||||||||||||97.4%] 4  [||||||||||||||||||||||||98.0%] 7  [||||||||||||||||||||||||98.0%] 10 [||||||||||||||||||||||||95.4%]
2  [||||||||||||||||||||||||93.5%] 5  [||||||||||||||||||||||||97.4%] 8  [||||||||||||||||||||||||94.6%] 11 [||||||||||||||||||||||||96.7%]
3  [||||||||||||||||||||||||95.4%] 6  [||||||||||||||||||||||||96.7%] 9  [||||||||||||||||||||||||96.7%] 12 [||||||||||||||||||||||||96.1%]
Mem[|||||||||||||||||                               10.4G/62.7G] Tasks: 153, 883 thr, 183 kthr; 12 running
Swp[                                                  0K/2.00G] Load average: 25.99 24.90 24.05
                                                                Uptime: 02:40:04
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
    1 root       20   0  220M  9148  6596 S  0.0  0.0  0:02.51 /sbin/init splash
 2214 uuidd      20   0 28144  1440  1256 S  0.0  0.0  0:00.00 ├─ /usr/sbin/uuidd --socket-activation
 1556 colord     20   0  310M 14180  9400 S  0.0  0.0  0:00.06 ├─ /usr/lib/colord/colord
 1573 colord     20   0  310M 14180  9400 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/colord/colord
 1568 colord     20   0  310M 14180  9400 S  0.0  0.0  0:00.00 │  └─ /usr/lib/colord/colord
 1552 xavier     20   0  489M 12824 11064 S  0.0  0.0  0:00.01 ├─ /usr/lib/gnome-settings-daemon/gsd-printer
 1571 xavier     20   0  489M 12824 11064 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/gnome-settings-daemon/gsd-printer
 1570 xavier     20   0  489M 12824 11064 S  0.0  0.0  0:00.00 │  └─ /usr/lib/gnome-settings-daemon/gsd-printer
 1459 root       20   0  431M 41048 12876 S  0.0  0.1  0:05.40 ├─ /usr/lib/packagekit/packagekitd
 1467 root       20   0  431M 41048 12876 S  0.0  0.1  0:00.03 │  ├─ /usr/lib/packagekit/packagekitd
 1465 root       20   0  431M 41048 12876 S  0.0  0.1  0:00.00 │  └─ /usr/lib/packagekit/packagekitd
 1420 root       20   0  282M  6832  6088 S  0.0  0.0  0:00.03 ├─ /usr/lib/x86_64-linux-gnu/boltd
 1427 root       20   0  282M  6832  6088 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/x86_64-linux-gnu/boltd
 1425 root       20   0  282M  6832  6088 S  0.0  0.0  0:00.00 │  └─ /usr/lib/x86_64-linux-gnu/boltd
 1384 xavier     20   0  328M 20640 16092 S  0.0  0.0  0:00.11 ├─ /usr/lib/ibus/ibus-x11 --kill-daemon
 1395 xavier     20   0  328M 20640 16092 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/ibus/ibus-x11 --kill-daemon
 1394 xavier     20   0  328M 20640 16092 S  0.0  0.0  0:00.00 │  └─ /usr/lib/ibus/ibus-x11 --kill-daemon
 1311 kernoops   20   0 56944   416     0 S  0.0  0.0  0:00.10 ├─ /usr/sbin/kerneloops
 1300 kernoops   20   0 56944   420     0 S  0.0  0.0  0:00.10 ├─ /usr/sbin/kerneloops --test
 1292 whoopsie   20   0  379M 12712 11084 S  0.0  0.0  0:00.01 ├─ /usr/bin/whoopsie -f
```

(b)


xavier@xavier-EWISPro9900G: ~
File Edit View Search Terminal Help

```
1  [|||||||||||||||||||||||100.0%] 4  [||||||||||||||||||||||||95.5%] 7  [||||||||||||||||||||||||99.4%] 10 [||||||||||||||||||||||||97.4%]
2  [||||||||||||||||||||||||97.4%] 5  [||||||||||||||||||||||||98.7%] 8  [||||||||||||||||||||||||98.7%] 11 [||||||||||||||||||||||||98.7%]
3  [|||||||||||||||||||||||100.0%] 6  [||||||||||||||||||||||||98.1%] 9  [||||||||||||||||||||||||95.5%] 12 [||||||||||||||||||||||||98.7%]
Mem[|||||||||||||||                                 9.68G/62.7G] Tasks: 172, 1093 thr, 186 kthr; 12 running
Swp[                                                  0K/2.00G] Load average: 29.07 18.88 8.52
                                                                Uptime: 12:21:17
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
    1 root       20   0  220M  9096  6584 S  0.0  0.0  0:05.25 /sbin/init splash
 3558 uuidd      20   0 28144  1420  1240 S  0.0  0.0  0:00.00 ├─ /usr/sbin/uuidd --socket-activation
 3355 root       20   0  296M 11068  9628 S  0.0  0.0  0:00.00 ├─ /usr/sbin/cups-browsed
 3361 root       20   0  296M 11068  9628 S  0.0  0.0  0:00.00 │  ├─ /usr/sbin/cups-browsed
 3358 root       20   0  296M 11068  9628 S  0.0  0.0  0:00.00 │  └─ /usr/sbin/cups-browsed
 3354 root       20   0   98M  8156  6912 S  0.0  0.0  0:00.01 ├─ /usr/sbin/cupsd -l
 1506 colord     20   0  310M 14000  9228 S  0.0  0.0  0:00.06 ├─ /usr/lib/colord/colord
 1517 colord     20   0  310M 14000  9228 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/colord/colord
 1515 colord     20   0  310M 14000  9228 S  0.0  0.0  0:00.00 │  └─ /usr/lib/colord/colord
 1505 xavier     20   0  489M 12608 10848 S  0.0  0.0  0:00.00 ├─ /usr/lib/gnome-settings-daemon/gsd-printer
 1518 xavier     20   0  489M 12608 10848 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/gnome-settings-daemon/gsd-printer
 1514 xavier     20   0  489M 12608 10848 S  0.0  0.0  0:00.00 │  └─ /usr/lib/gnome-settings-daemon/gsd-printer
 1410 root       20   0  431M 41408 12940 S  0.0  0.1  0:06.63 ├─ /usr/lib/packagekit/packagekitd
 1432 root       20   0  431M 41408 12940 S  0.0  0.1  0:00.03 │  ├─ /usr/lib/packagekit/packagekitd
 1430 root       20   0  431M 41408 12940 S  0.0  0.1  0:00.00 │  └─ /usr/lib/packagekit/packagekitd
 1366 root       20   0  282M  6756  6008 S  0.0  0.0  0:00.02 ├─ /usr/lib/x86_64-linux-gnu/boltd
 1376 root       20   0  282M  6756  6008 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/x86_64-linux-gnu/boltd
 1374 root       20   0  282M  6756  6008 S  0.0  0.0  0:00.00 │  └─ /usr/lib/x86_64-linux-gnu/boltd
 1337 xavier     20   0  328M 20564 16016 S  0.0  0.0  0:00.09 ├─ /usr/lib/ibus/ibus-x11 --kill-daemon
```

(c)

Figure 5.12: Resource utilization of ROS and Gazebo simulator during (a) one robot system simulations (b) two robot system simulations (c) three robot system simulations at mid-way.

a minimum overlap of maps. When using three robots, the overlap of maps can become larger and overlapping exploration under local explore module (during communication intervals) can cause the extended time making three robot system less effective.

Another identified issue was the faulty calculations on the cluster evaluation stage of the *Server system* due to failures of individual *SLAM Systems*. These failures of the SLAM system could either be due to internal pose-graph calculation failures or inefficient parameter tuning. The failed SLAM systems returned distorted point clouds, which were transmitted to *Server System* and were used for map merging and cluster calculation, causing the calculations to be faulty. The Figure 5.13is one such instance where the far left corner of the map has failed to align correctly.

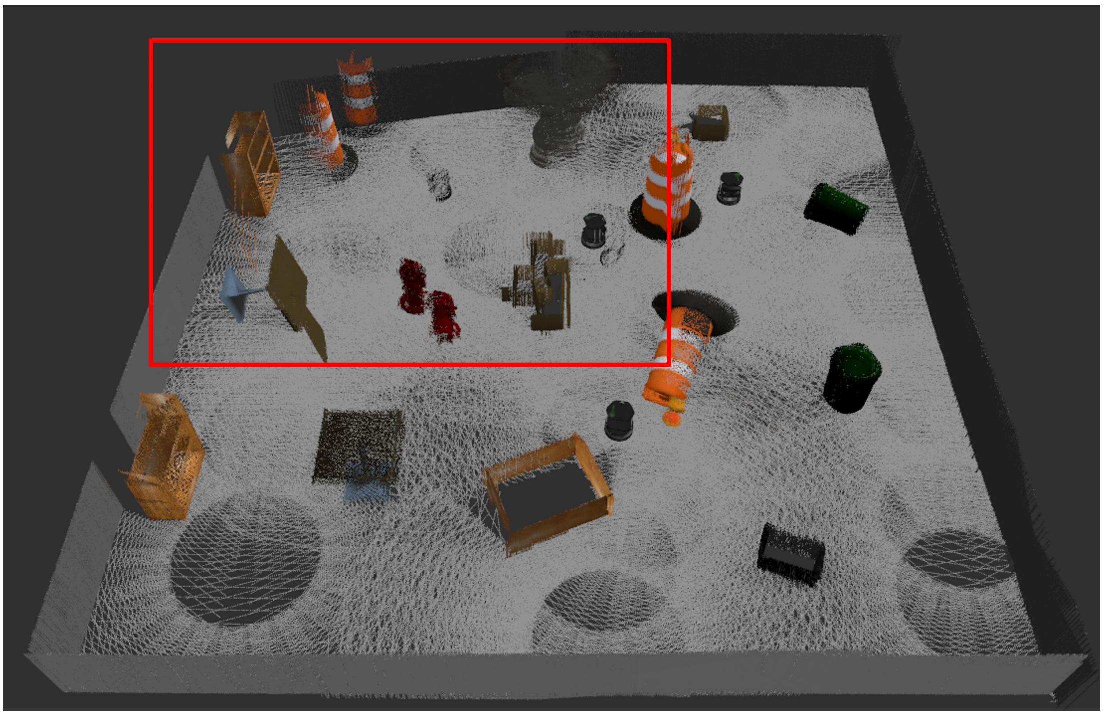
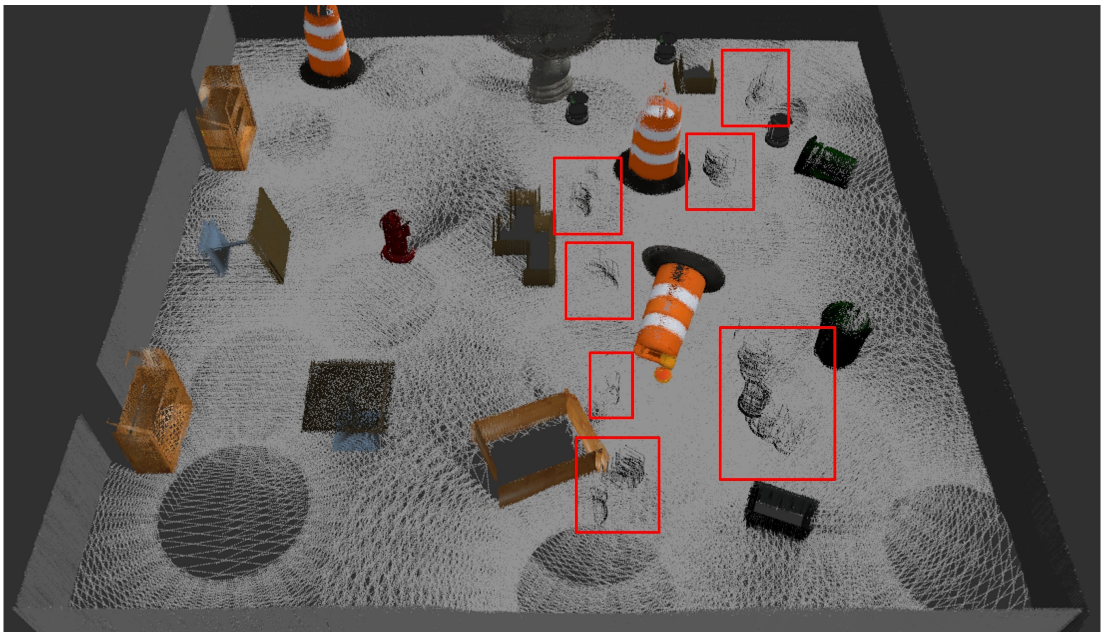


Figure 5.13: SLAM system failure

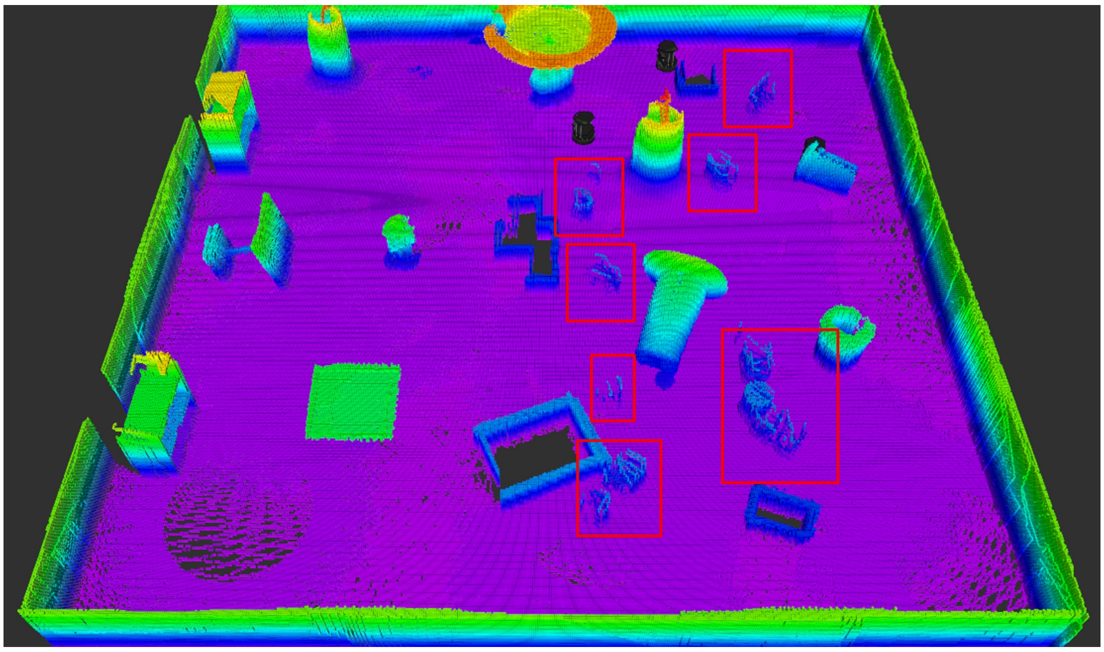
As mentioned, in the current implementation, we use $RTAB-Map$ as the SLAM system and use the in-built octomap server to calculate the octomap used by the *Planner Module* and the *Explore Module*. The SLAM system only adds point clouds to the existing map and does not allow for the removal of point clouds or portions of point clouds over time. Since the octomap is calculated from the point cloud map from the SLAM system, the octomap loses its quality of probabilistic updatability. If the octomap is calculated directly from the camera sensor data instead, it allows for probabilistic updatability when new sensor data of the same region become available over time.

In the current implementation, when a dynamic obstacle appears in one robot's active sensor region or appears in the area already mapped but gets scanned by the robot during movement, it gets registered in the point cloud and gets included in the octomap. When a dynamic obstacle leaves a robot's active sensor region or leaves an already mapped area scanned afterward, it does not get removed from the point cloud map since it does not support point cloud removal. Since the octomap has lost its quality of probabilistic updatability, the obstacle does not get removed from the octomap either.

The other issue we identified in our system, the inability to separately distinguish dynamic obstacles, can be explained using the explanation mentioned earlier. The Figure 5.14illustrates several instances where one robot has entered or moved out of another robot's active sensor region. Those dynamic obstacles have been registered on the point cloud map as obstacles more significant than the original obstacle due to movement and appear similarly on the octomap.



(a)



(b)

Figure 5.14: Dynamic objects that were visualized as obstacles (a) in point cloud (b) in octomap

Chapter 6

CONCLUSIONS AND RECOMMENDATIONS

In this research, we presented a two-stage octomap based exploration and navigation system for unknown environments. We used octomaps to model the world and then to evaluate the unexplored region and calculate navigation paths. We proposed a system that can function as either a single robot exploration system or a multi-robot exploration system depending on the size of the area to be covered and the speed required.

We evaluated our single robot system instance comparatively with the Explore-Lite system using simulations. The results show that our proposed system can explore 38% faster than the Explore-Lite system while achieving the same coverage of the area.

Comparison between one robot, two robots, and three robot systems show that our proposed multi-robot system is effective and successful. A Multi-robot system using two robots is 48% faster than the individual robot system, and a multi-robot system using three robots is 38% faster than the individual robot system. However, we cannot make any other quantitative claims, such as the most effective number of robots for the selected environment, due to various issues, such as lack of computational resources and optimization.

In the future, all the modules including the *Server System* need to be improved and optimized using parallelization and memory optimization approaches. This would allow the system to be used on computers with less computational resources and has the potential to increase the speed as well. In addition to that, the system needs to be tested on the field on a real robot system. This requires parameter tuning to suit the physical limitations of the robot, would also require unit testing of each module and ultimately testing of the whole system.

Another future task would be the development of a SLAM system that utilizes non-rigid deformation for loop closing would be helpful for feature-based initial position unknown map merging on the *Server System*, eliminating the requirement for knowing the initial position of robots for map merging and coordination in multi-robot systems. However, this can reduce the speed and effectiveness of the exploration due to the long processing time and the significant overlap of the map required for successful merging.

Separation of the mapping and octomap calculation functionalities at the robot level would resolve the dynamic obstacle issue shown in Figure 5.14. *SLAM System* needs to be used to calculate the correct position of the robot and create the map of the environment while a separate instance of the Octomap server needs to be used to calculate the octomap environment based on the depth sensor data. This separation would allow dynamic obstacles to be identified in the octomap without getting identified as a stretched-out obstacle, as in Figure 5.14b.

References

[1] R. Dubé, A. Gawel, C. Cadena, R. Siegwart, L. Freda, and M. Gianni, "3d localization, mapping and path planning for search and rescue operations," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2016, pp. 272–273.

[2] E. Ackerman, "Team costar trains robots for exploring caves on earth and space," Jun 2021. [Online]. Available: https://spectrum.ieee.org/team-costar-robots-earth-space

[3] H. Pozniak, "Finders, keepers: Search and rescue robots evolve," Jan 2020. [Online]. Available: https://eandt.theiet.org/content/articles/2020/01/finders-keepers-search-and-rescue-robots-evolve/

[4] A. Chilian and H. Hirschmüller, "Stereo camera based navigation of mobile robots on rough terrain," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4571–4576, 2009.

[5] R. L. Klaser, F. S. Osório, and D. F. Wolf, "Vision-based autonomous navigation with a probabilistic occupancy map on unstructured scenarios," *2014 Joint Conference on Robotics: SBR-LARS Robotics Symposium and Robocontrol*, pp. 146–150, 2014.

[6] F. Bourgault, A. Makarenko, S. B. Williams, B. Grocholsky, and H. Durrant-Whyte, "Information based adaptive robotic exploration," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 540–545, 2002.

[7] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun, "Towards object mapping in non-stationary environments with mobile robots," *IEEE/RSJ Interna-*

tional Conference on Intelligent Robots and Systems, vol. 1, pp. 1014–1019, 2002.

[8] X. Han, Y. Leng, H. Luo, and W. Zhou, "A novel navigation scheme in dynamic environment using layered costmap," *2017 29th Chinese Control And Decision Conference (CCDC)*, pp. 7123–7128, 2017.

[9] K. Ratnayake, "Motion planner to explore unknown rough terrain," 2019, unpublished Bachelor's Thesis.

[10] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, pp. 46–57, 1989.

[11] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189–206, 2013.

[12] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, 1985.

[13] D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pp. 692–697, 2012.

[14] A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta, "Navigation in three-dimensional cluttered environments for mobile manipulation," *2012 IEEE International Conference on Robotics and Automation*, pp. 423–429, 2012.

[15] C. Wang, L. Meng, S. She, I. M. Mitchell, T. Li, F. Tung, W. Wan, M. Meng, and C. D. Silva, "Autonomous mobile robot navigation in uneven and unstructured indoor environments," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 109–116, 2017.

[16] K. Schmid, T. Tomic, F. Ruess, H. Hirschmüller, and M. Suppa, “Stereo vision based indoor/outdoor navigation for flying robots,” *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3955–3962, 2013.

[17] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. Gerkey, “Outdoor mapping and navigation using stereo vision,” in *Springer Tracts in Advanced Robotics.* Springer Berlin Heidelberg, 2008, vol. 39, pp. 179–190.

[18] J. J. Lopez-Perez, U. H. Hernandez-Belmonte, J. Ramirez-Paredes, M. A. Contreras-Cruz, and V. Ayala-Ramírez, “Distributed multirobot exploration based on scene partitioning and frontier selection,” *Mathematical Problems in Engineering*, vol. 2018, pp. 1–17, 2018.

[19] U. Jain, R. Tiwari, S. Majumdar, and S. Sharma, “Multi robot area exploration using circle partitioning method,” *Procedia Engineering*, vol. 41, p. 383–387, 12 2012.

[20] C. Nieto-Granda, I. John G. Rogers, and H. I. Christensen, “Coordination strategies for multi-robot exploration and mapping,” *The International Journal of Robotics Research*, vol. 33, no. 4, pp. 519–533, 2014. [Online]. Available: https://doi.org/10.1177/0278364913515309

[21] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, “Coordinated multi-robot exploration,” *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[22] B. Yamauchi, “A frontier-based approach for autonomous exploration,” *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pp. 146–151, 1997.

[23] V. R. Jisha and D. Ghose, “Goal seeking for robots in unknown environments,” *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4692–4697, 2010.

[24] M. Keidar, E. Sadeh-Or, and G. Kaminka, “Fast frontier detection for robot exploration,” *The International Journal of Robotics Research*, vol. 33, pp. 281–294, 05 2011.

[25] P. VaisakhV, “Quick goal seeking algorithm for frontier based robotic navigation,” *International Journal of Computer Applications*, vol. 100, pp. 1–7, 2014.

[26] M. S. Ramanagopal, A. P.-V. Nguyen, and J. L. Ny, “A motion planning strategy for the active vision-based mapping of ground-level structures,” *IEEE Transactions on Automation Science and Engineering*, vol. 15, pp. 356–368, 2018.

[27] J. Horner, “Map-merging for multi-robot system,” Bachelor’s thesis, Charles University in Prague, Faculty of Mathematics and Physics, Prague, 2016. [Online]. Available: https://is.cuni.cz/webapps/zzp/detail/174125/

[28] B. Yamauchi, “Frontier-based exploration using multiple robots,” in *Proceedings of the Second International Conference on Autonomous Agents*, ser. AGENTS ’98. New York, NY, USA: Association for Computing Machinery, 1998, p. 47–53. [Online]. Available: https://doi.org/10.1145/280765.280773

[29] F. Benavides, C. Ponzoni Carvalho Chanel, P. Monzón, and E. Grampín, “An auto-adaptive multi-objective strategy for multi-robot exploration of constrained-communication environments,” *Applied Sciences*, vol. 9, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/3/573

[30] G. Amorín, F. Benavides, and E. Grmapin, “A novel stop criterion to support efficient multi-robot mapping,” in *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, 2019, pp. 369–374.

[31] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, “A flexible and scalable slam system with full 3d motion estimation,” in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. IEEE, November 2011, pp. 155–160.

[32] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “Orb-slam: A versatile and accurate monocular slam system,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[33] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[34] R. Elvira, J. D. Tardos, and J. Montiel, “Orbslam-atlas: a robust and accurate multi-map system,” *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov 2019. [Online]. Available: http://dx.doi.org/10.1109/IROS40897.2019.8967572

[35] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM3: an accurate open-source library for visual, visual-inertial and multi-map SLAM,” *CoRR*, vol. abs/2007.11898, 2020. [Online]. Available: https://arxiv.org/abs/2007.11898

[36] M. Labbé and F. Michaud, “Long-term online multi-session graph-based splam with memory management,” *Autonomous Robots*, vol. 42, no. 6, pp. 1133–1150, 2017.

[37] P. Schmuck and M. Chli, “Ccm-slam: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams,” *Journal of Field Robotics*, vol. 36, no. 4, pp. 763–781, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21854

[38] L. Riazuelo, J. Civera, and J. M. M. Montiel, “C2tam: A cloud framework for cooperative tracking and mapping,” *Robotics Auton. Syst.*, vol. 62, pp. 401–413, 2014.

[39] S. S. Ali, A. Hammad, and A. S. T. Eldien, “Cloud-based map alignment strategies for multi-robot fastslam 2.0,” *International Journal of Distributed Sensor Networks*, vol. 15, no. 3, 2019.

[40] A. Cunningham, M. Paluri, and F. Dellaert, “Ddf-sam: Fully distributed slam using constrained factor graphs,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3025–3030.

[41] H. A. Daoud, A. Q. M. Sabri, C. Loo, and A. M. Mansoor, “Slamm: Visual monocular slam with continuous mapping using multiple maps,” *PLoS ONE*, vol. 13, 2018.

[42] D. Calisi and D. Nardi, “Performance evaluation of pure-motion tasks for mobile robots with respect to world models,” *Autonomous Robots*, vol. 27, pp. 465–481, 2009.

[43] N. Botteghi, B. Sirmaçek, M. Poel, and C. Brune, “Reinforcement learning helps slam: Learning to build maps,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 43, pp. 329–336, 2020.

[44] R. Daher, “Map to map: From slam to cad maps and back using generative models map to map: From slam to cad maps and back using generative models,” Ph.D. dissertation, 12 2020.

[45] M. Kegeleirs, D. Garzón Ramos, and M. Birattari, “Random walk exploration for swarm mapping,” in *Towards Autonomous Robotic Systems*, K. Althoefer, J. Konstantinova, and K. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 211–222.

[46] M. H. Shayesteh and M. M. Raeisi, “Technical issues of mrl team for 2021 robocup rescue simulation virtual robot competition.”

[47] J.-H. Kim, X. Lin, N. Kanyok, A. Shaker, P. Poudel, I. Cardenas, N. K. C. Cabrera, H. Jeong, Gokarna, and P. Sharma, “Robocup rescue 2019 tdp virtual robot simulation atr (us),” 2019.