DEVELOPMENT OF COORDINATING ALGORITHM FOR RENDEZVOUS TASK OF MOBILE ROBOTS

NG JIN SIANG

UNIVERSITI TUNKU ABDUL RAHMAN

DEVELOPMENT OF COORDINATING ALGORITHM FOR RENDEZVOUS TASK OF MOBILE ROBOTS

NG JIN SIANG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

May 2023

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	[illegible]
Name	:	Ng Jin Siang
ID No.	:	1804136
Date	:	21/5/2023

APPROVAL FOR SUBMISSION

I certify that this project report entitled **DEVELOPMENT OF COORDINATING ALGORITHM FOR RENDEZVOUS TASK OF MOBILE ROBOTS** was prepared by **NG JIN SIANG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Dr. Shalini a/p Darmaraju
Date	:	22/5/2023

Signature	:	[illegible]
Co-Supervisor	:	Dr. Kwan Ban Hoe
Date	:	22/5/2023

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to everyone who contributed to the successful completion of this project. First and foremost, I am grateful to my research supervisor, Dr. Shalini a/p Darmaraju, and co-supervisor, Dr. Kwan Ban Hoe for their invaluable advice, guidance, and enormous patience throughout the development of this research.

I am also deeply thankful to my loving parents and friends, who provided me with unwavering support and encouragement whenever I needed it. Without their help, I would not have been able to complete my final year project with such success.

ABSTRACT

The coordination of mobile robots is a crucial component in a variety of applications, including search and rescue, environmental monitoring, and transportation. It is essential that the robots work together to achieve a common goal efficiently and safely. In this project, the focus was on the rendezvous task, which involves bringing multiple robots to a common location. To accomplish this task, a coordinating algorithm was developed that allows the robots to work together in a distributed manner. Each robot has a limited view of the environment, and they communicate with their neighbouring robots to share information about their current position. The proposed algorithm was developed using MATLAB and implemented within ROS2, which are widely used in the robotics industry. The algorithm ensures that all robots converge to the rendezvous point while avoiding collisions with each other. It achieves this by selecting a leader or multiple leaders and then having all other robots converge towards the position of the leader, based on the information received from neighbouring robots. Overall, the proposed algorithm provides an efficient solution for the coordination of mobile robots in the rendezvous task, where the algorithm is capable of handling different group scenarios and is coordinate-free. It has the potential to be used in a wide range of applications, such as flocking control, making it a valuable contribution to the field of robotics.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AUV	Autonomous Underwater Vehicles
API	Application Programming Interface
BFS	Breath First Search
GPS	Global Positioning System
LMR	Legged Mobile Robot
MATLAB	Matrix Laboratory
MRS	Multi-Robot System
ROS	Robot Operating System
RTOS	Real Time Operating System
TF	Transform Library
SSR	Safe Sensing Range
UAV	Unmanned Aerial Vehicle
WMR	Wheeled Mobile Robot

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

As technology advances in robotics, countless opportunities for useful applications in diverse domains are emerging. While some applications can utilize a single robot to complete a task, there are scenarios that require multiple robots to accomplish what is impossible for a single robot.

A Multi-Robot System (MRS) can be understood as the application of multiple robots working together to achieve a certain objective (Jawhar *et al.*, 2018). MRS architectures offer different trade-offs in terms of reliability, scalability, and coordination (Jawhar *et al.*, 2018). The centralized architecture provides a centralized decision-making process but is prone to single point failures and scalability limitations. On the other hand, hierarchical architecture allows for scalability but can suffer from reduced reliability when failures occur at higher levels of the hierarchy. Decentralized architecture, being the most common category, offers robustness and adaptability to failures but poses challenges in maintaining synchronization and coordinating actions when mission objectives change. Ultimately, the choice of architecture depends on the specific requirements of the MRS application, balancing factors such as system reliability, scalability, and coordination complexity (Jawhar *et al.*, 2018).

Numerous robots with different abilities can work together to deal with complex tasks, and if one or more robots fail, it will not affect the completion of the task. MRS has been used in several domains for applications such as warehousing, search-and-rescue, environmental monitoring, and exploration. MRS can be made up of teams of homogeneous or heterogeneous robots. Teams of homogeneous robots have the same individual capability, whereas heterogeneous robot teams have different capabilities of robots.

Multi-robot systems have attracted the attention of researchers due to their potential for various applications. However, utilizing MRS comes with difficulties such as task division, synchronization, and coordination. Practical considerations of mobile robots further contribute to the complexity of multi-robot systems compared to single robots.

1.1.1 Mobile Robot

Mobile robots are becoming increasingly important in a variety of industries, from manufacturing and logistics to agriculture and healthcare. The ability to move autonomously from one place to another makes mobile robots versatile and adaptable to different environments and tasks. The types of environments where mobile robots operate can vary greatly, from the air to the ground and even underwater. In terrestrial environments, wheeled mobile robots (WMRs) and legged mobile robots (LMRs) are common. WMRs, in particular, are widely used in applications such as warehouse logistics, material handling, and transportation. They can be configured with different numbers of wheels, ranging from two-wheeled robots like the E-Puck 2 and Khepera-IV as shown in Figure 1.1 and Figure 1.2, to four-wheeled and even six-wheeled robots.

The selection of a configuration is dependent on the particular demands of the task and the surroundings in which the robot will function. For example, two-wheeled robots are more manoeuvrable but less stable than four-wheeled robots, while six-wheeled robots are better suited for rough terrain. Overall, the use of mobile robots, and in particular, wheeled mobile robots, is growing rapidly, and they are expected to play a significant role in the future of automation and robotics (Javaid *et al.*, 2021).

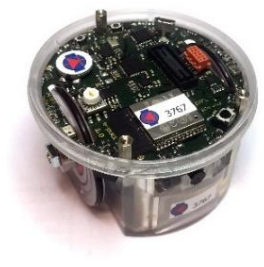


Figure 1.1: E-Puck 2 (GCtronic, no date).

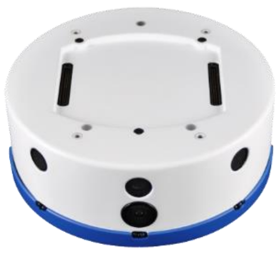


Figure 1.2: Khepera IV (*Khepera IV New - K-Team Corporation*, 2021)

Industry 4.0 technology enables factories to integrate autonomous mobile robots (AMRs) into their assembly lines without relying on external localization systems. Equipped with sensors and cameras, AMRs can autonomously navigate their environments. As AMRs continue to evolve, they will transform from simple management systems to predictive data systems, enabling manufacturers to make informed decisions during plant development (Javaid *et al.*, 2021). The acceptance of Industry 4.0 is increasing due to the widespread presence of enhanced computing capacity and automation, which many consider as the next industrial revolution.

1.1.2 Rendezvous Problem

Rendezvous is a crucial concept in many areas, including mobile robots, where it refers to the meeting of two or more people at a predefined place and time. This concept is fundamental to the field of the cooperative mobile robots, where it is referred as the rendezvous problem or the consensus problem (Potop-Butucaru, Raynal and Tixeuil, 2011).

The rendezvous problem has been well studied in the game theory, where it is introduced as a search robot (Alpern, 2011). A well-known example is that the princess and monster game, where a group of robots needs to find the hidden target while avoiding a mobile opponent. The game highlights the importance of the rendezvous problem, as it requires the robot to converge in a single location or position to achieve their goal.

In a multi-robot system, one of the most basic coordination tasks is multi-robot rendezvous. In a network of robot, rendezvous problem is where the distributed robot needs to converge at a same location either based on consensus or immediate goals (Lin, Morse and Anderson, 2004). The objective of the rendezvous problem is to achieve an agreement over the potential rendezvous point, which can be referred as landmark, with limited information flow described in the model of the network. The model of network involved a set of distributed mobile robots. Achieving this goal can be challenging, as robots often have limited information about the positions and movements of their peers. However, there are many different techniques that can be used to solve the rendezvous problem, including consensus algorithms, distributed optimization, and decentralized control.

Ultimately, the key to successfully solving the rendezvous problem is effective communication and coordination between robots. By interacting with each other and sharing information, robots can work together to achieve a common goal and accomplish tasks that would be impossible for a single robot to complete alone.

1.2 Importance of the Study

In multi-robot systems, the rendezvous task is important because it enables robot coordination and cooperation to complete challenging tasks. The task helps to efficiently manage resources such as energy, bandwidth, and processing power. It is adaptable and scalable, enabling multi-robot systems to adapt various environments and tasks.

For multi-robot systems to perform to their full capacity in a variety of applications, including search and rescue operations, exploration of uncharted territory, surveillance, and transportation, the rendezvous task must be implemented successfully. Therefore, the rendezvous task is a critical component of multi-robot systems that enables robots to work together towards a common goal.

1.3 Problem Statement

In multi-robot rendezvous, the sensing capability of mobile robots is a critical consideration. Most rendezvous control algorithms assume that each robot can measure the relative position of its neighbours by utilizing global localization systems, such as GPS, for external navigation. However, such methods may not be applicable in communication-denied environments like indoors or hostile areas.

An alternative approach is to use onboard sensors, which offer the advantage of being passive and not requiring wireless communication. These sensors can include cameras, LIDAR, and other range sensors, which enable the mobile robots to perceive their environment and detect other nearby robots without relying global localization systems. By utilizing onboard sensors capabilities, multi-robot rendezvous can better operate autonomously and without interruption.

However, simply relying on onboard sensors may not be enough to guarantee safe and efficient rendezvous in complex environments. There may be obstacles or other environmental factors that block the mobile robots' movement and increase the risk of collisions. Therefore, additional algorithms, such as collision avoidance, should be implemented to ensure safe navigation and successful rendezvous. These algorithms can use data from onboard sensors to detect and avoid obstacles, allowing the robots to navigate effectively in complex environments while avoiding collisions.

1.4 Aim and Objectives

This project aims to develop a coordinating algorithm for the rendezvous task for mobile robots. The detailed objectives are:

- To conduct a literature search on the existing coordinating algorithm for rendezvous task of mobile robots.
- To develop a coordinating algorithm in a decentralized approach for rendezvous task.
- To simulate the proposed algorithm and evaluate the scalability and performance of the proposed algorithm.
- To implement the developed algorithm in ROS2 and integrate it with the control and communication of the mobile robots.

1.5 Scope and Limitation of the Study

The scope of this project is to develop and implement a coordinating algorithm for rendezvous tasks for mobile robots. The project will focus on developing the rendezvous control law on the MATLAB platform and implementing it within ROS2.

However, this study has some limitations. We will be using a single integral model to represent the mobile robot, and kinematic constraints of the mobile robot will be ignored during the algorithm development phase. Additionally, within the simulation, the mobile robots are capable of recognizing each other, and have their own number tag within their system.

1.6 Contribution of the Study

The contribution of the study is introducing a novel control framework that enables robots to gather in different groups, facilitating efficient coordination and collaboration among them.

1.7 Outline of the Report

The report is divided into five chapters. Chapter 1 provides a general introduction to mobile robots and the rendezvous task, along with the problem statement and objectives. Additionally, the chapter provides a detail overview of the scope of the study inform the reader that the project is limited to certain events.

Chapter 2 provides a literature review on the type of rendezvous consensus algorithm and the overall structure of ROS and ROS2. This chapter includes detailed research, selection of solution, and literature review.

In Chapter 3, the methodology to achieve the objectives of this project is demonstrated from start to finish. The methodology includes the proposed solution for the rendezvous control, as well as the toolboxes used in MATLAB and ROS2.

Chapter 4 presents the results obtained through simulations during the development phase in MATLAB and compares the performance with state-of-the-art algorithms. Additionally, the chapter presents results obtained during the implementation phase in ROS2. Lastly, Chapter 5 concludes the project and provides recommendations for future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The development of mobile robots is crucial as they can replace human presence in extreme environments. In many scenarios, multiple robots with different capabilities are required to accomplish complex tasks. With the advancement in computing capacity and the abilities of mobile robots, extensive research has been conducted on the coordination and control of multiple mobile robots and has been applied in various applications. Coordination of multiple robots can include motion planning, formation control, and trajectory generation. However, this project will focus specifically on the rendezvous task for mobile robots.

2.1.1 Coordinated Control of Multi-Robot System

Coordination and control of multi-robot systems are essential since there are numerous robots in a system, and an algorithm for coordination between them is needed to control their cooperative actions (Verma and Ranga, 2021). In cooperation, robots need to not only achieve their own goals but also be aware of their partners' urgent tasks.

Therefore, an effective coordinating mechanism can assist each robot in selecting its actions to maximize the system's efficiency, taking into account sensing, information sensing, physical and environmental constraints, and inherent limitations. Coordinating approaches can be classified based on different parameters, and one possible classification is shown in Figure 2.1.

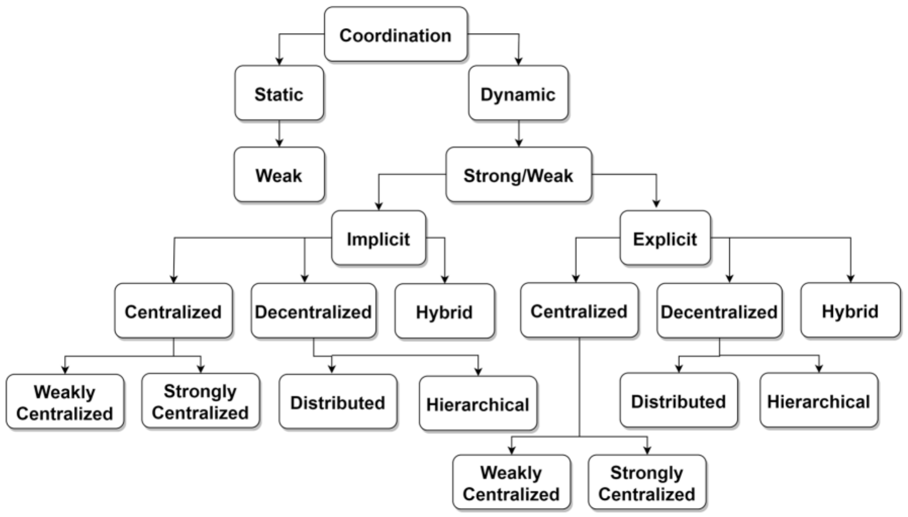


Figure 2.1: Classification of Coordination in MRS (Verma and Ranga, 2021)

Many multi-robot systems develop their coordination control using either strongly centralized control or decentralized control (Ke Xu, 2010). Strongly centralized control coordination can be achieved by using a single coordinating robot, which can be referred to as a server or leader. The server is responsible for making decisions for all the robots in the system. Global information, such as the environment and the location of the mobile robot, is shared. Strongly centralized control is typically used for a small number of robots within well-structured environments and is not suitable for dynamic environments because the control is not robust enough and may cause communication failures.

Decentralized control can be further classified into distributed approaches and hierarchical approaches, which are locally centralized. In distributed approaches of MRS, each robot executes its own algorithm completely autonomously (Verma and Ranga, 2021). There are no leaders within the architecture, and the system has all equal robots with respect to control. The advantages of distributed approaches are that they provide more robustness to failure; however, it is more complex to achieve coordination among the team of robots. Therefore, hierarchical approaches come into play, where the process of coordination is locally centralized. This is different from centralized control coordination in that the follower robot is not controlled by a single leader. The application of hierarchical approaches in MRS allows a group of robots to work on different tasks after being divided into a few groups. These approaches are less robust than distributed approaches but have less complexity and cost.

In order for a multi-robot system to be successfully implemented, various distributed control strategies have been developed to organize the robots while working or solving problems using local interaction rules. To meet the constraints required for a distributed multi-robot coordination algorithm to be implemented, the algorithm must satisfy several requirements (Cort&eacute;s and Egerstedt, 2017).

The first constraint is that the algorithm must be local, meaning that each individual robot can only act on the information it can access through its sensing capabilities or its active communication network. The second constraint is that the algorithm must be scalable and decentralized, meaning that it can be implemented in both small and large teams of robots.

The third constraint is that the algorithm must ensure safety. After the robots are deployed for their mission, the algorithm must prevent collisions between robots and with the environment. Lastly, the algorithm must be emergent, which means that global properties should emerge from the local interaction rules.

2.2 Rendezvous Control Strategy

The following section will discuss and introduce a few well-known rendezvous control strategies. The common objective of all algorithms is that the mobile robots to meet at a common location, but they have no information at first where the meeting location is at.

2.2.1 Hierarchical Consensus Algorithm

Hierarchical Consensus, can also be known as leader-follower consensus, consists of one or multiple leaders within the mobile robot network. There are multiple applications of leader in the network of robots, where one of the common uses is by setting the coordinate or the dynamics of the leader. This means that the leader robots, or robot, take on the responsibility of defining the desired position or motion pattern for the entire group. The follower robots in the network then adjust their movement or behaviours based on the information provided by the leader.

With a leader present, practical applications such as herding, which define the path for the entire group, can be achieved. The selection of a leader depends on various algorithms. However, leader robots are chosen when their capabilities far surpass those of the follower robots, such as advanced sensing systems, functional capabilities, coordination with external systems, or even the ability to handle complex scenario. By having these superior capabilities, the leader robot can efficiently navigate through obstacles, maintain the desired formation, and adapt to changing condition.

A Multi-Point Rendezvous Strategy, proposed by Parasuraman et al. (2020), is an efficient hierarchical rendezvous algorithm that begins by creating a local neighbour set using the local sensors of each robot to sense its neighbouring robots. Then, each mobile robot shares its list with its neighbours and constructs a global connectivity graph using a distributed algorithm in the network. The leader robot is selected based on its capabilities or function, such as recharging stations, or it could be chosen using Dijkstra's algorithm from the global connectivity graph to optimize the shortest route travelled for the mobile robot in the rendezvous task. Sub-graphs are also constructed using the global connectivity graph to reduce computational load. To build the shortest path trees, the distance cost from every node to the root node is optimized. Each tree node can have a parent or children, with leaf nodes being nodes that do not have any children. The hierarchical rendezvous algorithm allows each follower robot in the tree to move towards the root node to rendezvous. The researchers simulated five different scenarios to demonstrate the effectiveness of their developed rendezvous strategy, which performs better than the standard consensus and circumcenter-based consensus.

A bearing-aided hierarchical tracking method proposed by Luo et al. (2019), which uses bearings of robots and is coordinate-free. The algorithm starts by building a shortest path tree from the leader robot using Dijkstra's algorithm. The rendezvous strategy begins with the children nodes moving towards their parents. The process repeats until all robots reach the highest parent, the leader robot, where they rendezvous. Thus, the rendezvous task is completed by using iterative tracking movement from the leaf nodes of the generated tree to the root node. Their field experiment and simulation are noteworthy in that, even with mobility and communication faults in one node, the mobile robot is still able to rendezvous at the desired point, demonstrating the scalability of their algorithm in the simulation.

Kan et al. (2017), have developed decentralized time-varying controllers for nonholonomic robots that rely only on local sensing feedback from their neighbouring robots to achieve rendezvous. The algorithm can stabilize the mobile robots at the specified location and is capable of preserving the connectivity network and implementing collision avoidance along the way. A small portion of robots is equipped with advanced sensors, which can be identified as leaders, while the others only have a limited range sensor that can provide local feedback on the relative trajectory of neighbouring robots. The global objective, which is the rendezvous point, is known only by the leader robot, while the followers coordinate their motion by using local information from immediate neighbours. Numerical simulations are done to demonstrate the effectiveness of the algorithm and are illustrated in Figure 2.2. This work is noteworthy in that the algorithm does not require inter-agent communication to achieve consensus.

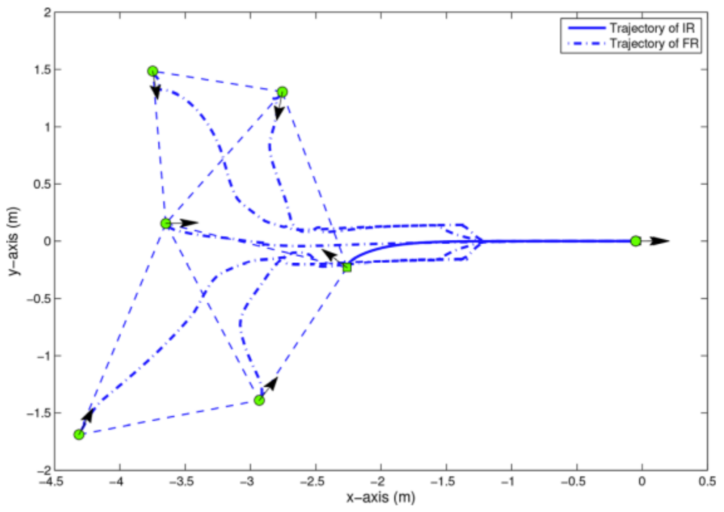


Figure 2.2: Numerical Simulation with Trajectory Line of Leader and Follower Robot (Kan *et al.*, 2017).

2.2.2 Deterministic Algorithm

Deterministic algorithm is a type of algorithm that only depends on its input, and there will be no randomness within the model. With the same input, the algorithm will always produce the same output. However, the main problem that deterministic algorithm needs to face is that the algorithm needs to break the symmetry.

This can be explained by giving a highly symmetrical network, such as an oriented ring, where the port can be labelled as follow: 0 as clockwise port and 1 as anti-clockwise port. Two identical mobile robot runs the same deterministic algorithm inside the oriented ring. If the agents are unable to mark any labels within their pathway, it can be said that the mobile robot will always use the same port and their distance will always be the same (Pelc, 2012). Therefore, randomization is required to break the symmetry as it can change how the robots determine their next port. Breaking the symmetry can be accomplished by either marking nodes or labelling the robot.

Ozsoyeller et al. (2022), proposed a deterministic rendezvous search algorithm which is m-RP. In the algorithm m-RP, we can identify two distinct roles in mobile robots, which one is active robots, and another is passive robots. m-RP algorithm labels each mobile robot to use the passive robot to break the symmetry, thereby achieving deterministic rendezvous. The algorithm does not require a connectivity graph to be built, which decreases the sensor range. The m-RP algorithm is practically promising since simulations demonstrate successful rendezvous even in asynchronous cases, where robots may not start searching simultaneously. The algorithm commences with an exploration stage consisting of two phases. In the first phase, the active robot starts searching for the waiting robot by drawing concentric circles, as shown in Figure 2.3. In the second phase, the active robot visits all waiting robots it has found. In phase 1, if mobile robot has not found any waiting robot, then the mobile robot will start the second round searching for waiting robot. The second stage is the rendezvous stage, where one leader robot terminates the exploration stage. The leader will select each foreign parent representative, which is the first waiting robot where other leader robots meet. This will inform another leader to end the exploration stage. Next, the leader robot will designate the rendezvous location as the center of a circle whose diameter corresponds to the distance between the farthest parent representative robot pair within the environment. The rendezvous task will be considered complete when all robots have arrived at the rendezvous location.

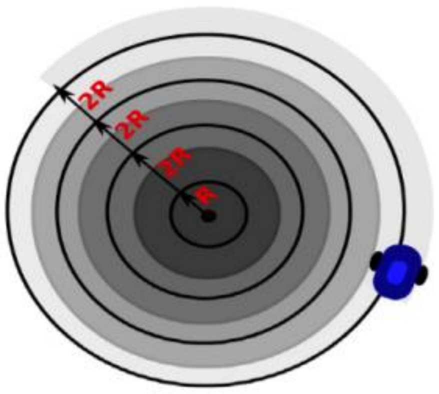


Figure 2.3: Active mobile robot executing m-RP (Ozsoyeller, Özkasap and Aloqaily, 2022)

Ozsoyeller & Tokekar (2022) developed a symmetric rendezvous algorithm that involves randomized and deterministic search. As identical mobile robots initiate with the deterministic algorithm simultaneously, the adjacent distance between two mobile robots will be the same, and thus the rendezvous task will not be complete. Therefore, randomization is needed to break the symmetric search at the start of the algorithm. The algorithm starts with every robot in a single condition and executes a randomized search mode that starts by flipping a coin at the beginning of the round. In the event that the coin toss results in tails, the robot will proceed to move to the right in both phase-1 and phase-2. Once a single robot encounters another single robot, both robots will switch to deterministic search mode. The direction of the deterministic search is dependent on the robot's current phase, with opposite directions assigned if the robots meet during phase-1 and the same direction assigned if they meet during phase-2. Figure 2.4 depicts one possible iteration of robot j when implementing the MSR, where the starting point is represented by a square icon and the end point by a circular icon.

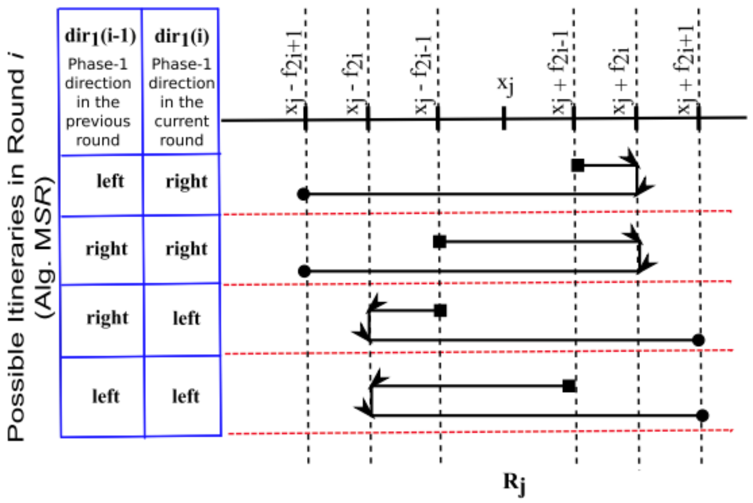

dir1(i-1)
Phase-1 direction in the previous round
dir1(i)
Phase-1 direction in the current round
Possible Itineraries in Round i (Alg. MSR)
left
right
right
right
right
left
left
left
xj - f2i+1
xj - f2i
xj - f2i-1
xj
xj + f2i-1
xj + f2i
xj + f2i+1
Rj

Figure 2.4: Possible Iteration of Robot j when executing MSR (Ozsoyeller and Tokekar, 2022)

Ando et al. (1999) proposed a deterministic and localized algorithm that uses a vision sensor attached to a mobile robot. The algorithm focuses on the mobile robot observing the relative positions of its visible neighbour robots in each step. It then moves towards its new position based on the observation using the computed algorithm, and then moves towards the location. The authors claimed that the algorithm is self-stabilizing, which means the robot is capable of starting from any initial position when no error occurs. With limited visibility of the robot, it can only see other robots within a distance of V, with no other robot between them. The algorithm is memoryless because the next moves of the robot are dependent only on what the robot can see at that moment. The algorithm is described as follows: At each round, the robot sends its own position and receives its neighbour's position. The robot then computes the circumcenter of the position between itself and its neighbour, and the robots move towards the circumcenter position while maintaining connectivity.

2.2.3 Bearing-only or Range-only Strategy

A control strategy that involves bearing-only and range-only measurements has been developed by Zheng and Sun (2014) for multiple nonholonomic wheeled robots to achieve rendezvous. The presented work includes four different controllers, with two being bearing-only controllers and two being range-only controllers, which have been verified using Monte Carlo simulations. The applicability and performance of the controllers are illustrated using e-puck robots in the experimental platform. A collision avoidance algorithm is also used in the experimental platform, which enables the controller to have a more practical design for realistic scenarios. The rendezvous strategy starts after the global connectivity graph is built. It is interesting to note that the authors use only one type of measurement in their controller: bearing-only measurement, where each robot can only measure the bearing angle of the detectable robot in their local frame, and range-only measurement, where a mobile robot can only measure the distance between the other robot that it can detect. Figure 2.5 shows the bearing angle of the simulated mobile robot.

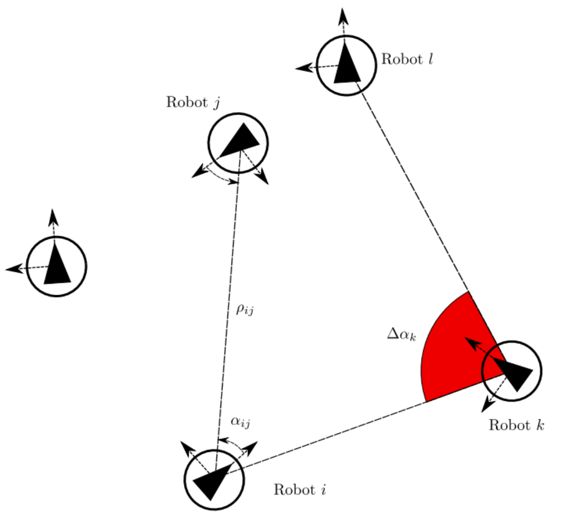


Figure 2.5: Bearing angle of the mobile robot (Zheng and Sun, 2014).

2.2.4 Multi-Group Rendezvous

An early multi-group rendezvous algorithm was introduced by Krishnanand & Ghose (2008), based on the theoretical principles of the glow-worm swarm optimization (GSO) algorithm. The algorithm allows swarm agents with restricted sensing ranges to divide into separate subgroups with the maximum luciferin value determined by the objective function pre-defined at the robots' distributed locations. In their simplified GSO model, they restricted their analysis to the local convergence of agents to a leader, similar to the hierarchical consensus algorithm. However, what differs is that the leader is chosen when they have the peak value of luciferin, and multiple peaks may produce multiple leaders. The goals of the rendezvous are also different, as the goal for GSO is to achieve the position of the local optimum of the objective function, such as luciferin, whereas the hierarchical consensus algorithm normally rendezvous at the dynamic leader.

2.3 Robot Operating System (ROS) and ROS2

ROS (Robot Operating System) is middleware that is open-source and has been rapidly developed and extensively used for robotics applications. It was created and maintained by Willow Garage and the Open-Source Robotics Foundation (OSRF) starting from 2007 (Quigley *et al.*, 2009). ROS is designed to promote code sharing among the robotics community with the intention of making progress faster by allowing researchers and developers to replicate and extend the results of other research groups. The following is a list of basic ROS components and terminology:

i. Nodes – ROS nodes serve as the fundamental building blocks of a ROS application and are responsible for running programs or processes.

ii. Messages – Messages are the medium between nodes, where nodes send messages to each other, either in standard format or application specific.

iii. Topics – Topics are channel that nodes used to exchanges messages.

iv. Services – Services enable synchronous communication between nodes by facilitating a request/response model. A server node only responds to requests made by a service client node, which is capable of sending requests and receiving responses. After the request and response have been fulfilled, the connection between the nodes is terminated.

v. Action – An action involves an action client sending a request to an action server, which then executes the requested action and provides ongoing feedback to the client during the process. This approach is typically employed when a response is expected to take a considerable amount of time, like services.

ROS operates by establishing a distributed system of nodes that can interact with one another using messages. The communication is facilitated through a publish-subscribe messaging system, where nodes broadcast messages to topics, and other nodes can subscribe to those topics to receive the message. Figure 2.6 shows an example of the basic structure of a ROS node.

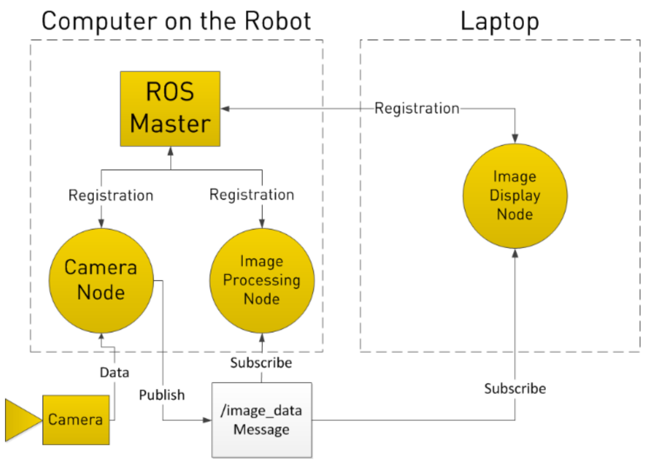


Figure 2.6: Basic Structure of ROS (Clearpath Robotics, 2014).

Despite its many benefits, ROS does have some centralized features, such as the ROS Master. Due to its centralized network configuration, there exists a central controller responsible for naming and registration services - this controller is known as the ROS master. Its primary objective is to facilitate the location of other ROS nodes on the network and enable communication between them in a peer-to-peer manner. However, this approach has its limitations, particularly in situations where the network's nodes are distributed across multiple computers, as it can be less robust. Another limitation of ROS is that it is not suitable for real-time control applications since it does not guarantee process synchronization and timing control (Reke *et al.*, 2020). Therefore, ROS2 has been developed to overcome these limitations.

The initial distribution of ROS2 was launched in 2017 with the objective of accommodating multiple robots working in teams, small and embedded platforms, real-time control, suboptimal networks, and multi-platform support (including Linux, Windows, and RTOS). Therefore, structural changes have been made, and new technologies such as Data Distribution Services have been adapted. The Figure 2.7 shows the differences in architecture between ROS2 and ROS.

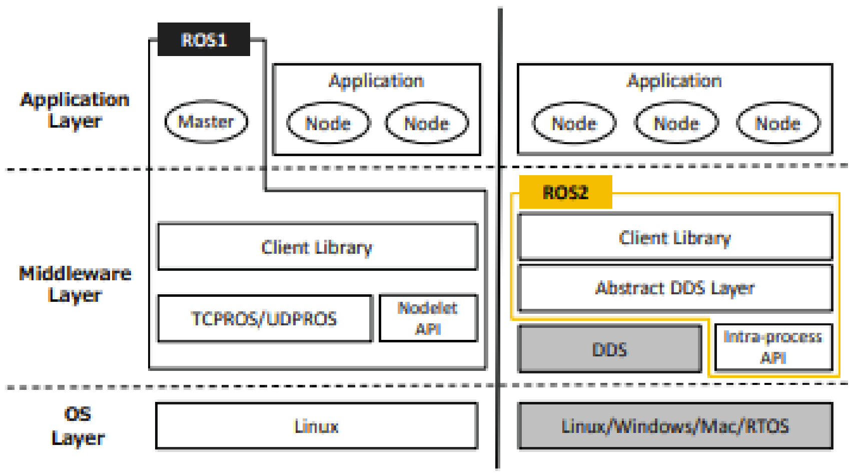


Figure 2.7: Architecture of ROS and ROS2 (Maruyama, Kato and Azumi, 2016).

There are notable differences between ROS and ROS2. While ROS is primarily geared towards Linux-based operating systems, ROS2 is more portable and can be implemented across various operating systems, such as Linux, Mac, Windows, and RTOS. Data transport in ROS is facilitated by TCPROS/UDPROS, and communication is governed by the ROS Master. In contrast, ROS2 leverages the DDS (Data Distribution Service) standard for communication, which boosts fault tolerance capabilities. Moreover, ROS2 offers flexible parameter settings through QoS (Quality-of-Service) control, enabling the reliability of communication to be adjusted. Additionally, in ROS2, every topic possesses the ability to store historical message data. Lastly, intra-process communication in ROS2 provides a better transmission mechanism compared to ROS (Maruyama, Kato and Azumi, 2016).

A novel addition to ROS2 is the Lifecycle Nodes component, which facilitates the proper startup and teardown of ROS2 nodes by utilizing state machine transitions. These nodes possess four distinct states: Unconfigured, Inactive, Active, and Finalized. This new feature guarantees the correct initialization of all nodes prior to system execution during system startup.

Additionally, the ability to perform online node restarting or replacement is made possible with Lifecycle Nodes, which is crucial for enhancing navigation applications within the Navigation Stack.

2.4 ROS2 Navigation Stack (Nav2)

Nav2 is a fresh implementation of the navigation toolbox in ROS2 that is comparable to the navigation metapackages found in ROS but features a new architecture, as illustrated in Figure 2.8.

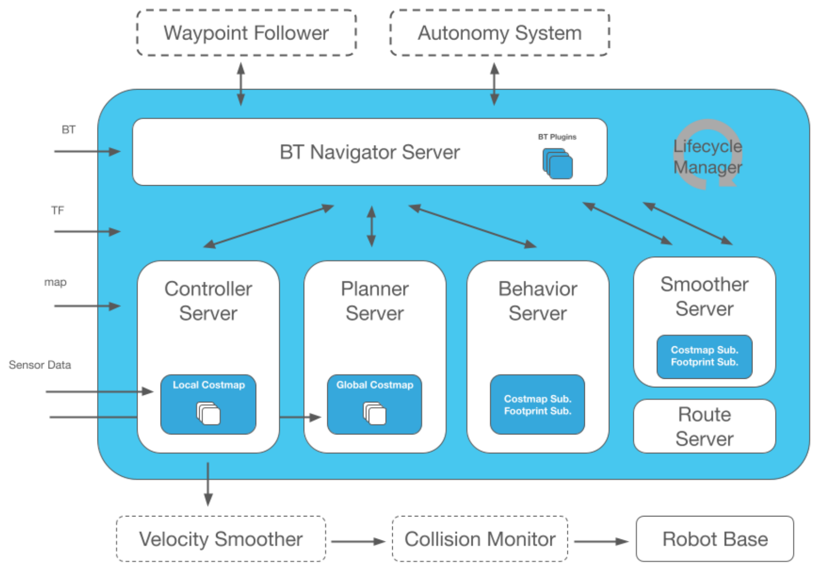


Figure 2.8: Nav Stack (Macenski et al., 2020).

Nav2 provides perception, planning, control, localization, visualization, and more to build highly sophisticated and reliable autonomous systems. It is capable of modelling a complete simulated environment based on sensor data, which leads to dynamic path planning. Nav2's modular and reconfigurable core allows for the inclusion of features like historical message data storage. The core includes a Behavior Tree (BT) navigator and also task-specific asynchronous servers that facilitate this functionality (Macenski *et al.*, 2020).

2.4.1 Behavior Tree (BT) Navigator

BT Navigator uses a behavior tree to have a formal structure for navigation logic. This refers to a task arrangement in the form of a tree structure, in contrast to a finite state machine (FSM) that may comprise numerous states and transitions. However, BT provides better reactivity and modularity in comparison to FSM.

Nav2 utilize the BT navigator upon BehaviorTree CPP V3 as the behavior tree library. The reason for this is that users are able to generate node plugins that can be assembled into a tree structure within the BT Navigator. These node plugins are loaded into the BT, and upon parsing the XML file of the tree, the designated names are linked together. This simplifies the building of complex navigation behaviours starting from the most basic ones.

2.4.2 Navigation Servers

The Planner, Controller, Smoother, and Recovery servers are action servers within Nav2. These action servers utilize several algorithm plugins to complete the assigned task. Typically, the Planner module is responsible for calculating a valid and, possibly, the optimal path from the current position to the goal position, while the Controller module is responsible for determining a control effort that enables the robot to follow the global planning based on the environmental representation. For example, numerous controllers anticipate the robot's trajectory and determine a path that is locally achievable at each update iteration. Recovery behaviours are a mainstay of fault-tolerant systems, which are utilized by the BT when a possible navigation failure occurs. Lastly, the Smoother is utilized to decrease the unevenness of a path and alleviate sudden rotations, all while increasing the distance from obstacles and elevated regions. This is achievable as the smoothers have access and utilized the global environmental representation (Macenski *et al.*, 2020).

2.4.3 Environmental Representation and Costmap2D

Environment representation is how a robot perceives its surrounding environment (Macenski *et al.*, 2020). Moreover, it serves as a central hub for different algorithms and data sources to merge their data into a unified space. This depiction is subsequently employed by controllers, recoveries, and planners to carry out their duties in a safe and efficient manner.

In ROS2, the existing environmental representation takes the form of a costmap. This costmap consists of a standardized 2D grid cells that have been assigned costs based on their classification as unknown, free, occupied, or inflated regions. The generated costmap is then utilized to calculate a global plan or sampled to determine local control efforts.

2.5 Summary

In conclusion, coordinate control for the rendezvous task can take different forms, either distributed or hierarchical in the case of decentralized control. In this project, we will be referring to the hierarchical consensus algorithm due to its simplicity and scalability. Due to its hierarchical structure and capability to handle faults and failures, the algorithm is a promising approach for coordinating large-scale systems with multiple agents. The ultimate objective of the project is to demonstrate the algorithm's effectiveness and practical applicability in a decentralized control scenario by successfully implementing it.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter will discuss about the methodology and the workplan of this research. The methodology will be describing the rendezvous algorithm and the simulation setup in MATLAB and ROS2. The workplan which includes the planning and managing of the project will be illustrated through a Gantt Chart.

3.2 Rendezvous Control Proposed Solution

3.2.1 Rendezvous Problem Setting

The rendezvous problem begins when mobile robots are deployed in an unknown environment without knowledge of their neighbouring robots' coordinates. The mobile robots are required to rendezvous with a pre-determined leader robot or an optimally chosen leader robot determined by the algorithm.

3.2.2 Multi-Point Rendezvous Control Law

At the start of the rendezvous task, each of the robots will build the global connectivity graph in a distributed manner by using **Algorithm 1**. If the number of points, M to rendezvous have been set, the connectivity graph will then have decomposed into M shortest tree path where the leader robot is the root of the tree graph. After that, **Algorithm 2** is then applied to each tree separately and simultaneously to start the rendezvous task. If there is no pre-determined leader among the mobile robots, **Algorithm 3** will be utilized to choose the best leader robots from the leader robots' candidate.

3.2.3 Assumption and Definition

Few assumptions and definition will be introduced to drive the rendezvous control law. Let $q_i \in R^2$ which denote the position of the robot i, where R^2 refers to the two-dimensional Euclidean space, which consists of ordered pairs of position of N mobile robots. Single Integrator Model will be used to describe the dynamics of mobile robot at first, where $\dot{q}_i(t) = v_i(t), i \in V = [1, \dots, N]$.

$v_i(t)$ denotes as the velocity for robot i at the instant time of t, and N is the number of mobile robots.

Firstly, we will assume that each of the mobile robot can only use their local sensor on board to detect and identify their neighbouring robots. No global localization which is centralized control or coordinate reference system is available for the mobile robots. The information which robot sense their neighbouring robots will share across the network and will be used to build and update the network topology. S_R will be defined as the maximum sensing range of the mobile robot, and SSR_i as the safe sensing range of mobile robot i, to enable the mobile robot to move freely as long as their neighbour is within the SSR.

By following graph theory (West, 2018), let $G = (V, E, W)$ denote as an unidirected weighted graph, which contains vertex set V and edge set E. Every edge, e ∈ in E is weight by a positive value and is defined as $W(e) \colon \rightarrow Z^+$. Two vertices are said to be connected when there is a path between them $\{(i,j) \in E \mid j \ \in \ N_i\}$, where N_i denotes as the neighbour sets for the robot i. The set of vertices give rise to subgraph of G where $S \subseteq V$ is the graph (S, E_S, W_S), where $E_S = \ \{(i,j) \in E \mid j \ \in \ S\}$.

A tree graph is a type of connected graph without cycles where a rooted tree has a root node as the highest hierarchy of the graph as shown in Figure 3.1. We define a rooted tree $T = (V_T, E_T, W_T)$, where node in the rooted tree has a parent-child relationship with its neighbouring nodes which depends on the hierarchy. In the rooted tree, we define $p(v)$, as the parent of the robot v, $c(v)$ as the children set of v. In the tree graph, leaf is a node that have no child.

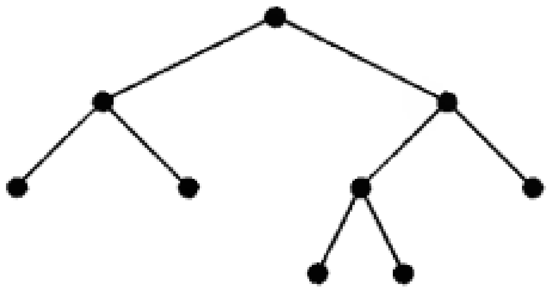


Figure 3.1: Rooted Tree Graph (West,2001).

Let $G(k) = (V_k, E_k, W_k)$ denotes as the connectivity/interaction graph where k is the iteration steps. In the graph G, every node in V represent a mobile robot, and each edge (i, j) in $G(k)$ indicates that two mobile robots i and j, are neighbours. The weight of the edge, $(i, j) \in E_{G(k)}$, is the relative distance between two robots i and j.

The rendezvous problem requires mobile robots to rendezvous into M number of groups, with a predetermined final rendezvous point. We will further assume that M number of leaders will be assigned as the leader of each group, where the leader will be the root of each graph, where $D \subseteq V$. The number of leaders will be denoted as $D_M \in D$. Let $U = V - D$ denote as the subset of mobile robot which are not leader robot and each mobile robot in U will be assigned to one leader from D. Let $R(u_i) \in D$ denote as the leader robot assigned to a robot $u_i \in U$.

The goal of the rendezvous algorithm is to have every mobile robot in U to meet their leader, D, i.e., when $\lim_{t\to\infty} ||q_D - q_i|| = 0,\ \forall i \in V(G) - D$.

3.2.4 Building the Connectivity Graph (Algorithm 1)

The rendezvous task starts with every mobile robot sense their neighbour robots with their local sensor and create a local neighbour set L_i. Every mobile robot will then share its list with neighbours and construct a global connectivity graph by using **Algorithm 1** from the local neighbour set L. This graph provides information regarding the edge and nodes for the mobile robot network. The flow chart of the algorithm is shown in Figure 3.2.

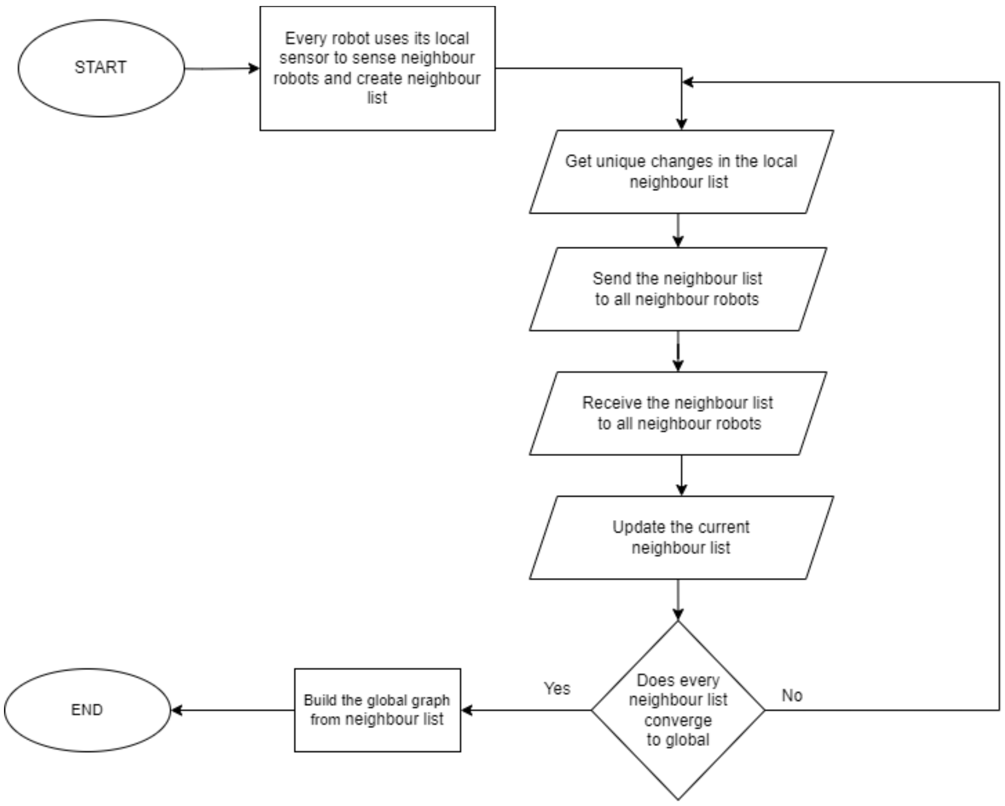

START
Every robot uses its local sensor to sense neighbour robots and create neighbour list
Get unique changes in the local neighbour list
Send the neighbour list to all neighbour robots
Receive the neighbour list to all neighbour robots
Update the current neighbour list
Does every neighbour list converge to global
Yes
No
Build the global graph from neighbour list
END

Figure 3.2: Building a Global Connectivity Graph (Algorithm 1).

3.2.5 Creating Multiple Group

The set of leaders can be selected either based on their advanced capabilities in comparison to the follower or using the **Algorithm 3** by using the connectivity graph to optimize the traversal distance by the mobile robot in the rendezvous process.

As the purpose is to have all follower mobile robot to meet with the leader with the minimum amount of energy and time spent, we denote each mobile robot $u_i \in U$ to select a potential leader robot $R(u_i) \in D$ and optimizing the objective function below.

$$R(u_i) = \arg\min d_{G(K)}(u_i, D_m) \tag{3.1}$$

where

$d_{G(K)}(u_i, D_m)$ is the sum of the path distance between robot i and other robot within the shortest path by using the Dijkstra’s algorithm.

By using the initial global connectivity graph, $G(0)$, M number of sub-graph will be constructed $G_M = (V_M, E_M, W_M)$. By using the sub-graph, M-shortest path tress $T_M = (V_M, E_{T_M}, W_{T_M})$ with the root nodes D_M will be constructed by optimizing the cost of distance from each node to the root node. Each of the node u_i within the graph tree will be assigned to a parent $p(u_i)$ and children c(u_i) if there is any. Please take note that nodes without children is named as leaf nodes.

3.2.6 Hierarchical Rendezvous Algorithm (Algorithm 2)

The purpose of this algorithm is to ensure that all mobile robots in the rooted tree, T_m converge at their root D_m. The algorithm works as follows. Firstly, all non-root robot will move towards their selected immediate parents in the hierarchical tree which the leader is the root node. The mobile robot which has children can move towards their immediate parents if only all of their children is within their safe sensing range, SSR. If not, the robot will then wait until all of their children reach its SSR. After the children robot meet their immediate parent, the children robot will then update their parent to the parent of their parents by updating the edge of the tree from $(u, p(u))$ to $(u, p(p(u)))$ in the T_m.

The iteration will then continue until all non-root's robots gathered at the root robot which is the leader robot. The algorithm also includes the parent u, can only move when all their children are within SSR_u. Figure 3.3 illustrate the flow chart of the **Algorithm 2.**

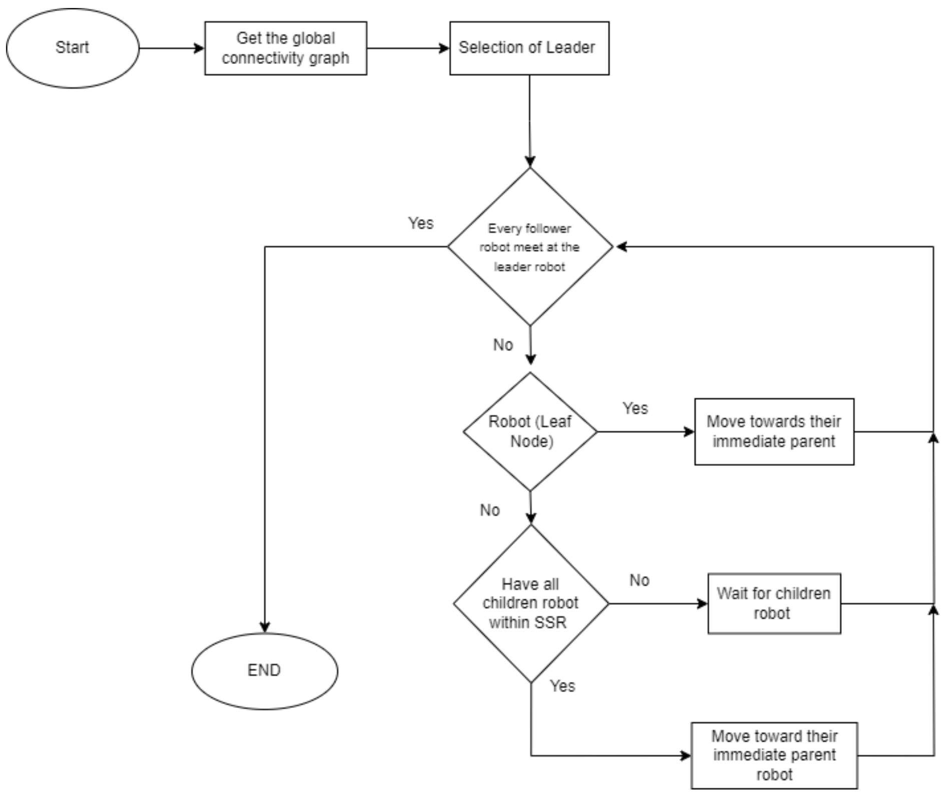

Start
Get the global connectivity graph
Selection of Leader
Every follower robot meet at the leader robot
Yes
No
Robot (Leaf Node)
Yes
Move towards their immediate parent
No
Have all children robot within SSR
No
Wait for children robot
Yes
Move toward their immediate parent robot
END

Figure 3.3: Hierarchical Rendezvous Control Algorithm (Algorithm 2).

3.2.7 Choosing Leader Robots (Algorithm 3)

In situation where no leader robot is predetermined based on their function, this algorithm will be utilized to choose M number of leader robots among the mobile robots in the network. At first, we will utilize the initial connectivity graph, $G(0)$ to create a table T_b, that contains a N-by-N matrix which contains the information of the shortest path distance between each of the robot pair, with each row represent one source/potential leader robot. By using T_b, we can choose M number of leader robots from out of N robots.

As for determining which mobile robot will be suitable for the role of the leader, we should relate to the goal of rendezvous, which is to achieve rendezvous in the shortest time possible. Therefore, by minimizing the maximum distance travelled from neighbour robots to the leader robot, we are able to use this concept to create a cost function. The cost function that will be used to find the optimal leader is as below

$$C(D) = \underset{\forall D_m \in D}{max} \underset{\forall u \in V_m}{min} d_{G(k)}(u, D_m) \tag{3.2}$$

The cost function is constraints by

$$L_m \leq |V_m| \leq U_m \tag{3.3}$$

and

$$d_{G(0)}\big(u, R(u)\big) \leq T_D \text{ }, \forall u \in (V - D) \tag{3.4}$$

The first constraint, (3.3) is to balance of all the group size of M whereas the second constraints, (3.4) is to restrict the maximum travelled by any robot in the rendezvous task. Other than that, we will be utilizing Brute-Force Search solution to go through which maximum distance from the neighbour robots are the smallest with the following equation.

$$D^* = \underset{D_m \in D_{pl}}{\arg min} \; C(D) \tag{3.5}$$

where

$D_{pl} = \binom{V_{pl}}{M}$ is the set of every robot’s solution for D, and D* is the optimal set of leader robots. Potential leader is denoted as pl.

Figure 3.4 shows the flow chart of the Algorithm 3, which is to calculate D^* to determine the suitable leader robot with the number of groups predetermined. Take note that the number of groups is the same of the number of leader robot selected.

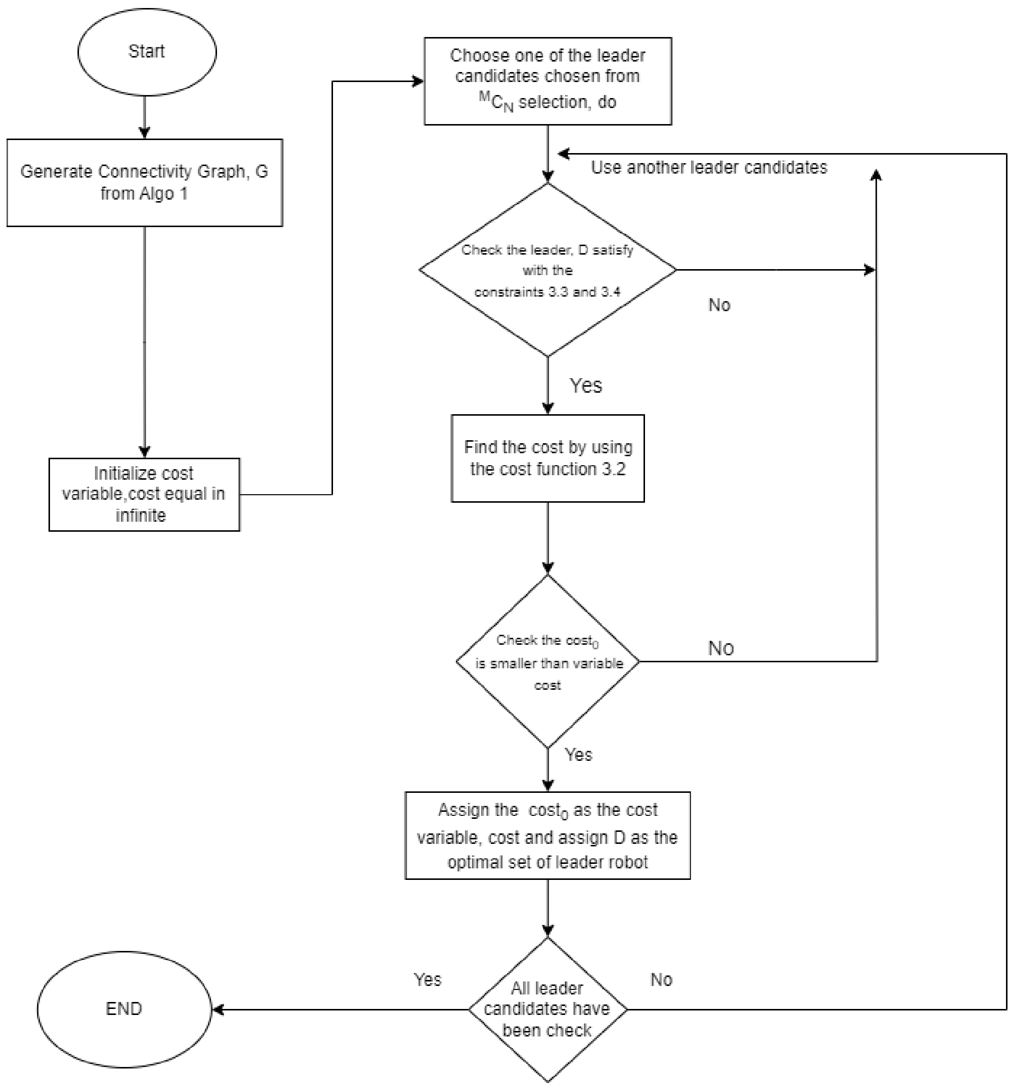

Start
Generate Connectivity Graph, G from Algo 1
Initialize cost variable,cost equal in infinite
Choose one of the leader candidates chosen from $^{M}C_{N}$ selection, do
Use another leader candidates
Check the leader, D satisfy with the constraints 3.3 and 3.4
No
Yes
Find the cost by using the cost function 3.2
Check the $cost_0$ is smaller than variable cost
No
Yes
Assign the $cost_0$ as the cost variable, cost and assign D as the optimal set of leader robot
All leader candidates have been check
Yes
No
END

Figure 3.4: Choosing Leader Robots (Algorithm 3).

3.3 Environmental Setup

This project will be utilizing MATLAB 2022a as the programming platform for the development of the proposed rendezvous algorithm and ROS2 for the implementation and validation of the proposed algorithm.

MATLAB is one of the most used programming platforms for simulation of different kind of systems and it is capable of simulating robotics systems. The strength of MATLAB is that there are toolboxes which target their own specific field and particular topics. Toolboxes may be available from MathWorks, third parties' company or individuals. This project will be utilizing Robotics System Toolbox and Navigation Toolbox created by MathWorks, and Mobile Robotics Simulation Toolbox by the MathWorks Student Competitions Team. Other than that, MATLAB offers a wide range of built-in functions for mathematical computations and simulations, making it suitable for algorithm development.

Aside from MATLAB, ROS2 is a flexible and powerful framework that enables the creation of distributed and modular robotics applications. With ROS2, robot behavior can be simulated and tested in a virtual environment before deploying on physical hardware, which provides a safe and cost-effective way to validate algorithms and optimize system performance.

Therefore, the combination of MATLAB and ROS2 offers a potent platform for the development, simulation, and validation of robotic systems and algorithms, making it possible to implement the proposed rendezvous algorithm efficiently and effectively using MATLAB's toolboxes and built-in functions along with ROS2's simulation tools and packages.

3.4 MATLAB Toolbox

This section will briefly introduce the toolbox used during the simulation in MATLAB.

3.4.1 Robotics System Toolbox

The Robotics Toolbox offers a comprehensive collection of functions that are essential for various robotics applications, covering areas such as kinematics, dynamics, and trajectory generation. Additionally, the Robotics System Toolbox includes algorithms that are specifically designed for mobile robots, encompassing tasks such as mapping, planning, and control. With the Robotics Toolbox, developers can easily access a wide range of pre-built functions and algorithms to develop complex robotics systems, speeding up the development process and improving the overall performance of the system.

3.4.2 Navigation Toolbox

The Navigation Toolbox in MATLAB is mainly used for creating, loading and visualising map which is used for navigation task for mobile robots. Maps are essential components within this project, as they provide a representation of the environment that the robot used for localization, path planning, and navigate.

3.4.3 Mobile Robotics Simulation Toolbox

This toolbox is capable of providing utilities for robot simulation and algorithm development. This toolbox is selected mainly because of the capabilities of configurating LIDAR and robot detector simulators within the occupancy grid maps, which is required to build the connectivity graph at the start of the algorithm. Other than that, it is also used for the 2D kinematics models for robot geometrics such as differential drive.

3.5 ROS2

This section will briefly introduce the toolboxes used in the simulation and workflow in ROS2.

3.5.1 Gazebo

Gazebo is a widely used multi-robot simulator that is fully open-source and supports a vast array of sensors and objects. Its design allows for the accurate reproduction of the dynamics of complex environments that robots may encounter, making it an excellent tool for testing and validating robot behavior in a simulated environment. Gazebo's versatility and extensibility make it a popular choice among researchers and developers working on various robotics applications, from simple single-robot simulations to complex multi-robot systems. By using Gazebo, developers can reduce the time and cost associated with physical testing and gain valuable insights into the behavior and performance of their robotic systems.

To simulate the rendezvous task of multi robots in Gazebo, an environment and a robot model is built which is shown in Figure 3.5 and Figure 3.6 respectively. The robot model is built with a differential drive control, which consist of 2 main wheels, and a chassis wheel. The robot model also consists of a LIDAR sensor to detect nearby object.

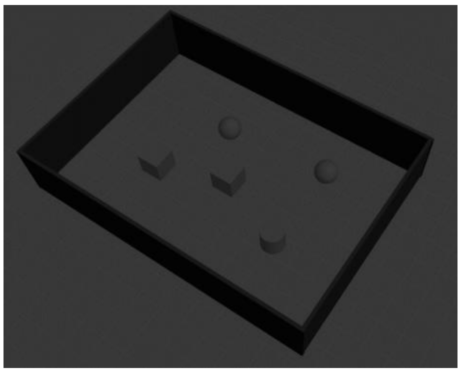


Figure 3.5: Obstacle-Rich Environment.

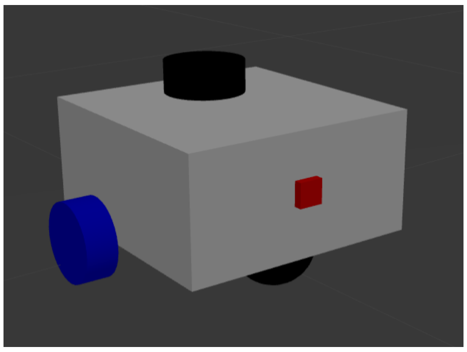


Figure 3.6: Differential Drive Mobile Robot.

3.5.2 ROS2 Navigation Stack (Nav2)

The costmap layer, controller, planner, behavior tree, and behavior plugins are among the plugin interfaces available to users for developing their own customized applications or algorithms. The navigation plugins used is tabulated in Table 3.1.

Table 3.1: List of Navigation Plugins Used.

Navigation Plugins	**Plugin Name**
Behavior-Tree Navigator	“Navigation To Pose With Consistent Replanning And If Path Becomes Invalid”.
Costmap Layers	Voxel Layer, Inflation Layer, Static Layer
Controller	DWB Controller
Planner	NavFn Planner
Smoother	Simple Smoother

3.5.3 SLAM Toolbox

The SLAM Toolbox used is developed by Steven Macenski, which is a set of tools and capabilities for 2D SLAM (Macenski and Jambrecic, 2021). By using the SLAM Toolbox, mapping can be done which mainly uses the sensor data and the odometry of the mobile robot. The map generated is known as global costmap, which will be stored in the map server. Therefore, in a multi-robot simulation setup, all mobile robots including the leader robot will share a common map server, but will have their own individual local costmap, which is generated by their own real-time sensor data.

There are four types of mapping mode in SLAM Toolbox. Online asynchronous mode is chosen because when running the SLAM algorithm, the sensor data are processed asynchronously as they arrive, rather than waiting for a complete set of datasets before processing. In comparison to offline mode, the entire dataset is processed at once, which can be computationally expensive and time-consuming. Asynchronous mode is preferred as it allow SLAM algorithm to update the map and estimate the robot’s pose in real-time.

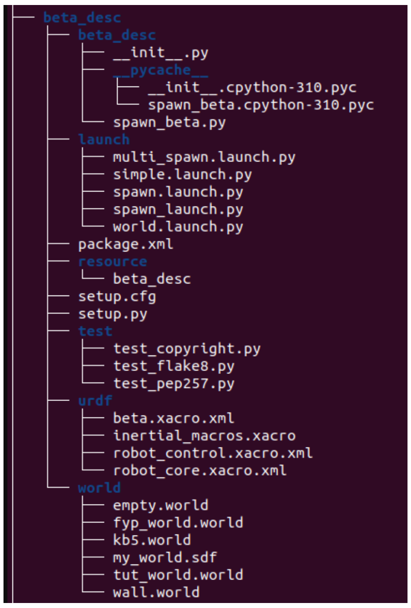
3.5.4 Transform Library 2 (tf2)

A robotics system typically uses multiple 3D coordinate frames, including the base frame and joint frame, that will change over time. Tf2 is a tool that can keep track of all these frames , and it can work within a distributed system. Therefore, ROS components on any computer within the system can access information about the coordinate frames of a robot. This is useful because we are able to determine the coordinate of each mobile robot and the distance between each robot which is required in our algorithm, and also implement additional formation control during the rendezvous task.

3.5.5 Workflow in ROS2

In this project, Linux Ubuntu 22.04 operating system was installed and used. Following the instructions on the official documentation website, ROS2 Humble which is one of the latest distributions, has been installed from the Debian packages. Three workspaces have been set up to maintain organized management of the source code, and Python has been used to create the packages. Within the Python package, package.xml file contain meta information about the package, setup.py containing instruction for how to install the packages, and setup.cfg is required when a package has executables.

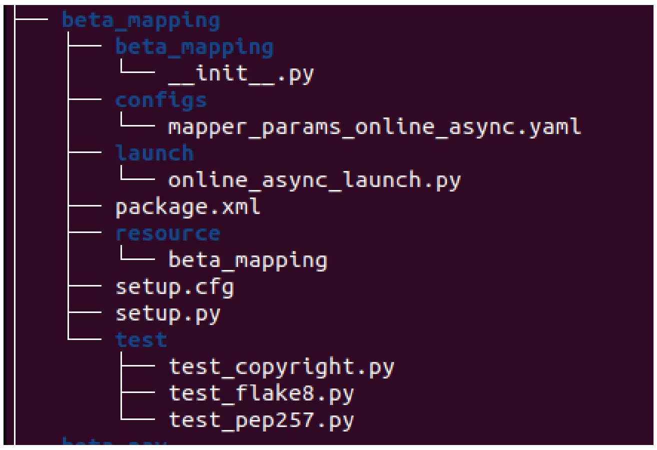
i. beta_desc – Within this workspace, it contains the URDF model of the differential drive mobile robot and the obstacle-rich environment, together with the launch file which is responsible for spawning multiple robots in Gazebo. The tree directory of the workspace beta_desc is shown in Figure 3.7.



```
├── beta_desc
│   ├── beta_desc
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-310.pyc
│   │   │   └── spawn_beta.cpython-310.pyc
│   │   └── spawn_beta.py
│   ├── launch
│   │   ├── multi_spawn.launch.py
│   │   ├── simple.launch.py
│   │   ├── spawn.launch.py
│   │   ├── spawn_launch.py
│   │   └── world.launch.py
│   ├── package.xml
│   ├── resource
│   │   └── beta_desc
│   ├── setup.cfg
│   ├── setup.py
│   ├── test
│   │   ├── test_copyright.py
│   │   ├── test_flake8.py
│   │   └── test_pep257.py
│   ├── urdf
│   │   ├── beta.xacro.xml
│   │   ├── inertial_macros.xacro
│   │   ├── robot_control.xacro.xml
│   │   └── robot_core.xacro.xml
│   └── world
│       ├── empty.world
│       ├── fyp_world.world
│       ├── kb5.world
│       ├── my_world.sdf
│       ├── tut_world.world
│       └── wall.world
```

Figure 3.7: Tree Directory of beta_desc.

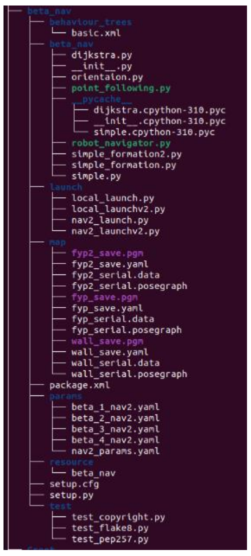
ii. beta_mapping – This workspace contains the parameter used for the SLAM Toolbox, and the launch file of the online asynchronous SLAM. The tree directory of the workspace beta_mapping is shown in Figure 3.8.



```
├── beta_mapping
│   ├── beta_mapping
│   │   └── __init__.py
│   ├── configs
│   │   └── mapper_params_online_async.yaml
│   ├── launch
│   │   └── online_async_launch.py
│   ├── package.xml
│   ├── resource
│   │   └── beta_mapping
│   ├── setup.cfg
│   ├── setup.py
│   └── test
│       ├── test_copyright.py
│       ├── test_flake8.py
│       └── test_pep257.py
```

Figure 3.8: Tree Directory of beta_mapping.

iii. beta_nav – In the beta_nav workspace, it contains maps which is generated by the SLAM Toolbox, parameters of the Nav2 plugins for each of the mobile robot, launch file of localization (AMCL) and Nav2, behavior tree plugin, and the rendezvous controller named as simple_formation2.py and dijkstra.py. The tree directory of the workspace beta_nav is shown in Figure 3.9.



```
├── beta_nav
│   ├── behaviour_trees
│   │   └── basic.xml
│   ├── beta_nav
│   │   ├── dijkstra.py
│   │   ├── __init__.py
│   │   ├── orientaion.py
│   │   ├── point_following.py
│   │   ├── __pycache__
│   │   │   ├── dijkstra.cpython-310.pyc
│   │   │   ├── __init__.cpython-310.pyc
│   │   │   └── simple.cpython-310.pyc
│   │   ├── robot_navigator.py
│   │   ├── simple_formation2.py
│   │   ├── simple_formation.py
│   │   └── simple.py
│   ├── launch
│   │   ├── local_launch.py
│   │   ├── local_launchv2.py
│   │   ├── nav2_launch.py
│   │   └── nav2_launchv2.py
│   ├── map
│   │   ├── fyp2_save.pgm
│   │   ├── fyp2_save.yaml
│   │   ├── fyp2_serial.data
│   │   ├── fyp2_serial.posegraph
│   │   ├── fyp_save.pgm
│   │   ├── fyp_save.yaml
│   │   ├── fyp_serial.data
│   │   ├── fyp_serial.posegraph
│   │   ├── wall_save.pgm
│   │   ├── wall_save.yaml
│   │   ├── wall_serial.data
│   │   └── wall_serial.posegraph
│   ├── package.xml
│   ├── params
│   │   ├── beta_1_nav2.yaml
│   │   ├── beta_2_nav2.yaml
│   │   ├── beta_3_nav2.yaml
│   │   ├── beta_4_nav2.yaml
│   │   └── nav2_params.yaml
│   ├── resource
│   │   └── beta_nav
│   ├── setup.cfg
│   ├── setup.py
│   └── test
│       ├── test_copyright.py
│       ├── test_flake8.py
│       └── test_pep257.py
```

Figure 3.9: Tree Directory of beta_nav.

Unlike the simulation in MATLAB, many preparations needed to be done before implementing the rendezvous control algorithm within the multi-robot system. Figure 3.10 shows the workflow of overall process.

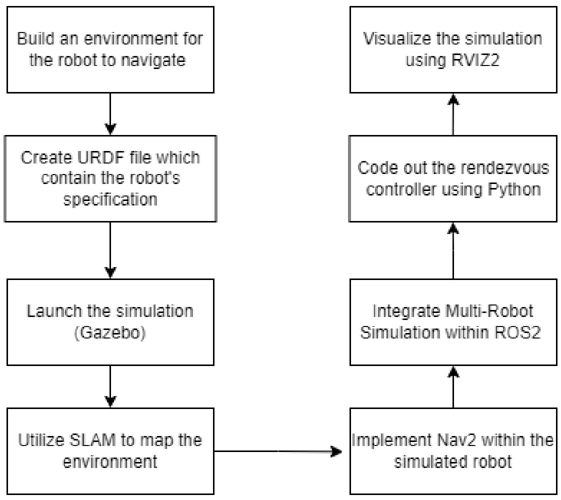

Build an environment for the robot to navigate
Create URDF file which contain the robot's specification
Launch the simulation (Gazebo)
Utilize SLAM to map the environment
Implement Nav2 within the simulated robot
Integrate Multi-Robot Simulation within ROS2
Code out the rendezvous controller using Python
Visualize the simulation using RVIZ2

Figure 3.10: Workflow of Overall Process.

The rendezvous controller is built by writing a Python script. Within the python script, a node named */goal_publisher* is built. The node is firstly subscribed to the *tf* node, which is used to determine the coordinate of each of the mobile robot and the distance between each of them. Dijkstra's algorithm is then implemented to determine which mobile robot is suitable to become the leader. After that, by publishing the coordinate of the leader to a topic named /*goal_pose* to each follower robot, the follower robot will then follow the path built by Nav2 to rendezvous with the leader. As the follower robot move towards the safe sensing range of the leader, then the */goal_publisher* will then send the coordinate of the final goal to the leader by publishing to */goal_pose.* Figure 3.11 illustrate the node graph of */goal_publisher*. A terminal is used to run the rendezvous controller, shown in Figure 3.12. The python script will log the process of the rendezvous task along the way.

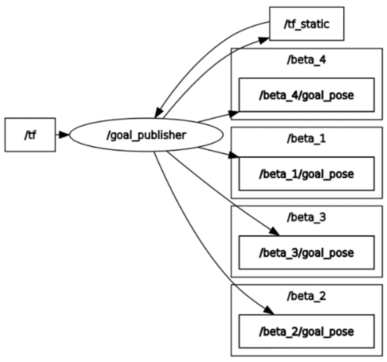
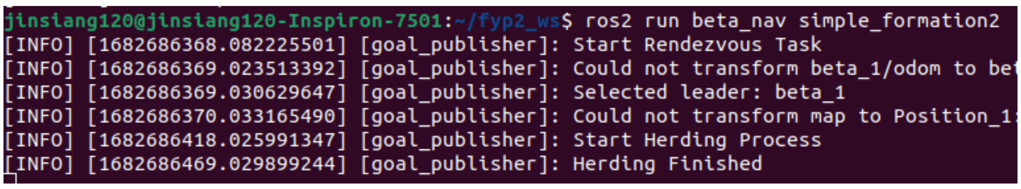


Figure 3.11: Node graph of */goal_publisher*.



```
jinsiang120@jinsiang120-Inspiron-7501:~/fyp2_ws$ ros2 run beta_nav simple_formation2
[INFO] [1682686368.082225501] [goal_publisher]: Start Rendezvous Task
[INFO] [1682686369.023513392] [goal_publisher]: Could not transform beta_1/odom to bet
[INFO] [1682686369.030629647] [goal_publisher]: Selected leader: beta_1
[INFO] [1682686370.033165490] [goal_publisher]: Could not transform map to Position_1:
[INFO] [1682686418.025991347] [goal_publisher]: Start Herding Process
[INFO] [1682686469.029899244] [goal_publisher]: Herding Finished
```

Figure 3.12: Terminal of Rendezvous Controller.

A simple formation control strategy is implemented within the rendezvous controller. When the rendezvous task begins and the selection leader is done, position of follower robot is set behind the leader robot, which is shown in Figure 3.13.

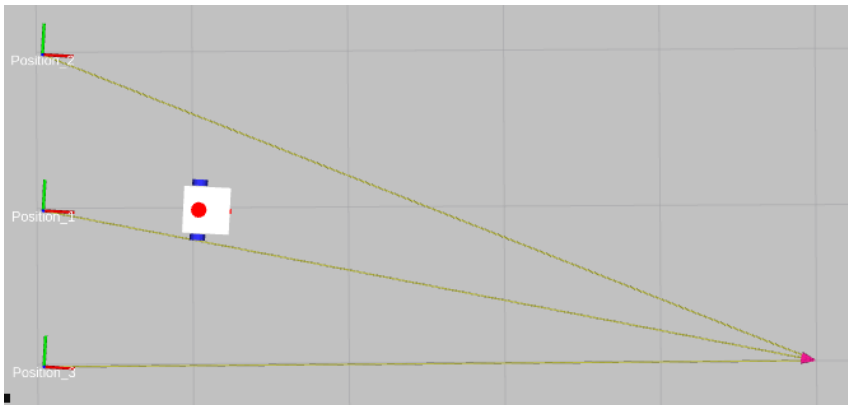


Figure 3.13: Position for Follower Robot.

3.6 Planning and Managing of Project Activities

This section describes the planning of project with the consideration of resources and time. Table 3.2 shows the Gantt Chart of the FYP 1 and Table 3.3 shows the Gantt Chart of the FYP 2.

Table 3.2: Gantt Chart of Project Part I.

No	Week / Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
1	Problem Statement and Project Planning	✓	✓												
2	Literature Review		✓	✓	✓	✓									
3	Analysis of Methodology of Project			✓	✓	✓	✓	✓	✓						
4	Report Writing								✓	✓	✓	✓	✓	✓	
5	Simulation and Coding										✓	✓	✓	✓	✓
6	Presentation and Report Submission														✓

Table 3.3: Gantt Chart of Project Part II.

No	**Week** **Project Activities**	**W1**	**W2**	**W3**	**W4**	**W5**	**W6**	**W7**	**W8**	**W9**	**W10**	**W11**	**W12**	**W13**	**W14**
1	Simulation of Project	✓	✓	✓	✓	✓	✓	✓	✓	✓					
2	Report Writing								✓	✓	✓	✓	✓	✓	
3	Preparation of FYP Poster										✓	✓			
4	FYP Presentation and Report Submission														✓

3.7 Summary

The rendezvous task of the mobile robot will be simulated using MATLAB as the programming platform. First, a connectivity graph will be built after the mobile robot is deployed randomly without knowing their initial location. Then, based on the predetermined number of groups set, a rooted tree graph is built with the leader as the root node. Each of the rooted tree graphs will then start the rendezvous process, where the leaf node (the follower) starts to meet its immediate parent. The process ends when every follower robot meets with the leader robot. The proposed algorithm will then be implemented in ROS2 by coding out the rendezvous controller.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

In this section, we present several simulation scenarios of the rendezvous task, along with the evaluation metrics used for the proposed algorithm. We then describe each scenario and discuss the achieved results.

4.2 Simulation in MATLAB

There are three scenarios used to verify and demonstrate the performance of the proposed rendezvous algorithm.

1. Single-Point Rendezvous
2. Multi-Point Rendezvous
3. Herding

Scenario 1 is used to verify the performance of the proposed algorithm in terms of scalability and convergence and compared to the other rendezvous algorithm. In Scenario 2, the capability of the multi leader function of the proposed algorithm is demonstrated and in Scenario 3, it shows that herding behaviour which the leader robots lead their follower robots to the final rendezvous location.

4.2.1 Scenario 1 – Single-Point Rendezvous

In this scenario, we investigate the performance and convergence properties of the algorithm when the number of leaders (M) is 1 and the number of follower robots (N) is varied between 10, 20, and 30. Additionally, we compare the performance of the proposed algorithm against the circumcenter algorithm mentioned in the literature review. The initial positions of the robots and the corresponding connectivity graph are shown in Figures 4.1, 4.2, and 4.3.

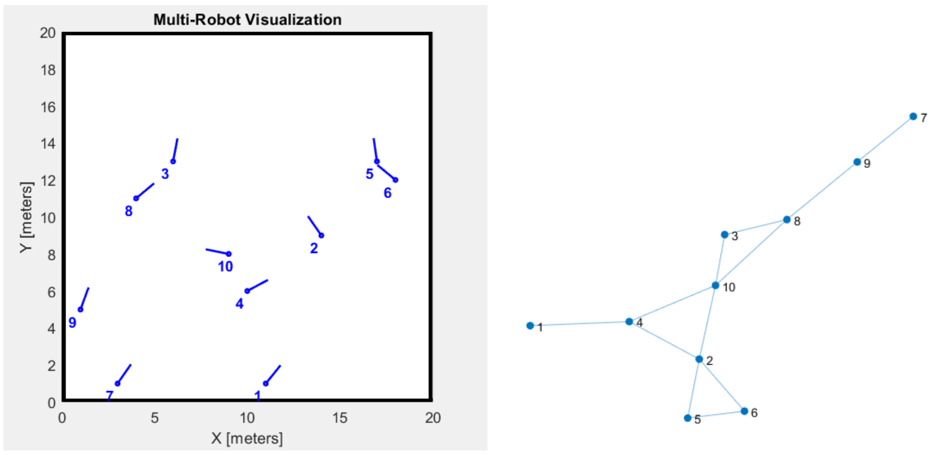


Figure 4.1: Initial Position of Mobile Robots with N =10 (Left) and Initial Connectivity Graph (Right).

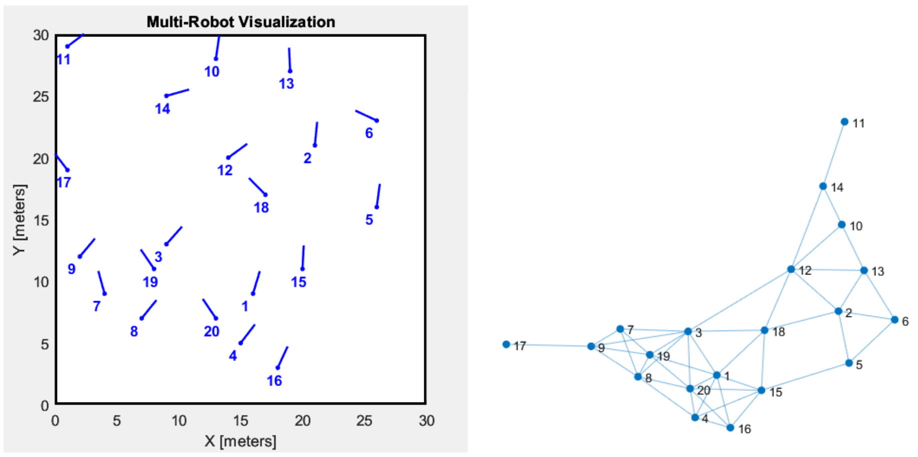


Figure 4.2: Initial Position of Mobile Robots with N = 20 (Left) and Initial Connectivity Graph (Right).

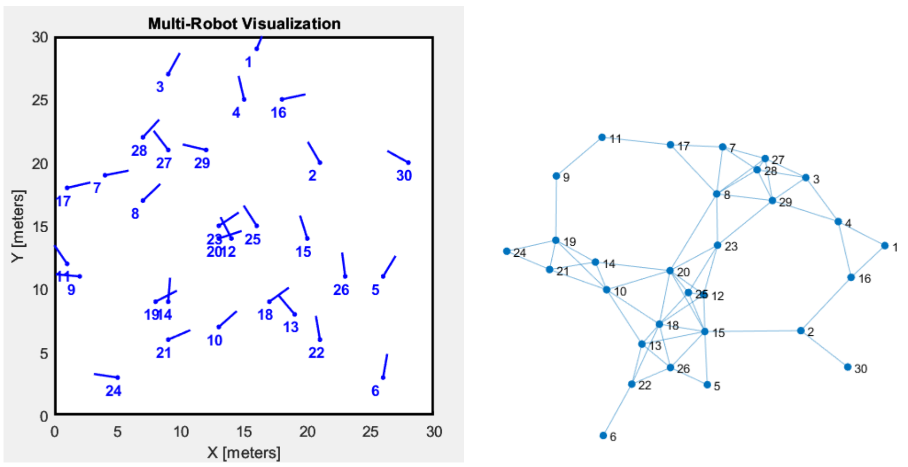


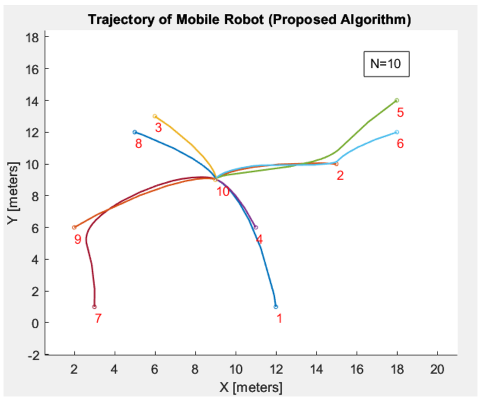
Figure 4.3: Initial Position of Mobile Robots with N = 30 (Left) and Initial Connectivity Graph (Right).

The results of the metrics are tabulated in Table 4.1, which includes total distance travelled and the time taken for the rendezvous task to finish. From Table 4.1, we can see that the proposed algorithm's total distance travelled is much better than the circumcenter algorithm, and the trajectory shown in Figure 4.4 (a), (c) and (e) is much smoother than the circumcenter algorithm illustrated in Figure 4.4 (b), (d) and (f). This is mainly because the proposed algorithm requires the root robot to rendezvous with their immediate parents and continue to rendezvous at the selected leader robot as shown in Figure 4.5. In contrast, for the circumcenter algorithm, each robot will compute a centre location to move by using their neighbour location, resulting in different points in different iterations, and hence leads to rougher trajectory. However, the time taken for the proposed algorithm is slightly slower than the circumcenter algorithm, which is mainly because the immediate parents need to wait for their child robot to reach to their safe sensing range before moving to their own immediate parents.

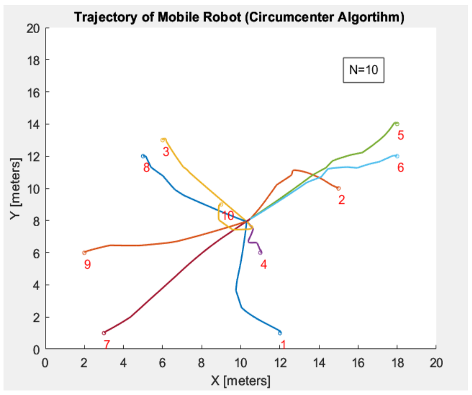
Overall, the proposed algorithm guarantees connectivity maintenance of connectivity graph, resulting in a high convergence rate which is similar to the circumcenter algorithm.

Table 4.1: Results of the Metrics in Scenario 1.

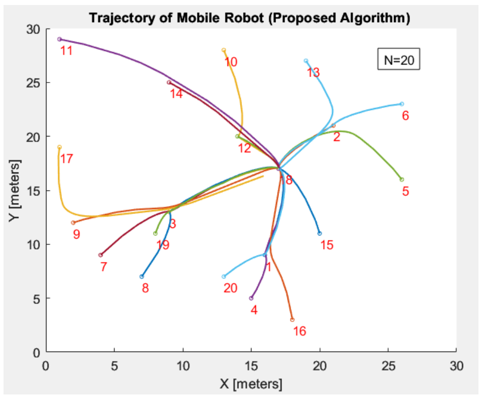
Algorithm	**Metric**	**N = 10**	**N = 20**	**N = 30**
Proposed Algorithm (M = 1)	Total Distance (m)	76.65	234.00	293.58
	Time (s)	51.4	76.7	59.5
Circumcenter Algorithm	Total Distance (m)	79.68	274.80	354.33
	Time (s)	36.4	58.4	54.1



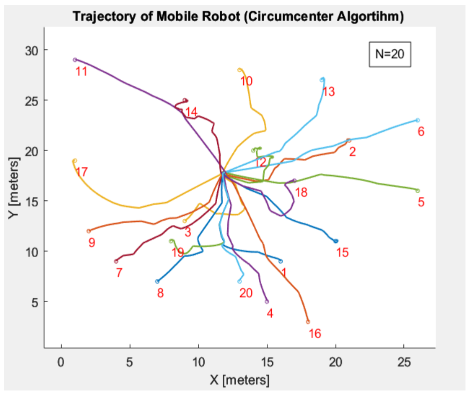
(a)



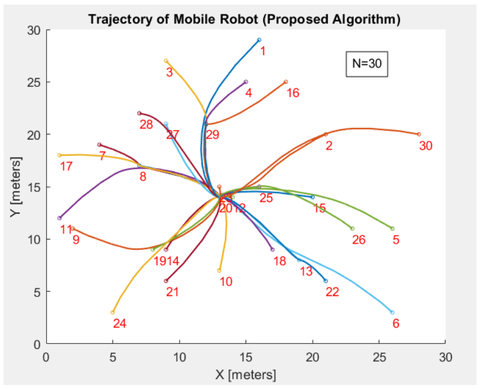
(b)



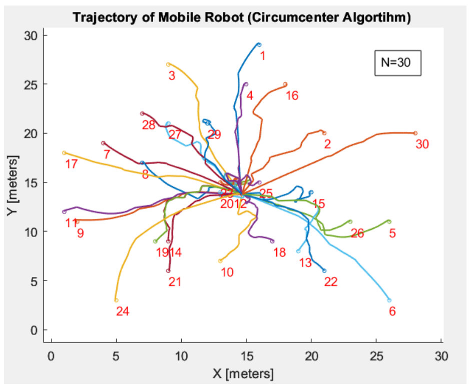
(c)



(d)



(e)



(f)

Figure 4.4: The results of Scenario 1 is presented – Proposed rendezvous algorithm (left) is compared to the circumcenter algorithm (right) with different numbers of mobile robots. The first row of figure, (a) and (b), displays the final trajectory when N = 10. The second row (c) and (d) displays the final trajectory when N = 20, and the third row (e) and (f) illustrates the final trajectory when N = 30.

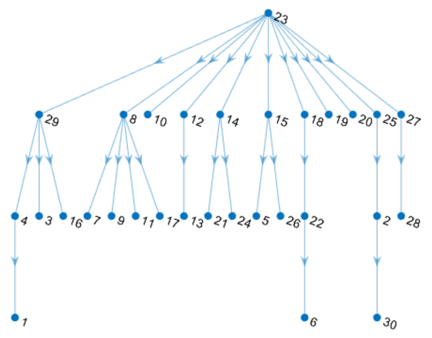
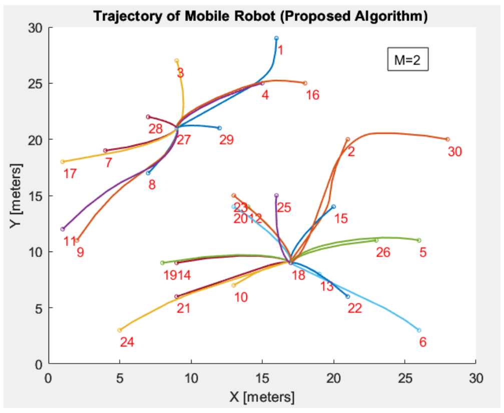


Figure 4.5: Rooted Tree of Proposed Algorithm (N=30).

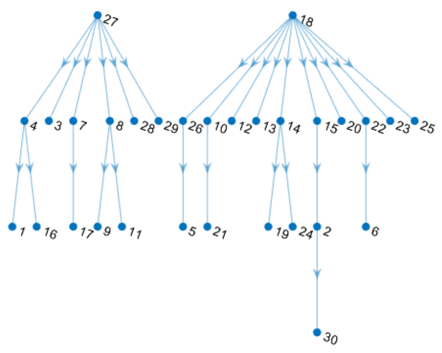
4.2.2 Scenario 2 – Multi-Point Rendezvous

The performance of the multi-point rendezvous with N = 30 is tested with different numbers of rendezvous points (M=2, 3, 4) to demonstrate how the proposed algorithm scales with the number of rendezvous points. Table 4.2.3 shows the trajectory of the mobile robots and the rooted tree. When the number of groups, M = 2, the two groups have sizes of 12 and 18 mobile robots, respectively. When M = 3, the groups had 10, 13, and 7 mobile robots, and the number of groups is set as M = 4, the number of mobile robots in each group were 5, 7, 8, and 10. The trajectory of all the mobile robots is smooth, similar to when M = 1, and they converge towards their leader robots. The number of followers within these groups varies in these scenarios. This is because the algorithm focuses on minimizing the cost function to have the least total distance travelled possible.

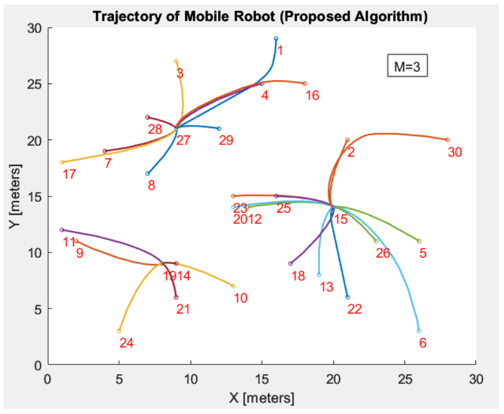
In Table 4.2, it can be observed that as the value of M increases, the total distance travelled, and the time taken to complete the rendezvous task decrease. Figure 4.7 shows that the convergence rate is significantly faster with M being 2 or higher compared to the circumcenter algorithm. Therefore, depending on the situation, it is advantageous to utilize a higher number of groups with ensured connectivity maintenance. The main reason for the decrease in total distance travelled is that follower robots have a shorter distance to the selected leader robot, which serves as the rendezvous point. Consequently, the rendezvous location varies in each scenario, and with a lower value of M, the rendezvous location tends to be closer to the center of the connectivity graph.



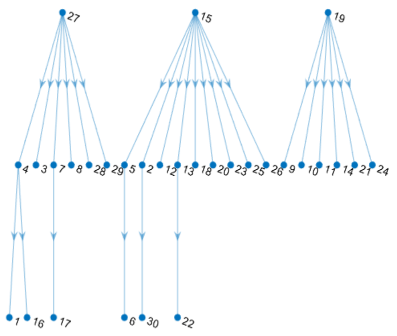
(a)



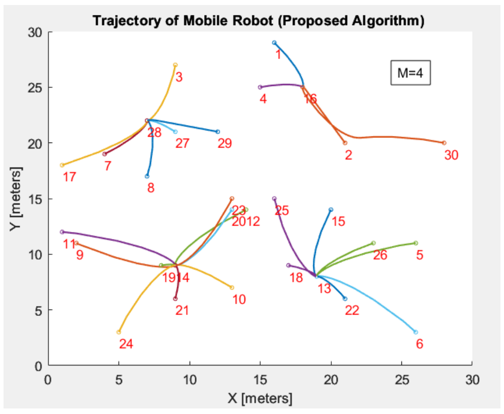
(b)



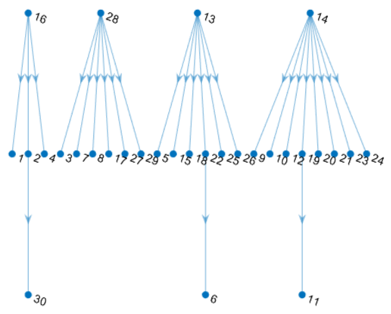
(c)



(d)



(e)



(f)

Figure 4.6: Results of Scenario 2 – Multi-point rendezvous where N = 30. At the first row, (a) show the final trajectory of mobile robots and (b) shows the initial rooted tree. At the second row, (a) show the final trajectory of mobile robots and (b) shows the initial rooted tree.

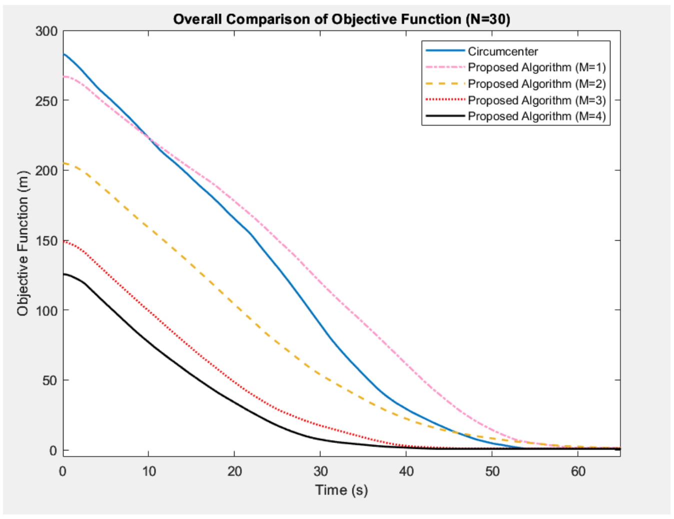


Figure 4.7: Overall Comparison of Objective Function (N=30).

Table 4.2: Results of the Metrics in Scenario 2.

Metric	M=2	M=3	M=4
D (m)	226.3200	180.75	148.71
Time (s)	64.8	51.9	42.3

4.2.3 Scenario 3 – Herding

In this scenario, the proposed algorithm will demonstrate the capability of the robot leaders of showing herding behaviour by providing two specific goals that are visualized as a red mark in the environment. The leader robots are expected to lead their followers towards the end goals. The scenario is simulated with N = 16 and M =2 as shown in Figure 4.8.

Figure 4.9 shows the steps of trajectory step in Scenario 3. The herding behaviour are achieved by setting up a Pure Pursuit Controller to the leaders which is provided by the Navigation Toolbox. The leader robots will start to move as their immediate children is within their safe sensing range. In Figure 4.9 (a) and (b), which demonstrate the leader robots (4,8) starts to herd and move to the final goal. As shown in Figure 4.9 (c), the mobile robots achieved rendezvous while herding to the final goal.

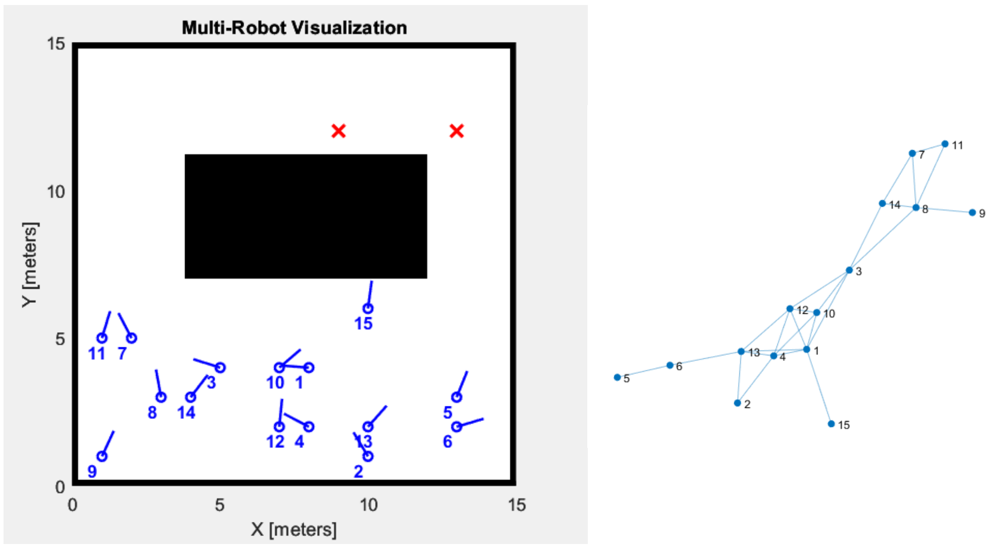
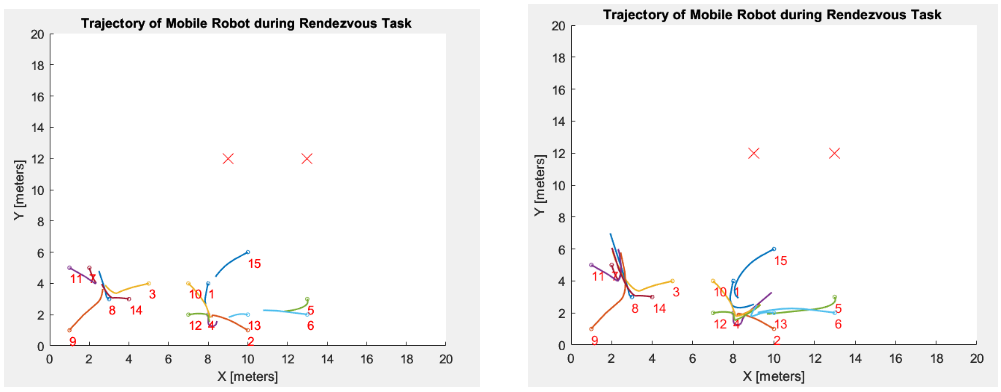
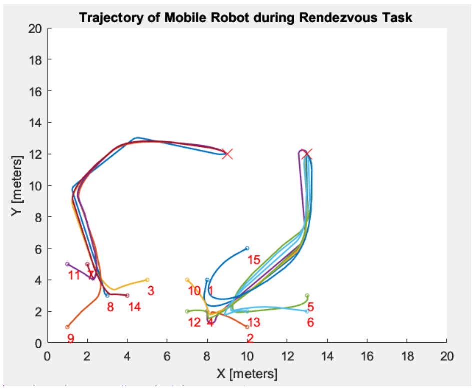


Figure 4.8: Initial Position of Mobile Robots with N = 16 (Left) and Initial Connectivity Graph (Right).



(a) t = 35

(b) t = 80



(c) t = 560

Figure 4.9: Trajectory of Mobile Robots during Herding at iterations (a) t = 35, (b) t = 80, (c) t = 580.

4.3 Simulation in ROS2

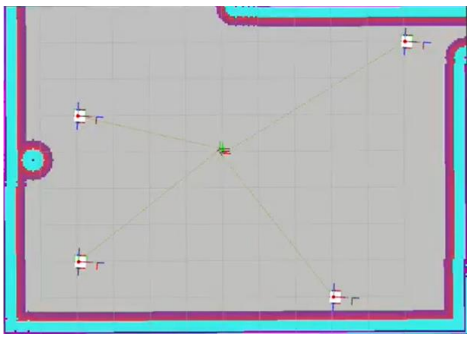
The proposed algorithm is then implemented in ROS2. Two scenarios are used to verify and validate the performance of the algorithm.

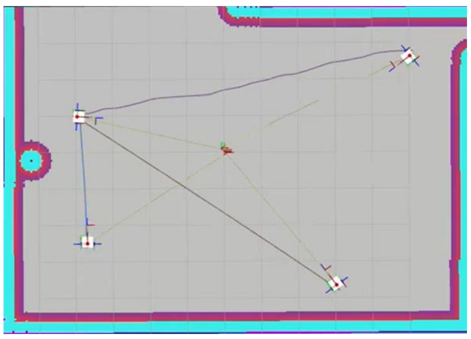
1. Obstacle-Free Environment
2. Obstacle-Rich Environment

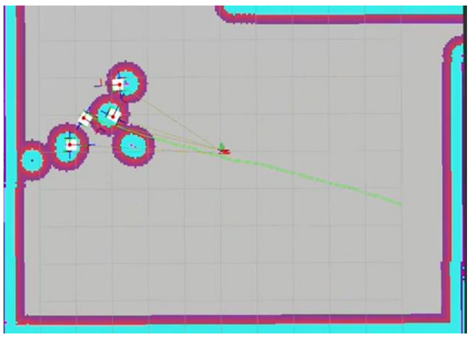
Two scenarios are presented in this project. In Scenario 1, mobile robots are deployed in an obstacle-free environment to perform a proposed algorithm that achieves a specific task. The proposed algorithm is evaluated by comparing the scenario with and without formation control. The formation control enables the robots to maintain a desired formation while performing the task. In Scenario 2, the capability of the mobile robots to navigate around static obstacles while performing the rendezvous task is demonstrated. The rendezvous task requires the robots to meet at a specific location. Similar to Scenario 1, both with and without formation control are implemented, and the proposed algorithm is evaluated. The results of the evaluation provide insights into the effectiveness of the proposed algorithm in different scenarios.

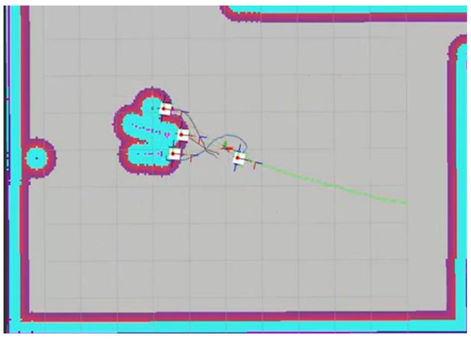
4.3.1 Scenario 1 – Obstacle-Free Environment

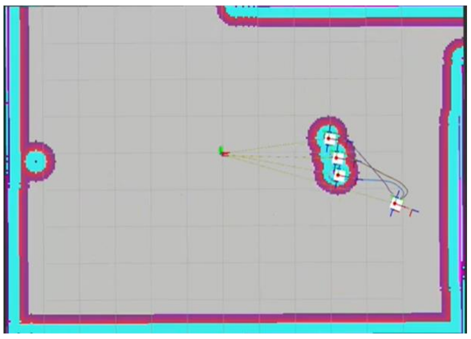
In Scenario 1, the mobile robots are expected to meet with the leader robot, where the leader robot are selected using the Dijkstra's algorithm. When all the follower robot reaches the safe sensing range of the leader robot, the leader will start leading the follower robot to the final location. Two different approaches have been simulated within this scenario: one is without formation control, and other with formation control. The differences are clearly shown after the rendezvous at the leader's position. The result of the simulation is shown in Figure 4.10 and Figure 4.11.


(a)


(b)


(c)


(d)


(e)

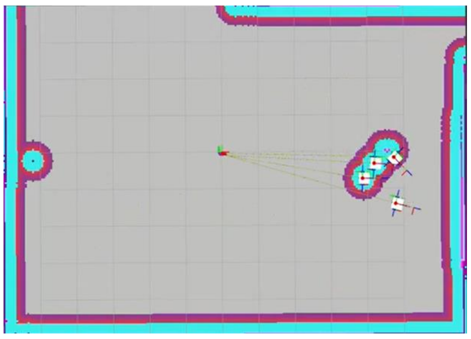
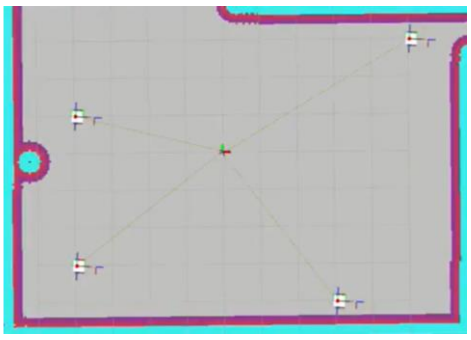
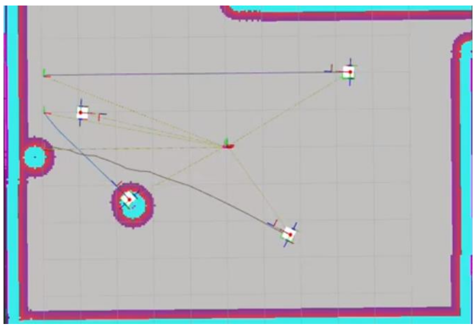
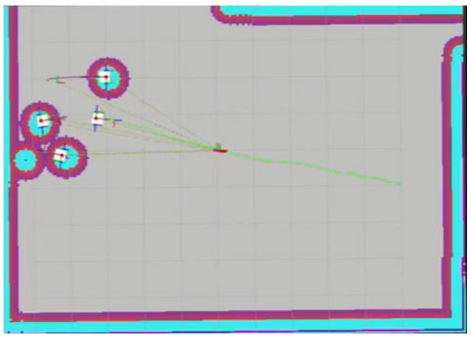
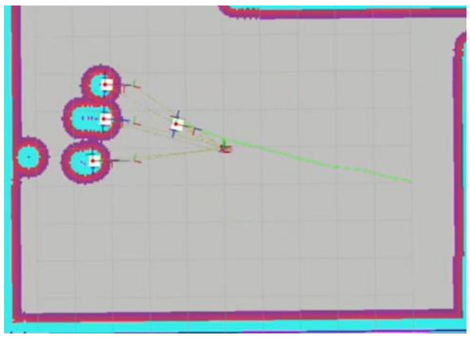

(f)

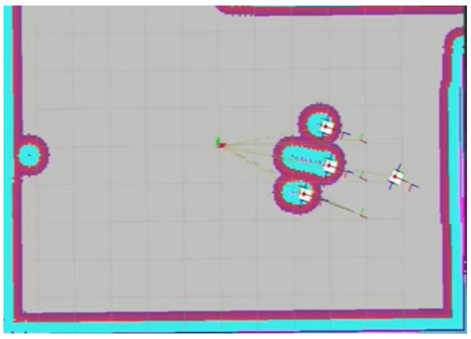
Figure 4.10: Results of Scenario 1 without Formation Control with different time stamped, (a) t = 0s, (b) t = 2s, (c) t = 75s, (d) t = 105s, (e) t = 150s, (f) t = 170s.


(a)


(b)


(c)


(d)


(e)

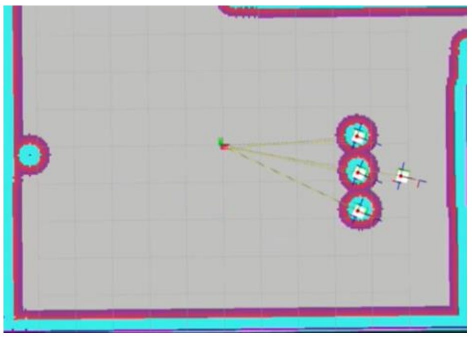

(f)

Figure 4.11: Results of Scenario 1 with Formation Control with different time stamped, (a) t = 0, (b) t = 2s, (c) t = 75s, (d) t = 105s, (e) t = 150s, (f) t = 170s.

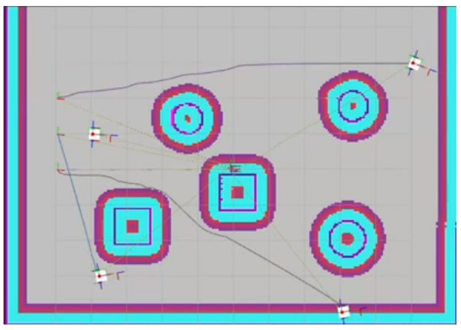
By comparing both Figure 4.10 and Figure 4.11, it become clear that the rendezvous position of the follower robot differ, as indicated based on the path visualized by RVIZ2. This is mainly because without formation control within the algorithm, and consequently, the follower mobile robots are expected to move to the position of the leader robot, while ignoring the presence of the leader. This behaviour may result in a cluster of follower robot around the leader robot, blocking the path of the leader robot after finishing the rendezvous and impeding the task need to be done.

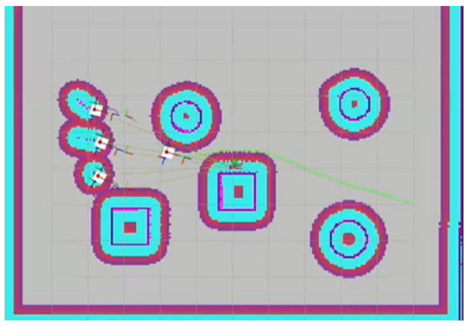
Other than that, in Figure 4.10 (d) and (e), it can be observed that the path of the follower mobile robots overlaps, which could potentially result in collision them. Even though each simulated mobile robot is equipped with a Nav2 local planner that utilizes LIDAR sensors to build a local costmap, it is important to take note that the refresh rate is only 5Hz. This refresh rate may not be sufficient for the mobile robots to dynamically avoid obstacles or even other mobile robots, especially when the robots are required to change direction frequently.

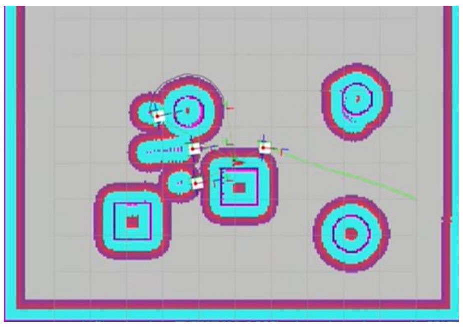
By implementing formation control within the proposed algorithm, this will enable the follower robots to rendezvous at the leader position with an expected formation, which is capable of avoiding clustering around the leader robot and ensure a smoother transition to the next task with minimizing any potential delays. Other than that, implementation of formation control ensures that the follower robots maintain a specific formation while moving to the final goal. It ensures the mobile robots can navigate safely without the risk of colliding with each other.

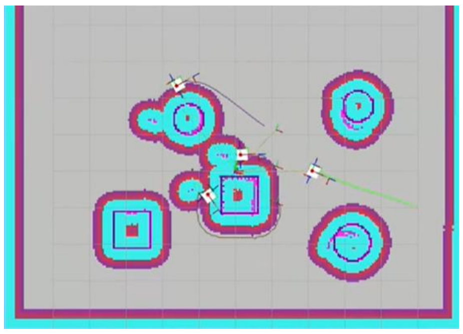
4.3.2 Scenario 2 – Obstacle-Rich Environment

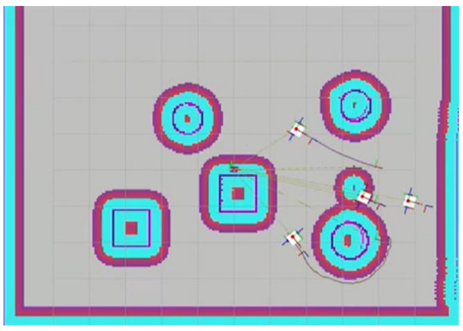
In Scenario 2, an obstacle-rich environment is created to test the ability of the mobile robots to navigate around static obstacles while performing the rendezvous task. With the help of the LIDAR sensors, the mobile robots can detect the obstacles and utilize the pre-built map to path around them. The result of the simulation is shown in Figure 4.12.


(a)


(b)


(c)


(d)


(d)

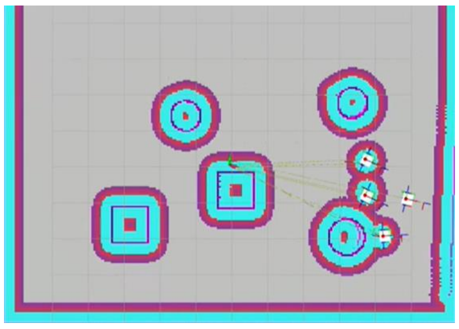

(f)

Figure 4.12: Results of Scenario 2 with Formation Control with different time stamped, (a) t = 0, (b) t = 2s, (c) t = 75s, (d) t = 105s, (e) t = 150s, (f) t = 170s.

In Figure 4.12 (c) and (d), this scenario highlights of the importance of formation control and careful path planning in the obstacle-rich environment. In this scenario, the leader navigates along a narrow pathway between the obstacles. As the follower robots navigate around the obstacle, the follower robot spread out to avoid the obstacle. When their expected goal is within the obstacle, the follower robot will stop moving, and wait after the goal is outside the obstacle, then the follower robot will navigate around the obstacle. Without formation control, the follower mobile robots might collide as shown in the figure below.

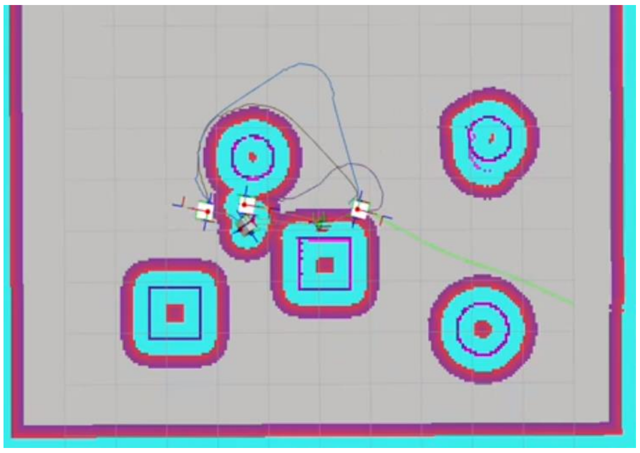


Figure 4.13: Collision among follower mobile robots.

4.4 Summary

The Hierarchical Rendezvous Control Algorithm has been successfully developed in MATLAB and implemented in ROS2. The proposed algorithm was then compared to the state-of-the-art rendezvous algorithm (circumcenter algorithm) in terms of convergence rate and performance, and it outperformed the latter in terms of total distance travelled. Furthermore, it required a similar amount of time to finish the rendezvous task. Additionally, formation control during herding was introduced during the simulation in ROS2.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In summary, all four objectives of the project have been successfully achieved. The Hierarchical Rendezvous Control has been developed using MATLAB to gather robots in a different number of groups and enable to coordinate the multiple robot's movement and action, ensuring they meet at a specific location and time.

Through different scenarios of simulation and experiments, the properties of the algorithm such as convergence rate and connectivity maintenance have been demonstrated. Performance and scalability were compared with the circumcenter algorithm. The proposed algorithm has also been applied in a potential application, which is the multi-group herding of a multi-robot system.

Moreover, the proposed algorithm has been implemented within a multi-robot scenario in ROS2. A formation control strategy has also been added to the proposed algorithm to ensure that mobile robots do not collide with each other while navigating during the rendezvous task.

5.2 Future Recommendation

In future work, real-time experiments could be included to further validate the proposed algorithm, as the current project only involved simulations. Additionally, the proposed algorithm was developed and tested on homogeneous robots. However, in reality, robot teams may consist of heterogeneous robots with different capabilities and characteristics. Therefore, further work could explore the possibilities of experimenting with the effectiveness of the proposed algorithm on heterogeneous robots. Moreover, the application domains of the proposed algorithm could be extended by investigating its applicability in other areas such as search and rescue missions and environmental monitoring. Lastly, integrating machine learning within the proposed algorithm to find optimal parameters could improve its effectiveness.

REFERENCES

Alpern, S. (2011) ‘A New Approach to Gal’s Theory of Search Games on Weakly Eulerian Networks’, *Dynamic Games and Applications*, 1(2), pp. 209–219. Available at: https://doi.org/10.1007/s13235-011-0009-4.

Ando, H. *et al.* (1999) ‘Distributed memoryless point convergence algorithm for mobile robots with limited visibility’, *IEEE Transactions on Robotics and Automation*, 15(5), pp. 818–828. Available at: https://doi.org/10.1109/70.795787.

Clearpath Robotics (2014) *ROS 101: Intro to the Robot Operating System*. Available at: https://doi.org/10.1007/BF02891710.

Cort&eacute;s, J. and Egerstedt, M. (2017) ‘Coordinated Control of Multi-Robot Systems: A Survey’, *SICE Journal of Control, Measurement, and System Integration*, 10(6), pp. 495–503. Available at: https://doi.org/10.9746/jcmsi.10.495.

EPFL educational and research mini mobile robot VERSION 2 (no date) GCtronic. Available at: https://www.gctronic.com/e-puck2.php

Javaid, M. *et al.* (2021) ‘Substantial capabilities of robotics in enhancing industry 4.0 implementation’, *Cognitive Robotics*, 1, pp. 58–75. Available at: https://doi.org/10.1016/j.cogr.2021.06.001.

Jawhar, I. *et al.* (2018) ‘Networking of Multi-Robot Systems: Architectures and Requirements’, *Journal of Sensor and Actuator Networks*, 7(4), p. 52. Available at: https://doi.org/10.3390/jsan7040052.

Kan, Z. *et al.* (2017) ‘Decentralized Rendezvous of Nonholonomic Robots With Sensing and Connectivity Constraints’, *Journal of Dynamic Systems, Measurement, and Control*, 139(2). Available at: https://doi.org/10.1115/1.4034745.

Ke Xu (2010) ‘Integrating Centralized and Decentralized Approaches For Multi-Robot Coordination’, *Rutgers The State University of New Jersey - New Brunswick ProQuest Dissertations Publishing* [Preprint].

Khepera IV New - K-Team Corporation (2021) K. Available at: http://www.kteam.com/khepera

Krishnanand, K.N. and Ghose, D. (2008) ‘Theoretical foundations for rendezvous of glowworm-inspired agent swarms at multiple locations’, *Robotics and Autonomous Systems*, 56(7), pp. 549–569. Available at: https://doi.org/10.1016/j.robot.2007.11.003.

Lin, J., Morse, A.S. and Anderson, B.D.O. (2004) ‘The multi-agent rendezvous problem - the asynchronous case’, in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*. IEEE, pp. 1926-1931 Vol.2. Available at: https://doi.org/10.1109/CDC.2004.1430329.

Luo, S. *et al.* (2019) ‘Multi-robot rendezvous based on bearing-aided hierarchical tracking of network topology’, *Ad Hoc Networks*, 86, pp. 131–143. Available at: https://doi.org/10.1016/j.adhoc.2018.11.004.

Macenski, S. *et al.* (2020) ‘The Marathon 2: A Navigation System’, in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2718–2725. Available at: https://doi.org/10.1109/IROS45743.2020.9341207.

Macenski, S. and Jambrecic, I. (2021) ‘SLAM Toolbox: SLAM for the dynamic world’, *Journal of Open Source Software*, 6(61), p. 2783. Available at: https://doi.org/10.21105/joss.02783.

Maruyama, Y., Kato, S. and Azumi, T. (2016) ‘Exploring the performance of ROS2’, in *Proceedings of the 13th International Conference on Embedded Software*. New York, NY, USA: ACM, pp. 1–10. Available at: https://doi.org/10.1145/2968478.2968502.

Ozsoyeller, D., Özkasap, Ö. and Aloqaily, M. (2022) ‘m-RENDEZVOUS: Multi-Agent Asynchronous Rendezvous Search Technique’, *Future Generation Computer Systems*, 126, pp. 185–195. Available at: https://doi.org/10.1016/j.future.2021.08.007.

Ozsoyeller, D. and Tokekar, P. (2022) ‘Multi-Robot Symmetric Rendezvous Search on the Line’, *IEEE Robotics and Automation Letters*, 7(1), pp. 334–341. Available at: https://doi.org/10.1109/LRA.2021.3126350.

Parasuraman, R. *et al.* (2020) ‘Multipoint Rendezvous in Multirobot Systems’, *IEEE Transactions on Cybernetics*, 50(1), pp. 310–323. Available at: https://doi.org/10.1109/TCYB.2018.2868870.

Pelc, A. (2012) ‘Deterministic rendezvous in networks: A comprehensive survey’, *Networks*, 59(3), pp. 331–347. Available at: https://doi.org/10.1002/net.21453.

Potop-Butucaru, M., Raynal, M. and Tixeuil, S. (2011) ‘Distributed Computing with Mobile Robots: An Introductory Survey’, in *2011 14th International Conference on Network-Based Information Systems*. IEEE, pp. 318–324. Available at: https://doi.org/10.1109/NBiS.2011.55.

Quigley, M. *et al.* (2009) *ROS: an open-source Robot Operating System*. Available at: http://stair.stanford.edu.

Reke, M. *et al.* (2020) ‘A Self-Driving Car Architecture in ROS2’, in *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE, pp. 1–6. Available at: https://doi.org/10.1109/SAUPEC/RobMech/PRASA48453.2020.9041020.

Verma, J.K. and Ranga, V. (2021) ‘Multi-Robot Coordination Analysis, Taxonomy, Challenges and Future Scope’, *Journal of Intelligent & Robotic Systems*, 102(1), p. 10. Available at: https://doi.org/10.1007/s10846-021-01378-2.

Zheng, R. and Sun, D. (2014) ‘Multirobot rendezvous with bearing-only or range-only measurements’, *Robotics and Biomimetics*, 1(1), p. 4. Available at: https://doi.org/10.1186/s40638-014-0004-5.

APPENDICES

APPENDIX A: Link for Code for MATLAB and ROS2

Link - https://github.com/jinsiang120/FYP/tree/main