university of groningen / faculty of science and engineering

Investigation of Frontier and Rapidly-Exploring Random Tree Based Robot Exploration Algorithms and Implementation of Multi Robot Exploration

Industrial Engineering & Management Bachelor Integration Project
University of Groningen
Faculty of Science and Engineering
June 16, 2023

Student: Rick Drenth, s4490444, r.drenth.4@student.rug.nl
1st Supervisor: Prof. Dr. Ir. Ming Cao, m.cao@rug.nl
2nd Supervisor: Dr. Ir. Gerald Jonker. g.h.jonker@rug.nl
Daily Supervisor: MSc. Bangguo Yu, b.yu@rug.nl

Abstract

This research focuses on the exploration of unknown area by a Jackal UGV robot using two different methods: frontier exploration and rapidly exploring random tree (RRT) exploration. Robots play a crucial role in exploring unknown environments. Therefore, it is necessary to understand and optimize these methods.
The experiments were performed in three maps of varying sizes using different levels of complexity. The simulations utilize a 2D LiDAR. The area coverage, time, exploration rate, and the robustness of both methods were analysed and compared. Despite localization inaccuracies and path planning issues, both methods achieved high completion rates. The frontier exploration method generally outperformed the RRT exploration in terms of speed, showing that Frontier exploration utilizes a more efficient exploration policy.
Improvements such as incorporating an exploration route and considering unreachable but obtainable areas are suggested. Furthermore, the inclusion of a multi-robot system for improved efficiency and how to implement a multi-robot system are explained. Lastly, the limitations and improvements of the 2D simulation in representing real life exploration are also discussed.

I. Introduction

Robots play a crucial role in exploring unknown environments, offering the potential to extend human exploration to unreachable and hazardous places and help in surveillance. Robot exploration also finds applications in household and logistics tasks. By designing robots capable of autonomous exploration, without human intervention, safety can be ensured and workload for humans can be reduced [1]. For example, Kuka Systems is a company that produces smart mobility systems. Another robots model from Kuka is a mobile robot with a robotic arm that can autonomously navigate the shop-floor to a destination before performing maintenance [2]. Another example from Kuka is a large robot platform that is used to transport large aeroplane parts during construction [3].
The progress in robot exploration algorithms since the 1990s has been significant. Many of these methods utilize frontier and/or random tree algorithms [4], [5], [6]. Therefore, understanding the limitations of these systems is crucial.
Frontier methods involve searching for boundaries of unknown areas [5], enabling the robot to move towards unexplored regions and reveal new boundaries. This process is repeated until no new boundaries are detected. Another commonly used algorithm is the random tree or rapidly-exploring random tree algorithm (RRT), which generates random samples in unknown areas to establish new exploration steps [7]. Optimization through the use of exploration policies to determine optimal paths is frequently employed. However, focusing on a single problem or situation does not guarantee success in alternative scenarios.

Exploration methods can encounter difficulties in complex environments, particularly when algorithms encounter obstacles [8]. A promising method to enhance the effectiveness of exploration robots in terms of area coverage and exploration time is the investigation of a multi-robot approach. Compared to a single robot, multiple robots can explore complex areas more rapidly. When using a multi-robot system, it is important to consider the optimal task distribution among the robots [6]. Hence, this thesis centres around two key objectives: Firstly, comparing the existing frontier and random tree algorithms. Secondly, exploring the feasibility of extending the current single robot system to a multi-robot system, along with the necessary task adjustments in the exploration policy. Unfortunately, due to time constraints, the implementation of task allocation policies could not be implemented in this thesis. Therefore, only possible improvements will be discussed and left open for validation in future research. Thus, only the single-robot algorithms will be simulated. The exploration algorithms were evaluated on area coverage, time efficiency, consistency, and robustness. The robots in this study utilize simultaneous localization and mapping (SLAM) to continuously update the map during the exploration process. The algorithms and analysis are developed using the Robot Operating System (ROS) and are applied in simulation using a Clearpath Robotics Jackal UGV robot, which is available to the university of Groningen for further validation in real-world scenarios. The Jackal UGV has been equipped with a 2D 270° range. The algorithms and robot code in ROS is available as open-source on GitHub. The code has been made suitable for a multi-robot system and updated to work in ROS Noetic, which is currently the newest version of ROS 1 and will be the last ROS 1 version available.

This thesis will first discuss relevant work and required knowledge, including the robot framework and characteristics. Subsequently, the implementation and simulation of methods will be explained before the results will be presented and discussed. Lastly, the final section will sum up the observation made and present promising trails for further research.

II. Related Works

More recent research has explored the integration of machine learning algorithms to enhance problem-solving efficiency and robustness [9]. Machine learning techniques require substantial time and data for robot training. Research into machine learning has shown that it can be very effective in complex systems, such as multi-robot interactions [10]. The downsides are that due to the behaviour following from a data set, it is hard to upscale a system to the real world. On top of that, the robots can find it hard to react to a dynamic environment. This results in abrupt behaviour.

Recent research has proposed hybrid-stochastic methods, combining frontier and RRT algorithms, to achieve high coverage percentages (around 95-100% in most cases) in semi-complex environments [11], [12]. However, these methods have not been extensively tested in more complex situations, such as realistic room setups or multi-room problems. In complex areas with obstacles, irregularities, or multiple rooms, a coordinated multi-robot approach can be more efficient [7]. While hybrid stochastic multi-robot systems have been tested on single-room complex systems ([11], [12]), further research can replicate and expand upon these systems to address more complex and specific problems.

Further research into multi-robot control in autonomous exploration would benefit the academic world. Communication between robots is crucial to prevent collisions and minimize redundant exploration of the same areas. In path planning, consideration must be given to communication among robots and their positions [4], [13]. Therefore, it is important to compare existing techniques with improved algorithms.

III. Proposed Methods

A. Robot Framework

The general process of autonomous robot exploration is visualized by a diagram (Figure 1. The steps shown are based on systems described in [1], [4], [6], [7], [14].

Robots can collect information about the area by making use of a distance sensor. This data then needs to be converted and added to a global map. The robots will have to place themselves in their own global map for orientation and path planning

purposes. The robot orientation is done using SLAM [4], [15].
After localization, the frontiers can be detected using one of the existing a frontier detection algorithms. If there are no frontiers detected, the exploration mission is seen as completed. While, if there are unexplored frontiers left, the exploration will continue. Different methods can use a slightly different definition of a frontier. Therefore, the specific definition will be discussed in their respective sections below. The exploration policy is the driver for most optimization after the detection of frontiers. Exploration policies usually consist of a cost function or a revenue function that decides which frontier is the most optimal movement goal. Exploration policies are imperfect and can cause backtracking or extra movement, hampering the exploration. Additionally, the robot could be stranded as a result of inadequate communication between the path planning and the exploration policy.
The path planning computes the actions which the robot has to perform to reach the target area. However, if the exploration policy does not account for future goals and movement, the path planning can send the robot towards an area where it cannot turn around.
Furthermore, the path planning requires constant updates about the environment. If the data is not gathered frequently enough, the path planning is unable to account for unforeseen obstacles that appear during movement. Obstacle collision prevention and general robot movement is set to find the shortest direct route to the movement goal. The path planning for the robot is done by the move base packages from the navigation stack. The navigation stack is included in ROS distributions.

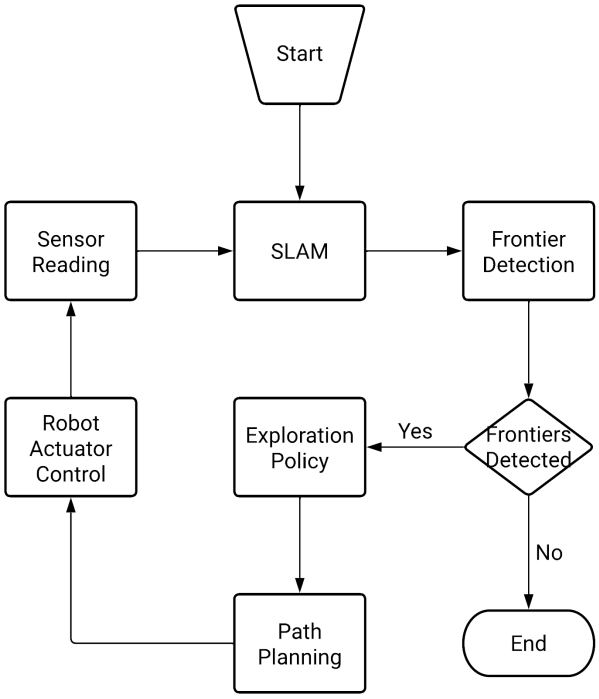

Fig. 1: Diagram showing the different steps in robot exploration

B. LiDAR Sensor

In practice, there are many occurrences where the accessible area will not be completely uncovered. First of all, no matter the technology used, the area will be represented by a grid pattern of some form. Because the real world is continuous and not discrete, the conversion from the real world to a representation will cause a loss of data. Secondly, in real-life situations, sensor data is imperfect. As mentioned by Yamauchi, sound pulses from the sonar system can fade after hitting a flat surface [5]. Therefore, data might be misrepresented or omitted in the representation of the environment. Sensors are also subjected to noise disrupting the readings.
Often, to limit the information disruption, a LiDAR is used. Instead of sound waves, a LiDAR uses electromagnetic pulses to measure distance [16]. By measuring the time it takes for the pulses to return and dividing it by the light speed, the distance can be calculated with a high accuracy. Because light travels faster than sound, the quality of measurements will increase compared to sonar readings. However, light reflection can be disturbed by weather conditions and reflections, restricting the use of LiDAR in outside situations. As a result, exploration algorithms are usually developed for inside situations. Thus, the performance of exploration algorithms in outdoor conditions can be limited.

C. SLAM

SLAM algorithms enable a robot to navigate and explore autonomously without relying on pre-existing knowledge about its environment nor on external tools such as GPS. SLAM algorithms iteratively update both the robot positions and the map based on sensor measurements.
SLAM requires knowledge about the robot position and odometry. Position information is in this case provided by the LiDAR sensor. In simulation, the odometry is given by the simulation environment. In real life, odometry data can be provided by an odometer or calculated from distance and time data. By combining the sensor measurements with estimation techniques, the position of the robot can be estimated and a map of the environment can be created.
In this thesis, gmapping is used for the SLAM algorithm [17]. gmapping builds the map of the environment by interpreting sensor data. The map is represented as a 2D grid of occupied points. Each cell holds a probability of occupancy. Using the sensor data, the weights of the probability can be adjusted, resulting in a more accurate representation [18].
When using the gmapping included in the ROS Noetic distribution, the package provides localization for the robot as well, based on sensor data.
The gmapping is supported by an Extended Kalman Filter (EKF). EKF predicts the robot state by using a linearized version of the equations of motion of the robot [19]. By using the speed and position of the last known point in time in a discrete model, the current position can be estimated. After the prediction, the robot updates the current position by using the measured data of the current state. By iteratively performing this process, the state estimate can continuously be given to the robot system. It must be noted that due to sensor noise and linearization of a dynamical model, the quality of estimations can be poor. In order to improve the quality, a higher number of reference points in the environment should be given. EKF fuses its own estimation with that of the gmapping, creating a more accurate localization.

D. Frontier Exploration

The basis of most exploration and mapping algorithms is built upon the foundations laid down by Brian Yamauchi in 1997 [5]. Frontiers are the boundaries of an unexplored space. When a robot moves towards unexplored space, the newly explored area can be mapped and the boundaries are shifted. Theoretically, by moving towards unexplored boundaries every iteration, the entirety of an accessible region can be explored.
As mentioned in the SLAM and Sensor section, the constructed map of an area can contain inaccuracies. Therefore, there will be discrepancies between simulation results and reality.
Based on the most recently obtained sensor data, points on the grid map can be classified as explored space, occupied space or unknown area. Using edge detection algorithms, the boundaries of the explored area are generated. An exploration boundary is present in any open area neighbouring unknown space. A frontier is defined as the connection of explored space and unknown area that allow enough space for the robot to pass through [5]. As a result, natural boundaries of accessible area are formed.
After updating the map, the robot will attempt to move towards the nearest frontier, regardless of resulting backtracking motion. The path planning algorithm uses the shortest possible path it can find without interacting with the occupied grid spaces to avoid colliding with obstacles. In the paper by Yamauchi, it is described that the robot only scans the area after reaching the newest frontier. Obtaining data while driving can cause the robot to change course towards a closer frontier when discovered.
The algorithm by Yamauchi has been implemented by Bovbel and is available on the ROS website [20]. The algorithm has been implemented using ROS and can be used in simulation and on real robots. However, for the sake of facilitation, another package is used, the m-explore package [21]. The m-explore package, behaves similarly to the package by Bovbel. In addition, this method improves the algorithm by gathering data while moving, instead of solely gathering data at a frontier. The m-explore package is updated for ROS Noetic, eliminating most incompatibility issues. This open-source package can therefore be used to test and compare the frontier algorithm to newer algorithms.

E. Rapidly-Exploring Random Tree Exploration

In search of a more time efficient exploration method, Umari and Mukhopadhyay developed the rapidly-exploring random tree algorithm, which combines the features of the frontier exploration algorithm and a random tree based algorithm [7]. The rapid random tree algorithm was originally developed by LaValle. It differs from the standard random tree algorithm by always taking the nearest node to the randomly selected point to create an edge [22], thereby vastly improving the speed of the algorithm. RRT is quick to find a path towards unexplored space. However, the path it finds is often not optimal due to re-exploring of space and non-optimal path planning.
Umari presents a method that uses the RRT algorithm to find frontier points. The algorithm first selects a random point. If the point can be found in the unexplored region of the map, the point is marked as a frontier. Every iteration, the tree is reset and restarted from the current robot position. After finding a new frontier, the robot will move towards that frontier similarly to the standard frontier exploration algorithm. This algorithm is called Local Frontier Detector. Umari combines the Local Frontier Detector with a Global Frontier Detector, which is not reset. The combination of both ensures that far points of the map can be reached before the tree is reset.
To reduce the number of data points, a filtering module is used to discard old frontier points and frontiers points that are close together by taking the centre point of a group of frontiers. The remaining points are used for the path planning. Umari created a weighted system to evaluate the frontier points based on the distance to the new point in a straight line and the expected information gain. The revenue is defined as:

$$R(x_{fp}) = \lambda h(x_{fp}, x_r) I(x_{fp}) - N(x_{fp}), \quad (1)$$

$$h(x_{fp}, x_r) = \begin{cases} 1, & if \ \ ||x_r - x_{fp}|| > h_{rad} \\ h_{gain}, & h_{gain} > 1 \end{cases} \quad (2)$$

Where λ is the weight given to the information gain, I the information gain, N the navigation cost, x_r the current location, x_{fp} the frontier position, h_{rad} an arbitrary radius from the robot and h_{gain} an arbitrary gain.

Formula 2 proposes that for a distance to the frontier larger than an arbitrary value, the gain will be minimized. Therefore, the algorithm creates bias towards shorter distances from the robot. As a result, the robot explores structurally, which reduces backtracking.
The information gain is a circle around the new point. The radius of the circle should be equal to that of the sensor data. This assumes that all robots used gain information in a circular shape around the robot. However, in reality, the LiDAR used in this system only has a 270° view. A weight is given to the information gain to ensure that the information gain and navigation cost have the same order of magnitude. Umari has implemented the algorithm using ROS. The ROS package is available on [23]. The package can used for simulation and real-world robots. Umari has also extended the package to include multi-robot control.

F. Multi-Robot Exploration

Multi-robot exploration makes use of coordinated exploration of an environment by a team of multiple robots. Instead of relying on a single robot to explore an area, multi-robot systems distribute the exploration task among multiple robots, allowing for more efficient and effective coverage. With multiple robots working simultaneously, the exploration process can be completed more quickly compared to a single robot. Each robot can explore a different region of the environment. Thereby, reducing the overall exploration time. By making use of multiple robots, it is possible to achieve better coverage of the environment. The robots can explore different areas simultaneously, increasing the exploration rate and coverage of a region.
Multi-robot systems can share information and observations with each other, improving the overall knowledge of the environment. The robots can collectively build a more accurate map or model of the explored area by sharing the data in a combined map. This map can then be used for further communication and planning of the multi-robot exploration. However, combining multiple maps is subjected to inconsistencies. For perfect map merging, the localization of the robots would have to be perfect. In reality, localization is never perfect [19], [18]. Therefore, merging maps using multiple robots can create an inaccurate map that

mistakenly removes or adds obstacles on the global map. Since the exploration policy relies on the overall map, inaccurate map merging can cause many issues in exploration.
Multi-robot systems are more robust compared to single-robot systems [24]. If one robot encounters a problem or malfunctions, other robots can continue the exploration task, minimizing the impact of individual robot failures. Furthermore, whenever an individual robot makes a mistake in its measurements of the surroundings, another robot can account for its mistakes and correct the map. However, the correcting system relies on overlap of sensor data and backtracking, which is not desired for an optimal exploration rate.
Multi-robot exploration requires more complex task allocation and coordination mechanisms compared to single-robot exploration. In the package used for this thesis, the multi-robot exploration relies on the RRT exploration algorithm. The RRT algorithm uses one lead robot to calculate possible frontiers [23]. It then calculates the most valuable frontier for each robot, selecting the most valuable of all and then assigning it to the corresponding robot. When assigning exploration goals, the RRT system does not account for overlap between the expected data gathered from new goals. As a result, overlap of data and backtracking situations are created, hampering the potential of the system.
The main change would be to start the random tree from all robots, alternating the robots or having a global position where the random tree starts branching. Without such a system, the branching favours the robot on which the algorithm is based. Secondly, the expected data has to be taken into account when calculating the optimal frontier. Lastly, the robots can focus on dividing the area in segments. By assigning each robot a segment, the overlap is minimized, and the random tree will find the edges of its boundary quicker. As a result, the overall performance will improve. However, as mentioned before, this reduces the chances for correction of localization. Furthermore, planning collision-free paths for multiple robots in real-time can be challenging, especially in complex and dynamic environments. Without proper localization and continuous communication with a hub or other robots directly, collision is imminent.
In the literature, it is common to consider a group of three robots working together. In this specific scenario, it will be assumed that the robots can communicate their positions with each other. It is important to consider communication in the path planning of the robots to prevent collisions and minimize the area that is explored multiple times [4], [13].
Unfortunately, using a multi-robot system requires a lot of computational power. As a result, the uses of multi-robot systems in reality are minimal, as it is difficult to provide enough capacity in exploration of area unknown to the users. Using systems on the shop floor, similar to the Kuka robots, would allow the user to install computing power within communication range.

G. Robot Operating System

1) ROS Environment: The Robot Operating System is open-source software that can be used as a tool for developing and controlling robots. ROS is already widely used by robot developers and is becoming increasingly popular [25]. The environment provides tools in the form of packages, written by the creators, Open Robotics, and by the public. The newer versions of ROS are supported on Linux, Windows, and macOS. However, most of the documentation is written for Linux Ubuntu. Since 2020 ROS has been remade into ROS 2 with ROS 1 remaining as the older version. ROS 2 is still in development and will be fundamentally different from ROS 1. As a result, existing libraries in ROS 1 can stop functioning. However, many of these libraries are very useful. Because it is non-trivial to convert these packages to ROS 2 and ROS 2 is still in development, ROS 1 will remain relevant. The most recent distribution of ROS 1 is ROS Noetic. ROS Noetic will be supported until 2025 [26]. At the time of writing, the previous distribution, ROS Melodic, is near its end of life. Meaning that, after May 2023 the system will not be supported any more. Since the aforementioned algorithms are only compatible with older versions of ROS (Melodic for Frontier exploration and Kinetic for RRT exploration) and both packages have not been updated since 2020, there is no guarantee they will work in Noetic. Therefore, part of this thesis is updating the packages whenever problems are encountered.
ROS makes use of a system of nodes and topics.

A node is an executable file in a package [27]. A topic is a communication service which nodes can use to exchange information by subscribing and publishing to nodes [27]. It is important to understand the ROS communication system to be able to debug any issues that may arise. The nodes and topics can also be visualized automatically by ROS using the rqt package. The visualization of nodes and topics is similar to a flow chart, making it easier to discover unconnected nodes and mismatching issues.
ROS is compatible with the Gazebo simulation environment. Gazebo allows developers to create their own robots and worlds. Robots are represented using URDF files which describe all the links of a robot. Gazebo converts the given URDF files to SDF files. The SDF files do not only describe the robot, they also describe the simulation world, dynamic movement and include physical properties [28]. Gazebo can subscribe to ROS topics and use the information about the robot state to simulate the movement.
For this project, a number of Gazebo worlds have been created for testing various properties of the exploration algorithms. The characteristics of these worlds will be explained in the methodology.
In combination with Gazebo, the rviz package can be used for the representation of the local and global maps [29]. This package enables basic functionality for use in mapping, such as directional goal instructions and display of maps. However, the frontier and RRT packages provide increased functionality with the rviz package. The package can show both local and global maps and frontiers. As well as the path the robot is currently traversing. By making use of the navigation stack, rviz can output the number of pixels in the map that are uncovered. Using this feature, the percentage of uncovered space can be calculated. It is also possible to keep track of the exploration time using rviz. The explored area and time are essential in the comparison of multiple exploration systems.

2) ROS Implementation: The algorithms are tested on the Jackal UGV (Figure 2. The Jackal UGV can be simulated using a ROS meta package, provided by Bangguo Yu [30]. A meta package contains a group of packages bundled into one large, interdependent package. Next to the use of the move base package to move the robot, the robot can be manually controlled using teleop input such as joysticks. The launch files will open the Gazebo environment with the world specified in the main launch file and the specified robot.
The meta package also contains demo launch files that launch all the necessary nodes to control the robot and represent the output in rviz, including localization, mapping, and movement goals.
In the main launch file, the world for testing can be specified by replacing the "name.world" with the desired gazebo world. For testing autonomous exploration algorithms, the joystick argument can be set to false. However, leaving it on true does not interfere with the exploration and can be useful to overwrite commands when testing real robots. The main changes to convert the algorithm for single to multi-robot use are the namespaces. All nodes and links need a robot namespace to differentiate between information from different robots. By setting the namespaces as arguments for the include file, different robots can be called using different names. For each robot a different move base node is also initialized. The move base package computes the path the robot has to take to reach a goal and sends the velocity commands. The move base package makes use of a global and local planner. The global planner calculates a path to the target location from the current position of the robot. To be able to calculate a path, the package requires the grid based map of the environment to account for obstacles. The move base package uses an A* algorithm to calculate the shortest path. The A* algorithm uses an approximation function to calculate the costs of movement of each grid point to the goal in search of the most promising path.
Combined with the global planner, a local planner is included to deal with dynamic obstacles. It uses a subset of the map, called a local map, for trajectory calculations. The simulation in this thesis do not make use of dynamic obstacles.
To deal with navigation failure, the move base package also includes recovery behaviour algorithms. The first step in performing recovery behaviour is to reset the costmap used for calculations, resulting in the most up-to-date information map. Next the package tries to recalculate the path and if necessary tries to rotate the robot.
Further down in the main launch file, the map-

merge package is called upon. The map-merge package merges the maps for use in multi-robot exploration.
Below the map merge package node, the transformation nodes can be found. The transformation nodes relate the position of the robot to the reference frame of the world. It is important to remember that when changing the initial positions in the robot launch nodes, the same initial conditions need to be set in the static tf transform publisher. The static transform publisher is used to communicate the initial positions of the robot with the world. Lastly, rviz is launched using robot 1 as a reference for the display of maps.
The jackal spawner, called using the main launch file, launches a number of different nodes. First of all, the description of the robot is launched. Then the control and teleop nodes are launched. The control is used whenever the robot uses autonomous exploration, whereas the teleop ensures that a controller or keyboard can be used for control. Lastly, the robot model is spawned using the URDF files. URDF files include all information about robot links and its representation in a simulation environment.
In the description launch file, the transform publisher node for continuous transform is initialized. As opposed to the static transform node, this node communicates the position of the robot while not stationary. In the URDF files, namespaces have been added in various places.
In the control launch file, the name space has been included as well. The file launches the EKF localization, included in the robot localization package. In the corresponding yaml file, one line has been added that was causing naming issues. Inside the robot localization package, a default name for the child frame can be found. The namespace does not automatically overwrite this frame. Therefore, the namespace has to be specified for the name base frame id. Furthermore, substitution values are set to true. This is needed to allow the namespace as an argument in the yaml file.
After the robots are set up, the gmapping demo can be activated in a different terminal. The gmapping demo launches the gmapping package. In the gmapping package, the LiDAR range can be set in the maxUrange parameter.
Next to initializing the gmapping, the exploration algorithm also has to be activated in a new terminal. In the RRT launch files, the initial costmap can be changed. There are many more files included in all the packages. However, this explanation was kept to those files that have been altered for implementation of the multi-robot system, essential for explanation of the system or files that had parameters changed. Changed parameters will be discussed in the methodology.

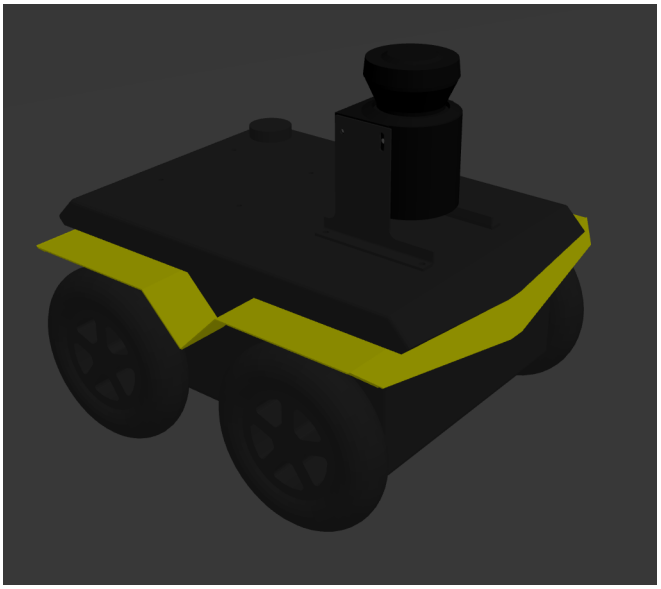

Fig. 2: Image of the Jackal UGV robot in a simulated environment.

IV. Experiments

A. Simulation Setup

The Jackal UGV in the simulations makes use of a 2D LiDAR with a 270° range of 8 m. For the experiments, the LiDAR range is increased from 8 m to 10 m. This was done to lower the chance of localization not working properly. The initial map size for both global and local maps was increased to a 20×20 m area, matching the size of the largest used map. This ensures that the robot can always discover new frontiers. The robot has the same initial position in all the maps. The initial position is set to the origin at (0,0), which is also the centre of the maps.
Three custom worlds were made for testing various aspects of the robot. Map 1 is the largest map (Figure 3. It has a size of 20×20 m. The square obstacles on the map are evenly spaced out. A lower density of obstacles was attempted beforehand. This proved to give many localization

problems. Therefore, the lower density maps were rejected in favour of higher density maps. A large room prevents the LiDAR from scanning the room instantly after turning and makes the room acceptable for testing obstacle interaction.

Map 2 is the smallest map and has a size of 10×10 m (Figure 4). The map features different types of obstacles. On all sides, small corridors and corners with small coves can be found. These obstacles force the robot to move around the obstacles, at times moving into unexplored space that is directly continuing in previously uncovered area. The meandering of the map creates a situation where the robot has to choose a target position that is possibly not the shortest total path. The walls create a situation where the robot has to choose between frontiers and makes detection of frontiers harder. Between the cylindrical objects, a situation exists where the LiDAR can see into the gaps, creating a new unknown area. However, the robot cannot uncover this area immediately due to the small intraversable gap. Because the area is a 10×10 m area, the LiDAR is able to scan the entire length of the area if there are no obstacles interrupting its view. The last map is a 15×15 m map (Figure 5). This map is larger than the 10×10 m map, forcing movement from one side of the map to another. The map features simple wall based obstacles. This is done to create a large chance of success for the exploration algorithms and thus focussing the test on the exploration rate on a larger map. Inspiration for the maps was taken from [11], [12].

To allow for proper analysis, the methods should each have a valid run on every map. A run is valid if the robot does not get stuck and can complete its entire programme. A programme is seen as complete when no new area is being discovered by the robot. A possible source of error is the inaccuracy of size of the uncovered area. This is the result of inaccuracies in the mapping process. Therefore, maps that have visible large inaccuracies have to be excluded and cannot count as a valid run.

Due to the randomness in the RRT algorithm and the inaccuracies of localization, the run closest to 100% coverage of five runs will be taken for comparison.

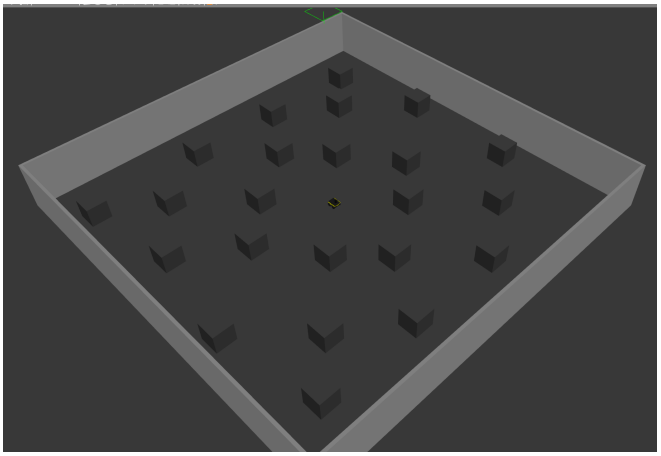

Fig. 3: Image of Map 1 in the Gazebo environment.

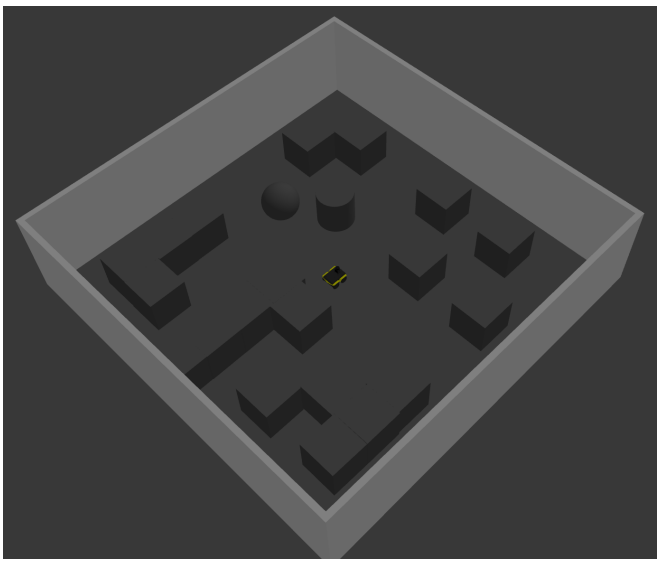

Fig. 4: Image of Map 2 in the Gazebo environment.

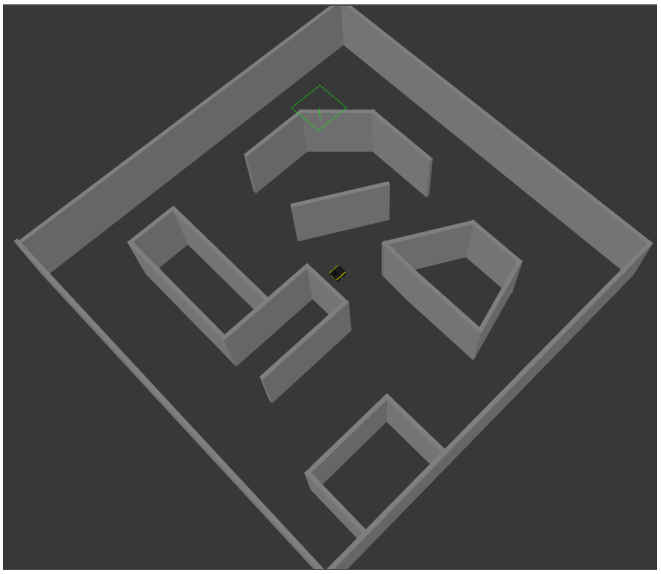

Fig. 5: Image of Map 3 in the Gazebo environment.

B. Results and Analysis

The area coverage and time is recorded and can be compared by an analysis of area coverage over time evolution and the total time and coverage of the map. The percentage of area uncovered can be calculated using the map size. To find the accurate map size, the map first has to be fully uncovered by controlling the robot manually using rviz. Even when manually controlling the robot, the localization is not perfect. Therefore, the maximum map size is not completely accurate. To show the robustness of the algorithms, the number of failed attempts before a successful run will also be noted. Lastly, the simulation of the robot uses a safety factor on the robot when calculating its possible path. As a result, some area can be seen as unreachable. However, this is only a problem in small areas due to the small safety factor. Therefore, it is not expected to raise any issues.
In table I, the results from the most successful run per method for every map can be found. The table shows the attempts needed before the presented run, the uncovered area and corresponding completion percentage, the average exploration rate and the improvement of the average exploration rate of the frontier method compared to RRT.
It can be seen that all methods are close to completion. Although, in three out of six cases, the completion is higher than 100%. This is due to the quality of the localization method, the simulation makes small non-rectifiable mistakes. It is non-rectifiable because the robot can not reach this area afterwards to correct its mistake and overwrite the data. Whenever there is a loss of localization, it can happen that the LiDAR scans the distance, but mistakes the location such that it does not recognize the walls. This effect can be observed in figures 12, 14, 15, 17. However, the effect does not occur in all situations. For example, in figure 16, the frontier algorithm never loses its positions in any of its runs. In figures 6-11, the explored area over simulation time graphs of the best runs for both methods on all maps can be found. In figure 13 the best run of the RRT algorithm on Map 1 is shown. On this map, the best run did not lose its position, although the effect occurred in other runs, which were viewed as lesser runs. Unfortunately, the issue occurs on almost every run for every map to varying degrees. Another possible cause is the obstacles not lining up correctly and allowing gaps in between the obstacles. However, since the issue mostly occurs through the walls, which signify the exploration boundaries, this is not likely to be the case. The uncertainty results in both an uncertain maximum and uncertain covered area. However, it was observed that the loss of localization usually happens towards the end of a run. Therefore, it is more useful to look at the exploration rate.
Furthermore, it was observed that many runs had to be aborted as a result of the robot getting stuck. There are three situations where the robot is getting stuck. First of all, there might be a frontier set close to a wall or other edge. Because the robot cannot drive in reverse, the robot cannot turn and is stuck in place. Secondly, the robot has a safety factor. This safety factor is used in calculations to make the robot bigger than its actual size. However, the robot often drives too close to a wall or obstacle until it is stuck. This problem can possibly be solved by increasing the safety factor. However, this also increases the minimum gap size needed to allow the robot to pass through. Lastly, the robot is getting stuck in situations where it is unclear why it is stuck. The robot indicates that it is stuck and tries to correct itself by turning. In most cases, the robot can continue its journey after some time. However, in some cases, the run had to be aborted.
As can be seen in table I, for the two smaller maps, the exploration rates do not differ much. However, the frontier exploration has a slight edge of around 10% in both maps, including the area outside the boundaries. Where, RRT is supposed to help in finding tight corners, this is not visible in the results. The contrary seems to be true, from observations it seems that the robot does not get stuck as often using the frontier method. This is possibly the result of the slight differentiation in the definition of a frontier. In general, the frontier method also causes less backtracking to previous locations compared to the RRT exploration. Due to the simple exploration policy of the frontier algorithm, it drives towards the edge of explored area. After moving, the robot will move towards the next closest frontier. This causes a fairly organized exploration, branching out from the origin. Although backtracking still happens, it is less common. The RRT algorithm has a more complex exploration policy. Instead of the closest point, the algorithm focusses on the most valuable

point. Often this point is located behind the already explored area. In this policy, the weight of distance can be altered. By changing the weight, the amount of backtracking can be reduced.
On the large map, the frontier algorithm has a lower average exploration rate than the RRT algorithm. However, as can be seen in figure 2, the frontier algorithm has an oddity where it was stuck for a long time, exploring little area. This behaviour is not expected for the frontier algorithm, since it does not branch out like the RRT method, but moves towards a new area immediately. If the time between 23 and 45 seconds were to be excluded, the new average exploration time becomes, 35577 Pixels/Sec, resulting in a rate improvement of 41% instead of the previous 21% decrease. This is the effect of the robot being able to traverse the large open space more easily than using the RRT method. The RRT method was struggling to find the points on the far side of the map.
Moreover, the exploration rate graphs do not account for the initialization period of the RRT exploration. Meaning that, after the algorithm has started, the random tree has to branch out first before finding any frontiers. As seen in the figures, the graph does not plateau at zero seconds. Indicating that the calculation only starts after first movement. This delay results in a skewed vision of the total time it takes. Thus, further favouring the frontier algorithm.
The current SLAM package is causing much trouble while gathering information, contaminating the data and creating large inaccuracies. The loss of localization of the package can be decreased by increasing the LiDAR range and increasing the number of obstacles. Using a larger number of obstacles increases the number of reference points that can be used for localization. Furthermore, a larger number of samples is needed to test the RRT algorithm. Since there is a high degree of randomness and uncertainty in the method, the amount of data needs to be increased and averaged to create a more accurate data set for comparison and analysis.
To summarize, the Frontier algorithm is generally faster and causes less backtracking than the RRT algorithm. Both methods are causing the robot to get stuck. Therefore, the main cause of the problem seems to be that the calculation of the movement does not take into account future movement. Consequently, the main improvement that can be made to the exploration policy is the use of an exploration route instead of a single point. Using a route and updating the route after reaching a point results in the prediction of future movement and therefore a more reliable exploration algorithm. However, an edge case would have to be included for situations where only one frontier is known.
A second improvement could be the inclusion of a second type of frontier. At the moment, the only possible positions considered are between the edge of known and unknown area. However, it is possible that there is an unreachable area that can be fully or partially uncovered using the sensor without the robot traversing the space. As a solution, a different type of frontier can be introduced that considers the best location for gathering information about unreachable but obtainable area. Because the robot in question does not make use of a 360° but rather a 270° LiDAR, the robot could gather information about its surroundings more efficiently if it uses either a 360° LiDAR or turns around its own axis at the frontier. The extra information gain helps to reduce localization problems. The robot could also arch its movement towards a place if that provides more new data while moving. However, this would increase the length of its path. Therefore, the inclusion of such a run would have to weigh up against the shortest path method in different situations.
Lastly, the explored situation has to be compared to reality. The translation of the discussed problem leaves a lot to be desired. The main issue is the fact that the robot struggles with spaces that are smaller than 1,5-2 m in diameter. In reality, almost all indoor areas have smaller gaps that the robot should be able to traverse. Therefore, the exploration methods analysed are currently not acceptable for use in real life. Furthermore, the robot cannot handle height differences or bridging and tunnelling obstacles that it can pass over or under. The inclusion of a 3D LiDAR as opposed to a 2D Lidar would enable the robot to navigate more complex terrain. For example, terrain with height differences or moving under obstacles such as tables. Increasing the problem from a 2D setting to a 3D setting increase the level of realism. However, it is well known that solving a 3D control problem is significantly more difficult than a 2D situation. Moreover, the initial position in all maps

was at the origin, located in the middle of the maps. In reality, an exploration is expected to happen from an entrance. Meaning that the robot has to start from a boundary position on the map. Thus, both the effect of starting at a boundary and the effect of different initial positions on the quality of exploration is unknown.

V. Conclusion and Future Research

The comparison between the Frontier algorithm and the RRT algorithm for autonomous exploration shows that the frontier algorithm generally outperforms the RRT algorithm in terms of exploration rate, robustness and backtracking, resulting in a more organized exploration. Furthermore, the RRT algorithm can struggle to find points on the far side of the map, leading to a lower average exploration rate on larger maps. Additionally, both algorithms face challenges, including issues with localization, getting stuck in certain situations, and non-optimal exploration. The understanding of these problems is essential for overcoming issues and improving exploration.

To improve the exploration process, several suggestions are proposed. First, the use of an exploration route instead of a single point is recommended to consider future movement and enhance reliability. A second improvement could involve introducing a different type of frontier to uncover unreachable but obtainable areas efficiently. Additionally, upgrading the LiDAR sensor to a 360° range or enabling the robot to rotate at the frontier would provide more comprehensive information and aid in reducing localization problems. Furthermore, the exploration methods should be validated using a better localization and mapping package to reduce inaccuracies and ensure reliable data collection.

It is important to note that the current exploration methods have limitations when applied in real life. The robot struggles to navigate through spaces smaller than 1.5-2 m in diameter. The current setup is also not able to handle height differences or bridging and tunnelling obstacles. To address these limitations, the inclusion of a 3D LiDAR sensor is suggested, which would enable the robot to navigate more complex terrain. However, solving a 3D control problem is known to be significantly more challenging than a 2D situation.

Overall, further research and improvements are needed and encouraged to enhance the performance and adaptability of exploration algorithm. The effect of starting positions and different initial positions on the quality of exploration also need to be investigated. The use of a multi-robot system, of which the beginnings have been set up in this thesis, could help in solving many of these problems.

References

[1] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, pp. 610–617, Apr. 2019. DOI: 10.1109/lra.2019.2891991.

[2] K. AG, *Mobile robots*, KUKA AG, 2023. [Online]. Available: https://www.kuka.com/en-us/products/mobility/mobile-robot-systems (visited on 06/15/2023).

[3] K. AG, *Kuka omnimove (agvs)*, KUKA AG, 2023. [Online]. Available: https://www.kuka.com/en-us/products/mobility/mobile-platforms/kuka-omnimove (visited on 06/15/2023).

[4] C.-Y. Wu and H.-Y. Lin, "Autonomous mobile robot exploration in unknown indoor environments based on rapidly-exploring random tree," *2019 IEEE International Conference on Industrial Technology (ICIT)*, Feb. 2019. DOI: 10.1109/icit.2019.8754938.

[5] B. Yamauchi, "A frontier-based approach for autonomous exploration," *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 1997. DOI: 10.1109/cira.1997.613851.

[6] J. M. Pimentel, M. S. Alvim, M. F. M. Campos, and D. G. Macharet, "Information-driven rapidly-exploring random tree for efficient environment exploration," *Journal of Intelligent & Robotic Systems*, vol. 91, pp. 313–331, Oct. 2017. DOI: 10.1007/s10846-017-0709-0.

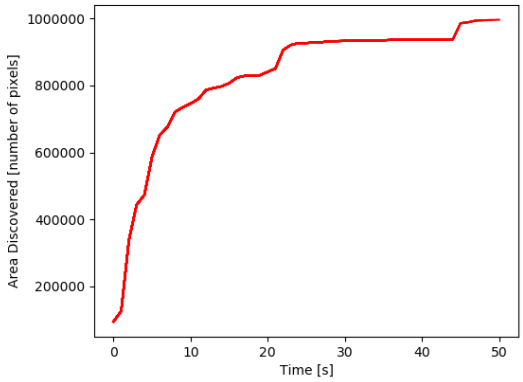

Fig. 6: Graph of the explored area over simulation time for Map 1 using frontier exploration.

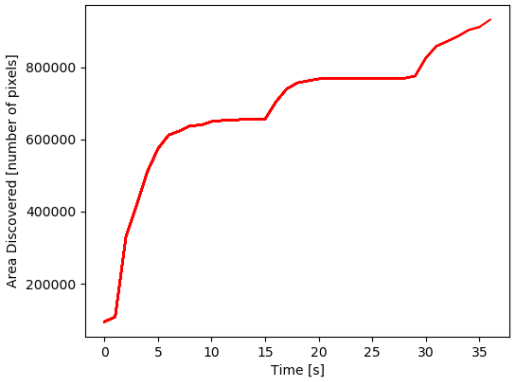

Fig. 7: Graph of the explored area over simulation time for Map 1 using RRT exploration.

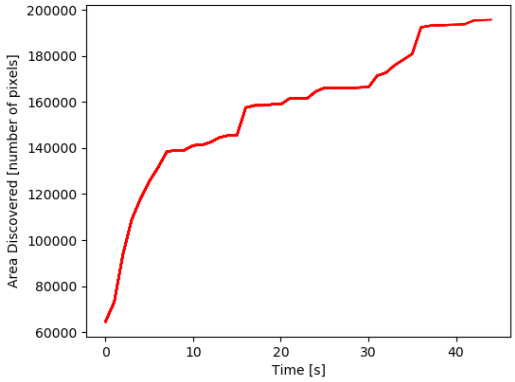

Fig. 8: Graph of the explored area over simulation time for Map 2 using frontier exploration.

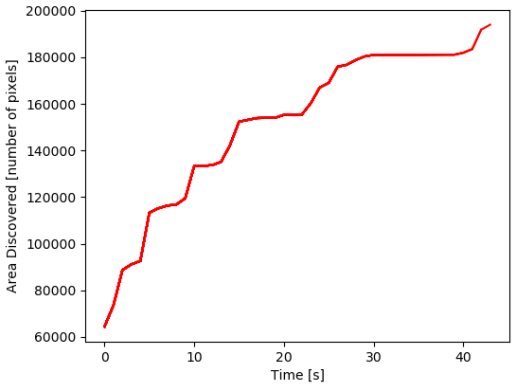

Fig. 9: Graph of the explored area over simulation time for Map 2 using RRT exploration.

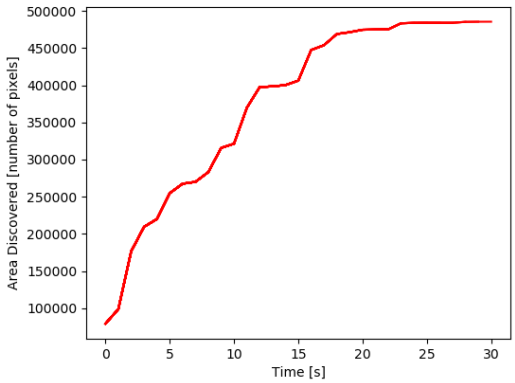

Fig. 10: Graph of the explored area over simulation time for Map 3 using frontier exploration.

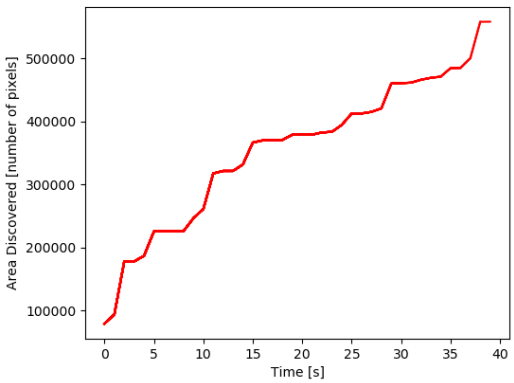

Fig. 11: Graph of the explored area over simulation time for Map 3 using RRT exploration.

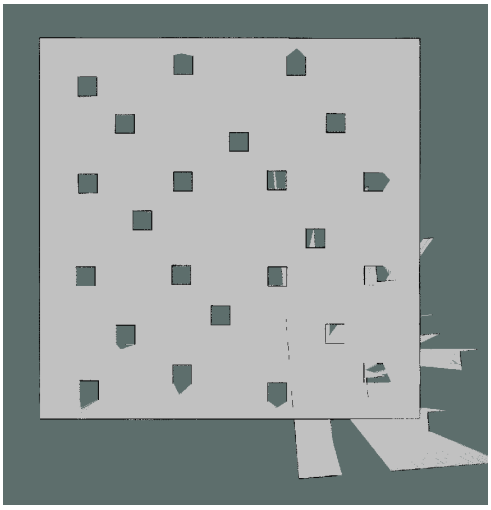

Fig. 12: Map of the explored area of Map 1 using frontier exploration.

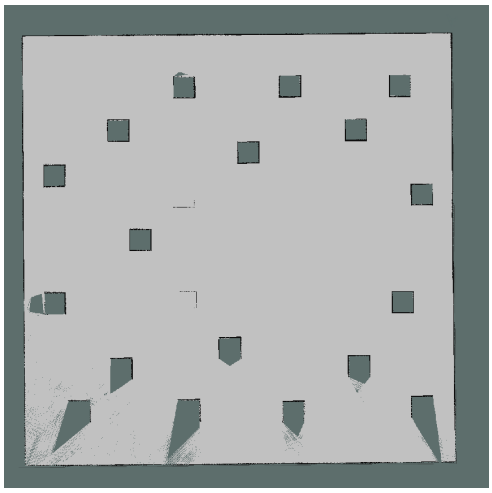

Fig. 13: Map of the explored area of Map 1 using RRT exploration.

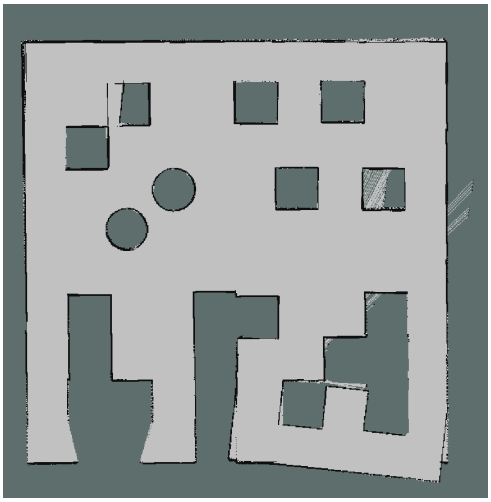

Fig. 14: Map of the explored area of Map 2 using frontier exploration.

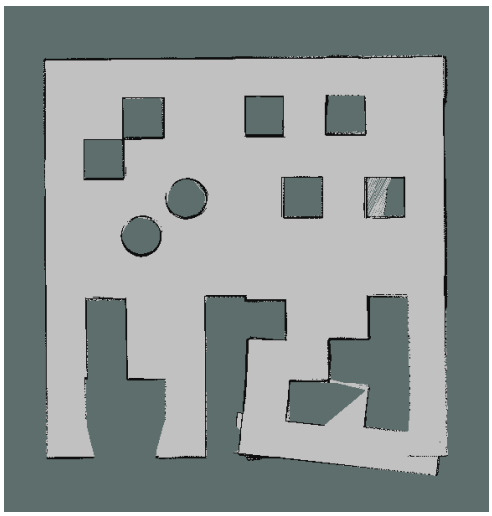

Fig. 15: Map of the explored area of Map 2 using RRT exploration.

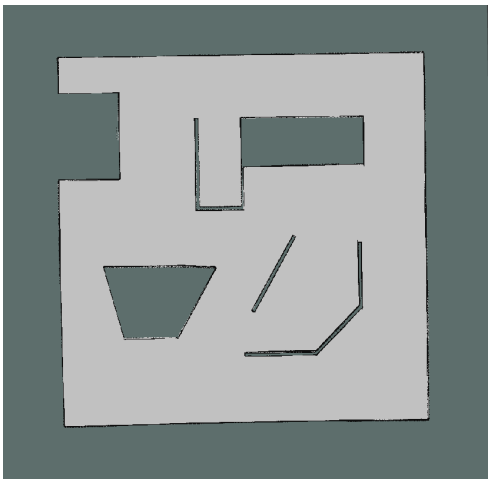

Fig. 16: Map of the explored area of Map 3 using frontier exploration.

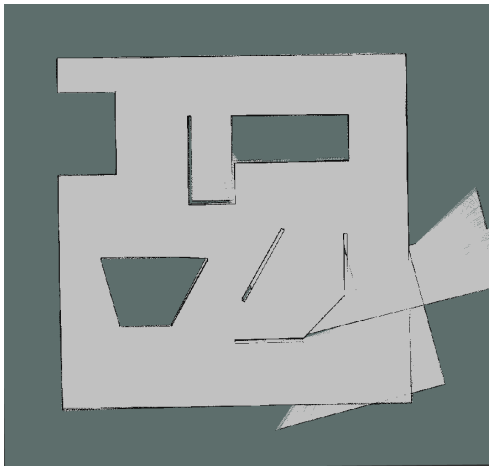

Fig. 17: Map of the explored area of Map 3 using RRT exploration.

TABLE I: Data from three different complex maps.

Map	Method	Attempts	Area (Pixels)	Max. Area (Pixels)	Completion (%)	Avg. Exploration Rate (Pixels/Sec)	Rate Improvement
Map 1	RRT	4	931194	952865	97.73%	25167	
Map 1	Frontier	3	996146	952865	104.54%	19923	0.79
Map 2	RRT	3	194009	195815	99.08%	4128	
Map 2	Frontier	4	196482	195815	100.34%	4569	1.11
Map 3	RRT	2	558056	485497	114.95%	14309	
Map 3	Frontier	1	485497	485497	100.00%	15661	1.09

[7] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017. DOI: 10.1109/iros.2017.8202319.

[8] D. Puig, M. Garcia, and L. Wu, "A new global optimization strategy for coordinated multi-robot exploration: Development and comparative evaluation," *Robotics and Autonomous Systems*, vol. 59, pp. 635–653, Sep. 2011. DOI: 10.1016/j.robot.2011.05.004.

[9] F. Chen, J. D. Martin, Y. Huang, J. Wang, and B. Englot, "Autonomous exploration under uncertainty via deep reinforcement learning on graphs," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020. DOI: 10.1109/iros45743.2020.9341657.

[10] L. C. Garaffa, M. Basso, A. A. Konzen, and E. P. de Freitas, "Reinforcement learning for mobile robotics exploration: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021. DOI: 10.1109/tnnls.2021.3124466.

[11] K. Albina and S. G. Lee, "Hybrid stochastic exploration using grey wolf optimizer and coordinated multi-robot exploration algorithms," *IEEE Access*, vol. 7, pp. 14 246–14 255, 2019. DOI: 10.1109/access.2019.2894524.

[12] F. Gul, I. Mir, L. Abualigah, and P. Sumari, "Multi-robot space exploration: An augmented arithmetic approach," *IEEE Access*, vol. 9, pp. 107 738–107 750, 2021. DOI: 10.1109/access.2021.3101210.

[13] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, pp. 376–386, Jun. 2005. DOI: 10.1109/tro.2004.839232.

[14] B. Lindqvist, A.-A. Agha-Mohammadi, and G. Nikolakopoulos, "Exploration-rrt: A multi-objective path planning and exploration framework for unknown and unstructured environments," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2021. DOI: 10.1109/iros51168.2021.9636243.

[15] Z. Zheng, C. Cao, and J. Pan, "A hierarchical approach for mobile robot exploration in pedestrian crowd," *IEEE Robotics and Automation Letters*, vol. 7, pp. 175–182, Jan. 2022. DOI: 10.1109/lra.2021.3118078.

[16] J. Lee, D. Shiotsuka, T. Nishimori, K. Nakao, and S. Kamijo, "Gan-based lidar translation between sunny and adverse weather for autonomous driving and driving simulation," *Sensors*, vol. 22, p. 5287, Jul. 2022. DOI: 10.3390/s22145287. (visited on 07/26/2022).

[17] B. Gerkey, *Gmapping - ros wiki*, wiki.ros.org, 2019. [Online]. Available: http://wiki.ros.org/gmapping.

[18] Z. Su, J. Zhou, J. Dai, and Y. Zhu, "Optimization design and experimental study of gmapping algorithm," *2020 Chinese Control And Decision Conference (CCDC) Hefe*, Aug. 2020. DOI: 10.1109/ccdc49329.2020.9164603. (visited on 06/15/2023).

[19] I. Ullah, X. Su, X. Zhang, and D. Choi, "Simultaneous localization and mapping based on kalman filter and extended kalman filter," *Wireless Communications and Mobile Computing*, vol. 2020, pp. 1–12, Jun. 2020. DOI: 10.1155/2020/2138643.

[20] P. Bovbel, *Frontier exploration*, wiki.ros.org, Aug. 2020. [Online]. Available: http://wiki.ros.

org/frontier_exploration (visited on 05/09/2023).

[21] J. HÃ¶rner, *M-explore*, GitHub, Nov. 2022. [Online]. Available: https://github.com/hrnr/m-explore.

[22] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001. DOI: 10.1177/02783640122067453.

[23] H. Umari, *Rrt exploration*, wiki.ros.org, Jun. 2020. [Online]. Available: http://wiki.ros.org/rrt_exploration (visited on 05/09/2023).

[24] L. Weining, Z. Tao, Y. Jun, and W. Xueqian, "A formation control approach with autonomous navigation of multi-robot system in unknown environment," *2015 34th Chinese Control Conference (CCC)*, Jul. 2015. DOI: 10.1109/chicc.2015.7260455.

[25] ROS, *Ros.org | why ros?* Ros.org, 2021. [Online]. Available: https://www.ros.org/blog/why-ros/.

[26] M. Yasuyuki, *Distributions - ros wiki*, wiki.ros.org, 2021. [Online]. Available: http://wiki.ros.org/Distributions.

[27] M. Luqman, *Ros/tutorials/understandingnodes - ros wiki*, wiki.ros.org, 2022. [Online]. Available: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

[28] O. S. R. Foundation, *Gazebo : Tutorial : Urdf in gazebo*, classic.gazebosim.org, 2014. [Online]. Available: https://classic.gazebosim.org/tutorials?tut=ros_urdf.

[29] W. Woodall, *Rviz - ros wiki*, Ros.org, 2018. [Online]. Available: http://wiki.ros.org/rviz.

[30] B. Yu, *Jackal-ros*, GitHub, Apr. 2023. [Online]. Available: https://github.com/ybgdgh/Jackal-ROS (visited on 05/09/2023).