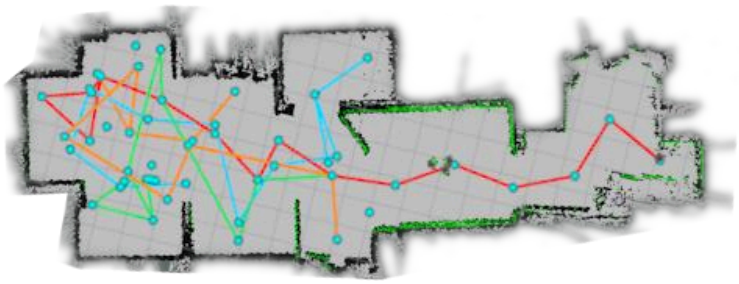


Multi-Robot Collaborative Autonomous Exploration

Efficiently Mapping an Unknown Area

Master's thesis in Systems, Control and Mechatronics & Complex Adaptive Systems

ANDERS WALLSTRÖM
EDWARD BRASK

DEPARTMENT OF Mechanics and Maritime Sciences

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS , SYSTEMS, CONTROL AND MECHATRONICS

Multi-Robot Collaborative Autonomous Exploration

Efficiently Mapping an Unknown Area

ANDERS WALLSTRÖM
EDWARD BRASK

Department of Mechanics and Maritime Sciences

Division of Vehicle Engineering and Autonomous Systems

CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2023

Multi-Robot Collaborative Autonomous Exploration
Efficiently Mapping an Unknown Area
ANDERS WALLSTRÖM
EDWARD BRASK

Master's thesis 2023
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
The finished map of an unknown environment after a successful exploration with four robots.

Chalmers Reproservice
Göteborg, Sweden 2023

Multi-Robot Collaborative Autonomous Exploration
Efficiently Mapping an Unknown Area
Master's thesis in Complex Adaptive Systems , Systems, Control and Mechatronics
ANDERS WALLSTRÖM
EDWARD BRASK
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

ABSTRACT

Autonomous exploration carried out by multiple mobile robots offers significant advantages in efficiently mapping areas. With advancements in communication technologies, collaborative mapping using multiple robots has become a feasible approach. In line with this, Ericsson has supported this thesis project which researches and develops a system using these technologies.

This master's thesis develops a scalable and collaborative multi-robot system for autonomous exploration. It combines the research domains of multi-robot Simultaneous Localization And Mapping (SLAM), autonomous exploration, and map merging to efficiently create a map of an unknown environment. The system was implemented in the Robot Operating System (ROS) and used node-based exploration strategies combined with computer vision techniques to perform collaborative exploration. This thesis serves as a proof of concept and can provide a framework for future research and development in this field. One potential application for collaborative exploration is search and rescue operations.

The collaborative aspects of the system have been effectively implemented and function well. Through extensive hardware testing in three distinct environments, the results demonstrate a significant increase in efficiency and lower exploration time with the collaborative system compared to a single robot exploring autonomously. However, there are accumulating errors within the system, limiting the SLAM accuracy compared to a single robot. Furthermore, the system's performance is highly dependent on the environment and the initial positions of the robots, making it challenging to predict absolute performance and determine optimal parameters in unknown environments.

Future work involves enhancing collision avoidance capabilities, dynamically adjusting exploration parameters based on the characteristics of the unknown environment, and transitioning from independent SLAM algorithms to an integrated multi-robot SLAM system to improve collaborative accuracy.

Keywords: Multi-robot system, Collaborative exploration, Autonomous exploration, SLAM, Map-merging, Scalable framework, Autonomous robots

PREFACE

This report presents the outcome of our master's thesis project conducted at Ericsson in Gothenburg, and the closing stages of our M.Sc degrees at Chalmers University of Technology. Ericsson is a world leading provider of IT and communication technology, and supported the research and development of this project. The interest from Ericsson's perspective is developing a proof of concept and evaluating the possibility of using multiple robots to autonomously explore an area, since this has ties to and is dependent on efficient communication technologies.

ACKNOWLEDGEMENTS

During the development of this master's thesis project, we had support from different people. We wish to express our thanks to Krister Wolff, who was our supervisor and examiner at Chalmers University of Technology. In addition, we want to thank our supervisor at Ericsson, Johan Nicander, for their valuable input and support during this time. We are also thankful to Henric Broström and Jonas Hansryd for their assistance and involvement in our work. Lastly, we would like to thank Ericsson for providing a nice and healthy environment where this thesis project could be carried out.

We greatly appreciate the assistance provided during this time. Thank you.

NOMENCLATURE

ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
FEP	Frontier Exploration Planning
IMU	Inertial Measurement Unit
RANSAC	Random sample consensus
ACO	Ant Colony Optimization
PSO	Particle Swarm Optimization
FEP	Frontier Exploration Planning
RRT	Rapidly-exploring Random Trees
RH	Receding Horizon
NBV	Next best view
ICP	Iterative Closest Point

CONTENTS

1 Introduction

With an increase in connectivity and the usage of different connected machine applications, smart systems that are able to collaboratively perform different tasks have been emerging in the world [1]. Many robotics and machine applications interact with their environment in several different meaningful ways to accomplish their tasks. As such they are required to be aware of their surroundings, which has placed a growing importance on perceiving the environments they operate in [2]. Establishing a digital representation of a robot's environment alleviates the system used and many unmanned autonomous vehicles greatly benefit from using such a depiction to understand and act in their environment [3].

The process of mapping a physical space can itself be automated with the help of Simultaneous Localization and Mapping (SLAM) algorithms [3]. SLAM enables an robot to determine its position relative to the surrounding environment and simultaneously create a digital representation of these surroundings [4]. For this reason advanced SLAM algorithms and their performance have been researched for usage in unmanned autonomous vehicles of different types, commonly unmanned grounded vehicles and unmanned aerial vehicles [5].

The subject of SLAM and autonomous mapping of an area has been a research topic for the past two decades [6]. As we move into an era of high throughput, low latency and reliable mobile communication [7], the possibility of having multiple robots sharing significant amounts of data in real time has increased substantially. Thus, using multiple robots to explore and map an area has gathered more interest for different applications, and will be the focus of this thesis. Collaboratively exploring and mapping an environment has the potential of mapping an area faster and with higher accuracy [8]. Commonly mentioned use cases for autonomously exploring and mapping an area are rescue missions, where response time and efficiency is essential [9], [10], [11], [12].

1.1 Aim

The purpose of this thesis is to develop and implement a proof of concept of a collaborative and scalable multi-robot centralized system that combines several robots to perform mapping and exploration in an unknown environment. This task can be divided into three key parts. Performing SLAM on separate robots in order to create their local maps of the environment, implementing a map merging algorithm that can produce a single global map from the data collected by the several robots running SLAM, and the collaborative autonomous exploration of the environment.

1.2 Specification of Inquiry

The main focus of this thesis is to develop an exploration strategy that makes use of multiple robots' merged local maps of the environment in order to deduce the proper path each robot should take to efficiently and completely map the remaining unknown environment. For this to function, the pipeline from the sensors of the robots to the centralized system needed to be implemented as well. The main parts required for this whole system to function can be divided into the following modules:

- **Robot module:** Each robot will on its own perform SLAM with simple sensors suitable for multiple small robots, e.g. a 2D lidar and an IMU. The output from an robot will be its position as well as its local map.
  Input: 2D lidar feed, IMU feed, or other
  Output: Position of robot, local map

- **Communication module:** Each robot will continuously send its data with position and the map to a centralized processing node over Ericsson's mobile communication network. In addition, the exploration paths will be sent from the Centralized Processing node back to all of the robots.
  Input, for each robot: Position of robot, local map, next exploration path
  Output, for each robot: Position of robot, local map, next exploration path

- **Centralized Processing module:** This module is the centralized part of the system and takes care of the map merging and the autonomous exploration.

  - **Map Merging node:** A centralized processing unit will receive all of the data from the different robots and merge the different maps into one cohesive global map. A multitude of solutions will be investigated in order to accurately, and in real-time, produce a single map of the environment built from multiple robots exploring the environment simultaneously.
    Input from each robot: Position of robot, local map
    Output: Global map
  - **Autonomous Exploration node:** This node will take the global map and produce the next exploration path for each robot to collaboratively explore the whole environment. If no global map has been produced the robots will continue exploring individually.
    Input: Global map
    Output, for each robot: Next exploration path

Since the Robot module and the Communication module will be based on previous works, this thesis will utilize the publicly available code for the SLAM algorithms and handle the communication aspects with built-in functionality in the ROS framework. The main part of this thesis will be to develop the collaborative autonomous exploration strategy which makes use of the map merging algorithm present in the Centralized Processing module. It will also investigate the performance of using multiple robots to perform collaborative exploration and mapping using SLAM algorithms.

1.2.1 Research Questions

As previously described this thesis will heavily revolve around the development and implementation of a collaborative autonomous exploration strategy, which can be generalized as research questions, RQs, that this thesis will answer and discuss.

- **RQ1:** How should the transformation between the local maps of the robots be deduced?
- **RQ2:** How should the local maps from the robots, with a known transform between them, be merged into a global map?

- **RQ3:** How should a multi-robot autonomous exploration strategy be developed so the system is scalable to several robots?
- **RQ4:** How should the collaborative exploration algorithm be designed to enable multiple robots to efficiently and autonomously map the environment, handling the general case when the initial positions of the robots are unknown?
- **RQ5:** Can one determine an optimal number of robots that can be used in unison to fully explore an unknown environment?
- **RQ6:** How does the accuracy of SLAM depend on the number of robots?

1.3 Problem Constraints

It was highly desired by Ericsson that the project could be demonstrated on hardware, and as such the multi-robot collaboration adhered to this general aim. The autonomous exploration implemented in this thesis is limited to indoor spaces. This was done in order to simplify testing and validation. Additionally since the project was developed in an office indoor area it was limited to grounded spaces. Furthermore, the environments explored are considered to be static, since it simplifies the system requirements for object detection and collision avoidance. Because of the focus on the collaborative exploration, we limited the project to make use of existing SLAM algorithms. Therefore, it was not considered to develop a new algorithm or solution for localization and mapping.

1.4 Related Works

Although SLAM and map merging will play a vital role in this thesis, the most closely related research topic is multi-robot exploration and the associated swarm exploration fields. The most relevant related works in this area will be explained and discussed below, and the theory of different SLAM approaches and map merging in 2D and 3D will be covered in Section 2.

1.4.1 Multi-Robot Exploration

As several robots are collaboratively exploring the environment, it is desirable to make this as efficient as possible, optimizing which routes the robots take and in which order they visit the unknown spaces. In 2005, [8] presents the Coordinated Multi-Robot Exploration strategy (CMRE), which is a key early work in this field. This approach's key idea is to make use of the traveling cost to the selected map-based frontier-points, which lie on the border between explored space and unexplored space, and their utility to determine which robot should explore a space. The utility is based on the probability that the target location is visible from other robots' target locations. This approach has been implemented and tested on real robots as well as in simulated environments. In this study the relative positions of the robots were assumed to be known and unknown relative positions was seen as a topic for further research, regardless this explicit coordination approach significantly reduced exploration times compared to only exploring with a single robot.

Other strategies include behavior-based autonomous exploration which is introduced in [13]. This algorithm is said to be applicable to single as well as multi-robot systems. The strategy is mainly based on reactive control which focuses on simplicity and good performance combined with some deliberative approaches regarding the environment. Five different behaviors are introduced, avoid obstacles, for obstacle avoidance, avoid past, to minimize time spent in already explored areas. There is also locate open area, to find the largest open area and continue exploration, disperse, which makes sure multiple robots split up and explore different areas, and lastly explore, which is the default exploring behavior provided other behaviors do not take precedence. The authors mention that there are no standard metrics or benchmarks to evaluate the exploration performance, however they conclude that the behavior-based approach produced good results due to using little computational power and not relying on complex targeting or mapping techniques.

The work in [6] uses a decentralized cooperative exploration algorithm, which builds upon a sensor-based random graph. Each robot uses a laser scanner and senses the surrounding area. Firstly local frontier points are extracted based on the readings from the laser scanner, then these frontier points are incorporated into the global graph. Each robot chooses its next frontier target from the global shared graph, and navigates to it with the graph. Coordination mechanisms are implemented to avoid collisions and increase the collaboration efficiency. Although the naming conventions are different, this work implements a node-based frontier exploration algorithm.

Another node-based frontier exploration algorithm is proposed in [11]. Although the naming schemes of these algorithms are vastly different, the underlying functionality is very similar to the work in [6] proposed more than ten years earlier. In addition, the trajectory planning is executed with a deep reinforcement learning approach to avoid dynamic obstacles, that is trained quickly with the help of human advisors and claimed to be the main contribution of this work. Regarding the multi-robot use case, a dynamic space divider is introduced that splits the map into partitions for each robot to explore. The proposed algorithm is decentralized, and the robots are only allowed to communicate with each other if they are within a certain distance. Furthermore, the robots know their initial starting positions. The proposed method is simulated and stated to work well compared to other common exploration algorithms.

In order to deploy multiple robots and carry out collaborative exploration, the authors in [14] proposed a decentralized exploration scheme. Each robot is running SLAM on its own and the local maps are merged together to a global map with the ”multirobot_map_merge” ROS package, proposed in [15]. The algorithm is based on a 30° field of view reading from the lidar sensor, where the robot aims to explore the borders of the room and dynamically adjusting its trajectory with the input from the distance measurements. The algorithm is verified in simulation, but due to the simplicity of the approach the performance is likely sub-par compared to [11].

1.4.2 Swarm Exploration

Exploration strategies can also rely on stochastic algorithms for multi-robot autonomous exploration. These stochastic strategies usually use swarm based approaches with a large amount of robots. Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) are the most common examples of well known stochastic algorithms used in autonomous exploration. The basic principle of PSO revolves around a swarm of particles, where each of these particles represent a candidate solution to the problem [16]. The ACO algorithm on the other hand uses positive feedback, where each ant

leaves a trail of pheromones behind which following ants detect and are attracted to [17].

In [18] the Darwinian PSO (D-PSO) from [16] is implemented to handle autonomous swarm exploration. D-PSO is defined as running several PSO algorithms in parallel and selecting the best fit, based on the ideas of natural selection. In this case D-PSO is implemented with modification in the inertial parameter and the addition of a fuzzy inferring system to handle dynamical trajectory modifications. The proposed algorithm is tested with simulations on a number of different maps, and it is also compared to other stochastic exploration algorithms. The performance is claimed to be good and up to 20 robots are simulated in a single swarm.

In [12] the ACO algorithm is applied to multi-robot autonomous exploration, where the proposed use case is search and rescue operations of a number of victims. A dynamical fuzzy logic controller based on ACO is proposed, where robots will be repelled by the trailing pheromones. The new algorithm is compared against other algorithms previously deployed by the authors, and achieved good general results where more of the map was explored but the target locations were found a bit slower.

A hybridization of the CMRE presented in [8] and the stochastic swarm algorithm called Grey Wolf Optimizer (GWO), which revolves around mimicking the natural behavior of a wolf pack during searching and hunting for prey, is presented in [19]. The GWO algorithm has some distinct features compared to the popular PSO and ACO algorithms, with a hierarchical structure of leading wolves. The leading wolves guide the rest of the pack towards the prey, and in the optimization algorithm they are the individuals with the highest fitness values. The proposed algorithm combines GWO with CMRE for multi-robot exploration purposes, and is demonstrated with simulations in 2D. Although this work was presented in 2019, the simulation results are only compared with CMRE, from 2005.

1.5 Motivation and Contributions

The interest in this project from Ericsson's perspective lies in exploring potential future applications that can benefit from mobile communication systems. This thesis will therefore provide a proof of concept for a system which showcases a collaborative robot application. The technologies used and expanded upon in this system intends to motivate the development of further connected applications.

In the fields of SLAM and autonomous exploration both areas have been researched with the collaboration of multiple robots as a focus. This thesis seeks to contribute with the combination of these two concepts in a multi-robot setting to collaboratively create a cohesive map of an unknown environment. Furthermore, several studies [20], [6], [11] develop autonomous exploration with known initial positions of robots without multi-robot SLAM. The more general case of unknown initial positions is investigated in this thesis together with multi-robot SLAM.

The system developed is a proof of concept of autonomously exploring an area with many small and scalable robots in a collaborative way. It is meant to show that the collaborative approach taken in the thesis is applicable to larger systems and that the ideas implemented could be extrapolated to projects in mapping 2D environments as well as 3D environments. The system built is a framework for multi-robot collaboration within the autonomous exploration field and provides groundwork for future research.

2 Theory

As mentioned this thesis aims to implement a multi-robot autonomous exploration system and strategy. In order to achieve this, the specific algorithms and parts of the system chosen need to be outlined and discussed accordingly. Therefore this section will discuss the several approaches used in different map merging algorithms in 2D and 3D, as well as provide a view of different 2D and 3D SLAM algorithms. This is because mobile grounded robots can still apply different 3D strategies by using cameras and other 3D sensors, and are not limited to the 2D plane. It will also give an in depth view on autonomous navigation, since its research is closely related to autonomous exploration. This is due to the nature of the autonomous navigation problem which is navigation utilizing a known map of an area, compared to autonomous exploration, which is the idea of navigating an unknown area and mapping it. The current state of the art of single robot autonomous exploration is also asserted.

2.1 2D SLAM and Maps

Simultaneous Localization and Mapping, SLAM, is the concept of a robot identifying its environment and localizing itself in it, which can be described by equation (2.1). The idea is to estimate the joint posterior distribution of its position $x_{1:t}$, and the map of the environment m, given measurements of odometry, $u_{1:t}$ and sensor data, $s_{1:t}$. Time is denoted t, thus $1:t$ represents measurements from time 1 until t [21].

$$p(x_{1:t}, m \mid s_{1:t}, u_{1:t-1}) = p(x_{1:t} \mid s_{1:t}, u_{1:t-1}) \cdot p(m \mid x_{1:t}, s_{1:t}) \tag{2.1}$$

The two right-hand factors of the equation can be deduced with Baye's formula, which is generally used to update the probability distribution of the chosen parameter given some data [21].

Common 2D SLAM techniques include GMapping, KartoSLAM, [4], and Cartographer [22]. GMapping uses a Rao-Blackwellized Particle Filter SLAM, first proposed in [23]. In [21], the RBPF was improved in order to reduce the computational load and memory required, and was shown to function well on a laptop, which today is more than 15 years old. This GMapping SLAM package is one of the most commonly used as of 2021 [24]. KartoSLAM on the other hand uses a graph-based SLAM approach, where each node in a graph represents a pose of the robot combined with sensor measurements. The edges between the nodes represent the robot's motion between the consecutive poses. This graph-based approach usually has advantages when mapping larger areas and environments, since KartoSLAM uses very little memory.

As described in [22] and [25] Google's laser-based SLAM algorithm Cartographer is a real-time mapping solution. It uses laser scans to iteratively produce submaps, which are then subjected to scan matching. It compares recent scans and submaps to eliminate the accumulation of pose estimation errors that occur during long usage. After the submaps are created they are checked for loop closure. Loop closure is the idea that the robot can recognize if it has previously visited an area. The submaps are represented by probability grids and the scan matching finds a scan pose which maximizes these probabilities at the scan points in each submap.

Occupancy grid maps is one of the more popular representations of 2D environments and is also the output of the commonly used 2D lidar SLAM algorithms GMapping, KartoSLAM, and Cartographer. The occupancy grid is a 2D matrix where each entry contains a number determined by whether the square is free, occupied or unknown [24]. See Figure 2.1 for a visualization of a small robot running the GMapping SLAM algorithm, and producing an occupancy grid.

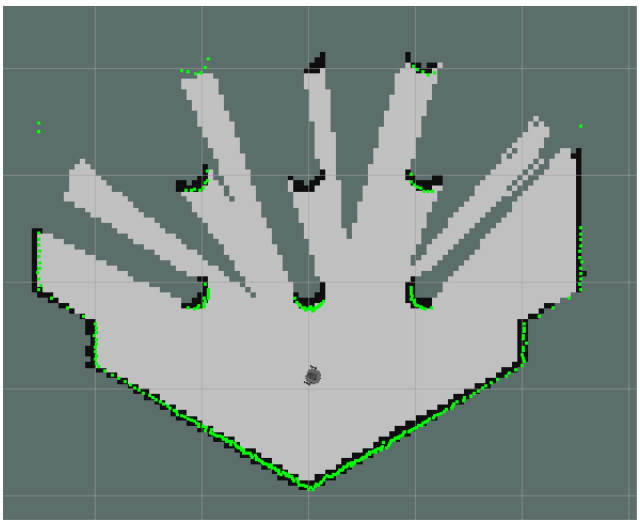


Figure 2.1: Simple occupancy grid map created with GMapping and visualized in RViz. Light gray areas represent unoccupied space, black represents occupied, green is the current 2D lidar measurements and the pixels with a dark gray color is unknown environment.

The output from the 2D SLAM algorithms can then be used to perform map merging, allowing multiple robots to combine their output into a single merged map. Occupancy grid maps have several methods for merging such as optimization, feature based and the Hough transform [26]. The most common 2D map merging algorithms will be further explained.

2.1.1 Occupancy Grid Maps: Optimization

The metric map merging problem can be solved with an optimization approach, which revolves around designing an appropriate objective function to maximize or minimize. This section presents a number of works focusing on an optimization approach.

A mathematical approach is taken in [27] where the map-merging process from occupancy grids is viewed as an optimization problem defined by three parameters, translation in X, translation in Y, and a rotation Ψ. Several common algorithms in the field are implemented, like multi-point hill climbing and simulated annealing and compared with a brute-force approach that is used as a reference. In addition, a novel algorithm based on motion planning is proposed and compared with the others, and proved to perform better given a number of test cases with a new similarity metric introduced by the authors.

Continuing the work from [27], in [28] the computational heavy load is sought be be decreased and is a key work in this field. A stochastic search algorithm relying on a heuristic similarity metric is presented. The heuristic similarity metric is claimed to be fast to compute, and provides gradients

in rotation and translation, which is used to guide the random walk search algorithm. Several real-world scenarios are sampled and tested on the algorithm, and generally provide good results even with maps that have a small percentage of overlap. In addition, a metric to measure if the merging was successful or not is introduced, and claimed to provide a reliable indication.

The study in [29] sought to decrease the computational load of the algorithm in [28] even further. A similar objective function is used, and an adaptive genetic search algorithm is proposed to find the optimum. The proposed algorithm is claimed to be more effective than the random walk proposed in [28], although very few test cases are shown. In [30] the authors claim to further improve the objective function from [28], and propose several genetic algorithms to solve the problem. Several experiments are carried out and it is claimed that the proposed algorithm can fulfill the merging tasks effectively.

2.1.2 Occupancy Grid Maps: Hough Transform

Research presented in [31], proposes a novel algorithm merging two occupancy grids. The project relies heavily on calculating different measurements on each of the maps, that are later used to find the translation between them. This is claimed to be a faster approach than any stochastic numerical searches. First the Discretized Hough Transforms are calculated for each of the maps and used for cross-correlation to determine a number of rotations that are promising. Then, the rotation is applied to the corresponding map, and the occupied cells are projected along the x and y axis respectively, to compute the so called X-spectrum and Y-spectrum. Again the cross-correlation is taken between these signals to deduce a number of possible translations. With a rotation and translation, the transform between the maps can be deduced, and are ranked given a formula. The algorithm is claimed to be fast, and a number of tests are performed and evaluated. Most of the tests are built upon very similar maps, e.g. when two robots separately have explored the same area. This paper is the foundation for several other Hough based matching algorithms.

In [32] the Hough Transform based map merging algorithm proposed in [31] is explored further and claimed to be improved so that maps with a small amount of overlap between them can be merged anyways. This is done by finding the overlaps with the Hough Transform and then deducing the remaining translation from the overlapping parts of the maps. The algorithm is verified in simulation and real-world experiments, and shown to achieve faster and more accurate results then the adaptive random walk method proposed in [28].

Expanding on their own work from [32], in [33] the Hough Transform based method is claimed to be further improved and provides better results with map merging with small overlaps, and the proposed algorithm is claimed to be fast and efficient. The algorithm is compared with several other map merging algorithms and achieves high accuracy with small amount of processing time. Although one experiment on a specific map shows that at least 50 % overlap is required to produce a high similarity score.

2.1.3 Occupancy Grid Maps: Feature Based Merging

The authors in [34] present an approach where two separate maps are updated simultaneously, an occupancy grid map and a point map. The first step of the proposed algorithm is to match the occu-

pancy grid maps, which provide a first estimate that the point maps further update with an approach such as Iterative Closest Point (ICP). This paper focuses mostly on the extraction of features from the occupancy maps and then matching them. Features from the Harris detectors are extracted, and then a modified RANSAC algorithm takes care of the matching and proposes several hypotheses of possible transformations. The algorithm is extensively tested and claimed to produce accurate results.

In [15], a map-merging heuristic algorithm is presented that builds upon computer vision concepts. The occupancy grids produced by SLAM are viewed as images. Thus, if parts of different maps overlap, the images will look similar in those areas. This project proceeds by extracting features, matching them and finding an initial transformation between them deduced from the number of inliers to the model, which in this case is a transform. Then a bundle adjustment is carried out to improve the initial estimation. This is a numerical method that builds upon the Levenberg–Marquardt algorithm to minimize the square of the reprojection errors. Finally the maps are merged with the corresponding transforms. This work can be found as an open source ROS package called *multirobot_map_merge*.

2.2 3D SLAM and Maps

Map merging in 3D has been researched with several different approaches and is often entwined with the SLAM algorithms that are used to acquire said maps. Mapping in 3D has seen an increased use of more complex sensors such as 3D lidars and cameras of different varieties. Thus, the datasets gathered are significantly larger and therefore set higher demands on the hardware in terms of memory as well as computing power. To counteract the inevitable increase of data when mapping in three dimensions instead of two, much research has opted to focus on extracting features from the environment to build feature-based maps [35] rather than the common metric grids.

2.2.1 3D Lidar SLAM

3D lidars have become widely used, especially outdoors, and they produce a dense point cloud with rich feature tracking capabilities. Its downsides are that it is highly expensive and requires a large amount of processing power [36]. The state of the art SLAM algorithms for 3D lidars such as [37] and [38] present impressive results applicable to autonomous cars, where IMU data are tightly integrated in the SLAM algorithms to improve the pose estimation and environment reconstruction, running in real time with a 10 hz sampling rate from the lidar.

2.2.2 3D Visual SLAM

Visual SLAM refers to SLAM algorithms based on camera input, where mono, stereo and RGB-depth (RGB-D) cameras are commonly used. In contrast to 3D lidar SLAM the hardware is relatively cheap. The most common visual SLAM algorithms are the ORB-SLAM family and RTAB-SLAM, which will be presented below [39].

In [40], the first ORB-SLAM algorithm is introduced for monocular cameras, building on many well-known computer vision concepts. This is a visual based SLAM that extracts features from the pictures, called oriented fast and rotated brief (ORB). The features are matched frame to frame to extrapolate the pose of the vehicle as well as a map of the environment in real time. In addition, loop closure is implemented that allows the algorithm to detect when it is looking at a previously visited or known environment and can thus ”close the loop”. The global bundle adjustment together with the loop closure improve the accuracy of the algorithm. Continuing the work from the first ORB-SLAM algorithm, in [41] the ORB-SLAM2 algorithm is proposed that enables the use of stereo and RGB-D cameras as well. These sensors allow for depth to be accurately estimated, which greatly improved the pose estimation and mapping. However, the newest state-of-the art ORB-SLAM3 algorithm in [42] supports monocular cameras used together with an IMU for accurate scale and pose estimation. This algorithm is the first in the ORB-SLAM family to integrate IMU data into the SLAM algorithm.

The point clouds generated from these ORB-SLAM algorithms are in general quite sparse, the RTAB-Map based SLAM algorithm presented in [43] produces a much denser point cloud. RTAB-Map or real time appearance-based mapping is a multipurpose graph-based SLAM. RTAB-Map is an open-source library which supports both visual and lidar based SLAM, and can be used with different sensor configurations. Similarly to other appearance based approaches RTAB-Map uses loop closure to determine if the location has been seen by the robot before. Loop closure commonly becomes more and more demanding due to the increase in the number of locations that need to be compared as the robot visits more locations. RTAB-Map uses multiple strategies to optimize this and manages to run loop closure in real time.

In [44] different SLAM configurations are compared. It is claimed that RGB-D cameras are usually preferred over stereo cameras since they can detect textureless surfaces. The extra field of view a lidar provides is a huge advantage over single cameras and is the better choice for 2D navigation. However, if 3D obstacles need to be detected a multi-camera setup is favorable in terms of cost compared to 3D lidars.

A comparison between the OctoMap and RTAB-Map is presented in [45], where the goal is to investigate the suitability of the different map techniques to map a dynamic environment to use for robot applications with a RGB-D camera. An OctoMap is a representation of 3D space, done in a very similar fashion to an occupancy grid, as it represents occupied, unknown and free space with voxels. RTAB-Map with its appearance based mapping is claimed to have significant advantages in recognizing features such as important details in rescue operations, while also being able to distinguish between water and solid ground. On the other hand, the OctoMap has been more widely used for unmanned aerial vehicle navigation with its spatial representation of occupied and free space, as well as being much more memory efficient than RTAB-Map.

2.2.3 Multi-robot SLAM and 3D Merging Algorithms

Performing SLAM in 3D requires processing of more data, regardless if it is lidar or visual based. The ORB-SLAM family relies on extracting ORB-features and matching them frame to frame, and thus will not produce a dense point cloud. The collaborative state of the art multi-robot SLAM algorithms utilizing feature based 3D SLAM, such as [46] and [47], are relying on these features to perform multi-robot SLAM and merge the different maps.

The authors in [46], introduce a multi-robot centralized visual-inertial SLAM algorithm called CVI-SLAM. They utilize visual-inertial SLAM to create a novel framework that is tested on many of the common visual SLAM datasets. The robots are both feeding the centralized server and receiving map updates from it, which is claimed to improve the pose estimation of each robot and to be the first collaborative SLAM algorithm with two way communication.

A novel centralized map merging algorithm built on Visual-Inertial (VI) SLAM is presented in [47]. Each robot is running VI-SLAM on its own, and maintains its local map, while continuously sending updated map information to the centralized server. The merging process is heavily relying on many of the computer vision concepts present in visual based SLAM algorithms, keyframe detection and matching, loop-closure and global bundle adjustment. The proposed algorithm is tested on many standardized datasets and compared with similar work such as CVI-SLAM [46], VINS-mono [48] and the merging aspect of ORB-SLAM3 [41], and achieves a state of the art accuracy and scalability with up to twelve robots running simultaneously.

Additionally, current research is also focusing on the map merging problem in 3D. In these cases it is assumed that the SLAM algorithms produce a dense point cloud or an OctoMap, which requires a 3D lidar, RGB-D or stereo camera. In current state of the art merging algorithms, such as [49], [50] and [51], they make use of these finished 3D maps from each robot as input to their merging algorithms.

In [49], an algorithm that merges 3D OctoMaps is proposed and tested in simulation. The usage of OctoMap based maps are claimed to be more memory efficient than their counterparts, and provides a foundation for mobile robots moving freely in 3D space. This work aims at providing an initial algorithm for extending OctoMaps to multi-robot systems, and focuses on the merging algorithm when the transformation between two maps are known. The merging algorithm is divided into several cases, depending on the resolution of the given OctoMaps and if the specific area is mapped by both of the robots or not.

The study in [50] works to further improve upon the ideas introduced in [49]. An algorithm which takes an initial guess of the transformation between the two OctoMaps that should be merged is put forth. The OctoMaps are transformed to a point cloud where ICP is applied on subsets of points to yield an improved transformation between the maps. Then the maps are merged with the same algorithm as in [49]. The algorithm is verified with simulations in Gazebo and is shown to compensate for inaccurate initial transformations.

The OctoMap itself can be used to perform ICP and deduce an accurate transformation between two maps, without the need to first transform it to a point cloud. That also brings along advantages with the tree based structure present in OctoMaps, where nearest neighbor searches can be carried out with reduced complexity. Utilizing these advantages, an algorithm is proposed in [51] that takes an initial transformation between two OctoMaps. With ICP performed directly on the OctoMap, a set of matching points can be found and used to refine the initial transformation. Then the OctoMaps are merged together with a similar approach as in [49]. The algorithm is tested both with simulations in Gazebo and on publicly available datasets, and produce good results when the initial transformation is accurate.

2.3 Autonomous Navigation

With more advanced SLAM algorithms producing better pose estimation and map accuracy, the use cases with these algorithms are increasing as well. The navigation problem refers to finding the optimal path from point A to point B in a given map of the environment, while avoiding obstacles. Some of the applications where autonomous navigation can be used involve warehouse logistics, delivery robots and indoor service robots [3].

In 2D environments, the output of 2D SLAM algorithms in the form of an occupancy grid is suitable for navigation purposes [35]. In [52] an overview of autonomous navigation algorithms is given. One of the most basic ideas for global navigation is based on graphs, where the graph nodes represent positions in the environment and then edges are distances between those nodes. The classic Dijkstra algorithm has commonly been used to solve the problem of finding the shortest distance from a start node to the goal node. Another algorithm for graph-based navigation is A*, which has been widely used the past years. It is a search algorithm which recursively explores the graph until the shortest path is found. Another common navigation strategy is the use of stochastic algorithms, where variants of PSO and ACO have been used. In recent research these types of algorithms are often combined with fuzzy logic or deep learning based approaches to reduce the complexity and computational load, while at the same time optimizing the path. In addition to these global map navigation algorithms, many obstacle avoidance and trajectory algorithms are presented and often used in unison with the global algorithms to detect and avoid obstacles and smoothing the path for the vehicles.

Another review of autonomous navigation algorithms is given in [3]. It claims that the global graph based planning algorithms such as A* and ACO, coupled with a local trajectory planner are often insufficient with accumulating errors along the pipeline. The ideas further explored in this work are based on reinforcement learning which takes input directly from the sensors, which was first introduced in 2013. The biggest advantages is that deep reinforcement learning has a low dependency on sensor accuracy and does not require an accurate map to function well. Deep reinforcement learning aims to maximize the reward through interaction with its environment, which is used for training. It has been widely used and researched the last couple years and has several different approaches within the field.

While autonomous navigation has traditionally been implemented on 2D vehicles with an occupancy grid map, the recent rise in popularity with unmanned aerial vehicles has caused a rise in 3D autonomous navigation as well, which comes with many additional challenges. One of them being that the map output from many 3D SLAM algorithms such as the ORB-SLAM family, are feature maps, and not 3D occupancy maps such as the OctoMap. Thus, they can not easily be applied or utilized in autonomous navigation, since feature based maps cannot guarantee to represent what is considered occupied space in the real world [53].

2.4 Autonomous Exploration

In contrast to navigation tasks, the exploration field revolves around efficiently exploring an unknown environment. As stated by Yamauchi [54, p. 146] “Given what you know about the world,

where should you move to gain as much new information as possible?". This is the core idea behind autonomous exploration. The work in [54] moves on to introduce Frontier Exploration Planning (FEP), which revolves around frontier points at the boundary between known and unknown space. When the robot visits a frontier point, it sees into the unknown space and the border of the known territory is expanded. Utilizing an occupancy grid for mapping, the algorithm takes the robot to the nearest, accessible and not visited frontier point with a depth first search in the map. This frontier-based approach relies on extracting frontiers from the map itself, and then carrying out path planning on the map in order to navigate to the chosen frontier. A visualization of the map-based FEP algorithm can be seen in Figure 2.2.

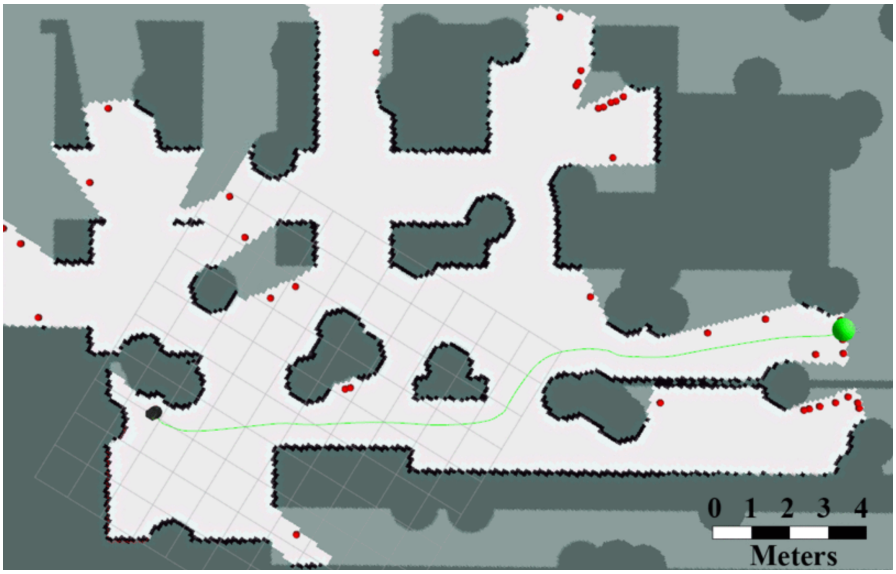


Figure 2.2: The map-based frontier exploration algorithm. The white cells are free space, black cells are occupied space and gray areas are unknown space. Red points are frontier cells extracted from the map, and the large green point is the chosen frontier cell to explore next. Source: [55]

Another frontier-based exploration approach is implemented in [20], relying on extracting frontiers from information nodes instead of from the map. The work aims to deploy physical information nodes that are used for observation, communication and navigation. The information nodes and frontier points are assembled in a graph with edges connecting them. This graph is then used for navigation purposes. However, the algorithm is only tested in simulation with virtual information nodes. A large number of nodes are required to explore a small area, making the use case of physical information nodes questionable. Further improvements of this algorithm are proposed in [11]. A visualization of the node-based frontier exploration algorithm can be seen in Figure 2.3.

FEP-based algorithms are widely used in autonomous exploration, and have become the standard for 2D based exploration [5]. Another popular alternative is the Rapidly-exploring Random Trees (RRTs) algorithm coupled with the Receding Horizon (RH) strategy [5], which has proven to be especially effective in 3D exploration. The RRT data structure is first introduced in [56], and utilizes random samples to construct a tree within the free region of a mapped space, with a bias of expanding the tree towards unexplored places. It is suitable for path planning and exploration algorithms, and a visualization can be found in Figure 2.4.

The RH approach is implemented for autonomous 3D navigation with a stereo camera in [57] and

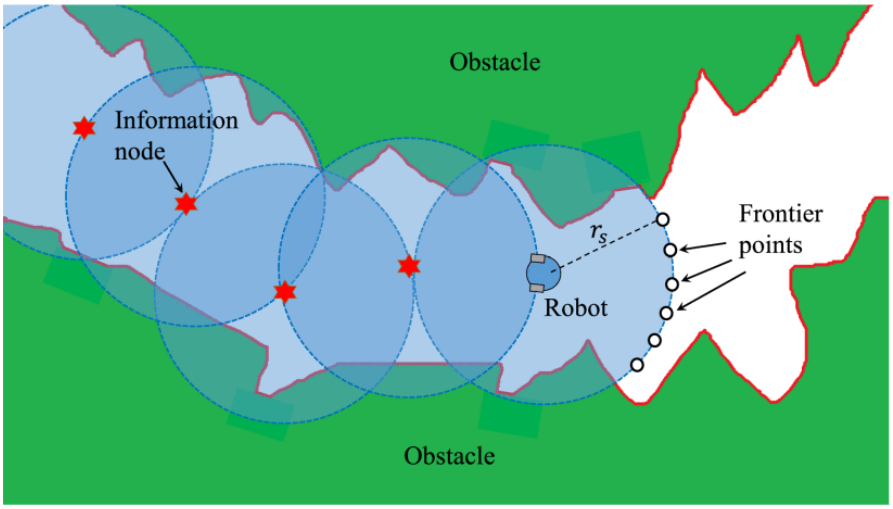


Figure 2.3: The node-based frontier exploration algorithm with a single robot. The nodes marked as information nodes are visited frontier-points. The blue circles represent the observation range. The robot is traveling towards one of the new frontier-points with the lowest cost. Source: [11]

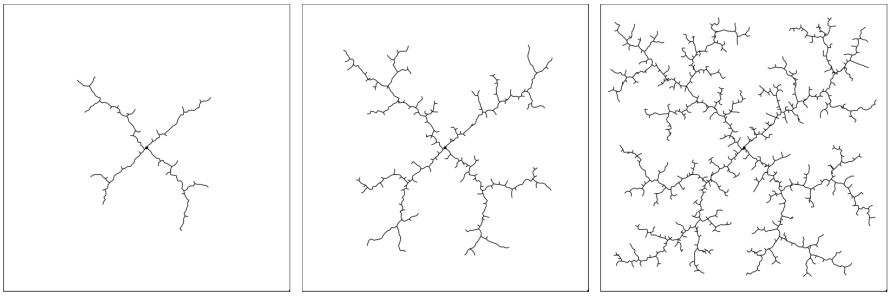


Figure 2.4: Visualization of the RRT data structure. From left to right, the tree is growing and tends to expand towards the corners of the square. In contrast to FEP, RRT-based exploration plans much further ahead with the tree search. Source: [56]

modified to a Receding Horizon Next Best View (RH-NBV). From a starting position, a number of iterations are executed to construct a RRT in the known, free space. Each branch of the tree is then evaluated based on how much unknown space can be explored. Then the robot takes one step in that branch, and the whole process is repeated. Thus, the robot moves towards the next best view in each iteration, hence the name. The algorithm is compared with a frontier-based exploration approach, and shown to scale better with regards to computational complexity.

In [9] the node-based FEP algorithm and RH-NBV are compared and combined in a single robot exploration algorithm. FEP is claimed to more fully explore the global environment at the risk of missing some smaller details in it, whereas RH-NBV is very good at fully exploring a smaller area with its details included, but may miss entire sections of the global map. The proposed algorithm has a hierarchical approach, where FEP plans the global exploration but RH-NBV takes care of the local path. The proposed algorithm is shown to perform well in simulations compared to other state of the art exploration algorithms. In addition the algorithm handles 3D movement well and could be utilized for autonomous drones.

An autonomous exploration and mapping based method constructed from a topology-grid hybrid map is presented by the authors in [58]. This approach fuses the ideas behind a topology map, which

has motion costs and information gain, with the grid maps, which represent space and are used for navigation and localization. This strategy also employs frontiers in the environment as a basis for the exploration, making use of detected unknown space as well as larger discrepancies in the distance measurements to note areas worth exploring. This combination of topological data together with occupancy maps makes the robot explore more of the area at a faster rate when compared to a naïve exploration and a forward simulation algorithm presented in earlier works in the field. According to the report the use of information gain and potential path cost from the topological map greatly reduced the required exploration time.

The exploration algorithm in [10], bases its decisions on a topological Semantic Road Map (SMR) and an occupancy grid which are deployed in parallel during exploration. It is claimed that the commonly used FEP with occupancy grids are computationally costly for large environments. The RRT based NBV algorithms on the other hand are claimed to be more efficient, although the trees need to be constructed at each iteration step, which in itself is stated to be unnecessary. This work proposes the SMR map to enhance the efficiency in NBV based autonomous exploration algorithms, where the key idea is to perform the NBV selection directly on the SMR. The proposed algorithm is tested in simulations and in the real world with a 2D laser scanner running GMapping, and compared to other RRT based algorithms, called Max Info and Nearest Frontier, and is shown to reduce the computational load and the required time to explore an environment.

Other approaches to autonomous exploration such as using reinforcement learning to guide the decision making is analyzed in [59], where they use Graph Neural Networks (GNNs) together with reinforcement learning. This approach uses exploration graphs as a generalized topological data structure to represent states and actions relevant to the exploration. This approach is mainly built on supervised learning and using cost and reward functions to enable the reinforcement learning to take place and later be able to guide the decision making in the autonomous exploration. According to the paper the reinforcement learning policies introduced provide the best real time performance among the multiple combinations of GNNs and different reinforcement strategies they tested in the project.

Similarly, a goal-driven autonomous exploration based around deep reinforcement learning is introduced in [60], where a global goal is used to guide the robots' general exploration strategy combined with local navigation policies. The work presents a deep reinforcement learning motion policy which it combines with a global navigation strategy in order to better locally explore unknown environments. The idea is that combining a goal-driven strategy with the deep reinforcement learning allows them to support each other in the exploration process. The shortcomings locally are supported by the fact that there is a global goal set, which in turn aids the autonomous exploration.

3 Method

This section explains the implementation of the system and how its many parts were developed to work in unison. It also addresses the hardware used and the details regarding the main modules that are the foundation of the collaborative exploration.

3.1 Hardware

After investigating the field of autonomous exploration and SLAM, it was decided that the TurtleBot3 Burger robot [61] provided the functionality and qualities that the project required and would be utilized as the medium on which the different algorithms and strategies were employed. TurtleBot3 was chosen due to its usability and compatibility with ROS and SLAM algorithms. This in addition to being relatively small and affordable made it the most attractive option for this project. Choosing to use TurtleBot3 facilitated the implementation of the desired system since it allowed the system to be simulated easily, and it can be seen in Figure 3.1. The TurtleBot3 comes equipped with a LDS-02 sensor, which has 8 m of range with a measurement error of up to 5% at $6-8$ meters [62].

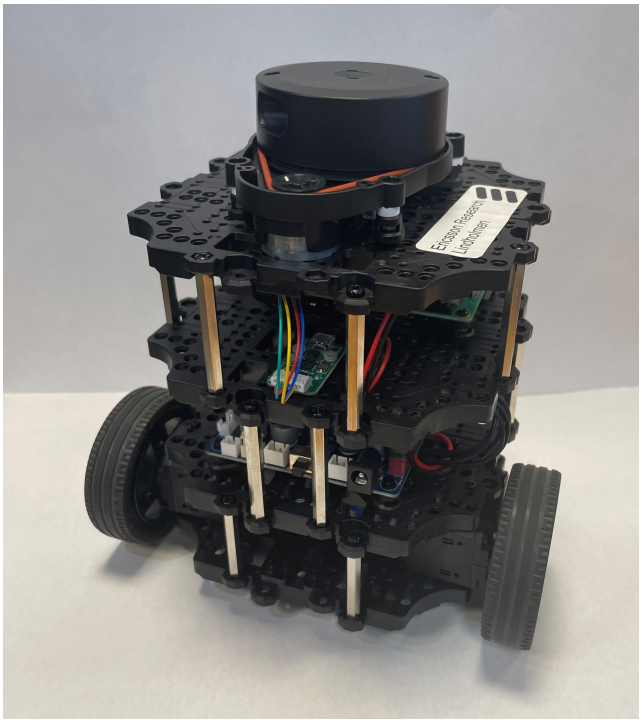


Figure 3.1: An assembled Turtlebot3, which was used in the project. On each side of the robot there is a wheel driven by an independent actuator. The cylinder on top is the lidar sensor.

When running several robots the system had a high demand on the computing power. The centralized processing and main system was therefore run with a computer that had a AMD Ryzen 9 5900X processor as well as a NVIDIA RTX 3090 graphics card. The communication was done with a Wireless LAN network. The data transfer required by the system was low. The computer running

the system sends about 105 kb/s and receives around 560 kb/s with four robots. The exact hardware requirements for the system were not measured, however these hardware components managed to run the collaborative autonomous exploration with ease.

3.2 System Overview

The system implemented in this thesis consists of three main modules. The Centralized Processing Module, which takes care of the map merging and autonomous exploration, the Robot module, that runs the low level functions as well as the SLAM algorithm GMapping for each robot and the Communication module. The Communication module was realized with the standard communication functionalities built into ROS, which required little setup and implementation to be used. Therefore, its details will not be further expanded upon despite the fact that the communication aspect of the system is crucial for the collaboration. Both the Centralized Processing module and the Robot module contains several nodes by themselves. An overview of the system can be seen in Figure 3.2.

3.3 Robot Core

The Robot Core node handles the core functionality of the TurtleBot3, including the drivers for the actuators and the lidar. This allows other packages to read the current scan message from the lidar, and the current odometry estimation of the position, in addition to sending commands of what desired velocity the robot should strive to follow.

3.4 GMapping

GMapping was the SLAM algorithm of choice used in this thesis. While cameras and other sensors are powerful tools and could be mounted on the robot, they are usually used in combination with feature-based SLAM algorithms. Feature based maps cannot be directly used for navigation, which is a disadvantage. As seen previously in Section 2.1, lidar-based SLAM options are commonly used in mobile robots and the TurtleBot3 with its lidar is designed to run such SLAM algorithms.

When selecting which 2D SLAM algorithm to use in the system the main candidates where GMapping, KartoSLAM and Cartographer due to usability and their integration in ROS. The three different SLAM strategies were tried out on a single robot early in development. GMapping was chosen on the basis of being most commonly used as well as having a good reported performance, which was consistent with what was seen when testing the three different algorithms. The parameters used by the GMapping algorithm can be found in Appendix B.1. GMapping takes the odometry estimation from the Robot Core and the scan values to produce an occcupancy grid of the environment and an estimation of the robot's position in that environment. GMapping is available as an open-source ROS package [63].

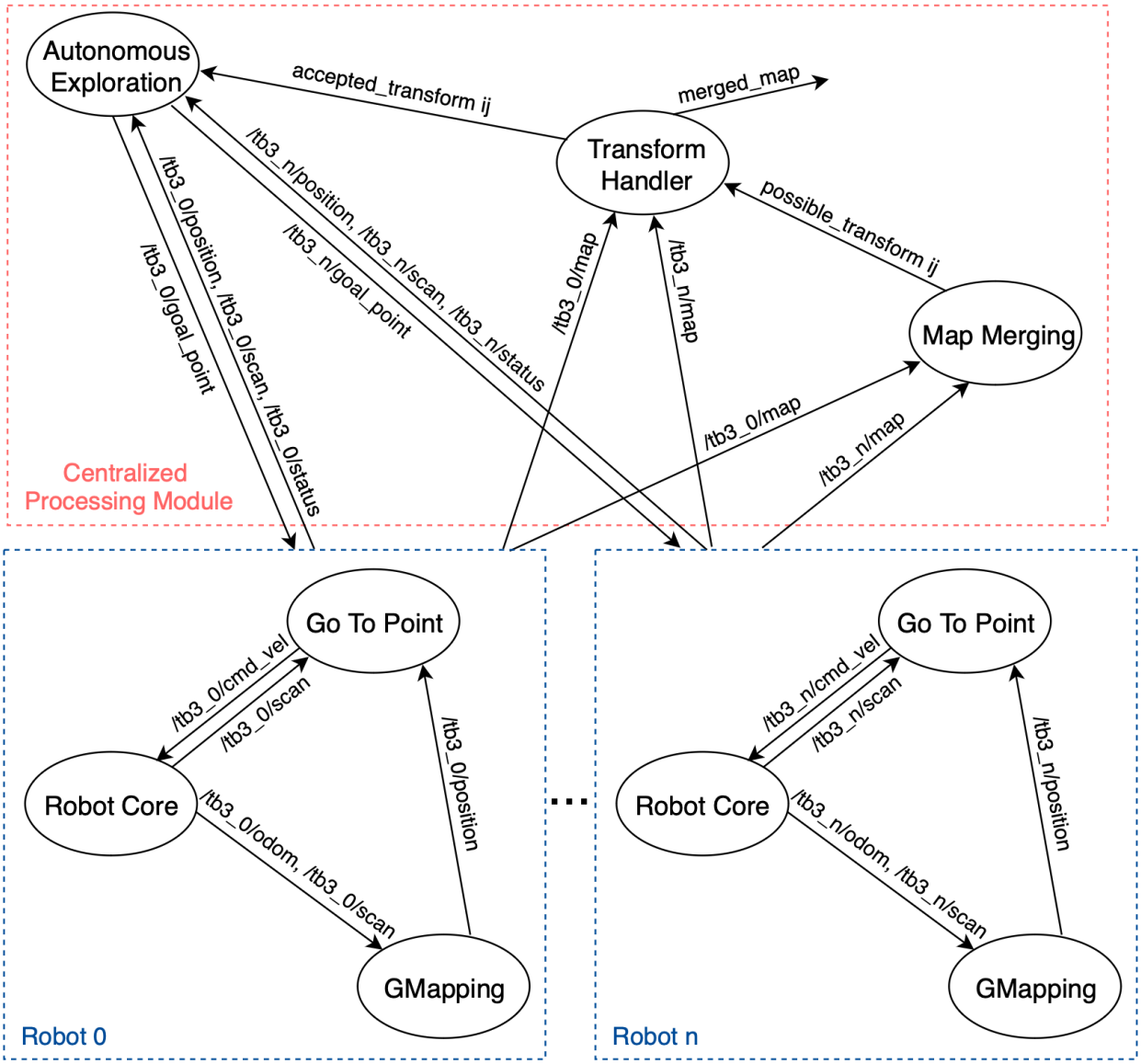


Figure 3.2: Overview of the system modules and their nodes when using multiple robots. The arrows between the nodes represent the flow of information. Only the Robot Core runs on the Raspberry Pi 4 on the TurtleBot3 robots, the rest of the nodes are executed in the central computer. There are $n+1$ Robot modules in the system, where all of those robots are sending their maps to the Map Merging node and all of them are also talking to the Autonomous Exploration node. The nodes Autonomous Exploration, Transform Handler and Map Merging are all part of the so called Centralized Processing module. The Autonomous Exploration, Transform Handler and Go To Point node has been implemented during this thesis, but the Map Merging node has been taken from [15] and the Robot Core was delivered with the TurtleBot3s. GMapping is an open-source ROS package.

3.5 Go To Point

The Go To Point node was implemented during this thesis and acts as a translation layer between the Autonomous Exploration node and the Robot Core. Go To Point takes a position as input and will

move the robot to that position by sending velocity commands. Firstly, the robot will rotate until it is facing the goal position, then it will smoothly accelerate to a set max velocity and when it gets closer to the goal it will decelerate until that goal position is reached. There are two parameters that are tunable for the user in this script.

- **max_ang_vel**: The maximum angular velocity the robot can achieve while rotating.
- **max_lin_vel**: The maximum linear velocity the robot can achieve while moving.

Additionally the Go To Point node detects collisions, see Section 3.6.6.

3.6 Autonomous Exploration

The Autonomous Exploration node is the main part of this thesis, and handles the exploration algorithm and the collaborative aspects. As stated before, Frontier Exploration Planning is the most common and widely used exploration algorithm [5]. The popular alternative to frontier exploration is the RRT based variants such as the RH-NBVP introduced in [57] where it is used for 3D exploration. However, while the RH-NBVP algorithm excels at fully exploring a smaller area with all its details, it may miss entire sections of the global map [9]. Since this thesis places a lot of value in fully exploring the environment, RH-NBVP will not be considered for this thesis. Although it is claimed to be more suitable for exploration in 3D environments and is one of the key works introduced in the last couple of years.

Frontier exploration algorithms are used with two variants, where the first one is a map-based frontier exploration algorithm used in [54], [55] and [8], and the second is a node-based frontier exploration algorithm used in [20], [6] and [11]. Since a node-based algorithm has built in navigation functionality by using the nodes themselves, additional navigation algorithms do not have to be utilized which is seen as an advantage. Map-based algorithms on the other hand require consistent maps to work well for navigation purposes, whereas the node-based approach produces a more general strategy and algorithm for autonomous exploration.

The autonomous exploration algorithm implemented in this thesis is based on the work by [20], [6] and [11]. These works aim to explore the environment as fast as possible. The goal of this thesis is to efficiently produce a complete map of the environment. The map is considered complete when the exploration algorithm has covered the whole environment. However, a key difference between the work in this thesis and the works it is inspired by is that all three of these works assume that the initial positions of all of the robots are known. In such systems, it is not necessary to deploy any sort of map-merging algorithms to find the relative positions of the robots, and the exploration problem is significantly simplified. In this thesis the initial positions of the robots are considered unknown to the system, which requires the previously mentioned map merging to work well. The premise is that several robots are exploring in the same area, but their relative positions are not known beforehand in the system. This requires the system to be able to handle independent exploration for each robot. When these relative positions are found the system can transition into a more collaborative state specifically for the robots that acquired these relative positions. Therefore the system's foundation is that each robot can single-handedly perform autonomous exploration, and collaborating seeks to improve the efficiency of this process.

The result of this is that the system is required to handle the approaches that a robot with no knowledge of other robots in the system will take to explore. It should manage this while adapting to allow for collaborative exploration to take place when robots have found their relative positions. This section will therefore explain the foundation that each robot operates with, how the robots can transition to a collaborative state, and the strategies employed when the robots have successfully begun cooperating.

3.6.1 Graph Structure

The robots will navigate from frontier point to frontier point, iteratively exploring the unknown environment. Once a frontier point is visited it will become a so called information node. All of these points are virtual ones in this thesis, and are stored in a graph data structure. The graph is defined as $\mathbb{I} = (\mathbb{V}, \mathbb{E})$, and $\mathbb{V} = \{v_1, v_2, v_3, ..., v_n\}$ is the set of n vertices. Each vertex contains a position in the 2-dimensional map and a value indicating whether it has been visited or not, $v_i = (x, y, f(i))$ where $f(i) \in \{0, 1\}$ and $i \in \{1, 2, 3, ..., n\}$. A vertex that has not been visited is a frontier point, and thus $f(i) = 0$. The edges represent undirected paths in the environment that are unoccupied and traversable for the robot. By definition each new frontier point will have at least one edge connecting it to the rest of the graph. Thus, the graph will be a connected undirected graph, where the weight of a given edge $e_{i,j}$ is the distance between vertex v_i and v_j, see Figure 3.3. The robots local coordinate system x_r, y_r will be used throughout the rest of this Section.

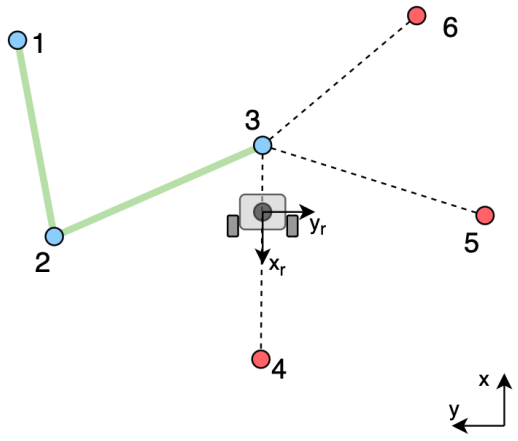


Figure 3.3: The connected undirected graph. The vertices are enumerated, where blue vertices are visited vertices and have become information nodes and red vertices are frontier points. Dashed lines represent edges between the vertices. The green lines are the trajectory the robot has traveled along the edges of the graph. A simplified model of the TurtleBot3 can be found in the middle, with a square body, circular lidar sensor and two wheels on the side. The robot is facing south in the picture where x_r, y_r is the robots local coordinate system, and x, y is the global coordinate system.

The correlation from robot to graph will be one to one if that robot is exploring by itself, and many to one if there are several robots collaborating. Meaning that several collaborating robots use the same graph.

3.6.2 Frontier Point Extraction

At each new frontier point the robots arrive at they will use their lidar sensors to extract new frontier points. The radius of the frontier extraction zone is determined by a tunable parameter, called exploration radius (*ex_radius*), which needs to be less than the lidar range. The other tunable parameter in this algorithm is called minimum explore width (*min_explore_width*), which marks the tightest spaces the robot can traverse and should thus be larger than the width of the robot. The overall strategy to extract frontier points can be divided into two steps.

1. **Extract possible frontier points**
   Based solely on the reading from the lidar sensor and not taking into account other nodes in the graph, open spaces surrounding the current position are extracted and possible frontier points are deduced. In spaces with obstacles the frontier points need to be at least half of the minimum explore width parameter away from the obstacle, to guarantee that the robots can traverse the edge to that point.

2. **Add frontier points to the graph**
   The possible frontier points from above are compared with the rest of the vertices in the graph. If the possible frontier point is covered by another vertex, that area has already been explored and the point will not be added to the graph.

The algorithm handling the first step and extracting possible frontier points can be found in Appendix A.1. The algorithm will look to add possible frontier points between obstacles if there is a wide enough path to that frontier point. Additional possible frontier points will also be added in open spaces. In a completely open space three possible frontier points will be added evenly around the robot's position. A visualization of the algorithm from Appendix A.1 can be seen in Figure 3.4.

It is not guaranteed that all the possible frontier points should be converted to actual frontier points in the graph. The second part of the frontier extraction process only adds the possible frontier points that lie in an unexplored area, see Appendix A.2. If any of the possible frontier points lie within the exploration radius of any other vertex in the graph and that vertex is reachable, that point should already have covered from the other vertex and that possible frontier point is not added to the graph. A vertex is reachable from the current position if a straight line, with width equal to the minimum explore width, between the two vertices considered can be drawn in the environment and that line is not interrupted by obstacles. A visualization of the algorithm can be seen in Figure 3.5.

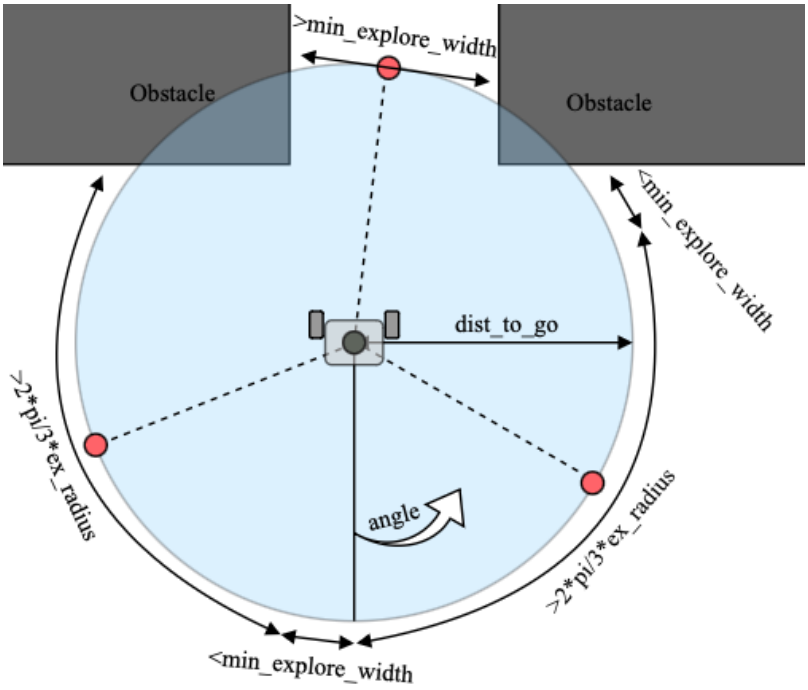


Figure 3.4: The frontier extraction algorithm from Appendix A.1. The blue circle represents the area of the exploration radius parameter, and does not correlate with the actual lidar sensor readings. Red dots are representations of the possible frontier points that are returned from the function. Positive angels are defined counter clockwise. The algorithm starts iterating through the lidar readings from where the "angle" arrow is defined in the local coordinate system of the robot.

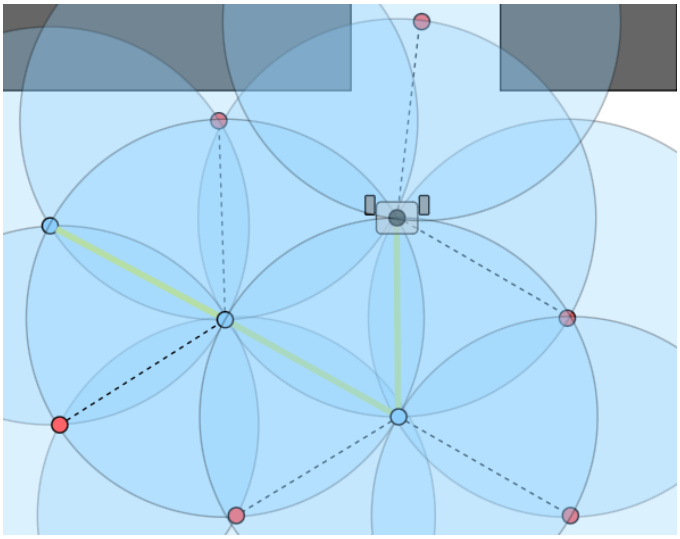


Figure 3.5: The frontier algorithm from Appendix A.2. The TurtleBot3 is located at the same spot as in Figure 3.4. The blue circles represent the area of the exploration radius parameter from all vertices, both previously visited vertices and other frontier points. Dashed lines are edges in the graph, while the green lines are the path the robot has traveled along some of the graph's edges. The red dots are the frontier points. The two frontier points with dashed lines to the robot's current position represent frontier points that have just been added to the graph.

3.6.3 Update Graph Edges

Whenever a robot arrives to a vertex, the system will try to add more edges to the graph. All the vertices in the graph are examined and if any vertex is reachable from the current position the robot arrived at, a new edge will be added in the graph between those vertices given that no such edge already exists. These edges help the robots efficiently navigate the graph and shortens the exploration time, see Appendix A.3.

3.6.4 Graph Fusion

When the robots start collaborating the two graphs of the two robots will be combined to a single graph. One of the robots will be considered to be the master robot and the other robot's graph will be fused into the master robot's graph. The calculation of the transform from the other robot's coordinate system to the master robot's coordinate system is handled in other nodes, and is explained in Section 3.7 and 3.8. The process of combining the graphs is shown in Figure 3.6, where two robots' graphs are fused.

Fusing the graphs allows the use of one large merged graph as an object which both robots involved in the fusion can use for navigation. However, the fused graph is disconnected in the early stages after the merge. Using a disconnected graph can cause problems for the exploration algorithm when new frontier points are selected as goals. This is remedied by adding an extra edge to the graph between the two vertices that are closest in distance to each other. The two vertices that become connected by the new edge are originally from different graphs, thus making sure that the fused graph becomes connected. This is showcased in Figure 3.6. Since the robots will have explored the same area to some extent in order to collaborate, see Section 3.7, there will exist an overlap of the graphs. Hence, edges will quickly be added to the fused graph from the update graph edges algorithm, which the robots can navigate with, in order to make sure that it is an advantage to share graphs between robots. The purpose of the extra edge mentioned is rather to make sure that the graph operations do not fail immediately after the fusion has occurred. Instead of having any significance for the exploration or navigation.

The graph fusion may cause some frontier points from one of the robots to lie in areas already explored by the other robot. These points were set prior to the merging of their maps and graphs. This is an issue, since for the robots to explore efficiently they should avoid exploring areas that have already been visited. The points that cause this issue should therefore not be explored provided that they fulfill the conditions that their position is within 0.95 of the exploration radius of at least two other visited vertices, or within the minimum explore width of one other visited vertex.

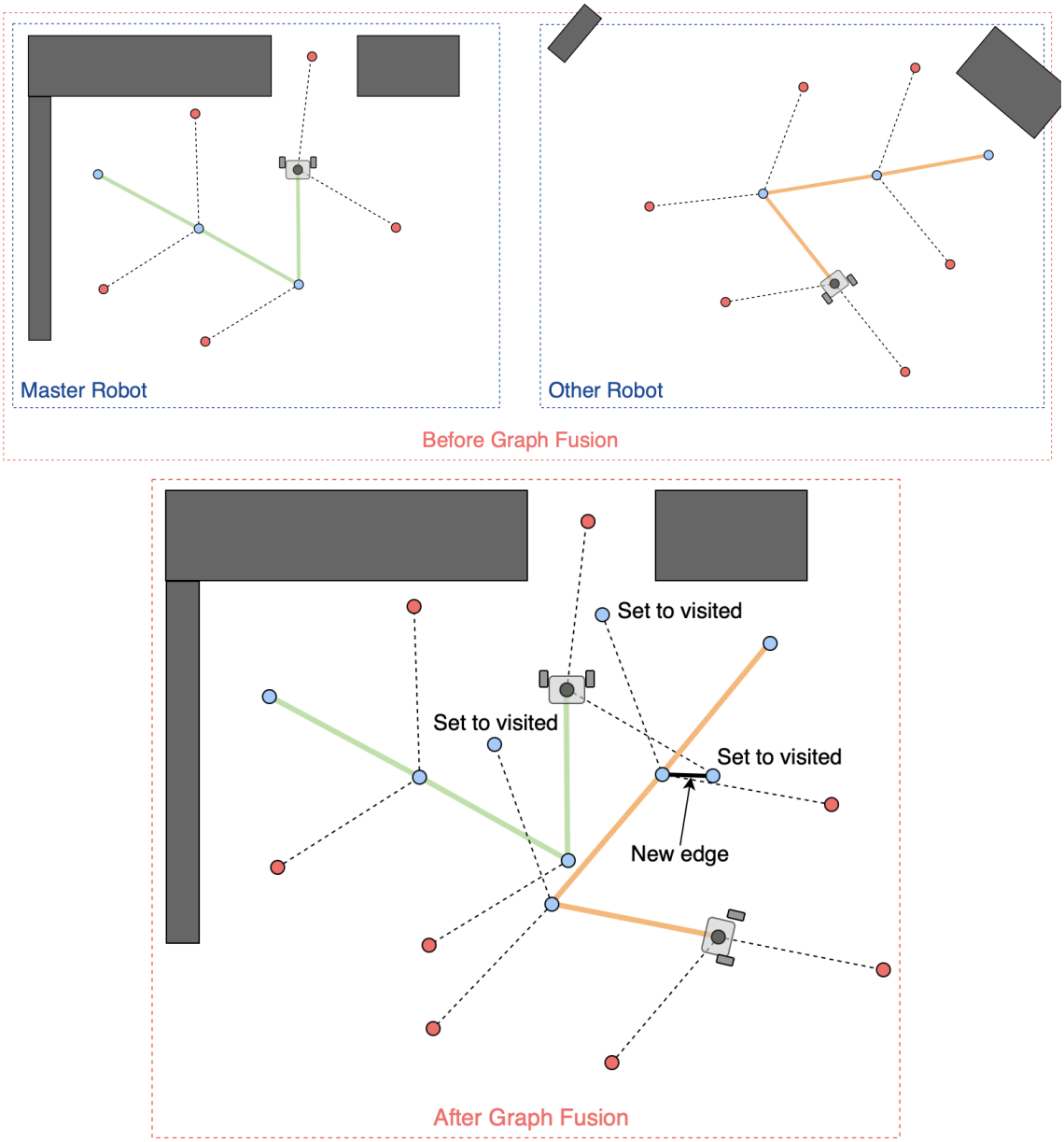


Figure 3.6: Two robots are starting to collaborate which requires their graphs to be fused. The green path is the master robot's trajectory and the orange path is the other robot's trajectory. The other robot's vertices will have their positions transferred to the master robot's graph's coordinate system in order to fuse the graphs. After the fusion, frontier points that lie in already visited areas are set to visited, since those areas are already explored. An extra edge is also added between the fused graphs to avoid having a disconnected graph. Note that the autonomous exploration algorithm itself will add more edges between the graphs as the exploration continues, and all edges in the graph are not visualized.

3.6.5 Frontier Selection and Navigation

The final step of the exploration algorithm is how to determine which frontier point to explore and then navigate to. All the current frontier points in the graph are examined and a cost value

is calculated. There are two cases for this cost value, one is if the robot is exploring on its own and the other is if multiple robots are collaborating. The single exploration cost value is based on the frontier point's distance from the current position and the distance from the starting position of the robot. Therefore, equation (3.1) is calculated for every frontier point in the graph, and the frontier point with the lowest cost value is chosen as the next goal. The cost value for vertex v_i is c_i, and $d(v_i, v_j)$ is the euclidean distance between vertex v_i and v_j. λ is a tunable parameter such that $0 \leq \lambda \leq 1$. If $\lambda = 1$, the exploration will behave similar to a depth first search in the environment, and if $\lambda = 0$, it will behave similar to breadth first search. A value of 0.85 was used throughout this thesis, since a depth first search in the environment should result in less going back and fourth and wasting time in the early phases of the exploration.

$$c_i = \lambda \cdot d(v_{curr}, v_i) + (1 - \lambda) \cdot d(v_{start}, v_i) \tag{3.1}$$

The robot navigates to the chosen frontier point according to the edges in the graph. Thus, if there is an unobstructed path from the current vertex to the frontier point the robot will go directly to the goal, otherwise it carries out a breadth first search in the graph to find a feasible path to the frontier point. Only the first point in the path is used for navigation in order to never go past any frontier points that lie closer than the chosen frontier goal. When the robot arrives at the next vertex the process will be repeated. A visualization of the algorithm can be seen in Figure 3.7.

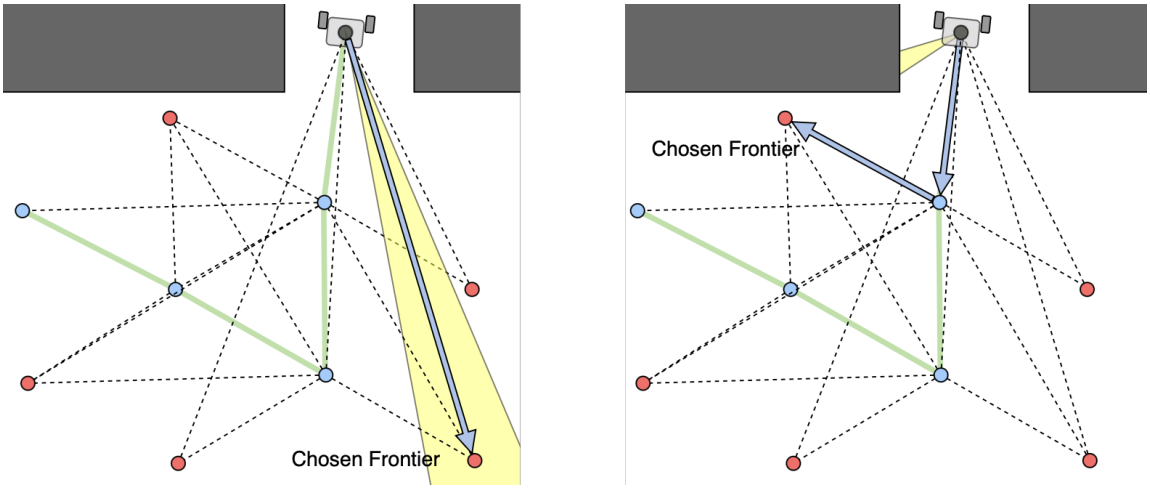


Figure 3.7: The frontier selection and navigation algorithm from Appendix A.4. Green points are visited vertices, red points are frontier points. The green lines are the path the robot has taken, and together with dashed lines represent the edges in the graph. The yellow sector represents a subset of lidar measurements. To the left, the chosen frontier point is visible from the current position, and the robot can go directly to it. To the right, the chosen frontier point is not visible from the current position, and the planned path is shown with blue arrows.

In a multi-robot setting the cost function behaves slightly differently to promote some different kinds of behaviors in order to ensure that the robots collaborate more efficiently and avoid interrupting each other. Selecting the same goal can be something that happens when multiple robots navigate with the same shared data. In order to make use of the fact that there are multiple robots exploring, selecting the same goal is not usually encouraged and as such the robots will choose different frontier points to explore if possible. However, in the case when there is only one frontier point present in the system,

having multiple robots move towards the same only possible point is desirable. Since beyond that point there may be more areas that are necessary to explore.

In order to aid exploring with multiple robots on many borders between known and unknown space the cost function used by the robots differs from when robots are independently exploring. The multi-robot cost function makes them more likely to select goals that are further apart from the other robots' positions. The multi-robot cost function can be written as equation (3.2), where ε is a small number guaranteeing that there exists no cases where the function will divide by zero. The other robots the current robot is collaborating with is defined by the set $\mathbb{R}$, which contains the indices of those robots. For example if robot 1 is collaborating with robot 2 and 3, $\mathbb{R} = \{2,3\}$. The vertex $v_{goal}^{r_j}$, where $r_j \in \mathbb{R}$, is the current goal point of robot r_j.

$$c_i = \sum_{r_j \in \mathbb{R}} \frac{d(v_{curr}, v_i)}{d(v_{goal}^{r_j}, v_i) + \varepsilon} \tag{3.2}$$

In essence the function makes use of the other collaborating robot's frontier goal, and uses that to determine whether a potential goal has a high cost value or not. The robot in question then selects the goal with the lowest cost value out of the possible frontier points, and navigates to it. This makes the robots spread out across the environment if possible but they will still travel towards the same areas if that is the only option available to them, see Figure 3.8.

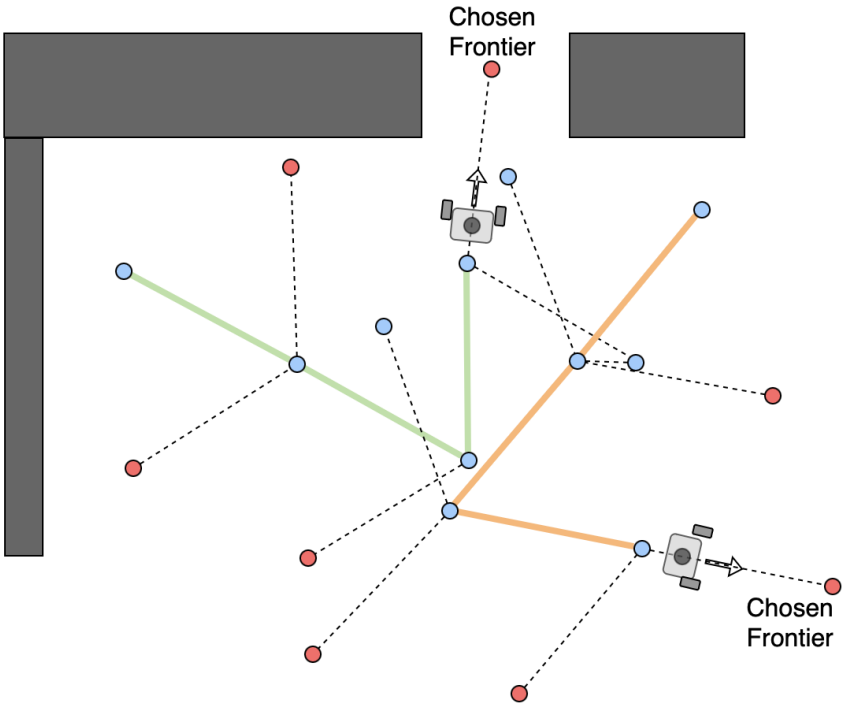


Figure 3.8: Example of the collaborative cost function, after the graphs have been fused. The robots will choose frontier points that makes them spread out in the environment so that different frontiers are explored at the same time. The arrow in front of the robots represents their current velocity.

The algorithm handling frontier selection and navigation can be found in Appendix A.4. When

multiple robots are collaborating there are some cases when one robot is standing in the way of another robot during frontier selection and navigation. The robot that becomes blocked off from its frontier point will in such cases pause its exploration to allow the other robot to move away, to not cause any unnecessary collisions between the robots. This behavior is separate from the collision avoidance system, since it should help preventing the robots moving towards the same area.

3.6.6 Collision Avoidance

There is a very simple collision avoidance system in place that simply interrupts the robot if there is an obstacle in front of it, implemented in the node Go To Point. The collision avoidance algorithm checks if there are any obstacles within a cone from the robot's position. The radius of the cone, r_c, and the angle θ_c, are calculated in equation (3.3), where the linear velocity of the robot is u, and the minimum explore width is w.

$$\theta_c = \frac{0.9 \cdot w}{r_c}, \quad r_c = w + \frac{3}{2}u \tag{3.3}$$

Therefore, the cone grows larger when the robot has a bigger velocity, in order to give more time for the robot to slow down. When the collision avoidance system is triggered due to the static environment the current edge will be removed, and if that leads to a frontier point that has no edges attached to it that frontier point will be removed as well.

The collaborative cost function will promote the robots to explore different frontiers at the same time. However, it is still possible for the collaborating robots to move towards the same goal, for example if there only exists one frontier point in the graph. Thus, a robot's collision detection might be triggered by another collaborating robot. The system will then proceed to not remove that frontier point or edge, which it otherwise normally does if an obstacle was blocking that pathway. Instead the interrupted robot waits and moves back slightly in order to give the other robot the opportunity to move out of the way, see Figure 3.9.

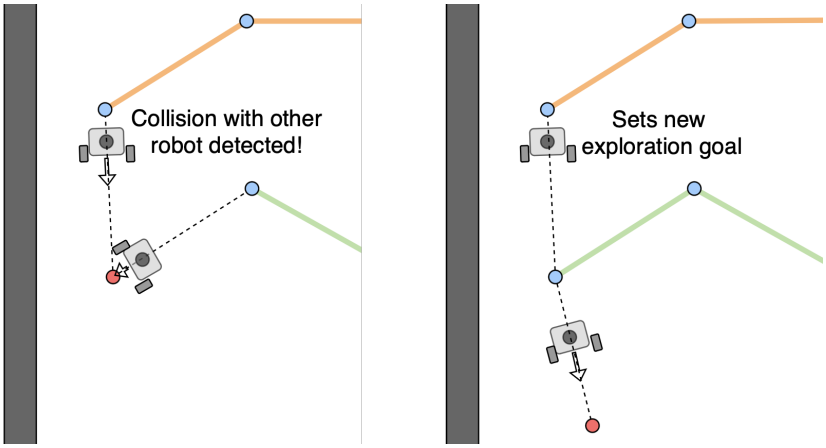


Figure 3.9: Example of the collaborative collision detection. In the left picture the top robot detects the other robot in front of its path. It will then proceed to wait which allows the other robot to move out of the area, see the right picture. Finally the interrupted robot will choose a new frontier goal. The arrow in front of the robots represents their current velocity.

The functionality above and the interruption case can be found in Appendix A.5. This function acts as the state machine for the robots and is divided into two cases, if the robot successfully reached its goal or if the collision avoidance system triggered an interruption during the process.

3.6.7 Map Coverage

The node-based frontier exploration algorithm implemented in this thesis aims to cover all unoccupied space with vertices. In a real world scenario there will be a significant amount of accumulating errors in the pipeline, most notably in the lidar readings where the measurement error increases with the distance, and the fact that the position calculated by GMapping is an estimation. This will impact the placement of the frontier points and makes it hard to draw any theoretical conclusions about the algorithms map coverage. However, by looking at Figure 3.5 again, in open spaces the frontier points will continue to form the pattern as shown in the figure, thus any point in open space will be covered by at least one blue circle.

In more challenging environments there is is a simple limitation in the exploration algorithm where spaces that are too tight are not traversable, due to the width of the robot. As mentioned the tunable parameter minimum explore width corresponds to this width and marks the tightest spaces where the robot will be able to explore. Obstacles with sharp corners also pose a problem. Consider the worst case scenario in Figure 3.10, where the algorithm is unable to see into the area between the obstacles and thus will not place a frontier point there. However, with a low enough value of the exploration radius parameter, any traversable path will eventually be spotted by the robot and it will be able to explore the whole area. Therefore the tuning of the exploration radius parameter is vital for the algorithm to explore the whole environment, where larger values will enable quicker exploration times, but increases the risk of the algorithm missing certain parts of the space.

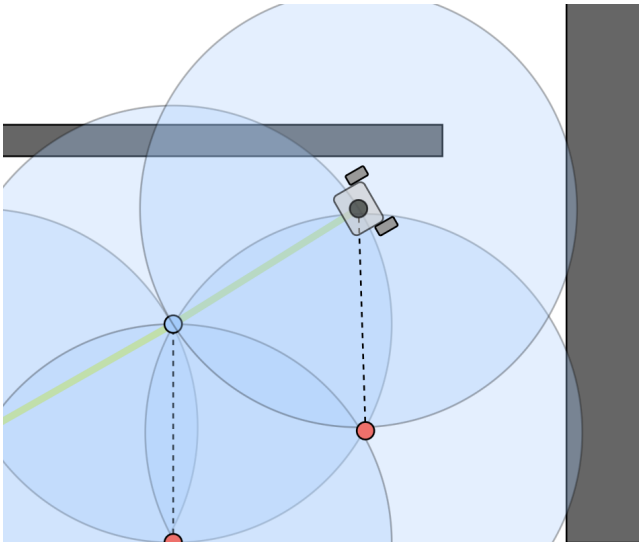


Figure 3.10: Example of when the node-based frontier exploration algorithm fails to explore a certain area of the map. The robot is unable to see into the narrow passageway in the upper right corner and therefore will not place a frontier point in that area.

3.7 Map Merging

The Map Merging node is an integral part of the Centralized Processing module. The thesis focuses on autonomous exploration where initial positions of robots are unknown, thus they are required to somehow determine where they are relative to each other in the environment in order to collaborate. In this case the system focuses on determining this by utilizing the occupancy maps the robots create. The Map Merging node initiates the collaborative exploration by determining how the robots' collected data correlate.

This thesis used the ROS package called *multirobot_map_merge* from [15], which is a feature based map merging algorithm built on computer vision concepts. As input it takes all the individual robots' maps, and outputs transformations between them as well as a merged map. Only the transforms are further used in this thesis. Since the TurtleBot3's move in 2D space and the map output from GMapping is a 2D map, the ground-truth transform consists of a translation in x and y, and a rotation around z, given the left-hand coordinate system used in ROS where the robot moves in the $x-y$ plane. There are several tunable parameters in this package, where the most important is called estimation confidence. With each transformation calculated by the package there is a confidence describing how good the transformation between the maps are, which lies in the range $[0, \frac{10}{3}]$. If that given confidence is lower than the estimation confidence parameter, that transform will be ignored. Thus, it is essentially a threshold that determines how accurate the merging needs to be for the package to consider it a valid transform.

An estimation confidence threshold of 1.2 was used throughout this thesis, since that is relatively high and thus the possible transforms returned by the Map Merging node will have a large probability of being accurate.

Although this package remains mainly unaltered in the project, the overall structure is still important

to understand since it influences the rest of the system implemented in this thesis. With some simplifications the general structure of the *multirobot_map_merge* package can be broken down into the following steps.

1. **Read maps from GMapping**: All the robots' current maps are saved.
2. **Convert maps to pictures**: The occupancy grids are converted to pictures. Each cell in an occupancy grid contains a value depending on if it is occupied, free or unknown.
3. **Extract image features from the map pictures**: With OpenCV, an open-source computer vision library [64], image features are extracted from each picture. This operation returns a set of points in the pictures coordinate frame, and for each point a matching feature vector.
4. **Extract matching features**: For all possible pairs of map pictures, the corresponding features are matched with OpenCV to produce a list of features that are present in both of the maps.
5. **Calculate the affine transform with RANSAC**: Using OpenCV, for each image pair, the Random Sample Consensus (RANSAC) algorithm is applied to the matches of the image features to estimate the affine transform between them. This affine transform contains a rotation, 2D-translation and a scale factor. RANSAC randomly samples a subset of matching image points that are used to estimate the model, which in this case is an affine transformation. Utilizing that model the rest of the image points are compared to produce a number of how many of them overlap, called inliers. The amount of inliers divided by the total amount of matching image points provides a value that indicates the performance of the model. A value close to zero indicates a very poor model, whereas a value close to one indicates an excellent model, where most of the matching image points are overlapping given that model. This value is then translated to the confidence value for each possible pair.
6. **Choose a master robot**: The robot with the most amount of affine transforms that gave a confidence value above the estimation confidence parameter is chosen as the master robot.
7. **Pair affine transforms in master robot's frame**: The affine transforms from the master robot's frame to the other robots can be found by looking at a tree of the transform chain. Using this, pairwise transforms to the other robots can be established. For example, given master robot 1, and other robots 2 and 3, where transformation $1-2$, as well as $2-3$ has been found. The walk through the tree will multiply the transform $1-2$ with $2-3$, and return two affine transforms, $1-2$, and $1-3$. In other words, all the found transforms will be expressed in the master robot's coordinate frame. This means the master robot's map has a unit transform applied to it, and that robot's local coordinates are chosen to be the global coordinates.

Some general observations can be made about this algorithm. Firstly, the steps above are executed in order at a given interval, where the resulting affine transforms are not saved or reused. Since RANSAC is based on random sampling, it means the package can find a really good affine transformation with a high confidence between two robots at one instance, but in the next one it finds a significantly worse transform with a low confidence value. Additionally, the master robot can change at any time. Secondly, the affine transform is a best effort, there are no guarantees that the output is correct. Thirdly, the affine transformation output is not forced to consist of a pure rotation and 2D-translation. It nearly always contains a small incorrect scale factor as well. Despite

these disadvantages there are several advantages gained by using this package as well. The package is multithreaded and fast to run and allows maps to be merged with a relatively small amount of overlap, theoretically speeding up the collaborative exploration.

3.8 Transform Handler

To achieve the desired behavior when a transform is accepted and from what criterion, the Transform Handler node was added in between the Map Merging node and the Autonomous Exploration node. The function of the Transform Handler node is to listen to all the possible transforms returned by the Map Merging node, refine the possible transforms and find the best one in a small solution space. This was done to find more accurate transforms faster. The refined transform is only accepted if a specific metric is high enough, indicating that the transform produces a good match between the maps. A transform only has to be accepted once in order for the robots to start collaborating, since this means that the transform between one robot's coordinate system and the master robot's coordinate system has been found and the robots know each others positions. The transform handler will send the first accepted transform to the Autonomous Exploration node to initiate the collaboration. Although after the first transform is accepted, the transform handler package still continues to listen to the updated versions of the same transforms, since transforms with even larger similarity indices will provide a better estimation of the other robot's position. If such a transform is found the relative positions of the robots will be updated.

The similarity index is used to determine when a transform produces a good match between the maps and is inspired from [33]. In order to calculate the similarity index the possible transforms need to be modified first. The first step in the transform handler is to calculate the scale, rotation and translation from the affine transform. The scale cannot be converted to a rotation or a translation, and additionally the theoretically correct transform does not contain a scale factor since the environment never changes size and all the robots use the same map resolution. Hence, the scale factor is ignored. Define the master robot's map as m_1, and the other robot's map as m_2. Each map contains a number of cells with an occupancy value and a position p. The transformation, $\mathbf{T}$ to be applied to m_2 consists of a rotation and a translation, see equation (3.4).

$$\mathbf{T} = [\mathbf{R}\ \mathbf{T}] = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Where θ is the angle around z, t_x is the translation in x and t_y is the translation in y. The transform $\mathbf{T}$ is applied to the homogeneous coordinates of all the cells in m_2, such that $m_2' = \mathbf{T}m_2$. How similar the maps are is defined by the agreement term, agr, see equation (3.5).

$$agr(m_1, m_2) = \#\{\forall p \in m_1 \mid (m_1(p) = m_2(p)) \wedge (m_1(p) \neq -1) \wedge (m_2(p) \neq -1)\} \tag{3.5}$$

How dissimilar the maps are is defined by the disagreement term, dis, see equation (3.6).

$$dis(m_1, m_2) = \#\{\forall p \in m_1 \mid (m_1(p) \neq m_2(p)) \wedge (m_1(p) \neq -1) \wedge (m_2(p) \neq -1)\} \tag{3.6}$$

Where # is the cardinality of the set and a cell value of -1 indicates unknown space. Finally the similarity index V of the master robot's map and the other robot's transformed map can be calculated, see equation (3.7).

$$\mathbf{V}(m_1, m_2') = \frac{agr(m_1, m_2')}{agr(m_1, m_2') + dis(m_1, m_2')} \tag{3.7}$$

V will thus be a number between zero and one, where zero indicates no similarities between the maps and one implies that it is a perfect match. Essentially it becomes a measure indicative of how equal the maps are in the area where the maps overlap, see Appendix A.6. The similarity index is used to accept transforms, where the transform is only accepted if the similarity index is larger than 0.95. This threshold was deduced by testing the algorithm and visually confirming the accuracy of the accepted transform. Additionally the similarity index is used to find the optimal transform in a small solution space, 5° in rotation and 1 meter in translation. The optimum is found by brute force calculations around the possible transform returned by the map merging node. This means that the disregarded scale factor in the affine transform will not have any significant impact on the accuracy of the accepted transform.

Additionally, the transform handler also merges the maps with all the accepted transforms and produces a merged map in the master robot's coordinate system, see Appendix A.7. For each position in the maps, the merged map cell value of that position is set to the max value present in that position, see equation (3.8).

$$M(p) = \max(m_1(p), m_2'(p)) \tag{3.8}$$

Where M is the merged map. This means that if any of the collaborating robots' SLAM algorithm decides that there is a wall in a certain space, the merged map will contain that wall. Only if all the collaborating robots decide that a known space is free, the merged map of that space will be represented as free.

3.9 Calculations of Statistics

Some of the data gained from testing the system has to be calculated during the autonomous exploration. In addition to this, some details regarding the trajectory lengths need to be clarified. The trajectory length of each robot is calculated by using the vertices in the graph and summing the total distance of the edges the robot travels along. This means that when robots drift slightly or are affected by similar issues the added length the robot will travel is not added to the total length calculated. Regardless, it is still a measurement which provides a valuable indication of the system's performance. Most of the measurements from the exploration were normalized with the results gained from one robot in order to enable comparison between several different environments and robots.

3.10 Test Environments

The test environments used were some of the standard indoor environments at Ericsson. By starting with a small and challenging environment the system's parameters can be refined and tenacious enough to handle easier environments in terms of tight spaces and exploration. A smaller test environment also gave the advantage of speed, which meant that more variations and values of the parameters were feasible to test. This smaller environment was called Environment A, and was an office area with multiple office desks and small meeting rooms. Therefore there were many tight spaces the exploration was challenged with and multiple rooms that had to be entered. However, the space had to be customized with some obstacles in order to keep it enclosed. Thus, the environment could not be precisely the same for every test although it was set up as similarly as possible for each test with some human error, since testing spanned several days. The space can be seen in Figure 3.11.

Figure 3.11: Environment A. This area is an office space at Garaget, Ericsson Lindholmen. The chairs were removed since the feet of the chairs are below the lidar sensor readings of the robots, therefore the robots are unable to see the feet of the chairs. There are two small rooms included in this area which has a relatively tight entrances, making the exploration challenging for the algorithm.

Another environment that was used was a larger area, denoted Environment B. This could be considered an easier environment, since it was an open space with no separate rooms for the robots to explore. A picture of this area is shown in Figure 3.12, showing a recreational area at Ericsson.

Figure 3.12: Environment B. This area is called the Recreation Hub at Ericsson Lindholmen.

The last environment, Environment C, was a conference room made for 14 people. This meant that it was an enclosed space. It contained a large table in the middle with two large cylindrical legs. In order for the area to be more interesting to explore an extra wall was added by combining some obstacles. This indoor area is shown in Figure 3.13. The aim was that the area was a middle ground in difficulty and size compared to environment A and B.

Figure 3.13: Environment C. A conference room at Ericsson Lindholmen. The chairs were removed for the runs in this environment.

4 Results

The results of the multi-robot collaboration will be presented in this section. It will be explained how the exploration radius parameter was established and if the robots should start distant or close to each other. It also summarizes the parameters used and shows performance of the collaborative autonomous exploration system for a range of one to four robots in three different environments.

4.1 Determining Exploration Radius

The first environment the robots explored was Environment A, see Figure 3.11, with the purpose of establishing the exploration radius used in the autonomous exploration algorithm. The exploration radius has a large influence on the node-based exploration and navigation for robots exploring independently as well as collaboratively, and is heavily dependent on the environment. Eleven different values of the exploration radius parameter were tested, and three runs were executed for each value with a single robot. The averages of the most important statistics were calculated and can be found in Figure 4.1.

By observing these results some general trends come to light. Firstly, there are some stochastic parts in the system resulting in a relatively large amount of variance between the runs. Secondly, with a smaller value of the exploration radius the robot will take more time to explore the environment, with a higher node density and a longer trajectory. However, the average map area follows another trend. With larger values it is smaller, but for values lower than 1.5 the average map area was relatively stable at around 28 square meters. That is because the robots are unable to explore most of the environment with large values of the exploration radius, resulting in a partial map that is significantly smaller than the actual environment. To exemplify the effects of the exploration radius three pictures are presented in Figure 4.2, showing the final map after the exploration has finished.

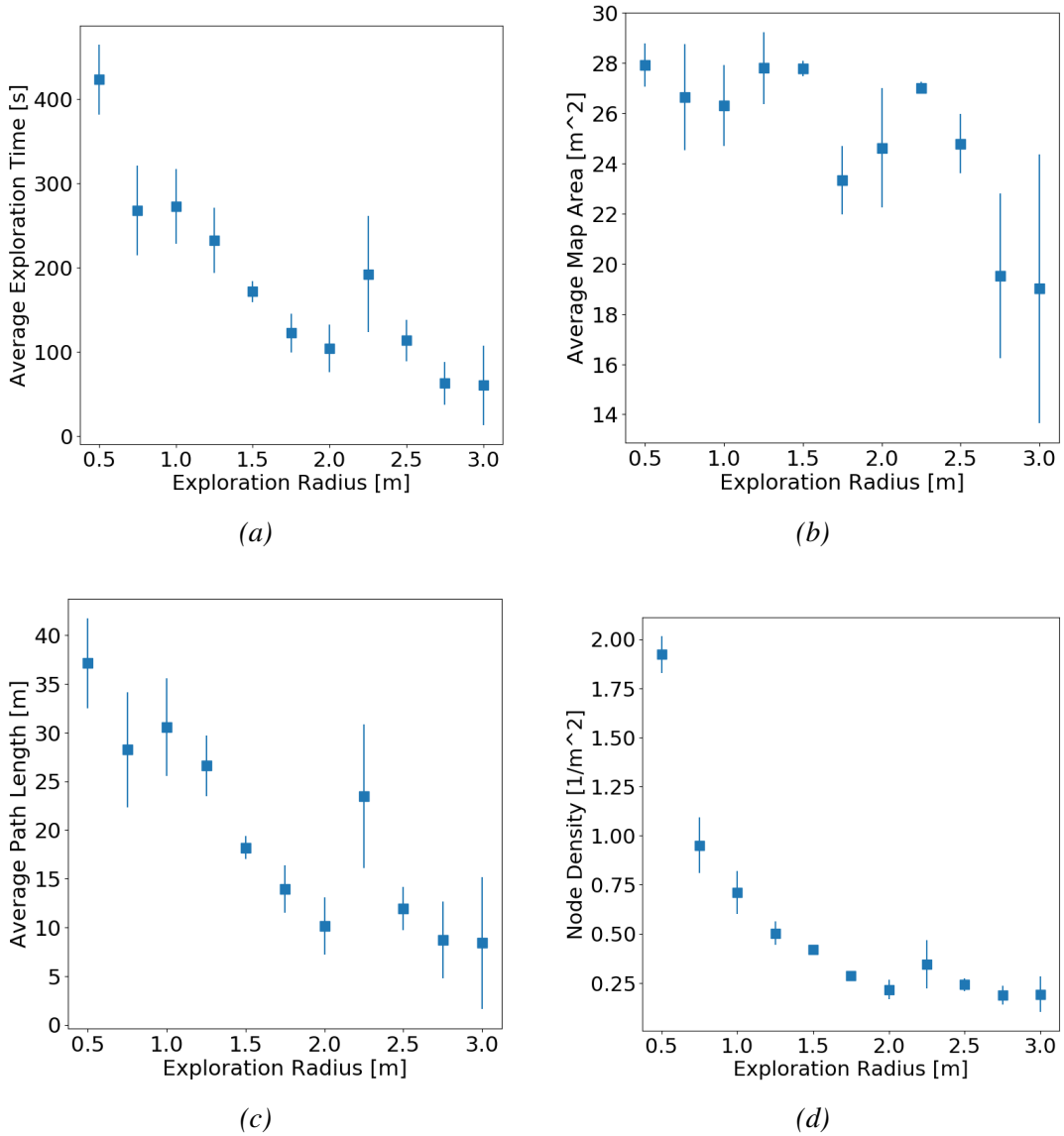


Figure 4.1: Results from the exploration runs with a single robot in Environment A, utilizing different values of the exploration radius parameter. Three runs were recorded for each value, and the averages were visualized for each respective statistic together with the standard deviation. The standard deviation of three samples is not a representative statistic, but in this case it is only used to give an idea of how different the results can be from run to run. The node density is calculated from the average number of nodes divided by the average map area for that value of exploration radius.

(a) Exploration Radius: 2.75 *(b) Exploration Radius:* 1.75 *(c) Exploration Radius:* 0.75

Figure 4.2: Three examples from Environment A showing the resulting map of the exploration in RViz, with descending values of the exploration radius from left to right. Blue nodes are visited frontier points together with the green robot trajectory.

The first notable difference is that the robot did not explore the whole environment in Figure 4.2a, when the exploration radius was large. Comparatively in Figure 4.2b, the robot managed to visit all rooms and see the whole map, however this was not a reliable outcome in this environment with many narrow passages. On the other hand, in Figure 4.2c, the robot has a low exploration radius and therefore visits many more frontier points and corners of the room. At this value it reliably explores the whole environment.

The critical point for the exploration radius parameter where the robot was able to reliably explore the whole environment, was when its value was around 1.5 or lower. Specifically at a value of 1.5 the robot managed to explore the entire environment in all of the test runs. Whereas even for lower values of the parameter the robot sometimes did not manage to explore all of the environment. There is a trade off between reliably covering the whole environment in the exploration and doing it in a reasonable amount of time. The exploration time increases very quickly with lower values of the parameter. A value of 1.0 was seen as a good compromise, allowing the robot to reliably and fully explore the environment in an acceptable amount of time. Additionally, this environment was seen as quite difficult with many narrow sections for the robot to travel through. Setting the parameter to this value should allow the robot to explore unknown environments that are not more difficult to traverse than Environment A.

4.2 Distant or Close Starting Positions

Different initial positions of the robots were tested in Environment A. The goal of these tests were to clarify whether the robots should start close to each other or in separate ends of the area when performing the exploration. Starting close to each other allows the robots to merge much earlier and start collaborating. Whereas starting far from each other allows them to explore different areas earlier at the cost of merging much later or not at all. The exploration is considered done when all

robots have independently explored the environment or when a majority of robots have explored the environment collaboratively. A comparison between the average exploration times for ten runs for each set of different number of robots, where they either started close to or far away from each other, is shown in Figure 4.3.

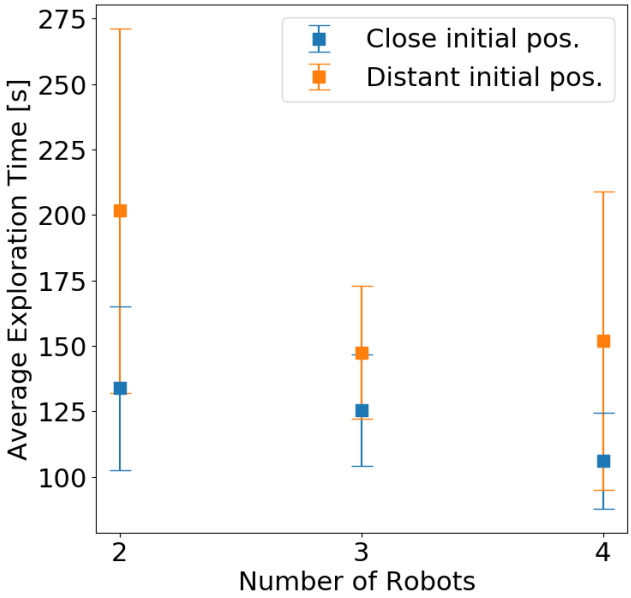


Figure 4.3: Ten exploration runs were carried out for each starting position for each number of robots in Environment A. When the robots start close to each other the average distance between them was around 1.5 *meters, and when they started distant from each other they were put in the opposite sides of the environment.*

The longer exploration times with distant initial positions most likely occur due to the fact that the system could not merge the maps of the robots, which resulted in the robots needing to explore the environment independently before the exploration was considered finished. For instance, when starting two robots far apart they only managed to merge five times out of ten, compared to when they were starting close to each other were they merged ten times out of ten. There is more variance in the exploration times when the robots are starting further apart, but there is still a clear time difference between the averages. For the rest of the runs the robots started close to each other in all three environments.

4.3 Summary of System Parameters

The important exploration parameters used in the system could not all be tuned due to time constraints. Instead they were chosen with the experience gained during the development of the system. The additional exploration parameters were minimum explore width, maximum angular velocity and maximum linear velocity. Minimum explore width was set to be roughly twice the width of the robot and the two velocities were chosen with exploration time and the safety of the robots in mind. Due to the simple nature of the collision avoidance the velocities were set so that if robots were to collide they would have no risk of damaging each other. A summary of the important parameters in the

system can be found in Table 4.1.

Table 4.1: Summary of the parameter values used in the system with brief descriptions. The parameters cost value, estimation confidence and similarity index threshold were established in Section 3.

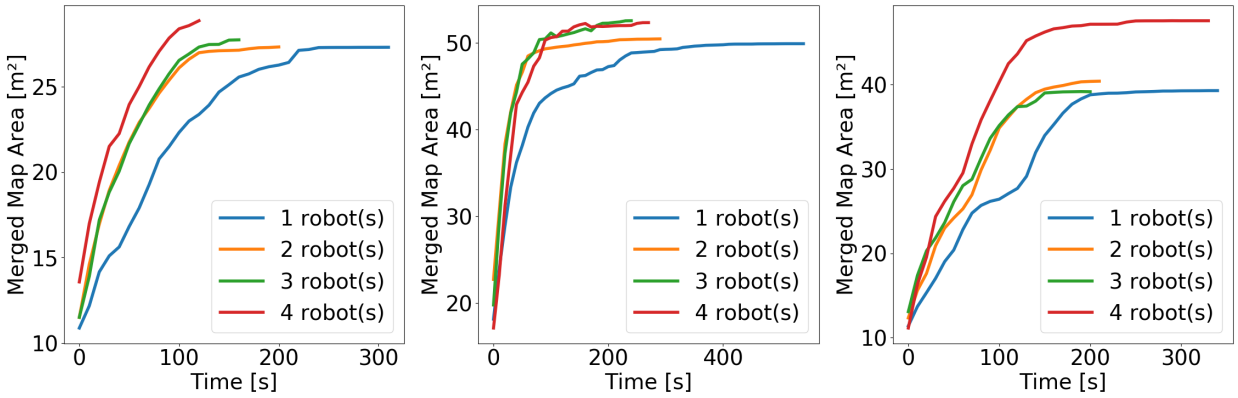
Parameter name	**Value**	**Description**
ex_radius [m]	1.0	Determines node density, due to being essential in the distance between frontier points.
min_explore_width [m]	0.35	The minimum exploration width used by the robot to infer how close to objects it can be when traversing a space.
max_ang_vel [rad/s]	2.0	The maximum speed the robot is allowed to rotate with around its z-axis.
max_lin_vel [m/s]	0.3	The maximum linear velocity the robot is allowed to use when exploring.
cost value (λ)	0.85	Determines whether the exploration should act more akin to a depth first search of breadth first search.
estimation_confidence	1.2	The ratio of confidence required for the Map Merging node to be allowed to suggest a transform.
sim_index_threshold	0.95	The threshold used to determine whether two maps overlap enough to merge and initiate collaboration.

4.4 Autonomous Exploration Performance

The results presented here were gathered during a large number of tests from several different days. In each of the three environments ten runs for each number of robots were executed. The map area covered by the robot or robots as well as the total exploration time are the main areas of interest in this project. Therefore the results highlight the performance of the model using these factors as main evaluation points, while also taking into consideration other indicators relevant for the collaboration and exploration. An example video of a test run in an environment similar to Environment C can be found here `https://www.youtube.com/watch?v=NoEtjPzyVsc`.

4.4.1 Exploration Rate

In Figure 4.4 the performance of different number of robots collaboratively exploring the three different environments can be seen. The merged map area generally grows more quickly as more robots are used, indicating a faster exploration rate.



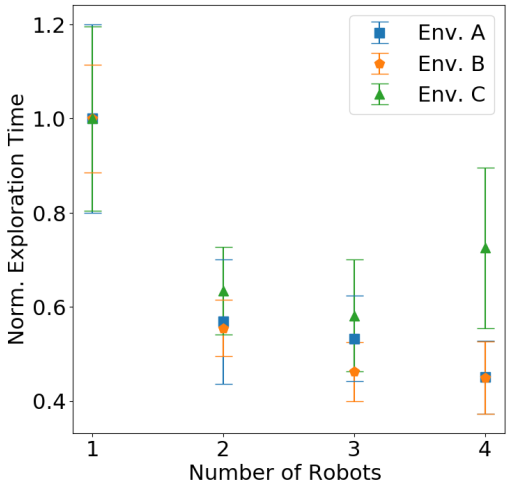
(a) Environment A *(b) Environment B* *(c) Environment C*

Figure 4.4: This figure shows the merged map area versus time from the exploration runs with one to four number of robots in all the environments. Ten runs were recorded for each number of robots and the averages were projected over time. The area of the merged map was saved at a frequency of ten seconds. The values are padded to the length of the longest exploration time for each set of robots, and as such the slope of the curves are more relevant than the time the exploration was finished. For a more accurate reading of the exploration times, see Figure 4.5a.

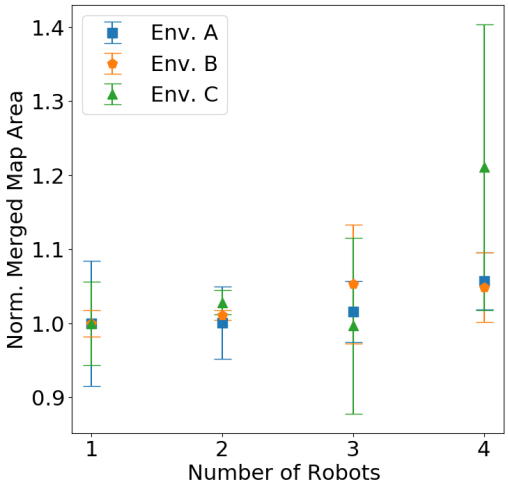
It is also apparent that the finished merged map area is larger when using several robots in the system, despite the fact that all number of robots fully explored the environment the majority of the time. The cause of this trend will be further addressed in Section 4.4.2.

4.4.2 Exploration Statistics

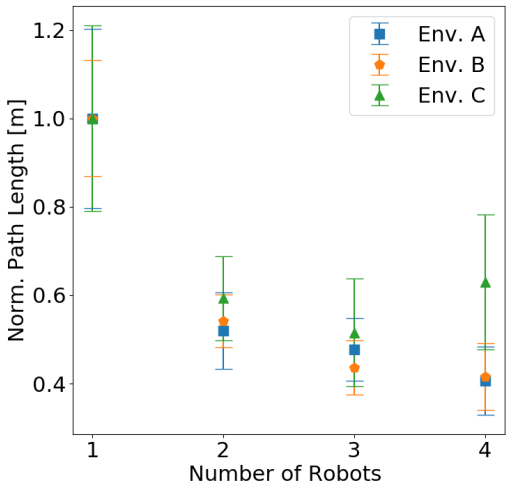
The rest of the statistics from the test runs are presented here. Sometimes the system was unsuccessful and crashed and other times it did not manage to fully collaborate between all the robots. The stability and success rate of the system is further explained in Section 4.5. The data was normalized with a factor determined by a single robot's exploration. This allowed for comparison between different environments to produce a more generalized view of the performance. For each case of one, two, three or four robots the initial positions were kept constant. The statistics for these runs can be found in Figure 4.5.


Norm. Exploration Time
Number of Robots
Env. A
Env. B
Env. C

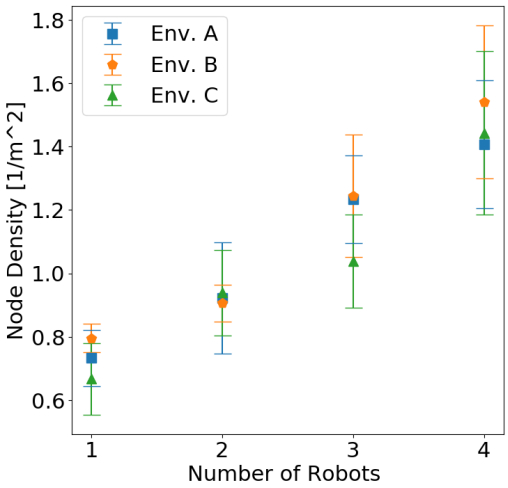
(a)


Norm. Merged Map Area
Number of Robots
Env. A
Env. B
Env. C

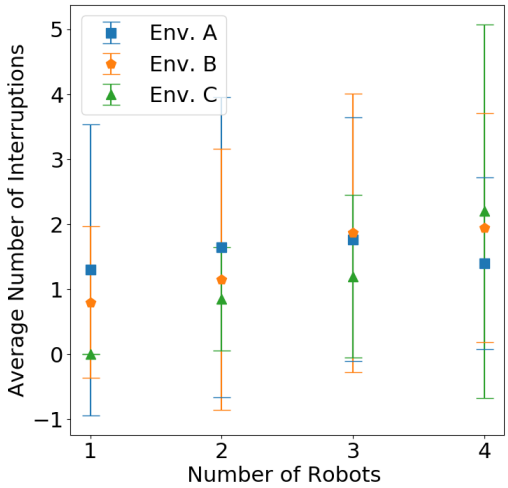
(b)


Norm. Path Length [m]
Number of Robots
Env. A
Env. B
Env. C

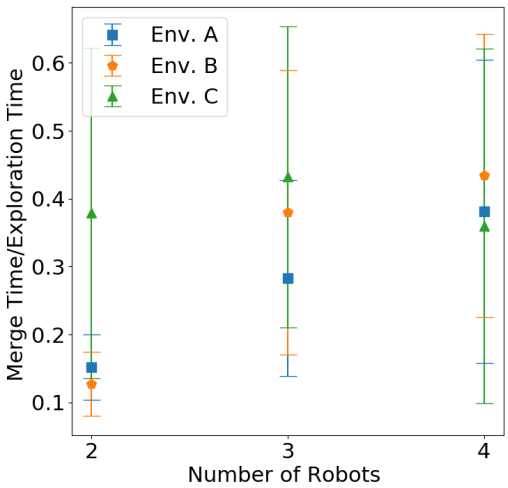
(c)


Node Density [1/m^2]
Number of Robots
Env. A
Env. B
Env. C

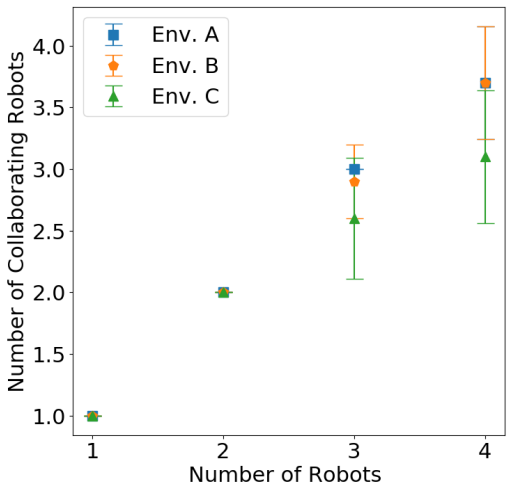
(d)


Average Number of Interruptions
Number of Robots
Env. A
Env. B
Env. C

(e)


Merge Time/Exploration Time
Number of Robots
Env. A
Env. B
Env. C

(f)

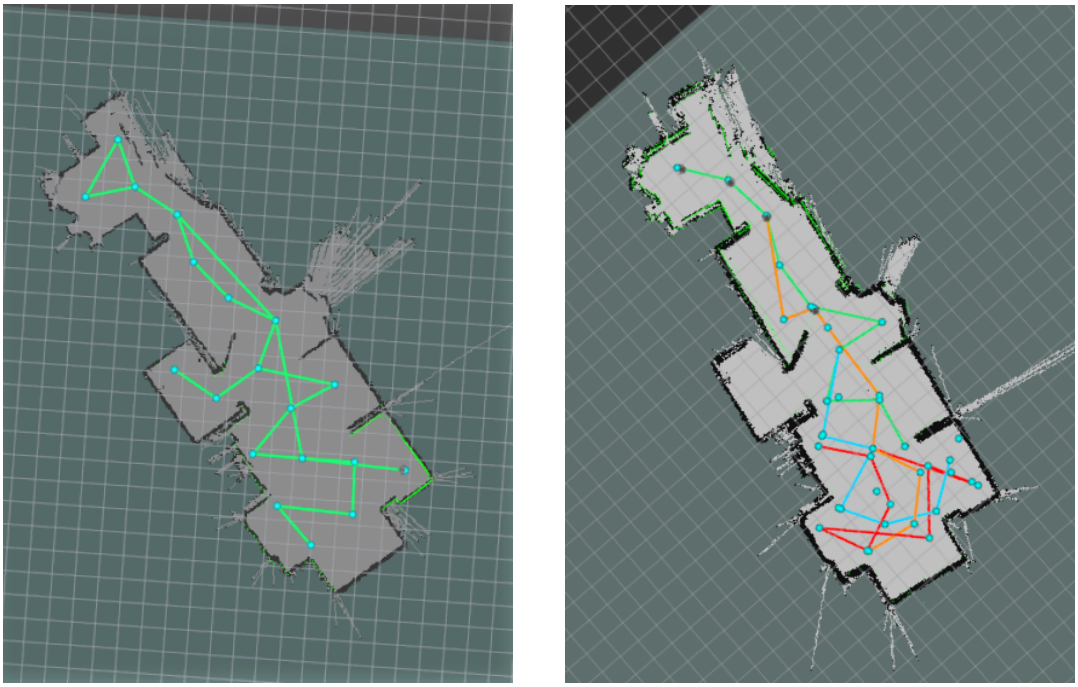


(g)

Figure 4.5: Results from the exploration runs with one to four number of robots in all the environments. Ten runs were recorded for each number of robots and their averages were calculated along with the standard deviation.

The most significant results from the runs with several robots is Figure 4.5a, where the normalized exploration time versus the number of robots is shown. The average exploration time for each number of robots is calculated and normalized with the average exploration time for one robot exploring that specific environment. There is a clear trend where the exploration is faster with more robots, although there are diminishing returns. The only outlier to this is the exploration in Environment C, where the exploration time increase from three to four robots. In this environment three robots had the fastest exploration time. The path length as shown in Figure 4.5c looks almost identical to the exploration time, since they are closely linked.

While the size of the explored area naturally varies a certain amount due to the stochasticity of the system, a trend can be seen that the normalized area of the final merged map increases as more robots are used and collaborate, see Figure 4.5b. The average map area of the merged maps are calculated for each number of robots, and normalized with the average value from one robot in that environment. As shown in Figure 4.6, when multiple robots map the walls in the surrounding environment they tend to become thicker which alters the amount of map area they are considered to cover.



(a) The merged map and trajectory of one robot exploring Environment A.

(b) The merged map and trajectory of four robots exploring Environment A.

Figure 4.6: Two examples of the finished map in RViz using one and four robots respectively in Environment A. The robot trajectories are either green, blue, orange or red.

In Figure 4.5b there are relatively large variances in the merged map area with three and four robots in Environment B, and even larger variances in Environment C. That was because there were some runs were the robots started to collaborate with a transform that was far from the ground truth. This will result in a merged map that is larger than it should be. The statistics of how common these runs were can be found in Section 4.5. An example of an exploration run with a poor accepted transform can be found in Figure 4.7.

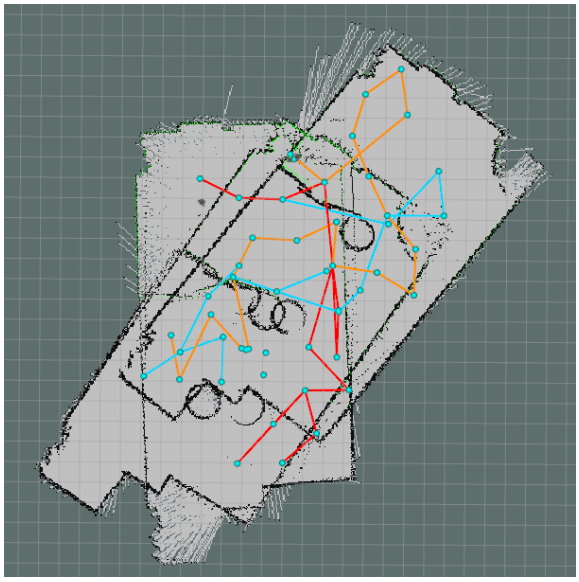


Figure 4.7: The merged map and trajectory in RViz of four robots exploring Environment C. The accepted transforms are far from the ground truth resulting in an inaccurate merged map where three out of four robots have started to collaborate. This will not only produce a merged map that is wrong and too large, the robots' relative positions will not be accurate either. Which is why some of the robots' trajectories passes through walls.

The node density is growing with the number of robots, as seen in Figure 4.5d. The average number of nodes in the merged graph is calculated and normalized with the average map area of one robot exploring that environment, as the map area of one robot more accurately resembles the actual size of the environment. This might be a consequence of the fact that with more robots exploring, more nodes will be placed in an overlapping area before the robots have started collaborating. When they start to collaborate these overlapping nodes will therefore be considered visited. This trend can also be seen in Figure 4.6, where there are several blue nodes that none of the robots have traveled through.

Regarding the number of interruptions shown in Figure 4.5e, the results show a very large degree of variance. There seems to be a small trend that with more robots the number of collisions increase. This trend could be a consequence from the likelihood of the robots colliding with each other increasing as more robots are added. Robots can also be interrupted several times in a row if they get stuck or repeatedly attempt to travel close to an obstacle with a sharp angle. These interruptions each count as separate instances which can be a cause of the large variance.

The average merge time over average exploration time in Figure 4.5f indicates that the average merge time ratio is growing with the number of robots, except for in Environment C where it is relatively constant. The average merge times for each number of robots is calculated, and normalized with the average exploration time for that number of robots. Thus, Figure 4.5f shows the average ratio of how long the robots are exploring on their own before they start to collaborate. Runs with more robots will provide more merge time values, which forces the system to find more transforms in order to merge successfully. Therefore, an increase in merge times as more robots are added to the system is reasonable, as seen in Environment A and B.

Figure 4.5g shows that the number of collaborating robots at the end of the exploration in Envi-

ronment C was on the lower end compared to the other environments. This means that the robots had a harder time merging and collaborating in this environment. By combining this with the fact that the robots were slow to start collaborating, as shown in Figure 4.5f, it is indicated that the *multirobot_map_merge* package struggled to find accurate transforms in Environment C. Other notable trends in Figure 4.5g is that there are diminishing returns with more robots.

Together with the number of collaborating robots as shown in Figure 4.5g, where the average number of collaborating robots at the end of the exploration is shown. It is significantly harder for three and four robots to find out their relative positions in Environment C, indicate that the *multirobot_map_merge* package struggled to find accurate transforms in Environment C. Other notable trends in Figure 4.5g is that with more robots it is more difficult to utilize all of the robots in the exploration.

4.5 System Stability and Success Rate

The system built managed to conduct autonomous exploration well. During the gathering of results the system was mostly stable and seldom failed to achieve its goal. However, with many subsystems working together to achieve the collaboration and with several robots operating in the same space, the system was prone to failing from time to time. The cause varied and was often due to the robots having a hard time identifying each other with only their lidars being in each others plane of visibility. This paired with a naive and simple collision avoidance caused the robots to sometimes be a hindrance to each other without managing to detect a collision. In turn this caused problems for GMapping and affected the localization and mapping process negatively. However this was not a common occurrence and the robots more often than not managed to break apart and continue exploring. Other system failures could be caused by the robots merging their maps with a very poor transform, see Figure 4.7. Accepting such a transform causes navigation problems which could lead to the system failing in its exploration. However, in general the system performed well and the stability of the system is presented in Table 4.2.

Table 4.2: The amount of finished as well as failed runs are presented in the table. Failed runs are the runs when the system failed completely. The runs considered finished are when the system managed to explore the environment until the algorithm deemed the exploration complete. The stability rate is simply calculated by taking the number of finished runs divided by the total amount of runs.

Number of Robots	Runs	Finished Runs	Failed Runs	Stability Rate
One Robot	30	30	0	1.0
Two Robots	41	40	1	0.9756
Three Robots	46	40	6	0.8696
Four Robots	44	40	4	0.9091
Total	161	150	11	0.9317

The runs with one robot come from ten runs in each environment for a total of 30. For other number of robots there are twenty runs from Environment A, since different initial positions were tested. The

other two environments contribute to all number of robots with ten runs each. Additional runs come from the fact that ten successful runs were taken in each environment. Thus, if the system failed an additional test was executed.

The finished runs could either be successful, partially explored or contain a poor merge. A successful run was defined as a run where the system managed to fully explore the environment, in contrast to only partially exploring it. A poor merge is when a transform far from the ground truth is accepted, resulting in an inaccurate merged map. Among the finished runs most of them were successful with a few occurrences when the system did not manage to fully explore an area. The success rate of the system can be found in Table 4.3.

Table 4.3: The number of times the environment was fully explored is presented in the table for the different environments. Runs in which the system did not fully explore the environment or merged poorly do not contribute to the success rate. The success rate is calculated by taking the number of fully explored runs divided by the amount of finished runs.

Environment	Finished Runs	Fully Explored	Partially Explored	Poor Merge	Success Rate
A	70	51	17	2	0.7286
B	40	36	1	3	0.9
C	40	32	4	4	0.8
Total	150	119	22	9	0.7933

The success rate of the system accurately reflects the difficulty that the different environments had. Environment A being the most difficult for the system while environment B was the easiest with the most open space, and Environment C was somewhere in between. Many of the partially explored cases in Environment A were due to a small conference room in the middle of the environment, which was particularly hard for the robots to enter. A high stability rate coupled with an overall success rate close to 0.8 reflects that the system is capable of somewhat reliably exploring a range of unknown environments.

5 Discussion

This thesis combined the research from multi-robot SLAM together with map-merging and created a collaborating system to autonomously and efficiently map an unknown environment. It showed that collaboratively exploring an environment with unknown initial positions was faster than single robot exploration. There are several aspects that could be improved, most notably being the exploration algorithm and collision avoidance. This section will discuss the implications of the results as well as its shortcomings and future works.

5.1 Research Findings

The results show that increasing the number of robots in the system improves the performance, which means that the collaborative aspects of the system show some prospects, see Subsection 4.4.2. The system benefits from being scalable to multiple units. However, due to the size of the environment as well as the size of one robot, one can suspect that at some point adding more robots will cause too many issues and interruptions for the system to function. The system is therefore unlikely to be scalable to a very large degree, which would be more akin to swarm exploration. Currently the system is also limited by the fact that there only exists one robot which is considered the master robot. This drawback reveals itself when using several robots. Due to this fact, when using four robots they cannot collaborate in pairs. If the system could use more master robots that would be an improvement which would make it more robust and allow for better scalability. Furthermore this issue will grow larger as more robots are added. As mentioned adding more robots causes an increase in the amount of issues and interruptions that occur in the system, see Figure 4.5e. The performance of the exploration should therefore at some point start worsening as more robots are added, depending on the environment and starting positions. In Environment C this turning point occurred at three robots, as shown in Figure 4.5a.

Using a node-based exploration algorithm made the collaborative aspect of the system more intuitive, since the robots simply made use of each others nodes in their collaborative exploration and navigation. The collaborative exploration algorithm was designed to further encourage the robots to explore separate areas when they have found their relative positions. The robots used nodes as goals and in the collaborative exploration they were inclined to explore different goals that were far away from other collaborating robots, see Subsection 3.6.5. This strategy allowed the collaborative system to efficiently explore the entire environment the majority of the time. Additionally the collaborative part of the system was very stable, and there were very few issues after the majority of the robots started to collaborate. Nearly all system failures were observed to occurred prior to the collaborative algorithm being initiated. As will be further explained in this section, the collision avoidance and inaccurate accepted transforms were the main culprits lowering the performance.

One question this thesis aimed to answer was whether an optimal number of robots to explore with could be found. The results point to clear benefits from using multiple robots, however this benefit pans out quickly as more robots are added and there is generally no clear optimal number except in Environment C. There was also no indication of the environment size having an impact on the optimal number of robots that should be used. However, this could be a consequence of the test areas being relatively small in the first place. This combined with the fact that merging and collaborating

was the main exploration time saver makes it difficult to argue that adding more robots will always yield a better result. Although there is no clear indication that the map area affects the optimal number of robots to use, the optimal number is still heavily dependent on the environment and its features. Since the system aims to explore an unknown environment finding an optimal number naturally becomes a problem that is very difficult to answer. For instance, if the area to be explored is one long corridor. In such an environment, if the robots starts in one end of the corridor, one robot will likely explore just as well as several robots since adding more robots will not really provide any benefits to the exploration. In general it is easy to define an environment that will disprove a given number of robots being the most optimal for any scenario.

From the testing done with different initial positions it could be seen that it was advantageous to have the robots start close to each other, as shown in Figure 4.3. This enabled the robots to almost always merge their maps and start collaboratively exploring. Collaborating lowered the exploration time significantly, which can also be seen by the massive gap in exploration time between the usage of one robot compared to using two robots. With distant initial positions it took a longer time for the robots to start collaborating, which resulted in longer exploration times. Therefore close initial positions were favored by the system. However, this simultaneously highlights the fact that the system is dependent on the robots' starting positions. While it does manage to explore with unknown initial positions, which is one of the main questions that the thesis investigates, purposefully placing the robots in poor starting positions will likely impact the performance of the system quite heavily. Thus, the system's parameters, exploration time and stability is dependent on the environment explored as well as the robots' starting positions in that environment.

A part of the system which required a lot of attention was how transforms were handled. This also had many possible tunable parameters which had to be determined. The main method used was explained in Subsection 3.7 and 3.8. Two of the queries that this thesis aimed to investigate was how transformations between maps should be handled and how two maps with a found transform between them should be combined to a global map. Due to the GMapping SLAM algorithm not being perfect, when having multiple maps of the environment that needed to be combined it was seen as a reasonable choice to be as strict as possible. Any point that was occupied in any of the maps would be considered occupied in the merged version as well. Therefore an area would only be considered free if that was the case in all robots' maps. This was the chosen strategy mainly since it can be considered to be the most safe way of approaching the merging of several maps, since all features are taken into account. However, this also causes errors and inaccuracies in the independent SLAM algorithms to accumulate. Since the merged map will be the result of summing all robots' maps their errors will be included as well.

5.2 Limitations and Future Work

The system implemented in this thesis reflected the intuitive feeling that more robots collaborating could yield a better result while also showing several interesting implications and limitations of this idea. These limitations of the system together with possible improvements in its many aspects will be discussed in this section.

5.2.1 Autonomous Exploration

Although the exploration algorithm performed well in general it was far from perfect, where the largest flaw was the need for tuning of the exploration radius parameter. The testing in Subsection 4.1 tuned the exploration radius for a difficult environment. This made the system reliable and usable in a wide range of environments while completely exploring the environment the majority of the time. However, since the parameter was not tuned for every new environment, the exploration could also be seen as being suboptimal. Having this sort of parameter which depends on the environment when the aim is to explore unknown environments is naturally a drawback.

Furthermore, a value of the exploration radius parameter that is too low will cause the system's robots to visit many unnecessary frontier points and thus take longer time than necessary to explore the environment. Exploration time was viewed as one of the most important performance indicators in this thesis. If the exploration radius is too large on the other hand, the likelihood of not exploring the whole environment increases. Implementing an algorithm that dynamically adjusts the value of the exploration radius would be a great solution. For example, if the robot enters a large room it is desirable to place a new frontier point in the middle of the room. From there, all the exits from the room are interesting to explore, and a new frontier point should be placed at all of these exits. By analyzing the lidar values it should be possible to extract these features in the environment and base the placements of the new frontier points on these features, and thus remove the necessity of the exploration radius parameter.

Along with the exploration having a reliance on the environment there were other weaknesses in the system. An important one being that the system had to succeed in all of its aspects or subsystems for the autonomous exploration to be effective and accurate. The exploration algorithm required new frontier points to be placed in favorable positions in order to explore the whole environment. This is not always easily achievable, especially in narrow environments. In addition to this, the Map Merging node had to find a good transform for the robots to start collaborating. If not, then the system essentially performed similarly to exploring with a single robot. The worst case though is if a transform is found and accepted but then GMapping loses accuracy from e.g. collisions, which might render that particular robot's map very inaccurate. This will in turn be applied to the merged map as well, leading to a potentially useless re-creation of the environment. Being heavily reliant on each part lowered the system's stability. Regardless, the stability of the system was still relatively high on average.

5.2.2 SLAM

In addition to the system being reliant on each part for overall satisfactory results, it is also a best effort system where the inaccuracies are adding up. GMapping itself is an estimation algorithm, and the Map Merging node has no guarantees of finding the ground-truth. This chain of subsystems result in the phenomenon of a larger merged map when more robots are exploring together. In other words, any small inaccuracies by SLAM in one robot can not be compensated for by any other robot, since all the SLAM algorithms are independent in this system. An integrated multi-robot SLAM solution on the other hand would perform SLAM simultaneously for all the collaborating robots together. When the system has figured out an estimation of their relative positions, the integrated multi-robot SLAM algorithm could use the data from all of the robots together to estimate one global map and

all of the robots' positions in it simultaneously. This could potentially allow more robots to create a more accurate map, since the system makes use of all robots' sensor data.

5.2.3 Collision Avoidance

A major lacking aspect in the system was the naive collision avoidance, presented in Subsection 3.6.6. This was also the main reason that made the system crash or fail when exploring. Sometimes when exploring the robots managed to interlock their wheels, and did not manage to dislodge from each other. This resulted in the robots severely blocking each other and the exploration eventually failing since the SLAM algorithm did not manage to adjust to the robot being blocked by another robot. Improving the collision detection and avoidance would increase the stability of the system from an already satisfactory level to an even higher one. It would also likely improve the exploration time since the current course of action when the robot is interrupted is to stop moving and wait a small amount of time and then start moving again. Having a proper collision avoidance that could avoid this would accumulate these small time saves and improve the exploration performance. A possible improvement of the collision avoidance system would be to implement a neural network that bases the robots trajectory on the lidar readings. As such, the network could be trained to dynamically avoid obstacles and adjust its path towards the goal.

5.3 Map Merging

In addition to the exploration being heavily dependent on the environment, the Map Merging node with the ROS package *multirobot_map_merge* from [15] also performed differently depending on the environment and the features in that environment. Since the package converts the maps to pictures, and extracts and matches features from those pictures as explained in Subsection 3.7, an environment containing more distinct features should allow the package to more easily find enough inliers in the model and therefore find better transforms quicker. Features in this case refers to any object that will produce an obstacle in GMapping. In Environment C the Map Merging node had a harder time to find transforms between the robots, as indicated by the number of collaborating robots at the end of the exploration compared to Environment A and B, see Figure 4.5g. This is likely due to there being less distinct features and sharp corners in that environment. Thus, optimal system performance would at least require tuning the estimation confidence in the Map Merging node for each environment. As stated above with the exploration radius, any tunable parameters that depend on the environment in a system made to explore unknown environments are not desirable.

Another interesting fact observed during the test runs were that the estimation confidence fluctuated quite a lot, even though the two maps in question stayed very similar. For instance, when two robots both had explored the majority of Environment C it would often take a value of around 1.0, but one second later it would be 0.7, with no visible changes in any of the robots' maps. One possible explanation to this is that the RANSAC loop runs for a relatively small amount of iterations, thus the models confidence in the found transform might fluctuate a lot since it is based on too few iterations of random sampling.

A feature based map merging algorithm should theoretically be able to merge maps with a relatively small amount of overlap, and thus be able to handle the case with distant starting positions decently.

Take for example the case with three robots exploring Environment A, where one robot starts in the bottom, another in the top and the third in the conference room to the left, see the map in Figure 4.6. After a short while the robot starting in the bottom has moved up to the middle of the map and meets the robot starting in the conference room. In this situation the square meters of overlap of their maps should be relatively large, about the same value as in the beginning of an exploration run when they are starting close to each other. The Map Merging node should thus find many features from this overlapping region and be able to extract a good transform between the maps. However, the Map Merging node struggled to do this. The package benefited greatly from the robots starting close to each other and having very similar maps for the package to find accurate transformations. Once again this could be a consequence of the RANSAC loop running too few iterations, since the number of matching features should be large enough. It could also be a consequence of very similar features in the non-overlapping parts of the maps, leading to the RANSAC loop suffering from too many inaccurate combinations. It hard to reason about how many distinct 2D features can be found solely based on the walls of an environment, but one can guess many of them look very similar.

Finally the behavior of the Map Merging node as a whole was not ideal for the use case in this thesis, since none of the found transforms are saved and reused in the next time step. Thus, the Transform Handler node had to be added in between the Map Merging node and the Autonomous Exploration node and act as a filter, as explained in Subsection 3.8. A large improvement to the system as a whole could be made if the Map Merging node was modified to the desired behavior, for instance, saving the found transforms and running the RANSAC loop longer for those cases. Additionally the master robot could change at any moment, which would also need to be fixed in order to never allow a collaborating robot revert back to exploring on its own. Lastly as mentioned previously, using several master robots would allow four robots to explore in pairs before all four of them can collaborate.

6 Conclusion

To conclude, this research has explored the effectiveness of collaborative autonomous exploration and has expanded on the collaborative aspects of state-of-the-art exploration techniques. The primary focus of this thesis was to develop a system that integrates multi-robot SLAM and autonomous exploration to create an efficient collaborative exploration algorithm. Additionally, the system utilized the *multirobot_map_merge* package from [15] in order to handle the general exploration case where robots have unknown initial positions. While this proof of concept system does not have immediate commercial applications, its framework can be extrapolated to similar projects in 3D environments, particularly in search and rescue missions.

Extensive testing was conducted on hardware, involving up to four TurtleBot3 Burger robots collaboratively exploring three different unknown indoor environments. The system was adequately reliable and shown to be scalable to a certain extent with a parallel software framework running an independent SLAM algorithm for each of the robots. The results clearly indicate efficiency gains in collaborative exploration compared to single robot exploration, and lowered the exploration time significantly. However, it was evident that adding more robots to the system yielded diminishing returns even though it improved the performance. While the use of multiple robots lowered the exploration time, the collaborative robots did not produce a more accurate reconstruction of the environment than a single robot due to accumulating errors.

The system encountered certain challenges, including poor collision detection prior to collaboration and performance dependency on the specific environment being explored. Optimal performance in each environment required tuning exploration parameters for that environment, as well as Map Merging parameters to initiate collaboration. Moreover, the initial positions of the robots had an impact on performance. In conclusion, the system's performance in a completely unknown environment heavily relied on the tuning from previous challenging environments, making it difficult to predict based solely on the results. Future work should focus on improving collision avoidance, dynamically adjusting exploration parameters based on the unknown environment, and implementing an integrated multi-robot SLAM algorithm to enhance collaborative accuracy.

By successfully combining the fields of multi-robot SLAM and autonomous exploration, this research has made significant strides in collaborative exploration. While there are still improvements to be made, this work paves the way for more advanced and efficient collaborative exploration systems in complex and unknown environments.

Bibliography

[1] J. E. H. Michael E. Porter. (2014) How smart, connected products are transforming competition. [Online] Available: https://hbr.org/2014/11/how-smart-connected-products-are-transforming-competition (accessed: 2023-014-17).

[2] A. Rodic, D. Katie, and G. Mester, "Ambient intelligent robot-sensor networks for environmental surveillance and remote sensing," in *2009 7th International Symposium on Intelligent Systems and Informatics*, 2009, pp. 39–44.

[3] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[4] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2d slam techniques available in robot operating system," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2013, pp. 1–6.

[5] I. Lluvia, E. Lazkano, and A. Ansuategi, "Active mapping and robot exploration: A survey," *Sensors*, vol. 21, no. 7, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/7/2445

[6] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli, "The sensor-based random graph method for cooperative robot exploration," *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 2, pp. 163–175, 2009.

[7] G.-R. Barb and M. Otesteanu, "5g: An overview on challenges and key solutions," in *2018 International Symposium on Electronics and Telecommunications (ISETC)*, 2018, pp. 1–4.

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

[9] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient autonomous exploration planning of large-scale 3-d environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699–1706, 2019.

[10] C. Wang, D. Zhu, T. Li, M. Q.-H. Meng, and C. W. de Silva, "Efficient autonomous robotic exploration with semantic road map in indoor environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2989–2996, 2019.

[11] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14 413–14 423, 2020.

[12] M. Mendonça, R. H. C. Palácios, E. I. Papageorgiou, and L. B. de Souza, "Multi-robot exploration using dynamic fuzzy cognitive maps and ant colony optimization," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2020, pp. 1–8.

[13] J. S. Cepeda, L. Chaimowicz, R. Soto, J. L. Gordillo, E. A. Alanís-Reyes, and L. C. Carrillo-Arce, "A behavior-based strategy for single and multi-robot autonomous

exploration," *Sensors*, vol. 12, no. 9, p. 12772–12797, Sep 2012. [Online]. Available: http://dx.doi.org/10.3390/s120912772

[14] A.-C. Stan, "A decentralised control method for unknown environment exploration using turtlebot 3 multi-robot system," in *2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2022, pp. 1–6.

[15] J. Hörner, "Map-merging for multi-robot system," Prague, 2016. [Online]. Available: https://is.cuni.cz/webapps/zzp/detail/174125/

[16] J. Tillett, T. Rao, F. Sahin, and R. Rao, "Darwinian particle swarm optimization," *Indian International Conference on Artificial Intelligence*, 2005. [Online]. Available: https://scholarworks.rit.edu/other/574

[17] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[18] D. Wang, H. Wang, and L. Liu, "Unknown environment exploration of multi-robot system with the fordpso," *Swarm and Evolutionary Computation*, vol. 26, pp. 157–174, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650215000711

[19] 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.

[20] J. Kim, "Cooperative exploration and networking while preserving collision avoidance," *IEEE Transactions on Cybernetics*, vol. 47, no. 12, pp. 4038–4048, 2017.

[21] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate slam with rao–blackwellized particle filters," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30–38, 2007, simultaneous Localisation and Map Building. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092188900600145X

[22] X. Fan, Y. Wang, and Z. Zhang, "An evaluation of lidar-based 2d slam techniques with an exploration mode," *Journal of Physics: Conference Series*, vol. 1905, no. 1, p. 012021, may 2021. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1905/1/012021

[23] K. Murphy and S. Russell, *Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks*. New York, NY: Springer New York, 2001, pp. 499–515. [Online]. Available: https://doi.org/10.1007/978-1-4757-3437-9_24

[24] Y. K. Tee and Y. C. Han, "Lidar-based 2d slam for mobile robot in an indoor environment: A review," in *2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)*, 2021, pp. 1–7.

[25] B. Xu, Z. Liu, Y. Fu, and C. Zhang, "Research of cartographer laser SLAM algorithm," in *LIDAR Imaging Detection and Target Recognition 2017*, Y. Lv, W. Bao, W. Chen, Z. Shi, J. Su, J. Fei, W. Gong, S. Han, W. Jin, and J. Yang, Eds., vol. 10605, International Society for Optics and Photonics. SPIE, 2017, p. 1060509. [Online]. Available: https://doi.org/10.1117/12.2292864

[26] I. Andersone, "Heterogeneous map merging: State of the art," *Robotics*, vol. 8, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2218-6581/8/3/74

[27] S. Carpin, A. Birk, and V. Jucikas, "On map merging," *Robotics and Autonomous Systems*, vol. 53, no. 1, pp. 1–14, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889005001041

[28] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006.

[29] X. Ma, R. Guo, Y. Li, and W. Chen, "Adaptive genetic algorithm for occupancy grid maps merging," in *2008 7th World Congress on Intelligent Control and Automation*, 2008, pp. 5716–5720.

[30] H. Li, M. Tsukada, F. Nashashibi, and M. Parent, "Multivehicle cooperative local mapping: A methodology based on occupancy grid map merging," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 2089–2100, 2014.

[31] S. Carpin, "Fast and accurate map merging for multi-robot systems," *Autonomous Robots*, vol. 25, no. 3, pp. 305–316, 2008. [Online]. Available: https://doi.org/10.1007/s10514-008-9097-4

[32] S. Saeedi, L. Paull, M. Trentini, M. Seto, and H. Li, "Map merging using hough peak matching," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4683–4688.

[33] ——, "Map merging for multiple robots using hough peak matching," *Robotics and Autonomous Systems*, vol. 62, no. 10, pp. 1408–1424, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889014001134

[34] J.-L. Blanco, J. González-Jiménez, and J.-A. Fernández-Madrigal, "A robust, multi-hypothesis approach to matching occupancy grid maps," *Robotica*, vol. 31, no. 5, p. 687–701, 2013.

[35] S. Yu, C. Fu, A. K. Gostar, and M. Hu, "A review on map-merging methods for typical map types in multiple-ground-robot slam solutions," *Sensors*, vol. 20, no. 23, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/23/6988

[36] X. Xu, L. Zhang, J. Yang, C. Cao, W. Wang, Y. Ran, Z. Tan, and M. Luo, "A review of multi-sensor fusion slam systems based on 3d lidar," *Remote Sensing*, vol. 14, no. 12, 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/12/2835

[37] P. Geneva, K. Eckenhoff, Y. Yang, and G. Huang, "Lips: Lidar-inertial 3d plane slam," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 123–130.

[38] W. Ding, S. Hou, H. Gao, G. Wan, and S. Song, "Lidar inertial odometry aided robust lidar localization system in changing city scenes," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4322–4328.

[39] N. Ragot, R. Khemmar, A. Pokala, R. Rossi, and J.-Y. Ertaud, "Benchmark of Visual SLAM Algorithms: ORB-SLAM2 vs RTAB-Map," in *2019 Eighth International Conference on Emerging Security Technologies (EST)*, Colchester, United Kingdom, Jul. 2019. [Online]. Available: https://hal.science/hal-02343341

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

[41] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[42] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[43] S. Das, "Simultaneous localization and mapping (SLAM) using RTAB-MAP," *CoRR*, vol. abs/1809.02989, 2018. [Online]. Available: http://arxiv.org/abs/1809.02989

[44] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831

[45] K. T. D. S. De Silva, B. P. A. Cooray, J. I. Chinthaka, P. P. Kumara, and S. J. Sooriyaarachchi, "Comparative analysis of octomap and rtabmap for multi-robot disaster site mapping," in *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*, 2018, pp. 433–438.

[46] M. Karrer, P. Schmuck, and M. Chli, "Cvi-slam—collaborative visual-inertial slam," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2762–2769, 2018.

[47] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli, "Covins: Visual-inertial slam for centralized collaboration," in *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2021, pp. 171–176.

[48] L. Platinsky, M. Szabados, F. Hlasek, R. Hemsley, L. D. Pero, A. Pancik, B. Baum, H. Grimmett, and P. Ondruska, "Collaborative augmented reality on smartphones via life-long city-scale maps," in *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2020, pp. 533–541.

[49] J. Jessup, S. N. Givigi, and A. Beaulieu, "Merging of octree based 3d occupancy grid maps," in *2014 IEEE International Systems Conference Proceedings*, 2014, pp. 371–377.

[50] ——, "Robust and efficient multirobot 3-d mapping merging with octree-based occupancy grids," *IEEE Systems Journal*, vol. 11, no. 3, pp. 1723–1732, 2017.

[51] M. Drwięga, "Efficient integration of octree based maps in multi-robot system," in *2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR)*, 2018, pp. 487–492.

[52] F. Gul, W. Rahiman, and S. S. N. Alhady, “A comprehensive study for robot navigation techniques,” *Cogent Engineering*, vol. 6, no. 1, p. 1632046, 2019. [Online]. Available: https://doi.org/10.1080/23311916.2019.1632046

[53] Y. Lu, Z. Xue, G.-S. Xia, and L. Zhang, “A survey on vision-based uav navigation,” *Geo-spatial Information Science*, vol. 21, no. 1, pp. 21–32, 2018. [Online]. Available: https://doi.org/10.1080/10095020.2017.1420509

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

[55] 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, no. 2, pp. 610–617, 2019.

[56] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998. [Online]. Available: https://www.cs.csustan.edu/~xliang/Courses/CS4710-21S/Papers/06%20RRT.pdf

[57] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, “Receding horizon "next-best-view" planner for 3d exploration,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1462–1468.

[58] S. Liu, S. Li, L. Pang, J. Hu, H. Chen, and X. Zhang, “Autonomous exploration and map construction of a mobile robot based on the tghm algorithm,” *Sensors*, vol. 20, no. 2, p. 490, Jan 2020. [Online]. Available: http://dx.doi.org/10.3390/s20020490

[59] F. Chen, J. D. Martin, Y. Huang, J. Wang, and B. Englot, “Autonomous exploration under uncertainty via deep reinforcement learning on graphs,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 6140–6147.

[60] R. Cimurs, I. H. Suh, and J. H. Lee, “Goal-driven autonomous exploration through deep reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 730–737, 2022.

[61] ROBOTIS. (2022) Robotis e-manual: Features. [Online] Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications (accessed: 2023-03-08).

[62] ——. (2022) Robotis e-manual: Appendix lds-02. [Online] Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_02/ (accessed: 2023-03-07).

[63] GvdHoorn. (2019) gmapping. [Online] Available: http://wiki.ros.org/gmapping (accessed: 2023-05-04).

[64] OpenCV-team. (2023) About. [Online] Available: https://opencv.org/about/ (accessed: 2023-04-25).

Appendices

A Algorithms

A.1 extractPossibleFrontierPoints

Algorithm 1 extractPossibleFrontierPoints(*robot_idx*)

$ranges, angles \leftarrow getLidarRanges(robot\_idx)$
$upp\_it \leftarrow start\_it$
$low\_it \leftarrow start\_it$
for each d_i in *ranges* **do**
  if $d_i \geq ex\_radius + \frac{1}{2} min\_explore\_width$ **then**
    $upp\_it \leftarrow i$
    $dist\_to\_obstacle \leftarrow min(ranges[low\_it - 1], ranges[upp\_it + 1])$
    $dist\_to\_obstacle \leftarrow min(dist\_to\_obstacle, ex\_radius)$
    $possible\_frontier\_angle \leftarrow (angles[low\_it] + angles[upp\_it])/2$
    $angle\_diff \leftarrow angles[upp\_it] - angles[low\_it]$
    if $dist\_to\_obstacle \cdot angle\_diff \geq \frac{2\pi}{3} \cdot ex\_radius$ **then**
      $frontier\_angles.add(possible\_frontier\_angle)$
      $low\_it \leftarrow i + 1$
      $upp\_it \leftarrow i + 1$
    end if
  else
    $dist\_to\_obstacle \leftarrow min(ranges[low\_it - 1], ranges[upp\_it + 1])$
    $dist\_to\_obstacle \leftarrow min(dist\_to\_obstacle, ex\_radius)$
    $possible\_frontier\_angle \leftarrow (angles[low\_it] + angles[upp\_it])/2$
    $angle\_diff \leftarrow angles[upp\_it] - angles[low\_it]$
    if $2 \cdot dist\_to\_obstacle \cdot \sin(\frac{1}{2} angle\_diff) \geq min\_explore\_width$ **then**
      $frontier\_angles.add(possible\_frontier\_angle)$
    end if
    $low\_it \leftarrow i + 1$
    $upp\_it \leftarrow i + 1$
  end if
end for
return $frontier\_angles$

Where *robot_idx* is the index of the current robot and $getLidarRanges(robot\_idx)$ function reads the current distances and at which angles they occur from the lidar sensor for the given robot.

A.2 extractAndAddFrontierPoints

Algorithm 2 extractAndAddFrontierPoints($robot\_idx$)

$graph \leftarrow getGraph(robot\_idx)$
$frontier\_angles \leftarrow extractPossibleFrontierAngles(robot\_idx)$
$current\_vertex \leftarrow getCurrentVertex(robot\_idx)$
$x, y, yaw \leftarrow getCurrentPosition(robot\_idx)$
for each v_i in $graph.vertices$ **do**
  if $p2pDistance(current\_vertex.data, v_i.data) \leq 2 \cdot ex\_radius$ **then**
    $possible\_neighbours.add(v_i)$
  end if
end for
for each fa_i in $frontier\_angles$ **do**
  $frontier\_point.x = current\_vertex.data.x + cos(fa_i - yaw) \cdot ex\_radius$
  $frontier\_point.y = current\_vertex.data.y + sin(fa_i - yaw) \cdot ex\_radius$
  $add\_frontier \leftarrow true$
  for each pn_i in $possible\_neighbours$ **do**
    if $(p2pDistance(frontier\_point, pn_i) \leq 0.95 \cdot ex\_radius$
      and $checkPointReachability(robot\_idx, pn_i))$ **then**
      $add\_frontier \leftarrow false$
      $brake$
    end if
  end for
  if $add\_frontier$ **then**
    $new\_vertex \leftarrow graph.addVertex()$
    $new\_vertex.data \leftarrow frontier\_point$
    $new\_vertex.visited \leftarrow false$
    $graph.addEdge(current\_vertex, new\_vertex)$
  end if
end for

Where $getGraph(robot\_idx)$ returns the current graph for that robot containing all the edges and vertices. $getCurrentVertex(robot\_idx)$ returns the current vertex that robot is standing in and $getCurrentPosition(robot\_idx)$ returns the position and orientation as estimated by Gmapping, in the local coordinate frame if the robot is exploring on its own or in the global coordinate frame if it is exploring collaboratively. $checkPointReachability(robot\_idx, p)$ uses the lidar data to determine if the point p is visible and reachable from the current robots position, if that's the case it returns $true$ and otherwise $false$. $p2pDistance(p1, p2)$ calculates the distance between two points.

A.3 updateGraphEdges

Algorithm 3 updateGraphEdges(*robot_idx*)

```
graph ← getGraph(robot_idx)
current_vertex ← getCurrentVertex(robot_idx)
for each v_i in graph.vertices do
    if not graph.edgeExists(current_vertex, v_i) then
        if checkPointReachability(robot_idx, v_i.data then
            graph.addEdge(current_vertex, v_i)
        end if
    end if
end for
```

A.4 chooseAndNavToFrontierPoint

Algorithm 4 chooseAndNavToFrontierPoint(*robot_idx*)

```
graph ← getGraph(robot_idx)
current_vertex ← getCurrentVertex(robot_idx)
start_vertex ← getStartVertex(robot_idx)
found_goal ← false
min_cost_val ← largeNumber
for each v_i in graph.vertices do
    if not v_i.visited then
        dist_curr ← p2pDistance(current_vertex.data, v_i.data)
        dist_start ← p2pDistance(start_vertex.data, v_i.data)
        if not collaborating(robot_idx) then
            cost_i ← λ · dist_curr + (1 − λ) · dist_start
        else if collaborating(robot_idx) then
            cost_i ← collaborativeCostFunction(robot_idx, v_i)
        end if
        if cost_i ≤ min_cost_val then
            frontier_goal ← v_i
            min_cost_val ← cost_i
            found_goal ← true
        end if
    end if
end for
if found_goal then
    if checkPointReachability(robot_idx, frontier_goal.data) then
        goToPoint(robot_idx, frontier_goal.data)
    else
        occupied ← checkOccupation(robot_idx, frontier_goal)
        if not occupied and graph.edgeExists(current_vertex, frontier_goal) then
            graph.removeEdge(current_vertex, frontier_goal)
            if graph.outEdges(frontier_goal) == 0 then
                graph.clearVertex(frontier_goal)
            end if
        else if occupied and graph.edgeExists(current_vertex, frontier_goal) then
            sleep(ex_radius/max_lin_vel)
            chooseAndNavToFrontierPoint(robot_idx)
        else
            frontier_goal ← breadthFirstSearch(robot_idx, frontier_goal)
            goToPoint(robot_idx, frontier_goal)
        end if
    end if
else
    explorationFinished()
end if
```

Where $goToPoint(robot\_idx, p)$ is the low lever function that takes a point p in the global coordinate system, rotates that given robot so it is facing p and then drives it forward until that point is reached. A callback is raised from the $goToPoint(robot\_idx, p)$ when the robot reached the goal, or if it is interrupted. $checkOccupation(robot\_idx, frontier\_goal)$ checks if any other robots collaborating with the current robot is standing in the frontier goal. $breadthFirstSearch(robot\_idx, frontier\_goal)$ performs a breadth first search in the graph to find the shortest path from current position to the chosen $frontier\_goal$, and return the first vertex in that path.

A.5 statusCallback

Algorithm 5 statusCallback(*robot_idx*)

```
graph ← getGraph(robot_idx)
msg ← getGoToPointMsg(robot_idx)
current_vertex ← getCurrentVertex(robot_idx)
frontier_goal ← getFrontierGoal(robot_idx)
x, y, yaw ← getCurrentPosition(robot_idx)
if msg == "Succeeded" then
    current_vertex ← frontier_goal
    current_vertex.visited ← true
    extractAndAddFrontierPoints(robot_idx)
    updateGraphEdges(robot_idx)
    chooseAndNavToFrontierPoint(robot_idx)
else if msg == "Interrupted" then
    occupied ← checkOccupation(robot_idx, frontier_goal)
    if not occupied then
        graph.removeEdge(current_vertex, frontier_goal)
        if graph.outEdges(frontier_goal) == 0 then
            graph.clearVertex(frontier_goal)
        end if
    else
        sleep(ex_radius/max_lin_vel)
    end if
    new_vertex ← graph.addVertex()
    new_vertex.data ← (x, y)
    new_vertex.visited ← false
    graph.addEdge(current_vertex, new_vertex)
    frontier_goal = new_vertex
    goToPoint(robot_idx, frontier_goal)
end if
```

This algorithm could be seen as a state machine for the robot. If the current robot has been interrupted by another collaborating robot it will sleep for a while to allow the other robot to move out of the way.

A.6 calculateSimilarityIndex

Algorithm 6 calculateSimilarityIndex(m_1, m_2, T)

$agr \leftarrow 0$
$dis \leftarrow 0$
for each y in $m1.width$ **do**
  for each x in $m1.height$ **do**
    $occ\_val\_m_1 \leftarrow m1.data[x,y]$
    $(x\_m2, y\_m_2) = T^{-1}(x,y)$
    $occ\_val\_m_2 \leftarrow m2.data[x\_m2, y\_m2]$
    if $occ\_val\_m_1 == -1$ or $occ\_val\_m_2 == -1$ **then**
      $continue$
    else if $occ\_val\_m_1 == occ\_val\_m_2$ **then**
      $agr \leftarrow agr + 1$
    else if $occ\_val\_m_1 ! = occ\_val\_m_2$ **then**
      $dis \leftarrow dis + 1$
    end if
  end for
end for
$similarity\_index \leftarrow \frac{agr}{agr + dis}$
return $similarity\_index$

Where T is the transform between the other robot's map, m_2, and the master robot's map m_1.

A.7 mergeMaps

Algorithm 7 mergeMaps(m_1, m_2, T)

$M = occupancy\_grid()$
for each y in $m1.width$ **do**
  for each x in $m1.height$ **do**
    $occ\_val\_m_1 \leftarrow m1.data[x,y]$
    $(x\_m_2, y\_m_2) = T^{-1}(x,y)$
    $occ\_val\_m_2 \leftarrow m2.data[x\_m_2, y\_m_2]$
    $M.data[x,y] \leftarrow max(occ\_val\_m_1, occ\_val\_m_2)$
  end for
end for
return M

Where T is the transform between the other robot's map, m_2, and the master robot's map m_1. M is the merged map in the coordinate frame of m_1

A.8 collaborativeCostFunction

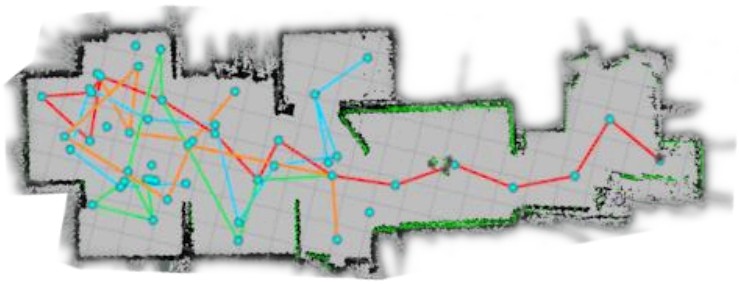
Algorithm 8 collaborativeCostFunction($robot\_idx, v_i$)

$current\_vertex \leftarrow getCurrentVertex(robot\_idx)$
$other\_robots \leftarrow getOtherRobots(robot\_idx)$
$cost\_val \leftarrow 0$
$epsilon \leftarrow smallNumber$
$dist\_curr \leftarrow p2pDistance(current\_vertex.data, v_i.data)$
for each r_i in $other\_robots$ **do**
$frontier\_goal\_r_i \leftarrow getFrontierGoal(r_i)$
$dist\_other \leftarrow p2pDistance(frontier\_goal\_r_i.data, v_i.data)$

$$cost\_val = cost\_val + \frac{dist\_curr}{dist\_other + epsilon}$$

end for
return $cost\_val$

Where v_i is the frontier point. $getOtherRobots(robot\_idx)$ returns the other robot indices that are currently collaborating with robot $robot\_idx$. The collaborative cost function is used in Appendix A.4.



Multi-Robot Collaborative Autonomous Exploration

Efficiently Mapping an Unknown Area

Master's thesis in Systems, Control and Mechatronics & Complex Adaptive Systems

ANDERS WALLSTRÖM
EDWARD BRASK

DEPARTMENT OF Mechanics and Maritime Sciences

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

B Additional System Parameters

B.1 GMapping Parameters

Table B. 1: Summary of the important parameter values used in GMapping that deviate from the standard values. A description of the package and its parameters can be found in [63].

Parameter name	Value
map_update_interval	2.0
maxUrange	6.0
minimumScore	50
srr	0.0001
srt	0.0002
str	0.0001
stt	0.0002
linearUpdate	1.0
angularUpdate	0.2
temporalUpdate	0.5
particles	100
xmin	-12
xmax	12
ymin	-12
ymax	12
delta	0.025
transform_publish_period	0.05

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY