Multi-Robot Exploration In Network-Uncertain Indoor Environments

An approach based on adaptive signal strength prediction

I.A. Kalkman

TU Delft

Multi-Robot Exploration In Network-Uncertain Indoor Environments

An approach based on adaptive signal strength prediction

by

I.A. Kalkman

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Tuesday July 2, 2024 at 12:30 PM.

Student number: 4585607

Project duration: September 1, 2023 – July 2, 2024

Thesis committee:

Dr. ir. J. Sijs,	TU Delft, supervisor
Dr. J. F. P. Kooij,	TU Delft
M. Spahn,	TU Delft
Ir. H. A. Steenbeek,	Royal Netherlands Marechaussee, supervisor

The cover image was generated with Stable Diffusion XL 1.0 with a prompt generated by ChatGPT 3.5.

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

I would like to thank my supervisors Professor Joris Sijs and Anne Steenbeek for their patience, guidance and expertise during this learning experience. It was enjoyable, insightful and challenging in equal measure. I would also like to thank my friends, family and partner Nancy for their continued support and the proofreading of so many pages.

I.A. Kalkman
The Hague, June 2024

Summary

In this thesis, an autonomous multi-robot system for indoor exploration in limited network environments is proposed. The specific use case is search and rescue where the operators must have access to the most up-to-date information, necessitating the requirement for communication maintenance. This requirement is satisfied in a novel way by using signal strength measurements collected by the robots to update the network model, thus reducing the overly conservative nature of current approaches. The network model chosen is an adaptation of existing path loss models as they balance complexity and performance given the information available during multi-robot exploration.

The proposed system consists of a central server with multiple robots which perform tasks that are readily distributable, whereas the central server performs complex iterative optimizations such as merging the local maps and running task assignment. While fully distributed architectures offer theoretical benefits such as scalability and robustness, in practice, these benefits are currently not achieved due to the complexity of splitting iterative optimizations such as mapping and task assignment among multiple nodes. This results in distributed architectures requiring more time and bandwidth to solve these problems, which is why this work implements a central architecture. While iterative optimizations are more efficient when performed centrally, scalability is limited due to the branching of possible options during task assignment. To combat this, an action space formulation, called an action graph, is proposed that is unique to each robot and reduces the number of actions by merging similar ones.

Both the dynamic network prediction model and the action graph formulation are shown to hold promise by experimentation. However, more work is needed before real-world use is feasible.

Contents

1

Introduction

Mobile robots have been successfully employed for a wide variety of tasks such as crop monitoring, industrial inspections and disaster response [83] [64] [12] [34]. While the implementations are impressive and add significant value for the end user, the mobile robots that are being used in the real world, especially in unknown environments, lack autonomy. They are often tele-operated or deployed in small numbers where the individual robots operate independently or semi-independently. The next step in mobile robotics is the application of cooperating systems of robots that can complete tasks faster and more efficiently than an individual robot. Moreover, through their cooperation, they can achieve tasks that are impossible for single-robot systems. Robot systems or subsystems that can cooperate in unknown environments have long only existed on paper or in strictly controlled laboratory environments [43] [66]. This is slowly changing as can be observed in competitions such as the Darpa SubT challenge, where teams of robots had to search for and localize artefacts placed in austere, complex environments [22]. This thesis aims to extend this multi-robot research domain, by presenting a method of achieving autonomous cooperation in unknown indoor environments in the absence of external networks with a focus on providing better situational awareness to operators in the search and rescue (SAR) domain.

Currently, some tools exist for this task that are used in practice [69]. However, they are tele-operated and their range is unreliable in indoor environments due to the presence of walls and other obstacles. Moreover, they only provide a video and audio stream to the operators, requiring them to interpret and memorize the indoor environment. One example of such a system is provided in Figure 1.1. These characteristics mean that first responders can only get an indication of the situation in the first few meters of a building, before they could lose the video feed of their robot due lack of a high bandwidth connection. Moreover, they might not know the exact reason for the disconnection, adding to the uncertainty of the situation.

Figure 1.1: Example of currently used robot and operator console for gaining insight in dangerous environments [69].

This generic scenario consists of four problems that require a solution. Firstly, tele-operating these robots limits the number of robots that can be deployed to the number of qualified personnel present, since each robot is directly controlled by an operator. This constrains the effectiveness of the system as multiple robots cooperating can solve tasks faster than a single robot. In addition to being faster, having multiple robots in the system would allow for some to act as network relays, increasing the area that can be explored. Secondly, only showing a video stream requires the operators to interpret the images they see and create a mental floor plan. They must memorize this floor plan and other pertinent information they have observed during exploration and then brief their colleagues before discussing possible courses of action. This methodology results in several points of failure and the loss of valuable time. Thirdly, the loss of connection between the operator and the robot(s) during exploration can create additional confusion in a situation where clarity is required. In the best case, this can lead to loss of valuable time in the rescue process while in the worst case, the first responders lose their only remote access to the building and will have to put themselves in more danger than necessary. Finally, when the operators do decide to enter a building, they have no information about the networking environment and whether or not they and their equipment will be able to communicate with the other operators in the building or support elements outside. Suddenly losing connection with operators during the SAR operation can again add more confusion to the situation.

1.1. Problem Statement

In this section, the problem statement will be presented in a concise manner with the aim of extracting the requirements the system must meet for the proposed search and rescue use case. The problem that has been sketched out in the scenario discussed above can be summarized in one sentence with several subproblems that are present in current approaches.

First responders do not have a robust method of acquiring information deeper into the interiors of buildings before deciding on a course of action.

Subproblems of current methods:

- Tele-operation limits the number of deployed robots.
- Robots cannot explore deep into buildings due to communication failure resulting from lack of inter-robot cooperation.
- Only having raw video requires operators to interpret, memorize and share data, increasing the likelihood of mistakes.
- Robots can disconnect suddenly, increasing confusion in a stressful situation.
- Providing no information about the network environment when operators do choose to enter adds confusion when connections break after entry.

From the problem statement and the subproblems, several requirements can be formulated. These range from high-level information that the system must provide to the end user, to performance requirements that ensure its functionality in the envisioned environments.

System Requirements:

- Require no user input for the exploration process.
- Ensure network connection between the robots and the outside base station in a direct or multi-hop fashion.
- Create a real-time, human readable 2D floor plan showing obstacles, unexplored space and predicted network strength.
- Require no prior information about the environment to function.

The first requirement dictates that the system must be autonomous in that it does not require control inputs from the operators allowing them to focus on their other tasks while the environment is explored. The second requirement states that connection between the robots themselves and the team outside is maintained at all times. This is crucial because during the operation, the first responders can decide to enter the building at any point in time based on the information provided by the robot team. For this reason, it is desirable that the team receive the most up-to-date information possible and thus making

it undesirable for robots to disconnect from the network to return at a later point with their gathered data. The third requirement dictates the outputs the system must provide to the operators. It must provide a 2D representation of the environment from which the operators must be able to intuitively understand the layout of the building with a distinction between explored and unexplored space. The system must thus merge the local maps from each robot into one global map. Additionally, when the first responders decide to enter the building, an estimate must be provided on where communication between the operators and the outdoors is possible and where not. Finally, since the system is aimed towards disaster response, specific information about the environment is unknown and thus the system must not require it.

The requirements and the subproblems can be used to formulate several research questions to help guide the literature research. The research questions are listed below and will correspond to the structure in Section 2.

Research Questions:

- How should the robots be organized?
- How can the robots' local maps best be merged into one global map?
- How can the network connection be modeled in a dynamic scenario where the robots can extend network coverage?
- Which task assignment algorithm is suitable for real-time performance in environments where the number of task combinations scales exponentially with the number of robots, while extending the communication range of the robots through cooperation and respecting the networking constraints?

1.2. Scope of Work

In this thesis, an autonomous multi-robot system will be proposed that provides first responders with insight and oversight of dangerous indoor environments with no external network coverage. Due to the open nature of the robotics community, several general tools and solutions that are required for many robotics systems can be used without requiring replication. First, the Robotic Operating System, ROS2 is a set of software libraries and tools that are specifically designed for the development of robotic systems [48]. This framework provides, amongst other functions, simulation, communication and visualization tools that greatly simplify the testing and implementation of robotic systems. Moreover, there is broad community support, providing off-the-shelf applications for specific functionalities such as mapping and navigation. Additionally, developing a system according to a widely used standard, as is the case for ROS2, increases the use for the community at large, since it will take less effort to incorporate parts of this research into future work performed by others should they so desire. For these reasons, the system will be implemented in ROS2.

The specific subsystems that can be used, due to them being implemented in ROS, are the multi-robot mapping and navigation. For navigation, Nav2 by Open Navigation can be used. This is an open-source, professionally supported ROS2 navigation stack that is used by more than 100 companies [47] [53]. This will simplify the system's design. Moreover, using a pre-existing state-of-the-art navigation system will allow for more time to be spent on refining the network model and its integration into the task assignment step. For mapping and localization, several well-documented and suitable frameworks exist such as COVINS-G [57], maplab2.0 [21] and Swarm-SLAM [44]. The main effort for mapping will consist of identifying a suitable framework amongst the many candidates.

The research gap that is identified in Section 2 lies in improving the use of signal quality estimates in exploration algorithms. Thus, the focus of this work will primarily be proposing a new method for refining and mapping real-world signal strength estimates and adapting an exploration algorithm to make use of this data and ensure that the network connection is maintained. The key in the implementation of the network prediction in the exploration step will be to find a generalizable and transparent algorithm that accepts networking constraints in a straightforward manner. Additionally, it must perform well in cases such as these, where the number of possible actions the system can choose scale exponentially with the number of robots in the system, increasing the time required to find the optimal solution rapidly.

After the network prediction and exploration algorithms are designed, a combination of simulations and real-world experiments will be performed to test the functionality of the system.

1.3. Report Outline

In Section 2, the formulated research questions will be answered. The main goal of the section is to identify possible algorithms and frameworks that can be used in the exploration system or, if these are not sufficient, find the research gaps that exist for the search and rescue use case. In Section 3, the proposed system will be presented using the answers to the research questions as a basis for the design decisions. After the theory of the previous chapters, the experiments that have been performed will be presented in Section 4 in combination with a discussion of the results. Finally, a conclusion is presented in Section 5.

2

Related Work

In this section, the research questions that have been formulated in Section 1 will be used to identify the relevant fields of research. Firstly, the different categories of multi-robot systems based on computation distribution will be presented. This is relevant since a choice must be made on whether the robots should be centralized, decentralized or distributed and how this fits with the design requirements as described in Section 1. Second, the different approaches to cooperative mapping will be discussed. This will lead to a choice on which C-SLAM framework to incorporate into the broader system. Thirdly, different methods of maintaining a stable network connection and predicting signal strength between robots will be discussed. Finally, multi-robot task assignment algorithms will be investigated.

2.1. Multi-Robot Architectures

When looking at methods to classify multi-robot-systems (MRS), there are proposed taxonomies that report on the different aspects of the MRS going back multiple decades [25]. While these taxonomies are useful , they are not standardized and quite in depth. With the aim of narrowing the research scope, the only distinction that is required for now is whether the system will be centralized, decentralized or distributed as defined by [6]. Choices made based on these three high-level categories will inform the direction of research with respect to the research questions without overcomplicating the process. These three network architectures are shown in Figure 2.1 [6].

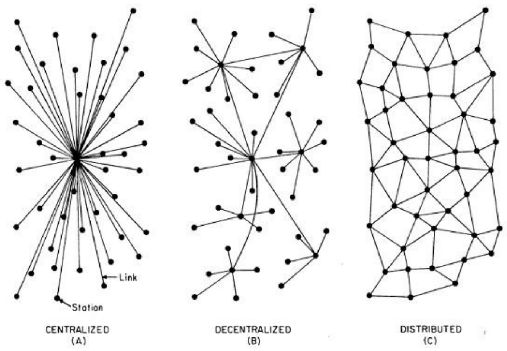

Figure 2.1: Centralized, decentralized and distributed networks [6].

Centralized systems, as can be seen from the first image in Figure 2.1, are systems where each node is only directly connected to one central node that processes all information and manages the other nodes in the network [6]. In the context of robots, this means that robots send back their sensor data to the central node which then, for example, merges the map or computes the next courses of action. Subsequently, the central node can communicate the results back to the agents.

The are two main advantages of centralized systems. Firstly, no complex bookkeeping is required to figure out which robot has what information. Each node always knows where to send its data as well as where to receive instructions from, greatly reducing the communication complexity of the system [43] [66]. Secondly, each node, or robot, can be significantly cheaper since it does not have to perform complex calculations, reducing its required computational capacity. This increases the feasibility of deploying larger numbers of robots, improving the performance of the system through scale. A downside of centralized systems is that, since all computations are performed on a single system, the scale of these systems is limited by the computational power of the central node as well as the connectivity of the system. Additionally, a centralized system has a single point of failure. If the central node fails, effectively the whole system fails due to the limited autonomy of the individual agents.

Decentralized systems, as depicted in the second image in Figure 2.1, are systems that consist of several clusters whose cluster centers are connected to the central node [6]. Each node sends its data to its cluster center. The cluster center then performs the necessary calculations and communicates this to the central nodes which is connected to the other clusters. This adds some complexity in the form of cluster management in exchange for more robustness to failure. If the central node fails, the clusters can continue their local operation while global communication is being restored. Likewise, if a cluster center fails, only its cluster will be affected, leaving the other clusters still in working order. A decentralized system is the middle ground between complexity and robustness, sharing in their benefits and drawbacks.

Distributed systems take the final in the direction of independence at the cost of more complexity. All nodes are connected to their direct neighbors, resulting in global coverage as shown in the third image of Figure 2.1 [6]. The advantages are that no large, central computer is required to perform complex calculations which leads to robustness, since there is no longer a single point of failure. Instead, the computations are divided amongst the robots in the network who then jointly produce a solution. In theory, this leads to maximal robustness since if any one robot fails, the problem is divided amongst the robots that are left and can still be solved. Additionally, since robots only communicate with their direct neighbors, less bandwidth is required for the network to function. The downside of distributed systems is that, depending on the application, they can be difficult to implement due to, for example, the complexity of splitting a problem to be solved by the nodes.

At first glance, the choice between the three networking types for the robot system seems to come down to a trade-off between scalability, management complexity and robustness to failure. However, several factors add nuance. Firstly, the operational requirement for the network to always be connected to a base station, as discussed in Section 1, impacts the system by introducing the base station as a single point of failure. Distributed systems will thus lose some of the robustness benefit since, if the base station fails and the operators lose their connection to the wider system, they lose their access to the information the robots gather, regardless of whether the robots can continue the exploration or not.

Second, the benefits and costs of a network architecture are not the same when applying the trade-off to each subsystem. For example, when looking at distributing a communications network, the costs that are inherent to the distributed nature are minimal. Finding the best routing path for communications is a well-established research area called routing problems, which have a long academic and real-world legacy being used in consumer applications, such as home WiFi repeaters [20]. However, in robotics, distribution does not just mean routing communications. Complex iterative optimizations must be efficiently divided amongst the robots such as is the case in multi-robot mapping [44]. Each step of the optimization requires communication with all relevant neighbors before the next step can be performed. This additional complexity currently completely negates any benefit that distributed networks theoretically have in scalability, limiting the size of the networks to a few robots before real-time performance is no longer possible or network bandwidth constraints are reached [79] [43] [44]. In contrast, centralized cooperative mapping methods such as maplab2.0 have been implemented for real-world problems in systems with more than a dozen robots [21] [72].

In conclusion, the choice for a network architecture cannot simply be made on a top level. While distributed systems hold great promise on paper, in practice they often underperform due to the difficulty in efficiently splitting the problem among the nodes. Moreover, the iterative nature of some optimizations requires constant communication and coordination between the nodes, therefore massively increasing the bandwidth and time required even surpassing centralized systems [44] [11]. When deciding on

the network architecture, the individual subsystems and their requirements have to lend themselves to decentralization or distribution and the costs and benefits might vary significantly between the subsystems. This is further compounded by the requirement of continuous connectivity, negating some of the advantage in robustness that distributed systems enjoy. The inherent simplicity of managing and designing for centralized systems is attractive and will be leading, unless a subsystem can be run in isolation on a single robot. In that case, it will be distributed to remove some load from the central server and increase the autonomy of the agents.

2.2. Multi-Robot Mapping

A map of the environment is not only the most important output of the system for the first responders, it is also crucial in the exploration process itself, since the system uses this map to keep track of previously explored sections in order to efficiently cover the area. The function of this subsystem is to be able to merge the mapping information from the individual robots and present it as a single, global map. Mapping and localization is not a trivial problem because it is a coupled problem. To map an area, the position of the robot must be known and to determine the position of the robot, without external positioning systems such as GPS that are not available in indoor environments, a map of the area is required [66]. The main method for achieving this global understanding for single robots is to solve these two problems simultaneously by performing Simultaneous Localization and Mapping (SLAM). This technique can be extended for multiple robots and is called Collaborative SLAM (C-SLAM), where a group of robots solve the coupled problem of mapping and localization as a team. However, C-SLAM is not simply multiple robots performing SLAM in parallel. The robots co-operate, sharing information to improve their localization estimates and jointly create a global understanding of the environment.

2.2.1. Problem Formulation

In this section the coupled nature of the C-SLAM problem will be illustrated by describing one of the mathematical problem statements of C-SLAM as described by [45]. There are several different formulations, however, the one by [45] displays the inter-robot measurements clearly. Let the set of state variables of both the robot and the landmarks in the map for robot α be represented by X_α and its set of measurements by Z_α. The formal goal of SLAM is to maximize the posterior of X_α. This posterior is shown in Equation 2.1.

$$p(X_\alpha|Z_\alpha) \tag{2.1}$$

All analyzed frameworks employ a smoothing approach with the Maximum A Posteriori (MAP) formulation in which the robots' previous movements, obtained trough odometry, are used to aid the estimation. For a single robot, α, this results in the following solution X^*_α of the MAP problem that is decomposed using Bayes' theorem as shown in Equation 2.2.

$$X^*_\alpha \doteq \arg\max_{X_\alpha} p(X_\alpha|Z_\alpha) = \arg\max_{X_\alpha} p(Z_\alpha|X_\alpha)p(X_\alpha) \tag{2.2}$$

In Equation 2.2, $p(Z_\alpha|X_\alpha)$ is the likelihood of the observed measurements given the state variables and $p(X_\alpha)$ is the prior distribution of the robots' motion measured by the odometry. Intuitively, this means that the SLAM problem entails the attempt to find the set of landmarks and robot poses X^*_α that is most likely to be the result given the measurements and prior trajectory estimations. This formulation can be extended to the multi-robot case with robots α and β for the case where the initial states of the robots relative to each other can be estimated as is shown in Equation 2.3.

$$(X^*_\alpha, X^*_\beta) \doteq \arg\max_{X_\alpha, X_\beta} p(X_\alpha, X_\beta|Z_\alpha, Z_\beta, Z_{\alpha\beta}) = \arg\max_{X_\alpha, X_\beta} p(Z_\alpha, Z_\beta, Z_{\alpha\beta}|X_\alpha, X_\beta)p(X_\alpha, X_\beta) \tag{2.3}$$

In Equation 2.3, $Z_{\alpha\beta}$ is the set of inter-robot measurements between robots α and β. These inter-robot measurements are often referred to as inter-robot loop closures and are used to stitch the separate trajectories of the robots together and correct drift errors in odometry measurements. Often at the

beginning of the SLAM session, the transforms between the robots' initial poses are unknown due to a lack of these inter-robot measurements. As a result, the trajectories of the robots cannot be merged until $Z_{\alpha\beta}$ is no longer empty. This means that the robots effectively solve the single-robot SLAM formulation until inter-robot loop closures are detected.

2.2.2. A Generic C-SLAM Framework

While each C-SLAM framework is different, on a high level, they follow a similar process. In this section, this process will be described before comparing and selecting the most promising frameworks. C-SLAM frameworks create global understanding of an environment by finding inter-robot loop closures - scenes that are observed by multiple agents - and using their pose estimates with respect to the scene as anchoring points for merging the different submaps. An optimization is then performed to minimize some error function of the global map, refining the initial estimate. The different subproblems that need to be solved in a C-SLAM framework are shown in Figure 2.2. The first is the odometry method, which extracts features from the sensor data and measures the movements of the robot to provide an initial estimate of the robot's position. In parallel, raw sensor data is used to create a descriptor of each observed scene at some interval. This descriptor is a compact way to represent the observations of the robot which can be readily compared with the descriptors from other robots to perform place recognition. The goal of place recognition is to find scenes that the other robots have observed before, the inter-robot loop closures, so that the separate trajectories of the robots can be merged.

Place recognition is not a perfect process, however, and depending on the method used, significant numbers of false positives can slip through, necessitating a method to reject these outliers. Outlier rejection checks some consistency metrics between the two observations in order to determine the likelihood of two observations being of the same scene. After rejecting the outliers, the frameworks employ some form of loop closure prioritization, since the use of multiple robots inevitably leads to a large number of loop closures which can drastically increase the time to convergence for the optimizer. This prioritization is done based on the likelihood that they improve some measure of the system's accuracy and subsequently only evaluating a fixed number of loop closures that score best.

After verification and prioritization, the detected loop closures are added to the map which is represented as a graph structure, with robot and landmark poses on the vertices and odometric and loop closure constraints on the edges. This map is then optimized by minimizing some function that describes the residual error or loss when all constraints are applied. This process is called Pose Graph Optimization (PGO). These steps are performed continuously during the session.

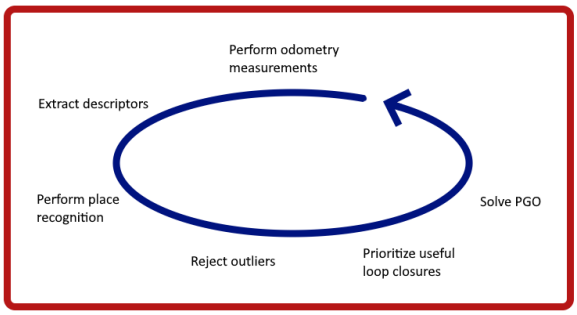

Figure 2.2: A generic C-SLAM framework.

There are many different kinds of C-SLAM algorithms and over the past few years, the trend in C-SLAM research has been to implement the systems in real-world environments instead of only in simulations or strictly-managed laboratory environments [45]. There are many proposed C-SLAM frameworks and, in order to narrow down the potential candidates, a table with requirements specific for C-SLAM has been created and is shown in Table 2.1.

Requirement	Description
Real-time	The system must be able perform SLAM in real-time to facilitate exploration and live decision making.
No GPS	A true SLAM system does not rely on GPS since that would immediately solve the localization problem. The robot system will be deployed in GPS-denied areas and must thus be able to function without.
Indirect inter-robot loop closures	A robot must be able to detect previously visited locations without aids such as beacons or markers left by other robots. It must also not rely on robots being outfitted with specific markers for recognition.
Open-source	The source code of the framework must be freely usable and ideally well-documented, describing the steps required for reproduction and the presence and effect of the tunable parameters.
Implementable	To prove the claimed theoretical performance and demonstrate its validity, the framework must at least be implemented with a multi-robot dataset on multiple systems connected via a wireless network. The systems can simply be different computers instead of full robots. Ideally, the framework is implemented in a real-world environment on a physical fleet of robots.
No a priori knowledge of and access to the environment	There is no knowledge about the exact environment except for the fact that it is indoors. The framework must thus generalize to several types of indoor environments such as offices, parking garages and houses. The framework must not rely on markers for localization that have been placed in the environment before the exploration mission.

Table 2.1: Requirements for C-SLAM frameworks.

The C-SLAM frameworks that meet the requirements as listed in Table 2.1 are listed below. A summary of the frameworks in table form is presented in Appendix A.

- COVINS-G [57]
- LAMP 2.0 [18]
- CCM-SLAM [67]
- maplab 2.0 [21]
- DOOR-SLAM [43]
- D^2-SLAM [79]
- Kimera-Multi [17] [70] [71]
- Swarm-SLAM [44]

2.2.3. Framework Comparison

Comparing C-SLAM frameworks is not as straightforward as looking up benchmarked values in a table. This is because the frameworks consist of multiple software modules and different hardware which can be used, often interchangeably, making it convoluted to replicate [45]. Moreover, SLAM evaluation techniques, such as the KITTI benchmark [40], are still not as commonplace for C-SLAM as they are for single-robot SLAM [45]. This results in each framework designing their own evaluation procedure and choosing which other frameworks they are compared to. Another problem encountered in the analyzed frameworks is that the used datasets are often of a single trajectory that is manually split into multiple trajectories for the different robots, making it difficult to ensure that the manually selected loop closures are sufficiently realistic [45]. This manual splitting also means that even if different frameworks are tested by their authors on the same SLAM dataset, comparisons on trajectory errors will still not be one-to-one since the datasets are split differently and the inter-robot loop closures will not be the same, possible creating a mismatch between the test scores and the system's real-world performance. On top of this, when multi-robot datasets are being used, they are often collected by the authors of the frameworks themselves and are not used by other frameworks.

The cause of some of the problems identified here seems to be the relative novelty of real-world C-SLAM. Almost all of the analyzed frameworks have only been introduced in 2022 or 2023 and some are still being actively developed. This leads to a lack of objective, third-party comparisons of the frameworks. In order to decide which framework should be chosen for the multi-robot system designed in this thesis, qualitative criteria will have to be used. An overview of the key features of each analyzed C-SLAM framework is presented in Table A.1 in Appendix A. The most important qualitative characteristics are judged to be heterogeneity, modularity and whether the system supports ad-hoc networking.

The reasoning behind this is that heterogeneity ensures that different robots with various sensors and configurations can be used in parallel. For example, aerial drones can be used to rapidly, but roughly,

explore an environment with a lightweight monocular camera, whereas ground robots with more computing power and more advanced sensors such as depth cameras and LiDARs are used to create a more accurate map and perform object detection, executing exploration strategies with the information received from the aerial robots. This makes place recognition challenging, since the perspectives from the aerial and ground vehicles are radically different as is explored in [57]. Moreover, the descriptors used for place recognition are either for visual or LiDAR data, requiring some sort of data processing to compare different sensor modalities directly as done by [21].

Modularity is crucial since it ensures that the framework is adaptable in case of new developments in, for example, descriptor types for object detection, outlier rejection methods or optimization schemes. Although many frameworks are modular in theory, in the sense that they consist of different subsystems, these can be highly interdependent, making it complex to swap any one module without heavily modifying the whole system [21]. Moreover, modularity allows for entire modules to be added when desired since the framework is designed for modification. For example, Swarm-SLAM and LAMP 2.0 have recently proven the benefit of loop closure prioritization. It is highly desirable to posses the ability to readily implement these types of innovations into the chosen C-SLAM framework without needing to modify more than is strictly necessary.

Finally, ad-hoc networking is valuable since the robots will be operating in poor networking conditions and will be required to create, extend and maintain their own network. This means that connectivity cannot be guaranteed and the robots must be able to dynamically reconnect with the network. Ideally, the robots will spend most of their time being connected to one another or the central server, since this allows for exploration strategies to be based on the latest information. This means that it is not required for robots to function independently for extended periods of time, say more than a few minutes. However, the framework should be able to recover from short network disruptions.

These three requirements are judged as being the most relevant for the intended application and will be used to eliminate frameworks that are missing more than one. Only one framework, maplab 2.0, meets all three of the primary requirements. Additionally, Swarm-SLAM and COVINS-G are missing at most one and will thus be included in the ranking. The frameworks that are missing more, in combination with concerns about implementational readiness due to the lack of deployment outside of laboratory conditions, can be removed from consideration. These are DOOR-SLAM, CCM-SLAM, D^2-SLAM, Kimera-Multi and LAMP 2.0. The first four are lacking in both modularity and heterogeneity as can be observed from Table A.1. LAMP 2.0 is not heterogeneous and does not support ad-hoc networking.

In conclusion, COVINS-G, maplab 2.0 and Swarm-SLAM passed at least two out of three qualitative requirements, with maplab 2.0 performing best out of the three. In particular, it excels in heterogeneity and modularity due to its ability to incorporate any sensor and odometry method that are compatible with each other. Unlike with COVINS-G and Swarm-SLAM, users can freely add different features and other resources to the map that best fit the type of sensors that are used. While COVINS-G can handle both aerial and ground robots cooperating, it is limited to the visual modality. Swarm-SLAM can handle both stereo cameras and LiDARs. However, the LiDAR can only be used in addition to a camera and the different sensors cannot detect loop closures in each other's data. Additionally, maplab 2.0 allows for almost every sub-component to be exchanged for another, such as descriptors or optimization techniques. Finally, it features experimental modules such as LiDAR projection and semantic loop closures that are a first step towards multi-modal loop closures.

2.3. Signal Strength Prediction

The function of the network model is to predict the network coverage of the environment, both for ensuring connectivity of the robots during the exploration and informing the first responders about possible communication dead zones when they decide to enter the building. Signal prediction is used in several forms in multi-robot systems. The goal is to predict whether or not a robot will stay connected to the network while it moves to its goal pose. There are various types of models that can predict network strength using many parameters and physical phenomena [31]. However, in robotics applications, they can be roughly grouped in the following five categories [3] [7]:

- **None:** Robots do not make assumptions about whether or not they can communicate with each

other. They only communicate if they happen to be connected.

- **Traces:** Robots communicate by leaving messages, such as beacons or tags, for one another in the environment to convey information.
- **Disk models:** Two robots can communicate if they are within a fixed distance, d_1, regardless of obstacles.
- **Line-of-Sight (LoS):** Two robots can communicate with each other if they are connected by an unobstructed line that is typically limited to some distance, d_2.
- **Signal:** Two robots can communicate if the estimated signal power between the two is higher than some threshold.

Using the requirement for continued connectivity, the first two options can be discarded since they cannot be used to predict or enforce a network connection at all times. The other three are all relatively similar in the sense that they provide some estimate of where the system can be connected and where not.

The most basic form of these models is the disk model, where robots are assumed to be able to communicate if they are within some predefined distance irrespective of obstacles between the robots [3] [75]. The advantage is that this model is robust and simple to implement. The downside is that it does not take into account any information about the environment such as the impact obstacles have on signal propagation. This means that for this application, where continued connectivity is a hard requirement, the connection radius would have to be strict, limiting the area the robot network can cover with a fixed number of robots.

The next model in terms of complexity is the LoS model. This model assumes connectivity between two arbitrary points on the map if a straight line of maximum length, d, can be drawn between the two without intersecting any walls. The LoS model bypasses the need to collect data about the environment to use in its signal quality estimate, by only allowing connections that do not intersect obstacles. This model can potentially extend the coverage with respect to the disk model in relatively sparse environments. In dense environments such as indoor environments with a lot of rooms and hallways, its estimates could be overly conservative, since it would require a robot at each corner to extend the network around the obstacles that the signal could perhaps propagate through without problems. This model adds some complexity with respect to the disk model in the form of checking for obstacles, which can improve the signal estimate in sparse environments. However, its performance can be limited in dense indoor environments.

The final category, which adds the most complexity, consists of models that estimate signal power between the transmitter and receiver to determine whether or not two positions are connected. These models vary in complexity. However, in the case of mobile robotics the choice is often limited because, for example, information about the materials obstacles are made from or their surface roughness is not available, making accurate ray-tracing unfeasible. Moreover, the real-time constraint of exploration also limits the allowable computational complexity of these models. The model chosen in almost all cases is the path loss model due to its balance between accuracy and computational complexity [59] [7]. Moreover, it is based on empirical measurements of many different frequency channels, which makes it representative for WiFi-based systems in indoor environments [59].

In short, disk and LoS models are the simplest to implement. However, they artificially constrain the estimates to be more conservative than necessary. Signal models predict signal power between a transmitter and receiver which can mitigate the artificial constraints imposed by disk and LoS models. Signal models such as the path loss model warrant further investigation, which will be performed in the following sections after a basic introduction to signal propagation theory.

2.3.1. Signal Propagation Theory

In order to gain a better understanding and appreciation of the benefits and drawbacks of the two main signal propagation models that will be presented shortly, some basic theory must be discussed first. The type of network that is under consideration for this robotic system is a mobile ad-hoc network (MANET) using the 802.11n standard for Wi-Fi networks. Currently, the most used frequencies for Wi-Fi networks are 2.4 and 5 GHz with signal wavelengths of 12.5 and 6 cm respectively. The main phenomena that

Wall Material	Wall Loss [dB]	
	Thin Wall	Thick Wall
Layered drywall	2	-
Concrete	10	15
Glass	2	4
Wood	6	-
Brick	7	-

Table 2.2: Table of typical signal attenuation values for various materials both for thin (<15 cm) and thick (>15cm) walls [59] [68].

affect signal propagation of these networks are path loss, shadowing, reflection, refraction, diffraction and scattering [31] [33].

Path loss describes the dissipation of the power radiated by the transmitter (Tx) as a function of the distance between the Tx and receiver (Rx) and channel properties such as frequency. It does not model any interaction with the environment and is only based on the distance between the Tx and Rx. It thus models LoS signal propagation.

Shadowing is the effect of obstacles between the Tx and Rx absorbing some fraction of the transmitted signal. This presents a problem since the extent to which a signal is absorbed or let through is dependent, amongst others, on the material of the obstacles which are typically unknown. This is especially the case for robot exploration, making it difficult to model accurately. Typical signal attenuation values for several materials are shown Table 2.2.

Reflection, diffraction and scattering are different from path loss and shadowing in the sense that their interaction with the environment leads to a change in signal path. These changed paths take different amounts of time to reach the receiver, resulting in a phase shift that can be constructive or destructive. This phenomenon is depicted in Figure 2.3. The resulting signals are referred to as multi-path signals.

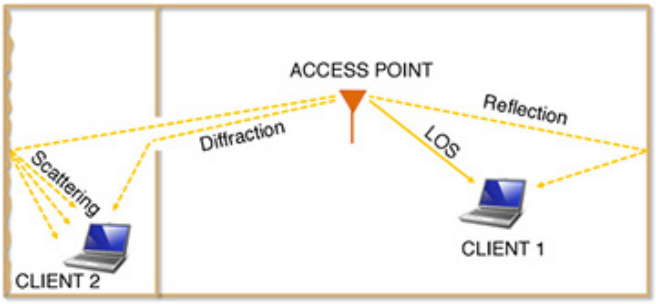

Figure 2.3: Signal Propagation by Line-Of-Sight (LOS), reflection, diffraction and scattering [33].

Reflection happens when a wave collides with smooth obstacles whose surface smoothness deviates on a much larger scale than the wave length of the signal [33]. In accordance with Snell's Law, part of the signal is then reflected and another part is refracted [32]. Scattering, in contrast to reflection, happens when a wave collides with a rough obstacle whose surface smoothness deviations are in the same order of magnitude as the wavelength of the signal. The signal is then scattered, meaning that several different rays around the angle of reflection are created. Scattering has a larger loss of energy due to the same signal covering a larger area. Additionally, the different signals interfere with one another and reach a receiver with slight time differences [33] [31].

Diffraction is a wave's ability to change its propagation direction, or bend, around sharp corners such rooftops, street corners and doorways [33]. It can be understood with the Huygens-Fresnel Principle, which states that all points on the signal wavefront can be considered as point sources for a secondary wavefront called a wavelet [31]. Diffraction can be modelled accurately. It is too computationally costly and complex, however, to be used in wireless channel modeling, let alone in a real-time scenario. The most used simplification is the Fresnel knife-edge diffraction model. However, that still involves solving

multiple complex integrals resulting in approximations being used.

The received signal strength is a superposition of all the effects described above and can be calculated with, for example, ray-tracing methods. However, to do so with any accuracy requires significant information about the environment and computational power. For the cases when this environmental information or the computational complexity is not available, many different propagation models based on empirical measurements have been created for multiple environments and frequency ranges. These models are computationally lightweight and do not require much information about the environment at the cost of prediction accuracy. The two signal power prediction models that are used in robotics will be discussed further below.

2.3.2. Gaussian Process Regression

The work of [3] proposes using Gaussian Process (GP) regression to model the signal strength based on performed signal strength measurements in an attempt to bypass the conservative nature of disk and LoS models. Noisy measurements performed by the robots can be used to estimate the parameters of the signal strength function by maximizing the observations' log-likelihood. The function that is obtained can then be used to estimate signal strength in the unvisited regions if enough measurements have been collected. An image of a map created by six robots around a static transmitter is shown in Figure 2.4. This approach is used by others in their multi-robot exploration proposals such as [78].

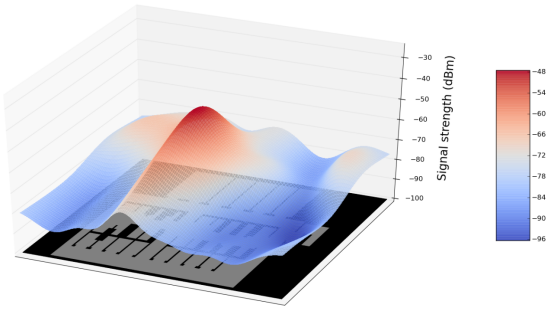

Figure 2.4: Communication map created by six robots around a single source in the center of the map [3].

This approach is promising since it provides the robots with a more realistic estimate of the signal strength than the circle, LoS or current path loss models. Moreover, it provides an uncertainty estimate of the signal strength, making it possible to judge the confidence in the prediction and reject unreliable ones. Its advantage is that it makes no assumptions about the environment and solely relies on signal strength measurements. However, since measurements are used to directly fit a signal strength function, the function that is obtained by the GP regression is only valid if the robots do not impact the network quality when moving. In Figure 2.4, there is only one, static, source that the robots are connected to and the robots themselves have no impact on the signal propagation. This is not the case in a distributed mesh network, where the robots serve as network relays as well. In that case, the signal strength function that is created by the GP would not hold, since when the robots move to explore the area, the networking nodes also get rearranged, thereby invalidating the previously obtained function. This could be mitigated by only using the most recent signal measurements to create the signal model. However, this would still mean that the signal model is only valid in the direct vicinity of the robots under the condition that the network topology does not change significantly leading to high levels of computational loads which is already the case for a static network [3].

2.3.3. Path Loss Models

Path loss is a measure of how much of the transmitted signal power will reach the receiver. Specifically, in robotics applications such as [7], path loss exponent models are used where an exponent, γ, incorporates many model parameters such as antenna and ceiling height [31]. Typical values of γ are between 1.6 and 3.5 for same-floor indoor environments [31]. The higher this exponent, the larger the estimated loss will be at a given distance.

The basis of path loss models is the free space loss presented in Equation 2.4 as proposed by Friis' equation from 1946 [28] [31]. In this equation, P_t and P_r are the power fed to the transmitter and the power available to the receiver, respectively. G_r and G_t are the gain of the receiver and transmitter, respectively, and d is the distance between the transmitter and receiver. Finally, λ is the wavelength of the signal.

$$\frac{P_r}{P_t} = G_t G_r (\frac{\lambda}{4\pi d})^2 \tag{2.4}$$

Received power is often expressed in dBm which is the decibel value of a power, x, relative to a milliwatt. Thus 1 W of power is equal to $10log_{10}(x/0.001) = 30dBm$ [31]. This is done for convenience since the logarithmic scale is less cumbersome to use with values that span large ranges. To accomplish this, Equation 2.4 needs to be rewritten to convert its initial power units of Watts to dBm as is shown in Equation 2.5.

$$P_r(dBm) = P_t(dBm) + 10log_{10}(G_t G_r) + 20log_{10}(\lambda) - 20log_{10}(4\pi) - 20log_{10}(d) \tag{2.5}$$

Equation 2.5 is referred to as a link budget equation, since it expresses the received power of a signal through a given frequency channel, or link, as a function of the transmitted power and all the losses the signal experiences [31]. Equation 2.5 can be rewritten to return the free space path loss as shown in Equation 2.6 [31].

$$L_{FreeSpace} = 10log_{10}(\frac{P_t}{P_r}) = -10log_{10}(G_t G_r (\frac{\lambda}{4\pi d})^2) \tag{2.6}$$

Equation 2.6 only represents the free space loss in an outside environment. To include the effect of walls, floors and the ceiling on the signal strength, the model will need to be expanded. First, modeling the signal propagating through an empty indoor environment is not the same as an empty outdoor environment. The signal interacts with the floor and the ceiling, impacting its strength. This can be modeled using the path loss exponent, γ, as shown in Equation 2.7 [31] [3]. This function is referred to as the distance loss. The free space component in this equation is incorporated as the free space loss at a reference distance d_0, and can be abbreviated by P_{d_0}. The reference distance is typically 1 m [59] [31]. While P_{d_0} can be calculated, it can also be obtained empirically which will more accurately incorporate system parameters such as antenna properties.

$$\begin{aligned} L_{distance} &= -10log_{10}(G_t G_r (\frac{\lambda}{4\pi d_0})^2) - 10\gamma log_{10}(\frac{d}{d_0}) \\ &= P_{d_0} - 10\gamma log_{10}(\frac{d}{d_0}) \end{aligned} \tag{2.7}$$

Finally, propagation through obstacles such as walls is not yet taken into account. The impact that obstacles have on the propagation of a signal can be modeled as a linear loss on a logarithmic scale [5]. Currently, this is accomplished by counting the walls between the transmitter and receiver and multiplying the number of walls with a constant, the Wall Attenuation Factor, W_f [5] [7] [8]. W_f can differ wildly based on the material and thickness of the wall. Concrete has a W_f that is 5-7.5 times higher than drywall as is shown in Table 2.2. However, in current robot systems, W_f is chosen as a constant beforehand to simplify the process. This approach is shown in Equation 2.8, where N is the number of walls the signal path intersects.

$$\begin{aligned} L_{path} &= L_{distance} - L_{wall} \\ &= P_{d_0} - 10\gamma log_{10}(\frac{d}{d_0}) - W_f N \end{aligned} \tag{2.8}$$

In Figure 2.5, [7] have used the concept of distance loss and wall loss, with a static W_f, to create a function, Γ_i, that predicts signal strength. They have shifted the path loss function from Equation 2.8 to

be positive by setting a maximum communication range, $c = 30m$, and shifting the path loss function upwards so that it equals zero at a distance $d = c$. This function is then used to create a utility function that encourages the robots to stay within communication range by assigning lower utilities to positions with lower predicted signal strength.

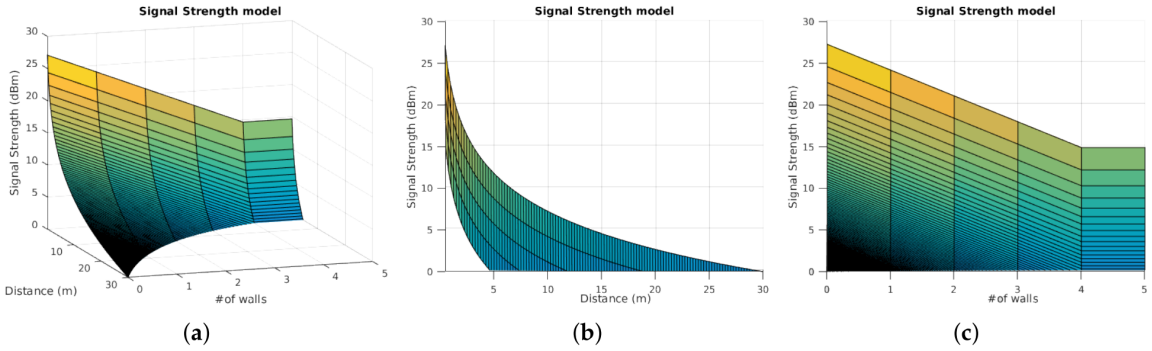

Figure 2.5: Signal strength model that expresses path loss as signal strength [7]. The effect of walls on signal propagation is modeled as a constant loss that is set in advance.

A limit of this approach is similar to the disk model, where the requirement for continuous connectivity has the effect of requiring strict parameters. In this case W_f will need to be quite high, which limits the coverage of the system since not all walls will be made of thick concrete and signal attenuation through walls varies heavily depending on the material and thickness of the wall as is shown in Table 2.2. In the work by [59], a differentiation is made based on the type of wall which is then added as prior knowledge for their path loss model. However, their work is not in the robotics field and they simply estimate the wall type manually. This approach is unfeasible in the context of autonomous exploration since this information cannot be known beforehand. A possible improvement to the state of the art of path loss models could be a method to estimate W_f for each wall segment, to update the initially strict W_f during exploration and enable less conservative behavior.

In conclusion, several different models exist for estimating signal strength in indoor environments. The main trade-off is between model accuracy and complexity with an upper limit on the latter due to the lack of specific environmental information. If a model is highly inaccurate due to its simplicity, it must be overly conservative in its estimates when continuous network coverage is required. Path Loss models seem to occupy the middle ground in this trade-off, however, in their current form, due the assumption of a static, predetermined W_f, they also suffer from overly conservative estimates. A current research gap is thus the incorporation of real-time signal measurements to create a communication map that is resilient to the changing topology of the network. Ideally, this communication map would combine the strengths of the path loss models, lightweight computation and validity for any network topology, and GP regression, being based on measurements instead of overly conservative a priori estimates and providing an uncertainty estimate of the prediction.

2.4. Multi-Robot Exploration

Exploration can be defined as the process of navigating unknown environments, aiming at the complete coverage of an area of interest while minimizing the overall exploration time [11] [14]. As with C-SLAM, extending exploration to a team of robots can yield significant benefits with respect to the reduction of exploration time and the introduction of redundancy to the failure of a single robot [15] [14] [11] [43]. In the specific use case of exploration in environments with no external networks, using multiple robots can even be a necessity to extend the range of the communications network to enable a sufficiently high coverage of an environment.

Since it is known that the Nav2 stack will be used, the task of the exploration algorithm will be to identify tasks and assign them to the robots so as to explore the environment, while maintaining continuous connectivity between the robots and the base station. The need to navigate to the goals while avoiding

collisions can already met by using the Nav2 stack and will thus not be part of the scope. The main aim of this section is to identify possible planning algorithms that can enforce constraints as described in Section 1. The requirements that are most relevant for the exploration planning algorithm are maintaining network coverage while the robots explore and the planner requiring no a priori information about the environment to function.

Robotic exploration in general, and multi-robot exploration in particular, are widely studied fields of research. However, most efforts that focus on multi-robot exploration in real-world environments work towards making the system more robust to robots disconnecting instead of enforcing continuous connectivity [3] [11] [50]. Moreover, of the approaches that do maintain continuous connectivity, most do not have a fixed base station and are only focused on maintaining or encouraging connectivity between the dynamic robots [7]. To compensate for the lack of studies that fit perfectly in the current scope, instead of only looking at complete continuously connected multi-robot exploration proposals, this section will be organized in three subsections that investigate the identification of tasks, the maintenance of connectivity and the assignment of tasks to robots respectively.

2.4.1. Task Identification

Goal or Task identification is the process of identifying places of interest that the robots should visit. This strongly depends on the sensors available to the robot and the representation of the environment, but the most well-known is the Occupancy Grid [7] [3]. The Occupancy Grid, as introduced by [27], is a representation of the environment in which the map consists of cells where each cell can either be unknown, empty or occupied. In ROS2, the Occupancy Grid uses 8-bit signed integers where a value of -1 denotes unknown space and a value between 0 and 100 is assigned to the other cells, referring to the probability that that there is an obstacle in that cell. The user specifies a threshold with the default being 65, meaning that if a cell contains a value, $v \geq 65$, it will be seen as occupied. If a cell contains $0 \leq v < 65$ it is free space [55].

When aiming to explore a previously unknown environment, the most widely used method for identifying positions of interest is to identify frontier cells [7] [3] [38] [37]. Frontiers are cells on the border between free and unexplored space. The reasoning is that they are poses that, if visited, can increase the knowledge about the environment since unknown space can be observed from them [81] [80]. In Figure 2.6, a representation of a map showing the frontiers is presented, where the frontiers' centroids are the tasks that the robots must move to.

The concept of frontiers was first introduced by [80] over two decades ago. However, it is still successfully used today, even in real-world, complex, heterogeneous Multi-Robot Systems (MRS) such as [72] and [61]. Many different algorithms have been designed to efficiently detect these frontier cells and process them into goal poses [38] [37]. However, investigating these is beyond the scope of this thesis since state-of-the art approaches are fast enough that, realistically, they will not be the performance bottleneck during exploration. In Section 3.3.1, the method for detecting these frontiers will be presented.

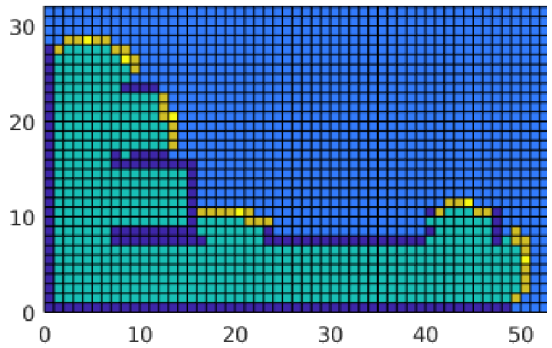

Figure 2.6: Map showing frontier centroids where light blue cells are unknown space, dark blue cells are obstacles, green cells are free space and orange cells denote the frontiers. The frontier centroids are yellow cells in the center of a frontier [7].

2.4.2. Connectivity Maintenance

In the literature that proposes methods for continuous network coverage for MRS, two classes of approaches can be identified. The connection can either be maintained through the proactive prediction of signal strength which is integrated into the task assignment step, or in a reactive fashion during the exploration where behavior is modified based on measurements. The predictive methods used in robotics use one of the methods described in Section 2.3 to predict whether or not signal strength is sufficient between two positions on the map. The integration of this prediction into task assignment is handled in different ways. In some cases, configurations that are predicted to lead to disconnection are ignored by the planner entirely by assigning a utility to them that is lower than any other action, including idling, ensuring that they will never be chosen [63]. Inversely, instead of heavily penalizing disconnection, another approach is to reward connection with a so-called connection utility which depends on the predicted signal strength [8] [7]. An example of such a connection utility function is shown in Figure 2.7. Since signal strength varies with robot position, these utility functions are dynamic and change for each configuration of robots. In other work, risk of disconnection may lead to explicit behaviors such as directly assigning another robot to act as a relay [54], or even having a dedicated robot for placing static relay nodes in the environment [58] [72]. In these works, (optimal) relay placement becomes another problem to solve.

The reactive models are fundamentally different in the sense that they do not require methods of predicting signal strength and connectivity. Instead, they rely on live signal strength measurements to adapt behavior. One approach is to use a virtual spring-dampener model, where the real-time signal measurements are converted into a force that keeps the robots connected. This the ability to configure the robots such that the assigned goals can be reached [52]. An example of the abilities of the robots to self-configure is shown in Figure 2.8. Another approach proposes to use a relay-placement algorithm that uses the signal measurements to detect when a robot is about to leave the network and dynamically assigns another robot to act as a relay [1]. This algorithm is intended to be compatible with most task assignment strategies.

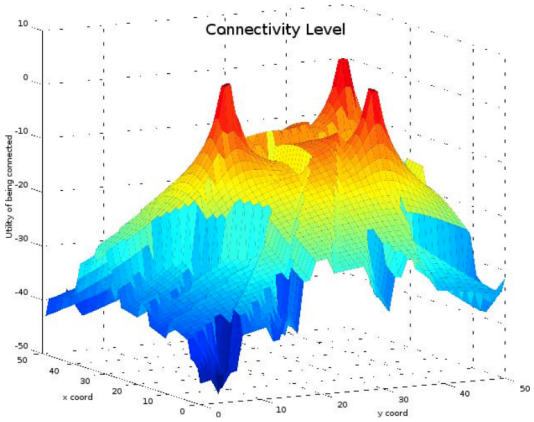

Figure 2.7: An example of a connection utility function for one configuration of three robots in an indoor environment [7]. The three peaks are at the current positions of the three robots. The signal strength function from Figure 2.5 is used to predict signal strength. The worse the signal strength is predicted to be, the lower the connection utility. The sudden drops in the utility correspond to the walls present in the environment. The effect of these types of utility functions is that positions that have strong connections to the other robots will be prioritized.

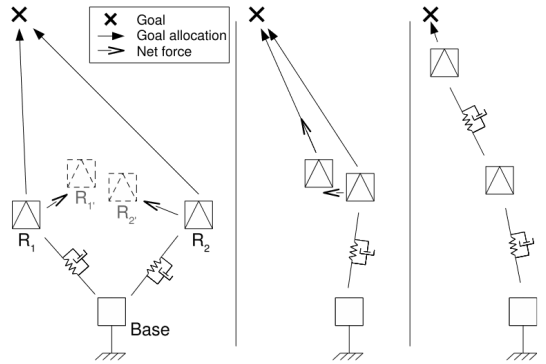

Figure 2.8: Sequence showing how robots can reconfigure themselves based on net forces that are determined on the signal strength between the robots [52].

The advantage of these reactive methods is that they do not require a prediction of signal strength. However, a robot will only figure out that it will be unable to reach the target while it is approaching it. This means that it will have to wait for another robot that has moved towards its own task to reposition itself in order to extend the network, leading to suboptimal behavior such as robots driving longer distances and exploration taking longer than could be the case with the predictive methods. Especially for search and rescue cases where speed is a critical factor, having robots wait on one another unnecessarily is highly undesirable, thus predictive methods seem to be a better fit.

2.4.3. Task Assignment

Task assignment for multiple robots is concerned with distributing the identified tasks amongst the robots and is referred to as Multi-Robot Task Allocation (MRTA). Specifically, MRTA is a supervisory intelligence level in the control architecture that adds intelligence to the system, allowing it to perform concurrent tasks through the collective behavior of the robots [16]. MRTA is a type of combinatorial optimization that consists of finding the best solution in a large, but finite discrete search space. Solving MRTA optimally is generally not feasible as it is an NP-hard problem, making it too costly to solve for real-time, dynamic MRS missions [60] [76] [16]. Due to this costliness, the main research with respect to MRTA has been focused on developing algorithms that are near-optimal [60].

Task Taxonomy

In order to understand the type of MRTA problem in this use case, the type of tasks encountered must first be defined. To aid in this aspect, a taxonomy can be used that was first introduced by [30] and later expanded by [41]. This taxonomy first determines the level of interdependence of a task, after which they can be sorted based on the following three axes:

- **Single-Task or Multi-Task:** Robots can execute one task at a time (**ST**) or multiple tasks at once (**MT**).
- **Single-Robot or Multi-Robot:** Tasks require a single robot (**SR**) or multiple robots (**MR**).
- **Instantaneous Assignment or Time-extended Assignment:** Each robot executes one task without future plan (**IA**) or a sequence of tasks is assigned to a robot over a planning horizon (**TA**).

The level of interdependence of the tasks is divided into four types as presented in Figure 2.9. First, when tasks are independent of one another, a task can be completed in any order without impacting the utility of the other tasks of any robot. Second, when tasks have in-schedule dependencies, the utility of a task for a given robot depends on tasks that it has completed earlier. When tasks have cross-schedule dependencies, the utility of a task for a given robot depends on tasks completed by other robots. Finally, tasks with complex dependencies are tasks whose utility for a robot depends on the actions completed by others. However, the tasks can also be decomposed in multiple ways resulting in one task consisting of multiple sets of smaller subtasks. Instead of only having to answer the question

who should perform a task and when, it must additionally be determined which set of subtasks should be used. It is thus a combination of task decomposition and task allocation [16].

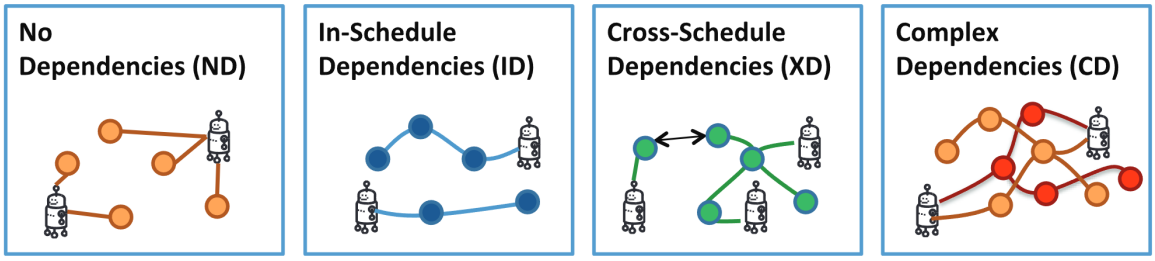

Figure 2.9: Taxonomy of task interdependence [41]. The circles denote actions, solid lines represent agents' path and arrows between tasks represent constraints. The different colored paths in the rightmost image represent the multiple possible task decompositions.

For multi-robot exploration in general, the utility of a task depends on the actions taken by other robots. This means that multi-robot exploration problems consist of either Cross-Schedule Dependendent (XD) or Complex Depependent (CD) tasks. The current use case can be classified in multiple ways. For example, a task could be assigned to a group of robots where the group members can dynamically become relays when required as is proposed by [52]. Instead of allocating tasks to groups of robots, in this work, the task type is classified as ST-SR-TA. Tasks will be positions on the map for the robot to move to. Each robot can only perform a single task at a time (ST) and a single robot is required for a task (SR). To take into account the fact that the ultimate goal locations, frontiers, can be far away from the current positions of the robots, they can be decomposed into subtasks, a sequence of which is assigned to the robots (TA). Using the taxonomy as proposed by [30] and [41], the final task type will thus be XD-ST-SR-TA or CD-ST-SR-TA, assigned in a scheme that allocates new tasks to robots after they have completed their current ones.

Requirements

To narrow down the possible candidates for the MRTA role, several requirements can be formulated similar to the procedure followed for identifying a suitable C-SLAM framework. These requirements are presented in Table 2.3.

Requirement	Description
Real-time	The system must be able to explore in real time. Thus the chosen approach must scale well to larger numbers of robots which result in larger action spaces.
Centralized	The algorithm must be centralized. As discussed in Section 2.1, distributed systems can suffer from long convergence time and bandwidth requirements in iterative optimizations, such as MRTA. Additionally, the base station will always be a single point of failure due to the connectivity constraint, regardless of the architecture used. This means that, in terms of robustness to failure, there is no difference between the architectures in this case. To simplify the implementation of the networking constraints, a centralized architecture will be pursued.
Connectivity constraint	The MRTA approach must allow for active network constraint enforcement. This means that the robots must be able to coordinate in order to determine whether an action will lead to a disconnection. This somewhat ties in with the real-time requirement since coordination between robots worsens the scaling problem of MRTA [11].
Maturity	The MRTA approach must be implementable in the sense that the presented approach has been tested on real robots or in extensive simulations. Alternatively, algorithms that are proven and widely used in other fields of study will also be accepted.
Suitable for task type	The MRTA approach must be suitable for the XD-ST-SR-TA and CD-ST-SR-TA task types.

Table 2.3: Requirements for the MRTA approach.

Overview

Modern MRTA approaches can be sorted into two broad categories [16]. The first category consists of approaches based on market mechanisms, where robots are provided with a simulated economic market to trade tasks. Robots determine their own utility for each task given a reward and cost function [60]. One approach is to auction the tasks in a variety of methods ranging from one task at a time, to parallel or combinatorial schemes. The other approach is based on consensus, where robots create a preference list for tasks based on their utility after which these lists are shared with other robots

and potential conflicts can be resolved. This results in a similar effect as the combinatorial schemes with much lower computational overhead [60]. Market-based approaches are promising and still widely researched in multiple areas, such as tasks with time deadlines for warehouse management [85]. However, they are rarely tested in real-world experiments and research into methods for complex, dynamic scenarios with unreliable communications is scarce, making them less suitable for search and rescue applications [16].

The second prominent category consists of approaches based on optimization methods. Possibly the most famous optimization method for task assignment is the Hungarian Algorithm [42]. It is still widely used, for example, in warehouse automation and has also been applied in decentralized systems. However, the Hungarian Algorithm scales poorly to larger groups of robots and tasks and is more suitable for IA than TA tasks [16]. Other approaches such as Integer Linear Programming problems have been introduced as well. However, scaling performance still remains a bottleneck affecting real-time performance. Thus, the current trend in research seems to be moving more towards using meta-heuristic methods by splitting the problem into task assignment and path planning to reduce the computational complexity [16] [51]. This trend somewhat fits with the current problem, where the path planning will be handled by the Nav2 stack and only the assignment step itself needs to be performed.

There are many meta-heuristic methods, ranging from bio-inspired algorithms such as Ant Colony Optimization or Genetic Algorithms to methods that are based on simulations of metal annealing [51]. Another well-known approximate optimization technique that is gaining in popularity in the robotics domain in general and multi-robot exploration in particular is Monte Carlo Tree Search (MCTS). It is a technique that lends itself particularly well to problems that can be represented as trees of sequential decisions and has gained in appeal after it was successfully implemented for a computer playing the boardgame go [13]. Its large number of applications paired with its versatility and promising performance, specifically for optimizations with many possible options, make it an excellent candidate for the current task assignment problem [11] [10]. Another advantage of MCTS is that it is an anytime algorithm, meaning that it can always return a feasible answer after any iteration which is refined over time. This ability is desirable in time-critical applications such as disaster response, where the quality of the solution can be traded off against response time [51]. MCTS meets all the requirements set in Table 2.3. Additionally, its simplicity, inherent scalability and popularity in multiple domains such as Game Theory, Artificial Intelligence and MRTA leads to a broad user base, increasing the implementability of the algorithm through real-world use and documentation. In the following subsection, the theory behind MCTS will be presented.

Monte Carlo Tree Search

The MCTS process is depicted in Figure 2.10. First, in the selection stage, the process starts at the base of the tree and moves through the nodes, selecting the most promising one at each step using the so called Tree Policy. This process continues until a leaf node - a node whose children have not yet been searched - is selected. This leaf node is expanded in the expansion step where it is added to the tree. Next, in the simulation step, the expected reward of the recently expanded leaf node, is determined using the a simulation called the Default Policy. The final step is then to backpropagate the newly estimated leaf node value through its branch to the root. In this stage, the number of times each node is visited is updated as well. This process repeats, iteratively creating a graph that biased towards solutions that return higher rewards. In this example, each vertical level represents one step in time and horizontally aligned nodes represent the different actions the system can take at the previous point in time. When a stopping condition, such as the number of iterations, is met, the path is selected by moving down from the root and selecting the action node based on some metric of the most visits and highest rewarded nodes [13]. This path will have to be recomputed continuously since the known environment expands during the exploration process.

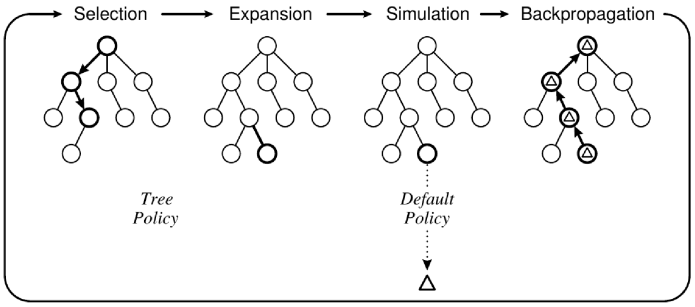

Figure 2.10: The high-level MCTS process by [13].

There are several different Tree Policies - strategies for scoring nodes - during the selection phase. The most common method by far is to use the Upper Confidence Bound for Trees (UCT) as presented in Equation 2.9 [13] [36] [10] [11], where UCT(c) is the score for a child node, c, $w(c)$ is its reward, C is a scalar constant, p_v is the number of times the parent node has been visited and c_v is the number of times the child has been visited. The reward term, $w(c)$, in general leads to greedy, exploitative behavior since it only focuses on the reward that is obtained by executing that action, whereas the second term adds a larger value to child nodes that have not been visited as often as the others. These two terms are used to balance the exploration versus exploitation behavior of the planner. The value of C depends on the exact implementation and is typically found empirically [11].

$$UCT(c) = w(c) + C\sqrt{\frac{2ln(p_v)}{c_v}} \tag{2.9}$$

The implementation of the Default Policy is highly specific to the use case. However, for games with clear end states such as chess or go, the simplest case is to make random uniform moves until the game ends and use whether the game was won or not for backpropagation [13]. In multi-robot exploration, however, the end state is reached only when all frontiers have been visited and no new frontiers are found. Since the map is not known beforehand, it is not possible to know when the end state is reached. A common solution is to simulate the planner for a fixed number of steps in the future and use the estimated reward given some function based on the number of cells that are observed [11].

There are several recent works on robot path planning with MCTS. However, the work by [11] seems to most closely match the current multi-robot exploration use case. In their work, MRTA is implemented by each robot running their own MCTS implementation in a distributed fashion as is shown in Figure 2.11. Each robot samples points from a forward facing arc, where sampled points that lie closer than 15° to an existing sample are rejected. The reward for a point in space is calculated by finding the number of previously unknown map cells that will be observed from the position. The tree is then expanded by sampling new points from the leaf node after the MCTS selection phase. The simulation phase is then run by randomly selecting new actions to perform and then finally backpropagating the estimated reward. This approach allows for planning into unexplored space, correcting the tree if new map information shows that it would to lead to a collision.

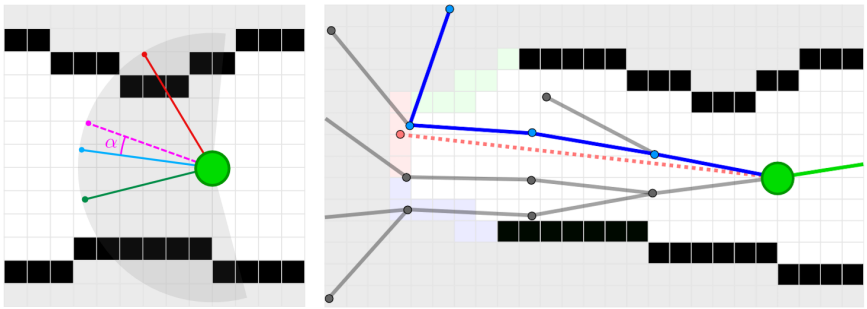

Figure 2.11: A possible distributed MCTS approach by [11]. The sampling step is shown in (a) where any points that lead to a collision such as the red solid line is rejected. A section of the grown tree is shown in (b) where the red dotted line is used to guide the tree growth to the frontier and the blue branch indicates the plan chosen by the planner.

Coordination is achieved by a robot sharing its plan when it is in communication range with others. The other robots will save the n most recent plans from each robot and randomly sample one of those for each robot. The sampled plan is then used to modify their own reward function, allowing the robot to avoid areas that the other robots are likely to visit. The authors present impressive results for their MCTS approach, outperforming two modern multi-robot planners based on rapidly-exploring randomized trees and potential field methods in terms of exploration time by 30% [74] [84]. However, they do note that their coordination approach has high computational costs, introducing delays, which emphasizes the detrimental scaling properties of coordinated or centralized MRTA.

The limitations of the approach by [11] for this specific use case is that no assumption of connectivity to other agents is made. As described in Section 2.3, this method of sporadic communication when other robots happen to be in range is insufficient in this case where continuous connectivity is a hard requirement. However, since the aim is to centralize the approach, one of the connectivity maintenance techniques as described in Section 2.4.2 can be used. Another aspect of the MCTS approach that would need reconsideration is that, currently, robots only choose actions on a forward-facing arc. Moving backwards seems to only be an option when there are no points to sample ahead of the robot. This is problematic because, while it reduces the size of the action space, robots cannot choose to idle or move backwards at any time, limiting their ability to cooperate. An example is in narrow hallways when a robot needs to wait for others to pass or when it needs to act as a stationary relay extending the network. The authors also mention that only collisions with the walls are modeled. Robots are allowed to collide without repercussions in their simulations.

MCTS is thus a valid and competitive planning algorithm for MRTA, showing promising results with respect to modern alternative multi-robot planning approaches. However, even with its favorable scaling properties, when close cooperation is attempted, it still suffers from the exponential growth of candidate actions as evidenced by the coordination strategy of [11]. A centralized approach will allow for efficient application of the communication constraint by simply refusing any configuration of robots that is predicted to lead to a disconnection, since it has a global view of all robots and their actions. Moreover, [10] has shown that a centralized approach finds promising plans faster than distributed approaches at the cost of somewhat poorer performance over time due to branching. However, centralization does mean that there is a single MCTS process that has to iterate through all the possible combinations of robot actions, instead of being distributed and run in parallel by the robots. While the hardware of the centralized node can be more substantial than on a mobile platform, it still means that additional care has to be taken to minimize the action space for each robot by removing uninteresting actions. As can be seen from image (b) in Figure 2.11, there are multiple paths running in parallel down the hallway that could be merged to reduce the action space size. Moreover, the asymmetric tree growth that makes MCTS so effective must be stimulated. This could be done in several ways such as minimizing the number of branching points in the action space or by choosing a specific Tree Policy. This will be further discussed in Section 3.3. In short, while MCTS is a highly promising planner for MRTA, especially in the centralized case, several improvements could still be made to speed up the identification of promising plans.

2.5. Summary

In conclusion, this section has investigated the research question as presented in Section 1. Firstly, the different types of multi-robot system architectures have been identified and benefits and drawbacks for each with respect to the current search and rescue use case have been presented. Secondly, the most promising frameworks for cooperative multi-robot mapping, or C-SLAM, have been analyzed. Due to the lack of objective benchmarks for C-SLAM, three qualitative criteria, heterogeneity, modularity and ad-hoc communications, have been established to judge the frameworks' performance in this specific use case. Thirdly, the various methods for modeling network communication have been discussed together with a short introduction to signal propagation theory. Finally, the research into multi-robot task assignment has been presented. The trend in this research is moving away from continuous connectivity and, instead, towards making the robotic agents more autonomous and robust to disconnections. The reason behind this is the scale problem where, if robots must be directly coordinated, finding the optimal path can take so long as to make real-time exploration unfeasible. In this case, where continuous connectivity is required, there is not a ready-made approach that can be followed which means that some adaptation is required. The research questions as proposed in Section 1 are presented below with a brief answer as a summary.

Research Questions and Answers:

- **Question**: How should the robots be organized?
- **Answer**: Complex, iterative optimizations should be centralized due to current limitations and complexities with distribution, while tasks that can be isolated to individual robots should be distributed to improve the scalability of the system.
- **Question**: How can the robots' local maps best be merged into one global map?
- **Answer**: Cooperative mapping can be performed by using a C-SLAM framework. maplab 2.0 is the most promising C-SLAM framework for this use case due to its proven track record in complex, real-world environments while meeting all three qualitative criteria.
- **Question**: How can the network connection be modeled in a dynamic scenario where the robots can extend network coverage?
- **Answer**: The most promising approach for mobile ad-hoc networks consists of using a path loss model to predict signal strength. Path loss models are empirical models that offer a good balance between the computational complexity, the environmental information required and the accuracy of the model.
- **Question**: Which task assignment algorithm is suitable for real-time performance in environments where the number of task combinations scales exponentially with the number of robots, while extending the communication range of the robots through cooperation and respecting the networking constraints?
- **Answer**: In this research field, the primary problem seems to be the scaling issue where, due to the number of possible combinations of robots, the time required find the optimal solution makes real-time operation unfeasible. Monte Carlo Tree Search (MCTS) is a graph search algorithm that is widely used in systems that play complex board games, such as chess or go, due to its ability to deal with their large numbers of possible actions. This ability, in combination with its promising performance in recent robotics applications, make it an excellent candidate for this system.

3

Proposed System

An overview of the proposed system is presented in Figure 3.1. The main concept is to create a heterogeneous multi-robot system that is able to integrate multiple types of robots such as aerial and ground robots to make use of their specific strengths and cover the shortcomings of any one platform. The system architecture chosen is hybrid as it features centralized and distributed processes. This hybrid architecture is advantageous specifically in the case of required continuous connectivity. While, in theory, distributed systems are more fault tolerant and possess better scaling properties than centralized ones, in practice, as explained in Section 2.1, this is currently not realized especially in iterative optimization schemes as are common in C-SLAM and coordination strategies for multi-robot exploration [11]. Instead, they often perform worse, being characterized by longer convergence times and higher bandwidth use [44]. In this specific case, the lack of a single point of failure − one of the main benefits of distributed systems − is reintroduced by the connectivity constraint, somewhat limiting the full benefit of distributed systems.

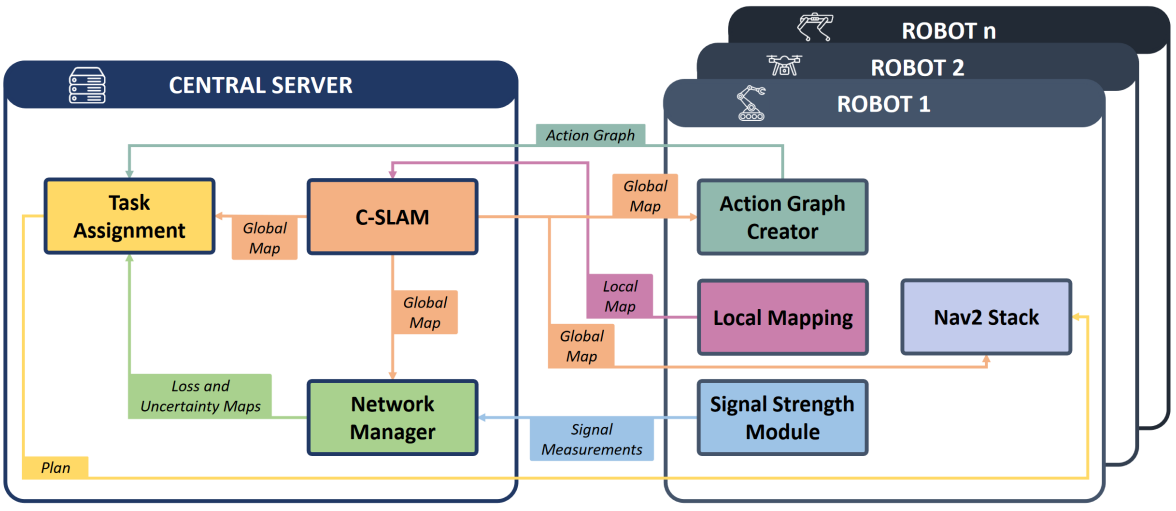

Figure 3.1: Overview of the proposed system showing the centralized and distributed components and the information they exchange.

For this reason, the processes that will be centralized are performing C-SLAM, running the Network Manager and coordinating the exploration process by assigning tasks to the robots. The chosen C-SLAM framework is the proven and flexible maplab 2.0. The Network Manager manager consists of an update and prediction step that uses a novel adaptation of the path loss models. It uses actual signal measurements to improve the network estimates over time. Finally, the task assignment is performed by

running Monte Carlo Tree Search (MCTS) using a heavily pruned action space representation referred to as an action graph.

The distributed processes are the navigation, the local mapping and the action graph creator. Navigation will be handled by the Nav2 stack as described in Section 1. Local mapping is part of the C-SLAM algorithm and creates the robot's local map sending it to the server for merging. Finally, the action graphs are a heavily pruned graph representation of the action space allowing for better scalability with multiple robots by reducing the number of system state combinations.

In this section, the different components of the full system will be presented together with the chosen approach to implement them. First mapping and navigation will be discussed, followed by the network model, then, the creation of the action graphs and finally the task assignment algorithm.

3.1. Mapping and Navigation

The C-SLAM method that is chosen for this system is maplab 2.0 [21]. In addition to obtaining the best score in the comparison in Section 2.2, maplab 2.0 has been sucessfully implemented in various complex, real-world scenarios [72]. Finally, its documentation in literature as well as code repositories. Its flexibility does not only extend to the ability to work with different sensor modalities, it is also a proven platform for heterogeneous robots combining ground and areal robots.

maplab 2.0 is a centralized C-SLAM framework, where the robots perform single-robot SLAM locally and send their local map to a central server which attempts to merge these local maps into a global one. Its system diagram is presented in Figure 3.2. The server side map merging of maplab 2.0 will also run on the central server as shown in Figure 3.1, providing a global map in the form of an Occupancy Grid as output to the robots and the network manager.

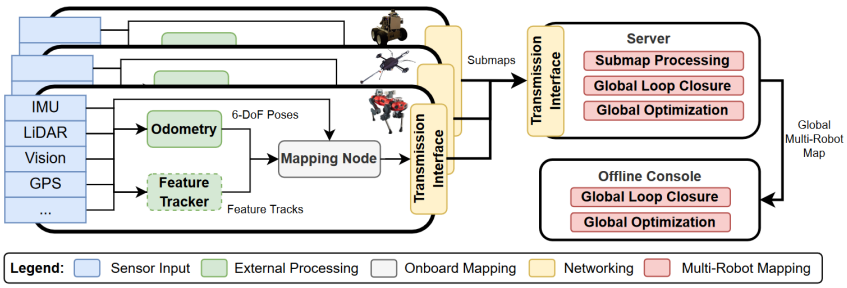

Figure 3.2: maplab 2.0 C-SLAM framework [21].

As discussed in Section 1.2, navigation will be handled by the Nav2 stack [53] [47]. The Nav2 stack has many features, options and plugins. However, for brevity, the ROS implementation is abstracted away, only showing the high-level functionality.

Tasks in the form of goal positions will be presented by the central server to each robot. The robots then individually use the Nav2 stack running locally, navigating on the global map, to find a valid route from the current robot position to the goal. This route is called the global plan and can be found using different path finding methods. Concurrently, a global costmap encourages the path finder to prioritize areas far away from obstacles by adding cost to a proposed plan if it moves the robot closer to them.

When the next plan has been found, the Nav2 stack provides the robots with the correct velocity commands to follow the trajectory. While the robots follow the global plan, a dynamic window approach is used to locally adjust the global plan in close proximity to the individual robots to account for new map information and dynamic obstacles. Should a robot get stuck, the Nav2 stack also features recovery behaviors such as reversing a short distance or rotating.

3.2. Network Model

The choices for network models, as described in Section 2, consist of active and reactive models. The active models attempt to predict network strength for use during the planning phase, whereas reactive

models use real-time measurements to modify the system behavior. For this system, the choice has been made to integrate an active model since reactive models do not take network strength into account during the planning phase. This can lead to sub-optimal performance when robots are assigned to tasks they will not be able to perform leading to more robots being required than is strictly required by, for example, assigning a task to multiple robots to ensure that there are spare robots that can act as relay if necessary.

When it comes to active models, the choice is between Gaussian Process (GP) regression and path loss models. While GP regression can fit a function to the resulting network strength when the network access points are static and robots' movements do not impact the network coverage, it is less useful when the underlying network topology keeps changing due to the robots acting as network access points, invalidating the function continuously. Moreover, they are computationally expensive to implement. A desirable benefit that is inherent to the GP process, is that it provides an uncertainty estimate of the prediction. When it comes to path loss models, they are computationally lightweight, however, they are currently overly conservative due to the use of a static wall attenuation factor W_f. In short, the method for modeling signal strength must be less unnecessarily conservative than the current path loss models, while maintaining its validity for changing network topologies. Finally, it must also provide some measure of the uncertainty of a signal strength prediction.

The model chosen to estimate signal strengths is an adaptation of the path loss model that is currently used in robotics. To avoid the overly conservative signal strength estimates that are currently the norm in path loss models, the initially conservative wall loss factor, W_f, will be updated based on measured signal strengths instead of being set to a fixed constant beforehand. In addition to being less conservative than a preset constant, estimating the wall attenuation factor of walls that the signal propagates through can isolate the effect of changing network topologies since the materials the wall is made out of do not change with respect to the robots' positions. The network strength can then be predicted by using Equation 2.8 and the estimated wall loss factors. The rest of this chapter consists of three parts. First, the method by which the relevant wall sections are identified will be presented. After which, the chosen strategy for updating the estimated properties of the relevant wall sections is discussed followed by the prediction mechanism itself.

3.2.1. Identifying Intersections

Before anything can be updated or path loss predictions can be made, the wall segments that are intersected by the signal path must be identified. This is done with the aid of a wall map that has the same dimensions as the global map created by C-SLAM. In this map, the obstacles that are believed to be part of a wall are separated from noise such as legs of tables and chairs. This process is displayed in Figure 3.3. First, the unknown space is removed from the map such that the only value that is not zero is an obstacle. The binary map is dilated to increase the thickness of each obstacle with the aim of joining any broken wall segments together. It is then eroded to remove the artificial scaling without breaking the joined wall segments. The result of this dilation and erosion is shown in image (b) in the figure. Note that in image (a) showing the Occupancy Grid, several walls consist of broken segments that have been joined in image (b).

In order to judge whether a is noise or a wall, the obstacle cells are sorted into their continuously connected areas, or contours, as is shown in image (c). The different colors represent different continuous areas of obstacle cells. Walls tend to be long and slender while noise tends to be relatively square and small. Additionally, a bounding box that contains a wall might have a square footprint, such as those who connect one or more rooms, however, it will be relatively sparse. This means that two characteristics can be used for filtering. The first is the aspect ratio of the bounding box around the contour. If the aspect ratio is close to one, the obstacle will have a square footprint which suggest that it could be noise. The second characteristic is the ratio of occupied to free cells inside a bounding box, also called the extent. If the extent is small, there is a lot of free space in the bounding box indicating that the contour is likely a wall. If a contour has a square footprint and a high extent, it can be discarded as noise, the other contours are accepted as walls. The final result of the filtered map is shown in image (d).

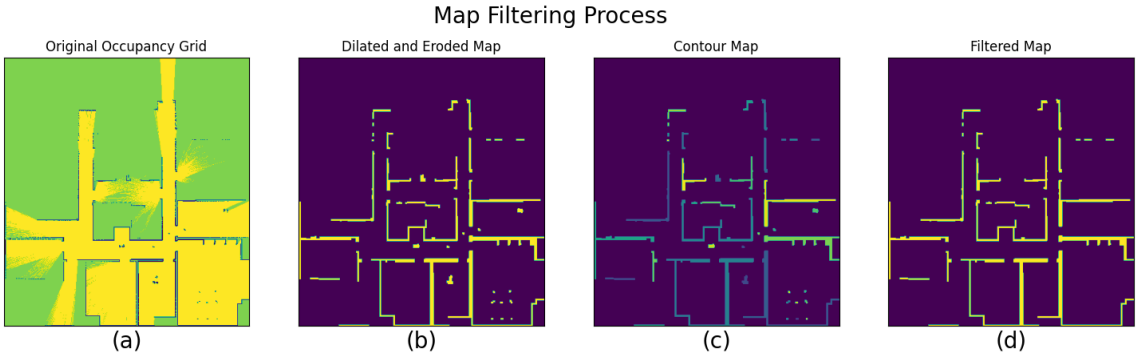

Figure 3.3: The map filtering process showing the original Occupancy Grid in (a), the dilated and eroded map that connect broken wall segments together in (b), the different continuously connected contours in (c) and the result of filtering these contours based on the extent and aspect ratio of their bounding box in (d).

Using this wall map, intersections can be identified by drawing the signal path, a straight line between the transmitter and receiver, and checking where it intersects the walls on the map. Intersections are instances that have a radius, contain a wall attenuation factor, uncertainty and position in the map frame. They are stored in a list and when a prediction is needed or an update is performed, the list of existing intersections can be queried. If one of the intersections in the list lies within the set radius of the query, the new signal is said to pass through that intersection and its values can be used for the prediction. If during the update or prediction step a new intersection is discovered, its instance is created and it is added to the list.

3.2.2. Updating Wall Attenuation Factors

A widely used tool for updating system states based on noisy measurements is the Kalman Filter [77]. Since the wall does not change over time, there are no system dynamics to model. However, the Kalman update rule can be used by itself to improve the estimate and its uncertainty of W_f given new measurements performed by the robots. The system would require an initial estimate and uncertainty. This means that when walls are identified for the first time, they will be assigned an initial, conservative W_f similar to current methods. However, they are also assigned a high uncertainty which will then be updated with the W_f estimate over time. This allows the robots to act in a less overly conservative fashion as they map their environment by improving the signal without disconnecting.

The update equations are presented in Equation 3.1 [77]. The subscript $_n$ refers to a point in time. The previous time step is denoted by the subscript $_{n-1}$. K_n is the Kalman Gain matrix which dictates how much the new measurement will impact the new signal strength estimate, x_n and the estimate uncertainty P_n. It is governed by the uncertainty of the previous estimate, P_{n-1}, and the measurement uncertainty R_n. H is the observability matrix which encodes the relationship between the state vector x_n and the measurement vector, z_n.

$$\begin{aligned} K_n &= P_{n-1}H^T(HP_{n-1} + R_n)^{-1} \\ x_n &= x_{n-1} + K_n(z_n - Hx_{n-1}) \\ P_n &= (I - K_n)P_{n-1} \end{aligned} \tag{3.1}$$

In Figure 3.4 two different scenarios are presented that the robots can encounter. Each robot will perform multiple measurements in short succession and send its average and variance to the network manager. Equation 2.8 can be rewritten to isolate the total wall loss from the measured average path loss. This is shown in Equation 3.2. The variance and wall loss can then be used in Equation 3.1 to obtain the new W_f and uncertainty estimate for each wall segment that the measurement intersects.

$$\begin{aligned} L_{walls} &= L_{measured} - L_{freespace} \\ &= L_{measured} - P_{d_0} - 10\gamma log_{10}(\frac{d}{d_0}) \end{aligned} \tag{3.2}$$

In image (b) in Figure 3.4, only a single measurement and uncertainty exist. However, there are two intersections that can have a different W_f. In the case where more intersections exist than measurements, the measurement consists of the combined impact of each intersection and can be written as presented in Equation 3.3. For this example, as shown in image (b) in Figure 3.4, let each wall be divided in three segments and the measurement intersect segments two and five where x_n is the state vector containing all intersections in the system.

$$\begin{aligned} L_w &= (x_2 + x_5) + noise \\ &= Hx_n + noise \\ &= \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + noise \end{aligned} \tag{3.3}$$

From Equation 3.3 it follows that the observability matrix, H, is a row vector that contains zeros except for the indices of the wall segments that are intersected by a measurement. There it contains ones. This leads to HP_{n-1} and Hx_{n-1} being equivalent to the sum of the previous wall segment uncertainties and the sum of previous W_f factors respectively. This holds since R_n, the variance of a measurement and z_n, the measured signal strength are scalar values and not matrices. K_n will thus be a vector with a length equal to the state vector x_n. Each index holds a value, k with $0 \leq k \leq 1$ that determines how much of the total L_{walls} is attributable to that wall segment.

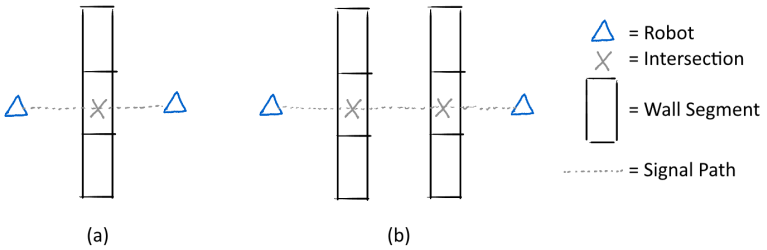

Figure 3.4: Example of a measurement between two robots being used to update the W_f of a wall section. In the left image (a), a measurement intersecting a single wall is depicted, whereas in image (b) on the right, the measurement intersects two walls.

3.2.3. Predicting Path Loss

To provide an estimate of path loss, the W_f and W_σ can be used in combination with the length of the signal path. When a signal quality estimate is required, a straight line, representing the signal path, is drawn and, using the method described in Section 3.2.1, the relevant intersections are identified. Their W_f can then be entered into Equation 2.8 together with the distance between the two points to obtain the estimated signal strength between the two positions. Additionally, the network manager is set to predict poor signal strength if the sum of the intersections' uncertainty, W_σ, is above some threshold. When only using the intersections, the unknown cells are not taken into account since they are based on a binary map where only the walls have a value other than zero as shown in image (d) in Figure 3.3. Since it is not known whether these cells are free or occupied, the number of unknown cells that the signal path crosses, u, is scaled by a constant, a, and added to this uncertainty estimate as well. The uncertainty of the signal strength estimate through n intersections is presented in Equation 3.4.

$$SignalUncertainty = \sum_{i=1}^{n} W_{\sigma_i} + a \cdot u \tag{3.4}$$

In conclusion, the network model used is a novel expansion of currently used path loss models. Instead of using an overly conservative constant for the impact of walls on signal propagation, signal strengths

are used to update the W_f factors using the Kalman update step, allowing for more accurate network modelling and, with it, less unnecessarily conservative exploration behavior.

3.3. Action Graphs

A significant problem to solve is narrowing down the set of possible actions, or action space, for each robot. If this is not done, there is an impractically large number of actions to choose from, making any type of real-time optimization unfeasible. The action space must thus be narrowed down to contain as few irrelevant or similar actions as possible. The first step is to identify positions of interest. The most widely used form is to identify frontiers as presented in Section 2.4.1 and will be used here as well. Since close coordination of the robots is required to maintain connectivity, using the frontiers as tasks directly is unfeasible since they can be arbitrarily far away from any robot, making it difficult to manage connectivity without more granular control. Moreover, due to their different path lengths, robots would need to wait while others complete their tasks before creating a new plan. To address these coordination and the timing issues, the use of an action graph as a complete yet sparse representation of the complete action space is proposed.

The main aim of the action graph is thus to assist with task assignment by narrowing down the action space of the robots. The two identified methods of generating an action space that also allow for planning network maintenance is to either divide the map into a grid where the actions available from a cell are its eight direct neighbors, or to sample points on an arc ahead of the robot on a fixed distance. Sampling points on the map is the more efficient of the two since it removes many uninteresting actions from consideration, making the action space more sparse. This is crucial, because having more actions to choose from will slow down the task assignment step. In the case where the robots need to coordinate, this is especially problematic since the number of possible combinations of actions scale exponentially with the number of actions available to each robot. This quickly becomes problematic, making real-time performance unfeasible [11].

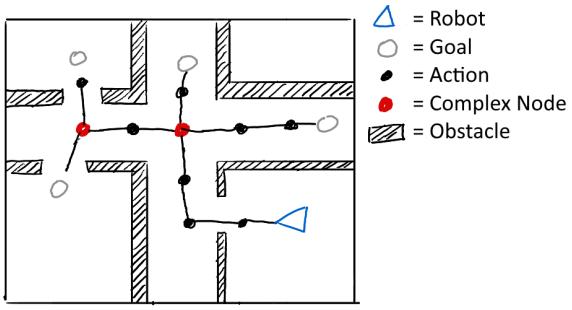

Figure 3.5: The proposed action graph structure showing the robot and its reduced action space.

The proposed action graph structure is presented in Figure 3.5. The robot, represented as a triangle in the figure, can move along the nodes in the graph, choosing from the nodes that are directly connected to its current position. In this example, a node only holds a position for the robot to move to. However, in more complex scenarios, each node can be augmented with more specialized actions specific to the individual robot such as opening a door. From the image it can be seen that from each node, except for the root, at least three actions can be taken. The robot can move forwards and backwards along the graph as well as standing still. This last behavior is required when multiple robots move closely together in tight spaces such as doorways, or when a robot needs to act as a network relay for others. One important advantage of this proposed graph structure, besides reducing the number of possible actions, is that in the graph structure as presented in the image, there are only two positions that require the robot to make a complex choice about which goal to pursue. Keeping the number of complex nodes, shown in red in the image, to a minimum will allow for a more narrow exploration of actions over time, making it easier to discriminate between promising and poor actions with respect to their future utility. This process will be further elaborated on in Section 3.4. The rest of this section will present the process of obtaining the action graph from the global map.

Instead of creating the action space live during the task assignment as is currently done, it is created and heavily pruned beforehand in a distributed fashion by each robot. Creating the graph before the task assignment stage, instead of sampling individual nodes during assignment, allows for more accurate modeling of a node's usefulness as will be presented in Section 3.4. And since the action graph is specific to each robot and only requires the global map as input, distributing this task is straightforward with respect to the centralized approach and it reduces the impact on performance when using multiple robots. Moreover, since the intent is to use different types of robots, each robot will require its own action graph tailored to its workspace and the actions it can perform. For example, if the system consists of one aerial drone and one quadruped with an arm on its back, the aerial drone will have no problem flying over obstacles such as a desk, while the quadruped can perform unique actions such as opening doors. This diverse set of capabilities means that each robot requires its own action graph.

The graph creation process is displayed in Figure 3.6. First, each robot finds a path from its current location to each goal node using its previous action graph as a seed for path planning. The paths are then segmented into sections of constant length, where each node represents an action the robot can take and the edges represent the sequential nature of the actions. After the segmented paths are created, nodes that are close together must be merged to further reduce the number of nodes in the action space using the previous action graph as a template. In this way, each robot's action graph will be incrementally grown during the exploration process. The rest of this section is divided in three sections with the first presenting the method chosen to identify the goals, followed by the path finding process and finally the segmenting and merging of the paths are presented in one subsection.

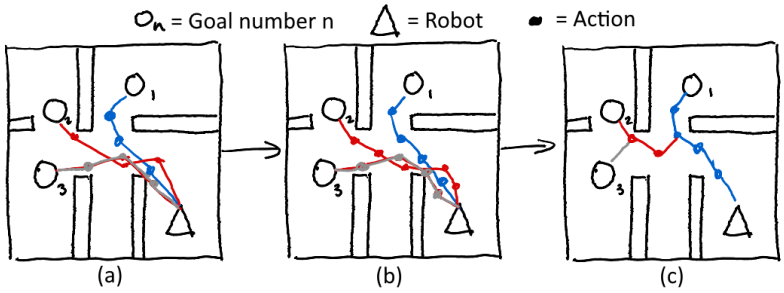

Figure 3.6: The action graph creation process where in image (a) the individual paths to the goals are found. In (b), the paths are segmented into constant sections and in (c), the paths are merged.

3.3.1. Goal Identification

Since the goal is the exploration of an unknown environment, suitable goal locations for the robot to move towards are frontiers located on the border between free and unknown cells. These frontiers can be found using different methods, however, the method chosen here is to use the same contour detection approach as is used to filter the map for the network manager. The process is displayed in Figure 3.7. First, the contours of area of the combined free and wall cells with in relation to unknown space are found shown in (b). The obstacle cells are then identified in (c) and subtracted from (b) in (d). This leaves the frontier area between free and unknown space. Image (e) shows the centroids of these frontier areas which will be referred to simply as frontiers and used to direct the exploration process.

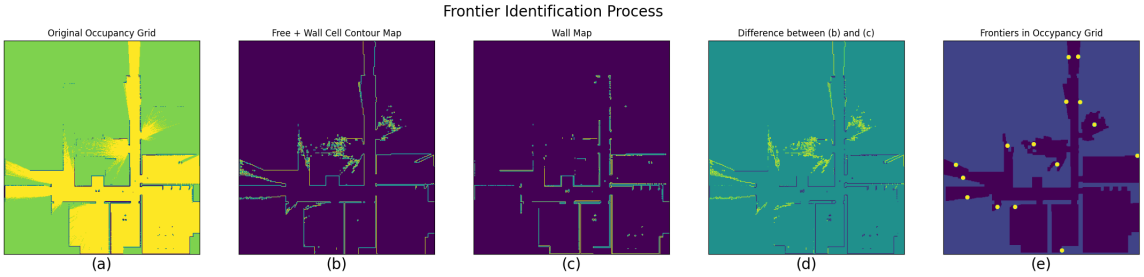

Figure 3.7: The frontier identification process shown in five steps starting with the original Occupancy Grid in (a). The contours between the combined wall and free cells and the unknown cells in (b), and the contours around the walls in (c). The difference between (b) and (c) which represents the border between free and unknown space is shown in (d) and the centroids of these borders are accepted as frontiers shown as yellow disks in (e).

3.3.2. Path Finding

Finding the paths from the robot to each frontier, as shown in image (a) of Figure 3.6, is accomplished by using an adapted version of the widely used RRT* algorithm [35]. While RRT* has excellent performance in terms of speed, it can encounter difficulties in tight spaces such as doorways which are common in indoor environments. The reason RRT* can struggle with these environmental features is that the tree is grown by attempting to connect the new node to a node in the tree that is closest. However, in environments with thin walls and narrow doorways, the closest nodes are often on the other side of the walls between rooms. Since there is only a small space close to the doorway where a sampled node will result in a successful connection, this can lead to RRT* getting stuck without having covered all rooms. To prevent this and further speed up the path finding, the standard RRT* will be modified in three key ways which are displayed in Figure 3.8. To reduce the time required to find the paths, instead of running the RRT* process from scratch each time, at the start of the path finding step, the action graph from the previous time step will be used as a seed for path finding.

First, instead of only considering the closest node for connection, the n closest nodes will be tested for validity, increasing the number of nodes that can be used to grow the tree before needing to resample. This already prevents the algorithm from getting stuck. Second, to bias the sampling stage more towards uncovered areas and decrease the time required for an acceptable solution, a disk with radius r will be cut out of the sampling space. This increases the chance that a point that is further from the current tree will be sampled. Note that increasing r too much will limit the optimization of the paths due to a lack of sampled nodes to rewire the graph with. Image (c) in Figure 3.8 shows an exaggerated case for illustrative purposes. Finally, instead of only adding a single node at a time, a greedy routine will be added to RRT* where as many nodes as possible will be connected to the tree in the direction of the sampled node. This is somewhat analogous to variable step size RRT variants such as the work by [46]. However, by taking large steps, doorways can be skipped since nodes sampled from inside of a room have no node in the hallway to connect to, resulting in the algorithm resampling often and slowing down. Adding more nodes with a spacing smaller than a typical doorway, as is proposed here, ensures that the tree will grow rapidly down hallways and through doorways. Experimental results comparing regular RRT* to the modifications discussed above will be presented in Section 4.2.1.

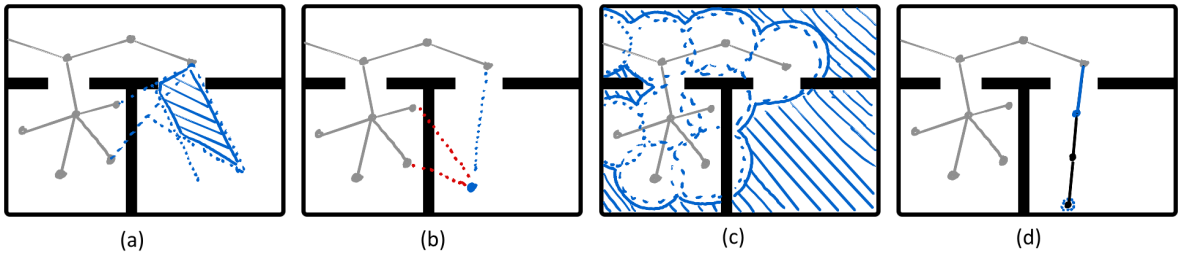

Figure 3.8: The three modifications made to the standard RRT* algorithm. Image (a) shows the area in blue where a sampled node will lead to a successful connection for standard RRT*. Image (b) shows the first adaptation where checking more than only the closest node for validity can lead to more accepted samples, leading to any sampled node that can be connected without collision to the node in the hallway being accepted. Image (c) shows the modifications made to the sample space where the blue area can be sampled from, biasing tree growth towards new areas further away from the current tree. Image (d) shows the greedy growth routine where the sampled node in the dotted blue circle would previously only result in the solid blue node being added. Now, as many nodes as will fit are added as well to increase tree growth down narrow hallways and through doorways.

3.3.3. Path Segmentation and Merging

The paths that each robot has found must now be segmented in order to create actions of constant length that can be merged. The segmentation not only helps with planning and timing by standardizing the distance between each task, it also aids in merging the graph more smoothly. Merging is performed by specifying a step size, s, and, for each edge in the paths from the base to the goals, calculating how many steps will fit in it. The extra nodes can then be added and connected, resulting in a path from the base to the goals with a smaller, near-constant step size. The reason why segmentation is performed after the path finding stage, instead of specifying the smaller step size while finding the paths, is that a small step size during the path finding leads to slower path finding due to more nodes being required and rewired at each step. In contrast, segmenting only the few paths that are required can be done quickly. Note that this only returns actions of approximately equal length. While this is judged to be sufficient for now, in Section 4 it will be investigated if this leads to robots having to wait for others to finish due to them being assigned shorter tasks. The segmenting step is shown in image (b) in Figure 3.6.

After the segmented paths are created, nodes that are close together must be merged to further reduce the number of nodes in the action space and bundle the important decisions into a few complex nodes, which results in a graph structure as presented in image (c) of Figure 3.6. Similar to path finding, instead of creating the final action graph from scratch, the previous action graph can be used as a template to connect the new path segments to. This has the added benefit of resulting in a persistent action graph where the robots will always be able to perform actions that were open to them during the entire exploration process. The reason why this is desirable is that RRT* only finds the shortest path to the goal without taking the network constraints into account. In the case where the action graph only consists of the shortest paths from the current position to the goal, a complex indoor environment can lead the shortest path to a frontier leading to a disconnection, while a longer path would not. This problem is somwhat migitgated by expanding the action graph iteratively, since the robots can always move back to a previous position from which the goal position is reachable given the networking constraints. It must be noted that the persistent action graph does not guarantee that a solution is found in all of these complex cases. It can still be the case that from an end node, the shortest path to a goal goes through an area that cannot be covered by the robot system. It is left for future work to improve the action graph process and prove that it is guaranteed to find a solution if one exists. One starting point would be to find multiple routes to each goal instead of only the shortest one. The challenge in that approach would be to identify an appropriate stopping condition that can guarantee that all significant paths have been found.

Merging duplicate or similar nodes will reduce the branching problem during task assignment by decreasing the total number of options during task assignment. To accomplish this, each path is merged in sequence. It starts a path's goal node and moves back to the root. Each node in the path is added to the action graph until a previously added node can be connected to the previous action graph. A node can be connected if it is within some, collision free, distance of the previous action graph. The

resulting graph becomes the base for the next path. The next paths can then be merged with this new, expanded tree in the same way, ensuring that as few similar nodes are accepted into the action graph as possible given the paths to the goals. This process is shown in image (c) of Figure 3.6.

In conclusion, the function of the proposed action graph is to heavily prune the action space for each robot, attempting to mitigate the branching problem by reducing the number of possible actions that the task assignment stage must choose from. This branching is a significant problem for multi-robot systems that are strictly coordinated since it quickly leads to real-time performance becoming impossible. In the proposed approach, each robot creates its own unique action graph in three stages. Firstly, it finds the shortest paths from its current position to the frontiers. Secondly, it segments these paths to create actions of constant length and to ease the final step where the different paths are merged together to create the final action graph. This process is iterative, where the previous action graph is used as a starting point for the path finding and merging stages.

3.4. Task Assignment

Monte Carlo Tree Search (MCTS) has been chosen as the algorithm for the task assignment stage. The main reason for this choice is its simplicity, performance and promising results in recent studies [11]. Its simple instruction set makes adding constraints straightforward and its motivation for certain actions are transparent and easily adjustable if necessary. A major problem of task assignment for multi-robot exploration, especially when close cooperation is required, is that the number of possible actions quickly grows out of control, even if a minimal action graph such as the one proposed in Section 3.3 is used. MCTS is specifically suitable for these types of problems because it moves iteratively through the action space and only requires utilities and constraints to be calculated for configurations that are directly reachable from the current configuration in the process. This results in real-world performance that is orders of magnitude faster in comparison to exact methods [36]. Additionally, MCTS is an algorithm that has found widespread use in game theoretic settings and this adoption increases the chance of successful implementation due to the abundance of reference material.

In this centralized case, each node in the MCTS tree is a set of one action per robot for all robots on their respective action graphs as shown in Figure 3.6. Each robot starts the planning stage at their root node, denoted by 0 in the figure. From their they can either move to one of the neighboring nodes on their action graphs or idle on their current node. Running MCTS centrally, means that the total number of children of an MCTS node is equal to the product of the number of actions each robot can take. Thus, if there are five robots that can each take three actions (moving one node forwards, backwards or idling on their action graph), the current MCTS node has 234 children. These 234 children can then each also have at least 234 children, illustrating the scaling problem when performing centralized task assignment with multiple robots and highlighting why minimizing the action space for each robot is vital. The proposed method for generating a minimized action space is described in Section 3.3.

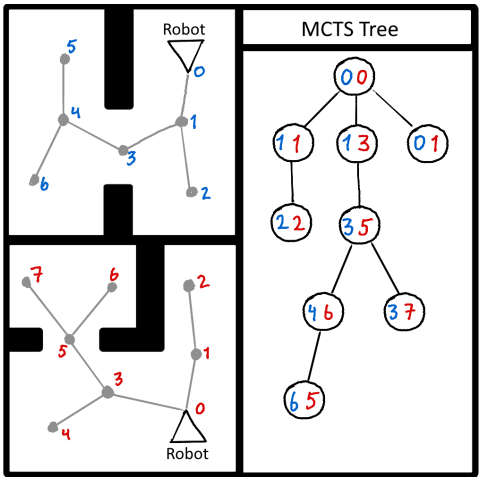

Figure 3.9: Schematic depiction of centralized MCTS tree creation with two different robots. The action graphs of both robots are shown in the two images on the left where their individual actions are denoted in red and blue starting at their root denoted by 0. These actions are then combined to create the global MCTS tree. A section of the tree during creation is shown on the right.

While the scaling problem would be mitigated by distributing the MCTS process to the robots, the reason it is faster is because they do not take one another's actions into account. As shown by [11], although distributed MCTS can perform well with multiple robots, as soon as coordination strategies are introduced, adding more than two or three robots leads to lackluster performance. The problem is similar to the PGO process in C-SLAM. It is not trivial to efficiently distribute an iterative optimization and currently, it leads to a longer convergence time and more bandwidth use than centralized solutions [44]. Instead of distributing the MCTS process and then designing coordination strategies, a centralized scheme is proposed here that runs on the same server as the C-SLAM process.

3.4.1. Problem Formulation

While the current action graph formulation can hold any type of spatial or action type information, the current implementation is intended for 2D exploration where each state $x \in \mathbb{R}^2$ is represented by a node in the action graph. Let x_i^r denote one of the m actions that robot r can take from its current state x_0^r. The root state of the full system, X_0, can then be described as the set of all robot root states for all n robots such that $X_0 = \{x_0^0, ..., x_0^n\}$. A possible new system state from the root state, X_0, can then be denoted as $S_i \in S(X_0)$ where $S(X_0)$ represents the space of all available states given the root node X_0. The state space $S(X_t)$ contains all possible system states X given the system state X_t at time t. A path from the system's current root node, $p(X_0)$, can then be defined as a sequence of system states $\{X_0, X_1, ..., X_T\}$ where T is the planning horizon. Let $P(X_t)$ represent the space of all possible paths originating from system state X_t. As is presented in Equation 3.5, the problem can then be formulated as finding a path $p(X_0) \in P(X_0)$ that has the highest expected reward from all other paths in $P(X_0)$ for a given reward function $R(X_0, p)$. Once this path has been determined, the robots can move a set number of nodes down that path before executing the MCTS algorithm again.

$$p(X_0) = \arg \max_{p \in P(X_0)} \mathbb{E}[R(X_0, p)] \tag{3.5}$$

3.4.2. Constraint Enforcement

For now, the two constraints that are placed on the task assignment stage are to guarantee communication and to prevent collisions and allow for coordination through doorways and narrow hallways. First, a state X is only viable if its constituent robot states x are within communication range with one another and the base station. Second, no two robot states x in the system state X can lie within some critical distance, d_c, of each other. Connection is ensured during the expansion step when the children of a leaf node, X_L, are indexed. If a child consists of a configuration of robots that is predicted to be disconnected, it will be removed from the MCTS tree. Determining the network connection of a state is

done by generating a network graph where the base station and the robots at their new positions are the nodes of the graph. The edge between two nodes represents the signal path and is only added if their signal strength is predicted as sufficient using the method described in Section 3.2. After all signal strength predictions have been made, the graph is checked for disconnections. If the graph is connected, the system state is also network connected and the first constraint is met. System states that contain robot states that are too close together are removed from the MCTS tree as well.

3.4.3. MCTS Policies

As can be seen from Figure 2.10, there are two policies that can be used to tailor the MCTS planner to the specific use case of multi-robot exploration. The first is the reward for each node that is used in the selection phase, the Tree Policy P_T. The second policy is the Default Policy, P_D, which is used to roughly estimate the reward of a new node with the aim of getting an overview of which nodes are worth exploring further. The Tree Policy consists of the Upper Confidence Bound for Trees equation as presented in Equation 2.9. Its implementation for this case is presented in Equation 3.6. The reward term will consist of an intrinsic reward R_{in}, which is the stand-alone worth of a node, and a reward that represents rewards obtained by future nodes, the expected reward R_{ex}. The intrinsic and expected rewards are scaled by a user-defined constant, $0 \leq \alpha \leq 1$, that balances the short-term gain of the intrinsic reward with the long-term gain of the expected reward.

$$\begin{aligned} R^{X_t} &= \alpha R_{in}^{X_t} + (1-\alpha) R_{ex}^{X_t} \\ P_T &= \arg \max_{X \in A(X_t)} \{R^{X_t} + C\sqrt{\frac{2ln(p_v)}{c_v}}\} \end{aligned} \tag{3.6}$$

Typically for robotic exploration, the intrinsic reward of a node is calculated by counting how many unknown map cells are expected to be observed when the robot executes an action by counting the number of unknown cells inside of a predetermined sensor radius [11]. In this case, since the action graph does not plan past the frontiers, a node that is not a frontier will contain very few unknown cells. Moreover, the methods for determining the number of cells that can be observed from a node is questionable, since it is likely that walls and other obstacles will limit the view of the sensor thus giving an overly optimistic estimate of one node at the cost of another. This can be observed from image (b) in Figure 2.11 in which the MCTS planner runs past the frontiers where, realistically, it can not give an accurate estimate of the number observable cells in order to distinguish two nodes.

In this case, however, since more knowledge about the wall attenuation factors of a wall segment, W_f, can lead to less restrictive exploration as described in Section 3.2, the intrinsic value of an MCTS node can be represented by an estimation of how much it reduces the uncertainty of the W_f of the segments that are intersected by the signal paths between the robots, as displayed in image (b) of Figure 3.4. The higher the current uncertainty of the wall segment, the more useful it is to explore and thus the higher intrinsic reward that should be assigned to that configuration of robots. Additionally, to emphasize the usefulness of reaching a frontier, an extra intrinsic reward of constant value is added to the reward. In future work, it can be investigated whether exchanging this constant for a more dynamic representation of information gain improves the planner. The intrinsic reward is presented in Equation 3.7, where $R_{in}^{X_t}$ is the intrinsic reward for the system state X_t, c is a scaling factor and σ_j is the uncertainty of intersection j where I is the total number of intersections for the state. N_f and R_f are the number of frontiers reached in that system state and the reward for reaching a frontier respectively. Since there are multiple robots in the system, multiple frontiers can be reached in a single system state. Finally, to prevent rewards being allocated more than once, the uncertainty of the wall segments and the frontiers in the action graph are updated for the child states of X_t with their new expected values. The updated uncertainties are only used during the MCTS stage. During execution of the plan, the real uncertainties are calculated according to the procedure presented in Section 3.2 and the new, actual, uncertainty map is provided to the MCTS algorithm when a new plan is required.

$$R_{in}^{X_t} = c \sum_{j=0}^{I} \sigma_j + N_f R_f \tag{3.7}$$

The expected reward of a state X_t, $R_{ex}^{X_t}$, is a representation of any rewards that can be obtained in the future if the actions in state X_t are performed. It will be calculated by taking the average of the rewards, R, of the direct children of state X_t and multiplying it with a dampening constant γ. This is done during the backpropagation stage in accordance with Equation 3.9. The reason the rewards must be averaged is that, since MCTS grows the tree iteratively, nodes that have been visited more often can have more branches making it seem like there are more rewards for that course of action. Alternative strategies could be to only keep the highest expected reward instead of averaging or even to include some metric of the spread of the rewards, such as its standard deviation, and using that during selection. These strategies will be left to explore in future work. When a leaf node is first expanded, it will not have any child states that have been visited and thus no expected reward from backpropagation. This is where the Default Policy, P_D, is used to simulate the reward.

As mentioned before, in games with clear end states, this simulation can be performed until an end state is reached. In robotic exploration, however, the end state is only reached when no more frontiers are present, which cannot be determined beforehand in unknown environments. A solution to this, as implemented by [11], is to simulate the MCTS for a set number of iterations past the leaf node and to count some metric of the estimated number of observed cells obtained during this simulation as the simulated reward of the leaf node. The limitation of this method is that it is somewhat unrealistic as mentioned before. Especially in complex indoor environments with many rooms, planning past a frontier will often result in using unobtainable rewards due to the many walls blocking the path. Alternatively, some metric for the distance, such as Euclidean or Manhattan Distance, between the robot poses in the state and the frontiers can be used to nudge the robots towards the frontiers. A downside of these approaches that sample their nodes live from the map during MCTS is that these metrics can be unreliable in complex environments due to the large number of doorways and hallways. To combat this, the Default Policy is based on the action graph representation as proposed in Section 3.3 which has global knowledge of the real path for a robot to each frontier. This means that the actual path the robots would take to each frontier is known and potentially unreliable metrics are thus not needed.

When the MCTS process reaches a leaf node, an estimate of that node's future usefulness is required to determine whether it will be further explored in future cycles. This is provided by the Default Policy, P_D, which is a method to provide this estimate without actually exploring the node. Usually, this means that a simulation must be performed. In this case, however, due to the action graphs containing the real distances from each robot position to each frontier, the expected reward does not have to be simulated. The estimated reward of a leaf node, $R_{es}^{X_t}$, will thus be calculated by using the locations of the frontiers and the distance on the action graph between a robot and a frontier. The nodes on the action graph that correspond with frontier positions are known beforehand and, using Dijkstra's algorithm, the distance between the robot node and the frontiers can be calculated. This distance, d, is then used to scale the reward obtained for reaching a frontier, R_f, using a function $f(d)$. $f(d)$ must sufficiently penalize the frontiers that are further away to ensure that close frontiers are not ignored in favor of many that are further away, since this would lead to many partially explored sections and robots needing to backtrack to finish the exploration. However, in the case where all frontiers are far away, the rewards must still be distinguishable enough for a robot to move in their direction along its action graph. P_D is presented in Equation 3.8, where first, the estimated reward for a single robot, R_{es}^r, is calculated, where F is the number of frontiers in the system state. R_{es}^r is then summed for each robot in the system to represent the estimated reward of the leaf node X_t.

$$\begin{aligned} R_{es}^r &= R_F \sum_{k=0}^{F} f(d_k) \\ f(d) &= \frac{1}{2d} \\ P_D = R_{es}^{X_t} &= \sum_{r=0}^{n} R_{es}^r \end{aligned} \tag{3.8}$$

From the leaf node, this reward is then backpropagated through its branch to the root node in the final step of the MCTS cycle. During backpropagation, the expected reward for a leaf node is allowed to trickle through the tree to allow the selection stage to re-evaluate the chosen path using updated

values. Relevant statistics such as the number of times the node has been visited, n_v, are updated as well. The iterative update of the expected reward of state X_t, $R_{ex}^{X_t}$, is presented in Equation 3.9. In this equation, a newly updated expected reward in the child state X_c, $R_{ex}^{X_c}$, which is either the output of the backpropagation's previous step or the Default Policy if the child node is a leaf node, is multiplied by a dampening constant γ. To prevent having to search all the child states of X_t again, the current expected reward, R_{ex}^{curr}, and the number of times X_t has been visited, n_v, can be used to update the average expected reward.

$$R_{ex}^{X_t} = \frac{n_v R_{ex}^{curr} + \gamma R_{ex}^{X_c}}{n_v + 1} \tag{3.9}$$

This four-stage process is then repeated for a number of iterations, after which the path can be extracted by starting at the root node and selecting the next node based on which child has been visited most often. This process is repeated until the planning horizon, T, is reached. T depends on how many actions the robots can execute before the environment has changed enough to require a new plan. In a complex, indoor environment, frontiers will be close to the robots. If T is set too large, the robots will reach the frontiers and then drive back since the action graph does not allow planning beyond the frontiers. This means that ideal value for T equals 1, meaning that robots will execute one action before a new plan is made.

In conclusion, the task assignment stage will perform MCTS using each robot's action graph to sample new system states from. The Tree Policy will consist of a weighted sum of the intrinsic and expected reward. The intrinsic reward, $R_{in}^{X_t}$, is a combination of how many frontiers are reached and how much the uncertainty of the signal path intersections is reduced for system state X_t. The expected reward, $R_{ex}^{X_t}$, is a measure of the future reward that can be achieved if the system reaches state X_t. It is first calculated for a leaf node using the Default Policy which uses each robot's action graph to find their distance to the frontiers. These distances are scaled and summed to find the total expected reward for a leaf node. The expected reward for each node is then updated during the backpropagation stage to also incorporate its children's intrinsic rewards after they have been searched as well. MCTS is repeated for a number of iterations after which the next most promising action, given the current system state, is extracted. This process is repeated when the system reaches its new state.

3.5. Summary

An overview of the proposed system is provided in Figure 3.1. It consists of a hybrid architecture where some, difficult to distribute, tasks are performed centrally, while others that can be run in relative isolation in the robots are distributed. The centralized tasks are creating the global map, running the network manager and the task assignment. The global map will be created by the C-SLAM framework maplab 2.0 because of its success in complex real-world scenarios. Additionally, it supports ad-hoc networking and, compared to the other investigated frameworks, it is the most flexible and heterogeneous. The network manager will be a novel addition to the path loss models, where the wall segments are updated with signal strength measurements using the Kalman update rule. This will allow the system to be less overly conservative than the current path loss models while also incorporating an uncertainty estimate. The task assignment stage will be based on MCTS, where a system state consists of each robot taking one action on their action graphs. These system states will quickly branch out of control which is why MCTS, with its iterative search, is chosen.

The distributed processes will be running the Nav2 stack, executing the local component of maplab 2.0 and creating the action graphs. The Nav2 stack will provide each robot with the capacity to reach its assigned actions while avoiding collisions with any dynamic obstacles. Static obstacles will not have to be avoided since the nodes and edges of the action graph do not pass through obstacles. However, should any errors occur during planning, the Nav2 stack will prevent critical failures due to collisions. The action graph is a heavily pruned representation of the action space of each robot. It is created in a distributed fashion since it is unique to each robot's capabilities and only requires the global map and the previous action graph to be created. The process entails finding the paths from the robot's position to each frontier on the global map, segmenting these paths into constant sections and then merging them with the previous action graph to iteratively create the final graph.

4

Experiments

To test the functionalities as introduced in Section 3, a prototype must be implemented that is as simple as possible without compromising the subsystems to be tested. The functionality that is of interest is the estimation of signal strength and its effect with respect to the conservative nature of current path loss models. Additionally, the creation of the action graphs will be tested and compared to the current approach of sampling points from the map during the exploration. Finally, the planner will be tested in several toy problems to verify the coordination between the robots. For these experiments, the C-SLAM framework is replaced by a simple map merger on the server side in combination with the ROS2 SLAM Toolbox to reduce the time required for implementation. The downside of this is that a full C-SLAM system is much more accurate and robust than merging a map based on the odometry estimates of the robots. Moreover, a C-SLAM framework does not require initial pose estimates to be provided, which is desirable for real-world operation. However, for these experiments, a map merger is sufficient to test the other subsystems. The rest of the system is implemented as presented in Section 3. The system will be implemented in ROS2 and simulations are performed in Gazebo.

4.1. Network Manager

This section is focused on investigating the validity of the network model and comparing the concept of updating wall segments to the static case. Since the exploration system will only be tested in simulation, it is important to investigate the performance of the network model separately in a real-world setting. The network model will be tested in three separate scenarios. The first test is in a static environment where the network propagation is modeled and measured in places without any obstructions between the transmitter and receiver. The main goal is to validate the path loss model for different free environments such as a room and a long hallway. This is relevant since the model will not be able to tell the difference between the two scenarios even though, in theory, the different wall configurations of an open room and long hallway should have an effect on the signal propagation as described in Section 2.3.1. Since the path loss is an empirical model that has been widely used for a long time, it should perform relatively well. However, due to its simplifications with respect to real signal propagation, it is important to know what safety factors, if any, should be included to compensate for any erroneous estimates especially due to the wide variety of indoor environments.

The second experiment will focus on validating the representation of the wall attenuation factors, W_f, and their updates using the Kalman update rule. This will be done by creating a map of a room and the adjacent hallway and driving a Turtlebot3 Burger robot [62] on the other side of the wall with respect to the base station and letting it perform several updates as described in Section 3.2. During these updates, the measured signal and uncertainty, estimated W_f and the updated W_f and W_σ will be measured to verify the convergence of the predicted signal strength to the measured value. This experiment will be performed in several configurations with multiple intersected walls between the transmitter and receiver. The main aim is to prove the validity of the wall segment representation and the ability to be less conservative during exploration than the other signal models by updating the wall segments.

The final experiment dedicated to the network manager will consist of multiple robots and a base station jointly creating the network map in a simulated environment using Gazebo. In this experiment, the impact of updating the wall segments will be compared with the current approach of assigning them a conservative constant value beforehand. The comparison will be in terms of communication range and robots required to achieve full network coverage. The reason behind the choice to simulate this procedure is the difficulty in implementing the mesh network in ROS2. The mesh network will be required for the real-world application since robots must extend the network coverage during exploration. While a candidate mesh networking protocol has been identified in the Better Approach To Mobile Ad-Hoc Networking (B.A.T.M.A.N) [56], this will be left for future work due to the complexity of implementation. Therefore, the individual elements of the network manager will be tested in the real world to investigate the validity of the model itself, whereas the full cooperative communication map creation is run in a simulation.

For the free space and wall loss experiments, the system consists of a Turtlebot3 robot and the base station which is a Teltonika RUTX14 router. The Turtlebot3 runs its turtlebot node which manages the onboard hardware such as the laser scanner, the encoders for the odometry as well as the slam toolbox and a custom ROS2 driver that measures signal strength, calculates the average and variance of several measurements and publishes them to a ROS2 topic. The Network Manager, as presented in Section 3.2 runs on the central server which is a regular laptop.

4.1.1. Free Space Propagation

To test the effect that different types of rooms have on the indoor free space propagation, several measurements without any obstacles between the transmitter and receiver will be performed in two different scenarios. The model cannot detect the difference between rooms since it is based on distance and the number of crossed walls and will thus provide identical predictions. However, as discussed in Section 2.3.1, signal reflections play a role in the received signal strengths which should show up differently in the measurements for different room geometries. For this experiment, a narrow hallway is compared to an office. The expectation is that when reflections play a significant role, such as in a hallway, pronounced peaks and valleys from constructive and destructive interference respectively should be visible in the signal strength. The results are displayed in Figure 4.1. The variance of the measurements is left out since this was negligibly small with respect to the measured values. The highest recorded variance was 2 dBm^2.

First, in an open-ended hallway that is 1.20 m wide, the base station will be placed at one end with a distance of 1 m from the wall. The robot will start at a distance of 1 m from the base station and will drive down the middle of the hallway as is depicted by the blue trajectory in Figure 4.1. The measurements will be performed at 1 m increments by driving the robot to the next measurement point, waiting for 20 seconds for the robot to settle. The robot is then allowed to collect measurements every four seconds for one minute after which the average and variance are recorded. The same procedure is then repeated for the office with a width and height of 6.4 m and 10.6 m, respectively.

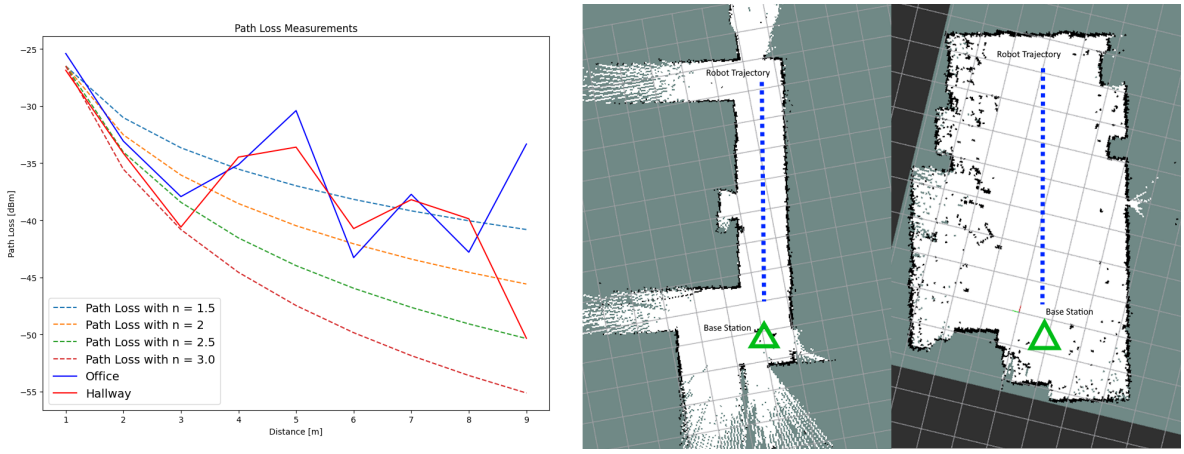

Figure 4.1: Signal strength measurement inside an office and a hallway. The left image shows the measurements compared to the path loss predictions for different path loss exponents. The middle and right images show the maps created during the experiment for the hallway and office, respectively.

From the figure it can be observed that a value of -26 dBm can be used as P_{d0} for path loss predictions in other experiments. The path loss exponent that fits the measurements best is $1.5 \leq n \leq 2$. Interestingly, except for the final measurements at 9 m from the base, the measurements for both the office and the hallway are quite similar. Since they do both show fluctuations around the modeled path loss, the reflection is present. However, they seem to be impacted by reflections in a similar way, suggesting that the the office is not wide enough with respect to the hallway to show a significant decrease in the effect of signal reflections. The reason why the final measurements diverge so strongly seems to be that, since the hallway is open-ended and the office is not, the signal reflects off the back wall back towards the robot, increasing the received signal power at the final measurement location.

Another observation that can be made is that, when comparing the fluctuations in these free space measurements with the reference W_f values from Table 2.2, the impact of reflections is larger than the W_f of most walls. Wall loss values from the table range from 2-15 dB. This could lead to difficulties when trying to isolate the effect of a wall from a measurement as is proposed in Section 3.2. The impact of reflections could lead to the same wall, made of the same material, being assigned a different W_f along its length.

In this experiment, it has been shown that prediction accuracy in free space, while providing an indication of signal strength and roughly follows the path loss model, suffers significantly from reflections and the other propagation phenomena not modeled by the path loss model. The fluctuations are in the same order of magnitude as the reference values for wall loss, making isolation of wall segments as presented in Equation 3.2 difficult. From Figure 4.1, it can be observed that the impact of propagation phenomena not modeled by the path loss model is significant on the measured loss, both in narrow hallways as well as in open rooms. Finally, the impact of these phenomena on the measured signal strength can be seen from the relatively low path loss of the final measurement in the office. The final measurement position is just 1 m away from the back wall of the room where the signal reflects back towards the robot, increasing the received power significantly. The measurements in the hallway do not exhibit this behavior due it being open-ended.

4.1.2. Wall Loss

To validate the concept of improving the network prediction by updating the wall loss, W_f, a similar experiment to the free space propagation will be performed. Firstly, the map must be filtered to remove noise and only leave the wall cells. Secondly, the correct wall segment must be identified and updated using Equations 3.2 and 3.1. Finally, the new prediction can be calculated using Equation 2.8 and compared to the performed measurements.

A map is created with the SLAM toolbox and the Nav2 stack, after which the robot is driven to the other side of the wall and the network manager is enabled. The results are presented in Figure 4.2. The first observation is that system can successfully filter the noisy map and distinguish noise from the legs of

tables and chairs from actual walls as can be seen by the noise near the base station being left out of the colored wall. The procedure for this filtering is presented in Section 3.2.1. Additionally, the system can isolate the correct wall segment, as shown in blue, and update its loss value using the procedure described in Section 3.2. In just a few measurements, the prediction has converged to the measured value and the W_σ has converged to zero as can be seen from Figure 4.2.

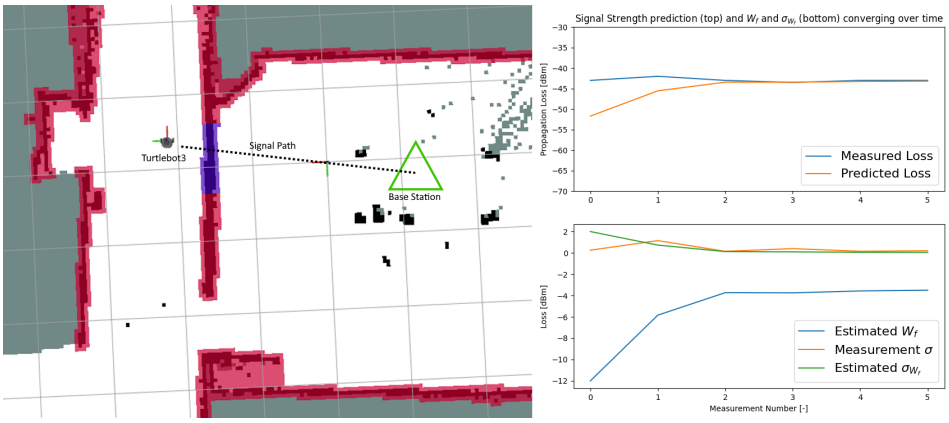

Figure 4.2: Results showing the isolation of a wall segment and the update cycle of its W_f in a real-world environment. The estimated values at measurement zero represent the initial estimates used before a measurement is available. The walls are initialized at a high loss value which is visualized as a red overlay on the wall. The wall segment that is crossed by the modeled signal path is then assigned the updated, much lower, value which results in the blue color. The graphs on the right show the update cycle for the blue intersection from the left image.

4.1.3. Communication Map Creation

To visualize the networking environment and the communication map and to illustrate the added value of updating the wall segments with the network manager, this experiment will consist of a base station and several robots in simulation. The robots will explore the environment both in terms of obstacles and signal strength. Each robot will continuously collect signal strength measurements where the base station and the other robots, if in direct communication, are the transmitters. The aim is to show how the wall segment update procedure can extend the effective communication range of the robots with respect to current path loss implementations. To simulate the network in Gazebo, a ground truth map has been created by assigning a W_f to wall segments. The robots sample from this map during operation and construct the signal strength using Equation 2.8. Noise is added to this signal strength and is then published as a measurement. The network manager receives these measurements and updates its wall segments accordingly. While this is not a realistic measurement strategy, it does show that the system is robust to noise and the relative difference between static and updated wall segments is still valid as is shown in Figure 4.2, where real-world Wi-Fi measurements are used. The results of this experiment are displayed in Figure 4.3.

It needs to be noted that this example is highly specific. The exact results depend heavily on the environment. A denser environment with more walls means that updating the wall segments will have a higher impact than in sparser environments where wall loss plays a smaller role in the overall path loss. Moreover, the results depend on the initial W_f assigned to each intersection in case of the constant W_f scenario, as well as the values entered into the ground truth map. For this experiment, the initial W_f has been chosen as 10 dB to represent a thin concrete wall as shown in Table 2.2. The ground truth map has been created by assigning wall sections a random integer value between 3 and 10 dB to represent the different types of walls. An example of a ground truth map created using this procedure is shown in image (f) of Figure 4.3.

From this experiment, it can be concluded that updating the wall segments has a significant impact on the expected communication range of the robots. As can be seen from Figure 4.3, in image (c) and (e) where the wall segments that have been updated are shown in blue, the robots will be allowed to explore in a less constrained fashion due to the increased communication range. In this example, by updating the wall segments, the maximum communication range of the robots is increased from 15 to

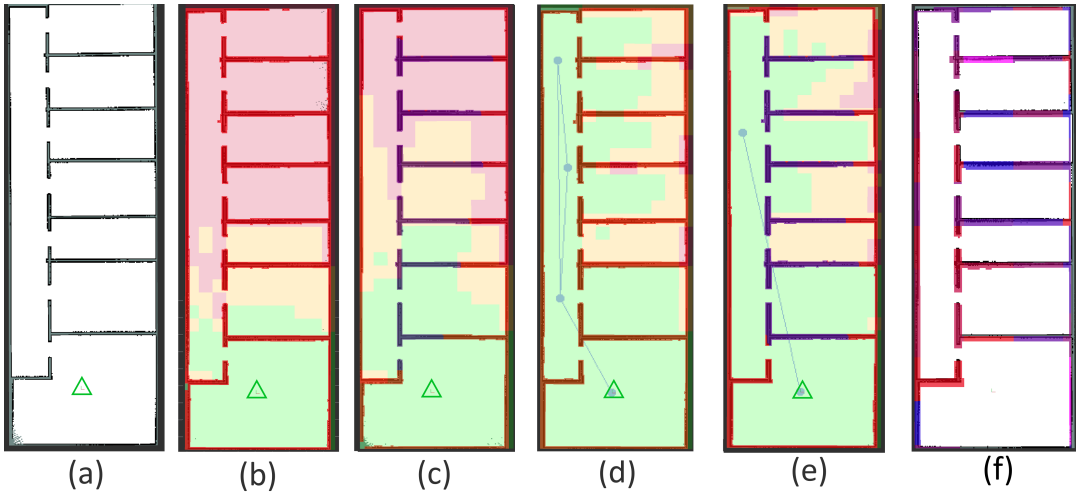

Figure 4.3: Heatmaps in RViz2 showing signal strength with and without updating the wall segments. In (a) the Occupancy Grid of the environment is shown. In (b) and (c) the communication range of the base, located at the green triangle, is shown for constant and updated wall segments respectively. The updated wall segments are shown in blue on the walls. (d) and (e) depict the minimum number of robots required to achieve full network coverage of the environment for constant and updated wall segments respectively. The positions of the robots are depicted by the blue disks while the modeled propagation paths are shown as blue lines, connecting the robots and base station.

22 m and the number of robots required, in addition to the base, for network coverage is reduced from three to one.

One issue that arose during the experiments is that wall segments are only updated as the robot passes by them. It can sometimes be the case that poor network quality is predicted. However, in order to improve the estimates through a measurement, the robot has to move into the zone where poor connectivity is predicted. This can lead to the robot not being allowed to explore an area that it would have been able to, given the actual W_f. To mitigate this, in future work, intersections should be detected independently of the update process. Intersections that have been detected and are close to others with updated values can then be updated themselves, where the measurement uncertainty is a metric of the distance between the two intersections. This will lead to the network prediction being able to provide updated estimates further ahead of the robot, thereby increasing the system's exploration performance. Additional real-world tests must be performed on the validity of updating neighboring segments since it could lead to overly optimistic estimates.

In conclusion, updating the wall segments increases the communication range of the system, reducing the number of robots required to maintain the communication network. However, the current implementation updates the network in close proximity to the robot, limiting some regions that could have been explored. To mitigate this, intersections should be detected separately from the update step and the update step should be modified to include not only the current intersection, but also its close neighbors.

4.2. Action Graph Creation

Before running the exploration framework in simulation, the action graphs must be tested. This will be done in two separate experiments. Firstly, the modified RRT* algorithm used for path finding, as presented in Section 3.3.2, will be tested and compared with the regular RRT* as a baseline. Secondly, the full action graph creation process will be presented in a Gazebo simulation, comparing it to the live sampling techniques in terms of node density. The limits of the presented action graph representation will be investigated and will be discussed in Section 5.

4.2.1. Path Finding

The main aim of this section is to validate the modifications that have been made to the RRT* algorithm to improve its performance in dense, indoor environments as described in Section 3.3.2. The experiment will consist of running both the baseline RRT* and the modified versions ten times on the same map that was used previously, where a goal node has been placed in each room as shown by

the yellow disks in image (a) of Figure 4.4. The many doorways and corners make this map suitable since it will highlight the problems encountered by the baseline RRT* algorithm. A run of one of the algorithms goes on until it manages to successfully connect 250 nodes to its tree. The recorded values are time taken, samples taken and number of goals reached. For both algorithms, a step size of 0.7 m and a rewiring distance of 2.5 m was used. For the modified RRT*, the closest 5 nodes will be checked for validity and a disk with a radius of 0.3 m will be cut out of the sample space around each node. The average and standard deviation of the recorded values are presented in Table 4.1.

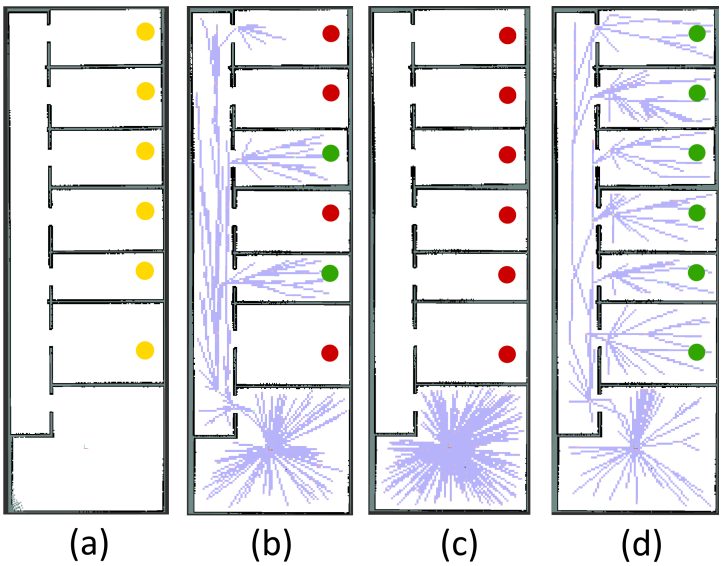

Figure 4.4: Trees grown by the path finding algorithm for the baseline RRT* compared to RRT* with the proposed modifications. Image (a) shows the Occupancy Grid with the goal locations as yellow disks. Image (b) displays the typical performance of the baseline RRT*, reaching some, but not all goals. Image (c) shows the worst case scenario for the baseline encountered during the experiment. The tree does not manage to grow outside of its starting area and puts down all 250 nodes in close proximity to the base. Image (d) shows the typical performance of the modified RRT* algorithm.

Algorithm	Time [s]	Points Sampled [-]	Goals Reached [-]
Baseline RRT*	0.50 $\pm$ 0.22	507.80 $\pm$ 198.79	2.20 $\pm$ 1.17
Modified RRT*	0.24 $\pm$ 0.08	199.80 $\pm$ 43.67	6.00 $\pm$ 0.00

Table 4.1: Results of ten iterations of each algorithm, where an iteration is completed after connecting 250 nodes to their trees. The values presented are the average values $\pm$ their standard deviations.

From Figure 4.4 and Table 4.1, it can be observed that the modified RRT* algorithm indeed outperforms the baseline, reaching all of the goals consistently while requiring fewer sampled nodes and less time on average. Moreover, while the baseline reaches a few of the goals most of the time, on two of the ten iterations it failed to grow its tree beyond the starting room, placing all of its 250 nodes in one room close to the base as is shown in image (c) of Figure 4.4. This happens because there is only a small area at the bottom of the hallway where a successful connection can be made to grow the tree out of its starting room. Moreover, if a node is placed close to the wall of the doorway, even if a node is sampled that could be connected successfully to another in the room, the closest node will be on the other side of the wall leading to a rejected sample. This effect is presented in image (b) of Figure 3.8.

The effectiveness of the modifications is also evident when looking qualitatively at the difference between image (b) and (d) in Figure 4.4. For the modified algorithm in image (d), the branches in the hallway are much sparser than in image (b), showing the effect of the greedy node connection strategy where only one sampled point at the end of the hallway can cover large parts of the hallway in nodes that can then be used to branch into the adjacent rooms. The sample space reduction technique, that removes areas close to the tree from the sample space, then ensures that, when such a branch has been grown through the hallway, the next points to be sampled will more likely be in the connecting

rooms. As mentioned in Section 3.3.2, the radius of the circles drawn around the existing RRT tree must not be too large since that would result in too few nodes sampled around the tree, leading to poorly optimized paths. Finally, testing several of the closest nodes for validity instead of only the closest one increases the chances that a node in the hallway will be considered for connection. The proposed modifications allow the path finder to rapidly and consistently explore both sparse and dense environments, relying less on the exact parameter tuning. This is desirable since the system will be deployed in a wide range of indoor environments without prior knowledge. In conclusion, the three modifications to the baseline RRT* work in tandem to improve the performance of the path finder, leading to a higher reliability and more consistent performance.

4.2.2. Graph Creation

To show the action graph creation procedure and investigate its limits, a more complex office environment will be used than in the last experiments. The environment will be in a partially explored state where all the elements of the action graph creator, the frontier detection and path finding, segmentation and merging will be required to create the final action graph. The results will be compared to live sampling approaches that sample the action graph from the map during the MCTS process. The comparison will be made both in terms of node density and the number of complex, branching nodes. The paths found by RRT*, the merged paths and an example of a live sampling approach are shown in Figure 4.5.

Live sampling techniques, as shown in image (c) in Figure 4.5, will have multiple parallel paths running down a single hallway. These have been merged into a single path in the proposed approach reducing the number of nodes linearly. Moreover, as can be seen from image (b) of Figure 4.5, the number of complex nodes has been reduced to only the positions with significant direction changes, instead of having multiple branching points all pointing towards the same direction as is the case in (c). In the example from image (c) in Figure 4.5, the planner has searched fourteen nodes to plan four actions down a hallway, the first nine of which, arguably, contain the same information. Moreover, the example contains five complex nodes, the first three of which result in actions in the same direction. The latter two could be merged due their proximity, further reducing the number of nodes and thus the branching problem. While having a more dense action space might seem to be a critical issue in distributed approaches, when the robots are coordinates to improve exploration efficiency, real-time performance suffers significantly [11]. This shows that when coordination is required, the number of nodes in the action space should be minimal to reduce the possible number of system configurations.

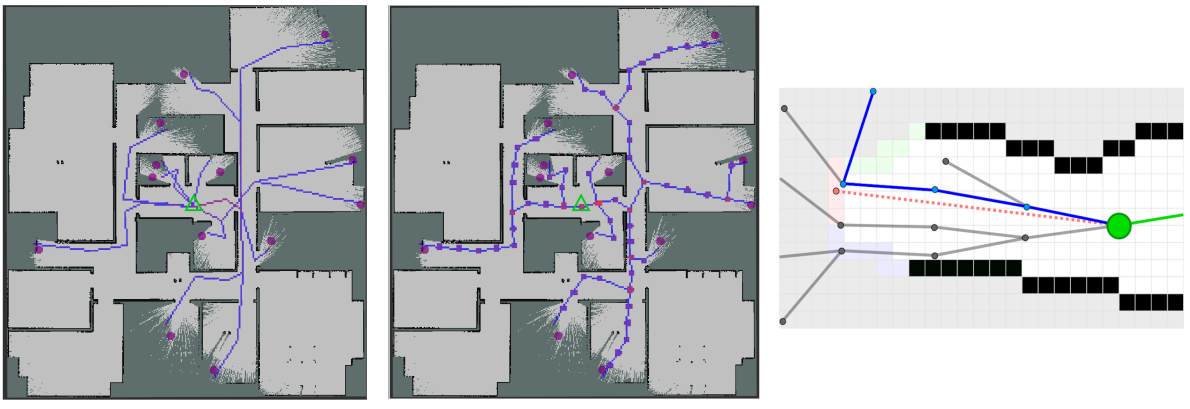

Figure 4.5: Action graph creation where the map with the detected frontiers and the paths found by RRT* are shown in (a). The current position of the robot is in the middle of the map shown by the green triangle. The result of the subsequent path segmentation and merging process is shown in image (b). An example of a live sampling approach is shown in (c).

While the previous example shows the action graph as created from scratch when the environment is already partially explored, in reality, the action graphs will be created iteratively during the exploration as discussed in Section 3.3. This will inevitably lead to inefficient paths due to the goals shifting during the exploration without an optimization or correction routine present. To investigate these shortcomings,

the full system including MCTS task assignment as presented in Section 3 is implemented. The only difference is the use of the map merger and SLAM Toolbox instead of the full C-SLAM framework to reduce implementation complexity. Single-robot exploration will be performed without connectivity constraints, allowing the robot to iteratively create the full action graph that it uses during its exploration. This can then be compared to the action graph created from scratch in the same, but fully explored environment.

The exploration will be repeated three times, while recording the time taken for each planning step during the exploration cycle. The action graph consistency in terms of action length will be recorded as well. The path finding parameters are the same as in Section 4.2.1 and the segmentation step size, s, is set to 1 m. Several stages of the exploration process are presented in Figure 4.6 while the numerical results are presented in Table 4.2.

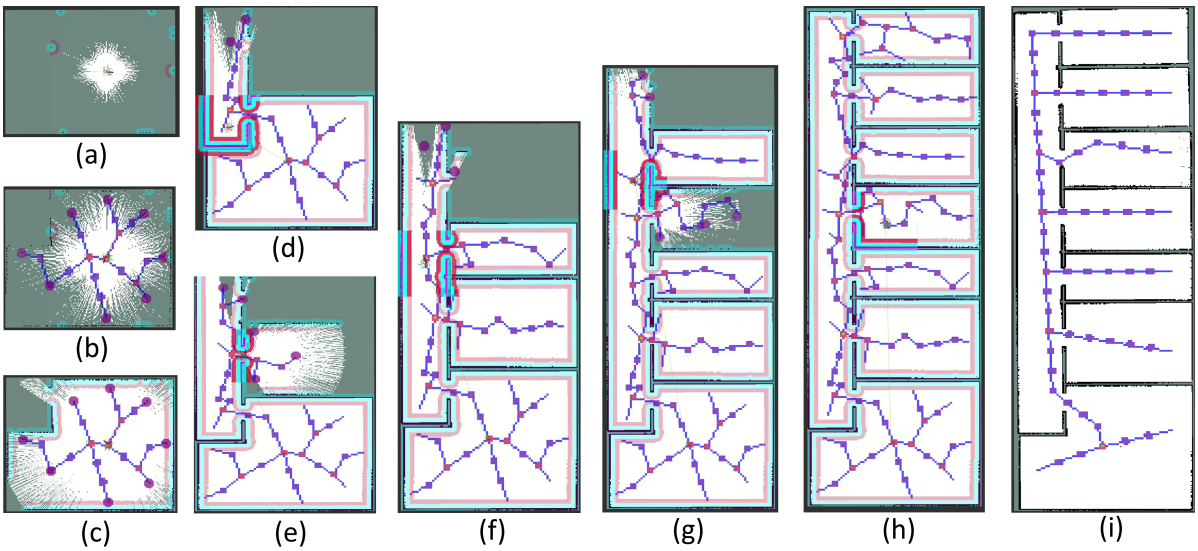

Figure 4.6: Action graph creation during MCTS based exploration. Frontiers are shown as purple disks. The nodes of the graph represent the action space and the brighter anode is, the more children are connected to it. Images (a) to (h) show the exploration and graph creation process, while (i) shows a typical action graph when created from scratch in the fully explored environment.

Process	MCTS Cycle Time [s]	PF Time [s]	AC Creation Time [s]	Action Length [m]	Nodes in AC [-]	Number of Cycles [-]
Iterative	0.019 $\pm$ 0.0010	0.07 $\pm$ 0.021	0.031 $\pm$ 0.0038	0.92 $\pm$ 0.23	112.00 $\pm$ 4.3	138.67 $\pm$ 12
Hindsight	-	0.28 $\pm$ 0.042	0.03 $\pm$ 0.011	1.014 $\pm$ 0.073	62.43 $\pm$ 2.6	1.00 $\pm$ -

Table 4.2: Average results of three runs of the iterative graph creation during exploration compared to the action graph that can be created after the map has been explored. The values shown are averages $\pm$ their standard deviations. PF = Path Finding, AC = Action Graph.

From images (a) to (h) in Figure 4.6, it can be observed that the robot can successfully create the action graph iteratively during the exploration process. However, it is not particularly smooth. There seem to be two major reasons for this. The first is that, since the previous action graph at any point in time remains unaltered, any frontiers that are not perfectly on the optimal path will be the point at which the newly found paths will be connected. This can lead to detours in the action graph as are visible in the middle room in image (h). A much larger impact on the roughness of the action graph can be attributed to the way in which the 2D laser scanner of the Turtlebot3 functions. It will only designate a cell as free space if it lies between the robot and an observed obstacle. This becomes problematic in long hallways as it leads to a V-shaped pattern of unknown cells, since the robot can only observe the side walls. This directly results in a zig-zag pattern through the hallway as can be observed at the unexplored end of the hallways of images (d), (e) and (f). This pattern is absent from image (i) where

the action graph is created in one step after the map has been explored. Both of these factors can be mitigated by the use of an optimization step to smoothen the action graph.

When analyzing the numerical results in Table 4.2, the action graph creation process is fast enough for real-time performance. When comparing the action length of both processes, the averages are reasonably close to the target of $s = 1m$. However, the standard deviation of s for the iteratively created graph is an order of magnitude larger due to the lack of optimization of the action graph in between the planning cycles. This means that the action length is more inconsistent in the unoptimized graph, which can lead to inefficient coordination if robots with shorter action lengths have to wait for others to complete their actions. To mitigate this to some degree, task assignment is allowed to start when all robots are within a set distance of their target. However, in future work, a more realistic measure of the estimated time to complete an action is required, since changes in direction also take time which is currently not incorporated into the estimate. During the optimization step, the distance between two actions must be corrected using this measure of the required time.

When looking at the time required for path finding and the creation of the action graph, it can be concluded that there is room to add complexity in terms of extra steps to improve the action graph without impacting real-time performance of the system significantly. Additionally, the graph could also be checked for the possibility to merge the separate branches. This is necessary in environments such as in Figure 4.5 where there are several interconnecting hallways. Branches that grow through each hallway separately currently have no way of being connected. This is problematic because a robot will have to drive back to the common parent node of both branches before it is allowed to move to the other branch. A possible solution is to connect every node pair that could have an unobstructed edge if the ratio of their Euclidean distance to distance on the graph is below some threshold, indicating that they are close to each other on the map but far apart on the action graph.

In conclusion, the distributed action graph creation process succeeds in lowering both the number of total nodes and the number of complex nodes with respect to live sampling methods. Additionally, it is fast enough to run in real-time. However, in future work, an additional optimization step is required to smoothen the graph to reduce unnecessary movements and to take the influence direction changes have on action length into account for node spacing. The optimization step must also join separate branches when required.

4.3. Task Assignment

In this section, the task assignment scheme will be tested. The main aim is to show the behaviors that have been induced only through the assignment of rewards while respecting the requirements. As presented in Section 3.4, the two requirements will be that the robots maintain the network connection between themselves and the base station and that they prevent collision between themselves. The expected behaviors are firstly that robots will form relay links where one robot gives up its exploration tasks to become a network relay for the others. Secondly, the collision constraint is expected to result in coordination of the robots when moving through the hallways and doorways. For these experiments, the full system has been implemented except for the C-SLAM framework. As explained at the start of this chapter, in its place, the SLAM toolbox and a map merger are used.

4.3.1. Behavior Analysis

The first behavior to verify is the relay formation. To analyze whether the desired behavior is actually induced, the same office test environment is used as before. Since the network manager updates the walls during the exploration, the heatmap from image (c) in Figure 4.3 can be used as a reference for where the first robot should stop exploring and become a relay. The results from these tests are shown in Figure 4.7. Image (a) shows the heatmap which represents the network coverage of the full system. Image (b) and (c) show the action graphs and the positions of the two robots. From the figure it can be observed that the relay behavior is shown as expected. One of the robots keeps back as marked in image (b), allowing the other robot to continue exploring.

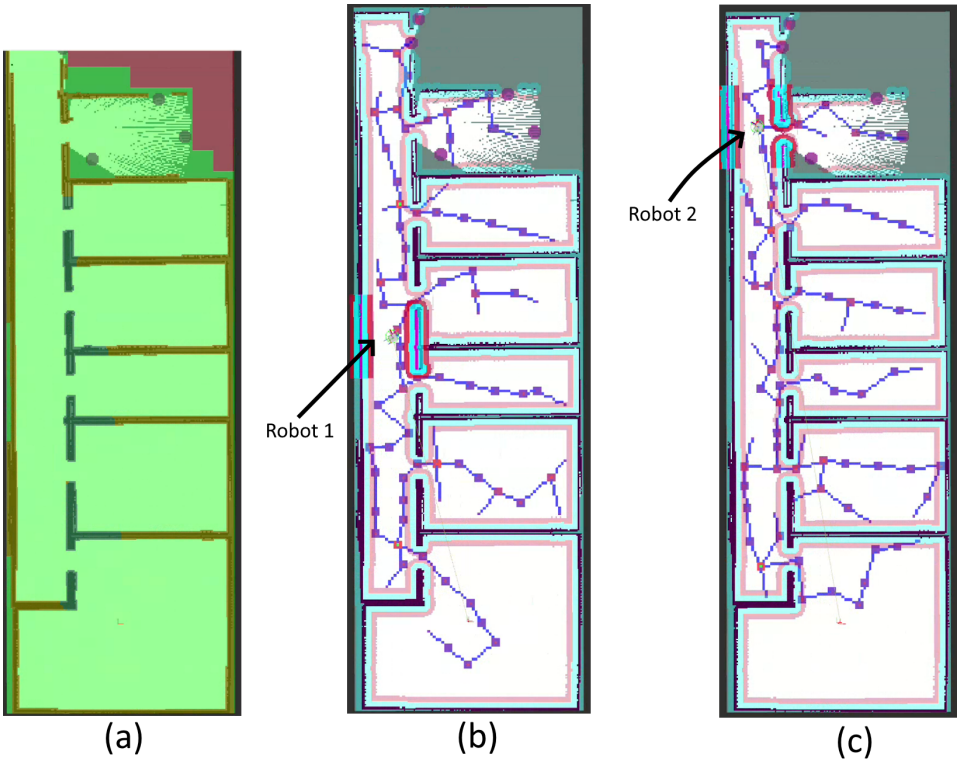

Figure 4.7: Relay behavior of Robot 1 extending the communication range such that Robot 2 can explore.

The second behavior to investigate is the coordination between the robots in narrow spaces to avoid collisions. Two sequences of images are presented in Figure 4.8, showing two robots, where the robot in sequence (a) moves to a state further away from the frontiers to let the robot in (b) pass. During exploration, the robots exhibit this behavior when necessary to avoid collisions in the narrow environment and during the performed tests, the robots have never collided.

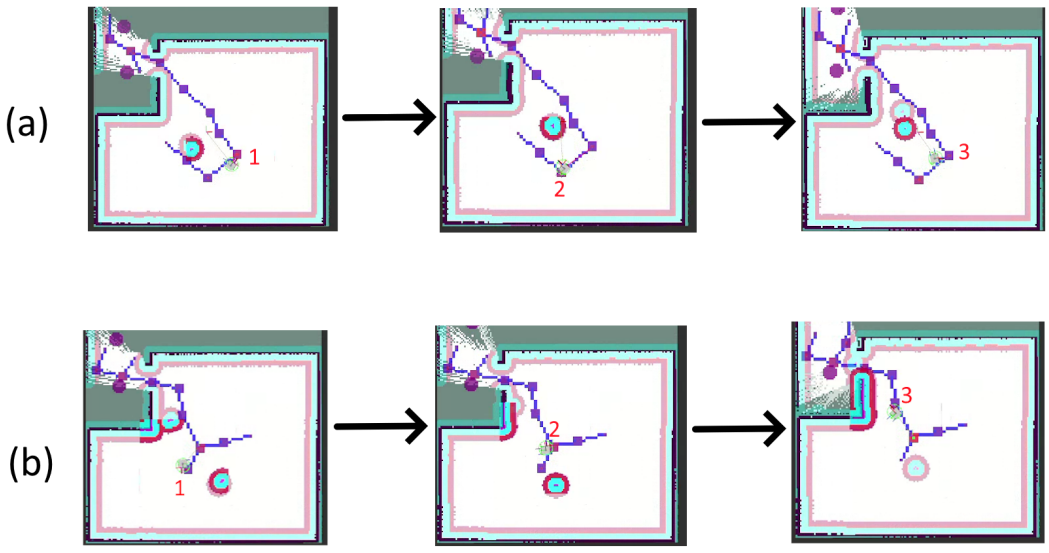

Figure 4.8: Sequence of movements for two robots showing the collision constraint by the robot in sequence (a) moving backwards such that the one in (b) can pass.

4.3.2. Limitations

While the proposed system in general works quite well and several successful explorations with one to three robots have been performed, there are several limitations that inhibit more elaborate testing and comparison in different scenarios. These have to do with the action graph generation subsystem. Two issues compound to make the full exploration unreliable during longer use with multiple robots. First, the frontiers are often placed in unknown space, requiring the path finding stage to allow connections slightly past the frontier. In order for the path finder to connect these frontiers, it must be allowed to plan through unknown space. The problem is that robots may detect walls as a series of separate obstacles instead of a solid object. The cause of this is the Turtlebot's 2D laserscanner as discussed earlier in Section 4.2.2 with respect to the V-shaped indentations. This means that allowing RRT* to grow its tree through unknown obstacles can lead to the action graph intersecting walls. An attempt has been made to remedy this by using the wall map as proposed in Section 3.2.1 as an additional constraint for RRT*. However, while it reduces the number of branches grown through walls, it does not eliminate the problem entirely, especially at the far ends of the robot's sensing range. Thus, to prevent tree growth through walls, the path finding stage contains an early stopping condition to halt the RRT* process when the number of points in the sample space reaches a lower threshold. As a result, goals can be left out of the action graph erroneously, leading to a lack of rewards and an area being left unexplored.

The second issue is that, while the modified RRT* process is more reliable for dense indoor environments than the standard RRT*, the map of the environment is expanded in a highly erratic way during exploration. This leads to long, narrow patches of free space being created that are even more challenging for the RRT* process to grow its tree into than if the environment was already explored beforehand. This is mainly due the fact that these patches are at a sharp angle with respect to the hallway, requiring a node in the hallway to be in very close proximity for a successful connection. While the modified RRT* procedure is successful most of the time, it only takes the path finder getting stuck once to halt exploration. From Table 4.2, it can be seen that to fully explore the test environment from Figure 4.6, around 138 cycles are performed by each robot. This means that even a very low failure rate will eventually lead to the process getting stuck. The sample space threshold mentioned above does not generalize enough to both of these problems and requires such specific tuning that frontiers are still missed or the path finder gets stuck.

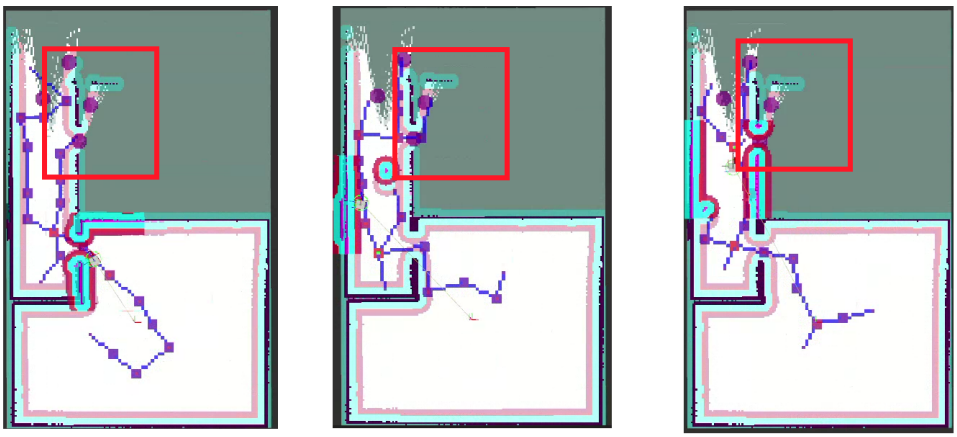

Figure 4.9: Images showing the erratic map growth for three robots simultaneously exploring the environment. The problem area is shown in the red rectangle for all three robots. Notably, only the robot in image (b) successfully connects the rightmost frontier to its graph shown by the blue line.

Both of these issues can be fixed by incorporating an additional optimization step in the action graph creation procedure. The path finder can then be allowed to grow its tree to all frontiers even if sometimes it intersects a wall. The optimizer must then, in addition to smoothening the graph and connecting separate branches, also reject edges that intersect with obstacles so they can be reconnected using the path finder with newly updated map information.

5

Conclusion

In this thesis, an autonomous multi-robot system for indoor exploration in limited network environments is proposed which updates the network model based on signal measurements. Initially, the problem, its subproblems and the subsequent requirements have been formulated. These requirements were translated into four research questions that were used to guide the research process.

Using the answers to the research questions provided in Section 2.5, a hybrid system is proposed where the global mapping, task assignment and network modeling is performed by the server. In turn, the robots create their local maps, perform signal measurements and create their action graphs.

Several experiments have been performed, testing each subsystem as well as the complete exploration scheme. While the individual subsystems showed promise, additional work is required to realize their full potential. Firstly, the network model was tested. While updating the wall segments significantly increases the communication range of the robots, allowing for faster exploration through network coverage with fewer relay robots, the path loss model has turned out to be too simple to accurately estimate signal strengths. Several propagation phenomena, such as reflection, are not modeled in path loss models. However, they have been shown to significantly impact the real-world measurements. To remedy this, future work could focus on identifying the most relevant of these phenomena and subsequently devising a light-weight way to incorporate them into path loss models.

Secondly, the action graph formulation was investigated. The action graph is shown to significantly reduce the number of nodes in the action space with respect to methods where the graph is sampled during exploration. However, its iterative nature and lack of further optimization results in an inconsistent graph, with longer paths to the goals than necessary in combination with unequal action lengths. Future work could implement an additional step that optimizes the graph in terms of action length with respect to actual action duration and distance to goals by smoothening and joining separate branches. This optimization would also solve the largest problem encountered in the final experiments of the complete system during exploration, where the persistent nature of the action graph leads to strict requirements for path finding and merging which, in some cases, resulted in system failure.

In conclusion, while the presented system shows significant promise with respect to the goal of providing first responders with insight deeper into the interiors of buildings, additional work is required before real-world operation is feasible.

References

[1] Razanne Abu-Aisheh et al. “CARA: Connectivity-Aware Relay Algorithm for Multi-Robot Expeditions”. In: *Sensors* 22.23 (2022). ISSN: 1424-8220. DOI: 10.3390/s22239042. URL: https://www.mdpi.com/1424-8220/22/23/9042.

[2] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. *Ceres Solver*. Version 2.1. Mar. 2022. URL: https://github.com/ceres-solver/ceres-solver.

[3] Francesco Amigoni, Jacopo Banfi, and Nicola Basilico. “Multirobot Exploration of Communication-Restricted Environments: A Survey”. In: *IEEE Intelligent Systems* 32.6 (2017), pp. 48–57. DOI: 10.1109/MIS.2017.4531226.

[4] Relja Arandjelovi, Petr Gronat INRIA, and Josef Sivic INRIA. “NetVLAD: CNN architecture for weakly supervised place recognition Tomas Pajdla CTU in Prague ‡”. In: *IEEE* (2016).

[5] Paramvir Bahl and Venkata N Padmanabhan. *RADAR: An In-Building RF-based User Location and Tracking System*. 2000.

[6] P. Baran. “On Distributed Communications Networks”. In: *IEEE Transactions on Communications Systems* 12.1 (1964), pp. 1–9. DOI: 10.1109/TCOM.1964.1088883.

[7] Facundo Benavides et al. “An auto-adaptive multi-objective strategy for multi-robot exploration of constrained-communication environments”. In: *Applied Sciences (Switzerland)* 9 (3 Feb. 2019). ISSN: 20763417. DOI: 10.3390/app9030573.

[8] Facundo Benavides et al. “Multi-robot Cooperative Systems for Exploration: Advances in Dealing with Constrained Communication Environments”. In: Institute of Electrical and Electronics Engineers Inc., Dec. 2016, pp. 181–186. ISBN: 9781509036561. DOI: 10.1109/LARS-SBR.2016.37.

[9] Gabriele Berton, Carlo Masone, and Barbara Caputo. “Rethinking Visual Geo-localization for Large-Scale Applications”. In: *CvF* (2022). URL: https://github.com/gmberton/CosPlace..

[10] Graeme Best et al. “Dec-MCTS: Decentralized planning for multi-robot active perception”. In: *The International Journal of Robotics Research* 38.2-3 (2019), pp. 316–337.

[11] Sean ; Bone et al. “Decentralised Multi-Robot Exploration using Monte Carlo Tree Search Conference Paper Decentralised Multi-Robot Exploration using Monte Carlo Tree Search”. In: (2023). DOI: 10.3929/ethz-b-000624905. URL: https://doi.org/10.3929/ethz-b-000624905.

[12] *Boston Dynamics Apllications page*. URL: https://bostondynamics.com/industry/.

[13] Cameron B Browne et al. “A survey of monte carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[14] W. Burgard et al. “Collaborative multi-robot exploration”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 476–481 vol.1. DOI: 10.1109/ROBOT.2000.844100.

[15] Wolfram Burgard et al. “Coordinated multi-robot exploration”. In: *IEEE Transactions on Robotics* 21 (3 June 2005), pp. 376–386. ISSN: 15523098. DOI: 10.1109/TRO.2004.839232.

[16] Hamza Chakraa et al. “Optimization techniques for Multi-Robot Task Allocation problems: Review on the state-of-the-art”. In: *Robotics and Autonomous Systems* 168 (2023), p. 104492. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2023.104492. URL: https://www.sciencedirect.com/science/article/pii/S0921889023001318.

[17] Yun Chang et al. “Kimera-Multi: A System for Distributed Multi-Robot Metric-Semantic Simultaneous Localization and Mapping”. In: vol. 2021-May. Institute of Electrical and Electronics Engineers Inc., 2021, pp. 11210–11218. ISBN: 9781728190778. DOI: 10.1109/ICRA48506.2021.9561090.

[18] Yun Chang et al. “LAMP 2.0: A Robust Multi-Robot SLAM System for Operation in Challenging Large-Scale Underground Environments”. In: *IEEE Robotics and Automation Letters* 7 (4 Oct. 2022), pp. 9175–9182. ISSN: 23773766. DOI: 10.1109/LRA.2022.3191204.

[19] Siddharth Choudhary et al. “Distributed trajectory estimation with privacy and communication constraints: A two-stage distributed Gauss-Seidel approach”. In: *IEEE* (2016).

[20] M Scott Corson and Anthony Ephremides. “A distributed routing algorithm for mobile wireless networks”. In: *Wireless networks* 1.1 (1995), pp. 61–81.

[21] Andrei Cramariuc et al. “Maplab 2.0 - A Modular and Multi-Modal Mapping Framework”. In: *IEEE Robotics and Automation Letters* 8 (2 Feb. 2023), pp. 520–527. ISSN: 23773766. DOI: 10.1109/LRA.2022.3227865.

[22] *Darpa SubT challenge website*. URL: https://www.darpa.mil/program/darpa-subterranean-challenge.

[23] Konstantinos G Derpanis. “Overview of the RANSAC Algorithm”. In: *Image Rochester NY* 4.1 (2010), pp. 2–3.

[24] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. “Superpoint: Self-supervised interest point detection and description”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 224–236.

[25] Gregory Dudek et al. “A taxonomy for multi-agent robotics”. In: *Autonomous Robots* 3 (4 1996), pp. 375–397. ISSN: 09295593. DOI: 10.1007/BF00240651.

[26] Kamak Ebadi et al. “LAMP: Large-Scale Autonomous Mapping and Positioning for Exploration of Perceptually-Degraded Subterranean Environments”. In: (Mar. 2020). URL: http://arxiv.org/abs/2003.01744.

[27] A. Elfes. “Using occupancy grids for mobile robot perception and navigation”. In: *Computer* 22.6 (1989), pp. 46–57. DOI: 10.1109/2.30720.

[28] H. T Friis. *Friis: Transmission Formula*. 1946.

[29] Dorian Galvez-Lopez and Juan D Tardos. “Bags of Binary Words for Fast Place Recognition in Image Sequences”. In: *IEEE Transactions on Robotics* 28 (5 2012), pp. 1189–1198. ISSN: 15523098. DOI: 10.1109/TRO.2011.2179580.

[30] Brian P Gerkey and Maja J Matarić. “A formal analysis and taxonomy of task allocation in multi-robot systems”. In: *The International journal of robotics research* 23.9 (2004), pp. 939–954.

[31] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[32] Michael L Hawkins. “A generalization of Snell’s law”. PhD thesis. Monterey, California. Naval Postgraduate School, 1990.

[33] R.W. Heath. *Introduction to Wireless Digital Communication: A Signal Processing Perspective*. Pearson Education, 2017. ISBN: 9780134431840.

[34] *Howe and Howe fire fighting robots website*. URL: https://www.howeandhowe.com/civil/thermite.

[35] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[36] Bilal Kartal et al. “Monte Carlo tree search with branch and bound for multi-robot task allocation”. In: *The IJCAI-16 workshop on autonomous mobile service robots*. Vol. 33. 2016.

[37] Matan Keidar and Gal A Kaminka. “Robot exploration with fast frontier detection: Theory and experiments”. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. 2012, pp. 113–120.

[38] Matan Keidar and Gal A. Kaminka. “Efficient frontier detection for robot exploration”. In: *The International Journal of Robotics Research* 33.2 (2014), pp. 215–236. DOI: 10.1177/0278364913494911. URL: https://doi.org/10.1177/0278364913494911.

[39] Giseop Kim and Ayoung Kim. “Scan Context Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map”. In: *2018 IEEE RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018).

[40] *KITTI single-robot odometry benchamrking*. URL: https://www.cvlibs.net/datasets/kitti/eval_odometry.php.

[41] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. “A comprehensive taxonomy for multi-robot task allocation”. In: *The International Journal of Robotics Research* 32.12 (2013), pp. 1495–1512.

[42] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[43] P. Lajoie et al. “DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1656–1663.

[44] Pierre-Yves Lajoie and Giovanni Beltrame. “Swarm-SLAM : Sparse Decentralized Collaborative Simultaneous Localization and Mapping Framework for Multi-Robot Systems”. In: (Jan. 2023). URL: http://arxiv.org/abs/2301.06230.

[45] Pierre-Yves Lajoie et al. “Towards Collaborative Simultaneous Localization and Mapping: a Survey of the Current Research Landscape”. In: (Aug. 2021). DOI: 10.55417/fr.2022032. URL: http://arxiv.org/abs/2108.08325%20http://dx.doi.org/10.55417/fr.2022032.

[46] Ben Liu et al. “A variable-step RRT* path planning algorithm for quadrotors in below-canopy”. In: *IEEE Access* 8 (2020), pp. 62980–62989.

[47] Steve Macenski et al. “The Marathon 2: A Navigation System”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020. URL: https://github.com/ros-planning/navigation2.

[48] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[49] Joshua G. Mangelson et al. “Pairwise Consistent Measurement Set Maximization for Robust Multi-robot Map Merging”. In: *IEEE* (May 2018).

[50] Laetitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. “Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 26.1 (Sept. 2021), pp. 2017–2023. DOI: 10.1609/aaai.v26i1.8380. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8380.

[51] Hakim Mitiche, Dalila Boughaci, and Maria Gini. In: *Journal of Intelligent Systems* 28.2 (2019), pp. 347–360. DOI: doi:10.1515/jisys-2018-0267. URL: https://doi.org/10.1515/jisys-2018-0267.

[52] Alejandro R. Mosteo, Luis Montano, and Michail G. Lagoudakis. “Multi-robot routing under limited communication range”. In: *2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 1531–1536. DOI: 10.1109/ROBOT.2008.4543419.

[53] *Nav2 stack*. URL: https://navigation.ros.org/.

[54] Thomas Nestmeyer et al. “Decentralized simultaneous multi-target exploration using a connected network of multiple robots”. In: *Autonomous Robots* 41 (4 Apr. 2017), pp. 989–1011. ISSN: 15737527. DOI: 10.1007/s10514-016-9578-9.

[55] *Occupancy Grid Message*. URL: http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/OccupancyGrid.html.

[56] *Open-Mesh- B.A.T.M.A.N website*. URL: https://www.open-mesh.org/projects/open-mesh/wiki.

[57] Manthan Patel et al. “COVINS-G: A Generic Back-end for Collaborative Visual-Inertial SLAM”. In: (2023). DOI: 10.48550/ARXIV.2301.07147. URL: https://arxiv.org/abs/2301.07147.

[58] Yuanteng Pei and Matt W Mutka. *Steiner Traveler: Relay Deployment for Remote Sensing in Heterogeneous Multi-Robot Exploration*. 2012.

[59] David Plets et al. “Coverage prediction and optimization algorithms for indoor environments”. In: *Eurasip Journal on Wireless Communications and Networking* 2012 (2012). ISSN: 16871472. DOI: 10.1186/1687-1499-2012-123.

[60] Félix Quinton, Christophe Grand, and Charles Lesire. “Market Approaches to the Multi-Robot Task Allocation Problem: a Survey”. In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 107 (2 Feb. 2023). ISSN: 15730409. DOI: 10.1007/s10846-022-01803-0.

[61] Danny G. Riley and Eric W. Frew. “Assessment of Coordinated Heterogeneous Exploration of Complex Environments”. In: Institute of Electrical and Electronics Engineers Inc., 2021, pp. 138–143. ISBN: 9781665436434. DOI: 10.1109/CCTA48906.2021.9658770.

[62] *Robotis Turtlebot product page*. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/.

[63] Martijn N. Rooker and Andreas Birk. “Multi-robot exploration under the constraints of wireless networking”. In: *Control Engineering Practice* 15 (4 Mar. 2007), pp. 435–445. ISSN: 09670661. DOI: 10.1016/j.conengprac.2006.08.007.

[64] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. “A review of mobile robots: Concepts, methods, theoretical framework, and applications”. In: *International Journal of Advanced Robotic Systems* 16.2 (2019), p. 1729881419839596.

[65] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *Proceedings of the IEEE International Conference on Computer Vision* (2011), pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[66] Sajad Saeedi et al. “Multiple-Robot Simultaneous Localization and Mapping: A Review”. In: *Journal of Field Robotics* 33 (1 Jan. 2016), pp. 3–46. ISSN: 15564967. DOI: 10.1002/rob.21620.

[67] Patrik Schmuck and Margarita Chli. “CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams”. In: *Journal of Field Robotics* 36 (4 June 2019), pp. 763–781. ISSN: 15564967. DOI: 10.1002/rob.21854.

[68] Guillaume Tesserault, Nadine Malhouroux, and Patrice Pajusco. “Determination of material characteristics for optimizing WLAN radio”. In: *2007 European Conference on Wireless Technologies*. IEEE. 2007, pp. 225–228.

[69] *Throwbot General Page*. 2024. URL: https://reconrobotics.com/.

[70] Yulun Tian et al. “Kimera-Multi: Robust, Distributed, Dense Metric-Semantic SLAM for Multi-Robot Systems”. In: *IEEE Transactions on Robotics* 38 (4 Aug. 2022), pp. 2022–2038. ISSN: 19410468. DOI: 10.1109/TRO.2021.3137751.

[71] Yulun Tian et al. “Resilient and Distributed Multi-Robot Visual SLAM: Datasets, Experiments, and Lessons Learned”. In: (Apr. 2023). URL: http://arxiv.org/abs/2304.04362.

[72] Marco Tranzatto et al. “Cerberus: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge”. In: *arXiv preprint arXiv:2201.07067* (2022).

[73] Bill Triggs et al. *Bundle Adjustment-A Modern Synthesis*. URL: http://www.inrialpes.fr/movi/people/Triggs.

[74] Hassan Umari and Shayok Mukhopadhyay. “Autonomous robotic exploration based on multiple rapidly-exploring randomized trees”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 1396–1402.

[75] Jose Vazquez and Chris Malcolm. “Distributed Multirobot Exploration Maintaining a Mobile Network”. In: *SECOND IEEE INTERNATTONAL CONFERENCE ON INTELLIGENT SYSTEMS* (2004).

[76] Changyun Wei, Koen V. Hindriks, and Catholijn M. Jonker. “Dynamic task allocation for multi-robot search and retrieval tasks”. In: *Applied Intelligence* 45 (2 Sept. 2016), pp. 383–401. ISSN: 15737497. DOI: 10.1007/s10489-016-0771-5.

[77] Greg Welch, Gary Bishop, et al. “An introduction to the Kalman filter”. In: (1995).

[78] Bradley Woosley et al. “Multi-robot information driven path planning under communication constraints”. In: *Autonomous Robots* 44 (5 May 2020), pp. 721–737. ISSN: 15737527. DOI: 10.1007/s10514-019-09890-z.

[79] Hao Xu et al. “D^2SLAM: Decentralized and Distributed Collaborative Visual-inertial SLAM System for Aerial Swarm”. In: *arXiv preprint arXiv:2211.01538* (2022).

[80] 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. DOI: 10.1109/CIRA.1997.613851.

[81] Brian Yamauchi. “Frontier-based exploration using multiple robots”. In: *Proceedings of the Second International Conference on Autonomous Agents*. AGENTS ’98. Minneapolis, Minnesota, USA: Association for Computing Machinery, 1998, pp. 47–53. ISBN: 0897919831. DOI: 10.1145/280765.280773. URL: https://doi.org/10.1145/280765.280773.

[82] Heng Yang et al. “Graduated Non-Convexity for Robust Spatial Perception: From Non-Minimal Solvers to Global Outlier Rejection”. In: *IEEE Robotics and Automation Letters* 5 (2 Apr. 2020), pp. 1127–1134. ISSN: 23773766. DOI: 10.1109/LRA.2020.2965893.

[83] Darío Fernando Yépez-Ponce et al. “Mobile robotics in smart farming: current trends and applications”. In: *Frontiers in Artificial Intelligence* 6 (2023). ISSN: 26248212. DOI: 10.3389/frai.2023.1213330.

[84] Jincheng Yu et al. “Smmr-explore: Submap-based multi-robot exploration system with multi-robot multi-target potential field exploration method”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8779–8785.

[85] Luowei Zhou et al. “A balanced heuristic mechanism for multirobot task allocation of intelligent warehouses”. In: *Mathematical Problems in Engineering* 2014 (2014).

A

C-SLAM Framework Comparison Table

Framework	COVINS-G [57]	LAMP 2.0 [18]	CCM-SLAM [67]	maplab 2.0 [21], [72]	DOOR-SLAM [43]	D^2-SLAM [79]	Kimera-Multi [17], [70], [71]	Swarm-SLAM [44]
Sensor Modalities (Type)	Visual-Inertial (stereo, RGBD, T265 tracking camera)	LiDAR	Visual-Inertial (monocular camera)	Multi-Modal	Visual-Inertial (stereo camera)	Visual-Inertial (stereo and omnidirectional camera)	Visual-Inertial (stereo/depth camera)	Multi-Modal*
Number and Type of Robots Tested	12 Ground	4 Ground	3 Aerial	5 Ground and 3 Aerial simultaneously	2 Aerial	2 Aerial	8 Ground	3 Ground
Environment (datasets and live experiments)	Indoor, outdoor	Underground	Indoor, outdoor	Indoor, outdoor, underground	Outdoor	Indoor	Indoor, outdoor	Indoor, outdoor
Descriptor	BoW	-	BoW	Any	NetVLAD	NetVLAD	BoW	CosPlace, NetVLAD ScanContext
Feature Type	ORB, SIFT	-	ORB	Any	ORB	SuperPoint	ORB	Any
Outlier Rejection	RANSAC	ICM	RANSAC	RANSAC	RANSAC + PCM	RANSAC	RANSAC	RANSAC
Loop Closure Prioritization	Greedy	Loop closure accuracy, trajectory error reduction, RSSI beacon	Greedy	Greedy	Greedy	Greedy	Greedy	Spectral
Optimizer	LMA (PGO)	LMA (PGO)	LMA (PGO + BA)	Any	DGS	LMA	D-GNC	GNC
Ad-hoc Networking	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Modular	Odometry, sensor configuration, features	Odometry, sensor configuration	Odometry	Odometry, sensors descriptors, features optimizers map features	Odometry	-	-	Odometry, sensors, descriptors features
Network Architecture	Centralized	Centralized	Centralized	Centralized	Distributed	Distributed	Distributed	Decentralized
ROS Version	Melodic, Noetic	Noetic	Noetic	Noetic	-	Noetic	Melodic, Noetic	ROS2 Foxy
Code Repositories	GitHub	GitHub	GitHub	GitHub, Wiki	GitHub	GitHub	GitHub	GitHub

Table A.1: Summary of frameworks analyzed in Chapter 2.2. Note: The last row contains hyperlinks to the code repositories.

*While multiple modalities are supported in SWARM-SLAM, there is no built-in method for multi-modal loop closures. The LiDAR can only be used in addition to a stereo camera [44].

List of algorithms:

- **BoW:** A descriptor for visual place recognition [29].
- **NetVLAD:** A descriptor for visual place recognition based on convolutional neural networks [4]. It improves on BoW on recognition performance at the cost of training and computation requirements.
- **CosPlace:** Descriptor for visual place recognition that improves NetVLAD by requiring less data to train [9].
- **ScanContext:** Descriptor for place recognition in point cloud data [39].
- **ORB:** Rotation invariant and noise resistant feature for visual data matching [65].
- **SIFT:** Feature type for visual data matching [65]. Mostly superseeded by ORB.
- **SuperPoint:** Framework for feature and descriptor extraction [24].
- **RANSAC:** Technique for finding the static transform between to images [23].

- **PCM:** Pairwise Consistent set Maximization [49]. Outlier rejection technique based on pairwise internal consistency of two candidates.
- **ICM:** Incremental Consistency Maximization [26]. Outlier rejection technique that is a variant of PCM for point clouds.
- **Greedy:** Greedy loop closure prioritization that accepts the n most confident loop closures.
- **Spectral:** Loop closure prioritization technique based on algebraic connectivity of a reduced pose graph [44].
- **LMA:** Levenberg-Marquardt algorithm. A technique for solving nonlinear least squares problems [2].
- **BA:** Bundle Adjustment refines pose estimates of mulitple visual observations of the same scene [73].
- **DGS:** Distributed Gauss-Seidel algorithm for pose graph optimization in distributed systems [19].
- **GNC:** Graduated Non-Convexity algorithm for optimizing pose graphs [82].
- **D-GNC:** Distributed version of GNC [70].