Summary

Autonomous Mobile Robots (AMRs) are steadily becoming more prevalent for everyday use. Appliances such as lawnmowers, vacuum cleaners and warehouse *AMRs* are becoming increasingly autonomous, which necessitates the utilization of exploration and patrolling capabilities. Exploration allows a *AMR* to navigate and map unknown environments while patrolling allows it to traverse and monitor known environments. These capabilities are, however, often researched and developed in isolation and as such, little research has been done to tie them together. To tie these fields together, this paper proposes a pipeline that facilitates exploration and patrolling. This pipeline utilizes various existing problems of the domain in conjunction, which provides a bridge from exploration to patrolling. The pipeline utilizes a pre-existing autonomous exploration package to map unknown environments

with the *Gmapping* slam algorithm. Drawing inspiration from the *Art Gallery Problem* concerning how to place cameras to achieve map coverage, The map generated during exploration is processed and converted to a simple polygon. After this, triangulation is applied to partition the obtained polygon into a set of triangles, in which centroids are used to place the waypoints designated for patrolling. These waypoints can then be used for closed-path generation, which can be provided to the *AMR*. This phase also supports the partitioning of the closed-paths for distribution to multiple *AMRs* for patrolling. The proposal was evaluated through simulations conducted in five different rooms of small to medium

size. Those experiments measured the idleness of closed-paths with different numbers of partitions. The experiments found that idleness decreased across all test rooms, in correlation with partitions. Additionally, the findings suggest that the area of the room affects the plateauing effect of idleness, albeit no large room was experimented on. The exploration time was evaluated as well, the paper found that, compared to patrolling time, optimization of the exploration technique is required.

The proposed solution showed that a pipeline facilitating patrolling and exploration is possible, however, further studies are needed to optimize the proposed solution and verify its impact in very large environments.

Pipeline for Exploration and Patrolling Routes with Waypoints and Triangulation for Autonomous Mobile Robots

Magnus Mathiesen (mmathi19@student.aau.dk)*, Daniel Morratz (dmorra18@student.aau.dk)*, and Jacob Carstensen (jcarst19@student.aau.dk)*
*Department of Computer Science
Aalborg University
9220 Aalborg Øst, DK

***Abstract*—Autonomous robots' availability is steadily increasing, although much research is needed to build autonomous robot systems. Patrolling and exploration facilitation are important for building such systems, however, these areas are often researched separately. This paper examines how a system for autonomous robots can provide a bridge from exploration to patrolling in a pipeline capable of achieving both. By taking inspiration from the art gallery problem, Delaunay triangulation is applied to an explored area to generate waypoints that act as closed path loops for robots to patrol. These closed-paths can be partitioned into groups to distribute patrolling routes among multiple robots. The pipeline is evaluated through a set of experiments, testing exploration and patrolling capabilities to measure exploration time, and idleness in comparison to multiple closed-paths.**

***Theme:* P10 - Specialisation in Distributed Systems**
***Project Period:* 01-02-2024 - 14-06-2024**
***Project Group:* cs-24-ds-10-05**
***Page number:* 10**
***Date of completion:* June 14, 2024**

***Index Terms*—swarm robotics, patrolling, exploration, simulation, autonomous mobile robots, triangulation, polygonization**

Acknowledgments

We express our gratitude to Michele Albano, for supervising this project and supporting us with invaluable insight and experience needed to accomplish the goals of this paper. We are also grateful for the continuous feedback provided throughout the project. We also thank the authors of the Linorobot project and multiple notable projects/forks [1]–[6], for providing a solid base for this paper to build upon. We also extend our gratitude to Casper Schou from the Department of Materials and Production at Aalborg University for his help with numerous hardware problems.

AI tools were used during the creation of this paper. Grammarly [7] and LanguageTool [8] were used for proofreading, and ChatGPT [9] and GitHub Copilot [10] were used to assist with debugging and writing code.

The source code for this project can be found on GitHub at: https://github.com/P10-Exploration-and-Patrolling-Pipeline.

A video demonstration of this project can be found on YouTube at: https://youtu.be/9Ss8LBo-RQM.

Aalborg University, June 14, 2024

I. Introduction

Autonomous mobile robots (AMRs) have seen increasing use across a wide range of industries, from household appliances such as lawnmowers [11] and vacuum cleaners [12] to industrial tasks in monitoring [13] and warehousing [14]. The adoption of *AMRs* has surged, with a significant increase in operational robot stock reported in recent years, [15], particularly benefiting sectors like hospitality, healthcare, transportation, and logistics [15]. Although individual robots are capable, their full potential is often realized through teamwork in applications like forest fire detection and terrain mapping [13], [16]. This collaboration presents challenges, including exploration and patrolling, making it a key research area in developing multi-AMR systems that focus on optimization for tasks like mine detection [17] and area patrolling [18] to enhance collective performance and safety. While exploration and patrolling are key components in building multi-AMR systems in various domains, strategies for these are still typically researched and developed independently.

This paper aims to develop a bridge between exploration and patrolling by creating a pipeline for generating a series of patrolling routes from a region that has been explored. This will make it feasible for robots to autonomously explore an area and subsequently devise patrolling strategies without the input of a human operator.

Once an area is explored, the map can be transformed into a simple polygon. *The Art Gallery Problem* (AGP) is applied to partition an explored area [19] into a set of triangles that can be turned into a series of waypoints that can in turn be applicable for path planning.

The following sections will describe a pipeline tying exploration and patrolling together. Section II will cover previous work related to the topics of path planning and exploration, while subsection III-A - subsection III-C will present the problems in more detail. The pipeline is presented in subsection III-D along with a technical description of its components, as well as which challenges must be addressed. The results are evaluated in section V and with possible shortcomings and improvements discussed in section VI. The paper is then summarized and concluded in section VII whereby future development goals are addressed.

This paper is proceeding work from a semester project [20] developed by the authors of this paper. The aforementioned project as well as this paper build upon the work from the *Linorobot2* project and numerous variations [1]–[6].

II. Background Information

Most work on both *AMR* exploration and patrolling systems focuses on either one or the other. This paper aims to bridge this gap, integrating both into one system. Both will be explored here, with further explanation and expansion in subsection III-B and subsection III-C respectively. *AGP* and *Triangulation* will also be explored in this section, and eventually applied in the pipeline in subsection III-D.

A. Exploration Problem

The exploration problem can be defined as to *allow a robot, without any external system other than its sensors, to create a map of [an] environment and locate itself into this map.* [21]

The *AMR* must orient itself in its surroundings to explore and navigate that unknown environment. It can generate a map of the area while traversing new yet unexplored parts of its environment. *Simultaneous Mapping and Localization (SLAM)* is a technique that enables robots to perform exactly this task.

In *SLAM*, a robot makes relative observations of landmarks in its environment, measuring the relative distances between these landmarks allows the robot to place itself in the environment space [22]. This allows *AMRs* to determine their location within an environment and simultaneously create a map of that environment using its measurements [22], [23]. The technique is suitable for various *AMR* platforms such as *Unmanned Ground Vehicles (UGV's)*, *Unmanned Aerial Vehicles (UAV's)* and *Autonomous Underwater Vehicle (AUV's)* that each specialize in different terrain types to traverse [22].

SLAM-based algorithms compare favourably in comparisons between different exploration types of exploration algorithms, such as in a paper by *Andreasen et al.* [24]. This paper uses *Random Ballistic Walk (RBW)* [25] as a baseline for comparison, with *SLAM-based* algorithms generally outperforming the alternatives. A further comparison by *Trejos et al.* [21] compares *SLAM-based* algorithms to determine the best-performing ones. *KARTO-SLAM* [26] stands out as a well-performing choice in most use cases, with relatively low resource consumption.

A variation of *SLAM* known as *Multi-SLAM* is an extension of the *SLAM* technique to allow multiple robots to explore simultaneously and message discoveries among themselves, providing a collaborative knowledge base for the *AMRs* [27].

The specific implementation used in this paper is explored in subsection III-B.

B. Patrolling Problem

The Patrolling Problem can be defined as *surveillance tasks using multiple mobile robots, which involve frequent visits to every point of the environment?* [28]

Patrolling algorithms allow *AMRs* to provide systematic coverage of an explored environment. This is essential for monitoring and surveillance tasks, where environmental changes should be acknowledged and monitored. These algorithms should be scalable to *multi-AMR* systems.

A series of comparisons between various patrolling algorithms were published by Portugal et al. [28], [29]. Each environment was modelled as a graph, where the average graph idleness was used as an evaluation metric. This means that the time-delta between subsequent visits to a node in the environment graph should be as low as possible. The *Cyclic Algorithm for Generic Graphs (CGG)* [30] performed well across all the environments with varying amounts of *AMRs*. This algorithm generates a closed-path for each *AMR* to follow in a loop. However, worthy of note is that all algorithms improve dramatically, with all of them eventually converging on the same result if the number of *AMRs* is increased. This makes sense since the number of *AMRs* will eventually equal the number of nodes, making the optional strategy to stand still at a waypoint.

A notable difference between patrolling algorithms is whether they use online or offline planning strategies [31]. Online planning makes decisions in real-time, whereas offline planning utilizes pre-determined strategies. *CGG* uses offline planning, as it generates the closed-paths beforehand. If one *AMR* should become unresponsive, then *CGG* could be executed again, creating updated closed-paths for the number of remaining *AMRs*.

The specific implementation used in this paper is explored in subsection III-C.

C. Art Gallery Problem (AGP)

AGP is defined as *how many cameras do we need to guard a given gallery, and how do we decide where to place them?* [19]. This type of problem is similar to the challenges of *AMR* patrolling to ensure all areas are visited by a robot. Especially for surveillance or security tasks, minimizing blind spots is important to prevent breaches.

However, some key differences arise. When solving the art gallery problem, the solution involves placing stationary cameras to cover the entire area at various locations. Robots, especially *AMRs* are seldom stationary. Thus, the solution turns toward finding patrolling routes *AMRs* can travel to surveil an area.

D. Triangulation

Triangulation plays a significant role in solving the art gallery problem as a method of implementing efficient patrolling strategies for efficient coverage and movement of AMRs. This involves the decomposition of a polygon into triangles [19]. A method developed by B. Delaunay triangulates and ensures a maximal set of non-intersecting diagonals, such that the circumcircle of each triangle includes no points [32]. Decomposing a polygon in this manner avoids *sliver triangles* [32] which undesirable properties can lead to worse results [33]. Using this method, each *AMR* is assigned to patrol a set of waypoints based on the generated triangles for various tasks such as data gathering or monitoring.

III. Methodology

This section describes a pipeline that bridges the gap between exploration and patrolling tasks. Implementations of solutions to the problems described in subsection II-A and subsection II-B are detailed, with the pipeline being described in subsection III-D.

A. Architecture

A reference architecture for the system is illustrated in Figure 1. There is a *Coordination Server* which is responsible for communicating with the AMRs and continuously receiving information about map exploration, positioning, and patrolling. The *Coordination Server* collects the maps from the AMRs and provides them with closed-paths.

Both the simulated and physical environment are designed such that each AMR is equipped with a navigation server. This allows the AMRs to patrol, make decisions and execute tasks independently of each other and of the *Coordination Server*. The only thing it depends on from other systems is the closed-paths from the *Coordination Server*.

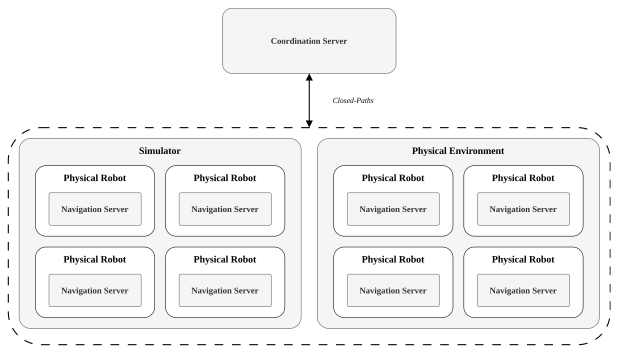


Fig. 1: Autonomous Reference Architecture for the *AMR* system.

Figure 2 shows the communication process in the pipeline. The communication process starts when autonomous exploration is initiated in the *explore lite* node. A start flag is then published to the *exploration listener* node to track the exploration progress. Once an exploration is finished and an environment is fully mapped, it will send a file path to the *Map Server* which shows where it should store the map of the explored area. The next step is then to publish the file location to the *Map Polygonization* node that is responsible for path generation, as shown in Figure 3

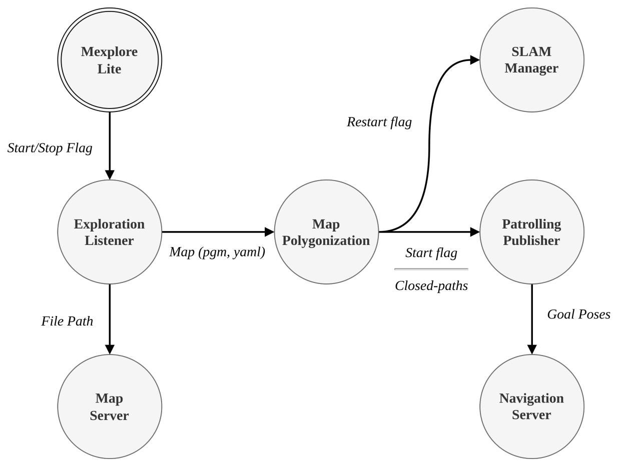


Fig. 2: Pipeline communication diagram.

The *Slam Manager* node's purpose is to start slam-toolbox with the proper configuration. After the *Map Polygonization* node is finished, the *Slam Manager* will be restarted to switch from localization to mapping mode to prepare for patrolling. The *Patrolling Publisher* node gives the closed-path to the *AMRs* as a set of goal poses to the *Navigation Server* which keeps track of which goal pose is the next destination.

B. Exploration Task

The *AMRs* need to initially traverse and map an environment, by using the *SLAM* method, as detailed in subsection II-A. This created map will eventually be used as the patrolling environment.

The specific *AMRs* used in this paper utilize the *Robot Operative System 2 (ROS2)* [34], which supports the *SLAM* library *SLAM-Toolbox* [35]. *SLAM-Toolbox* utilizes a modified version of *Karto-SLAM*, however, while the *SLAM-Toolbox* package provides the ability to map and localize within an environment, it does not provide the ability to *SLAM* autonomously without further additions.

The package known as *explore lite* [36] provides, in combination with the *ROS2* stack, the ability to explore and localize autonomously. The package was originally authored by J. Hörner [37] for *ROS*, and later ported to *ROS2*. It utilizes a frontier-based exploration method, based on work by B. Yamauchi [38]. However, due to the specific implementation of *explore lite*'s map merging, the *Gmapping* algorithm [39] is used instead of *Karto-SLAM*. Switching the specific *SLAM* algorithm could improve the performance of the overall system, and is discussed in subsection VI-B. *SLAM-Toolbox* and *Karto-SLAM* are still utilized in the patrolling task in subsection III-C.

C. Patrolling Task

The pipeline discussed in subsection III-D generates closed-paths patrolling routes for the *AMRs* to follow. These closed-paths are created from the map generated during the exploration task, as discussed in subsection III-B. This necessitates a topological graph representation of the environment to create patrolling routes through the *CGG* algorithm.

The graph is created from waypoints, which are generated through a modified implementation of the *AGP* [19] introduced in subsection II-C. Each closed-path is a series of these waypoints for each *AMR* to visit. One patrolling task involves visiting each waypoint in a specific order, which can be repeated indefinitely for continuous patrolling. This constitutes an offline planning strategy, where each *AMR* is assigned a closed-path at the start of patrolling, and they do not make any decisions during the task besides creating paths between waypoints and obstacle avoidance. This additional functionality is provided by *SLAM-Toolbox*, using *Karto-SLAM* to update the environment continuously.

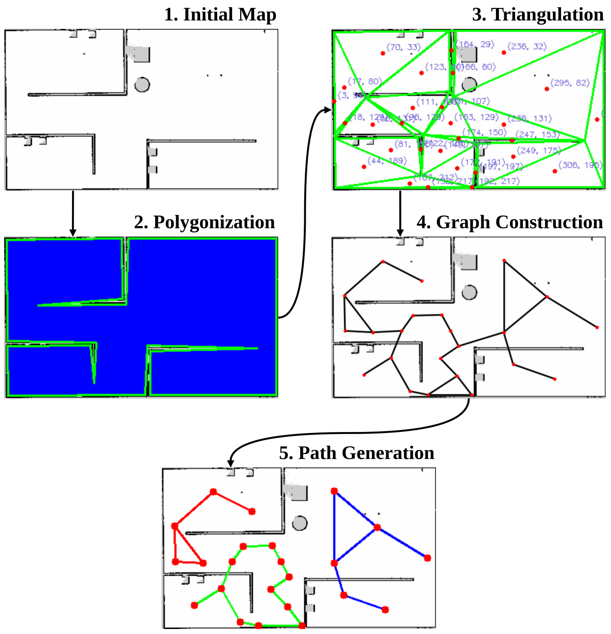


Fig. 3: Pipeline for generating closed-paths on map.

D. Pipeline

The pipeline converts a map to a set of *n* patrolling routes. The process starts with an initial map, acquired from the exploration step in subsection II-A. Four major phases follows; polygonization, triangulation, graph construction, and path generation. The entire process is illustrated in Figure 3.

The polygonization and triangulation phases take inspiration from the *AGP*, whereas the remaining phases are purpose-built for the pipeline.

The pipeline takes a map as a `.pgm` file, with a complementary `.yaml` file that describes the size and origin of the map, as input to start processing.

The map is initially preprocessed by applying a *Gaussian blur* [40], [41] to remove imperfections from the *SLAM* mapping output. A binary image that distinguishes between walls and floor area is created. Contours are found in the binary image, and filtered based on a predetermined threshold. The polygonization process is applied to these contours to create a 2-dimensional polygon which represents the layout of the map. The contours and polygon are shown in green and blue respectively in phase 2 in Figure 3.

The polygon is triangulated in phase 3 by applying the Delaunay triangulation method [32], [42]. Centroids are then found in the middle of each triangle. These centroids are designated as waypoints in the eventual patrolling routes. This differs from *AGP*, which places its surveillance points at the vertices of the triangles instead of in the centre. While placing a waypoint in the corner of a room is the most efficient approach with cameras, these waypoints end up along the walls of a room. The *AMRs* should instead enter an area and traverse paths that provide better coverage and flexibility for a moving surveillant. A discussion about potential improvements to achieve this goal and what constitutes good coverage can be found in subsection VI-B.

A graph representation can then be created from the waypoints in phase 4. Waypoints are connected by placing an edge to their nearest neighbour. However, this introduces problems that need to be accounted for, as the triangulation phase can create triangles and waypoints inside and outside of the polygon. All waypoints that are generated outside the polygon need to be removed as it is unreachable for an *AMR*.

In some cases, a nearest neighbour can reside on the other side of a wall. To prevent the creation of those edges, a check is made to determine whether an edge intersects a wall prior to addition. This prevents an *AMR* from traversing from one side of a wall to another, even if the waypoint coordinates are nearest neighbours. If such an intersection happens, the edge is saved in a list of rejected edges for later use. After all waypoints are connected, another check is made to search for disconnected sections in the graph called *islands*.

As illustrated in Figure 4, if such an island is discovered, an edge from the rejected edges list is added to the graph regardless of the edges' intersection with walls. This is acceptable due to the *AMRs* obstacle avoidance capabilities, ensuring it will navigate around any wall or obstacle to reach to the waypoint.

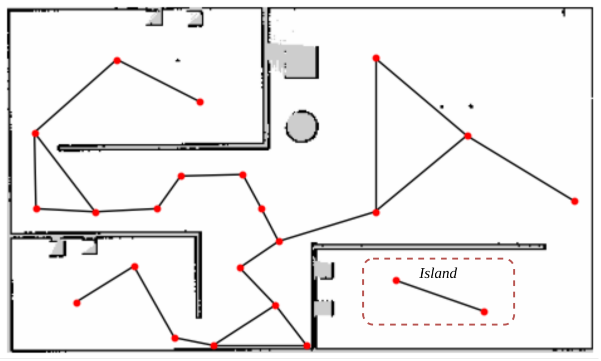


Fig. 4: Graph with two-node island.

Finally, the graph is partitioned into *n* roughly equally large subgraphs (one for each *AMR*) in phase 5, using the *Girvan-Newman* algorithm [43], [44] for community cluster detection. A closed-path can be generated from each subgraph utilising an algorithm for the *Travelling Salesman Problem* [45], [46]. The reason the graph isn't simply divided into *n* subgraphs with an equal number of nodes is that this method can lead to unbalanced results. The goal is to create closed-paths where each *AMR* takes the same amount of time to patrol its respective path. Two examples in Figure 5 illustrate this issue. In the left example, there are four roughly equal subgraphs. The blue subgraph has only five nodes but would require a significant amount of travel to reach a sixth node, making it comparable in size to the other subgraphs. An *AMR* travelling to the nearest nodes would take longer than if another *AMR* included those nodes in its closed-path. Similarly, in the right example, the blue subgraph has two fewer nodes than the green one. However, the distance to any nodes in the green subgraph is so great that it is more efficient for the *AMR* in the green subgraph to travel to the extra nodes than for the *AMR* in the blue subgraph to travel to any nodes in the green subgraph.

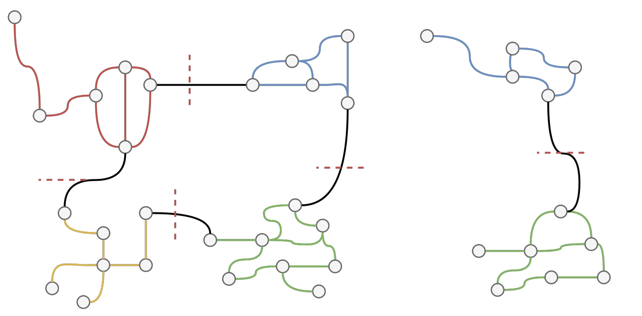


Fig. 5: Graph division with 4 paths (left) and 2 paths (right).

IV. Experimental Setup

Exploration was compared to the patrolling paths provided by the pipeline, to measure the impact of knowing the map in advance. The cycle time is captured during these phases of the experiments. Furthermore, key measures for patrolling algorithms are measured in the form of the worst and mean idleness of the patrolling routes during the experiments. Idleness in this case refers to the time between two visits to the same waypoint. The worst idleness should represent the longest time-slot where an area is unobserved. Overall, these measurements support the feasibility of a patrolling pipeline of this type. This is measured for 1–4 partitions, where a closed-path is generated for each partition of the map. This is to measure the impact that partitioning the closed-path and distributing the partitions to multiple *AMRs* has on the idleness.

Due to constraints (more on this in VI), a single robot has been utilized to carry out the experiments. However, a single patrolling robot can still give valuable results and emulate a multi-AMR patrolling scenario by traversing each closed-path sequentially and measuring the idleness of each closed-path independently. This ensures results that are similar to experimenting with multiple robots patrolling simultaneously. This should not impact the idleness metric, i.e. the worst idleness of the graph is simply equal to the worst idleness among the sequential closed-paths.

A. Simulation Setup

All experiments and simulations for this study are run on an *Ubuntu 22.04.4 LTS* system running on an *Intel i7-8700* CPU at 3.2 GHz, 16 GB of DDR4 2666MHz RAM and an *Nvidia 1060* with 3Gb of VRAM.

For each experiment, two parameters are provided to the pipeline, a `World` (environment) and `Number of Cycles`. The world parameter is the map used for the simulation and provided to the Gazebo simulator, and the number of cycles is the number of times the patrolling cycle will be performed. All experiments are run for 10 cycles. These parameters can be specified at startup, allowing for practical control contrary to hard-coding parameters directly in the pipeline. It also allows for setting up multiple experiments to be run in sequence.

To provide a variety of experimental scenarios, different rooms were selected, which is a set of experiment rooms [47] and a small warehouse [48] building to represent a more real-world-like environment. Each of the three experiment rooms increases in complexity, with each adding more obstacles. These experiment environments can be seen in Figure 6 and Figure 7 as well as full-scale images can be seen in Appendix F.

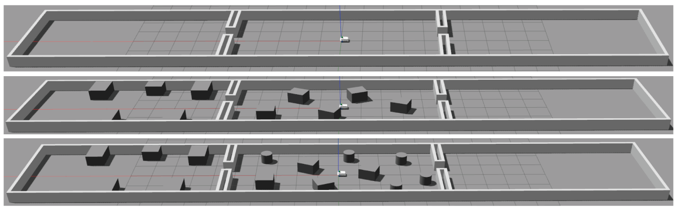


Fig. 6: Experiment room *No obstacles* (Top), *some obstacles* (Middle) and *Many obstacles* (Bottom) displayed collectively.

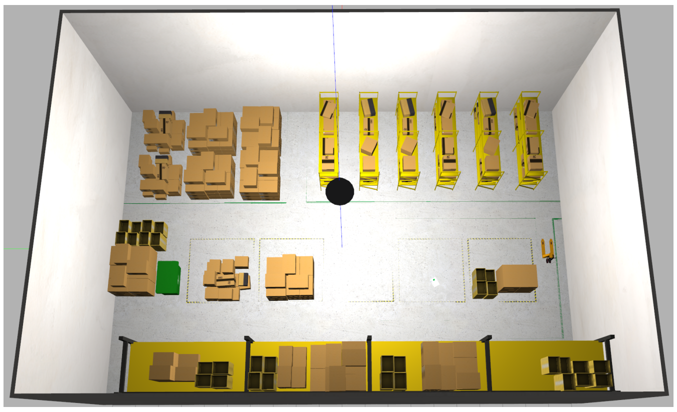


Fig. 7: Small warehouse Environment.

B. *Physical Setup*

The *AMRs* are differential drive robots consisting of two drive wheels capable of moving independently of each other and a free-turning swivel wheel. They are furthermore equipped with a LIDAR sensor for mapping the world, an IMU, a tick sensor for localization and a *Raspberry Pi 4* and *Teensy* microcontroller for logic flow.

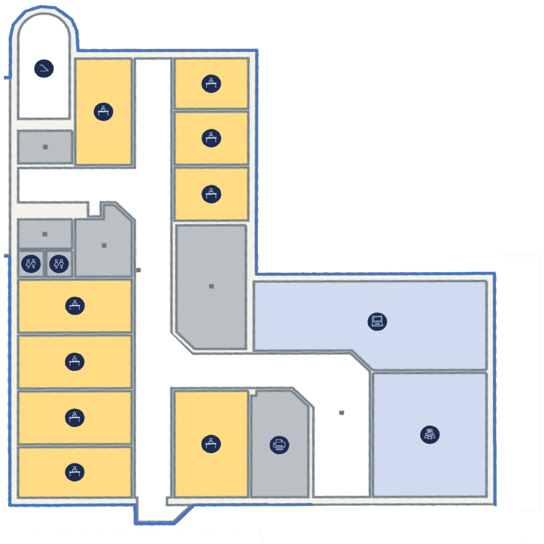


Fig. 8: Physical Exploration Area.

The experiments are carried out in the hallway shown in Figure 8 located at Aalborg University to test the physical robots in different environments. As with the experiments described in subsection IV-A a parameter `Number of Cycles` is given for each experiment and the idleness of a closed-path and cycle iteration is measured.

V. Results

This section will focus on the simulated results. The physical robots were unable to perform the experiments described in section IV due to hardware issues. This is described further in Appendix H.

A. *Simulation results*

The results of the idleness experiments are plotted in Figure 11 and Figure 12. They show the *Worst Idleness* and *Average Idleness* on the whole map across 10 patrolling cycles, with 1-4 closed-paths on each map. We can see a correlation between idleness and the number of *AMRs*, with average and worst idleness decreasing as the amount of closed-paths increases. Interestingly, some maps seem to plateau when the number of *AMRs* becomes large enough, lessening the benefit from adding more *AMRs*. The point where this plateau occurs should happen at a higher amount *AMRs* if the map is larger, and smaller if the map is smaller. Some results have been highlighted in this section as plots, with the rest of them being in Appendix G.

The average time to patrol a closed-path on a given map can be seen in Figure 13 for four closed-paths. Each patrol has been executed 10 times. This shows the relative difference between one patrol run and another. Some of the maps show little difference between subsequent runs, whereas others show a general downward trend.

A general trend across all five maps is the decrease in time as the cycle iterations increase. It is believed that the *AMRs* become more familiar with the map over time, as a result of its cost-map function, and thus find it easier to navigate. Such that it in a way becomes more "confident" in its navigational abilities on subsequent runs. The average time for all experiments on the warehouse map can be seen in Figure 34. This shows the improvement in the average time per closed-path when more *AMRs* are added to the patrolling. Each showed improvements when going from one to two closed-paths, with marginal improvements thereafter. Larger maps should show more pronounced differences when adding more *AMRs*.

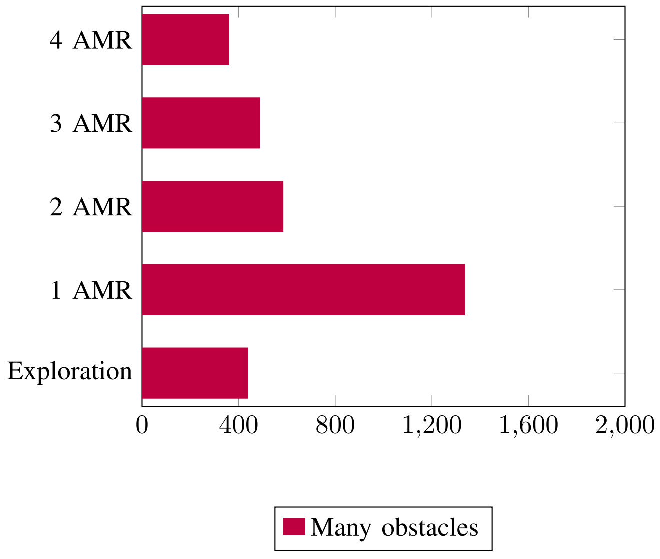


Fig. 9: Time to run exploration vs patrolling averaged over 10 executions.

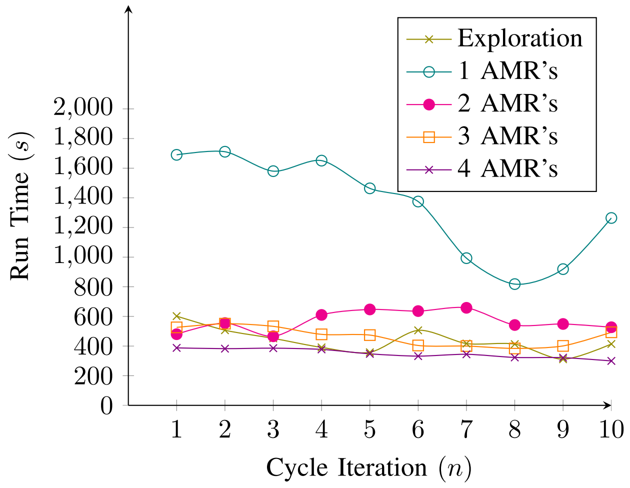


Fig. 10: Time to run exploration vs patrolling for 10 iterations.

A histogram of the average exploration time compared to the patrolling time of the *Many Obstacles* map can be seen in figure Figure 9. Whilst the results in Figure 9 indicate that exploration would be the faster option in comparison to patrolling, it is not necessarily possible to conclude. It can be seen in Figure 10 that the range of time spent on exploration varies on the currently simplistic map. If this were to be simulated on a much more complex map, it could indicate that exploration is not as feasible. A larger test environment should make this difference more pronounced. It also indicates that the cycle-to-cycle variance becomes less when the number of *AMRs* increases.

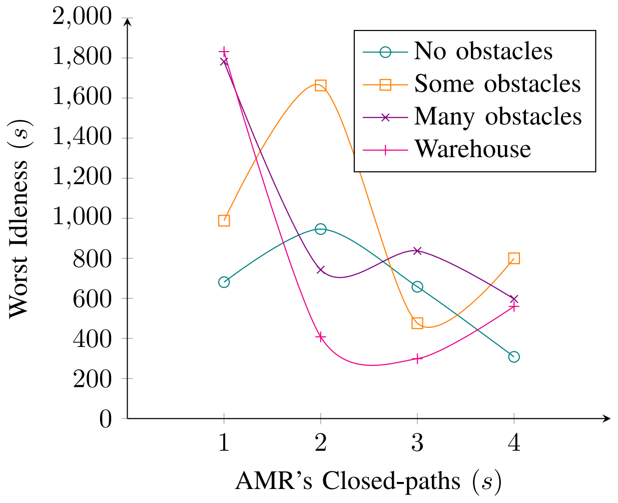


Fig. 11: Worst Idleness over the number of closed-paths averaged over 10 runs.

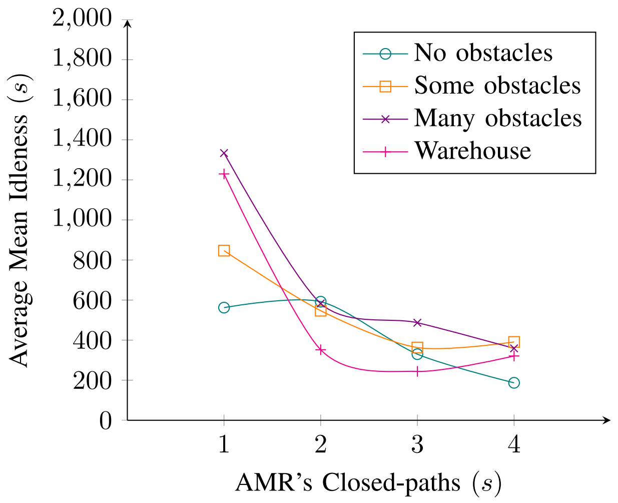


Fig. 12: Average Idleness over the number of closed-paths averaged over 10 runs.

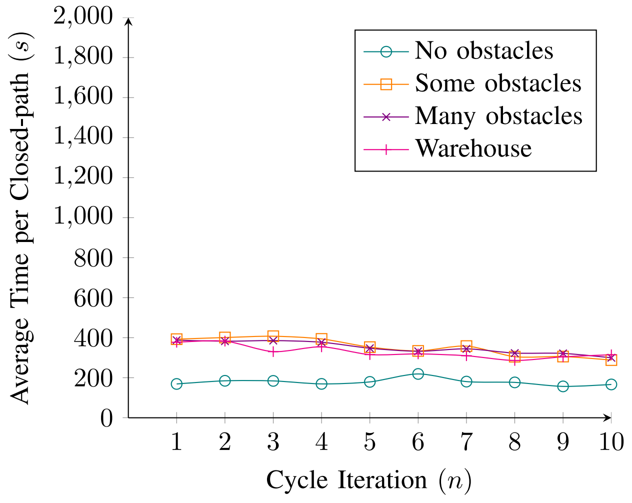


Fig. 13: Time over cycle iteration with 4 AMRs.

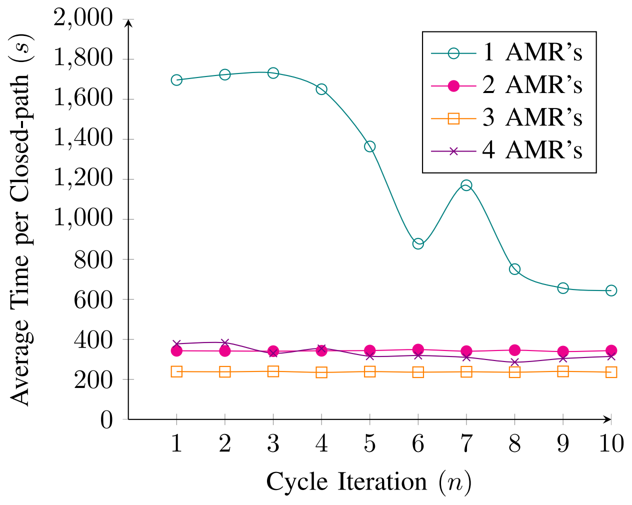


Fig. 14: Time over cycle iteration in Warehouse map.

VI. Reflections & Future work

This section explores possible improvements to the system, while also documenting some shortcomings.

A. *Shortcomings*

1) *Exploration:* The currently implemented exploration algorithm does not guarantee a fully explored map due to the random nature of the RBW. Additionally, for a given map, it can be difficult to tweak the exploration parameters optimally. This means that the patrolling task is more reliable if one wants to strive for coverage of the entire environment predictably.

2) *Larger Environments:* The results in section V suggest that the effect of the number of closed-paths in a patrolling plan might be correlated to the size of the patrolled environment. Hence, the testing scenarios could be expanded to include larger maps with features such as large disconnected areas, corridors, and large open areas. This would show the impact of multiple *AMRs* in complex environments.

3) Traction on Physical AMR: The physical *AMRs* were unable to perform the physical experiments, as mentioned in section V. This is largely due to slippage issues with the *AMRs* wheels. Possible solutions to this could include different wheels, as well as performing the tests in a different environment with less slippery terrain.

4) Multi-AMR support: The exploration and patrolling task currently only supports a single *AMR*. This is due to two problems. The first being the traction problems mentioned above, and the second being time constraints. There was not enough time to implement proper multi-AMR support, so the multi-AMR patrolling task was emulated using just one *AMR*, as explained in section IV. A future work task would be to include this feature.

5) Karto-SLAM in Explore Lite: The exploration algorithm described in subsection II-A, *Karto-SLAM*, was only implemented in the patrolling task, whereas the exploration task ended up utilizing a different algorithm. One of the advantages of *Karto-SLAM* is its low resource usage, which is ideal for physical *AMR* which might be resource-limited. For instance, the *AMRs* used for the experiments in this paper run on a *Raspberry Pi 4 8Gb*, so implementing a more efficient exploration algorithm might improve performance. It would also reduce the complexity of the system if the same *SLAM* algorithm could be used for both the exploration and patrolling tasks.

B. Improvements

1) Clustering of waypoints: The graph construction phase of the pipeline in subsection III-D could potentially be improved by reducing the number of waypoints in the final graph. This could be achieved with clustering of waypoints, where a community detection algorithm could be used to locate areas where multiple waypoints are in the same room or otherwise close to each other. The waypoints could be reduced to just one waypoint, reducing the number of waypoints that the *AMRs* need to traverse.

2) Highway detection: Some environments might have long corridors where the *AMRs* potentially could increase their movement speed safely. These corridors could function as highways for the *AMRs*. Detecting these highways might allow the *AMRs* to traverse their patrolling paths quickly.

3) Keep out zones: Complex real-world environments often have areas where patrolling is redundant or unimportant. Below tables, behind furniture, etc. Being able to beforehand select areas where the *AMRs* do not need to patrol would save time the *AMR* would otherwise have spent patrolling areas that are not of interest.

4) Robot disconnection: The current system sends a closed-path to each *AMR*, and lets them patrol their environment based on that. The system could be made more fault-tolerant if the coordination server mentioned in subsection III-A could detect unresponsive *AMRs*. Then the system could run the pipeline again with $n - 1$ *AMRs* and generate new closed-paths for the remaining *AMRs*.

VII. Conclusions

This paper proposed a pipeline that allows *AMRs* to explore an environment and automatically generate patrolling routes to cover that environment.

The pipeline utilized the Art Gallery Problem and Delaunay triangulation to create a series of waypoints that capture the areas of the environment that need to be patrolled. Closed-paths were constructed from these waypoints, creating closed-paths for each *AMR* to follow. They visit each waypoint in order in a cyclic manner.

The patrolling routes were tested in five simulated environments, where the pipeline was shown to generate feasible closed-paths effectively for patrolling. Worst and average idleness were measured at each waypoint to show the incremental improvements to the patrolling task from partitioning the environment into additional closed-paths. Exploration times have also been measured and compared to patrolling time, albeit the results were suboptimal. The patrolling setup provides a guarantee of coverage and less variance from cycle to cycle than the exploration setup, making the patrolling setup a viable solution for patrolling tasks.

While this paper shows indications that a pipeline can generate patrolling routes in conjunction with exploration, further studies still need to be done to optimize and verify if the impacts will be similar on a larger scale.

The software stack used in this project has been released as open source, allowing for usage in future projects.

Bibliography

[1] J. M. Jimeno, *Linorobot*, 2017. [Online]. Available: https://linorobot.org/ (visited on 02/05/2024).

[2] Linorobot, *Linorobot 2*, 2023. [Online]. Available: https://github.com/linorobot/linorobot2 (visited on 02/05/2024).

[3] AAU Smart Production Lab, *Linorobot 2*, 2022. [Online]. Available: https://github.com/AAUSmartProductionLab/linorobot2 (visited on 02/05/2024).

[4] A. Clement, *Linorobot 2*, 2022. [Online]. Available: https://github.com/Anders-Clement/linorobot2 (visited on 02/05/2024).

[5] A. Clement, *Linorobot 2 hardware*, 2023. [Online]. Available: https://github.com/Anders-Clement/linorobot2_hardware (visited on 02/05/2024).

[6] O. B. Lauritsen, J. Andersen, A. Clement, and C. Schou, *Traffic management for swarm production*.

[7] Grammarly Inc., *Grammarly*, 2024. [Online]. Available: https://www.grammarly.com/ (visited on 06/11/2024).

[8] LanguageTool, *Languagetool*, 2024. [Online]. Available: https://languagetool.org/ (visited on 06/11/2024).

[9] OpenAI, *Chatgpt*, 2024. [Online]. Available: https://openai.com/index/chatgpt/ (visited on 06/11/2024).

[10] GitHub Inc., *Github copilot*, 2024. [Online]. Available: https://github.com/features/copilot (visited on 06/11/2024).

[11] J.-C. Liao, S.-H. Chen, Z.-Y. Zhuang, B.-W. Wu, and Y.-J. Chen, "Designing and manufacturing of automatic robotic lawn mower," *Processes*, vol. 9, no. 2, 2021, ISSN: 2227-9717. DOI: 10.3390/pr9020358. [Online]. Available: https://www.mdpi.com/2227-9717/9/2/358.

[12] D. Etherington, *Irobot says 20 percent of the world's vacuums are now robots*, 2016. [Online]. Available: https://techcrunch.com/2016/11/07/irobot-says-20-percent-of-the-worlds-vacuums-are-now-robots/ (visited on 02/23/2024).

[13] M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 24–39, 2012. DOI: 10.1109/MRA.2011.2181683.

[14] R. Bogue, "Growth in e-commerce boosts innovation in the warehouse robot market," *Industrial Robot: An International Journal*, vol. 43, no. 6, pp. 583–587, Jan. 2016, ISSN: 0143-991X. DOI: 10.1108/IR-07-2016-0194. [Online]. Available: https://doi.org/10.1108/IR-07-2016-0194.

[15] *World robotics 2022*, 2022. [Online]. Available: https://ifr.org/downloads/press2018/2022_WR_extended_version.pdf.

[16] T.I.E. Industrial, *Benefits of industrial robots*, 2014. [Online]. Available: https://www.robots.com/articles/benefits-of-robots.

[17] V. N. K, K. B, N. B. Poojari, Dhanush, and D. R. K. M, "Swarm robotics for mine detection," *International Journal of Creative Research Thoughts Robotics Reports*, vol. 10, Jul. 2022, ISSN: 2320-2882. [Online]. Available: https://ijcrt.org/papers/IJCRT2207409.pdf.

[18] N. Basilico, "Recent trends in robotic patrolling," *Current Robotics Reports*, vol. 3, no. 2, pp. 65–76, Jun. 2022, ISSN: 2662-4087. DOI: 10.1007/s43154-022-00078-5. [Online]. Available: https://doi.org/10.1007/s43154-022-00078-5.

[19] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008, ch. 3, pp. 45–59, ISBN: 3540779736.

[20] M. Mathiesen, D. Morratz, and J. Carstensen, *Robuddy: A platform for exploration and patrolling with swarm robotics - pre-specialisation in distributed systems*, 2024.

[21] K. Trejos, L. Rincón, M. Bolaños, J. Fallas, and L. Marín, "2d slam algorithms characterization, calibration, and comparison considering pose error, map accuracy as well as cpu and memory usage," *Sensors*, vol. 22, no. 18, 2022, ISSN: 1424-8220. DOI: 10.3390/s22186903. [Online]. Available: https://www.mdpi.com/1424-8220/22/18/6903.

[22] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. DOI: 10.1109/MRA.2006.1638022.

[23] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006. DOI: 10.1109/MRA.2006.1678144.

[24] M. Z. Andreasen, P. I. Holler, M. K. Jensen, and M. Albano, "Comparison of online exploration and coverage algorithms in continuous space," in *Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 1: SDMIS*, INSTICC, SciTePress, 2022, pp. 527–537, ISBN: 978-989-758-547-0. DOI: 10.5220/0010975900003116.

[25] M. Kegeleirs, D. G. Ramos, and M. Birattari, "Random walk exploration for swarm mapping," in *Towards Autonomous Robotic Systems*, K. Althoefer, J. Konstantinova, and K. Zhang, Eds., Cham: Springer International Publishing, 2019, pp. 211–222, ISBN: 978-3-030-25332-5.

[26] X. S. Le, L. Fabresse, N. Bouraqadi, and G. Lozenguez, "Evaluation of out-of-the-box ros 2d slams for autonomous exploration of unknown indoor environments," in *Intelligent Robotics and Applications*, Z. Chen, A. Mendes, Y. Yan, and S. Chen, Eds., Cham: Springer International Publishing, 2018, pp. 283–296, ISBN: 978-3-319-97589-4.

[27] A. Howard, "Multi-robot simultaneous localization and mapping using particle filters," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1243–1256, 2006. DOI: 10.1177/0278364906072250. eprint: https://doi.org/10.1177/0278364906072250. [Online]. Available: https://doi.org/10.1177/0278364906072250.

[28] D. Portugal and R. P. Rocha, "On the performance and scalability of multi-robot patrolling algorithms," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2011, pp. 50–55. DOI: 10.1109/SSRR.2011.6106761.

[29] D. Portugal and R. P. Rocha, "Multi-robot patrolling algorithms: Examining performance and scalability," *Advanced Robotics*, vol. 27, no. 5, pp. 325–336, 2013. DOI: 10.1080/01691864.2013.763722. eprint: https://doi.org/10.1080/01691864.2013.763722. [Online]. Available: https://doi.org/10.1080/01691864.2013.763722.

[30] Y. Elmaliach, N. Agmon, and G. A. Kaminka, "Multi-robot area patrol under frequency constraints," *Annals of Mathematics and Artificial Intelligence*, vol. 57, no. 3, pp. 293–320, Dec. 2009, ISSN: 1573-7470. DOI: 10.1007/s10472-010-9193-y. [Online]. Available: https://doi.org/10.1007/s10472-010-9193-y.

[31] N. Agmon, N. Hazon, G. A. Kaminka, and T. M. Group, "The giving tree: Constructing trees for efficient offline and online multi-robot coverage," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2, pp. 143–168, Apr. 2008, ISSN: 1573-7470. DOI: 10.1007/s10472-009-9121-1. [Online]. Available: https://doi.org/10.1007/s10472-009-9121-1.

[32] SciPy, *Spatial data structures and algorithms*, 2024. [Online]. Available: https://docs.scipy.org/doc/scipy/tutorial/spatial.html.

[33] H. Edelsbrunner, X.-Y. Li, G. Miller, *et al.*, "Smoothing and cleaning up slivers," in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '00, Portland, Oregon, USA: Association for Computing Machinery, 2000, pp. 273–277, ISBN: 1581131844. DOI: 10.1145/335305.335338. [Online]. Available: https://doi.org/10.1145/335305.335338.

[34] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[35] S. Macenski and I. Jambrecic, *Slam toolbox*, 2024. [Online]. Available: https://github.com/SteveMacenski/slam_toolbox.

[36] C. A. Á. R. Tim Clephas Juan Galvis, *M-explore ros2 port*. [Online]. Available: https://github.com/robo-friends/m-explore-ros2 (visited on 05/16/2024).

[37] J. Hörner, *Map-merging for multi-robot system*, Bachelor's Thesis, Prague, 2016. [Online]. Available: https://is.cuni.cz/webapps/zzp/detail/174125/.

[38] 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.

[39] R. Yagfarov, M. Ivanou, and I. M. Afanasyev, "Map comparison of lidar-based 2d slam algorithms using precise ground truth," *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1979–1983, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:56596065.

[40] OpenCV, *Smoothing images*, 2024. [Online]. Available: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.

[41] OpenCV, *Smoothing images*, 2024. [Online]. Available: https://docs.opencv.org/3.4/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html.

[42] B. Delaunay, "Sur la sphère vide," pp. 793–800, 1934, ISSN: 0143-991X. DOI: 10.1108/IR-07-2016-0194. [Online]. Available: https://www.mathnet.ru/links/88ee44ca375c579737c255bbe74cf627/im4937.pdf.

[43] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. DOI: 10.1073/pnas.122653799. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.122653799. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.122653799.

[44] NetworkX, *Girvan_newman*, 2024. [Online]. Available: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.centrality.girvan_newman.html.

[45] J. B. Robinson, *On the Hamiltonian game (a traveling-salesman problem).* Santa Monica, CA: RAND Corporation, 1949.

[46] NetworkX, *Traveling_salesman_problem*, 2024. [Online]. Available: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.approximation.traveling_salesman.traveling_salesman_problem.html.

[47] Melih Erdogan, *Dataset-of-gazebo-worlds-models-and-maps*, 2019. [Online]. Available: https://github.com/mlherd/Dataset-of-Gazebo-Worlds-Models-and-Maps (visited on 06/08/2024).

[48] AWS Robotics, *Aws robomaker small warehouse world*, 2020. [Online]. Available: https://github.com/aws-robotics/aws-robomaker-small-warehouse-world (visited on 06/13/2024).

[49] Notion, *Notion*, 2024. [Online]. Available: https://www.notion.so.

Appendix A
Preliminaries

A. *Polybot*

The robots used in this paper were provided by the *Department of Distributed Systems* at *Aalborg University*.

B. *Linorobot2*

Linorobot2 is a package designed to ease the setup of physical robots. It contains a set of launch files and hardware drivers for different hardware components and models. The Polybots have been specifically designed to work with Linorobot2, making the use of it an obvious choice.

Appendix B
Implementation

This section shows new components added to the project, as well as some changes that have been made to existing software.

A. *RPLIDAR changes*

Minor changes had to be made to the RPLIDAR package as there was no launch file working for the RPLIDAR A1, as well as the scan mode had to be changed to function with the RPLIDAR A1.

B. *Explore Lite changes*

The Explore lite was changed to publish as a message once the exploration was finished. This allowed us to start polygonization, triangulation and generating patrolling routes.

C. *Exploration listener*

An exploration listener was implemented to listen for the end exploration flag by Explore lite to start the initial pipeline. This node also saves the map from the exploration and publishes the file path to map polygonization

D. *Map polygonization*

This node was made to receive the destination of the explored map and then activate all the internal triangulation, path generation and partitioning. Furthermore, this node publishes the file path to the patrolling publisher with where the patrolling path waypoints are located.

E. *Patrolling Publisher*

The purpose of the patrolling publisher is to run the CGG patrolling algorithm from the waypoints provided at the path received from polygonization. This node will also keep track of running either a set number of cycles or running indefinitely.

F. *Map CLI Publisher*

A CLI-based map publisher was made as a debug tool which allowed for publishing a map path directly to map polygonization. This made it possible to run the pipeline from the map polygonization phase, thereby bypassing the exploration stage and saving time during development.

Appendix C
Project Management

A. *Time Management*

A rough time-schedule was made at the beginning of the project period. This schedule can be seen in Figure 15. An updated version at the end of the project period can be seen in Figure 16.

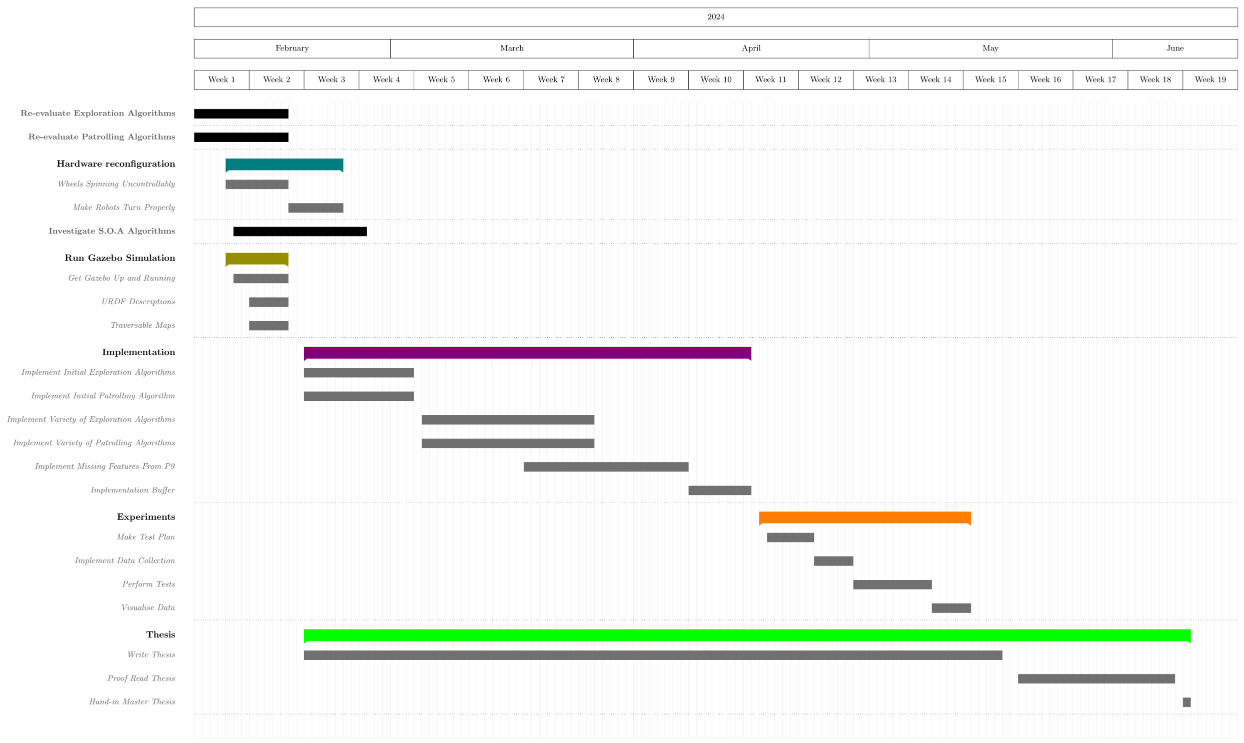


Fig. 15: Gantt Chart at beginning of project period.

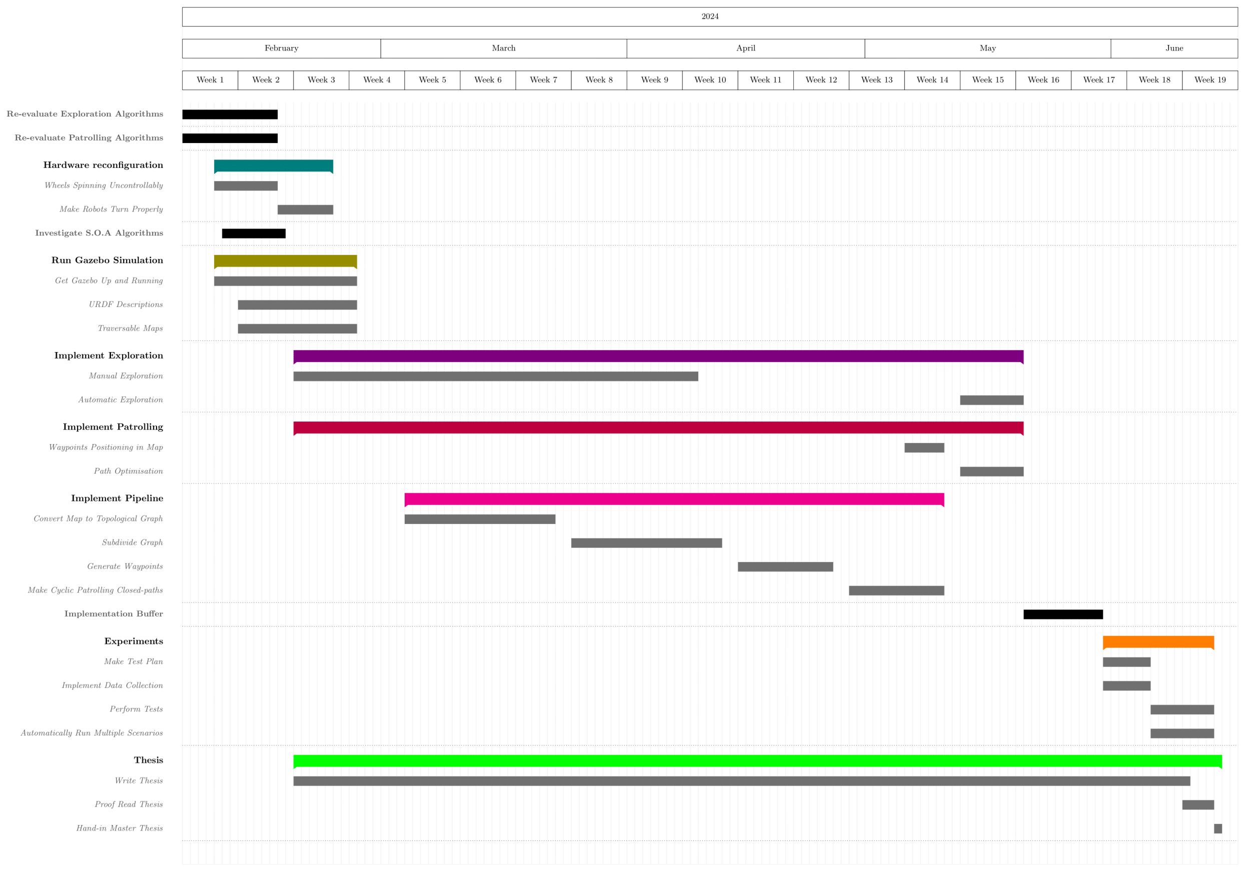


Fig. 16: Gantt Chart at end of project period.

The charts are distinctly different in the sense that the overall project direction changed throughout the project. Exploration was de-emphasised and the pipeline bridge between exploration and patrolling was added. The *Implementation* on the initial chart was split into three distinct groups of tasks, namely exploration, patrolling, and the pipeline. Separating these tasks allowed them to be worked on concurrently, and for each to gain more specific requirements. Some phases of the project also ended up taking a larger amount of time than at first expected. This was due to multiple tasks requiring a lot of configuration of the software stack, which revealed numerous bugs that interrupted progress on the overall project. This led to it being a very iterative process in which some tasks would have to be reset with different approaches, trying different configurations etc. Some of the major issues that led to a lot of time being spent were:

- **Incorrect Firmware:** The Firmware available for the teensy microcontrollers was inoperable. Once the correct firmware was acquired, a recurring issue was that the teensy microcontroller at random was unable to be flashed with new firmware when changes were made.
- **ROS2 Topic availability:** At stages of the process we had ROS2 topics that didn't start up correctly as there were incorrect configurations preventing nodes from starting up. The lack of proper debugging messages in the software stack led to this type of issue taking a lot of time.
- **URDF issues:** At the start of the project, no URDF was fitting the Polybot that the project worked with, so it was required to make one from scratch. This led to transformations between robot parts not being correct and the system not registering them correctly, and at times the robot model not being able to be displayed, thereby making simulation not possible to start.
- **PCB shorts:** The PCB's in the robots turned out to be a big cause of wasted time, as the original design for the custom PCBs was not made with soldering in mind. This mistake meant that the island that the pins on the board, which pins were supposed to be soldered to, made no contact with most of the boards. In the end, it required the PCB circuit traces to be scratched open to allow for new solder joints to be attached to the traces.

B. MoSCoW Analysis

A set of requirements are outlined in Table I, in which they are prioritised as either *Must Have*, *Should Have*, *Could Have*, or *Will Not Have*. The system described in section III is capable of exploration and patrolling tasks, which are tied together with a path creation pipeline.

	Requirement	M	S	C	W
R1	A user can specify a route for the robots to follow	X			
R2	A user can start an exploration task	X			
R3	A user can see the status of a task		X		
R4	A user can pause a task			X	
R5	A user can resume a task			X	
R6	The system is secure				X
R7	The pipeline can create a closed-path from an explored map	X			
R8	The AMRs can navigate around obstacles automatically	X			
R9	The AMRs can map their environment during exploration	X			
R10	The ARMscan update a map of their environment dynamically	X			
R11	The system supports exploration tasks	X			
R12	The system supports patrolling tasks	X			
R13	The system can perform multi-AMR exploration			X	
R14	The system can perform multi-AMR patrolling		X		
R15	The system captures comparable data from experiments	X			
R16	The system can run experiments unsupervised		X		
R17	The system can run multiple experiments automatically			X	

TABLE I: MoSCoW analysis of requirements.

Requirements **R1**, **R2**, **R3**, **R4**, and **R5** describe what a user can do with the system. They can initiate both exploration and patrolling tasks, as well as see the status of either task.

Requirement **R7** describes the capabilities of the pipeline. It should be able to take an explored map as input from an exploration task and produce a set of closed-paths for a patrolling task to use.

Requirement **R8**, **R9**, and **R10** describe the capabilities of the *AMRs*. They should be able to explore an environment, mapping while it does so. It should be able to update its environment on subsequent readings, and navigate around obstacles.

Requirements **R6**, **R11**, **R12**, **R13**, **R14**, **R15**, **R16**, and **R17** describe general capabilities of the system as a whole. The system should support both exploration and patrolling tasks. Ideally, it also allows for multi-AMR exploration and patrolling. The system can also be able to capture data from experiments, which can be scheduled and run unsupervised. Finally, the system is secure, making it fitting for tasks such as security patrolling and surveillance.

C. *Task Management Framework*

The project work was managed in the collaborative workspace tool *Notion* [49]. It uses a database-system which lets the user show the same data in multiple views. This means that the group could create a list of tasks, and these would automatically be converted to the other views. Examples of this can be seen in Figure 19, Figure 18, and Figure 17. They show a *Task Table*, *Kanban Board*, and *Timeline/Gantt Chart* respectively. The group would create a task in the *Task Table* view, where they would specify a time period, any parent or subtasks, as well as a status. The task will then be converted to a time-slot on the *Timeline* view which functioned as the group's calendar, as well as a card on the *Kanban Board* view. This allowed the group to keep track of the status of running tasks, as well as the project overall.

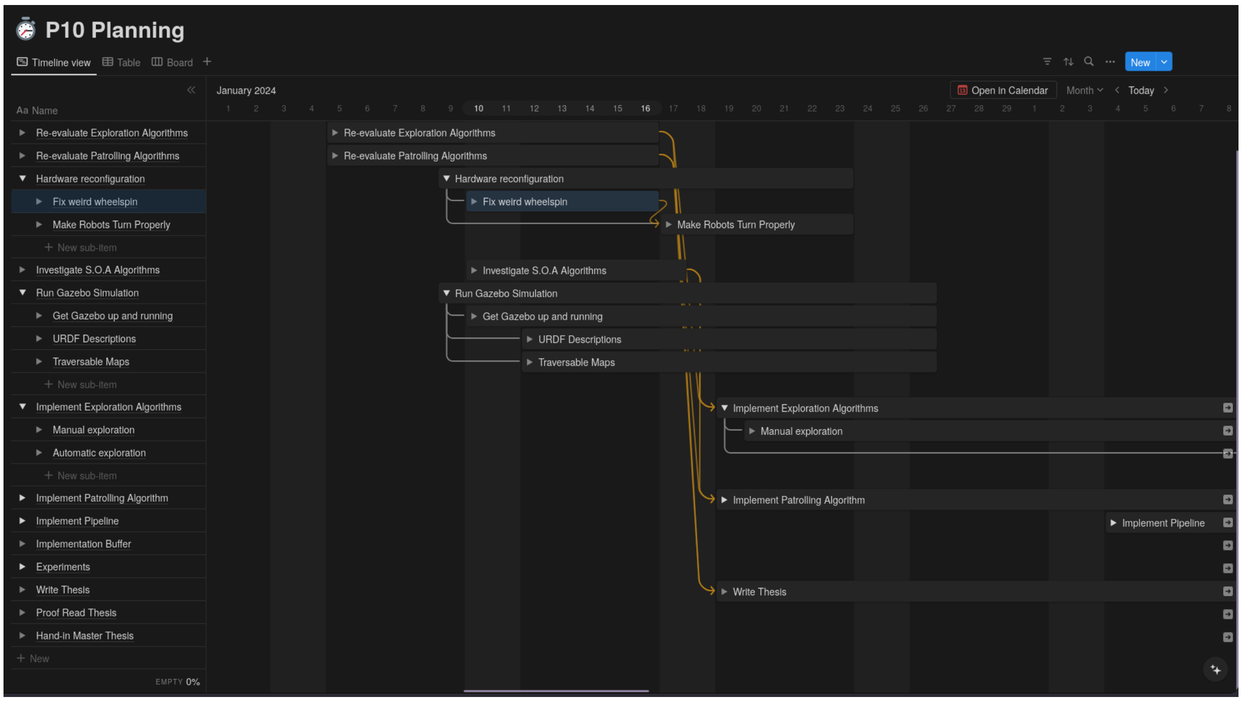
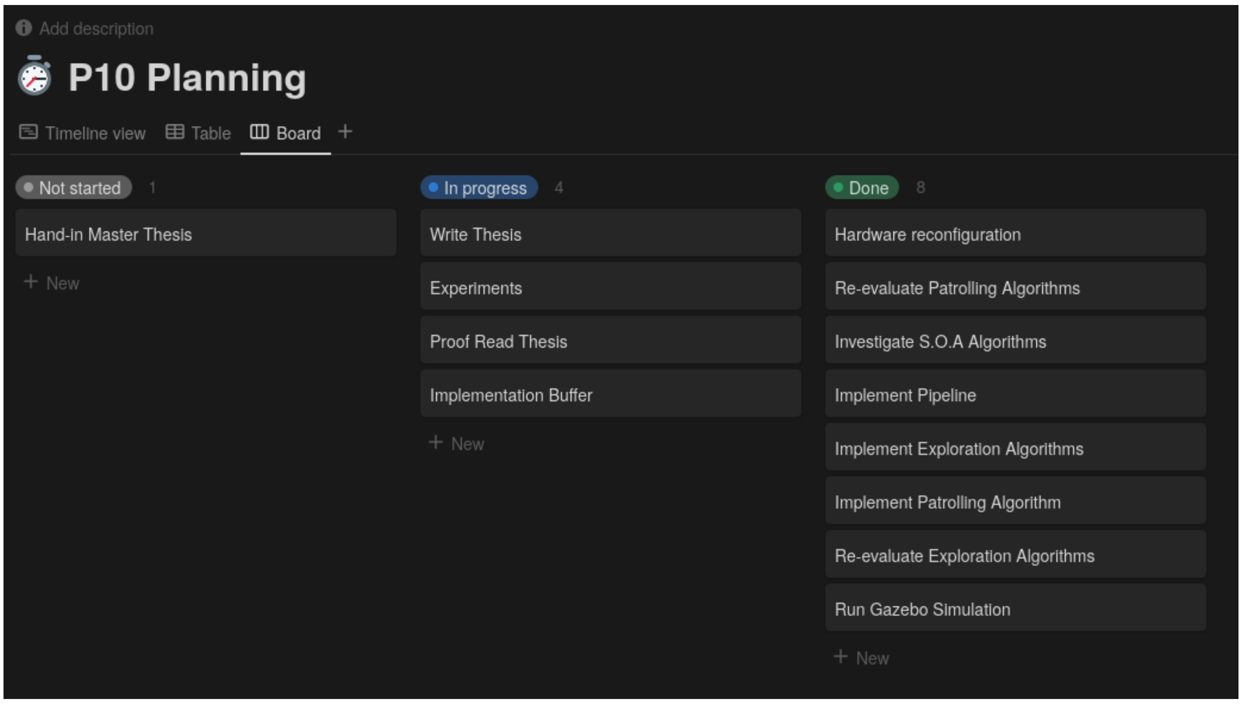


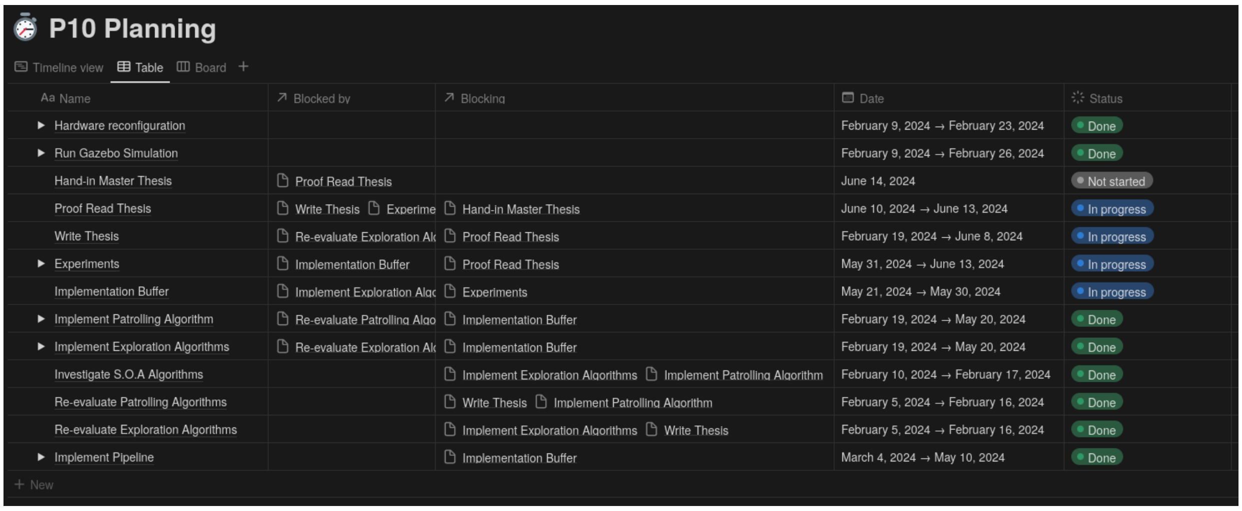
Fig. 17: Notion Gantt Timeline.



P10 Planning

Not started 1	In progress 4	Done 8
Hand-in Master Thesis	Write Thesis	Hardware reconfiguration
	Experiments	Re-evaluate Patrolling Algorithms
	Proof Read Thesis	Investigate S.O.A Algorithms
	Implementation Buffer	Implement Pipeline
		Implement Exploration Algorithms
		Implement Patrolling Algorithm
		Re-evaluate Exploration Algorithms
		Run Gazebo Simulation

Fig. 18: Notion Kanban Board.



P10 Planning

Name	Blocked by	Blocking	Date	Status
Hardware reconfiguration			February 9, 2024 → February 23, 2024	Done
Run Gazebo Simulation			February 9, 2024 → February 26, 2024	Done
Hand-in Master Thesis	Proof Read Thesis		June 14, 2024	Not started
Proof Read Thesis	Write Thesis, Experime	Hand-in Master Thesis	June 10, 2024 → June 13, 2024	In progress
Write Thesis	Re-evaluate Exploration Al	Proof Read Thesis	February 19, 2024 → June 8, 2024	In progress
Experiments	Implementation Buffer	Proof Read Thesis	May 31, 2024 → June 13, 2024	In progress
Implementation Buffer	Implement Exploration Algo	Experiments	May 21, 2024 → May 30, 2024	In progress
Implement Patrolling Algorithm	Re-evaluate Patrolling Algo	Implementation Buffer	February 19, 2024 → May 20, 2024	Done
Implement Exploration Algorithms	Re-evaluate Exploration Al	Implementation Buffer	February 19, 2024 → May 20, 2024	Done
Investigate S.O.A Algorithms		Implement Exploration Algorithms, Implement Patrolling Algorithm	February 10, 2024 → February 17, 2024	Done
Re-evaluate Patrolling Algorithms		Write Thesis, Implement Patrolling Algorithm	February 5, 2024 → February 16, 2024	Done
Re-evaluate Exploration Algorithms		Implement Exploration Algorithms, Write Thesis	February 5, 2024 → February 16, 2024	Done
Implement Pipeline		Implementation Buffer	March 4, 2024 → May 10, 2024	Done

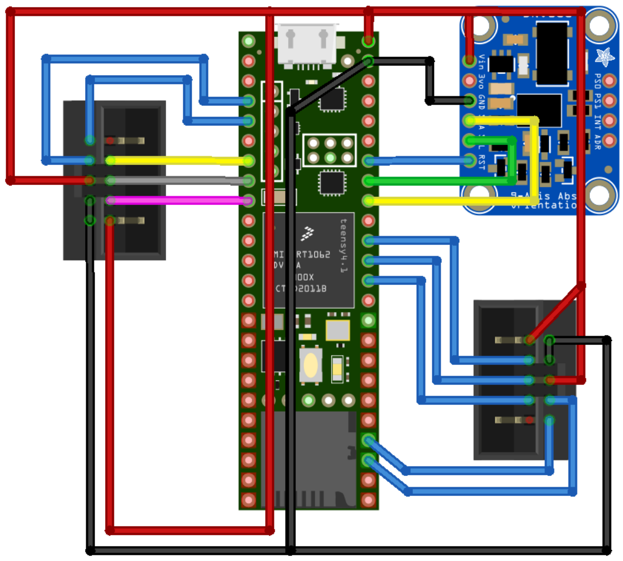
Fig. 19: Notion Task Table.

APPENDIX D
AMR PARTS LIST & WIRING DIAGRAMS

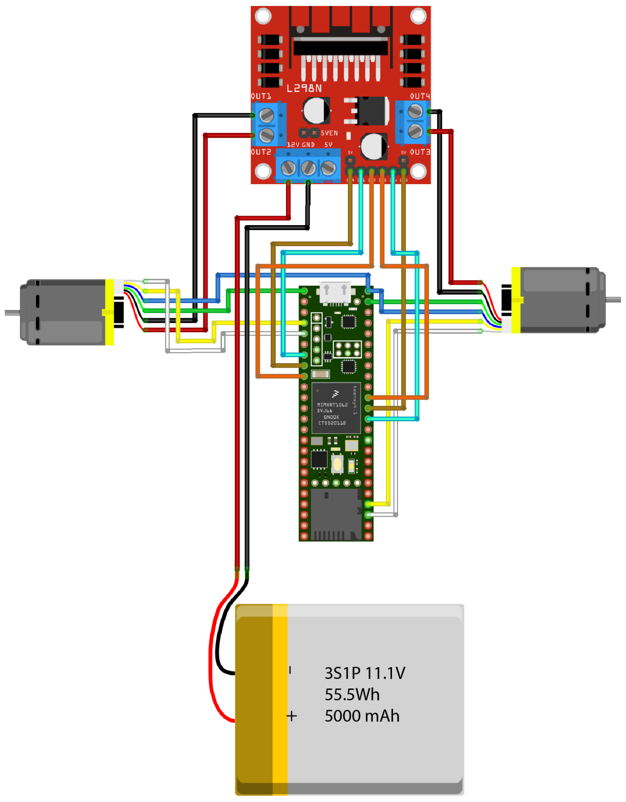
The following is a parts list for the *AMRs*:

- 1× *Raspberry Pi 4 8Gb*
- 1× *Teensy 4.1* microcontroller with custom breakout PCB
- 1× *lm2596* voltage step-down converter with display
- 1× *Adafruit BNO055* absolute orientation sensor
- 1× *RPLIDAR A1* LiDAR sensor
- 1× 5000 mAh 11,1V LiPo *3SIP* battery
- 1× *L298N* dual h-bridge motor controller
- 2× *Pololu* geared DC motor with encoders

A wiring diagram for the *Teensy* PCB can be seen in Figure 20a and a wiring diagram for the *AMRs* can be seen in Figure 20b. These figures are adapted from Appendix A and B of this project group's 9th semester project [20].



(a) Diagram of the Custom PCB breakout board for the *Teensy* microcontroller.



(b) Diagram of the wiring for the Robots.

Fig. 20: *Teensy* PCB diagram & robot wiring diagram.

Appendix E
Full-size Images of Architecture and Pipeline

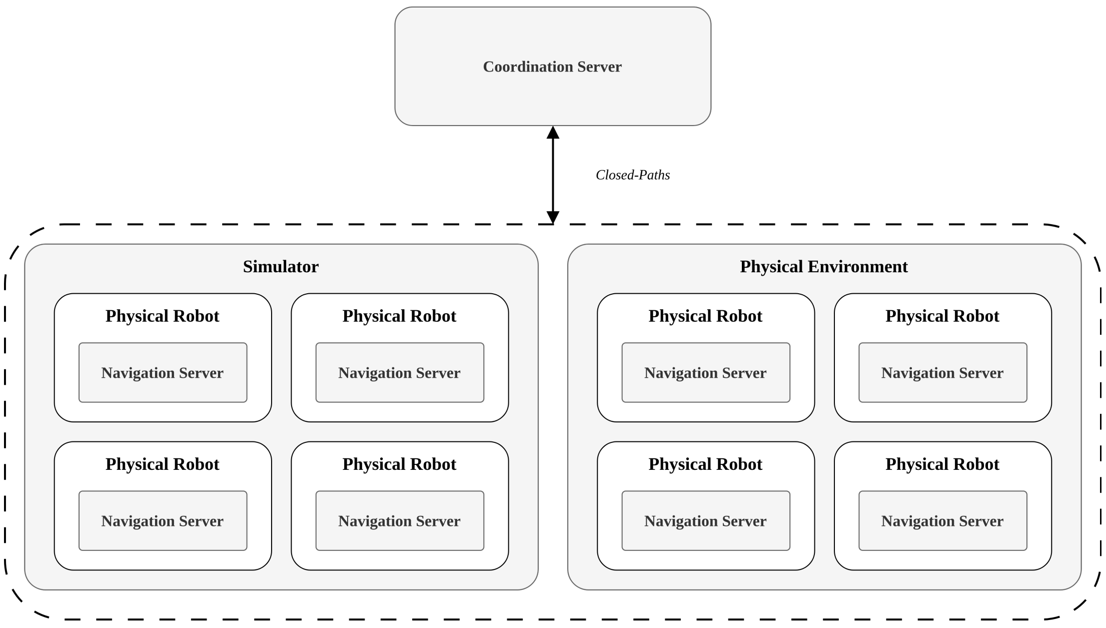


Fig. 21: Autonomous Reference Architecture for the *AMR* system.

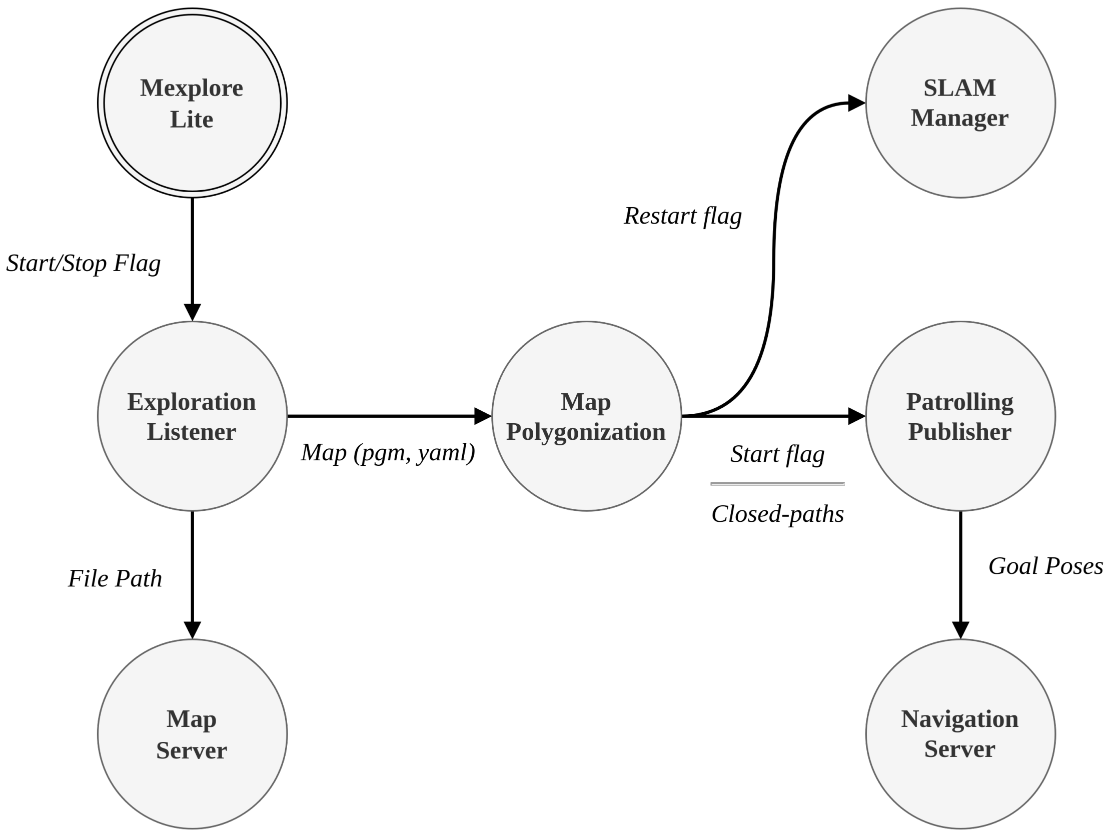


Fig. 22: Pipeline communication diagram.

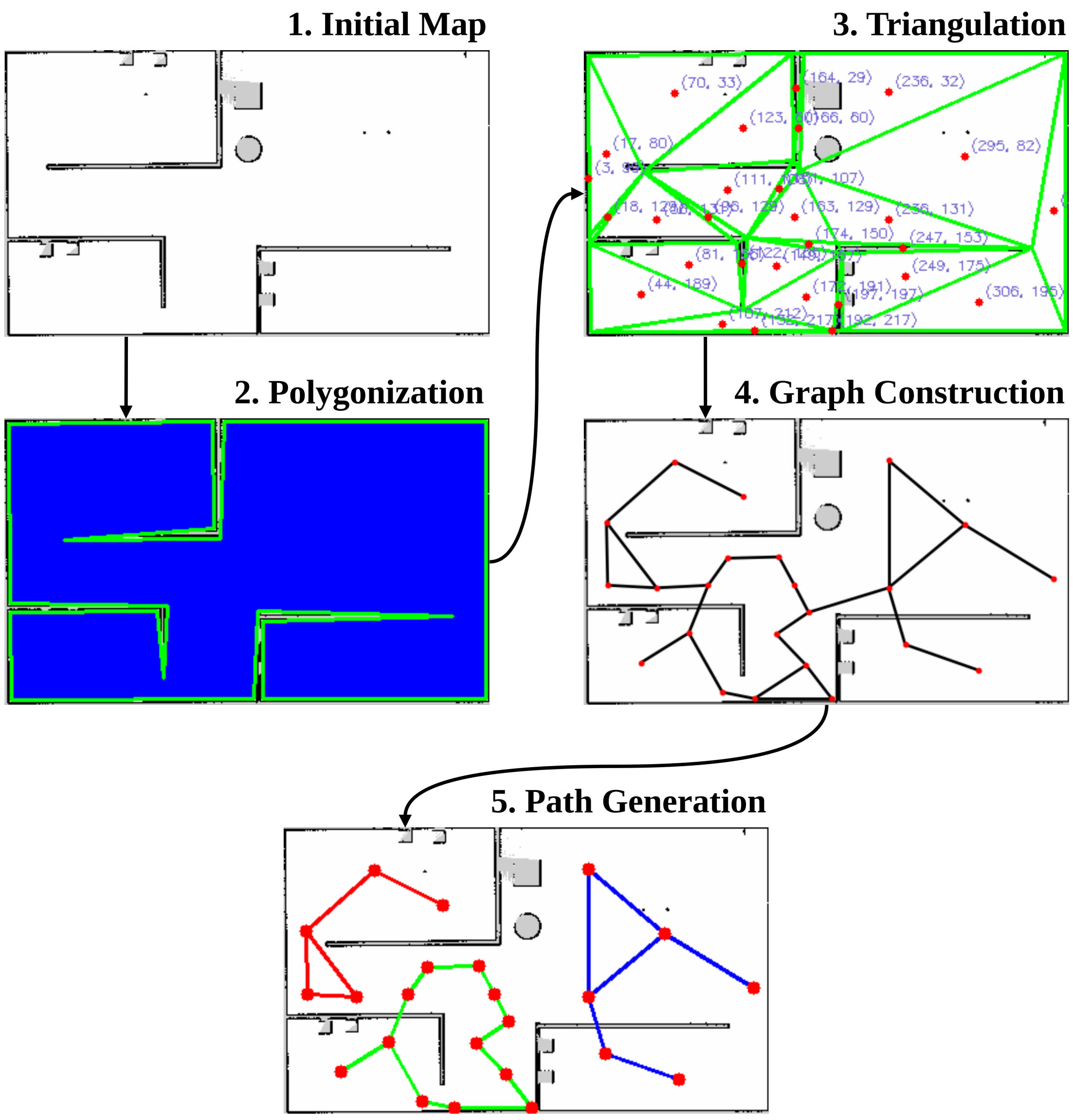


Fig. 23: Pipeline for generating closed-paths on map.

Appendix F
Simulation Environments Full-Size Images

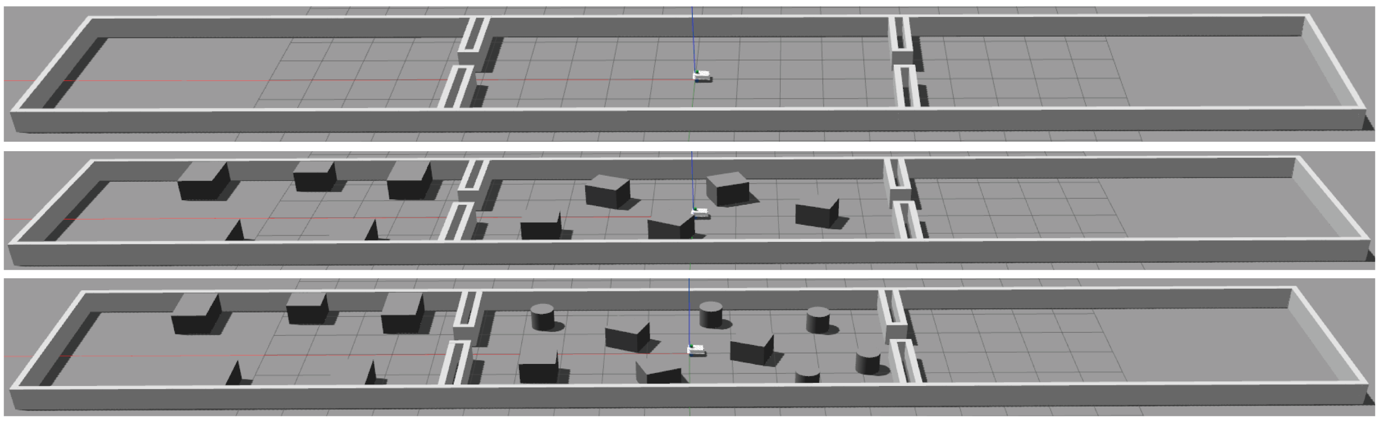


Fig. 24: Full-size simulation environments, a room with no obstacles, some obstacles, and many obstacles.

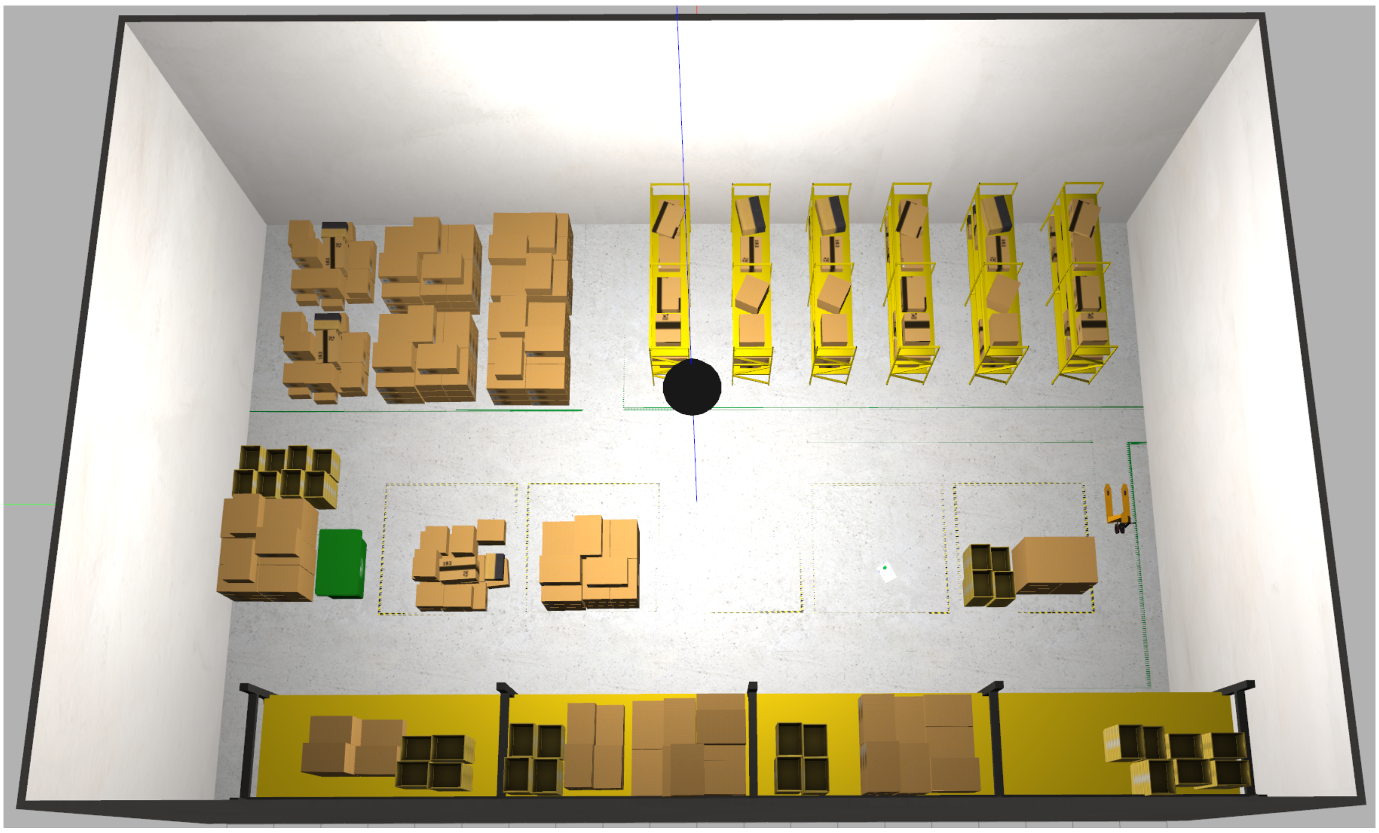


Fig. 25: Full-size simulation small warehouse environment.

APPENDIX G
ALL DATA PLOTS

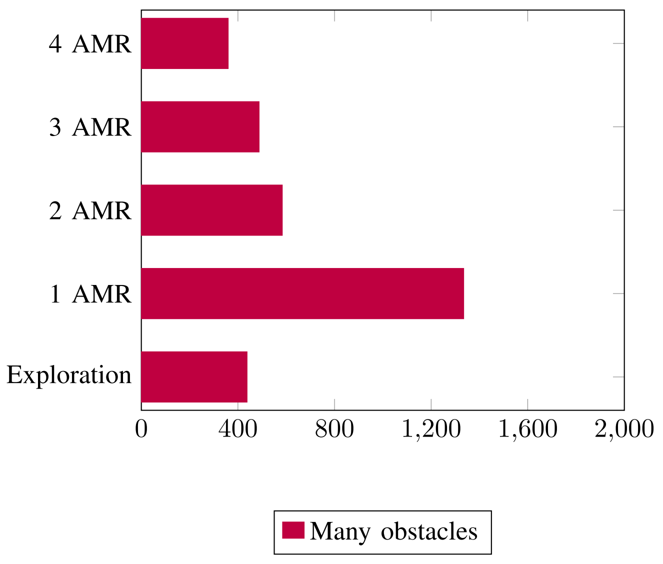


Fig. 26: Time to run exploration vs patrolling averaged over 10 executions.

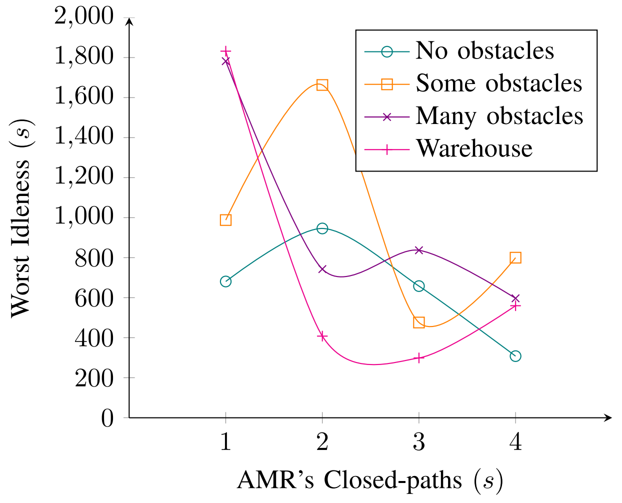


Fig. 28: Worst Idleness over number of closed-paths averaged over 10 runs.

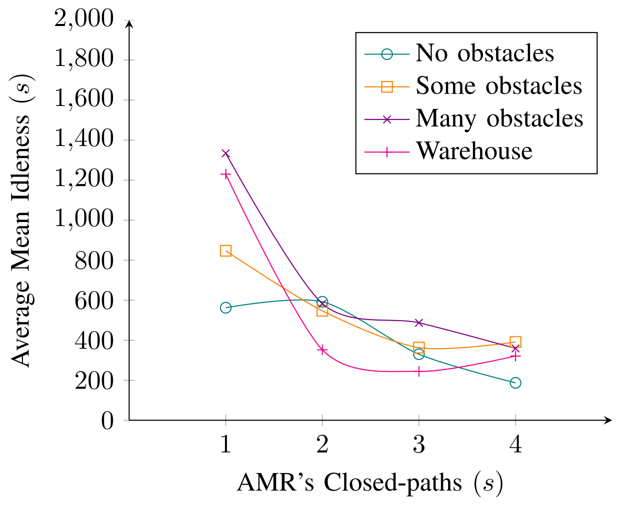


Fig. 29: Average Idleness over number of closed-paths averaged over 10 runs.

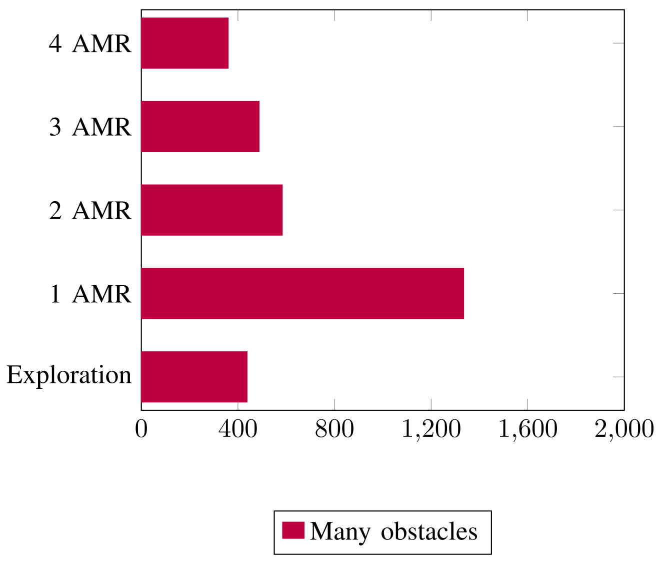


Fig. 27: Time to run exploration vs patrolling for 10 iterations.

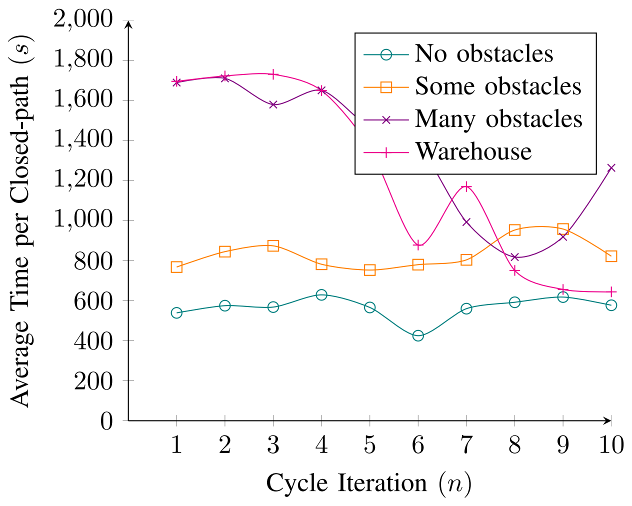


Fig. 30: Time over cycle iteration with 1 AMR.

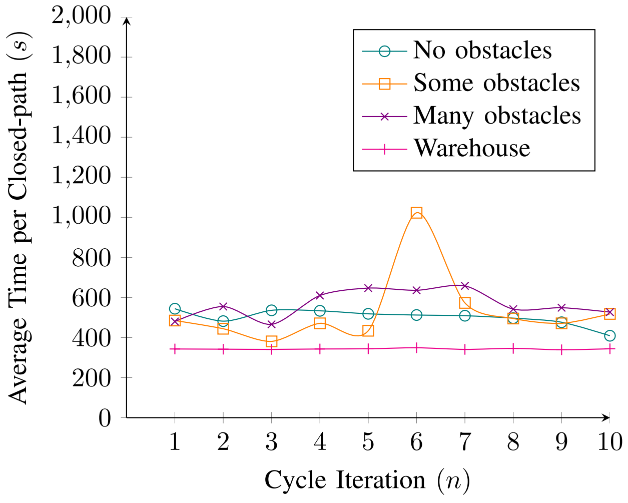


Fig. 31: Time over cycle iteration with 2 AMRs.

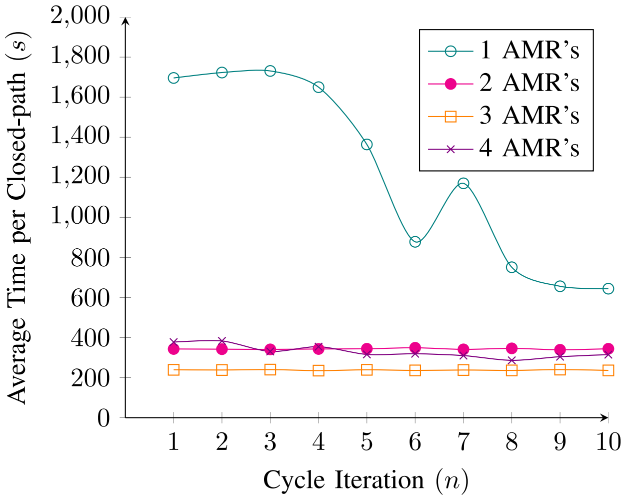


Fig. 34: Time over cycle iteration in Warehouse map.

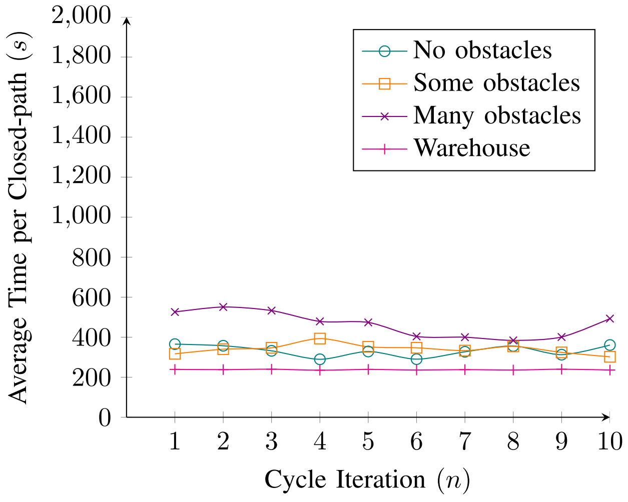


Fig. 32: Time over cycle iteration with 3 AMRs.

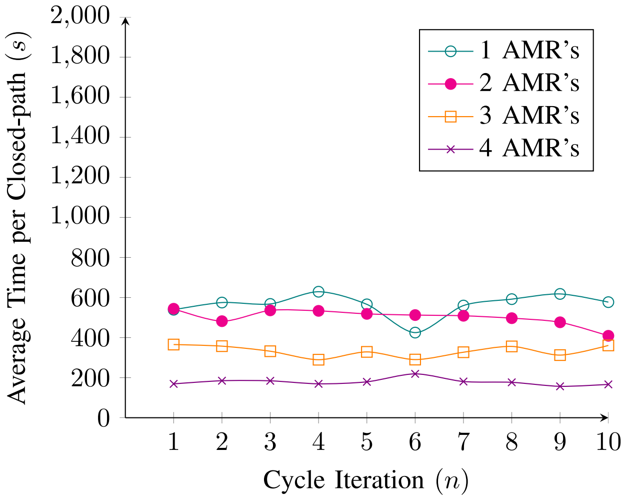


Fig. 35: Time over cycle iteration in the No Obstacles map.

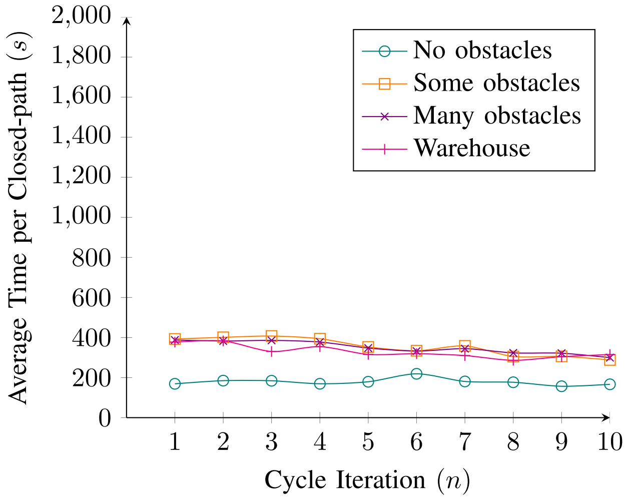


Fig. 33: Time over cycle iteration with 4 AMRs.

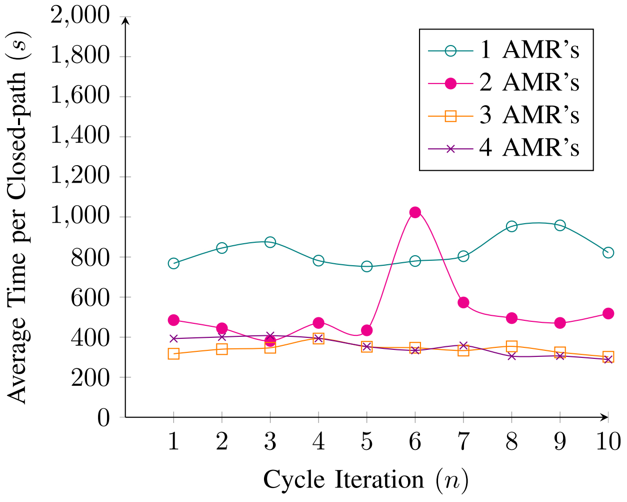


Fig. 36: Time over cycle iteration in the Some Obstacles map.

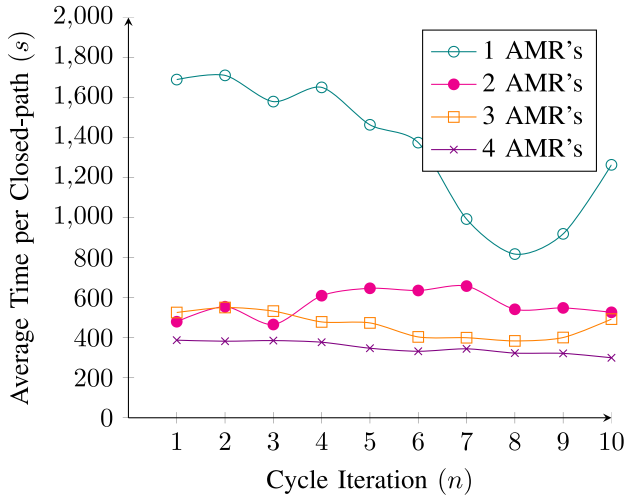


Fig. 37: Time over cycle iteration in the Many Obstacles map.

Appendix H
Physical Robot Experimentation

Execution of physical robot experiments was attempted, however low friction between the robot's wheels and the floor caused the robot to slip a lot, which brought it to a standstill. Although the robots were unable to explore, data regarding its initial surroundings was transmitted and the space around the robot was mapped correctly. This confirms that the pipeline also works on the physical robots. In Figure 38 the robot with original wheels didn't provide enough traction on the floor. Additional efforts were made to try to improve the traction by adding weight to the robot and new 3D printed wheels as seen on Figure 39

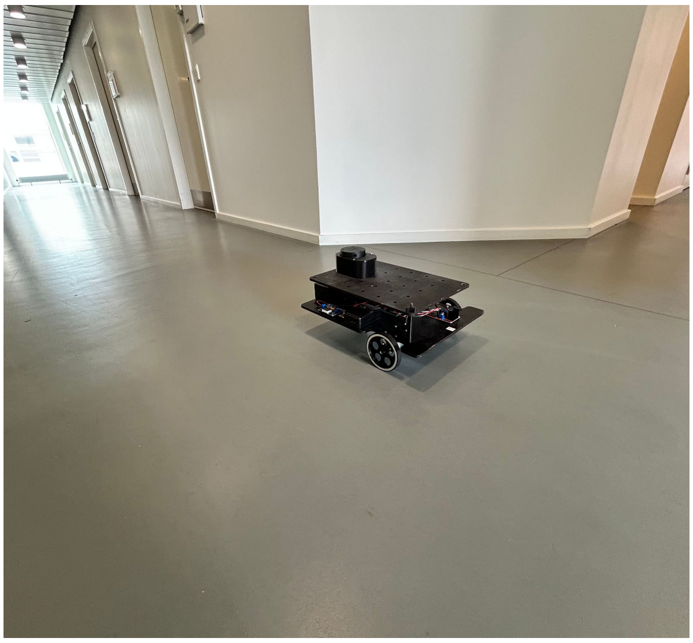


Fig. 38: Physical AMR with original wheels providing slippage

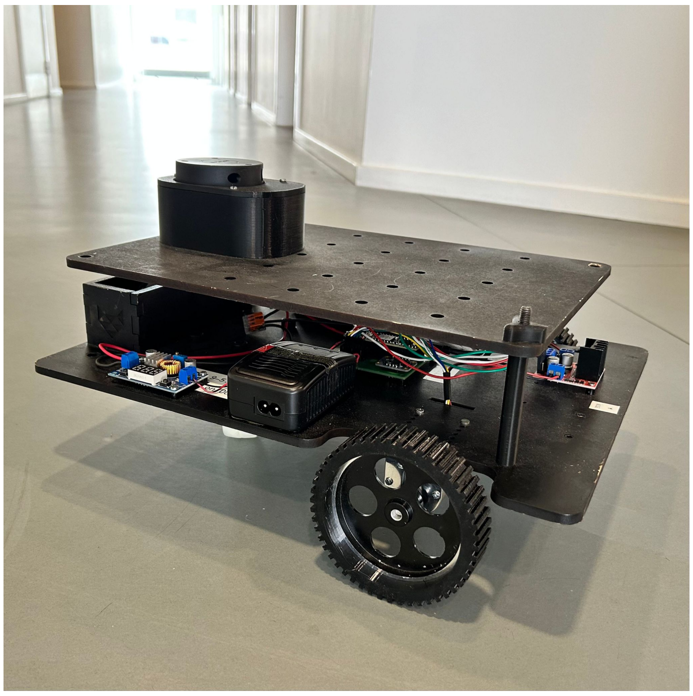


Fig. 39: Robot with new wheels