Informatics

Multi-Roboter Routenplaner mit MStar für ROS2

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Daniel Harringer, BSc.
Matrikelnummer 11775835

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Ass. Dipl.-Ing. Dr.techn. Markus Bader

Wien, 26. April 2024

Signed by:	Daniel Harringer
Date:	26.04.2024 17:10:41

This document is digitally signed!
This document bearing a qualified electronic signature has the same legal standing as a handwritten document in accordance with Article 25 (2) of Regulation (EU) No. 910/2014 of 23 July 2014 ("eIDAS-VO")
Validation: Information for checking the electronic signature can be found at: www.a-trust.at/pdf

Daniel Harringer

Signed by:	Markus Bader
Date:	26.04.2024 17:22:12

This document is digitally signed!
This document bearing a qualified electronic signature has the same legal standing as a handwritten document in accordance with Article 25 (2) of Regulation (EU) No. 910/2014 of 23 July 2014 ("eIDAS-VO")
Validation: Information for checking the electronic signature can be found at: www.a-trust.at/pdf

Markus Bader

Informatics

Multi-Robot Route Planner with MStar for ROS2

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Daniel Harringer, BSc.
Registration Number 11775835

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Ass. Dipl.-Ing. Dr.techn. Markus Bader

Signed by:	Daniel Harringer
Date:	26.04.2024 17:13:39

This document is digitally signed!
This document bearing a qualified electronic signature has the same legal standing as a handwritten document in accordance with Article 25 (2) of Regulation (EU) No. 910/2014 of 23 July 2014 ("eIDAS-VO")
Validation: Information for checking the electronic signature can be found at: www.a-trust.at/pdf

Signed by:	Markus Bader
Date:	26.04.2024 17:25:19

This document is digitally signed!
This document bearing a qualified electronic signature has the same legal standing as a handwritten document in accordance with Article 25 (2) of Regulation (EU) No. 910/2014 of 23 July 2014 ("eIDAS-VO")
Validation: Information for checking the electronic signature can be found at: www.a-trust.at/pdf

Vienna, 26th April, 2024

Daniel Harringer   Markus Bader

Erklärung zur Verfassung der Arbeit

Daniel Harringer, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Signed by:	Daniel Harringer
Date:	26.04.2024 17:15:33

This document is digitally signed!
This document bearing a qualified electronic signature has the same legal standing as a handwritten document in accordance with Article 25 (2) of Regulation (EU) No. 910/2014 of 23 July 2014 ("eIDAS-VO")

Validation: Information for checking the electronic signature can be found at: www.a-trust.at/pdf

Wien, 26. April 2024

Daniel Harringer

Acknowledgements

First, I would like to thank my parents, who have supported me actively and financially during my studies. In addition, I would like to thank them for always motivating me to finish my studies.

Special thanks go to Markus Bader for his guidance, support, and contributions throughout this thesis. I would also like to thank him for his patience during the stagnation phase and support over the last mile.

Abstract

Autonomous or semi-autonomous machines with the capability to move freely in their environment without humane guidance are called mobile robots. These robots often have various sensors and actors to interact with their surrounding environment and localise themselves. When aggregating multiple mobile robots in one system or environment, one speaks of a multi-robot system or robot fleet. Robots in a system work together to achieve a common goal, which has been broken down into several sub-tasks. Common use cases for robot fleets are warehouse, surveillance or search and rescue applications. In all these applications, the robots need to move from a starting position to a goal location without colliding with obstacles in the environment or other robots of the fleet. Different algorithms have been introduced to solve the path planning problem for multi-robot systems. This thesis aims to integrate the optimal and complete path planning algorithm M* into an existing framework and evaluate its capability to be used in real-world scenarios. Furthermore, assessing the quality of a plan computed by a multi-robot path planner is part of this thesis as well. In order to achieve this goal, multiple scenarios and environments are simulated with an increasing number of agents in the system. The results and findings are presented within this thesis.

Contents

CHAPTER 1

Introduction

A multi-robot system aggregates multiple moving agents that work closely together to achieve a joint goal. Depending on the application the robot fleet is used for, the goal can vary from moving objects from one location to another, guarding a high-security area, or exploring the environment to find a leak in a gas pipeline. All these applications have a common fundamental problem: they need to handle a high dimensional configuration space in order to compute a collision- and deadlock-free path for each of the agents in the system [1].

Path planning algorithms address this problem with two different approaches, coupled and decoupled. A coupled path planning algorithm searches the whole configuration space, which results in a high computational effort, but the generated paths are optimal solutions to the problem [2]. Nevertheless, it has been proven that computing optimal paths for multi-robot systems is NP-Hard [3]. Decoupled path planning algorithms are searching a low dimensional configuration space representing a portion of the full. Therefore, the computational effort is low, but solutions found by such algorithms are not guaranteed to be optimal or complete. Many researchers tried to combine these approaches to create an algorithm capable of finding an optimal solution within a certain period. One of the results is called M* and presented in this paper [1].

1.1 Showcases

Environments in which multi-robot fleets mostly operate can be divided into two parts. First, there are wider passages where agents can pass each other, for example, at hallway crossings. Second, narrow hallways allow only one robot at a time to move along, for instance, between industrial racks. These showcases bring certain challenges to the path planning problem. For example, the path planning algorithm needs to ensure efficient coordination of all robots in the fleet to prevent agents from blocking each other or deadlocking the whole system and finding a short path between each robot's start and the goal position. Due to being a NP-Hard problem, many different approaches of path planning algorithms are presented by research, such as Conflict-based Search (CBS) [2], [4] or the M* [1].

This thesis integrates both approaches in a multi-robot simulation framework [4] and tests their capabilities for applications in the mentioned showcases. The simulation framework is based on the Robot Operating System (ROS)[5]; it includes pre-built components to visualise data, such as the showcases and robots' actual position and communication infrastructure. The simulation framework consists of a self-localisation and a local behaviour controller for each robot. Self-localisation estimates the robots' position in the environment and the local behaviour controller steers the robot through it based on the plan's information and the position estimation. Additionally, there is a graph generator to compute a graph representation of the environment and feed it to the path planning algorithm.

A Multi-Robot Route Planner (MRRP) [4] based on a CBS approach and the M* planner are compared during this thesis. For this purpose, six scenarios have been created, which often have to be solved in real-world applications. These showcases are waiting for another robot to pass, avoiding another robot at a crossing, avoiding other robots at the start or goal position, and pushing robots blocking hallways to let other robots pass. Furthermore, the performance of both planners is tested in warehouse environments with different robot fleet sizes, such as 4, 8, 16, 32, and 64. For a detailed description, see Chapter 3.

1.2 Uncertainties in planning

A fundamental challenge of multi-robot systems is to compute a plan that navigates each agent to its goal location [6]. In Multi-Agent Path Planning (MAPF), all robots are guaranteed to reach their destination, but the main drawback is that they must precisely follow the generated plan. Due to simplifications when modelling the dynamics of a robot and errors in localisation and sensing, the computed plan cannot be executed exactly. Therefore, the need for robust plans against uncertainties in timing and localisation arises [7]. Researchers divided path planning algorithms into three categories using symbolic model checking techniques. Algorithms that have a chance of successfully finding a solution (weak planning), those that are guaranteed to compute a valid solution (strong planning), and such that achieve the goal by applying trial-and-error strategies to compute valid paths (strong cyclic planning) [8].

1.3 Research Questions

To address the challenge of uncertainties in plan execution and finding quality parameters for computed plans, the first research question, **RQ1**, has been formulated. Furthermore, **RQ2** and **RQ3** address the challenges of navigating multi-robot systems efficiently and safely through warehouse environments.

RQ1 aims to qualify a solution for multi-robot path planners, and this is answered using the computed paths for the warehouse environment.

RQ1: What are quality measurements of multi-robot planner results?

Generally, most path planners optimise execution time in the first place. Other optimisation parameters, such as energy consumption, are considered for special applications. However, besides execution time, route stability has been identified as a well-suited quality parameter for any robot application. It indicates how robust a computed plan is against timing uncertainties in the execution of the plan.

The second research question, **RQ2**, aims to evaluate the performance of both planners in the mentioned scenarios.

RQ2: How does the M* perform in contrast to the currently implemented conflict-based search in the domain of multi-robot applications?

The simulation results show that both the M* algorithm and the MRRP compute valid solutions for the above described scenarios. In general, the MRRP finishes the computation of a plan in a shorter time. In certain scenarios, it computes a plan six times faster than the M* algorithm. Additionally, the results show a higher memory occupation of M* during the planning phase.

The third formulated research question, **RQ3**, addresses the performance evaluation of the M* algorithm compared to the CBS in real-world applications.

RQ3: To what extent is the M* algorithm applicable to real-world multi-robot applications in terms of runtime and scalability?

The simulation results show that the M* algorithms' capabilities are limited for real-world scenarios with more than eight robots in a fleet. The success rate drops below 50% for any given path planning problem instance and the time to solution, which is the time needed to compute a valid plan between the start and goal positions, rises to up to half an hour. In contrast to that the MRRP can solve any given routing problem in around one second.

This thesis shows that the M* algorithm is suitable for multi-robot systems with up to eight robots. Still, solutions computed by this algorithm are generally more robust to uncertainties and do have a shorter execution time compared to solutions generated by the MRRP.

1.4 Structure of the Thesis

This thesis is structured as follows. Chapter 2 establishes a common knowledge base of the technology, frameworks, and multi-robot path planning algorithms. In Chapter 3, the problem is defined in a more detailed way, and the formulated research questions are presented. A description of how the components are implemented is given in Chapter 4. Furthermore, the experiments performed to answer these questions are described. Results are presented in Chapter 5 and discussed in Chapter 6. The last, Chapter 7, summarises the findings of this thesis and provides an outlook for future work.

CHAPTER 2

Background

Mobile robots are autonomous or semi-autonomous machines with the capability to move freely in their environment without human interaction or guidance. They are equipped with various sensors to map their surrounding environment and localise themselves in it. A multi-robot system is a set of two or more robots, often called a robot fleet. This fleet works together collaboratively in order to achieve a common goal. In the case of mobile robots, a common goal might be moving objects from one location to another, for example, in warehouses. Surveillance applications are another example of applications for multiple robots. This time, the common goal is to monitor the environment to guard a certain building or detect damage to a pipeline. These applications require a path between a start and a goal location, which the robots follow a so-called planner computes an individual, collision-free route for each robot in the fleet. Collision-free means that no robots collide with obstacles in the environment or with another robot of the fleet. Furthermore, the path planner should be aware of possible deadlocks and resolve them during planning. A deadlock can occur if two robots meet in a small hallway and both need to reach the other side, but there is no space to move aside and let the other robot pass. In order to avoid such scenarios, path planning algorithms for multi-robot systems have to take care of potential threats during the planning phase. Computing optimal solutions for multi-robot systems is proven to be NP-Hard [3]. One developed approach that encounters the problem of optimal path planning is Subdimensional expansion for multi-robot path planning [1], which is described in detail later in this chapter.

2.1 Paper Research

In preparation for this thesis, the supervisor recommended the following literature [4], [1], [9] and [2]. Furthermore, the supervisor advised the courses of a collaborative research group [10]. Based on these recommendations, paper research has been conducted using the snowball strategy and common research platforms such as IEEE Xplore and

Arxiv. Starting from the original M*, a search for variants and modifications has been performed. The search has been limited to the last ten years. It uses the following keywords in different combinations: subdimensional expansion, M*, multi-robot system, path planning algorithm, multi-robot path planning algorithm, optimisation and optimal path planner. Resulting papers are [11], [12], [7] and [13].

Literature research has been conducted starting from the paper about timing uncertainties in multi-robot systems [9] and with the following path planning, multi-robot system, (Timing) uncertainty, (time) independence, route stability, safe routes, online/offline replanning and route safety. By limiting the search to the past five years and using the snowball strategy again, the following literature has been found [7], Characterise solutions of planner [8], [6] and [14].

2.2 Robot Operating System (ROS)

The ROS is a set of open-source software libraries and tools that help researchers, professionals, and hobbyists build robot applications. It provides powerful developer tools, drivers for commonly used hardware, and state-of-the-art algorithms out of the box [5].

The basic concept of ROS is formed by Nodes, which form computational units and Topics, Services, Actions and Messages for commutation between these computation units.

A graphical representation of the communication patterns can be seen in Figure 2.1. Additionally, to the different communication patterns, also a command line tool is shown. ROS offers a wide variety of command line tools to debug and develop applications effectively and monitor communication between nodes.

A Node is a lightweight and modular unit, that represents and controls different aspects of the robot's functionality, such as sensors, actors, or navigation algorithms. ROS nodes communicate with each other by publishing messages on predefined topics. Topics are part of an asynchronous message-passing framework to distribute data over it. A node can subscribe to a topic to receive published data. If a node intends to share its data, it will publish it on a predefined topic. The communication framework, which is based on the publish-subscribe pattern, allows one-to-many as well as many-to-one data distribution [5].

ROS additionally provides a request-response communication pattern known as Services. A service is a named function that a node can offer or call. This communication pattern provides an easy data association between a request and a response, which can be useful when it is needed to ensure a task is received by another node or is completed. ROS2 allows a service client process to be non-blocking while calling its service server. Examples of tasks that benefit from the service pattern are parameter settings or performing a calibration [15].

A special communication pattern introduced in ROS2 are Actions. An action is a goal-oriented and asynchronous communication interface for long-running tasks with feedback

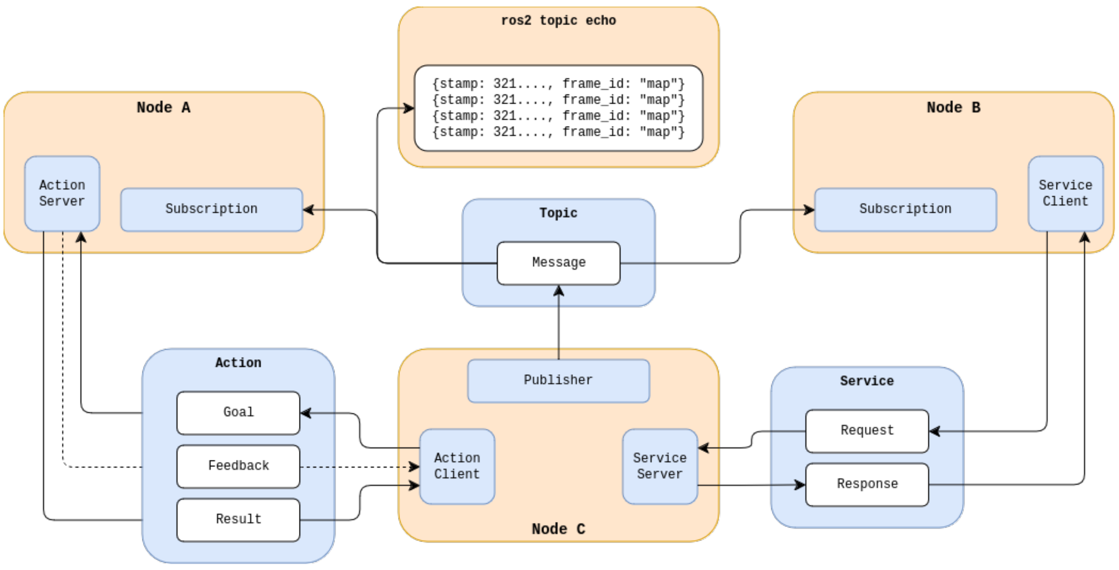


Figure 2.1: ROS2 communication pattern, based and adapted from [15]

and the ability to be cancelled. An Action extends a service. They are using a client-server model and additionally provide continuous feedback. The server provides feedback by publishing on a predefined topic, which allows the client to constantly monitor the state of the computation and make decisions based on this information. Since they are based on services, actions are non-blocking and organised under the node. Examples of long-running tasks include navigation, path planning, and other complex robot actions.

Since the newest major version, ROS2, the communication framework supports setting a Quality of Service (QoS) policy for each communication endpoint. These policies reach from as reliable as Transmission Control Protocol (TCP) to as best-effort as User Datagram Protocol (UDP), with many levels in between to ensure the best-fitting strategy for each network and application. Combining a set of QoS policies results in a QoS profile. These profiles can be specified for publishers, subscribers, service servers, and clients. A QoS profile can be applied independently to each instance of the mentioned entities. However, using different profiles, it is possible that communication partners will be incompatible and the message will not be delivered. In order to prevent communication inconsistency ROS2 offers many predefined QoS profiles. These will be applied automatically for common use cases such as publishers and subscribers, services, sensor data, and parameter calibration [5].

2.3 Path Planning Algorithms

In logistics, for example, in warehouses, mobile robots are responsible for manipulating certain objects in their environment. In order to move those objects from a start to a goal position, the robot needs to follow a path through the environment. The environment contains static and sometimes dynamic objects; static objects such as walls, industrial racks, and manufacturing machines are non-movable. Dynamic objects, including other mobile robots, humans, and packages, can move freely in the environment.

A path planning algorithm is responsible for finding a collision-free path between these two locations. In the case of multi-robot systems, the planner should also be aware of other robots' position in the environment during the planning phase.

2.3.1 Single robots

Path planning for single robot environments is discussed in detail in [2]. It also explains different types of map presentations and road maps for this problem. The most common single robot planning approaches, Dijkstra and A*, are also discussed extensively in [2]. In addition to the well-known A* algorithm, numerous variants are also used for path planning problems and were optimised for different criteria [16].

2.3.2 Multi-robot systems

In contrast to single-robot systems, two or more robots share the environment in a multi-robot system. Therefore, path planning algorithms for such systems have to ensure no collisions between robots during the execution of the plan. Another important aspect a path planning algorithm has to consider is the possible occurrences of deadlocks, either of multiple robots or the whole system.

In general, MAPF algorithms can be divided into two categories: coupled planning algorithms and decoupled ones.

Coupled algorithms consider all given robots in the environment as a single, interconnected system during the planning phase. This means that the paths for all robots are planned simultaneously, considering the interactions and constraints between each robot. Because coupled path planning considers the whole configuration space, which contains all possible paths of the system, the complexity grows exponentially with the number of robots in the system. Consequently, the computational complexity increases exponentially in the same manner. Because the whole configuration space is searched, coupled path planning approaches may be guaranteed to be collision-free and have minimal costs, which means they provide an optimal solution to the problem instance [17], [18].

Coupled path planning is a powerful approach when multiple robots need to navigate through the environment safely and efficiently as a part of a single system, such as transportation systems or collaborative manufacturing. However, planning paths come with computational challenges. It is a trade-off between precision and efficiency, especially with many robots.

Decoupled path planning algorithms first plan motions for each robot individually, ignoring interactions between them. Once these paths are planned, the algorithm considers these interactions and attempts to resolve conflicts. However, possible movements each robot can take are already constrained by the planned paths because decoupled planning approaches are typically unable to reverse their commitments from the first step. Therefore, their property of completeness is lost [2]. Since paths are planned independently, the searched configuration space is much smaller, and less computational power is needed to find a suitable solution. These configuration spaces are known as low dimensional search spaces and represent only a subset of the high dimensional ones [19], [20]. Additionally, in some cases, the algorithms can restore completeness when resolving collisions [2]. Nevertheless, it requires effective coordination and advanced collision avoidance mechanisms to ensure robots operate safely in the environment without deadlocking the whole system.

2.4 Subdimensional expansion for multi-robot path planning

The path planning algorithm used in this thesis is presented by the authors of this paper [1] and is called Subdimensional expansion for multi-robot path planning, also known as M*. The authors present Subdimensional expansion as an approach that shares the benefits of coupled and decoupled path planning approaches. The implementation of it, therefore, results in an algorithm called M*. The method manipulates the search space of existing search algorithms to decrease the computational cost of solving the problem. It starts by individually searching paths from the start to the goal configuration for each robot. Combining all computed paths generates a one-dimensional search space for the multi-robot system. If a robot-robot collision occurs in the combined configuration space, subdimensional expansion increases the dimension locally around the collision. This allows all robots involved in the collision to find alternative and collision-free paths around the collision region.

M* uses A* or a variant of it [16] as an underlying planner to compute the paths for the robot fleet, which is later used to guide the path planning for the multi-robot system. The main difference between M* and A* is the restriction of possible neighbours of a node by using a backpropagation set and a collision set.

Assume there exists a search graph G(V, E) that represents the environment in which the multi-robot system operates, agents can move along edges and pause their movements on vertices. The backpropagation set contains information about a collision and propagates this information back on all paths leading to this collision. That means the backpropagation set of a vertex v_k, which is a vertex of the search graph, holds all vertices for which v_k has been considered as a possible successor. Sub-Figure 2.2a shows the example graph with four nodes given to the path planning algorithm. The backpropagation set of a node is written below the node. Assume the algorithm considers the path A, C and D as the only possible way through the graph. Since vertex C is the successor of A, vertex A is added to the backpropagation set of C. It is assumed there is a collision on vertex D. The same applies to vertex D since it is a successor of A and C. Both nodes are added to the backpropagation set of D. Dashed lines in Sub-Figure 2.2b illustrate this backpropagation relation.

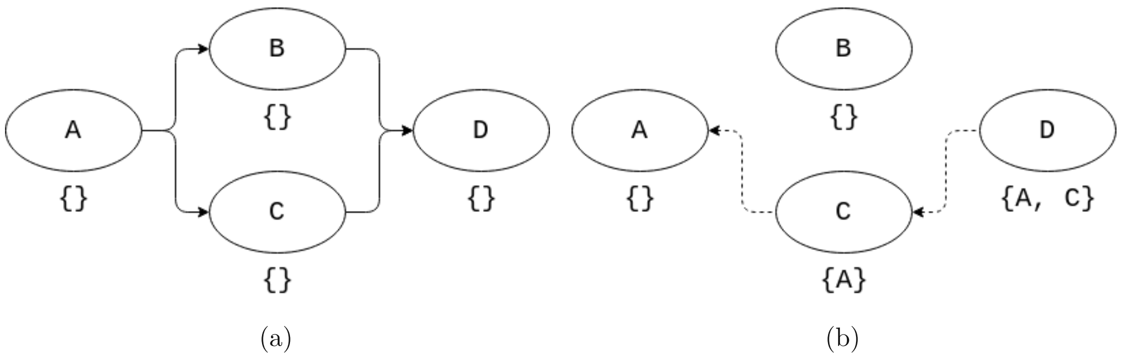


Figure 2.2: Example of the backpropagation set

The collision set of a vertex v_k in the graph of the search space contains a set of robots involved in a collision at this vertex. The collision can either happen at the vertex v_k or at some vertex v_l on a path that passes through the vertex v_k and has already been explored. In Sub-Figure 2.3a, an example graph is shown, which will be used for planning a path. The collision set is written below the node. Two robots are placed in the environment; robot r_1 is placed on vertex A, and robot r_2 is placed on vertex F. The goal node of robot r_1 is node E, and vertex D is the goal node of r_2. Since both vehicles are planned simultaneously, the algorithm will detect a collision on vertex D after the second step. Therefore, robot r_1 and r_2 are added to the collision set of each vertex along both paths as shown in Sub-Figure 2.3b.

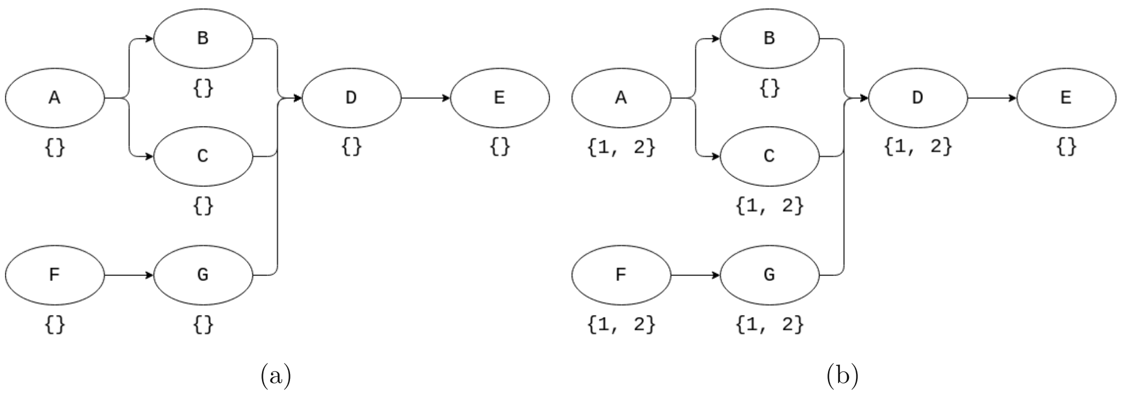


Figure 2.3: Example of the collision set

A working example of M*

In this section, the M* will be described based on a simple example configuration, as shown in Figure 2.4. Consider a team of three robots r_1, r_2, and r_3 operating on a four-connected grid, which represents a graph. The robots are moving towards their goals

g_1, g_2 and g_3. A combination of a letter and a number represents one coordinate in the graph A configuration is a triple of graph coordinates, always ordered robot coordinates by the robot indices. Following this notation, the start configuration can be written as (A3, D4, A1), and the goal configuration looks similar (C4, B4, D1).

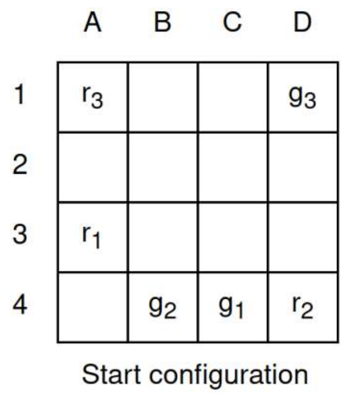
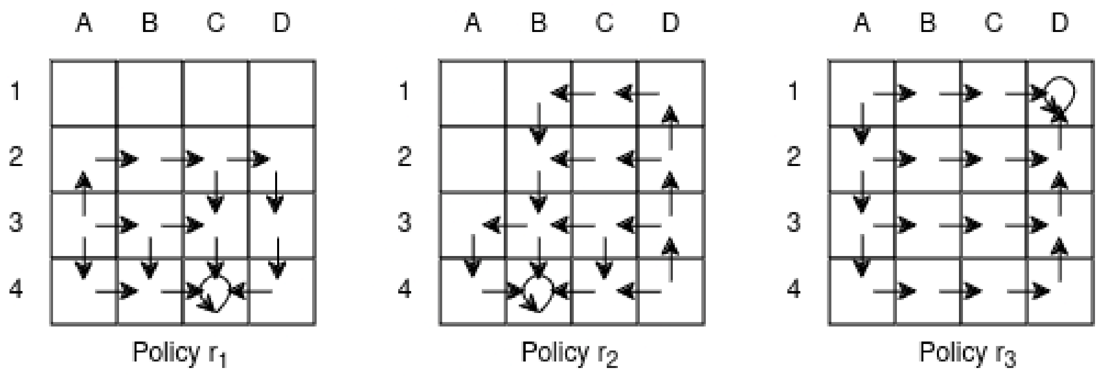


Figure 2.4: Start configuration of the multi-robot system

In the first step, the algorithm computes a policy for each robot individually. This policy does not consider any other robot in the system; therefore, any single robot path planning algorithm, like A* [16], can be used. Furthermore, this policy is not unique, so using another planning algorithm to construct the policy might lead to another outcome. The policy computed for this example can be seen in 2.5. Arrows in the grid indicate possible moves from one node to another. The policy does not consider this if there are no arrows from or to a grid tile.

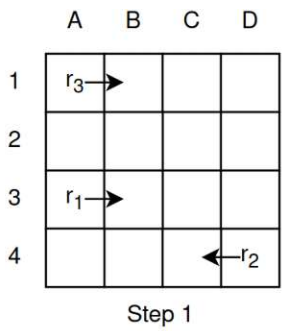


(a) Computed policy for r_1 (b) Computed policy for r_2 (c) Computed policy for r_3

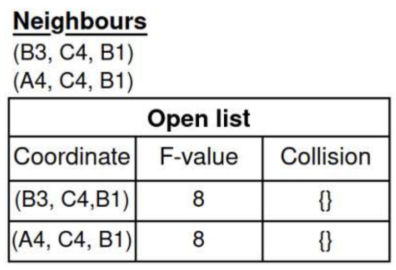
Figure 2.5: Computed policies for robots r_1, r_2 and r_3

After the policies are computed, M* starts searching the configuration space for conflicts between the individual policies. As A*, M* also maintains an open list of possible next nodes sorted by their f-value. The f-value of a node is the sum of costs to reach this node and a heuristic cost to go to the goal node. In the beginning, the open list contains the start configuration (A3, D4, A1), and the collision set is empty so that this node will be expanded. The Collision column stores the collisions at the actual or following nodes.

In this example, the start configuration has two possible neighbours with the same f-value, as shown in Sub-Figure 2.6b. Assume the robots are moving to the configuration (B3, C4, B1). Therefore, the graph, shown in Sub-Figure 2.6c, builds up. The solid line indicates the step taken by M*, and the dashed one shows the backpropagation path.



(a) Moves made by each robot in Step 1



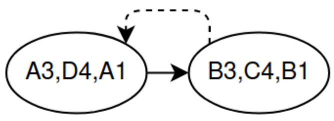
Neighbours

(B3, C4, B1)

(A4, C4, B1)

Open list		
Coordinate	F-value	Collision
(B3, C4,B1)	8	{}
(A4, C4, B1)	8	{}

(b) Neighbours and open list in Step 1



(c) Graph built after moving Step 1

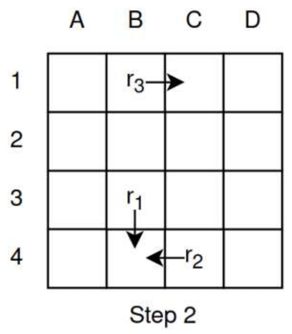
Figure 2.6: Actions taken, neighbours and open list and build graph in step 1

Assume that in the current configuration (B3, C4, B1), there is only one possible neighbour configuration: (B4, B4, C1). As seen in Sub-Figure 2.7a, this would lead to a collision of r_1 and r_2. Therefore, the collision set (B4, B4, C1) is updated from an empty set to a set containing robots 1 and 2.

Due to the collision in this step, the information about it is backpropagated, and all possible neighbours of (B3, C4, B1) are expanded. However, this expansion considers only nodes around the robots that would collide in the next step. Hence, only nodes around r_1 and r_2 are expanded; r_3 still follow its calculated policy.

The expansion of configuration (B3, C4, B1) creates nine possible neighbour states which are (B4, B4, C1) - the state why this expansion happens - (B4, C3, C1), (B4, C4, C1), (B3, B4, C1), (B3, C3, C1), (B3, C4, C1), (C3, B4, C1), (C3, C3, C1) and (C3, C4, C1). States that would lead to a collision again are discarded and not considered anymore. The open list of Sub-Figure 2.8b shows all listed and not discarded states and their f-values. Now, M* picks the configuration with the lowest f-value, and the robots are moved accordingly. Since no collision occurs in this step, M* will proceed from here.

Robot r_2 is at its goal position and will not move anywhere. This is possible since standing still is considered as an action with no cost. The robots r_1 and r_3 still have to move since they do not have yet reached their goal position. M* follows the computed policy to drive them to their goals, which opens the configuration (C4, B4, C1). Moreover, it executes the actions of the robots; this can be seen in Sub-Figure 2.9a. Sub-Figure

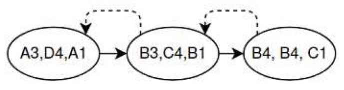


(a) Moves made by each robot in Step 2

Neighbours
(B4, B4, C1)

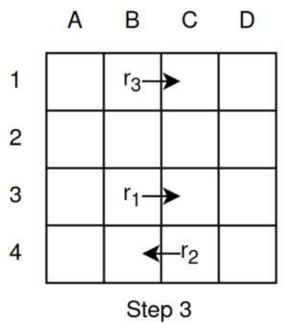
Open list		
Coordinate	F-value	Collision
(A4, C4, B1)	8	{}
(B4, B4, C1)	5	{1,2}

(b) Neighbours and open list in Step 2



(c) Graph built after moving Step 2

Figure 2.7: Actions taken, neighbours and open list and build graph in step 2



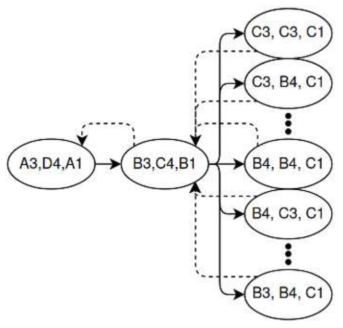
(a) Moves made by each robot in Step 3

Neighbours

(B4, B4, C1)	(B3, B4, C1)	(C3, B4, C1)
(B4, C3, C1)	(B3, C3, C1)	(C3, C3, C1)
(B4, C4, C1)	(B3, C4, C1)	(C3, C4, C1)

Open list		
Coordinate	F-value	Collision
(A4, C4, B1)	8	{}
(C3, B4, C1)	4	{}
(C3, C4, C1)	5	{}
(B4, C3, C1)	5	{}
(B3, B4, C1)	6	{}
(B3, B4, C1)	6	{}
(B3, C4, C1)	7	{}
(B4, C4, C1)	7	{}

(b) Neighbours and open list in Step 3



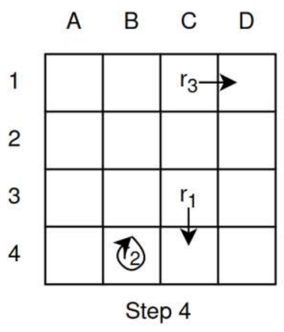
(c) Graph built after moving Step 3

Figure 2.8: Actions taken, neighbours and open list and build graph in step 3

2.10a shows that all robots reached their goal. Therefore, none of them took any further steps, and M* successfully completed the path planning for this scenario.

2.4.1 Variants of M*

M* is a state-of-the-art path planning algorithm that computes an optimal solution for multi-robot systems. Due to this property, it is a good baseline for further improvements in the field of multi-robot systems. This section summarises different directions of improvement of M*, such as fusing it with the multiple travelling salesperson problem. Moreover, by using the help of AI to reduce conflicts that occur during the planning phase. Furthermore, there is research going on to optimise the planning process not only for one objective, such as path length but consider more parameters, such as power

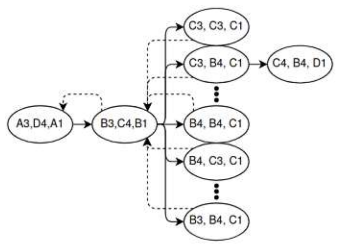


(a) Moves made by each robot in Step 4

Neighbours
(C4, B4, D1)

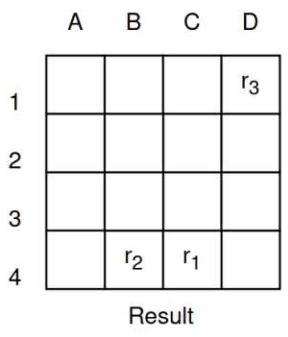
Open list		
Coordinate	F-value	Collision
(A4, C4, B1)	8	{}
(C3, C4, C1)	5	{}
(B4, C3, C1)	5	{}
(B3, B4, C1)	6	{}
(B3, B4, C1)	6	{}
(B3, C4, C1)	7	{}
(B4, C4, C1)	7	{}
(C4, B4, D1)	2	{}

(b) Neighbours and open list in Step 4



(c) Graph built after moving Step 4

Figure 2.9: Actions taken, neighbours and open list and build graph in step 4

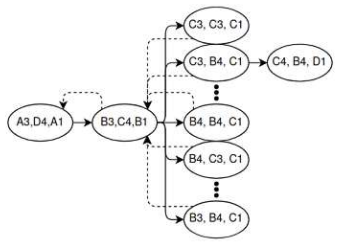


(a) Resulting positions after four action steps

Neighbours

Open list		
Coordinate	F-value	Collision
(A4, C4, B1)	8	{}
(C3, C4, C1)	5	{}
(B4, C3, C1)	5	{}
(B3, B4, C1)	6	{}
(B3, B4, C1)	6	{}
(B3, C4, C1)	7	{}
(B4, C4, C1)	7	{}

(b) Resulting open list after four action steps



(c) Resulting graph after four action steps

Figure 2.10: Resulting in an open list and graph after four action steps

consumption and environment coverage.

According to the authors of this paper [11], most research focuses only on one goal per robot. However, in real-world applications, a robot often has multiple goals before successfully completing a task. Due to this, the generated robot routes could be optimised for driving. In order to address this issue, the commonly known problem of multi-robot routing is extended with a multiple Travelling Salesperson Problem (mTSP). Therefore, the original M* is modified to solve the mTSP by policy construction and heuristic value, and conflicts are ignored in the first place. Afterwards, backpropagation similar to the original algorithm is performed. The authors also show that their solution is complete and optimal. Unfortunately, as the number of agents grows above ten, the success rate drops below 80% in a typical warehouse environment [11].

This paper [13] uses an attention-based neural network in combination with the M*. The authors are planning the robot fleet's paths and attempting to minimise collisions during the individual planning phase. Therefore, the model is fed with the observation of each robot after every action is taken by each robot individually. Additional information, such as the start and goal position, is also shared. The proposed approach, Learning-assisted M* (LM*) can find solutions faster than the original M* due to the reduced number of conflicts that must be resolved. Furthermore, the authors showed that their solution can generalise, which means it still calculates paths faster on maps that were not seen during training. Nonetheless, the model does not find the optimal solution. Which results in an approximately 10% more comprehensive solution [13].

Most path planning algorithms focus on optimising one specific metric: the path length from a start to an endpoint. In order to address also multiple objectives such as power consumption or time-to-completion, the authors of this paper [12] extended the M* planning algorithm to consider more metrics during the planning process. Applying existing solutions for multi-objective single robot search algorithms might lead to an inefficient solution, but they are used as a baseline for comparison. The proposed algorithm can successfully handle four objectives and overcome the baseline regarding average run time and solved scenarios [12].

2.4.2 MRRP Framework

The open source multi-robot framework presented in this paper [4] is used as the basis for this thesis. The authors presented an approach based on prioritised planning to target the issue of efficiently controlling mobile robots. Therefore, four scenarios Sequential, Wait, Avoid and Push, which occur in every multi-robot application are defined. Sequential means one robot follows another one. The Wait scenario is used to describe the situation when one robot has to wait until another has passed a specific area. With Avoid the authors are describing the dodging scheme. That means a lower-priority robot has to leave its path in order to let a higher-priority robot pass. Push is used if a lower-priority robot is already at his goal position and blocks a higher-priority one. Therefore, the higher-priority pushes the lower-priority one until the blocking one can be passed without collision.

In order to simulate these scenarios in different environments the authors designed a multi-robot framework. Figure 2.11 shows the structure of the framework used in the paper [4]. The structure shows only relevant parts for the planning and execution of paths. It does not deal with pick-up and drop-off stations or order management as it would be needed in warehouses.

It is assumed that each vehicle has the same computational power and is equipped with a similar set of sensors. A self-localisation node and a map server ensure that the mobile robot has a map of the surrounding environment and is able to estimate its position in it. The local motion controller is responsible to the steer the robot and following the given path of the local behaviour controller. The local behaviour controller receives a computed route from the multi-robot router. This route guides the robot on the basis of control points through the environment. The behaviour controller calculates the difference between the robot's estimated position to the received route, it then computes a path which is handed over to the local motion controller. Additionally, the local behaviour controller periodically publishes a robot-info message and receives similar messages from all other vehicles in the system. This message contains the robot's ID, its geometric shape, estimated pose with covariance, and many more. By comparing the estimated pose and the calculated path, the behaviour controller ensures that the following section is free to drive.

The multi-robot route planner consists of a map server, a map-to-graph node, and a multi-robot router. The map server publishes the map of the environment, which is then converted into a Voronoi graph by the map-to-graph node and sent to the multi-robot router.

The multi-robot router takes a Voronoi graph of the environment and generates a routing table. A single-robot router builds the core of the multi-robot router and computes a route for every robot. While planning the path the router ensures that vertices are not occupied simultaneously by prior planned routes. If the planning algorithm wants to expand a vertex in the graph in a certain time step it needs the permission of the route coordinator. The route coordinator checks if the vertex is occupied in the requested time step by any prior planned route. If this is the case, the issue is passed to a collision resolver in order to resolve it by extending the search graph. Supplementary, the multi-robot router is equipped with a priority and speed resolver in order to alter the priority and maximum speed of mobile robots if no solution is found.

2.5 Path planning with Time Uncertainty

Many multi-robot path planning algorithms assume that a robot will need a fixed amount of time when travelling on an edge. In practice, mobile agents cannot execute the planned path exactly for multiple reasons. For example, some dynamics of the agent cannot be modelled as detailed as would be needed. Furthermore, the measured values of sensors and localisation systems can deviate in a certain range from the real value, which leads to significant uncertainty in the robot's pose [7].

To categorise solutions to path planning problems, the authors of this paper [8] introduce the following characteristics: weak planning, strong planning and strong cyclic planning. The authors use symbolic model-checking techniques and implement them in different path planners [8].

If a solution has a chance of success, it is categorised as weak planning, which corresponds to an optimistic plan. For example, these are plans that can reach the goal if the action outcomes result in sequences that lead to the goal. A strong planning solution is guaran-

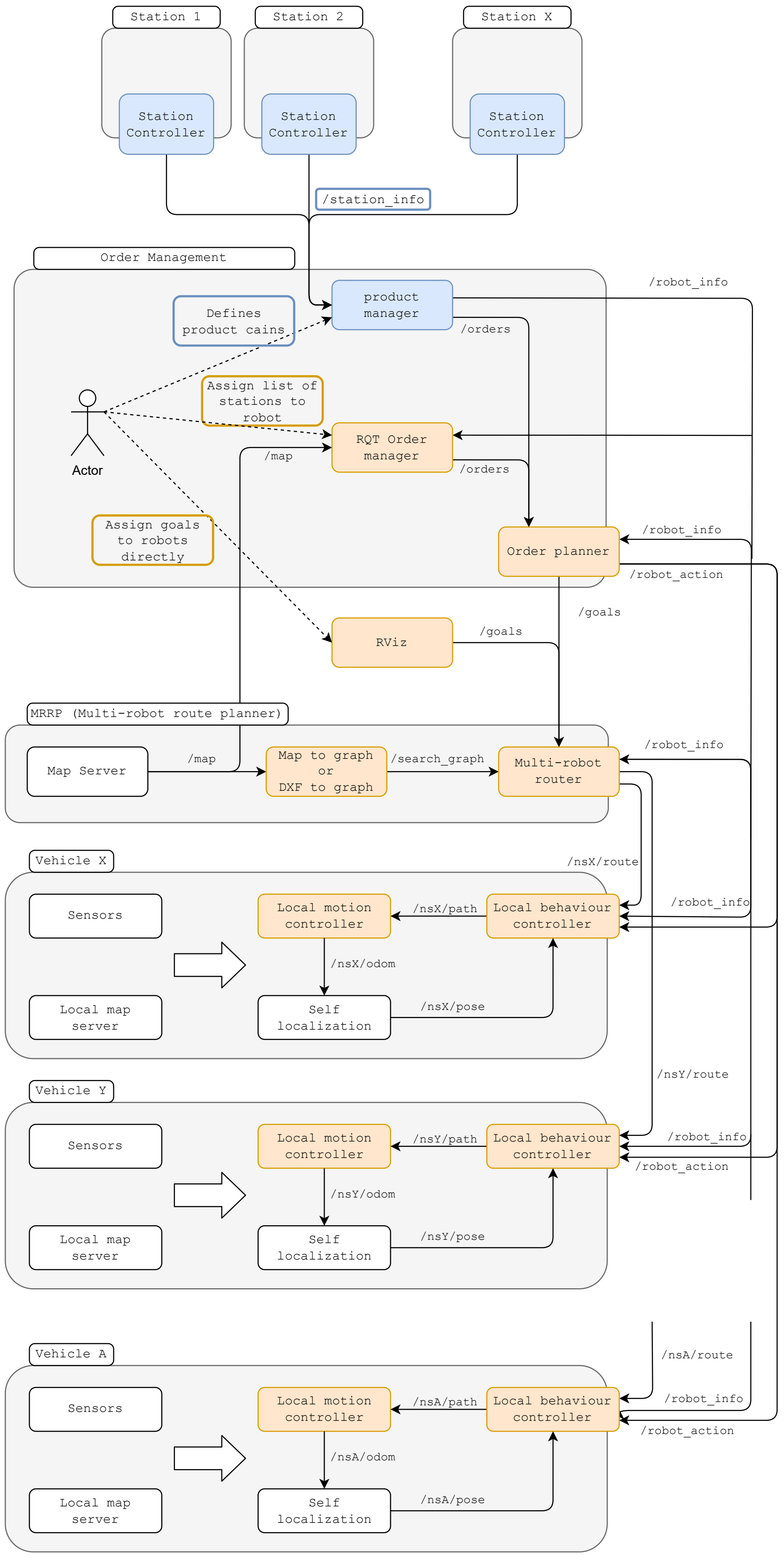


Figure 2.11: Multi-robot framework structure, adapted and based on [4]

teed to achieve the goal, which means that the goal is still reached independently of all uncertain actions. They are also referred to as safe plans. There might be cases where strong solutions do not exist, and weak solutions might not be accepted. In this case, the authors created the alternative of strong cyclic planning, which is the acceptable one. These are guaranteed to achieve the goal under certain circumstances.

The authors of the original M* algorithm paper [1] addressed the problem of uncertainty in their paper [7] by combining the M* algorithm with a belief space planning to introduce uncertainty M* (UM*). Additionally, an algorithm, PUM*, is introduced as an extension of UM*, which uses random restarts to enhance performance. Commonly belief space planning is used for single robot path planning problems. A belief space is constructed by choosing a simple probabilistic representation for the single agent position to reflect an imperfect localisation, for example, a Gaussian. Afterwards, a conventional path planning algorithm is used to compute a trajectory through the resulting belief space [7].

The authors were mainly interested in the problem when individual robots were highly capable in terms of computation power, and the main challenge arises from a lack of synchronisation between them.

The PUM* is compared to a rM* (introduced in [1]), and is shown that the PUM* solves problem instances of 40 agents with a similar success rate as rM* does by planning paths for robot systems of 180 robots. Nevertheless, if one of the 180 robots is not on time, the system is not guaranteed to finish the task save without collision. According to the paper, there are two major reasons for this. First, introducing uncertainty in the search space results in a larger space occupation by each robot. Secondly, the belief distribution runs behind each agent's nominal position, making constraining violations inevitable several times before they occur [7].

Considering time uncertainty during the planning phase models many different scenarios, but certain events are difficult to predict accurately. The authors of this paper [9] name the following example. In many automated warehouses, humans are responsible for packing items into boxes, and mobile robots move these packages around. The time a human needs to pack a box cannot be modelled precisely because it depends on many factors. However, the authors want to find a safe solution, as discussed above, and explore possibilities of online replanning. Online (re-)planning means modifying the robots' plan during the execution, unlike offline (re-)planning, where plans are computed before agents move. As a prerequisite for online replanning, all agents can sense the current time and location during the execution, and they have communication capabilities to inform all other robots in the fleet.

The authors chose a conventional CBS planner as a base and extended it to consider time uncertainty. The introduced algorithm performs significantly better in almost all cases [9]. Furthermore, sensing and communication capabilities are added to the agents. Therefore, every observation can be communicated to a central controller that may replan for multiple agents at once during the execution of the offline computed plan. The paper shows that online replanning techniques can reduce execution costs by a factor of 1.5

compared to the original offline solution [9].

Online replanning algorithms require runtime effort and additional costly infrastructure to provide steady networks and monitoring systems [6]. To address this problem, the authors of this paper [6] introduce a novel path planning approach called Offline Time-independent multi-agent Path Planning. Their studies investigate paths computed independently of timing uncertainties so that every robot within the system can act spontaneously and without any timing assumptions. Two versions to solve this problem are introduced, first using prioritised planning and, secondly, a CBS algorithm. The main challenge is to detect potential cyclic deadlocks, which is a NP-Hard problem [6]. The authors show that their proposed algorithm can solve problem instances on a grid map, represented as a four-connected graph, and up to 40 robots with a high success rate within 3 minutes. However, increasing the number of robots in the fleet reduced the success rate drastically.

Utilising probabilistic information about imperfect plan execution can reduce frequent and time-intensive replanning and plan execution failures [14]. Therefore, the authors of this paper [14] propose two robust plan execution policies for their Multi-Agent Path Planning with delay probabilities (MAPF-DP) problem. MAPF-DP is an extension of classical MAPF with the extra condition of robots which, would move, having a certain chance of staying on their actual vertex. The plan-execution policy controls how agents proceed along their paths with STOP and GO commands.

Policies can either be fully synchronised, which means each agent sends messages to all other agents or minimal communication where the policy identifies critical dependencies between agents and observes them during plan execution. The authors showed that combing the MAPF-DP and a minimal communication policy results in slightly smaller average makespans than their baseline, which is an adapted version of CBS and a Push and Swap solver.

2.6 Voronoi Diagrams

Path planning for a continuous configuration space is computationally infeasible; therefore, an abstraction of the configuration space is needed [2]. These abstractions are commonly known as road maps. A road map is a graph with a finite number of vertices, where each vertex represents a valid configuration in the search space [21]. Voronoi diagrams are well suited for mobile robot path planning applications [22], [23]. These diagrams guarantee a maximum clearance distance to all obstacles on the map at every point.

The generation of a Voronoi diagram is a two-step process. In the first step, the gird-map of the environment is transformed into a distance map using the Euclidean distance transformation. Secondly, a thinning algorithm is applied on the distance map to generate the Voronoi path because these paths can only be one pixel wide.

An approach to generate a Voronoi path is presented in [24]. The authors use the advanced thinning algorithm of [25], to create a one-pixel wide Voronoi path. Unfortunately, this has the drawback of running the thinning process multiple times before generating a suitable path. This paper [26] presents a thinning approach to reduce computation time by computing all saddle points, endpoints and maxima of a result of the distance transformation, which are the starting points for creating a Voronoi path. After detecting starting points, a maximum search is executed in the neighbourhood of it. The maximum found by this search is marked as the next starting pixel. This process will build up the paths and need less computation time.

The framework presented in Sub-Section 2.4.2 computes the Voronoi path similarly. A path segmentation is also applied to generate a graph for the path planning process. For segmentation, the pixels are separated into different categories: Path pixels and crossing pixels. Since the generated path is one pixel wide, each pixel with more than two neighbours is considered a crossing pixel. Starting from one crossing pixel, a Dijkstra expansion is performed to find and link all crossing pixels without a path pixel in between. After the crossing section is marked, another Dijkstra expansion, starting from the edge of an intersection, connects the different junctions and builds up the graph. The part between two crossings is referred to as a segment. A segment has a maximum length, and if this length is exceeded, the algorithm splits this segment into several smaller ones to fulfil the maximum length criteria.

CHAPTER 3

Problem Description

Multi-robot fleets often operate in warehouse environments. Such environments consist of broader passages, where robots can pass each other, and narrow hallways, allowing only one robot to drive through. Therefore, the challenge arises in coordinating the robots in the system to effectively navigate them through the environment without colliding with each other and ensure quality for the computed paths. This thesis focuses on comparing two path planning approaches, M* and CBS, against each other and defining quality measurements for the paths they computed.

Each robot in the fleet is equipped with the same set of sensors and actors, such as a LiDAR system[1], to perform a self-localisation within the environment. Additionally, every robot in the system has the same physical measurements and drives at the same speed.

The environment is mapped to a graph that is generated from a Voronoi diagram, see Sub-Section 2.6. Robots are not allowed to move freely in the environment. They are bound to the generated graph and can move along its edges. A graph vertex represents a crossing in the environment or is inserted to limit the distance between two vertices to a maximum edge length. Each robot is placed at a starting location and assigned a goal position within the environment, and the planning algorithm computes a route between these two points. Furthermore, only one robot at a time can move along an edge or stand still on a vertex.

[1]Light detection and ranging is a method to determine ranges with a laser by measuring time of flight of the reflected light.

The following research questions motivated this thesis:

RQ1: What are quality measurements of multi-robot planner results?

RQ2: How does the M* perform in contrast to the currently implemented conflict-based search in the domain of multi-robot applications?

RQ3: To what extent is the M* algorithm applicable to real-world multi-robot applications in terms of runtime and scalability?

Experiment 1

In order to answer **RQ1**, literature research is conducted to define quality measurements for computed paths of multi-robot path planners. Additionally, a ROS-Node will be implemented into the existing framework in order to perform these measurements and evaluations during the execution of the plan without directly interacting with the robot fleet.

Experiment 2

Simulation results will answer **RQ2**. Both the currently implementedCBS and during this thesis implemented M* planning algorithm, have to solve different scenarios. One simulation run will place robots at a certain starting location and assign goal positions for each one. During the simulation, multiple parameters are recorded to compare both implementations. One of the most important things is whether the algorithm used is able to solve the given situation. Additionally, recorded performance indicators are the computation time needed to solve the scenario and how many nodes are expanded by each implementation.

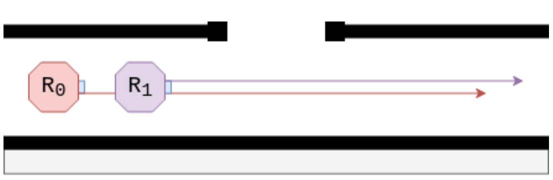
These scenarios were defined by the authors of the original implementation of the framework [4].

Figure 3.1 shows six different problem instances. In the first scenario, shown in Sub-Figure 3.1a two robots move sequentially. The second problem instance is shown in Sub-Figure 3.1b, where robot R_0 exits the hallway and robot R_1 crosses the path at the exit. Therefore, the second robot has to wait until the first one exits the hallway.

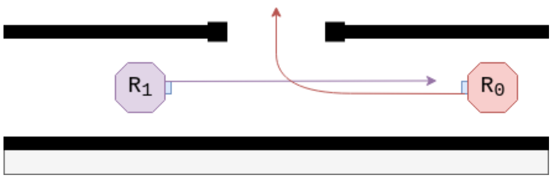
Within a multi-robot system, two robots often have to pass each other at a narrow spot. This is exactly what the third scenario (Sub-Figure 3.1c) describes. The path planning algorithm should find a solution to swap places of the robots.

Sub-Figure 3.1e shows robot R_1 blocking the way of robot R_0 because it has already reached the goal position. In order to allow R_0 to pass by, the first robot has to move back and forth again.

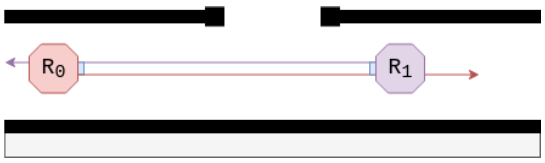
The last scenario is similar to the fifth one, but instead of robot R_1 reaching the goal in front of the hallway exit, R_1 stops somewhere between the exit and its starting position.



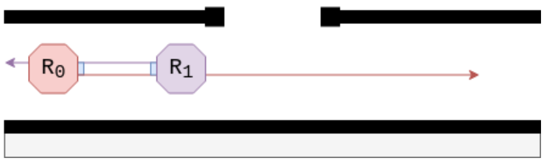
(a) **Scenario 1:** Sequential



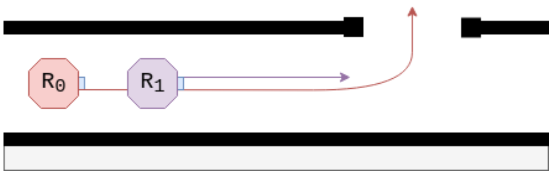
(b) **Scenario 2:** Wait



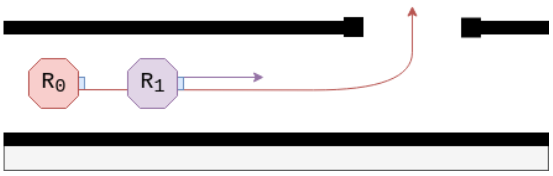
(c) **Scenario 3:** Avoid at Crossing



(d) **Scenario 4:** Avoid at Start



(e) **Scenario 5:** Avoid at Goal



(f) **Scenario 6:** Push

Figure 3.1: Common scenarios in multi-robot environments; based on and adapted from [4]

Therefore, it blocks the whole hallway for other robots. In order to prevent a deadlock, the conflict-based algorithm pushes the R_1 as far as needed towards the hallway exit.

Experiment 3

As well as the second research question, **RQ3**, will be answered using simulation results. Therefore, the framework described in Sub-Section 2.4.2 will be extended by another planning algorithm, the M*. One simulation run will set each robot at a predefined start position and assign an individual goal location. During the planning and execution phase, different parameters will be recorded, for example, the time used for planning, the execution time needed until every robot reaches its goal, and the number of collisions that have to be recovered. The number of robots will be increased after 50 simulation runs.

CHAPTER 4

Implementation

4.1 Terminology

This section should lay a common ground for the terminology used in this thesis.

The following glossary (Table 4.1) explains the most important terms and definitions used in this thesis:

Term/Definition	Description
Graph	A graph is the search graph the planner gets as input as a simplified representation of the environment. One vertex in the graph maps to exactly one segment in the environment.
Segment	A Segment refers to a certain area in the environment. It can have any shape and map to exactly on vertex of the corresponding graph.
Route	A Route is a list of segments describing a path from a start to an endpoint.
Routing Table	A collection of individual routes is called a routing table.
Policy	A policy is a collection of computed individual paths without further checks regarding robot-robot collisions. Introduces by the authors of this paper [1].

Table 4.1: Important terms and definitions

4.2 Integrating M* in to the Framework

The planning algorithm M* is used as the underlying planner for this thesis. In order to integrate this properly into the multi-robot frameworks, some changes have to be made.

As it can be seen in Figure 4.2 the multi-robot router node is replaced by the M*-planner node. The new path planner subscribes to the search graph message, published by the map-to-graph node and the goals messages that Rviz2. Rviz2^0 is a graphical interface for ROS2 that visualises robot information and displays topic data. It is extendable by user-defined plugins. As discussed in Section 2.4, M* computes a policy before starting the planning task for the multi-robot system. In order to compute a policy, a single-robot planner such as A* - or any variant [16] - can be used. If the robot fleet consists of many robots and the environment is complex, computing the policy can be time-consuming. Also, the algorithm has to perform pre-processing and post-processing steps for each robot individually.

Following the ROS design pattern for long-running tasks, the M*-planner node is split into two parts, as shown in Figure 4.1. The policy planner node implements an action server, and the path planner node implements an action client. After receiving the search graph and goals message, the action client of the path planner node requests a policy from the action server of the policy planner node. The computed policy is returned to the calling client, generating a plan according to the M* algorithm as described in Section 2.4. One big advantage of using this design pattern is that the policy planning algorithm can be easily exchanged for another if a better-fitting solution exists.

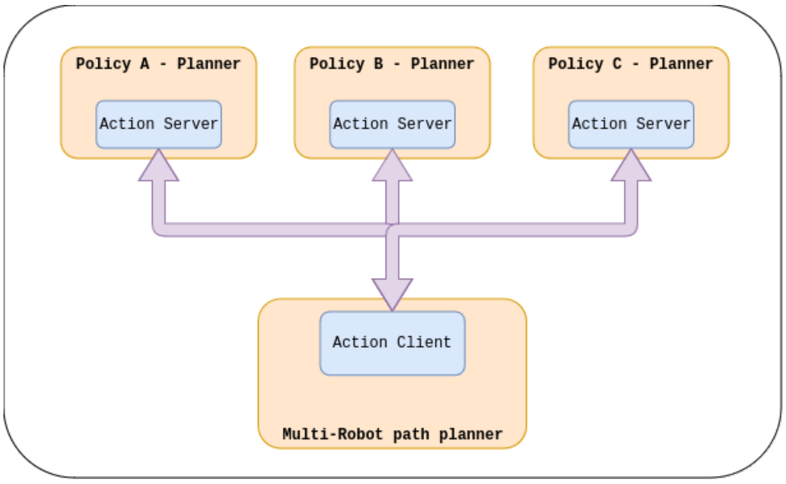


Figure 4.1: M*-planner schematics

A route that the originally implemented multi-robot route planner has planned differs from classic paths because these routes contain synchronisations points [4]. Synchronisation points are special nodes in the environment where agents can pause to let other agents pass. Instead of inserting synchronisation points in the routes, the path planning algorithm M* doubles the current vertex so robots can wait. Adding such points allows any vehicle to be parked at any vertex in the environment for some time. Publishing routes without synchronisation points to the mobile robots results in crashes since the local behaviour controller considers a sequence of multiple vertices as a single one. Therefore, an additional node, the route synchroniser node, is added to the framework. Furthermore, this preserves compatibility with the original framework functionality.

The route synchroniser node receives a routing table message from the M* planner node. Additionally, the synchroniser subscribes to the robot-info messages published by each vehicle. The robot-info message contains the pose estimation of the self-localisation node and is published by each vehicle individually. Routes are organised in Route segments, which are at least as long as the largest robot of the fleet.

Before running the synchronisation, the node processes every route segment. It calculates the midpoint and stores this instead of all the information for each segment. This will speed up the comparison process because the overhead for each comparison is minimised. Such a reduction is only possible since a straight line of sight connects a route segment's start and end points After publishing the first route segment, the synchroniser node periodically compares the robot's actual pose estimation to the current goal segments mid. If every robot has reached its goal segment, the next goals will be published to the whole fleet.

This synchronisation method assumes that robots in the fleet move at similar and constant speeds and that route segments have similar lengths. If not, the whole system could be slowed down or blocked by a single robot.

4.3 Quality measurement

Quality measurements for multi-robot planner results are important for assessing the effectiveness and reliability of the computed paths. One of the most used parameters for ensuring effectiveness is the execution time of a plan. Another one might be the length of the computed paths. It is the major goal of almost every path planning algorithm to minimise the execution time of a plan or to find the shortest possible path between two positions, besides avoiding robot-robot collisions.

However, sometimes not only the shortest path or minimise the execution time is a necessary parameter to be optimised for. For example, for long-running tasks might be of interest to optimise power consumption to ensure robots can run at least as long as needed to finish the task. Therefore, depending on the parameter that should also be optimised, the quality assurance parameter changes.

This thesis pays close attention to the following parameters: time to solution, execution time, additional nodes inserted into the graph, and how stable the computed routing table is.

Time to the solution is the computation time of a planner from getting the position of the goals until it finishes the planning task. The execution time is measured between the first move of the first robot and the last agent arrives at the goal's destination.

Path planning for multi-robot systems happens in various ways. Some algorithms compute paths for each robot individually in the first step and ensure a collision-free path in the second step; other algorithms combine these two steps. However, both approaches must insert or duplicate the environment graph vertices. Inserted or duplicated nodes directly reflect on the memory usage of an algorithm and, therefore, might be interesting because it determines the requirements of large-scale multi-robot applications.

In order to assess the route stability of the computed multi-robot plan, an additional ROS-Node is integrated into the existing framework, operating alongside the multi-robot planner (See Figure 4.2). The introduced module, called Route Stability-Node, evaluates the robustness of the computed plan. Upon receiving the routing table, the node will perform simulations of robot uncertainties without interacting with the actual system.

Due to the abstraction of the environment, as in the Voronoi diagram, the simulation of timing uncertainties is limited to the discrete space rather than the continuous one. In order to address uncertainties, the node dynamically adjusts the estimated positions of various robots. A robot can either be placed on the previously visited segment or the next intended segment or remain on the current segment. Afterwards, the route stability-node simulates these uncertainties by executing the plan accordingly and recording any potential collisions during the simulation. This allows an evaluation of the stability of the multi-robot systems plan under varying conditions.

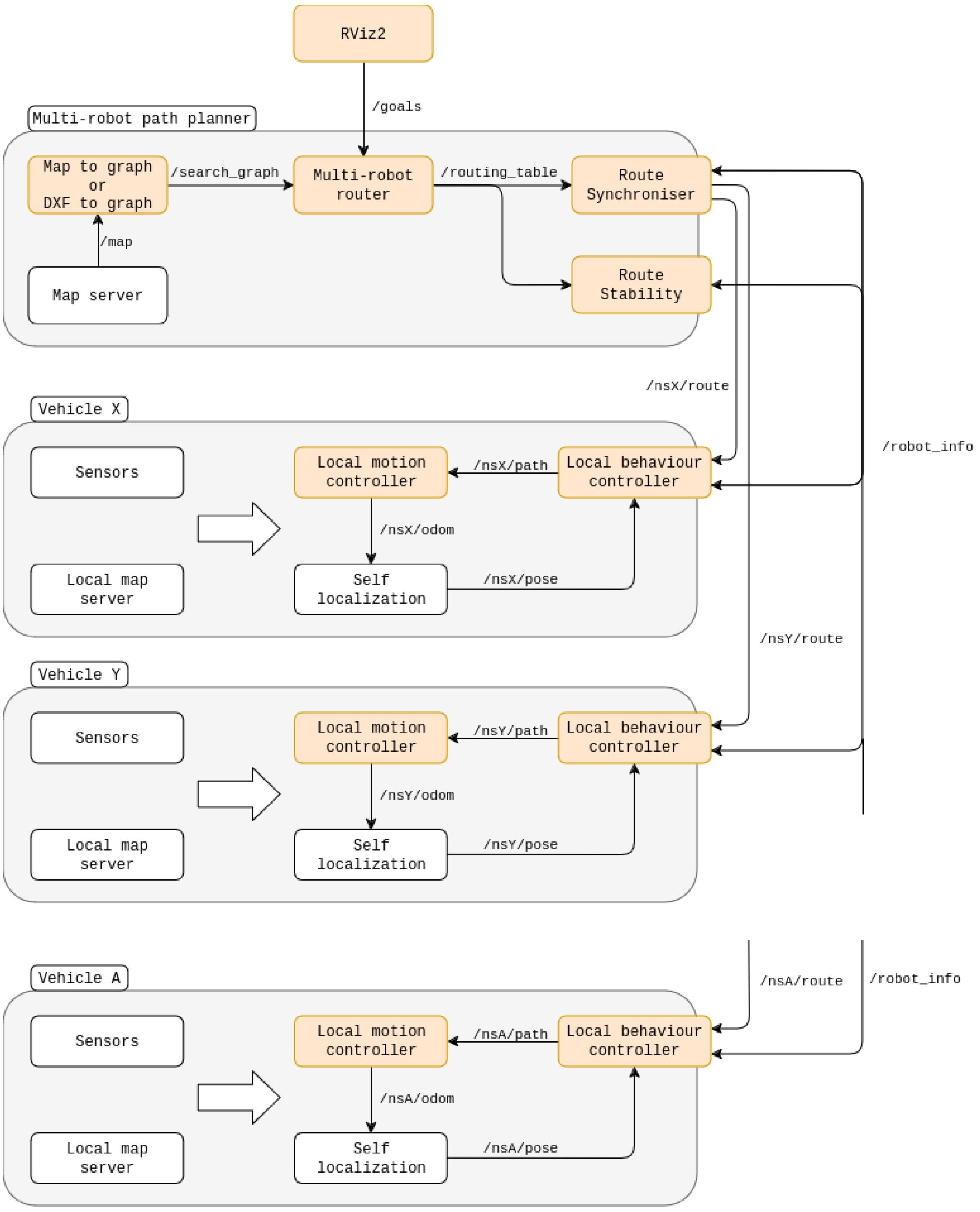


Figure 4.2: Updated system architecture of the multi-robot framework, adapted and based on [4]

CHAPTER 5

Results

This chapter presents the results to answer the formulated research questions in Chapter 3. The results are generated by simulations, which allow for gathering more results in a shorter period and increasing the number of robots independently of the given environment. The program Stage[1], part of the ROS environment, will be used as a simulation environment.

Section 5.1 presents the results of the quality measurements performed in combination with Experiment 3. In Section 5.2, the implementation of M* is compared against the, in the framework integrated, MRRP, by simulating special case scenarios. Section 5.3 will present the results of a warehouse simulation with an increasing number of agents. Again, the results are compared to those of the MRRP. The simulation environments will change depending on the number of robots in the system.

5.1 Quality Analysis

This section will present the results of quality parameters measured and evaluated during the simulation of the multi-robot system in the warehouse environment, which relates to Experiment 3. In order to compare both path planning algorithms only results successfully found a solution are considered in this section. The number of successfully solved problem instances varies depending on the number of robots in the fleet and also in the chosen environment.

The average execution time of each agent in the system is shown in Figures 5.1, 5.2 and 5.3. In the top left corner, the simulated environment is visible. A measurement of execution time starts with the first moving robot and ends after the last agent has

[1]Stage is a robot simulator. It provides a virtual world populated by mobile robots and sensors, along with various objects for the robots to sense and manipulate.

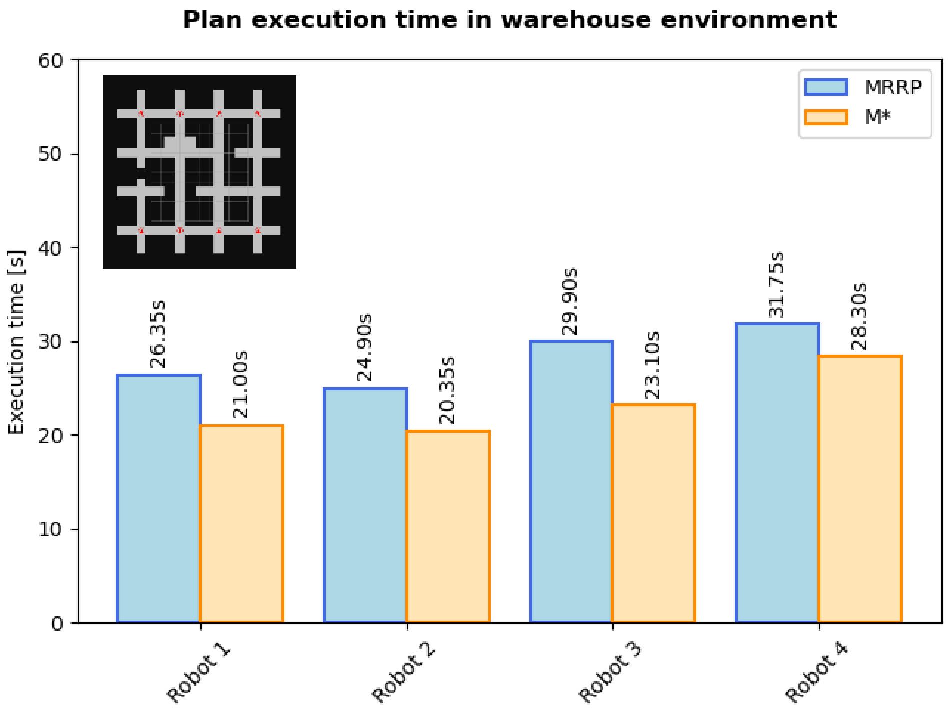


Figure 5.1: Plan execution time of robot fleet with 4 agents in the warehouse

reached its goal position. In general, the M* algorithm computes paths that are more time-efficient than the CBS path planner.

The M* path planning algorithm computes solutions that are up to 20% faster in total and 18% on average for a robot system with a maximum of four agents, see Figure 5.1. Similar results are achieved for a robot fleet with eight agents (see Figure 5.2), where the average execution time of a plan generated by M* is 17.6% faster compared to the CBS approach. As shown in Figure 5.3, the average difference in execution time between MRRP and M* for a fleet size of 16 robots is located by 7.1%.

Besides execution time, route stability is an indicator of the quality of a path. As described above, the stability node, which is integrated alongside the multi-robot path planner, receives the current executed path and the actual position of each robot in the environment. It then randomly alters the position by a fixed offset, for example, one node ahead or back on the actual path, simulating the whole system. The node will keep track of all collisions that might occur by simulating robot uncertainties. To evaluate the stability, all possible collision nodes are summed up and divided by the overall number of nodes existing in all simulations.

For example, take a multi-robot system with two robots. Moreover, each robot has a path with exactly eight nodes, and the position of the robot will be altered by one node. This results in nine different simulation settings, which need to be checked. Furthermore,

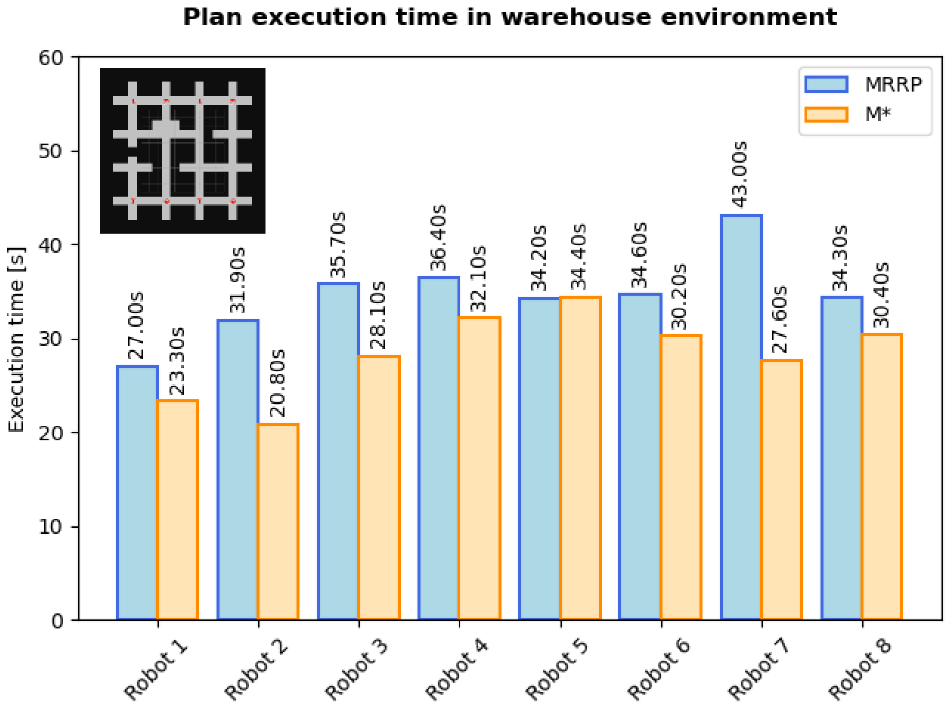


Figure 5.2: Plan execution time of robot fleet with 8 agents in the warehouse

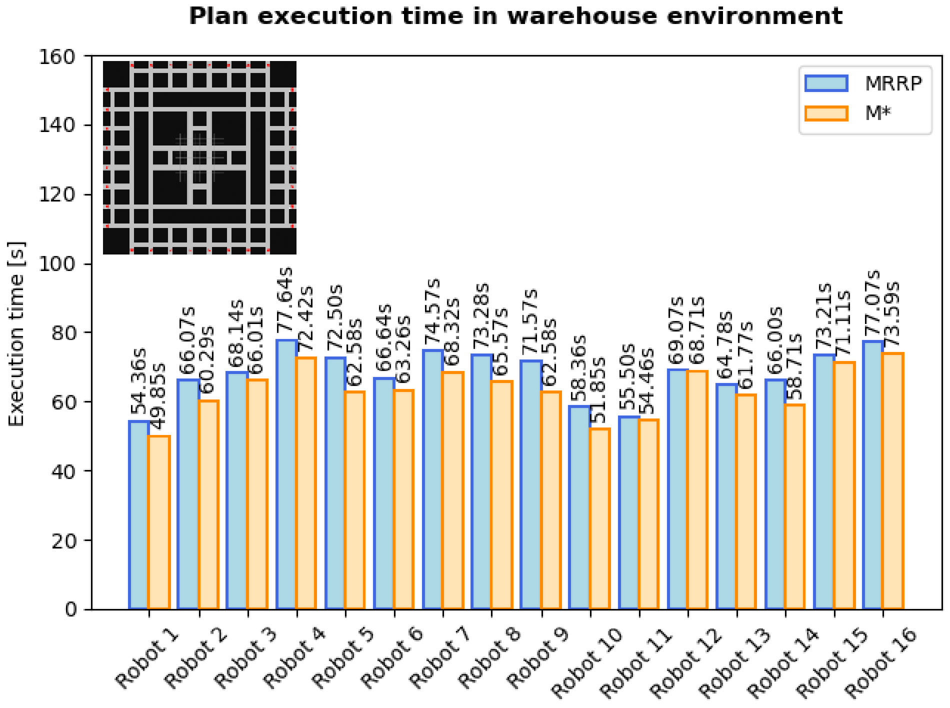


Figure 5.3: Plan execution time of robot fleet with 16 agents in the warehouse

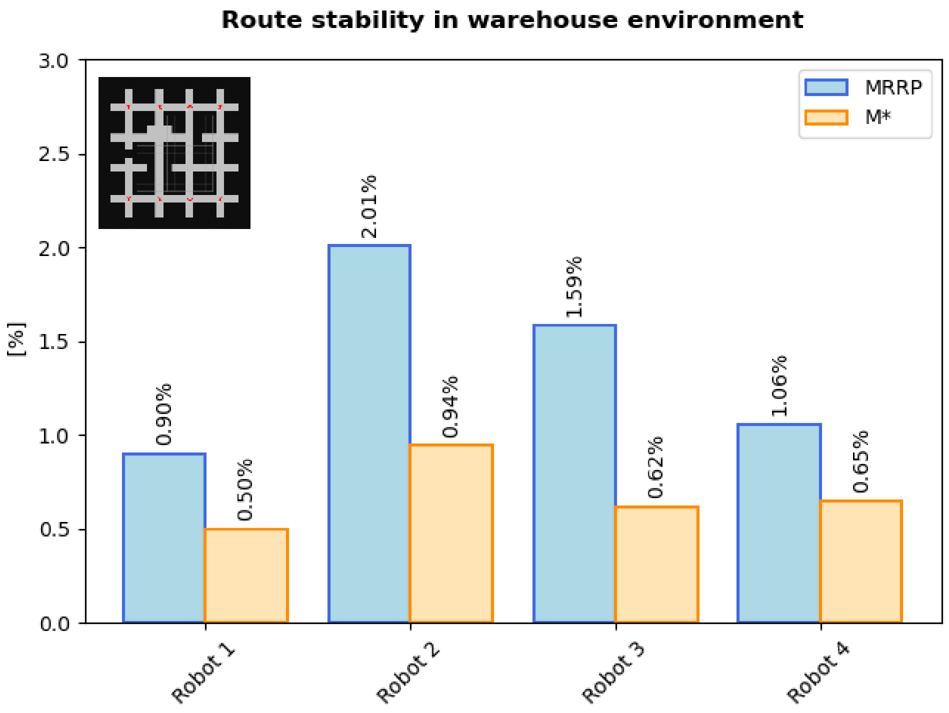


Figure 5.4: Stability of plan with 4 agents in the fleet in the warehouse

the overall number of nodes in all simulations for robot 1 is 72. Assuming in 16 out of these 72 possible nodes, these two robots would collide, then 22% of this route would have a possible collision. The planner has to take action to avoid collision due to timing and position uncertainties.

The results of this system simulations can be seen in Figures 5.4, 5.5 and 5.6, which shows the average stability of the route for each robot in the system. In the top left corner, the simulated environment is visible. Again, the M* algorithm computes paths that perform better in the analysis. It can be seen that most of the paths have at least 50% fewer potential collision nodes than the plans that are computed using MRRP.

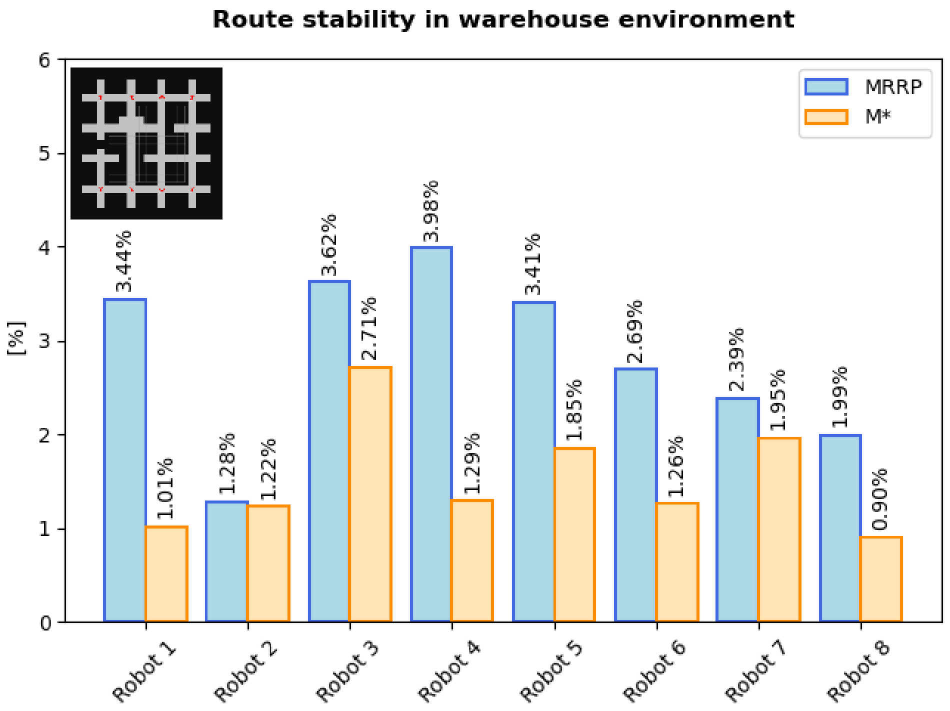


Figure 5.5: Stability of plan with 8 agents in the fleet in the warehouse

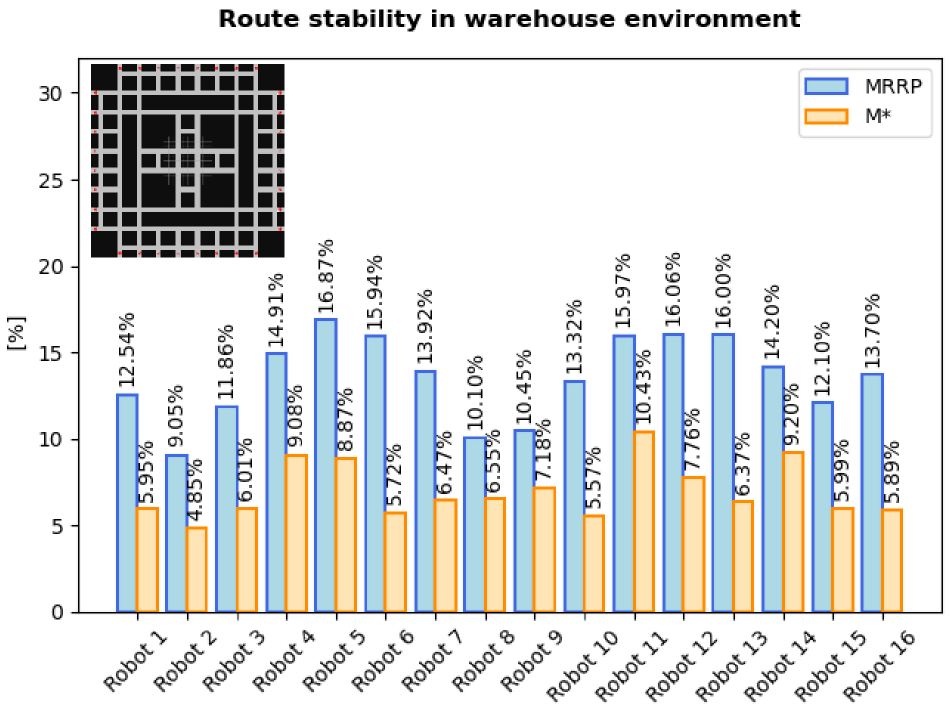


Figure 5.6: Stability of plan with 16 agents in the fleet in the warehouse

5.2 Scenario Simulation

This section will present the results of the simulation of various scenarios described in Section 3.

Each simulation will run multiple times. One run will place the mobile robots at their starting position and assign them goal locations within the environment. Each scenario describes a situation that might occur in real-world environments, so the planners are tested on them to measure the capabilities in real-world applications. The assigned goals are predefined and not randomly generated for each run, as in Section 5.3. Each path planner was able to solve every given problem instance within the given simulation time.

Nodes are counted as added if the planner expands or creates nodes not previously considered part of the robots by the planner. For example, for the M* planner, if two robots collide at node X, the planner expands all neighbouring nodes and adds them as potential successors of node X. All these potential successor nodes are added to the graph and, therefore, counted as added nodes. Like M*, the MRRP also adds vertices to the robot path that are not considered in the first step of path planning. These vertices are counted as added vertices for the MRRP planner. Also, planning time measurement starts when goals are received (MRRP planner) or the policy calculation is finished successfully (M* planner).

The average planning duration for each scenario and each path planning approach can be seen in Figure 5.7. It can be seen that the average planning duration of the M* algorithm is significantly higher, for certain scenarios than the average planning duration of the MRRP. It takes the M* planner up to 6 times as long as the MRRP planner to finish path computation for the given problem instances.

As shown in Figure 5.8, the M* planner adds much more to nodes' pre-planned paths than the MRRP planner per scenario; this results from the strategy of how M* expands nodes. The M* algorithm performs a decoupled path planning if there are not any collisions in the robot's path, but if a collision occurs, the algorithm switches to a locally coupled planning strategy and expands every possible node in the local area of the collision. This results in a higher number of added nodes than the conflict-based search approach the MRRP planner follows.

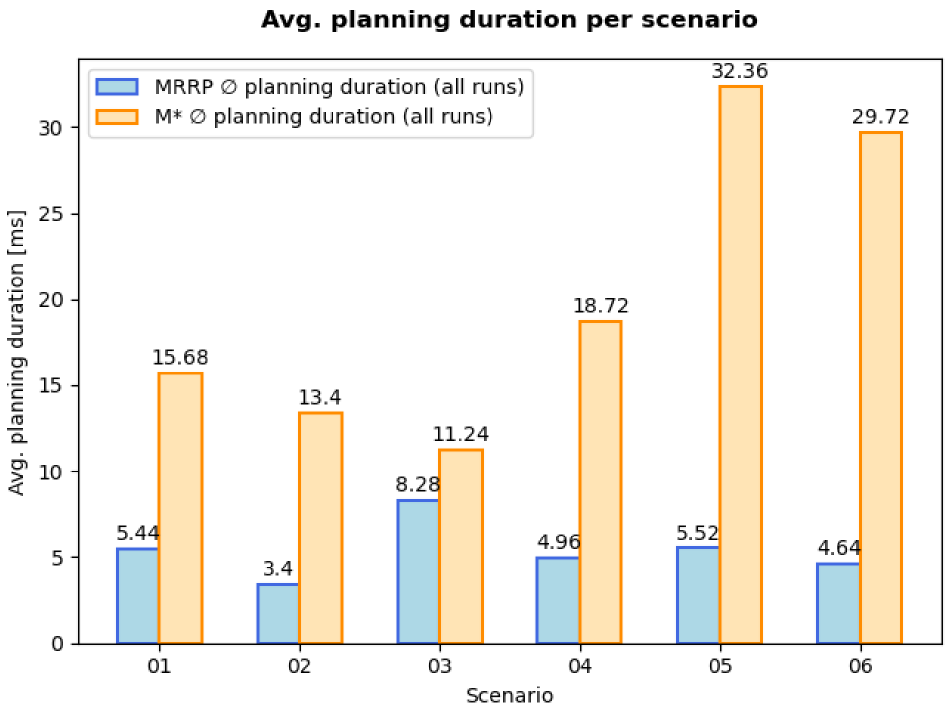


Figure 5.7: Avg. planning duration of MRRP and M* in different scenarios

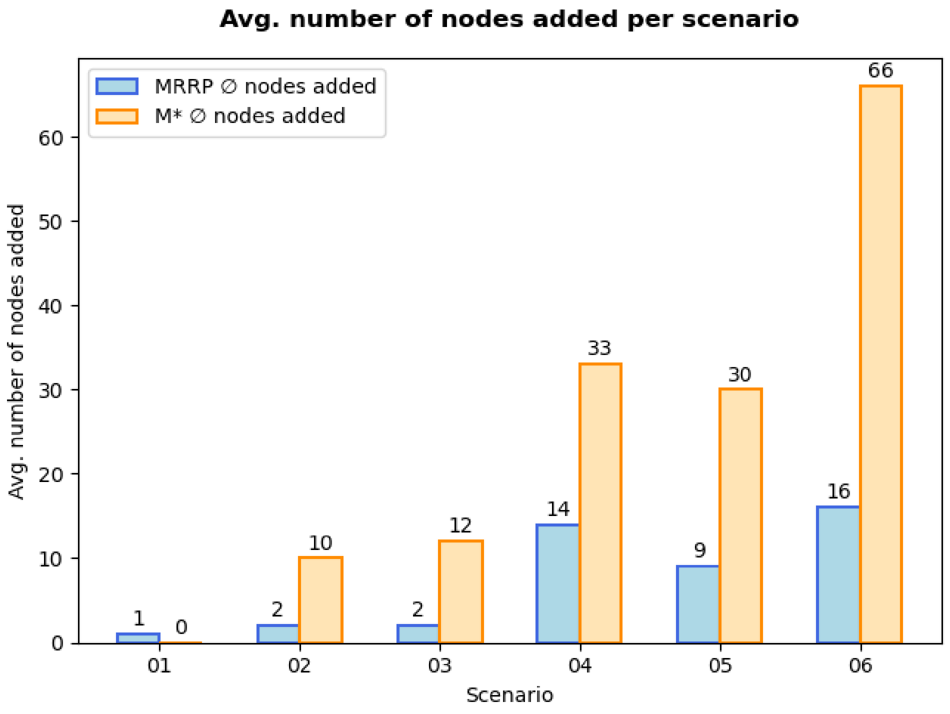
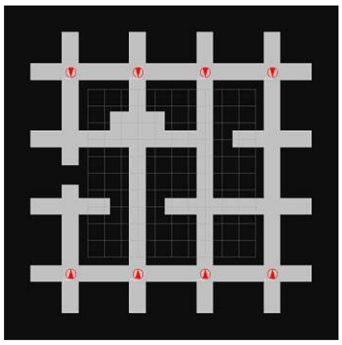


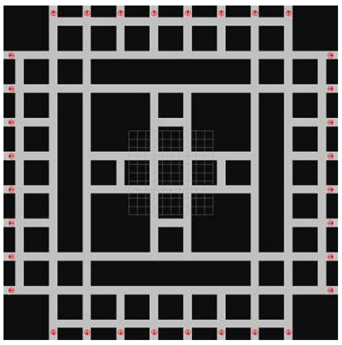
Figure 5.8: Avg. number of nodes added of MRRP and M* in different scenarios

5.3 Simulation of Warehouse environment

Modern warehouses and logistics applications heavily rely on robot fleets, which is why path planning algorithms are compared to each other in simulated warehouse environments in this thesis. In this section, the roots fleet is placed in different simulation environments, depending on the fleet size. Each robot has a single start and goal position assigned to it. Driving from start to goal is referred to as a task. A robot finishes its task successfully by reaching the goal position without colliding with another vehicle or blocking others in the system. Figure 5.9 shows the warehouse environments for systems with up to 8, 32, and 200 robots.

Each simulation will run multiple times; one run will place the robots at their starting positions and generate as many randomised goals as robots are currently in the system. The generated goals will be handed over to the MRRP and the M* planner to make their results comparable.


(a) Warehouse environments for up to eight robots


(b) Warehouse environment for up to 32 robots

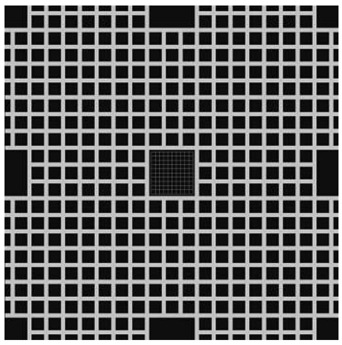

(c) Warehouse environment for up to 200 robots

Figure 5.9: Warehouse environments used for simulation

Figure 5.10 shows the success rates for the MRRP and M* planner over the growing number of robots. A simulation run is successful if the planner finds a solution to the given problem instance within a predefined period. Each planner's maximum allowed computation time has been set to 60 minutes.

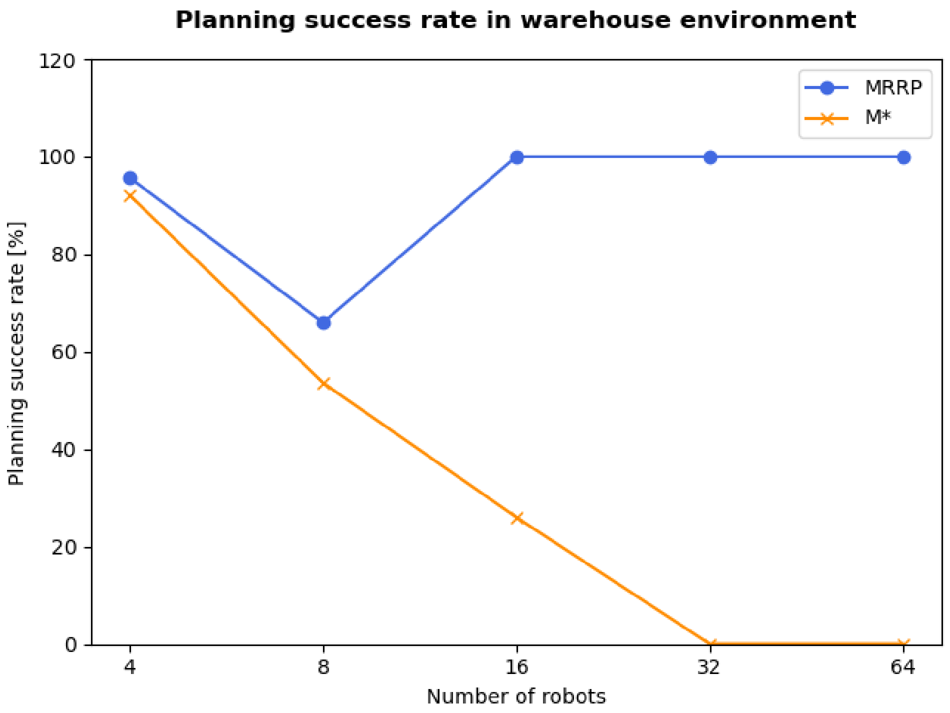


Figure 5.10: Success rate of MRRP and M* planner for different robot fleet sizes

In Figure 5.11, the average time to solution for the MRRP, and the M* planner can be seen. The planning time of the M* planner is measured after the calculation of the policy until the planner returns due to a found solution or it is forced to stop due to a timeout. The planning duration of the MRRP planner starts after the start positions and goal positions are mapped to the corresponding segments, and ends when the planner returns with or without a solution. The main difference between the measurements is the time scale. While the average planning time of the MRRP is measured in milliseconds, the planning duration of the M* algorithm is measured in seconds. The Figure shows the average time the planners need to compute a routing table for the multi-robot system. It is split up into measurements for all runs, including unsuccessful ones, and for successful runs only. For example, the MRRP needs, on average, 10.22 milliseconds to compute a valid routing table or to determine no possible solution. In contrast, the M* algorithm takes up to around 285 seconds on average to terminate its execution, regardless of whether there is a solution.

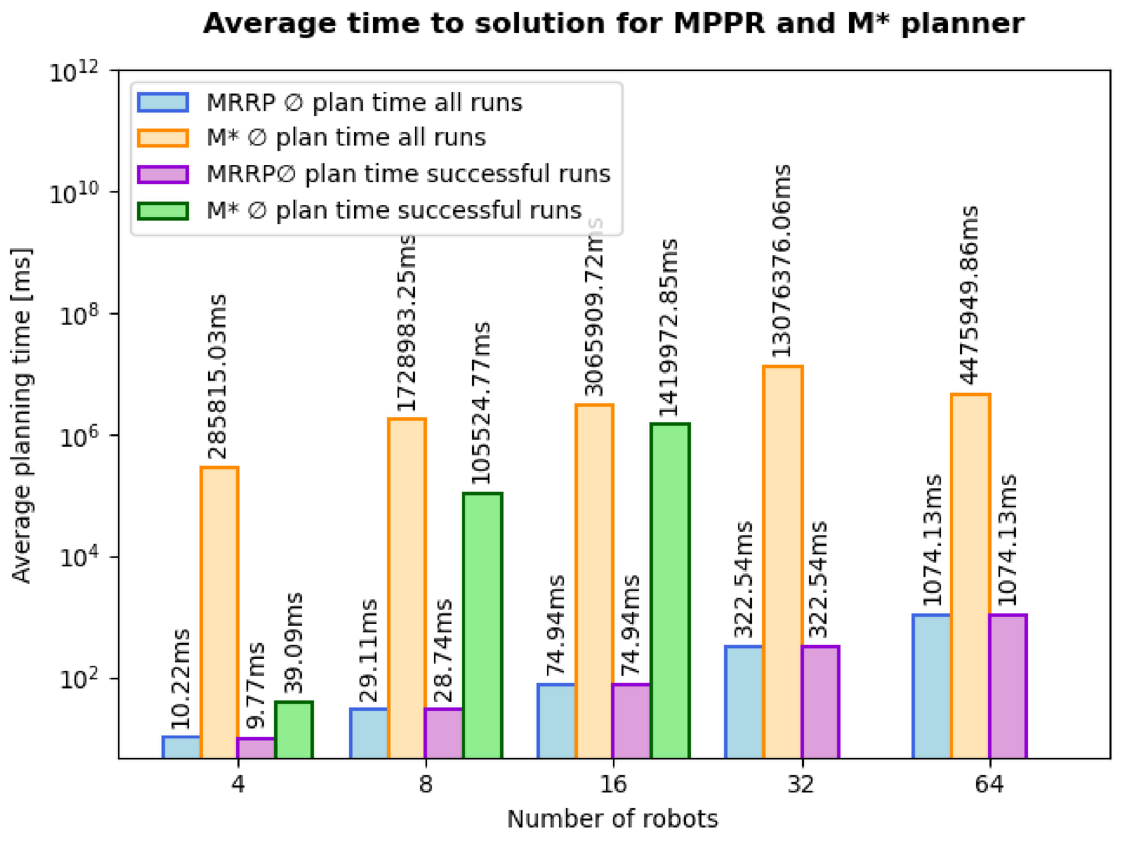
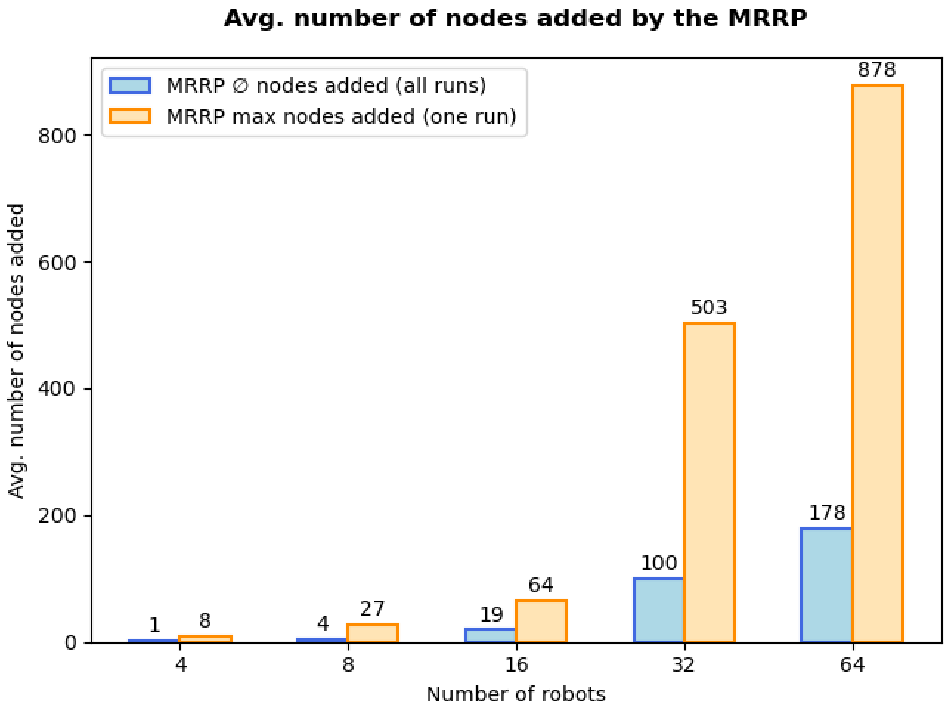


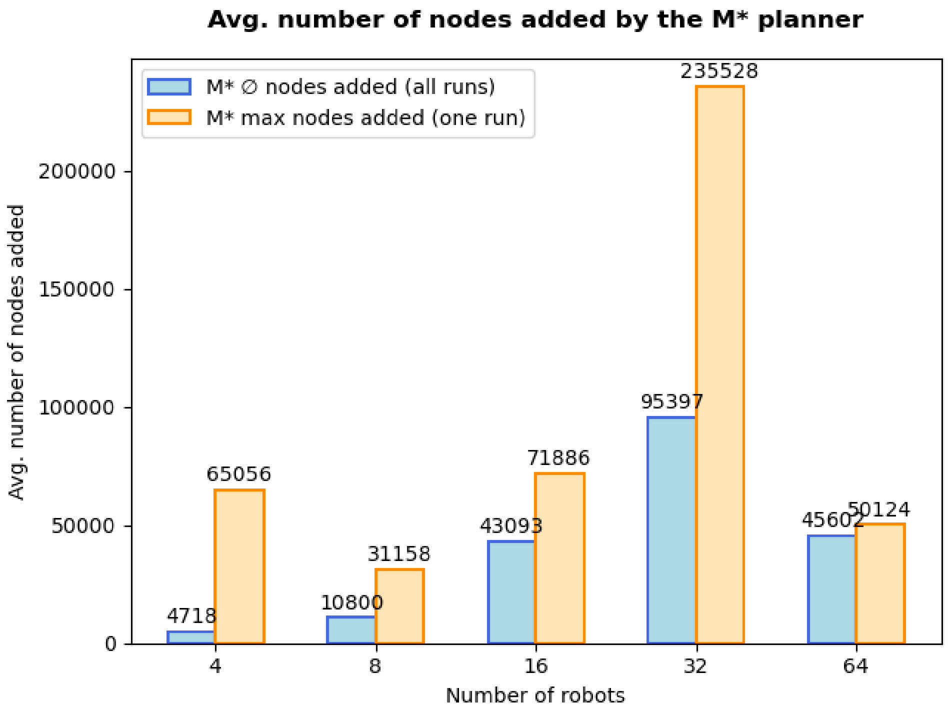
Figure 5.11: Avg. planning duration of MRRP and M* in warehouse environment

How many nodes are added on average to the graph for each planning approach can be seen in Figure 5.12. The added nodes are counted in the same ways as described in Section 5.2. This means added nodes are counted if the planner adds additional nodes in the robot path that were not considered in the first place when planning individual paths or computing the robot's policy.

Figures 5.13 - 5.28 show an example execution of a plan computed by both planners. On the left side, the execution of the plan calculated by M* can be seen; on the other side, the one computed by the MRRP planner. These Figures show that M* generates more efficient paths regarding the execution time because the plan is finished after roughly 40 seconds (Figure 5.23). In contrast to the M* algorithm, the MRRP planner generated a routing table that needed around 60 seconds to be executed (Figure 5.28).

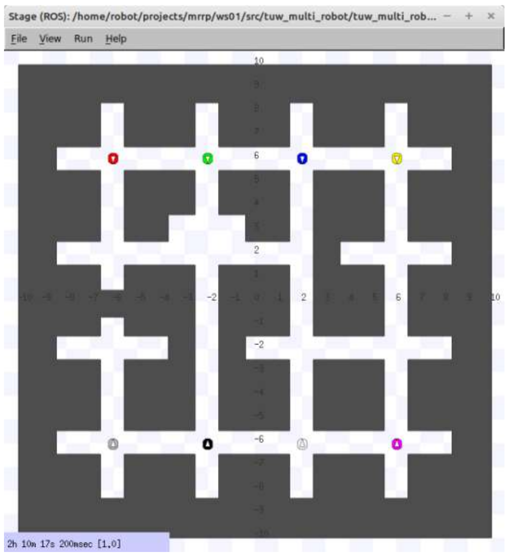


(a) Avg. number of nodes added by MRRP



(b) Avg. number of nodes added by M*

Figure 5.12: Avg. number of nodes added by MRRP and M* in warehouse environment


Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 17s 200msec [1.0]
(a) M* Planner

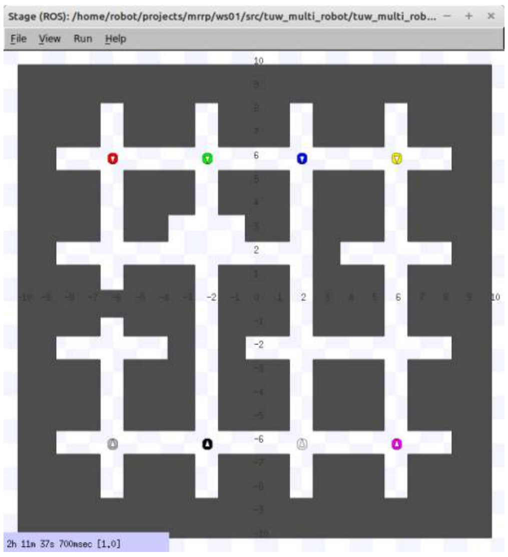
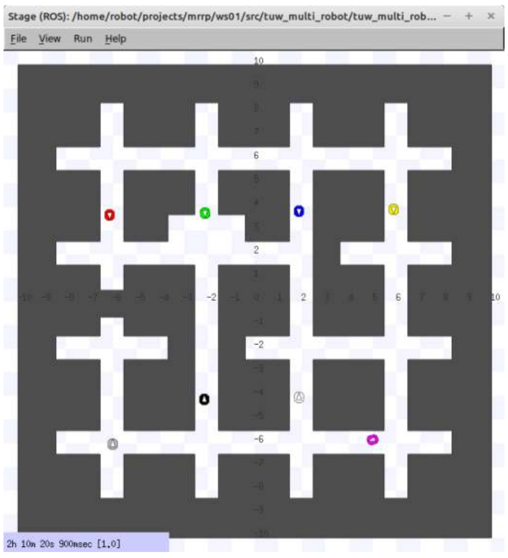

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 11m 37s 700msec [1.0]
(b) MRRP Planner

Figure 5.13: Robot system after 0 seconds


Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 20s 900msec [1.0]
(a) M* Planner

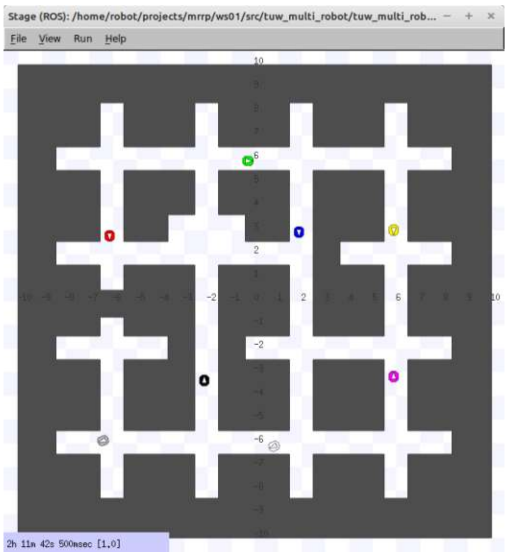
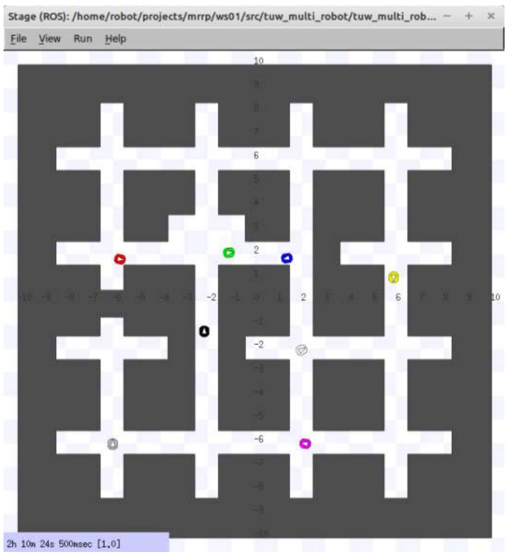
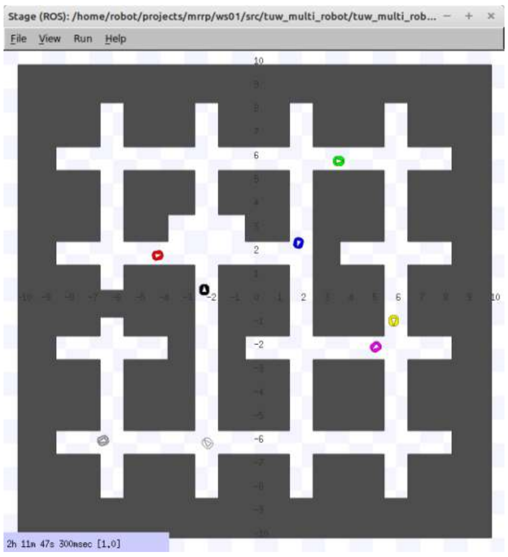

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 11m 42s 500msec [1.0]
(b) MRRP Planner

Figure 5.14: Robot system after 4 seconds

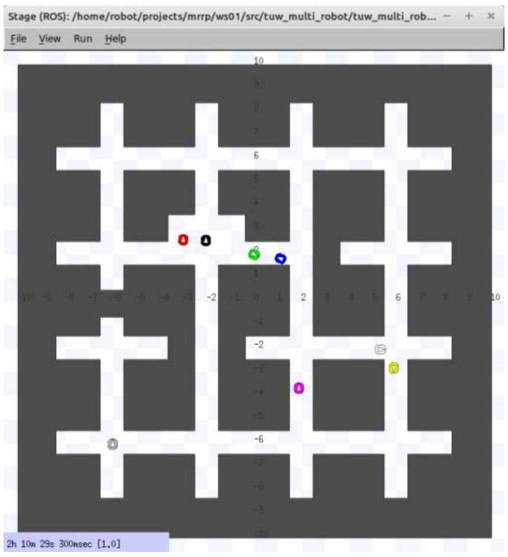

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 24s 500msec [1.0]

(a) M* Planner

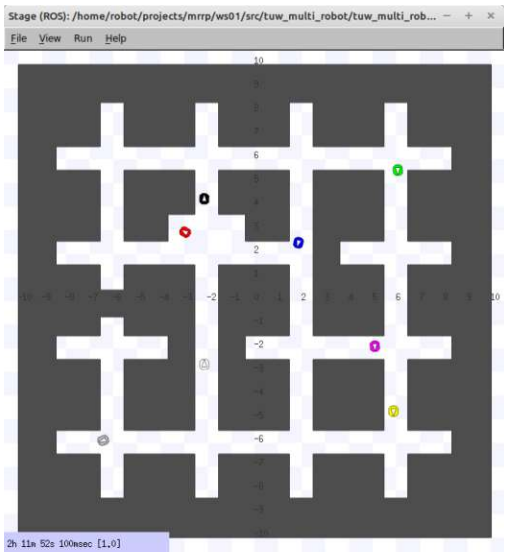

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 11m 47s 300msec [1.0]

(b) MRRP Planner

Figure 5.15: Robot system after 8 seconds

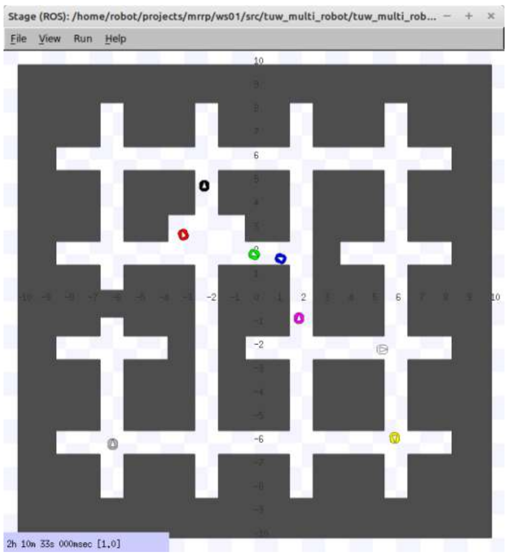

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 29s 300msec [1.0]

(a) M* Planner

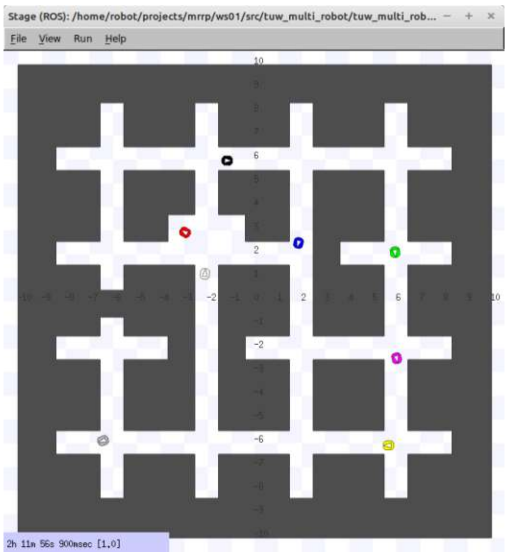

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 11m 52s 100msec [1.0]

(b) MRRP Planner

Figure 5.16: Robot system after 12 seconds

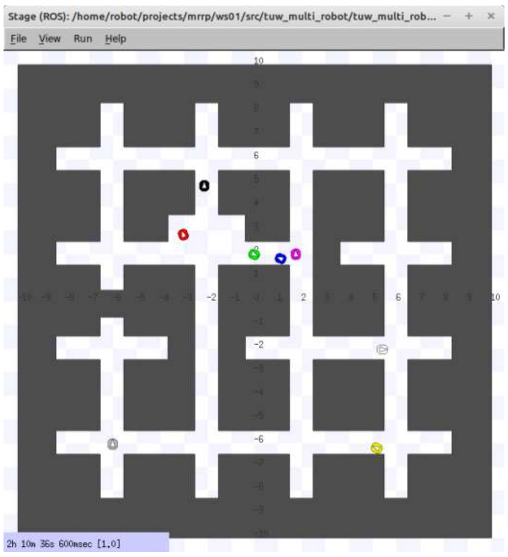

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 33s 000msec [1.0]

(a) M* Planner

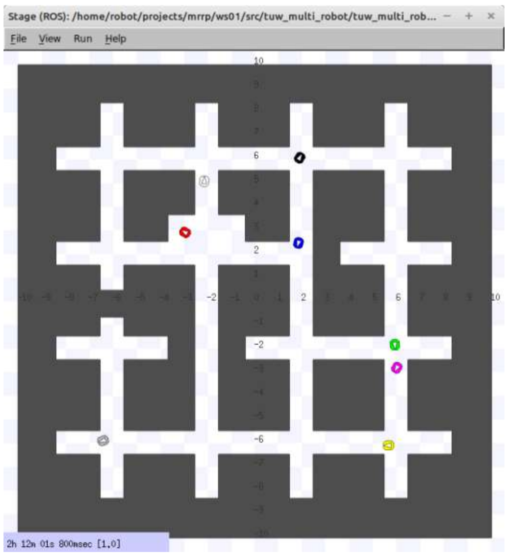

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 11m 56s 900msec [1.0]

(b) MRRP Planner

Figure 5.17: Robot system after 16 seconds

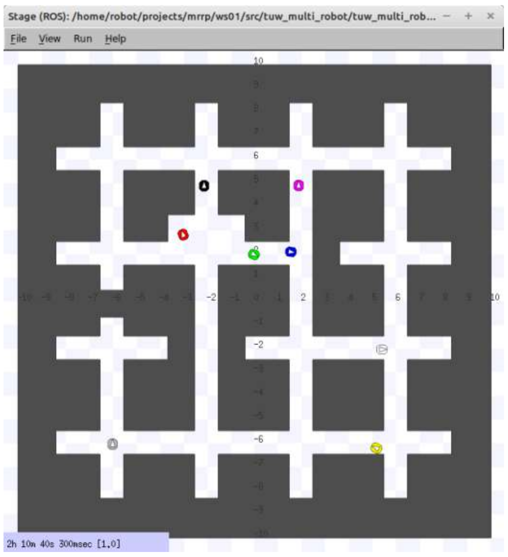

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 36s 600msec [1.0]

(a) M* Planner

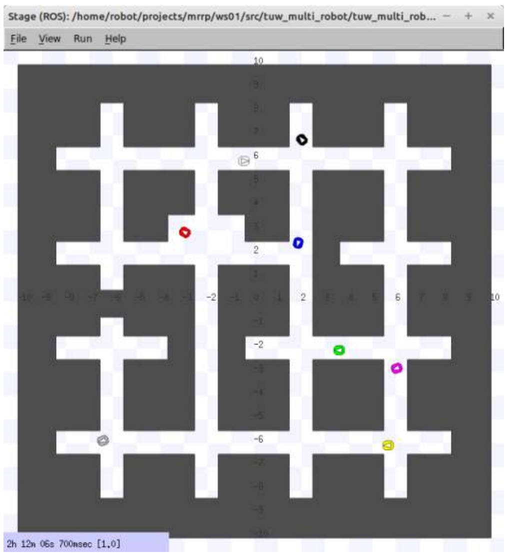

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 01s 800msec [1.0]

(b) MRRP Planner

Figure 5.18: Robot system after 29 seconds

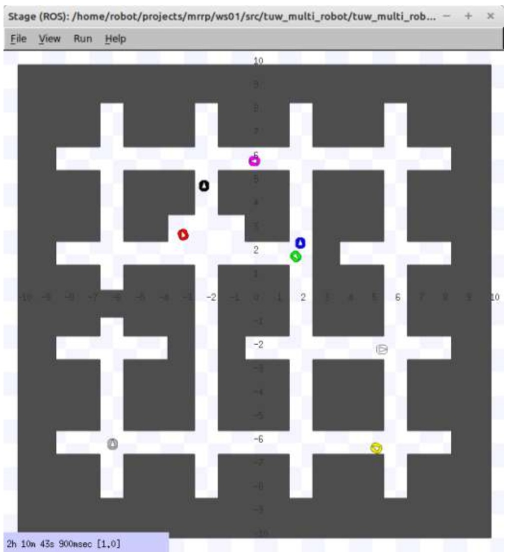

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 40s 300msec [1.0]

(a) M* Planner

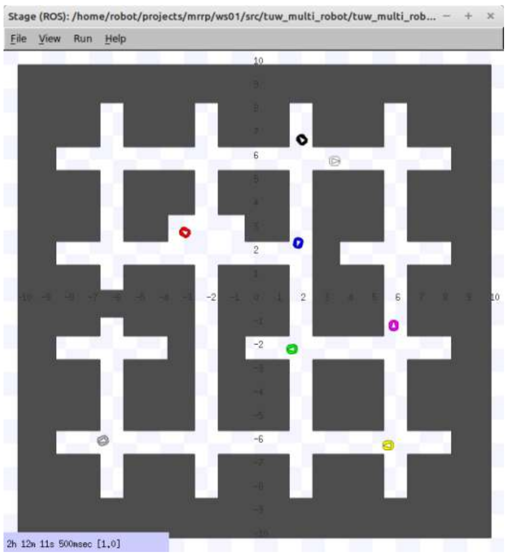

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 06s 700msec [1.0]

(b) MRRP Planner

Figure 5.19: Robot system after 24 seconds

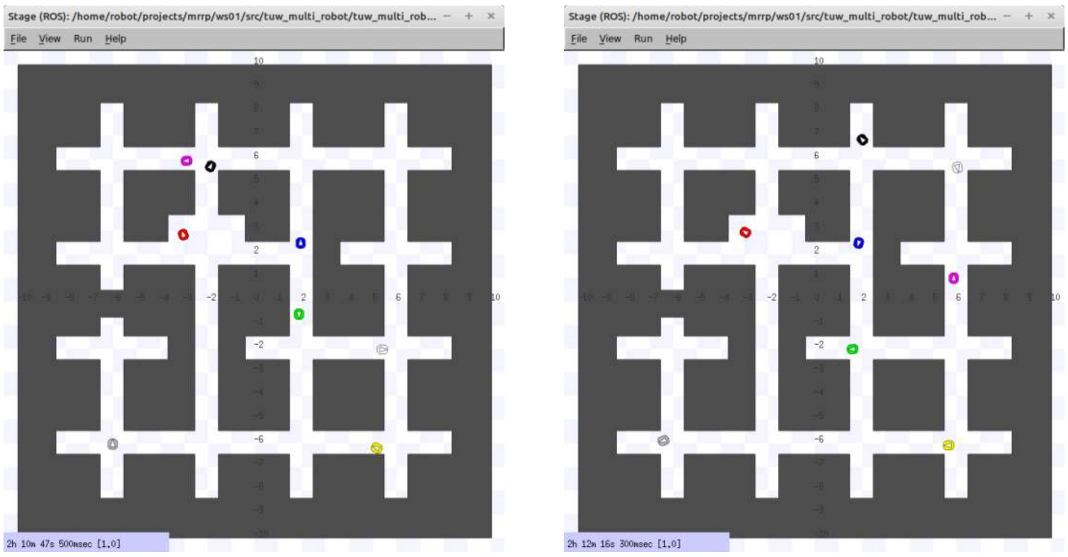

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 43s 900msec [1.0]

(a) M* Planner


Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 11s 500msec [1.0]

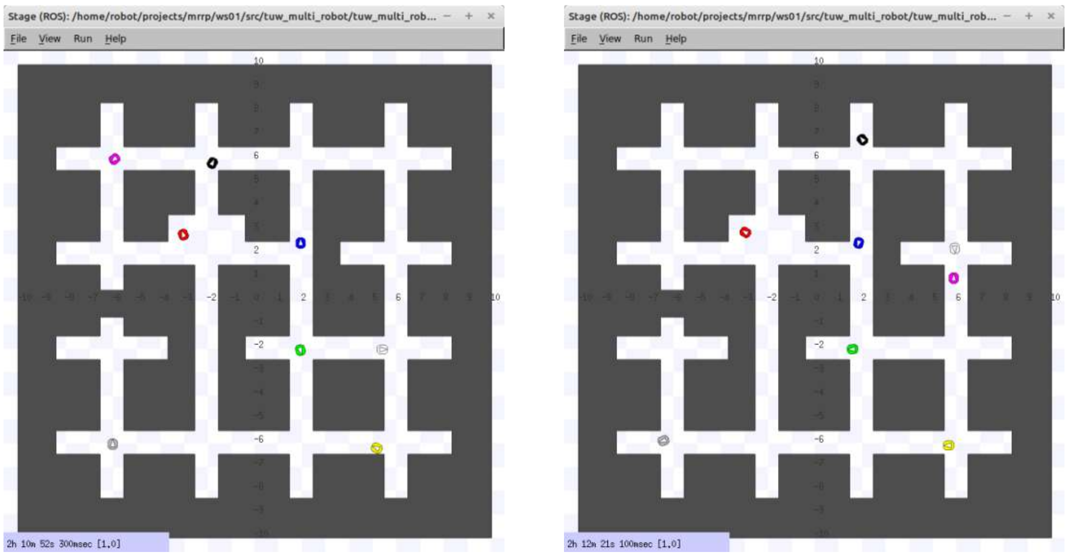
(b) MRRP Planner

Figure 5.20: Robot system after 28 seconds


Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 47s 500msec [1.0]
Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 16s 300msec [1.0]

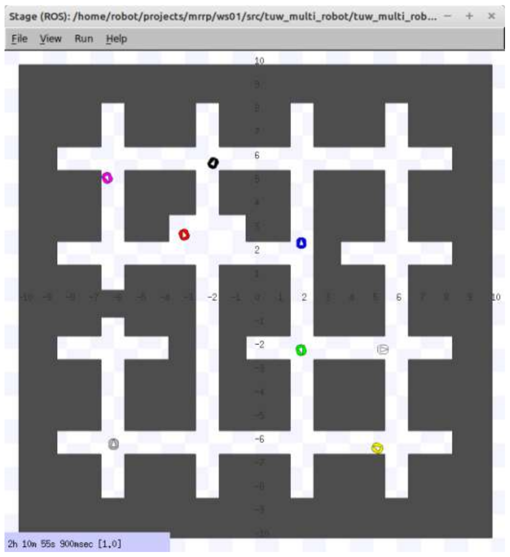
(a) M* Planner (b) MRRP Planner

Figure 5.21: Robot system after 32 seconds

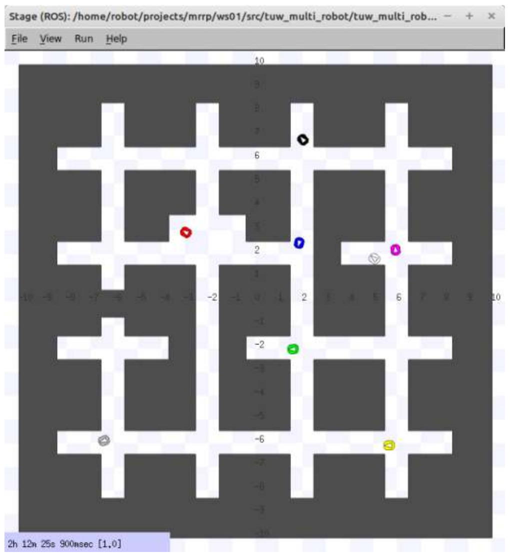

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 52s 300msec [1.0]
Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 21s 100msec [1.0]

(a) M* Planner (b) MRRP Planner

Figure 5.22: Robot system after 36 seconds

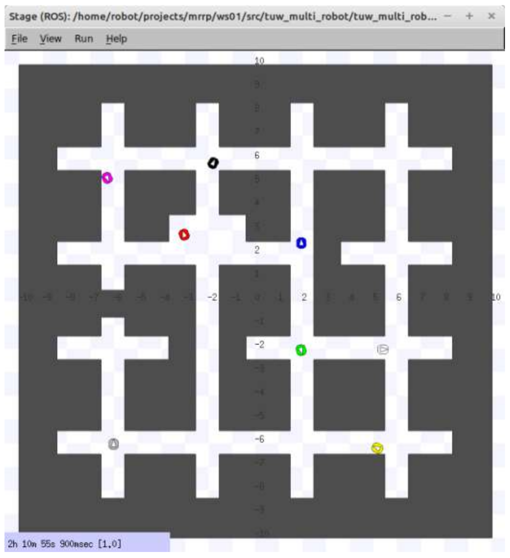

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 55s 900msec [1.0]

(a) M* Planner

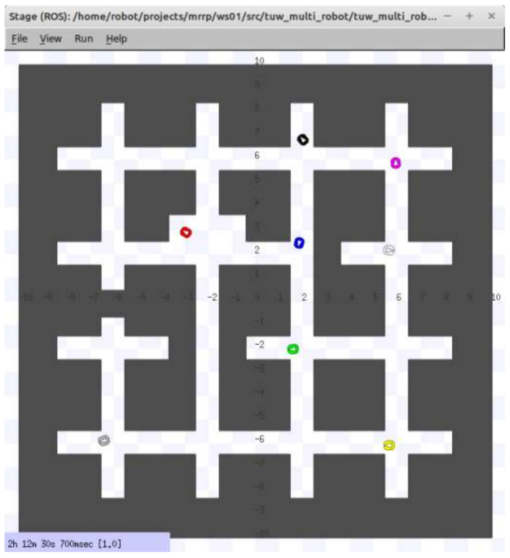

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 25s 900msec [1.0]

(b) MRRP Planner

Figure 5.23: Robot system after 40 seconds

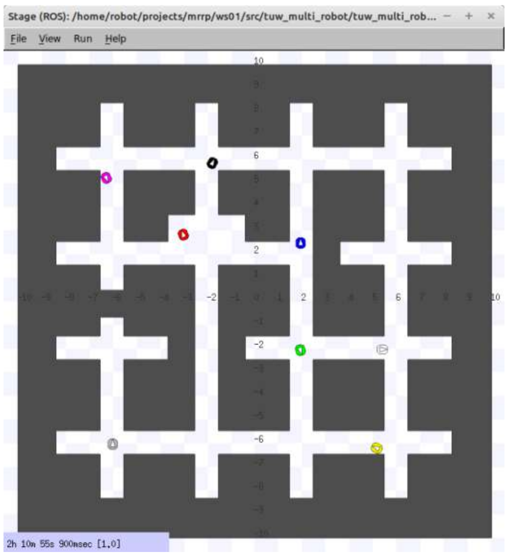

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 55s 900msec [1.0]

(a) M* Planner

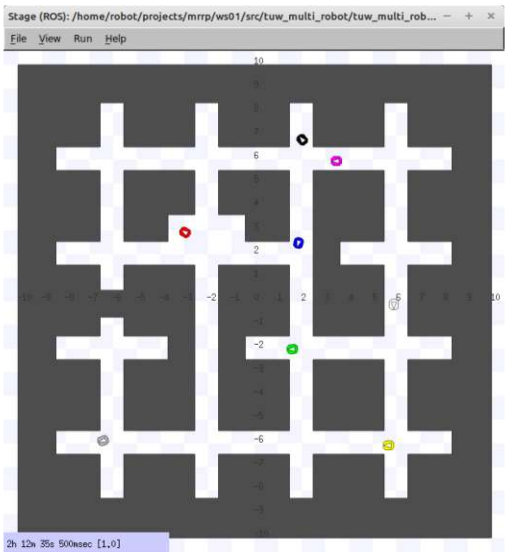

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 30s 700msec [1.0]

(b) MRRP Planner

Figure 5.24: Robot system after 44 seconds

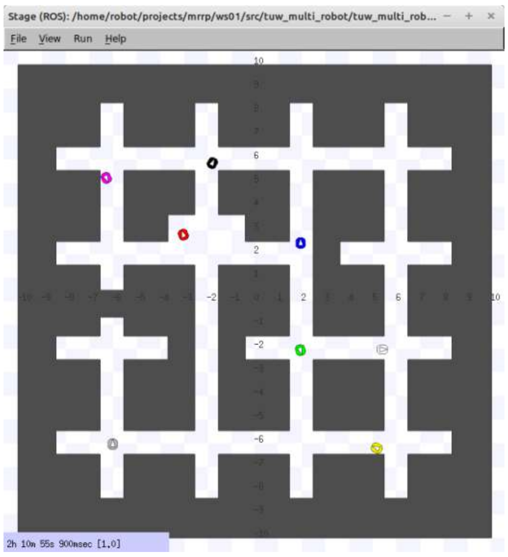

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 55s 900msec [1.0]

(a) M* Planner

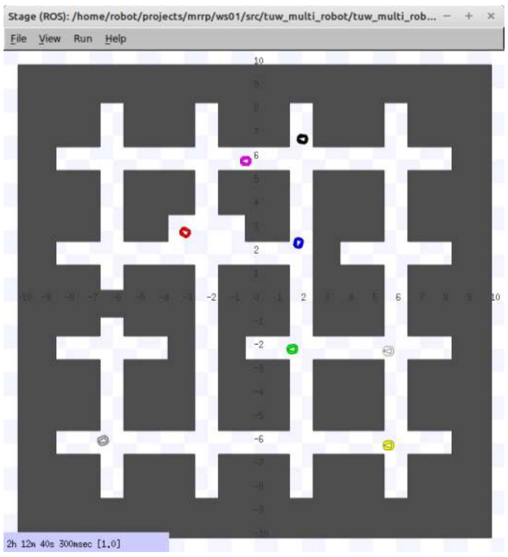

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 35s 500msec [1.0]

(b) MRRP Planner

Figure 5.25: Robot system after 48 seconds

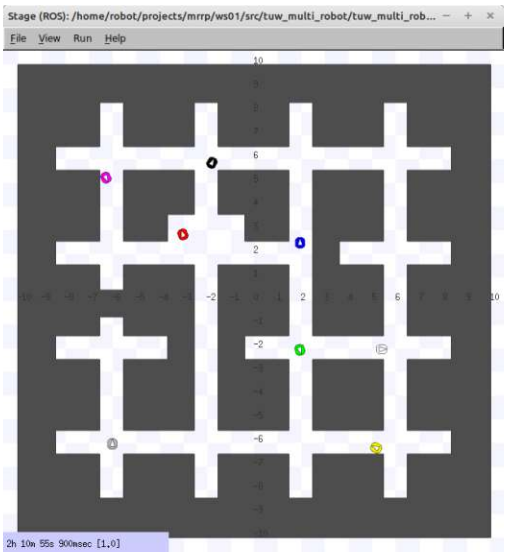

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 55s 900msec [1.0]

(a) M* Planner

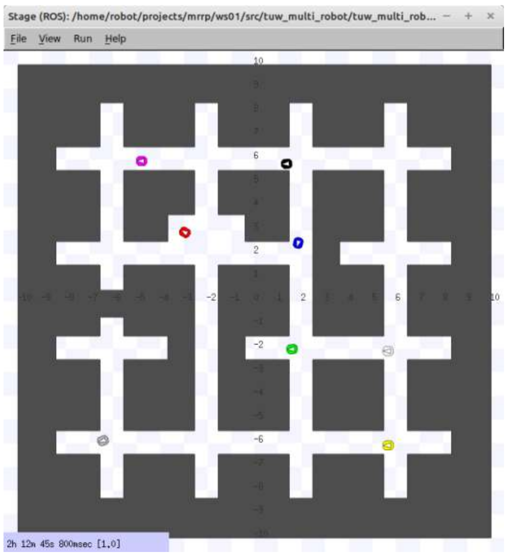

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 40s 300msec [1.0]

(b) MRRP Planner

Figure 5.26: Robot system after 52 seconds

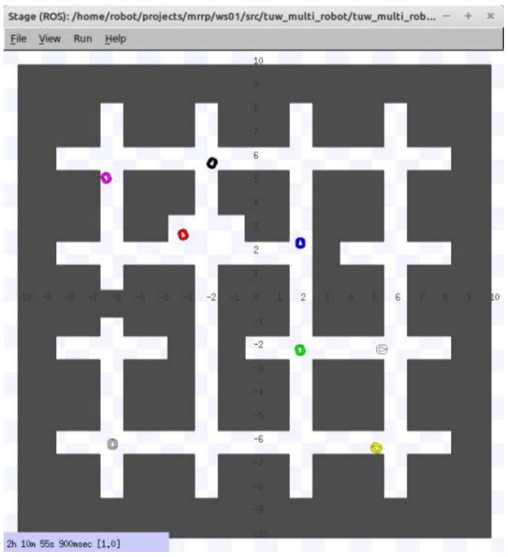

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 55s 900msec [1.0]

(a) M* Planner

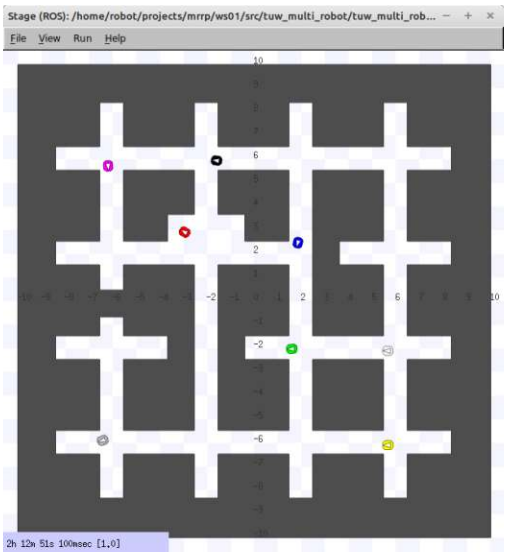

Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 45s 800msec [1.0]

(b) MRRP Planner

Figure 5.27: Robot system after 56 seconds


Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 10m 55s 900msec [1.0]

(a) M* Planner


Stage (ROS): /home/robot/projects/mrrp/ws01/src/tuw_multi_robot/tuw_multi_rob...
File View Run Help
2h 12m 51s 100msec [1.0]

(b) MRRP Planner

Figure 5.28: Robot system after 60 seconds

CHAPTER 6

Discussion

This chapter discusses the results presented in Chapter 5. Section 6.1 will analyse and discuss execution time and the stability benchmark of executed plans. In Section 6.2, the performance of the M* planning algorithm and the MRRP planner for solving the given scenarios will be analysed and compared to each other. An analysis to determine the real-world capabilities of the M* algorithm in terms of runtime and scalability in Section 6.3 will be given in the third section. Finally, the discussion in Section 6.4 answers the research questions formulated in Chapter 3.

6.1 Analysing execution time and route stability

Both planner M* and the CBS planner optimise their computed paths for execution time in the first place. In order to compare both algorithms with each other, only solutions where both planners have successfully computed a valid routing table are considered. The number of solutions used ranges from 35 for a fleet size of four robots to 10 for a fleet size of 16.

As seen in Figures 5.1, 5.2, and 5.3 the M* algorithm can outperform MRRP in almost every case. As stated in the previous chapter, M* computes paths that have between 7.1% and 18%, on average, less time spent on executing the plan compared to plans computed using the CBS approach. M* is a complete and optimal path planning algorithm that searches the whole configuration space when computing a solution to a given problem instance. Therefore, the execution time of plans computed using, the M* algorithm is lower than for paths planned using the implemented CBS algorithm.

The route stability indicates how many of the overall nodes in the path might have potential conflicts if the timing uncertainties of robots are included during the simulation. Figures 5.4, 5.5 and 5.6 show the average stability of each robot in the predefined warehouse environment. It can be seen that the M* planner produces much fewer potential conflict nodes in the CBS algorithm. A conflict node is a node in the environment graph where robots might collide if certain timing uncertainties are met. The low number of conflicting nodes might occur because the M* algorithm computes optimal paths, resulting in paths different from in planned paths using a CBS algorithm.

Overall, it be seen that problem instances are solved using the M* algorithm outperforms those computed by the implemented CBS algorithm.

6.2 Analysing performance by scenarios

In multi-robot applications, certain sub-problems of path planning occur repeatedly, regardless of the environment. A selection of these has been presented in Section 3; solving these scenarios efficiently should result in a performance gain. Therefore, the M* algorithm has been applied to all of these scenarios in order to evaluate its performance. The baseline is the CBS planner. Although the start and goal positions do not change for each run, a whole simulation consists of 50 runs in order to eliminate biases of scheduling threads on the CPU, message delay, and many more system-specific behaviours.

Figure 5.7 shows the average time to solution for the M* and CBS algorithms. It can be seen that the planning time of M* exceeds the planning time of the baseline by up to 5.8 times (Scenario 5/Sub-Figure 3.1e). Although M* needs around six times more time to solve the given problem instance, this is sufficient for real-world multi-robot applications because the required time is between 11.24 milliseconds and 32.36 milliseconds.

In contrast to the baseline, the M* algorithm added for nearly every scenario ten or more nodes on average, especially for the last scenario (Scenario 3.1f), M* added on average on 66 nodes. Figure 5.8 shows how many nodes are added on average for each scenario and planning strategy.

Overall, the MRRP planner and the M* planner have similar performance regarding computation time and expanding nodes for the given scenarios.

6.3 Analysing M* performance in real-world applications

As shown in Figure 5.10, neither the M* algorithm nor the CBS approach can solve all given instances for any number of robots. There are multiple reasons for this behaviour, which are different for both path planning algorithms. The first is MRRP, which implements a CBS path planning algorithm with different strategies to resolve collisions. For example, the possibility of rescheduling the robot priorities or setting the robot speed limit depending on the situation.

Nevertheless, the planner only considers part of the configuration space because it computes the robot's path in a decentralised manner. That is why only some possible solutions are considered and the planner does not always find a solution to the given problem. However, the success rate of the MRRP is always over 65%, and this algorithm was able to solve all path planning problems for a robot system of a size of 16 robots.

In contrast to the MRRP, the M* path planner searches locally through the whole configuration space to find a solution and slowly increments the search space when it can not find a solution. According to this, M* can always solve any given problem, but it takes time. For simulations of the real-world application, the upper bound is set to 60 minutes. If the planner has not found a solution until the timeout, the planning procedure will be cancelled and marked as a failure. This results in a low success rate of 34% in the case of a fleet size of 16 robots.

In order to evaluate the real-world capabilities of the M* algorithm and compare it against the MRRP, the duration of path planning has been recorded and can be seen in Figure 5.11. As stated above, the most significant characteristic is the difference in the measurement units. While the duration of the MRRP is measured in milliseconds, the M* takes at least seconds to compute a solution.

Despite different measurement units, the MRRP has a similar average time to solution for all runs and successful runs, which means the planner exists even in a case of unsuccessful planning in a shorter time than the M* algorithm. Furthermore, it can be seen that the amount of time needed for successfully computing a path using the M* algorithm drastically increases when the number of robots is doubled. In contrast, the planning duration for the MRRP increases steadily when the number of vehicles is doubled.

Additionally to planning time and successfully solved instances, the number of added nodes has been recorded for each path planner and presented in Figure 5.12. Nodes are counted as added if they are inserted after the individual planning phase of the MRRP or after policy calculation of the M* planner. This also includes nodes that might be in the original computed path but added again to create a node for a robot to dodge to avoid a collision.

The Figure shows that, on average, the M* path planning algorithm adds more nodes than the MRRP. This results from the fact that the M* algorithm expands nodes by searching locally through the whole configuration space, while the MRRP does not. So, if a collision occurs using the M* planning algorithm, the number of nodes added per collision is much higher than if MRRP has been solving the problem instance.

6.4 Discussing results regarding research questions

The first research question aims towards different parameters that qualify a solution for a multi-robot planner in different ways.

RQ1: What are quality measurements of multi-robot planner results?

In order to answer **RQ1**, a literature review has been conducted. In general, most path planners optimise plan execution time in the first place. Depending on the use case, other parameters such as fuel/power consumption might be of interest to be optimised instead of the execution time. However, in commonly used path planning algorithms, any kind of uncertainty is not considered during the planning or execution phase. Because considering these would result in a more complex problem, but the computed paths would have a higher resistance compared to those that do not consider uncertainties.

The second research question aims to evaluate the performance of the M* algorithm in common multi-robot environment scenarios.

RQ2: How does the M* perform in contrast to the currently implemented conflict-based search in the domain of multi-robot applications?

When planning a path for real-world multi-robot applications some scenarios occur repeating, making them interesting to solve to solve the whole path planning problem efficiently. In order to answer **RQ2**, the scenarios of this paper [4] have been chosen to be most relevant to solve. The M* algorithm has been compared to the MRRP framework of the same paper. As discussed in Section 6.2, there are no major differences between both implementations, besides the fact that M* will add many more nodes to the graphs to solve the problem instance. These results show the capability to use the M* algorithm to solve specific problem instances on a Voronoi diagram.

The third research question aims to evaluate the M* algorithm in real-world applications and to compare it against the MRRP.

RQ3: To what extent is the M* algorithm applicable to real-world multi-robot applications in terms of runtime and scalability?

Almost every multi-robot application, such as search and rescue, surveillance, warehouse environments, or container terminals, consists of many robots to optimise, for example, travelled distances. Therefore, special algorithms are designed, optimised, and tested against each other because this problem has been proven to NP-Hard [3]. In order to answer **RQ3**, multiple simulations have been performed, and the results presented in the previous Chapter 5 and Section 6.3. In contrast to the original paper, the M* algorithm performed path planning on a Voronoi diagram and not a four-connected graph. The algorithm has been implemented in an existing framework and compared to a CBS approach. It can be seen that M*, although its characteristics to be optimal and complete, does not perform well in real-world applications and is outperformed by the MRRP in terms of success rate as well as planning time.

This thesis compared the M* planner and the MRRP in multiple categories. The first category is time to solution, where MRRP outperforms the M* algorithm in real-world applications, especially if there are more than 32 robots in the fleet. When comparing the size of multi-robot fleets, both algorithms can handle it, but the MRRP is outperforming the M* algorithm again. Because the MRRP can successfully compute paths for 100 robots in the system. When adding nodes to the graph, the path planning algorithm allocates memory to store additional information; this relates to memory efficiency. If a collision occurs, the M* searches the whole configuration space. In contrast, the MRRP resolves the collision by rescheduling priorities and speed and letting agents wait on specific nodes in the graph. Both approaches generate valid solutions, but solutions of MRRP add fewer nodes. As stated in the previous section, both planning approaches can solve all scenarios. The Execution time is the time it takes from the first move by a robot until the last robot reaches its goal. Plans computed by the M* need less time to finish the execution compared to the MRRP. The characteristics of M* ensure that the algorithm computes optimal paths for each robot, which results in a lower execution time. Another criterion that benefits from this characteristic is Plan Robustness; the M* computes plans that are more robust to timing uncertainties than the generated routes of the MRRP.

	M*	MRRP
Time to solution	−−	++
Multi-robot fleet size	−	++
Number nodes added	−−	++
Solved Scenarios	++	++
Execution time	++	−
Plan Robustness	++	−

Table 6.1: Comparison between M* and MRRP

Due to the comparison, it can be seen that the MRRP is more suitable for real-world applications because it can handle bigger robot fleets and computes solutions for problem instances in a shorter time. However, the M* algorithm can generate more robust plans with a shorter execution time; this makes it more suitable as a baseline for benchmarks of other multi-robot path planning algorithms.

Table 6.1 summarises the findings of the algorithm comparison between the M* algorithm and the MRRP where ++ stands for a very good performance, + for a good performance, – for an okay performance and – – for a not good performance in the given category.

The next Chapter will conclude this thesis and present an outlook for future work.

CHAPTER 7

Conclusion

This thesis aimed to compare the implemented multi-robot route planner of the framework presented in [4] to the M* algorithm. Additionally, the capabilities of the M* algorithm should be tested for real-world applications such as warehouse scenarios and special cases. Furthermore, the computed plans of the different planners should be checked against quality measurements and compared to each other.

In order to evaluate the M* algorithm for real-world applications, it has been implemented into the existing framework, and several adaptions to the frameworks have been made. Additionally, a Route Stability-Node has been implemented to evaluate the quality parameters of the plan computed by the used planner.

As the results in Chapter 5 show, the M* algorithm and the MRRP have a similar performance when solving the presented instances, which reoccur in many multi-robot applications. However, when trying to solve problem instances that simulate real-world applications, the performance of M* drops drastically compared to the performance of the CBS-based approach. In contrast to the original paper [1], the M* algorithm uses a Voronoi diagram to compute paths for the robot fleet instead of a four-connected grid graph.

In order to measure quality parameters, the ROS-Node is implemented and works alongside the multi-robot path planner. The results show that although the M* algorithm needs much longer to find a solution, its computed routing table consists of more stable paths in contrast to the plan generated by the MRRP. Additionally, the execution time of a plan computed using M* is less than the time needed if the plan had been computed using a CBS-based approach.

Future work could integrate the Route Stability-Node directly into the path planner to check the stability of the plan before and during the execution. When integrating the node in this form, triggering an online or offline replanning process would also be possible, with additional security distance. For example, marking always one vertex ahead in the travel direction of the mobile robot as occupied. This would ensure additional safety distance for timing uncertainties during the plan execution.

List of Figures

Acronyms

CBS	Conflict-based Search. 2, 3, 18, 19, 21, 22, 32, 51, 52, 55, 57
MAPF	Multi-Agent Path Planning. 2, 8, 19
MAPF-DP	Multi-Agent Path Planning with delay probabilities. 19
MRRP	Multi-Robot Route Planner. 2, 3, 15, 31, 32, 34, 36–49, 51–57, 59
mTSP	multiple Travelling Salesperson Problem. 14
QoS	Quality of Service. 7
ROS	Robot Operating System. 2, 6, 7, 22, 26, 28, 31, 57, 59
TCP	Transmission Control Protocol. 7
UDP	User Datagram Protocol. 7

Bibliography

[1] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3260–3267, 2011.

[2] S. M. LaValle, *Planning Algorithms.* Cambridge University Press, 2006.

[3] J. Yu and S. LaValle, "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, pp. 1443–1449, June 2013.

[4] B. Binder, F. Beck, F. König, and M. Bader, "Multi Robot Route Planning (MRRP): Extended Spatial-Temporal Prioritized Planning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4133–4139, 2019.

[5] I. Open Source Robotics Foundation, "ROS - Official Homepage." `https://www.ros.org/`. [Online; accessed 11-April-2023].

[6] K. Okumura, F. Bonnet, Y. Tamura, and X. Défago, "Offline Time-Independent Multiagent Path Planning," *IEEE Transactions on Robotics*, vol. 39, no. 4, pp. 2720–2737, 2023.

[7] G. Wagner and H. Choset, "Path Planning for Multiple Agents under Uncertainty," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, pp. 577–585, June 2017.

[8] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "Weak, strong, and strong cyclic planning via symbolic model checking," *Artificial Intelligence*, vol. 147, no. 1, pp. 35–84, 2003.

[9] T. Shahar, S. Shekhar, D. Atzmon, A. Saffidine, B. Juba, and R. Stern, "Safe Multi-Agent Pathfinding with Time Uncertainty," *J. Artif. Int. Res.*, vol. 70, p. 923954, may 2021.

[10] N.-B. project, "MAPF - Official Homepage." `hhttp://mapf.info/`. [Online; accessed 10-April-2024].

[11] Z. Ren, S. Rathinam, and H. Choset, "MS*: A New Exact Algorithm for Multi-agent Simultaneous Multi-goal Sequencing and Path Finding," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11560–11565, 2021.

[12] Z. Ren, S. Rathinam, and H. Choset, "Subdimensional Expansion for Multi-Objective Multi-Agent Path Finding," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7153–7160, 2021.

[13] L. Virmani, Z. Ren, S. Rathinam, and H. Choset, "Subdimensional Expansion Using Attention-Based Learning For Multi-Agent Path Finding," 2021.

[14] H. Ma, T. K. S. Kumar, and S. Koenig, "Multi-Agent Path Finding with Delay Probabilities," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, Feb. 2017.

[15] 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, p. eabm6074, 2022.

[16] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, and E. Gunawan, "A Systematic Literature Review of A* Pathfinding," *Procedia Computer Science*, vol. 179, pp. 507–514, 2021. 5th International Conference on Computer Science and Computational Intelligence 2020.

[17] T. Standley, "Finding Optimal Solutions to Cooperative Pathfinding Problems," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, pp. 173–178, Jul. 2010.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[19] J. Banfi, A. Messing, C. Kroninger, E. Stump, S. Hutchinson, and N. Roy, "Hierarchical Planning for Heterogeneous Multi-Robot Routing Problems via Learned Subteam Performance," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4464–4471, 2022.

[20] X. Chen, Y. Li, and L. Liu, "A Coordinated Path Planning Algorithm for Multi-Robot in Intelligent Warehouse," in *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 2945–2950, 2019.

[21] R. Shome, "Roadmaps for Robot Motion Planning with Groups of Robots," *Current Robotics Reports*, vol. 2, pp. 85–94, March 2021.

[22] Q. Wang, M. Wulfmeier, and B. Wagner, "Voronoi-Based Heuristic for Nonholonomic Search-Based Path Planning," in *Intelligent Autonomous Systems 13* (E. Menegatti, N. Michael, K. Berns, and H. Yamaguchi, eds.), (Cham), pp. 445–458, Springer International Publishing, 2016.

[23] X. Chen, G.-y. Li, and X.-m. Chen, “Path planning and cooperative control for multiple UAVs based on consistency theory and Voronoi diagram,” in *2017 29th Chinese Control And Decision Conference (CCDC)*, pp. 881–886, 2017.

[24] Q. Wang, M. Langerwisch, and B. Wagner, “Wide Range Global Path Planning for a Large Number of Networked Mobile Robots based on Generalized Voronoi Diagrams,” *IFAC Proceedings Volumes*, vol. 46, no. 29, pp. 107–112, 2013.

[25] T. Y. Zhang and C. Y. Suen, “A Fast Parallel Algorithm for Thinning Digital Patterns,” *Commun. ACM*, vol. 27, p. 236239, March 1984.

[26] A. Nedzved, S. V. Ablameyko, and S. Uchida, “Gray-scale thinning by using a pseudo-distance map,” *18th International Conference on Pattern Recognition (ICPR’06)*, vol. 2, pp. 239–242, 2006.