FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO


# Multi-AGV Coordination for Heterogeneous Robots

**Diogo Pereira**

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Pedro Costa

Co-Supervisor: José Lima

July 1, 2022

# Resumo

Há uma necessidade crescente de utilização de veículos guiados autónomos (AGV) em ambientes industriais. A capacidade de movimentação e transporte autónomo de itens em ambientes industriais proporciona um aumento significativo na produtividade e eficiência. Esta necessecidade, assim como a possibilidade de controlar grupos de robôs heterogéneos, permite abordar simultaneamente uma vasta gama de tarefas com características diferentes no mesmo ambiente, aumentando ainda mais a produtividade e a eficiência.

Assim, a presente dissertação apresentará uma implementação de um sistema capaz de coordenar uma frota de robôs com robustez às diferentes morfologias e capacidades destes robôs. O sistema implementado deve ser capaz de planear um caminho seguro e eficiente para estes diferentes robôs.

De forma a atingir este objetivo, será implementado um módulo de decomposição de grafos. Este módulo será responsável pela decomposição dos grafos para melhorar o planeamento do caminho dada a heterogeneidade dos robôs, antes de aplicar um algoritmo de pesquisa de grafos. Este sistema utilizará o algoritmo de pesquisa de grafos *Time Enhanced A** (TEA*) para planear os caminhos dos robôs de uma forma segura e eficiente.

Por fim, para além do módulo de decomposição de grafos, será implementado um supervisor para este módulo. Este supervisor será responsável pela verificação da eficiência da decomposição dos grafos. Além disso, o supervisor, ao analisar o desempenho dos robôs no sistema, deve verificar se estes estão a executar os caminhos conforme o planeado.

Este projeto utiliza o simulador *Stage* e várias bibliotecas capazes de simular robôs aproximadamente reais assim como colisões com objetos e estruturas de forma a simular um ambiente industrial real. Este projeto usa também a framework ROS e a ferramenta de visualização RVIZ para ver a informação e a poder analisar de forma a adquirir conclusões.

De uma forma geral os resultados foram bastante satisfatórios. O sistema foi capaz de coordenar uma frota de robôs sem causar colisões ou *deadlocks* e mesmo mantendo uma boa eficiência nos caminhos obtidos e no tempo levado a obter estes caminhos. O algoritmo de decomposição também foi bastante bem sucedido sendo que foi capaz de raramente levar o supervisor a replanear os caminhos devido aos robôs se encontrarem temporalmente desalinhados com o caminho planeado, mesmo com a heterogeneidade dos robôs.

# Abstract

There is an increasing need for automated guided vehicles (AGV) in industrial environments. The capability of autonomous movement and transportation of items in industrial environments provides a significant increase in productivity and efficiency. This need, coupled with the possibility of controlling groups of heterogeneous robots, simultaneously addresses a wide range of tasks with different characteristics in the same environment, further increasing productivity and efficiency.

Thus, the present dissertation will present an implementation of a system capable of coordinating a fleet of robots with robustness to the different morphologies and capabilities of these robots. The implemented system must be able to plan a safe and efficient path for these different robots.

To achieve this task, a graph decomposition module will be implemented. This module will be responsible for the decomposition of graphs to improve the path planning, given the heterogeneity of the robots, before applying a graph-search algorithm. This system will use the Time Enhanced A* (TEA*) graph-search algorithm to plan the robot paths safely and efficiently.

Finally, a supervisor will be implemented in addition to the graph decomposition module. This supervisor will be responsible for verifying the graph decomposition efficiency and if the robot's behaviour is proceeding as expected compared to the planned one.

This project utilizes the Stage simulator and a pre-existing set of libraries capable of simulating the approximately real robot's features and collisions with the objects and structures to simulate a real industrial environment. It also uses the Robot Operating System (ROS) framework and the RVIZ visualization tool to view the data and conclude on the results obtained.

Overall the results were satisfactory. The system could successfully coordinate a fleet of robots without causing collisions or deadlocks while being quite efficient in both the paths obtained and the time taken to plan those paths. The graph decomposition algorithm was also quite successful as it caused the supervisor rarely re-plan paths due to the robots being temporal misaligned with the planned path, even with the heterogeneity of the robots.

# Acknowledgements

I would like to thank both my supervisors Pedro Costa and José Lima for all the help and support given during the realization of this dissertation as well as a special thanks to Diogo Matos for all the hours spent solving dumb mistakes and problems that came up when realizing this project. Without these three people this project would have been much harder to complete.

I would also like to thank my family for the support given throughout the whole process of this dissertation, in special both my parents who always kept me motivated and showed incredible patience when dealing with my lower moments.

I would like to thank my university friends who were always there to keep me company during this semester and helped me with problems whenever it was needed. They also kept me cheered up and motivated whenever it was most needed, which I will never forget, and hope that one day be able to re-tribute the same.

Finally, i would like to thank my hometown friends who kept cheering me up whenever i needed as well as my online friends who kept me company whenever i worked at home and provided me with company and much needed advice.

Diogo Pereira

# Contents

# List of Figures

# List of Tables

# Abreviatures and Symbols

| | |
|---|---|
| AGV | Automated Guided Vehicle |
| $C_{free}$ | Space not occupied by obstacles |
| $C_{occupied}$ | Space occupied by obstacles |
| $C_{space}$ | Configuration space of the environment |
| MRS | Multi Robot System |
| ROS | Robot Operating System |
| SRS | Single Robot System |
| TEA* | Time Enhanced A* |

# Chapter 1

# Introduction

In this chapter the context of the problem and motivations behind the dissertation will be presented, then, a view into the objectives of the work and, finally, a brief view into the document structure.

## 1.1 Motivation and Context

With the advances in the characteristics of autonomous vehicles, such as their autonomy or weight, along with the reduction in their cost, there has been an increase in their usage in industrial environments for a wide range of tasks. Furthermore, using systems made up of a group of vehicles with differences in their brands or morphological structures is becoming more common. The ability to coordinate a multi-robot system (MRS) with heterogeneous robots improves the efficiency of these vehicles in tasks where different vehicles are needed to perform them.

An MRS is composed of multiple robots capable of executing tasks that require movement from one place to another. The coordination between various robots with the same morphology is a complex task that is still being improved today. However, a system capable of controlling a fleet of robots that are not homogeneous adds another layer of complexity to the problem and is increasingly a necessity for today's industry, whether it is controlling robots of different brands or controlling robots that perform entirely different roles in the same system.

Path planning is a crucial aspect required for achieving coordinated and safe navigation of a system composed of multiple vehicles. With this, it is necessary to develop algorithms that can resolve the need for a vehicle/robot to reach a specific location without human interaction and without causing any collisions with obstacles or other robots in the MRS. In addition, reaching the destination is not the only goal. Finding an efficient solution is an essential aspect of these algorithms with crucial factors such as distance travelled, energy spent, or time spent. Because of this, many of the path planning methods are based on various algorithms that, on the one hand, aim to find the set of safe paths to the goal and, on the other hand, aim to link which of these possibilities lead to the most efficient solution [7].

## 1.2 Objectives

The main goal of this work is to implement a system capable of coordinating heterogeneous robots based on the Time Enhanced A* (TEA*) graph search algorithm. This goal will be achieved by implementing a graph decomposition algorithm that will decompose the graph used by the TEA* algorithm before planning the paths and make the system more stable and robust when using a fleet of heterogeneous robots.

The path planning implemented will be applied in a real-time system that already exists, developed by [8] and [9] which already has position detection and controllers to move the robots along specific lines, which is needed for graph-based coordination systems. Additionally, the Robot Operating System (ROS) framework will be used to implement the entirety of the system.

Finally, a real-time supervisor will also be implemented. This supervisor will be used to verify that all robots in the system are following the path that was planned and ensure that they do not pose safety concerns.

## 1.3 Document Structure

This document is constituted by the current introductory chapter and five other chapters. The following chapter, Chapter 2 contains the state of the art. This chapter starts by reviewing the concept of robot systems, where there is a definition of what is a single or an MRS, as well as a distinction between heterogeneous and homogeneous robot groups. Next, control architectures are explored. Here, three types of control architectures are exposed and compared, along with some examples. Then, the concept of path planning is researched, along with different methods, concepts, and examples, such as decomposition graph-based and sampling-based methods. To finalize, a view into other heterogeneous multi-robot coordination systems will be made.

The next chapter, Chapter 3 will present path planning implementation. This chapter will look into the implementation of the path planning algorithm used to coordinate the robots in the MRS, along with some extra features that were added to optimize this step further.

The Chapter 4 will present the system implementation. This chapter will present how the whole system was implemented, the design choices made and some explanations of system modules.

Chapter 5 will showcase the tests done to the implemented system, describe how they were made and the goal of the tests. Then, the results will be exposed and analyzed to assert some conclusions.

The final chapter, Chapter 6 will conclude the document with a review of the goals that were achieved, an analysis of the results obtained and, finally, a brief view into possible future work to improve the project.

# Chapter 2

# State of Art

## 2.1 Robot Systems

Robot systems are a method of automating manufacturing applications by improving time and cost efficiency and lowering labour. Presently, practically every manufacturing business uses these technologies.

While manual labour has dominated production for millennia, the robotic system has completely transformed the industry. Because of robot systems, manufacturers can now produce higher-quality products in a shorter time.

### 2.1.1 Single vs Multi-Robot Systems

Single robot systems (SRS) contain only one individual robot that can model itself, the environment, and their interaction [10]. Although these SRS excel at single-focused tasks, they fall short when some tasks are too inherently complex or impossible for them to perform, such as spatially separate tasks. For example, [11] gives a task that requires the activation of two keys separated by a considerable distance and must be activated simultaneously. This restriction inhibits an SRS from fulfilling the task due to spatial limitations.

The MRS, with homogeneous or heterogeneous robots, solve these limitations, as well as having the following advantages [12]:

- Better spatial distributions.
- Better overall system performance, with metrics such as time required to complete a task [13] or the energy consumption of the robots [14].
- Introduction of robustness that can benefit from the fusion of data and information sharing between the robots in the system, along with fault-tolerance that can benefit from information redundancy. For example, robots can use the position of other robots to localize themselves better [15].

- Better system reliability, flexibility, scalability [16] and versatility. The combination of groups of robots with different abilities habilitates them to complete more complex tasks and allows them to fail without affecting task completion.

### 2.1.2 Heterogeneous vs Homogeneous Robots

In an MRS, the robot groups can be heterogeneous or homogeneous. In a homogeneous robot group, the capabilities of the robots are identical, such as in [8, 17, 18], the overall advantage of a system with homogeneous robots is its inherent low complexity due to the task planning becoming more complex. On the other hand, in a heterogeneous robot group, robots have different capabilities such as different velocities, sizes and even traction types. The complex nature of this type of system comes with benefits regarding the optimization of the completion of tasks, providing greater group utility when compared with a homogeneous group [19].

## 2.2 Control Architectures

An MRS is characterized by having a group of robots capable of executing tasks in a flexible, efficient and fault-tolerant way. With this, there is a vital necessity to promote cooperative behaviour between the agents involved.

The control architecture is a crucial element of an MRS, being responsible for determining the system's capabilities, as well as its limitations [20]. Currently, two main control architectures stand out. These are distinguished by the possibility of granting autonomy to the robots or by having a central unit coordinating them. They can also be joined further to improve the overall control capability. These architectures are termed centralized, decentralized, and hybrid, respectively, and are presented in the following points.

### 2.2.1 Centralized Control Architecture

A centralized control architecture is composed of a central planning and control unit that concentrates the planning of the various strategies of the AGV [21], such as task scheduling, path planning and motion coordination, eliminating the communication between the robots. This means that this central unit is responsible for communicating with each AGV and coordinating them, facilitating the decision-making process, as only one agent holds all the information and controls the movements and decisions of the other agents in the system.

This type of architecture can guarantee optimal control performance, but the upper control system needs a lot of computational resources [22]. It also tends to have poor scalability as the computational resources needed increase exponentially as the number of robots in the system increases [23]. Additionally, due to the nature of the architecture, it also leads to a low tolerance to communication faults [7].

### 2.2.2 Decentralized Control Architecture

The decentralized control architectures, or distributed control architectures as referred to by some authors [22], aim to combat the centralized architectures' limitations by giving autonomous planning capabilities to each robot instead of a centralized system, ensuring more considerable fault tolerance, better scalability, and better capability to respond to dynamic changes in the environment. In this architecture, each robot can make its own decision independently, decide its path and communicate with other robots.

Within this type of architecture, strategies can be divided into two groups: priority control and path coordination. In some cases, it might be beneficial to implement these methods in series to achieve better results.

In priority control strategies, each robot plans its path, considering the paths of the other robots with a higher priority. It is usual also to implement an adaptive priority assignment to the robots to improve efficiency and eliminate cases where the robot cannot calculate a path due to higher priority robots, which can be helpful in high traffic environments.

In path coordination, each robot plans its path independently of the other robots by creating a set of strategies that resolve conflicts when trajectories of two or more robots coincide. These strategies are created by first finding the locations and instances where possible conflicts will occur, which is done by considering the velocity profile of the robots and obtaining temporal-based scheduling of the robot position [24] and then calculating them. After this, collision prevention strategies can be applied. A common strategy is to implement priorities on the crossing points between paths.

One of the major problems of this architecture is the local minima problem. This problem is due to each robot's ability to process locally available information and sensing capabilities [25].

### 2.2.3 Hybrid Control Architecture

The advantages of centralized and distributed architectures are combined with hybrid control architectures. These advantages allow for overcoming the low performance caused by the self-centeredness in the distributed architecture and reduce the lack of control flexibility in the centralized architecture [26].

As an example, [25] uses a hybrid system to combat the local minima problem of the decentralized architecture by dividing the system into a slow centralized component responsible for the path planning of each robot and a fast decentralized component responsible for motion control and routing solutions for each robot.

In another example, [27] uses a hybrid architecture to control unmanned aerial vehicles in a dynamic environment, benefiting from the quick response of decentralized architectures and utilizing the planning advantages of a centralized architecture.

## 2.3 Path Planning

Path planning is one of the most crucial aspects of autonomous movement systems. Path planning algorithms aim to establish paths between start and end points allowing mobile robots to navigate autonomously and safely in a working environment. They generate an optimal solution for a system to change from an initial state to a final state by avoiding static and dynamic obstacles in the environment [7].

Before presenting path planning-based methods, it is important to make a distinction between the following terms:

- Path Planning: this term describes, in geometric or mathematical terms, the path from an initial point to a final point avoiding obstacles [1]. This focuses on providing raw data and sometimes needs complementary methods to generate an optimal solution.
- Trajectory Planning: this term represents a path (obtained from path planning) as a function of time. In other words, it says the place where the robot should be positioned for each instant of time. Trajectory planning determines where vehicles move and how they should move along that path [7].
- Optimal Path Planning: this term introduces a cost function based on aspects, such as distance travelled, time or energy spent, and tries to find a set of paths which optimizes this cost function [7].

### 2.3.1 Configuration Space

In path planning, it is relevant to define the map of the environment in which the robots are inserted. A configuration space ($C_{space}$) is created to achieve this. The $C_{space}$ is a mathematical tool developed to collect all configurations and positions of a robot [28]. In this space, it is termed free space to any part that is not occupied by obstacles, represented as $C_{free}$. The space occupied by obstacles is called occupied space and represented as $C_{occupied}$. The $C_{occupied}$ can also contain a margin of free space around the obstacles creating a safety gap.

### 2.3.2 Decomposition graph-based methods

Decomposition graph-based methods are based on a grid decomposition of the environment to obtain a representation of the said environment in the form of cells and establish which cells are $C_{free}$ and which cells are $C_{occupied}$. After this distinction of the cells is made, the cells that belong to $C_{free}$ are jointed, through edges, creating a graph.

#### 2.3.2.1 Cell Decomposition

Cell decomposition of the environment is a method of dividing the environment to obtain a cell representation. These cells are then calculated as $C_{free}$ (if they are free) or $C_{occupied}$ (if they have

obstacles) and what neighbouring cells they can connect to, creating a structure known as a graph. The steps to find a path can be simplified as starting and end points are assigned to two cells, and then there is a search for a sequence of cells that go from the starting cell to the end cell.

This method can be divided into the exact and approximate cell decomposition.

### Exact Cell Decomposition

In this method, the $C_{space}$ faithfully represents the real environment, which means cells represent either free or occupied cells. This decomposition method aims to create simple geometry cells (usually convex polygons or trapezoids [1]), so it is easier to find the neighbouring cells and a path. As an example, in figure 2.1 it is possible to see a trapezoidal decomposition where cells are obtained by a set of vertical line segments created by the vertices of the obstacles:

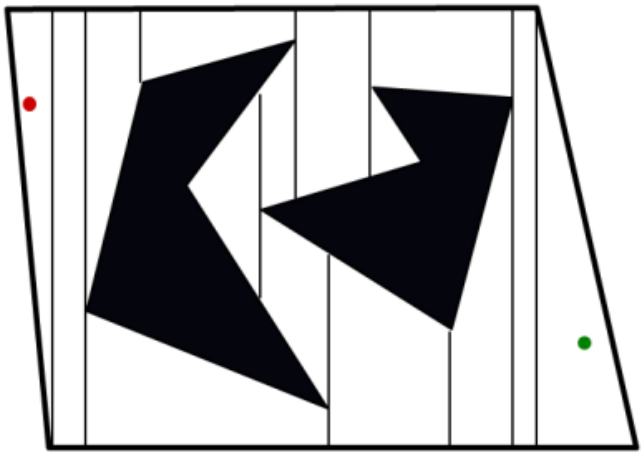


Figure 2.1: Example of a trapezoid decomposition [1]

### Approximated Cell Decomposition

With this approach, there is not an exact representation of the real environment. The cells can have three states: occupied, semi-occupied or free. This decomposition method uses simple geometric figures like squares. With this, the error between the real environment and the decomposition result is directly related to the size of the geometric figure used.

There are two main approaches to approximated cell decomposition. The first is the fixed cell. In this method, the cells take a pre-defined size, dividing the $C_{space}$ and then calculating the state of each cell, as can be seen in figure 2.2.

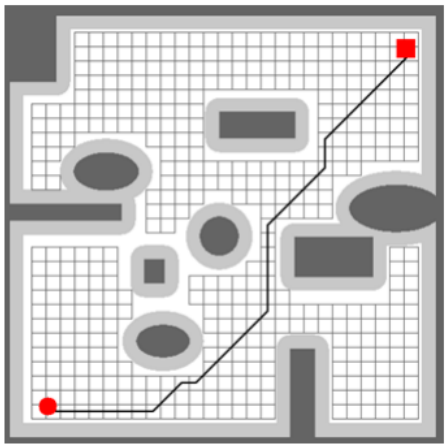


Figure 2.2: Example of a fixed cell decomposition [1]

The second approach is the quadtree approach, where the $C_{space}$ is divided into four cells, and each time a cell does not belong to $C_{free}$, it is subsequently divided into four cells again. This division is repeated recursively until a minimum cell size is met. Figure 2.3 shows an example of this decomposition.

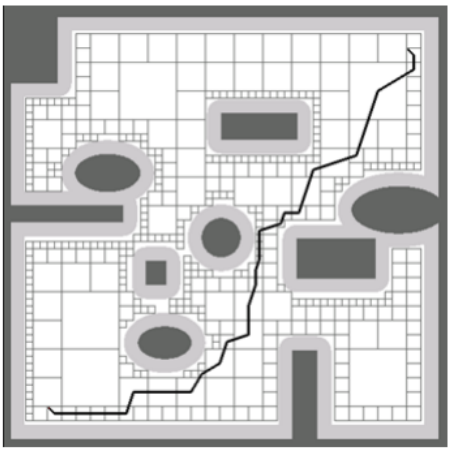


Figure 2.3: Example of a quadtree decomposition [1]

#### 2.3.2.2 Voronoi Diagrams

Voronoi diagrams can be summarized as a set of points that are equidistant from two or more obstacles, dividing the $C_{space}$ into several regions, as seen in figure 2.4, where edges $E_3$ and $E_1$ are equidistant from the points in the segment $S_1$. These regions contain one obstacle, and any point in that obstacle is closer to the obstacle in that region than any other obstacle in the $C_{space}$ [24].

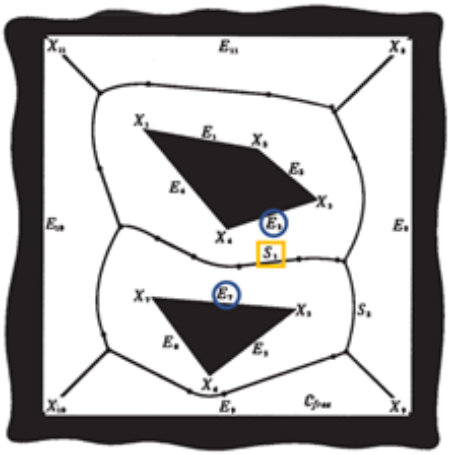


Figure 2.4: Voronoi Diagram example [2]

#### 2.3.2.3 Graph Search Algorithms

As mentioned previously, the present dissertation will focus on the search algorithm Time Enhanced A* (TEA*). This algorithm is based on the A* search algorithm as it complements this base algorithm by adding temporal layers for optimization and safety.

#### A* Algorithm

The A* algorithm is a heuristic search algorithm formulated using weighted graphs. Starting on a start node, the algorithm creates a tree that extends the paths one edge at a time until a specific criterion is met. For each iteration, the algorithm determines which path to extend based on the cost of the path $g(n)$ from the start node to the node $n$ and a heuristic function $h(n)$ that estimates the cost of the cheapest path from node $n$ to the end. Usually, in path planning problems, the heuristic functions are the Euclidean distance or the Manhattan distance, given by equations in 2.2:

$$Euclidean\ Distance: \quad h(x_n, y_n) = \sqrt{(x_n - x_{end})^2 + (y_n - y_{end})^2} \tag{2.1}$$

$$Manhattan\ Distance: \quad h(x_n, y_n) = |x_n - x_{end}| + |y_n - y_{end}| \tag{2.2}$$

Where $(x_n, y_n)$ is the position for current node $n$ and $(x_{end}, y_{end})$ the position for the destination node.

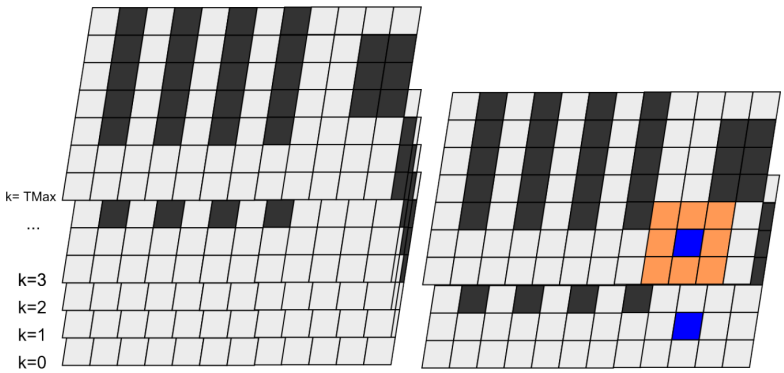
With these two values, A* chooses a path that minimizes $f(n)$ in equation 2.3:

$$f(n) = g(n) + h(n) \tag{2.3}$$

When there are no more paths to be extended, the algorithm finishes.

### TEA* Algorithm

Time Enhanced A* (TEA*) is created from the necessity of an algorithm suitable for multi-AGV systems that avoid collisions and deadlocks and guarantees the efficient execution of tasks [3]. This algorithm considers the positions of the robots and the various dynamic changes in the environment to calculate the most efficient path. TEA* is based on the A* heuristic search method. In this algorithm, in addition to the map's coordinates ($x$ and $y$), a third layer is added to the input map, the temporal layer, as seen in Figure 2.5, that is represented with a $k = [0, T_{max}]$, where $T_{max}$ represents the maximum number of temporal layers. The input map is obtained through cell decomposition of the environment space resulting in a grid representing empty and occupied cells.




(a) Input Map over the time. (b) Analysed Neighbor Cells

Figure 2.5: The input map over the various temporal layers (a) and the analysed neighbour cells focused on the AGV position (b) [3]

With this introduction of temporal layers, the neighbour cells of the current AGV position that will be explored now belong to the next temporal layer, as seen in figure 2.5 (b). Another addition is that the analyzed neighbour cells include the cell containing the AGV's current positions, represented by the blue cell in Figure 2.5 (b) [3].

When applying the TEA* algorithm in a multi-AGV environment, the first step is to convert every AGV's current position to an obstacle, allowing other vehicles to consider these as occupied cells and thus, calculating a collision-free path. To avoid deadlocks, these cells are only considered obstacles in the temporal layers $k = \{0, 1\}$. With this, vehicles can calculate a collision-free path in the instances after these moments.

The next step consists of analyzing the list of the missions of all the robots and calculating a path for each one. For this to work, a priority list must be created beforehand so that the robots calculate their path and define it as a moving obstacle for the other robots with a lower priority. These steps work inside a loop, updating the current AGV's positions and recalculating their paths [3].

### 2.3.3 Sampling Based Methods

Sampling-based methods are based on random mapping of the environment to achieve a path between a starting point and an ending point. This requires a mathematical representation that describes the environment space. This random sampling is carried out in the form of nodes or cells.

For each iteration, a new point is accessed and determined if it belongs in $C_{free}$, which, in this case, is connected with other nearby points, generating connections between the nodes that belong to the $C_{free}$, establishing a structure or graphic of the $C_{free}$ in $C_{space}$ [7].

These methods can be classified as active or passive. Active methods will provide the best path to the target by its procedure. On the other hand, the passive methods generate a network of paths from start to destination, thus being necessary that these are complemented with algorithms to obtain the best path [7].

#### 2.3.3.1 Probabilistic Roadmap

The basic idea of this algorithm is to take random samples from the $C_{space}$ and check if they belong to $C_{free}$. If so, these nodes try to connect with other nodes from $C_{free}$ to form a continuous and obstacle-free connection between them. These configurations and connections are added to the graph until the roadmap is dense enough.

Because this algorithm forms a graph, it is possible to use graph search algorithms to obtain an optimal path from a start node to an end node.

The fact that this is a passive method brings some advantages, such as the algorithm being lower computationally than other active methods and, most importantly, being able to reuse by other robots in the same network [7] and making it a competitive method in the MRS field. Some variants of this primary method vary the sampling strategy, as seen in Figure 2.6, and the connection strategy to achieve better performance [4].

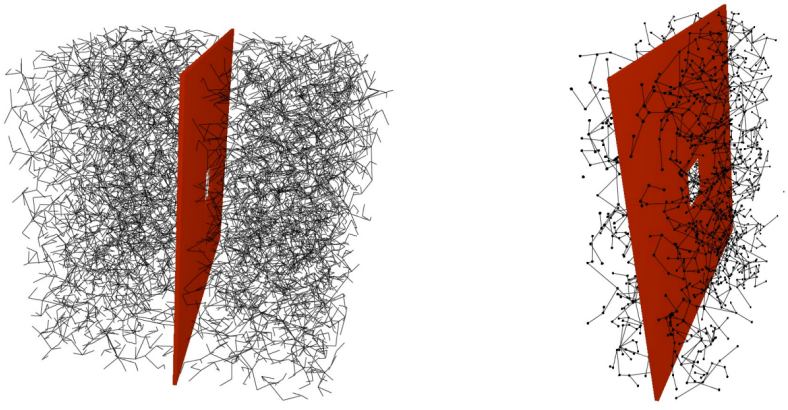


Figure 2.6: Roadmap graph obtained with different samplings [4]

#### 2.3.3.2 Rapidly Exploring Dense Trees

The main idea of this algorithm is to construct a tree incrementally that gradually improves the resolution. In the limit, the tree densely covers the $C_{space}$. In this algorithm, a dense sequence of samples is used to guide the incremental construction of the tree [5].

The algorithm for constructing a tree can be summarized as taking a starting sample; this sample forms a vertex and attempts to connect to the nearest sample; if there are no obstacles, the new sample is added to the tree as a new vertex, and the correspondent edge, as can be seen in Figure 2.7.

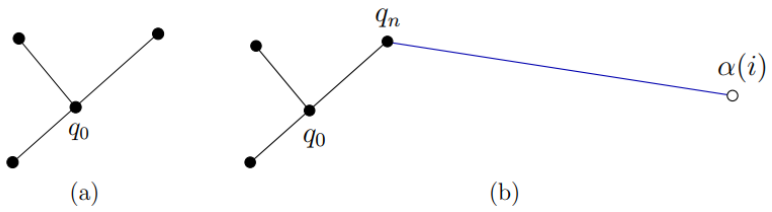



Figure 2.7: Example of an edge being added to the tree that connects the vertex $q_n$ to the sample $\alpha(i)$ [5]

If there is an obstacle, the edge travels up to the obstacle boundary and adds that limited edge to the tree, as demonstrated in figure 2.8.

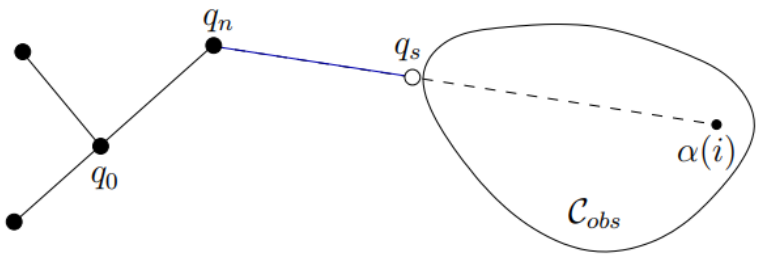



Figure 2.8: Example of an edge being added to the tree with obstacle detection [5]

As such, in each iteration, the branches grow outwards. It is an efficient procedure to explore the free space since when generating branches, the tree is growing and exploring the $C_{space}$ in parallel [7].

This algorithm can be used in a broad class of problems, such as in [29] where it is used to compute trajectories for hovercrafts and satellites in cluttered environments. Some interesting improvements to the algorithm are also shown in [30] where the algorithm integrates a pruning process with geometric collision detection to reduce the number of turning points in the path.

## 2.4 Other Heterogeneous MRS Coordination Methods

Various heterogeneous MRS coordination methods have already been explored in the field of MRS coordination. The methods may vary a lot in terms of approaches. However, the decision-making process, or rather, whether a centralized or a decentralized group architecture is used, seems to be the most significant point of contention. For example, [31] uses a fully distributed method to coordinate the various robots in the system which communicate with each other and use a coordination protocol to coordinate with each other and a monitor connected to the communication network to supervise the behaviour of the robots. A typical MRS that uses a centralized architecture is the GOFER [32] which uses a central task planning and scheduling system by viewing all the tasks that need to be resolved and what robots are available to perform those tasks. Furthermore, a hybrid approach is also possible, but the solutions they reach are often sub-optimal, as seen in the following study about MRS coordination [33]. The algorithm used in this dissertation's project will focus on a more centralized approach due to the nature of the TEA* algorithm.

## 2.5 Summary

In summary, this chapter has presented a few crucial notions for coordinating a multi-AGV environment with heterogeneous robots. Firstly, the definition of robot systems allowed for the distinction between usual homogeneous and heterogeneous systems. Then, the view into the different control architectures presented distinct ways to coordinate robots in an MRS, with some examples implemented in different systems. Finally, some path planning concepts and algorithms were presented that will be further explored during the dissertation phase.

# Chapter 3

# Path Planning Implementation

One of the essential features required for this project is the implementation of a path planning algorithm capable of coordinating all the robots in the system. As such, the TEA* graph search algorithm was chosen as the primary tool to coordinate the paths for all the robots since this tool can be very efficient in this environment. Furthermore, because one of the essential characteristics of a planned path is that it is deadlock and collision-free, the TEA* algorithm is perhaps one of the best options for this task.

The main advantage of the TEA* algorithm comes from introducing a third temporal layer capable of planning a path that does not collide with any other robot in time or reach other blocking problems such as deadlocks. This feature, coupled with the capabilities of obtaining optimal paths from the A* algorithm, the TEA* accomplishes all the necessary requisites for a robust path planning algorithm.

Starting with the creation of a system that can find efficient paths for robots without taking into account robot collisions, the A* algorithm was implemented. Using this A* algorithm, the TEA* algorithm is implemented by adding an extra layer to represent time. Using this layer, it is possible to prevent robot collisions or path-blocking problems.

It is important to note that the TEA* and A* algorithms are graph search algorithms and, as such, need a pre-defined graph to be able to plan paths for the robots. The definition of the graphs used in this project will be presented in chapter 4 where the implementation of the full robot coordination system will be showcased.

## 3.1 A* Implementation

The A* algorithm was implemented to control single robots as a starting point. This algorithm characterizes itself by the possibility of optimal searches done by testing each node and its neighbour nodes and obtaining a sequence of connections between those nodes (links) that minimize a cost function of the sum of nodes that make that sequence of links. These costs depend on the optimizing factor, such as time, distance, or energy. In this dissertation project, initially, distance will be the factor used to optimize the paths for each robot, and then time will be used.

These costs are calculated each time a neighbour node is explored in search of the lowest cost for the path. With this in mind, the total cost each time a neighbour node is explored is equal to the sum of all the costs incurred from the starting node to the current one $g(n)$ and the cost, based on a heuristic function, from the current node to the end node $h(n)$. This total cost $f(n)$ is the sum of the previous costs, as seen in equation 3.1.

$$f(n) = g(n) + h(n) \tag{3.1}$$

All the nodes in the graph have a state in this project's specific implementation, which starts as *VIRGIN*. At the start of a path planning, the node where the robot is located ($C_{robot}$) is set to *CLOSED*, has its costs calculated, and added to a list ("*O*" list). This list is a min-heap organized by the total cost so that the top of that list is always the lowest cost node. Then, in a loop, the lowest cost node ("*C*") from the top of the list is taken, and its neighbour nodes ($C_{neighbour}$) are explored. When exploring these neighbour nodes, the nodes are set to *OPENED*, node costs are calculated, the origin node is set as this node's parent node ($C_{father}$), and the explored node is thrown into the list. Whenever a node is taken from the list, it is set as *CLOSED*, so that the same node is not re-explored. This whole process is repeated until the node taken from the list is the node that corresponds to the destination node ($C_{destination}$), and the path is obtained by running through the parent nodes of each node recursively until the robot's original position is obtained. A path is obtained using this sequence of nodes and can be sent to the robot for it to follow. This algorithm is perhaps better explained in the pseudo-code presented in algorithm 1.

After the path of a robot is obtained, that path is sent to a robot. The A* algorithm is beneficial as it can find very efficient paths for single robots. However, in a multi-robot scenario, the totality of a robot's path would need to be set as an obstacle for another robot for it to be somewhat safe, leading to inefficient paths being obtained for lower priority robots and a higher probability of no possible paths being found at all. To solve this problem, the TEA* algorithm is implemented.

**Algorithm 1:** A* algorithm implemented

**Inputs :** $C_{robot}$ and $C_{destination}$
**Outputs:** $path$
Min-heap "$O$" is created;
$state(C_{robot}) = CLOSED$;
$f(C_{robot}) = h(C_{robot})$;
Insert $C_{robot}$ into $O$;
**while** $O \neq \emptyset$ *or* $i \leq Max_{iter}$ **do**
  Take lowest cost node $C$ from $O$;
  $state(C) = CLOSED$;
  **if** $C_{neighbour} = C_{destination}$ **then**
    Create vector $path$;
    **while** $C_{father}(C) \neq C_{robot}$ **do**
      add $C_{father}(C)$ to $path$;
      $C = C_{father}$;
    **end**
    return path;
  **end**
  **for** $C_{neighbour}$ *in* $C$ **do**
    **if** $state(C_{neighbour}) = VIRGIN$ **then**
      $g(C_{neighbour}) = g(C_{neighbour}) + costOfLink$;
      $h(C_{neighbour}) = distance(C_{neighbour}, C_{destination})$;
      $f(C_{neighbour}) = g(C_{neighbour}) + h(C_{neighbour})$;
      $C_{father}(C_{neighbour}) = C$;
      Insert $C_{neighbour}$ into $O$;
    **end**
    **else if** $state(C_{neighbour}) = OPENED$ **then**
      **if** $g(C_{neighbour}) > g(C_{neighbour}) + costOfLink$ **then**
        $g(C_{neighbour}) = g(C_{neighbour}) + costOfLink$;
        $C_{father}(C_{neighbour}) = C$;
        Insert $C_{neighbour}$ into $O$;
      **end**
    **end**
  **end**
**end**

## 3.2 TEA* Implementation

Taking the A* graph search algorithm and adding a third temporal layer enables the neighbour node that will be explored to belong to the next temporal layer, as seen in Figure 3.1. With this, it is possible to associate each link to the instant of time it will be occupied, preventing collision between robots or paths being blocked by other robots by creating deadlocks and creating far more efficient routes. Furthermore, in addition to exploring the neighbour nodes in the next temporal layer, the node itself is also "explored" and re-added to the list with a small increment to the cost. This addition is helpful for cases where the robot staying in place is more efficient than taking a longer route, such as having a robot wait for another robot to pass through instead of immediately attempting to take a longer route that would take longer.

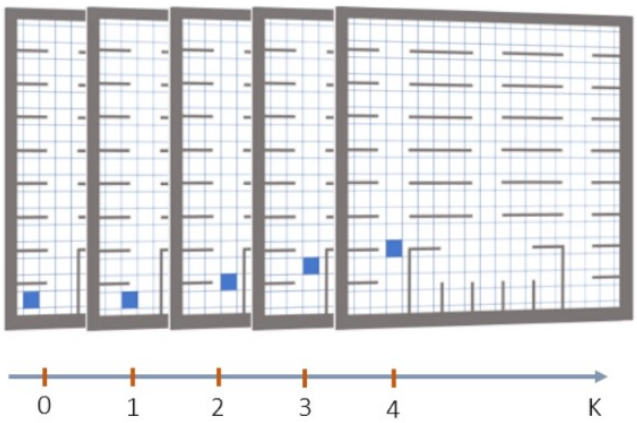



Figure 3.1: Propagation of the positions planned by TEA* through the temporal layers K [6]

The run-through of the TEA* algorithm is pretty similar to the A* algorithm. Firstly, instead of a single graph and states for each node, there is an array of graphs, and each node has a specific state and stored cost at a specific time $K$ which is known as "step". The algorithm runs for as many times as many robots that exist in the system, and, following a priority order, each robot plans its path through time and blocks that path to the lower priority robots by setting the nodes that belong to its path to the state *OBSTACLE*. The next robots plan their path, considering the new *OBSTACLE* nodes in time. A pseudo-code of this algorithm can be seen in algorithm 2.

It is essential to note that this algorithm is hard to scale. This problem is due to needing to store much memory regarding the array of graphs used to plan the path in time. On top of that, the algorithm also requires a lot of computational resources, namely read/write speed to and from memory, as the algorithm has to look for the next neighbour node to be explored constantly and update the values of the costs/states. One of the steps to reduce this potential read/write speed problem was utilizing a low-time complexity algorithm to store the graph's data. As such, in a graph, the nodes and links were stored in a C++ map container from the C++ standard library, where the keys of that map were the ID of the node/link, and the graph itself was stored in a

vector of graphs. The map container was helpful because the time complexity, in big O notation, of searching for a member given the key is $log(n)$. Using such a container is better than using more usual containers such as C++ vectors or deques, which would have at least $O(n)$ temporal complexity. Regarding using a vector to store the temporal layers of the graph, since the steps go from 0 to $K_{max}$ sequentially, the temporal complexity of accessing a specific temporal graph in that vector is $O(1)$.

Furthermore, many more minor changes had to be made to the order dispatcher, which takes in the orders and processes them by planning a path for each robot. The main changes were by changing how orders were handled because when a new order arrives for a robot, all the robots' paths need to be re-planned to ensure efficiency and safety for the new order's required paths. Additionally, there is a need to ensure that the robots are currently behaving as expected regarding their path. As such, it is vital to know where the robots are in real-time.

To further ensure safety, it is also common to give robots a bigger space gap between them. For example, instead of setting only the current node that the robot is, in time, as *OBSTACLE*, the neighbour nodes around that node are also set as *OBSTACLE*. Additionally, it is also common to set as *OBSTACLE* a few previous nodes in time as *OBSTACLE*. For example, if it is defined that three nodes in time are to be set as *OBSTACLE* then if a robot has a path $P_A = \{1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5\}$, when planning the path for that robot, in the step $K$ that the node (4) is expected to be *OBSTACLE*, both the previous nodes (3) and (2) are also set as *OBSTACLE* to ensure further safety. These gaps are implemented in a way to depend on a safety parameter called *SAFETY_GAP*, which sets both the number of previous path nodes that are set to *OBSTACLE* but also the depth of the neighbours of the current path node to be set as *OBSTACLE*. For example, if *SAFETY_GAP* is 4 and path is $P_A = \{1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5\}$, in the step where node (4) is going to expected to have the robot, the nodes (4), (3), (2) and (1) are set to *OBSTACLE* and the neighbour nodes of the node (4) and the neighbours or those neighbours are also set to *OBSTACLE*. Additional examples can be seen in Figure 3.2. A relevant note is that the smaller the *SAFETY_GAP* value, the more efficient the planned path is, but it also becomes less safe.

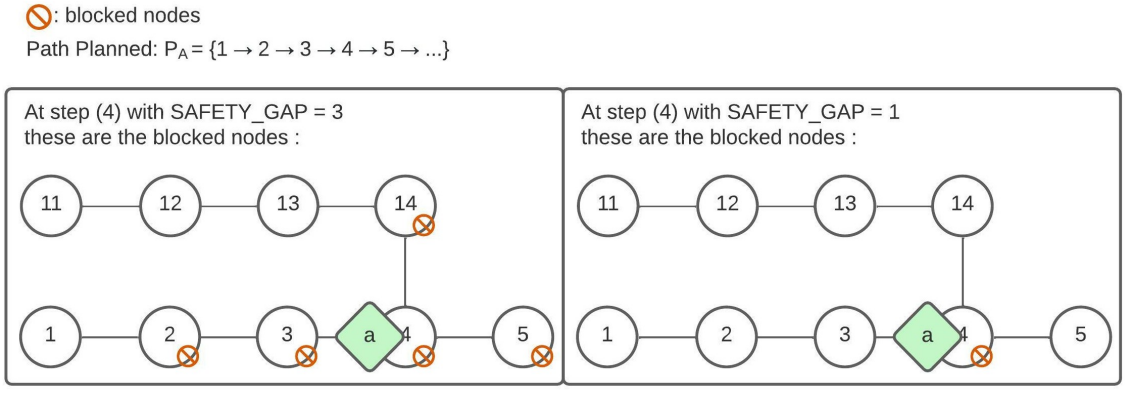



Figure 3.2: Additional examples of the path planning *OBSTACLE* setup

**Algorithm 2:** TEA* algorithm implemented

**Inputs :** $C_{robot}$ and $C_{destination}$ of all robots
**Outputs:** $path$ for all robots
Make $K = 0, 1$ of each $C_{robot_i}$ state as $OBSTACLE$;
**for** $robot_i$ *in* $Robots$ **do**
    Min-heap "$O$" is created;
    $state(C_{robot_i}) = CLOSED$;
    $f(C_{robot_i}) = h(C_{robot_i})$;
    Insert $C_{robot_i}$ into $O$;
    **while** $O \neq \emptyset$ *or* $i \leq Max_{iter}$ **do**
        Take lowest cost node $C$ from $O$;
        $state(C) = CLOSED$;
        $K(C_{neighbour}) = K(C) + 1$;
        **if** $C_{neighbour} = C_{destination}$ **then**
            Create vector $path$;
            **while** $C_{father}(C) \neq C_{robot_i}$ **do**
                $state(C_{father}) = OBSTACLE$;
                add $C_{father}(C)$ to $path$;
                $C = C_{father}$;
            **end**
            return path;
        **end**
        **for** $C_{neighbour}$ *in* $C$ **do**
            **if** $state(C_{neighbour}) = VIRGIN$ **then**
                $g(C_{neighbour}) = g(C_{neighbour}) + costOfLink$;
                $h(C_{neighbour}) = distance(C_{neighbour}, C_{destination})$;
                $f(C_{neighbour}) = g(C_{neighbour}) + h(C_{neighbour})$;
                $C_{father}(C_{neighbour}) = C$;
                Insert $C_{neighbour}$ into $O$;
            **end**
            **else if** $state(C_{neighbour}) = OPENED$ **then**
                **if** $g(C_{neighbour}) > g(C_{neighbour}) + costOfLink$ **then**
                    $g(C_{neighbour}) = g(C_{neighbour}) + costOfLink$;
                    $C_{father}(C_{neighbour}) = C$;
                    Insert $C_{neighbour}$ into $O$;
                **end**
            **end**
        **end**
    **end**
**end**

With this, the TEA* algorithm can plan efficient and collision-free paths for the various robots in the system. An example of this planning can be seen in Figure 3.3, where the robot (a), whose goal is to reach node 2, has a higher priority than the robot (b), whose goal is to reach node 1. With this, robot (a) blocks the most efficient path for the robot (b) but, at the same time, because robot (b) stays in place for one step instead of going through the longer route this ensures the most efficient path for both robots. The path result for a situation like this would be similar to $P_A = \{5 \rightarrow 4 \rightarrow 2\}$ and $P_B = \{6 \rightarrow 6 \rightarrow 4 \rightarrow 1\}$.

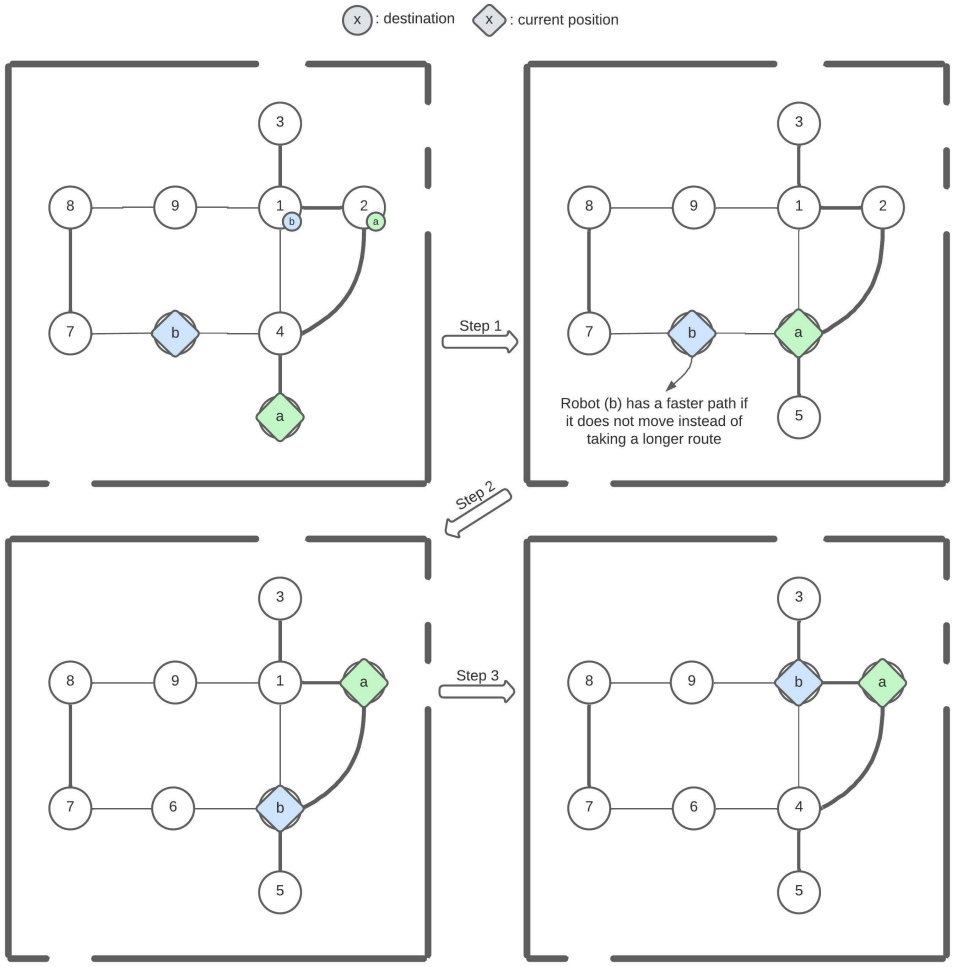



Figure 3.3: TEA* planned path in 3 temporal steps

### 3.2.1 Robot Priority Swapping

Even though the base TEA* algorithm can plan paths for all the robots in the system, extra tweaks exist to optimize it further. One of these can be the addition of a priority swap for the robots. This

swap is an essential feature as sometimes the base algorithm cannot find a path for the current setup of robot priorities. Figure 3.4 shows an example of this. In this situation, robot (a) has higher priority, and its goal is to go to node 1, while robot (b) wants to reach node 3. If the priorities are kept, the robot (a) would arrive at node (1) and block the path for the robot (b) infinitely. By changing priorities, the robot (b) can make its path to node (3), and robot (b) can make its way to node 1. Thus, having a list of possible priority setups can help eliminate this problem and find paths that the base algorithm would be unable to.

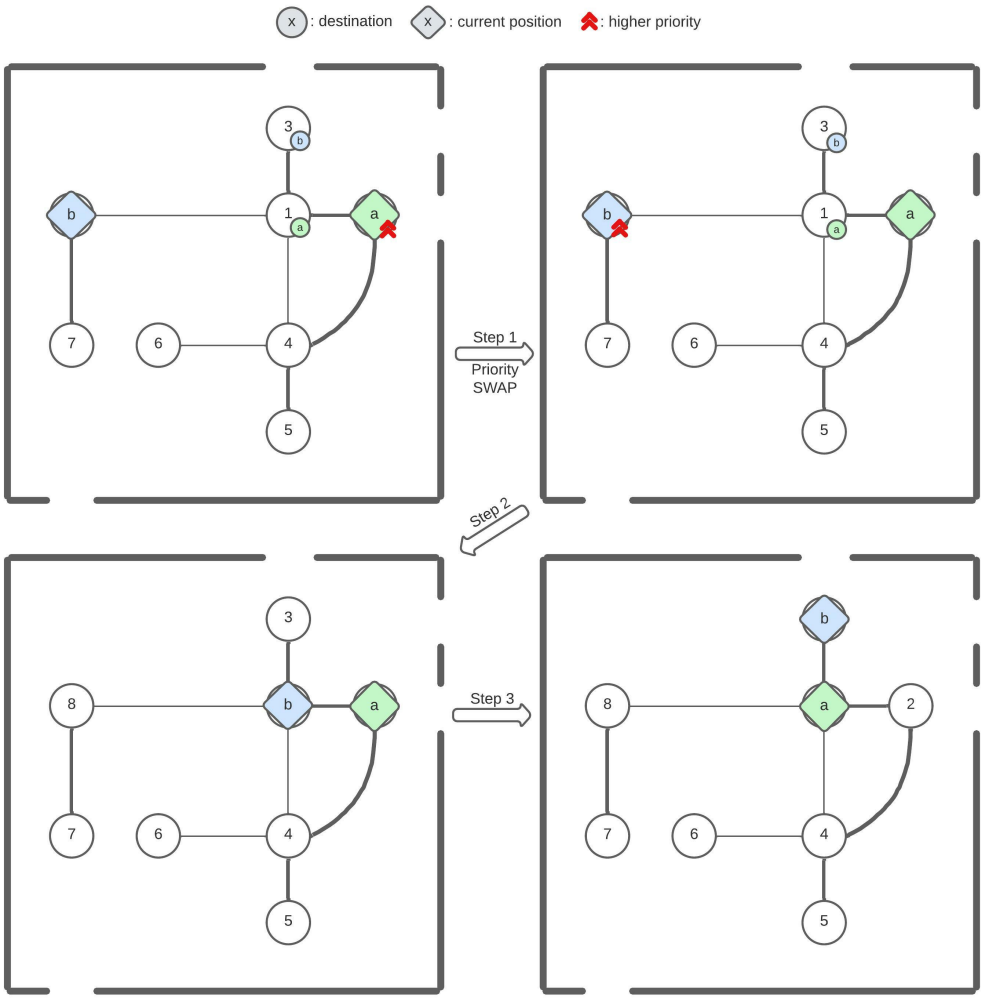



Figure 3.4: TEA* planned path with priority swap between robot (a) and (b)

It is important to note that this type of priority swap is only used in cases where the higher priority robot completely blocks the possible paths for the lower priority robots. Otherwise, if the lower priority robot can still find a course, even if it takes a very long time to achieve it, and it would have been faster to swap the priorities, that path will be the result of the path planning by

TEA*. With this in mind, some tweaks are possible by limiting the number of iterations that the algorithm runs through (the value of the $Max_{iter}$ on the algorithm 2) or even the maximum value of allowed temporal steps in a path. Limiting these would result in longer paths not being found and lead to a priority swap to attempt to find new paths, but it would also restrict the length of paths found in general. These two values are usually tweaked to ensure that the user is satisfied with the length of the paths found.

Furthermore, it would also be possible for the algorithm to run with all sequences of priorities possible and then choose the one that minimizes the global cost. This idea would result in more efficient paths for every robot generally. However, it would require way more computational power as well as processing time to run, as the number of iterations that the whole algorithm would need to run would increase exponentially with the number of robots in the system. As such, this idea of attempting every possible priority sequence is feasible but should only be used with about 3-4 robots due to its exponential increase in complexity.

Another important note is that even though priority swapping may allow all robots to reach their destination, there may be times when this does not happen. In some specific cases, either the robot with the higher priority can find a path, or, by swapping the priorities, only the robot with the original lower priority can find the path. When this happens, the priorities switch back to the original priorities, and only the robot with the highest priority gets a path. At the same time, the other robot is set as a permanent obstacle in time, and the user is informed that the entire order was impossible to accomplish. This was the approach implemented, but in a real industrial environment, it is more likely that if paths are not found for every robot in the order, the order is not processed at all, and an error is returned.

Although this is an overall optimal feature, it is also essential to consider some aspects. The main one is that the path planning of the robots may take a lot longer to complete since it may need to be repeated way more times than usual until a path for all robots is found. This feature is also not very scalable with the number of robots in the system. The more robots exist, the more options for priority swaps exist. A very high number of priority swaps can generate a lot of processing time to create paths for all robots, and because path planning may be needed while the robots are still running, this may be a significant cause for concern. A small change that can be applied to the algorithm is only using this priority swap at the beginning when all robots are stopped. This way, the algorithm may take more time without running the risk of possibly causing a collision from not updating a path quickly.

### 3.2.2 Moving Idle Robots

Another relevant feature to incorporate into the base algorithm is moving other robots when idle and in the way of another robot. This situation can be seen in figure 3.5. In this figure, robot (b) is idle and blocking the destination of robot (a). Having detected this, an order is generated to move the robot (b) to a close unoccupied station to stop blocking the destination of robot (a), thus having robot (a) proceed to its destination.

This solution can be achieved by first detecting if a path is not found because another lower priority robot blocks the destination by being stationary and plans the path ignoring that stationary robot. Then, by looking into the nearby stations of the stationary robot, it is possible to attempt to move the idle robot to the closest unused station, thus unblocking the way. This feature can only be applied to robots of lower priority, so combining this method with the previous process of priority swapping helps achieve more possible paths for the robots.

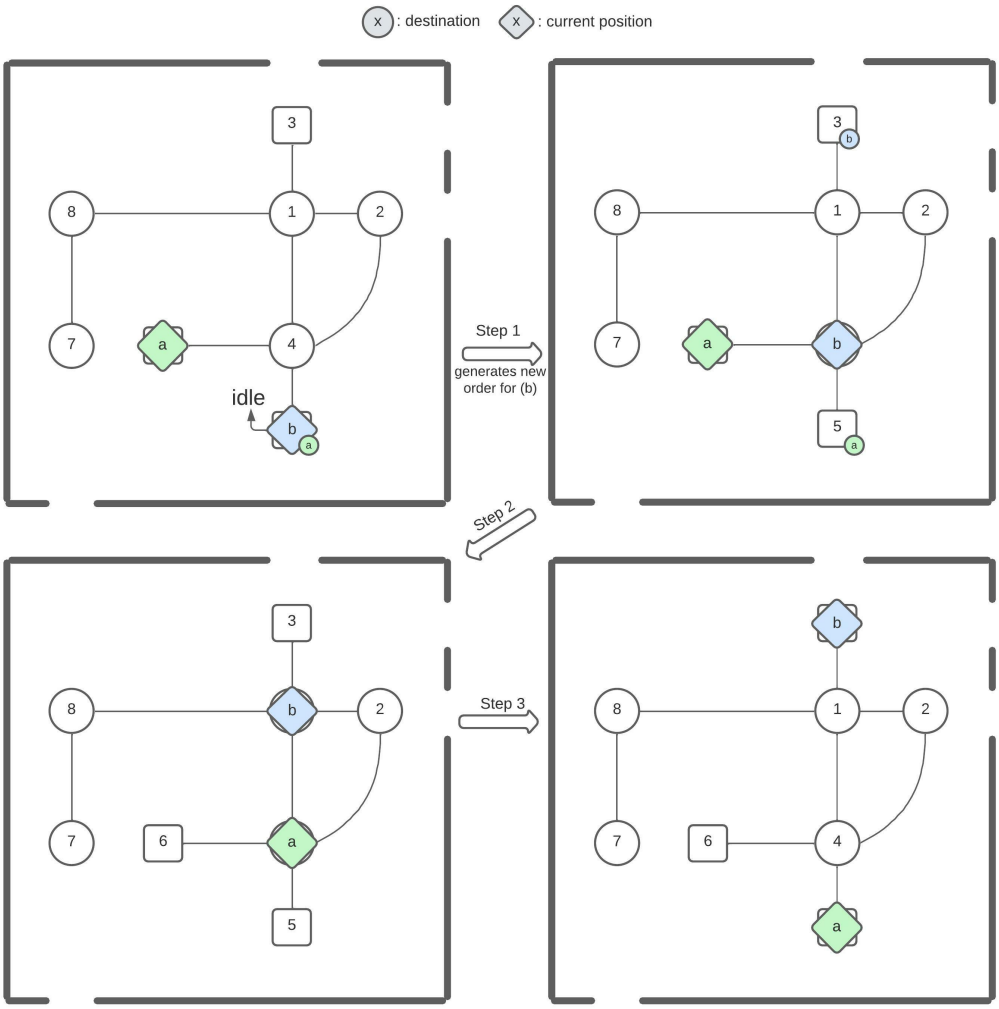



Figure 3.5: TEA* planned path by moving an idle robot

### 3.2.3 Graph Crossing Detection

One final feature that is relevant for planning paths for all robots in the system safely is detecting specific cases in which, even though within the same graph there is no connection between the

edges, if two robots pass through both those edges, it might cause a collision that is undetected by the core TEA* planning algorithm. Figure 3.6 shows some examples of this situation.

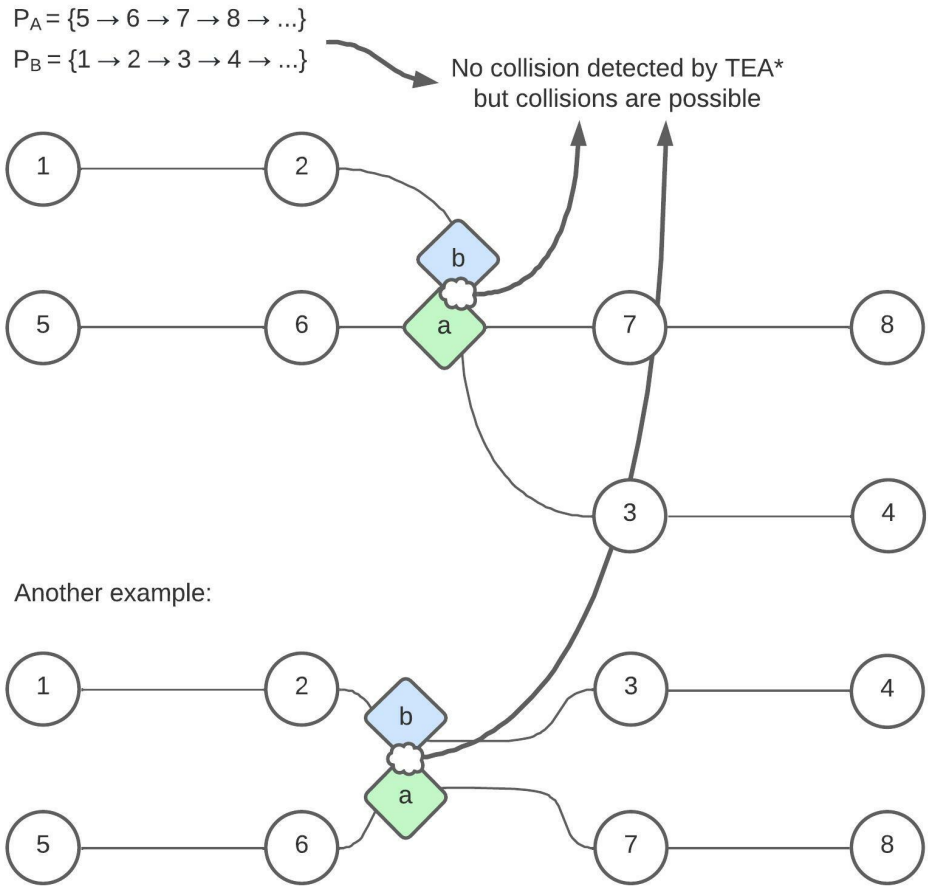



Figure 3.6: Examples of graph crossings causing collisions

This is an important feature, and it is needed to ensure total safety with the planning of the robots but is still in current development in other projects. Because of this, it is not implemented in this project. Implementing this feature will likely use the footprint of the robot and a set of exception tables to prevent these problems from happening.

## 3.3 Overview

This chapter presented the implemented path planning algorithms used to coordinate groups of robots. The TEA* algorithm is a competent tool effective for the coordination of multiple robots

regarding their paths, and one of the goals of this project is to show its capabilities and even more with the coordination of heterogeneous robot groups. Finally, with the description of the implemented TEA* path planning algorithm, it is possible to define the architecture of the global path planning system implemented.

# Chapter 4

# System Implementation

This dissertation's primary goal is to coordinate a group of multiple heterogeneous robots efficiently. This type of coordination is often required when controlling many robots in a large industrial environment as it can improve the system's overall efficiency by having multiple robots capable of doing different tasks. In this dissertation, the steps taken to achieve heterogeneous robot coordination were to implement a method capable of coordinating a group of robots and then improve it to be more robust to the heterogeneity of the various robots in the system. Furthermore, a requirement for the dissertation was the usage of the TEA* graph search algorithm for the path planning of the robots and that this system is implemented using the Robot Operating System (ROS) framework.

To showcase the capabilities of the implemented coordination system, a simulation environment is used to test various aspects of the system and make sure they are ready for a real robot environment.

## 4.1 Simulation Platform

In order to implement and test a system capable of coordinating a group of robots, it was necessary to recreate an industrial-like environment where the conditions are similar to a real one. Furthermore, this system needs to work with the ROS framework, which is a requirement for the dissertation.

Using the *Stage* simulator, various environments were set up to host the robots that were going to be coordinated by the implemented system. With this simulation, it was possible to simulate a nearly real industrial environment by creating walls, corridors and even objects that could collide with the robots. Furthermore, these objects could be moved to create obstacles for the robot's paths and test safety features. The robots themselves were also simulated in this simulator. The robots could localize themselves using simulated LIDAR sensors on their front and back, along with collision detectors that could detect if the robots had collided with anything, forcing them to stop.

To view all the data being circulated, such as the current robot's estimated location and the graph points, the RVIZ visualization tool from the ROS framework is used. With this tool, it is possible to visualize the robots running through the graph and have the option to send orders directed through the RVIZ tool for robots to move from one point to another. This tool can also be used to edit and save graphs instead of manually inserting coordinates and values in the manual definition.

### 4.1.1 Simulated Environment

Using the Stage simulator, a map of an existing zone was created to simulate an industrial environment. This map can define obstacles such as walls, doors and corridors, allowing a realistic simulation environment. This simulated environment with the group of robots can be seen in Figure 4.1. This environment was used not only to simulate the movement of the robots but also collisions with the walls and other robots in the system. Furthermore, the simulation also attempts to simulate a real robot by generating minor movement errors, equivalent to real situations such as slippage in the wheels.

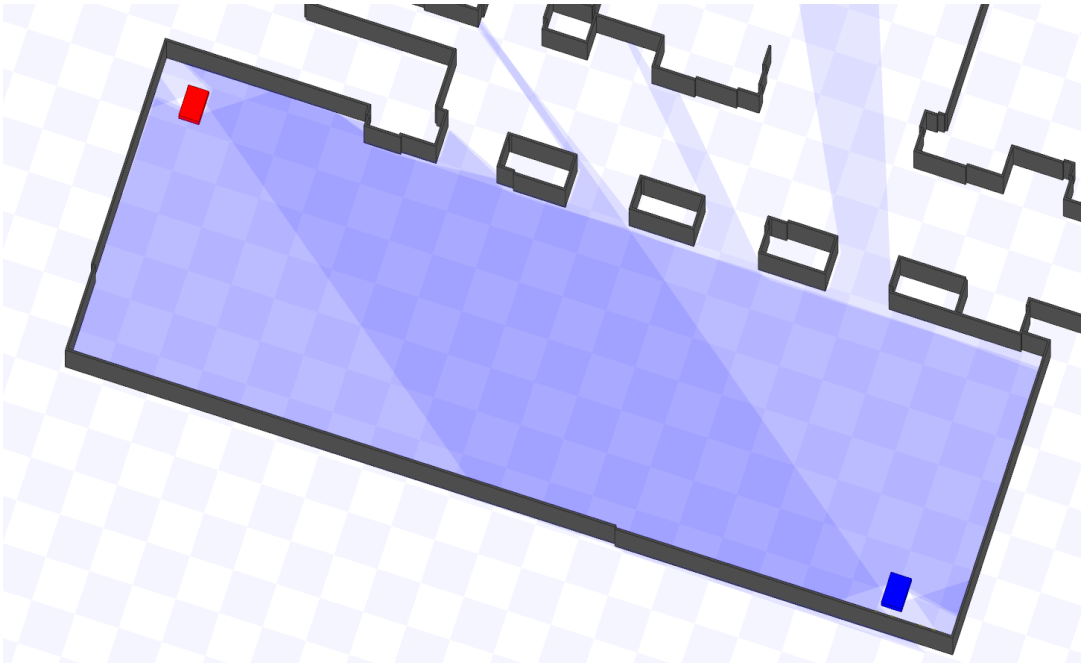


Figure 4.1: Initial simulation environment used.

Moreover, having a fixed environment allows for defining nodes and their respective links to identify possible paths for the robots to take, forming a graph. An example of this map and graph can be seen in Figure 4.2.

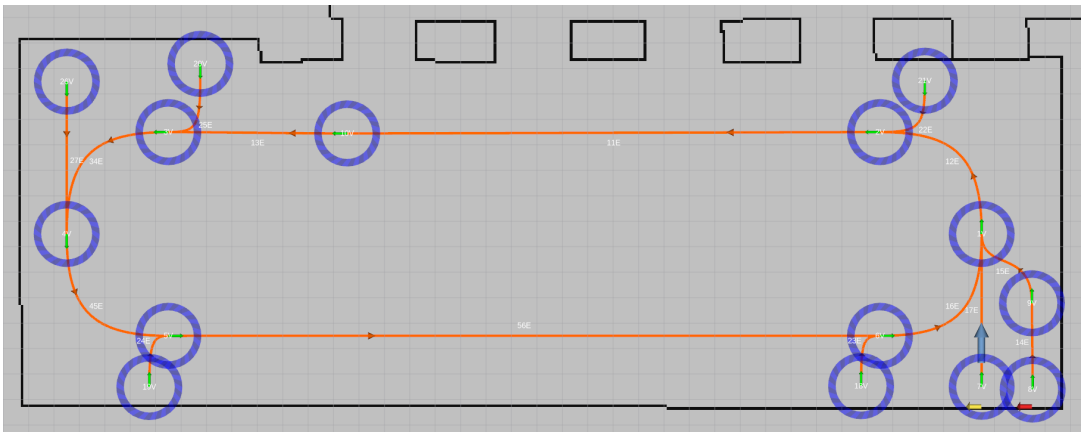


Figure 4.2: An undecomposed map and its graph used in simulations.

This graph definition can be seen using the RVIZ visualization package from ROS. This tool allows data visualization by reading published topics such as robot position and graph definitions. It uses these definitions to display the data sent and received to and from the path planning system. Using this data, it is possible to define what each data represents visually and obtain what is seen in Figure 4.2, where the blue arrow represents a robot, the blue circles represent each node and the red lines the definition of the edges between the nodes, along with a small green arrow to indicate the direction of movement expected from each node.

With this graph, it is possible to define the robot's $C_{space}$. Any robot that follows the path formed by the various edges should not encounter any obstacles. In sum, the $C_{space}$ is represented by this graph, where the various edges and nodes are the $C_{free}$ and everything else is the $C_{occupied}$.

### 4.1.2 Simulated Robots

The simulation uses various robots. These robots differ in various aspects that make them heterogeneous, such as their size, format and traction type. These differences make it possible to simulate an industrial environment where a group of heterogeneous robots are used.

These robots can be seen in the stage simulator, in Figure 4.1, as their defined form with a specific colour. These robots have collision detection algorithms running to detect collisions with walls, obstacles or other robots, and when that happens, they stop. Furthermore, their position is obtained and updated using two LIDAR sensors on their front and back and SLAM algorithms.

Additionally, using the libraries to control these robots already developed by [9] and [8], it is possible to command the robots to follow a specified edge in a defined direction.

## 4.2 System Architecture

This section will present the chosen control architecture for the system. Moreover, it will define the system's overall architecture, how the various existing modules interact with each other in order to coordinate the various robots inside the system and how they work internally.

### 4.2.1 Control Architecture

The chosen control architecture for this dissertation project is the centralized control architecture. The main reason for this is one of the requirements of the path planning of the robots, the usage of the TEA* algorithm as a graph search algorithm. Using this algorithm, there is a need to know where all the robots are positioned with this algorithm before planning a path [34].

Even though any of the described architectures mentioned in the literature review could be used, a decentralized architecture would require an enormous amount of information to be transmitted to and from each robot. Because of the nature of the TEA* algorithm, using that architecture would result in a very sub-optimal system. Furthermore, each robot would require a lot of memory and computation power to be able to process the algorithm. A hybrid system could be an option, but there is no real advantage to having any communication between the robots using the TEA* algorithm. There are some potential safety benefits to having robots communicate their position or status to each other or precautions in case the central system fails. However, these precautions are not the focus of this dissertation. As such, a centralized control architecture is used.

### 4.2.2 Implemented Modules

The system architecture can be described in the modules seen in Figure 4.3. These six modules describe the system used to coordinate the various robots.

Furthermore, on top of all these modules, a supervisor system is implemented to ensure that the robots behave as expected after receiving their paths.

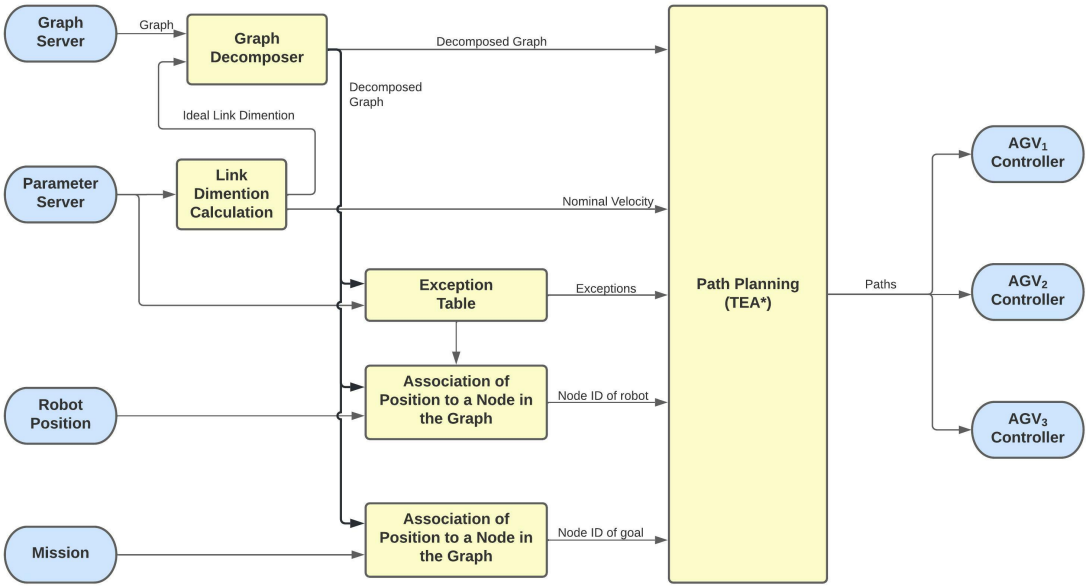



Figure 4.3: The architecture of the implemented robot coordination system

In sum, a short description of these modules can be given as such:

- Firstly, a graph is created. This creation can be done manually or automatically, but it is created manually in the case of this project. This graph represents the space where the robots can move, the crossing points, the turning points as well as loading and unloading points. This graph is then added to the graph server before the system starts.
- Given a parameter server that holds various characteristics, such as the robot's size and traction type, an ideal link dimension is calculated for the decomposed graph.
- Before the system starts, these graphs can be optionally decomposed. This decomposition means that the graph links are subsequently divided so that all links have a similar length. This decomposition is crucial for efficient heterogeneous robot coordination and will be further clarified in Sub-section 4.4.
- Once the system is started and given an order for a robot to move, both the robot's current position and the order's destination position are associated with their position relative to a node in the graph.
- Finally, a path can be calculated by knowing the node to which the robot belongs and its destination node. This path planning also considers notable exceptions in an exception table. Once a path is calculated, it is sent to each corresponding robot.
- On top of all, a supervisor also exists to ensure that the robots are following the paths planned without the risk of problems such as collisions and deadlocks.

Finally, a class diagram of the implemented system can be presented in Annex A.

## 4.3 Graph Definition

The graphs are defined manually using YAML serialization language and specifying various fields to define each node and link to realize this project.

The following fields define an edge:

- CURVETYPE : The type of curve of the edge ("linear" or "spline").
- ID : The ID of the edge.
- ORIGIN_ID : The ID of the origin node.
- DESTINATION_ID : The ID or the destination node.
- PARAMF : Forwards parameter used in spline curve definition. Further explained in 4.3.1.
- PARAMB : Backwards parameter used in spline curve definition. Further explained in 4.3.1.

- VELOCITYFORWARD : Velocity of robots going from origin node to destination node.
- VELOCITYBACKWARDS : Velocity of robots going from destination node to origin node.

Furthermore, the following fields define a node:

- FRAMEID : The frame ID of this graph.
- ID : The ID of the node.
- THETA : Angle of the node (green arrows on Figure 4.2). Further explained in 4.3.1.
- X : X position of the node in this frame.
- Y : Y position of the node in this frame.

Using these definitions, it is possible to define the graph and use it to plan paths for the robots in the system.

In addition to the base graph definition, other details were implemented in the system. One of these was the existence of stacked nodes. Stacked nodes are a method of connecting two different graphs by having one node of each different graph and stacking them in the same position and orientation. Whenever this stacking is detected, the links connecting to both nodes are connected internally so that both graphs can be used for planning paths. Additionally, some nodes are defined as station nodes. These station nodes are nodes where the robots are expected to be stationed when not completing any orders (idle status). These station nodes are defined by having a very high number for an ID.

### 4.3.1 Cost of the Links

Regarding the *costOfLink* value seen in the algorithm 1, this is the cost of a link that connects two nodes in a graph. The lines representing the various links can either be represented by linear equations or lines described by Bézier curves, depending on the CURVETYPE value defined in the graph definition (as seen in Section 4.3). While the costs of the linear equation line are pretty clear to calculate using the euclidean distance between the two points that make the nodes, the cost of the Bézier curves is not.

The graph server defines the Bézier curves as two points' positions and angles, from the origin node and the destination node, and two parameters, the "paramF" and the "paramB". These parameters describe the distance between the nodes and the two other control points utilized in defining a Bézier curve. In Figure 4.4 a better visualization of how the parameters describe the curve is shown.

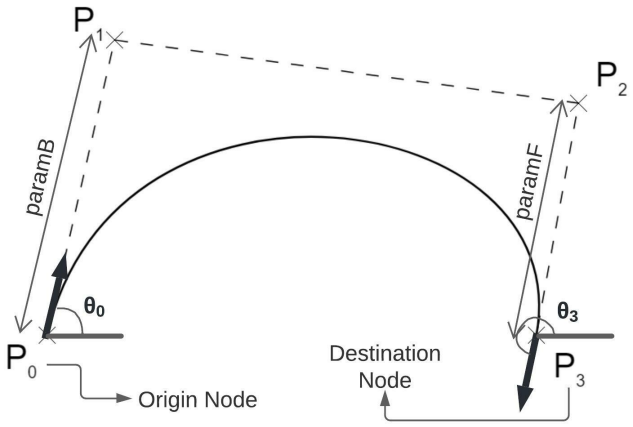



Figure 4.4: Visualization of the Bézier curve description

Using these parameters, it is possible to obtain the points $P_1$ and $P_2$ by utilizing the angle of the nodes and the distance to the points and, thus, the four points that define the Bézier curve are obtained: $P_0(x_0,y_0)$, $P_1(x_1,y_1)$, $P_2(x_2,y_2)$, $P_3(x_3,y_3)$. Finally, using these points, the Bézier curve that defines the link can be described [35] using the equations in 4.1 and 4.2.

$$\begin{aligned} x(\lambda) &= a_x\lambda^3 + b_x\lambda^2 + c_x\lambda + x_0 \quad ,0 \leq \lambda \leq 1 \\ y(\lambda) &= a_y\lambda^3 + b_y\lambda^2 + c_y\lambda + y_0 \quad ,0 \leq \lambda \leq 1 \end{aligned} \tag{4.1}$$

where,

$$\begin{aligned} c_x &= 3 \times (x_1 - x_0) \\ b_x &= 3 \times (x_2 - x_1) - c_x \\ a_x &= x_3 - x_0 - c_x - b_x \\ c_y &= 3 \times (y_1 - y_0) \\ b_y &= 3 \times (y_2 - y_1) - c_y \\ a_y &= y_3 - y_0 - c_y - b_y \end{aligned} \tag{4.2}$$

Using the 4.1 equations, it is possible to calculate the length of the curve and use that value for the cost of each link. In this specific implementation, since the curve is defined by a variation of $\lambda$ between 0 and 1, various points are taken, for example, one hundred different values between 0 and 1 that can be used to obtain one hundred points along the curve and use the distance between those points to get an approximation of the length of the curve.

Taking the length of the curve, if the aim is to optimize the algorithm to be distance-efficient, then the *costOfLink* is equal to the length of the curve, and if the goal is to be time-efficient, then the *costOfLink* is equal to the length of the curve divided by the velocity of that link.

## 4.4 Graph Decomposition

Due to how the TEA* algorithm works, the path planning returns a sequence of edges or vertices that define a path. Therefore, if the various edges' sizes are very different, the time it takes for a robot to travel an edge may be substantially different from another robot travelling another edge, which is even more substantial when the fact that different edges can have different velocities is taken into account. This difference can be dangerous as TEA* assumes that the time it takes to go through an edge is the same for all edges, and if that is not true, the path planned cannot ensure that there is no collision between robots or that potential deadlocks can occur. An example of this specific problem can be seen in Figure 4.5 where, without the decomposition of the graph (and assuming all link velocities are equal), the TEA* plans a path that leads to both the robots colliding due to TEA* reading that the robot B will be out of the node (5) before robot A attempts to go to that node, which does not happen in reality, leading to a collision between the robots. On the other hand, when the graph is correctly decomposed, the algorithm can see that robot B will need the node (5) when robot A needs it, and it will block the path for robot A accordingly.

Due to these problems, to ensure that the robots proceed safely through their path, it is necessary to decompose longer links into smaller links to minimize the differences between the time that it takes to travel between the different edges.

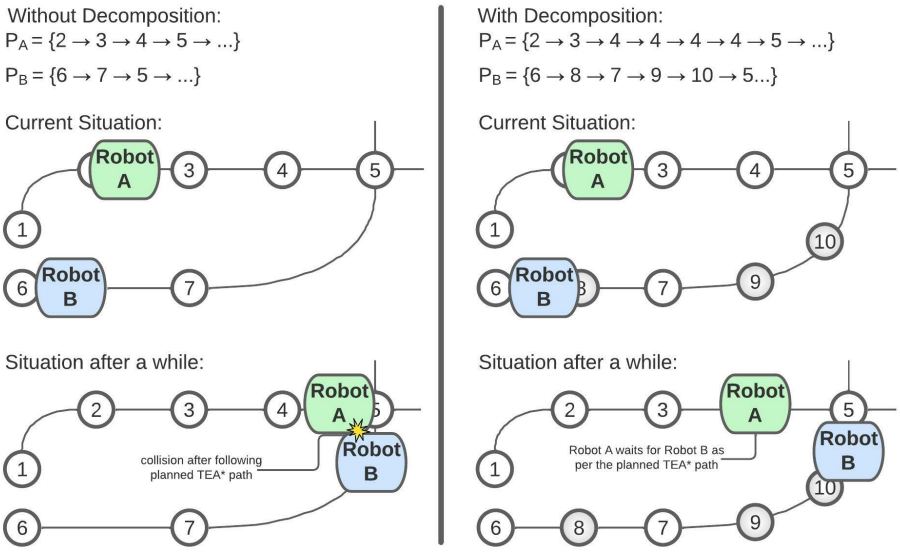



Figure 4.5: Example of collision due to lack of graph decomposition

Furthermore, this decomposition also improves the optimisation of the routes for the robots, as decomposing a lengthier edge into smaller ones allows for more than one robot to travel that edge sequentially. This example can be seen in Figure 4.6.

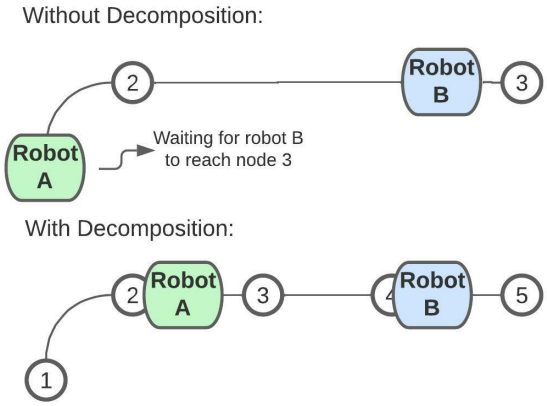



Figure 4.6: Graph decomposition example improving efficiency

With both these improvements, there is a significant advantage for the path planning of heterogeneous robots. With the decomposition of a graph, even heterogeneous robots can move efficiently through the planned path, as long as these robots are not lengthier than the length of the links. If deviations from the intended path arise, a supervisor can act to solve these problems. Further explanation of the supervisor and why the length of the new links cannot be lengthier than the size of the robots can be seen in Section 4.5.

### 4.4.1 Implementation

A pseudo-code for the algorithm used to decompose the graphs can be seen in algorithm 3. Before explaining how the algorithm is implemented, it is perhaps important to have a notion of what a "step" is in this context. The step of a link is the time it takes to go from the origin node of a link to the destination node of that link, at the specified velocity of that link. As such, a step can be described by the equation 4.3.

$$step\ of\ link = \frac{length\ of\ link}{velocity\ of\ link} \tag{4.3}$$

With the definition of step, the first phase of this algorithm is discovering a step value that will be used to decompose all other links. To do so, first, the following four values need to be defined:

- MIN_LENGTH : minimum length possible for a link, usually the size of the biggest robot.
- MAX_LENGTH : maximum length possible for a link.
- MIN_VELOCITY : minimum velocity possible for a robot (different from zero).
- MAX_VELOCITY : maximum velocity possible for a robot.

Using these four values, it is possible to define a minimum step and a maximum step of an interval of values that are possible to be used as the goal step for the decomposition of the graph. Using this range of values, various steps are tested by attempting to divide the various links around that value. The algorithm to divide the steps around a specific step value is shown in algorithm 3.

**Algorithm 3:** Algorithm for decomposition around a step value

**Inputs :** $goalStep$, $MIN\_LENGTH$
**Outputs:** $maxError$, $avgError$
**for** *link in Links* **do**
  $currStep = length(link)/velocity(link)$;
  $divError = currStep/goalStep - (int)(currStep/goalStep)$;
  **if** $divError \leq 0.5$ **then**
    $div = (int)(currStep/goalStep)$;
  **end**
  **else if** $divError > 0.5$ **then**
    $div = (int)(currStep/goalStep) + 1$;
  **end**
  `// Check if division causes length to be bellow minimum`
  **if** $length(link)/div < MIN\_LENGTH$ **then**
    $div = (int)(currStep/(MIN\_LENGTH/velocity(link)))$;
  **end**
  $newLinkLengthSize = length(link)/div$;
  $stepError = fabs(newLinkLengthSize/velocity(link) - goalStep)$;
  Update avgError and maxError accordingly
**end**

Using this algorithm, the various errors of the division of the links are taken and used to obtain the best possible step for graph decomposition. This goal step acquisition has two different approaches: minimization of the maximum error of decomposition and minimization of the average error of decomposition. On one hand, the minimization of the maximum error of the decomposition approach obtains a step that minimizes the maximum error by dividing the links when decomposing the graph around that step value. On the other hand, the minimization of the average error of decomposition approach obtains a step that minimizes all the divisions' errors on the links and uses the average of the errors of those divisions to obtain the step size that minimizes that overall average. The approach chosen to be used was decided based on testings done and can be seen in Chapter 5.

Once the new optimal step is obtained, this link is used to divide all the links in the graph as per the presented algorithm. This division is done by taking the links, checking if they can be divided more than once by that length, and dividing that link by the number of times that minimises the error to the new optimal link size. If the new length of the links after the division around the optimal step is shorter than the MIN_LENGTH parameter, the links are re-divided with one less division to ensure that the new length of the links are bigger than the MIN_LENGTH parameter.

Finally, when the new link divisions are obtained, the graph proceeds to be re-written to comply with these new sizes. This process implies the creation of new links and new nodes that connect

these new links. Furthermore, due to some links being complex curves, an extra step is needed to ensure that the decomposition does not alter the layout of those curves.

Whenever Bézier curves define the links, an extra step is taken regarding the curvature. This step is essential because if the curvature of the original link is not kept, it may lead to robots colliding with the environment or even generating sporadic movements that may affect performance. A few parameters are updated to solve this problem, the new nodes' angles added to divide the link and the "params" of the new curves. The new "params" values are obtained by dividing the original ones by the number of divisions that the link suffered. On the other hand, the new nodes' angles are not as straightforward. To obtain these values, firstly, the value of $\lambda$ that obtains the $x(\lambda)$ and $y(\lambda)$ corresponding to the node's position is retrieved. Then, using that value, it is possible to obtain two points $P_1(x(\lambda-0.01), y(\lambda-0.01))$ and $P_2(x(\lambda+0.01), y(\lambda+0.01))$. Using the line segment created by these two points and the equation in 4.4 the angle of that line is obtained, which is the angle of the new node.

$$\theta_{new} = atan2(y_{P_2} - y_{P_1}, x_{P_2} - x_{P_1}) \tag{4.4}$$

The graph definitions are stored in YAML files. Using these, edges and vertices of the graph are specified, and both the simulator and the path planner use these files to define the graphs internally, as explained in the Section 4.3. The decomposition algorithm will take these YAML files and edit them to add and change the various edges and vertices in the graph. Because YAML is a common data-serialisation language, various parsers and emitters of this already exist. Since the ROS framework is being used with the C++ language, the yaml-cpp library is used to emit and parse any data to and from any YAML file in this project.

After this process is complete, the graph should be decomposed as expected when the simulation starts.

## 4.5 Supervision

Because in an industrial environment, many variables exist at all times, a supervisor for such a system as a path planner is crucial to ensure safety and efficiency. As such, a supervisor was created to observe the planned path by the system and ensure that all robots within the system are following that path.

The built supervisor considers a fact when planning the robot's path: a link in the graph must not exceed the size of a robot in the system. This point is further assured by the graph decomposition algorithm, with more minor exceptions, such as the original graph having already links whose length is lower than the *MIN_LENGTH*. There are situations where the original graph must indeed have smaller links than the *MIN_LENGTH*, such as tight passageways. However, it is still possible for the supervisor to work, as long as it is known before the supervisor system starts that there are links with short lengths. This problem is solved by having a *SAFETY_GAP* value higher

on the path planner than on the supervisor, as will be seen bellow. Regardless, for the explanation of the supervision algorithm, assume that all links are lengthier than *MIN_LENGTH*.

Assuming a worst-case scenario that all links are the same size as the robots, and two robots are following each other in a path, as long as they are not over one step ahead of the other robot regarding the route planned, they will not collide. If such a thing happens, the supervisor acts and triggers a new path planning. In Figure 4.7 this can be seen.

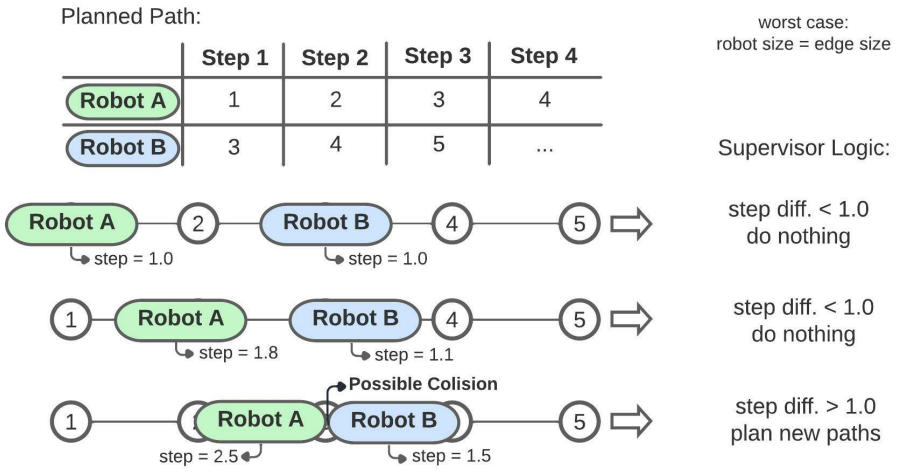



Figure 4.7: Supervisor actuation in the implemented system

Because of how the system works, in the path planner, a robot belongs to a node until it reaches another. Looking at Figure 4.7 when the replanning is triggered, the robot (a) that is still detected as being in node (2) will have a path that will make the robot (a) go back to node (2) and stay there for one step. In contrast, the other robot is detected as being in node (3) and plans a new path from that node, which means the robot (b) keeps advancing as he was, thus eliminating possibilities for collisions.

The value of the difference in the steps that trigger a new re-planning of the robots' paths may be dependent on the *SAFETY_GAP* parameter used on the TEA* path planning, which was mentioned in Sub-section 3.2. In the Figure 4.7 a value of *SAFETY_GAP* of 1 is used and, as such, the re-planning triggers when the step difference is close or above 1.0 steps. If the *SAFETY_GAP* were 3, the re-planning would have triggered when the step difference was above 3, and, as such, the robot A would have gone back to node (2) and stayed there for three more steps before proceeding. However, this bigger gap check, even though it could prevent collisions between the robots such as the example given, it can still lead to problems such as deadlocks due to a robot being too ahead or too behind in time. As such, in addition to the *SAFETY_GAP*, the paths of the robots are also taken into consideration. With these paths its possible to check if the paths of two robots being checked are ever going to affect each other within that *SAFETY_GAP* range, and, when such is

true, instead of checking for a step difference equal to the *SAFETY_GAP* value, check for the step difference of 1.0. Some examples of the supervisor acting can be seen in Figure 4.8.

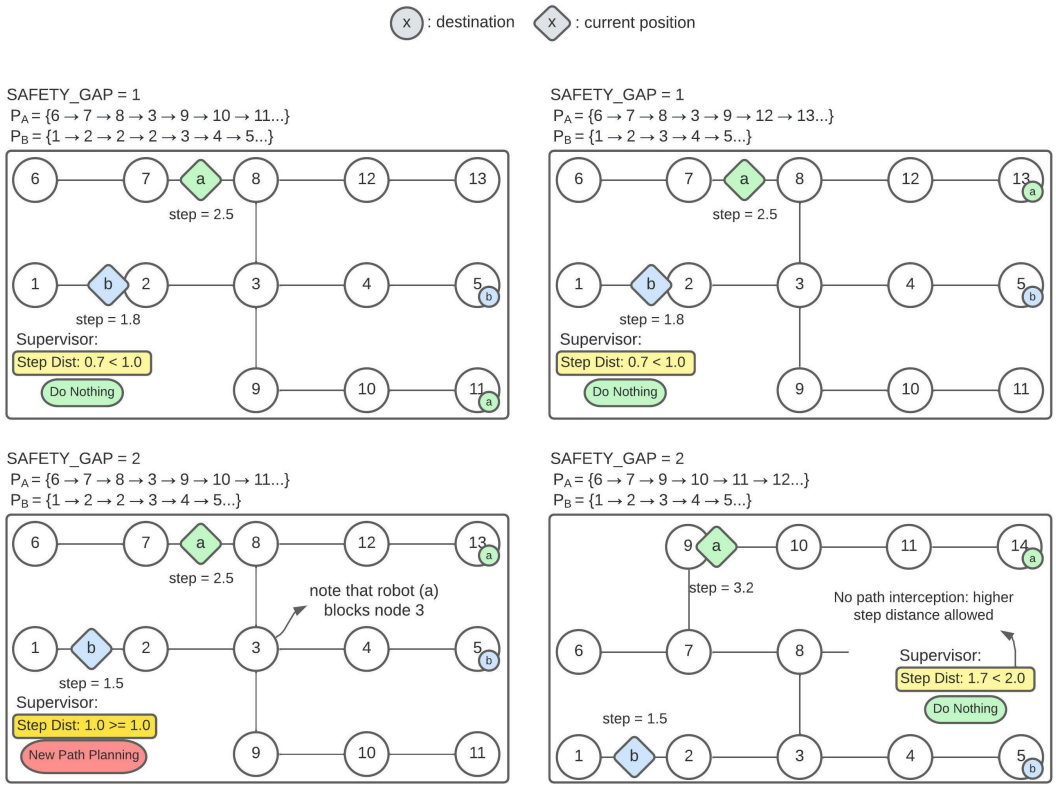



Figure 4.8: Example of path re-plannings triggered by the supervisor

To solve the problem of having a robot being bigger than the size of the smallest link in the graph is having a higher *SAFETY_GAP* value used in the path planning than in the supervisor. For example, if the robot's size is about double the size of the minimum length link if the value of *SAFETY_GAP* in the supervisor is double the value of *SAFETY_GAP* in the path planner, the same example shown in Figure 4.7 would happen but, instead or a robot occupying one link, the robot would occupy two links.

To calculate the current step the robots are in at any time, a routine is created to obtain the percentage of a link that a robot has moved. Using this value, plus knowing the current node the robot is going to and comparing it to the path calculated, it is possible to determine in time the progress of the path that the robot has been attributed (the robot's current "step") and use this value in the supervisor. A particular case is when another robot's path is blocking a robot. In this case, the robot being blocked is set to copy the current step of the robot blocking. This decision is made since the blocked robot is not moving and, as such, poses no threat of collision. Furthermore, the blocked robot only moves when the blocking robot finalizes the step that is blocking him, and, as such, the step of the blocked robot depends solely on the blocking robot.

It is significant to note that because heterogeneous robot groups are being used, it is both expected and necessary for the supervisor to act and trigger new path plannings from time to time. This expectancy is because the heterogeneous robots are likely to suffer small changes in their speeds due to slippage and different traction types and result in their path being temporally off-sync with the planned one, leading to problems due to how the routes were planned by TEA*. This problem is one of the main goals of the graph decomposition, aiming to reduce the number of times the supervisor has to act by attempting to minimise the length difference between all the links in the graph. One of the main goals of this project is to lower the number of times the supervisor acts regarding the graph decomposition algorithm used, as the system's performance is directly related to this.

## 4.6 Overview

This chapter presented everything that was implemented regarding the multi-robot coordination system, how the system was structured and presented in detail how the various internal systems work. The path planning system and the supervisor module are essential to ensure that the robots are correctly coordinated through their paths and kept secure while still being efficient. The addition of the supervisor module along with the graph decomposition algorithm also improves the system's response when dealing with heterogeneity in the group of robots it is attempting to coordinate.

# Chapter 5

# Tests and Results

This chapter will examine the tests and results obtained with the implemented path planning and supervision systems. Firstly the different environments used will be explored, where various maps and graphs were used. Then the various tests that were done will be showcased and explained. Finally, a review of the results and some conclusions will be presented.

## 5.1 Test Environments

Two different maps were used to emulate a real environment regarding the simulations. These two different maps aim to test both the path planner implemented and the stability of the graph decomposition algorithm used. Various tests were done to investigate the overall system's stability further using the different maps and numbers of robots in the system.

Firstly, a smaller map that represents a medium-sized room was used. This map is set with a low number of nodes in the graph and smaller paths, as seen in Figures 5.1 and 5.2. This map was used to test out the various extra features of the TEA* algorithm, such as robots blocking the path, priority swaps between two robots, testing the basic functionality of the graph decomposer algorithm, and the path planning system implemented.

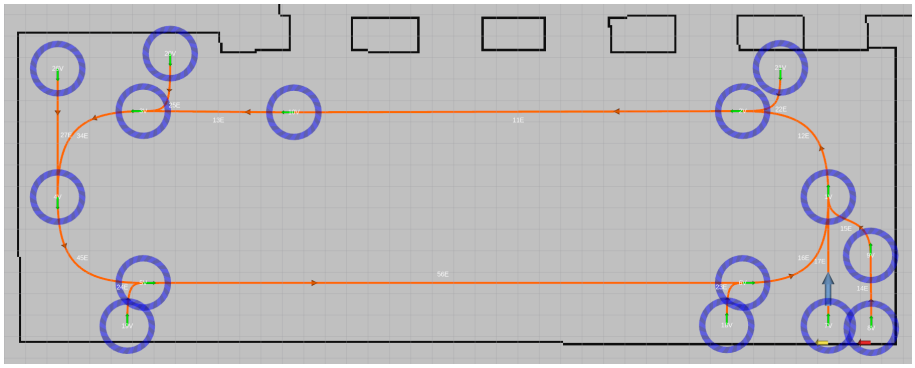


Figure 5.1: Small map and its graph

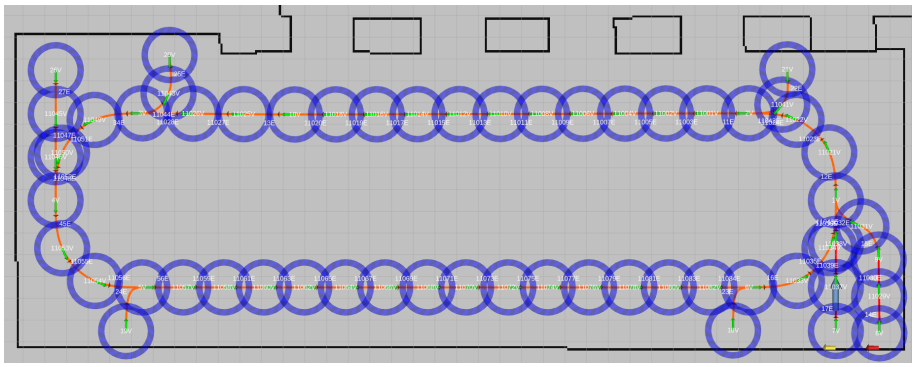


Figure 5.2: Small map and its graph decomposed

To test the performance of the path planning algorithm, the time the path planning algorithm took to complete its planning for all the robots simultaneously was measured. Furthermore, the decomposition algorithm and, consequently, the supervision system was also tested regarding the number of times it had to act and trigger a new path planning to correct the robot's path comparatively to the planned one.

Regarding the second map used, this map is much bigger and simulates a full-sized industrial environment, with access to various rooms and more complex paths. This map mainly was used to thoroughly test the capability of the graph decomposition algorithm due to the high complexity of the curves used and the extreme length differences of the links used in the graph. Additionally, this map was also used to test the capability of the path planning system regarding the time taken to plan a path using extensive paths and various robots. Finally, this map was also used to test the behaviour of the coordination system regarding heterogeneous robots by comparing the system's behaviour between homogeneous and heterogeneous robots. This graph and map simulated can be seen in Figures 5.3 and 5.4.

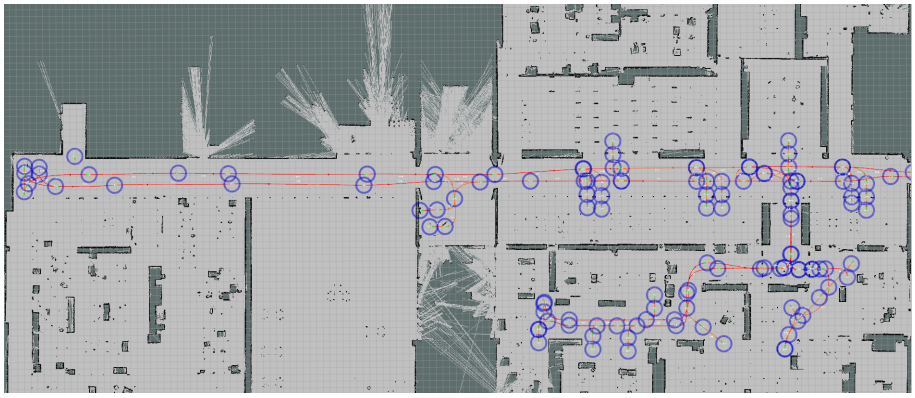


Figure 5.3: Big map and its graph

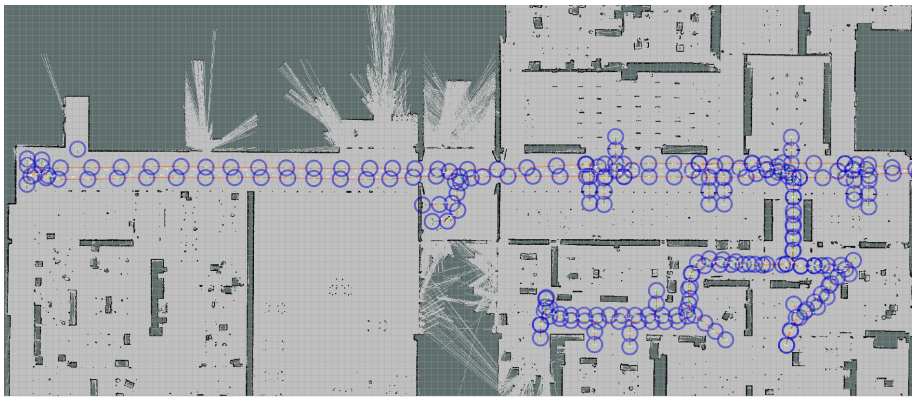


Figure 5.4: Big map and its graph decomposed

It is important to note that only a maximum of four robots were used in these tests. This relatively low number of robots is due to the high computational resources needed to simulate the robots with the big decomposed graph, limiting the number of robots that could be used. Furthermore, the main goal of these tests is still achieved by only having four robots in the system without harming the results.

This second map is more complex than the first map in nearly every aspect. It has way more complex paths, edges with drastically different velocities and generally way more path possibilities. With this added complexity, this map is excellent for extensively testing the graph decomposition and the supervision regarding more complex paths with more and different robots.

## 5.2 Tests Done

As mentioned previously, various tests were done using each map and a different number of robots. The first tests were done on the smaller map to check if the whole system was working as intended. Then, tests were done on the bigger map to thoroughly check the capabilities of the implemented decomposition algorithm and the supervision system. This section will look into how some tests we implemented and the primary goal of each of the tests done.

### 5.2.1 Small Map

Starting with the smaller map, the first tests were done. The first test's goals were to test the core of the path planning algorithm and if everything about the base TEA* algorithm was working correctly, which was necessary for any other test. Then, the extra features were tested.

The first of these extra features was the blocking feature. This test was mainly done by having two robots in nodes 7 and 8 in the graph in Figure 5.5 and having them move each to nodes 2 and 21, causing an interception in node 1. This is an essential test as blocking can be pretty common in an actual situation, and this feature must be working correctly before any other one.

These paths are expected to cause the higher priority robot to go through node 1 without stopping while blocking the path to the lower priority robot, who was waiting for the higher priority robot to arrive at the next node. Figure 5.5 shows an example of the situation.

A similar test was done with three robots that attempted to use node 1 at the same time, which resulted in the higher priority robot blocking the other two robots, the second priority robot moving after the higher priority robot and blocking the lesser priority robot, and, finally, the lesser priority robot moving. Having this working as expected, the next test was realized.

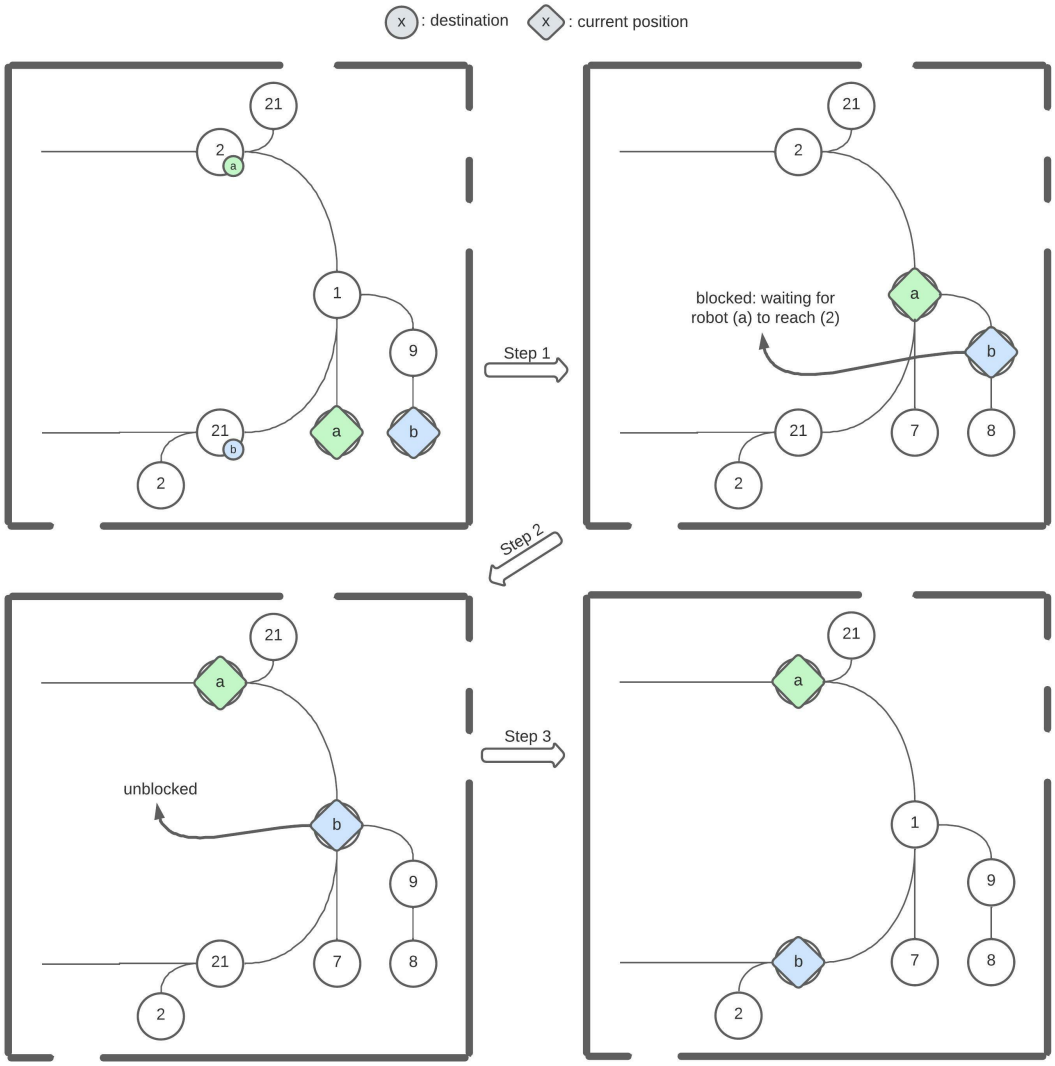



Figure 5.5: Example of a blocking situation

The priority swapping feature was the next test regarding the extra features of the TEA* algorithm. Again, taking a look a Figure 5.6, one of the tests realized was having a lesser priority robot staying in node 2 without any order. When a higher robot attempts to go to that node (or

pass through that node), the robot goes to the closest unoccupied station node, which, in this case, is node 21.

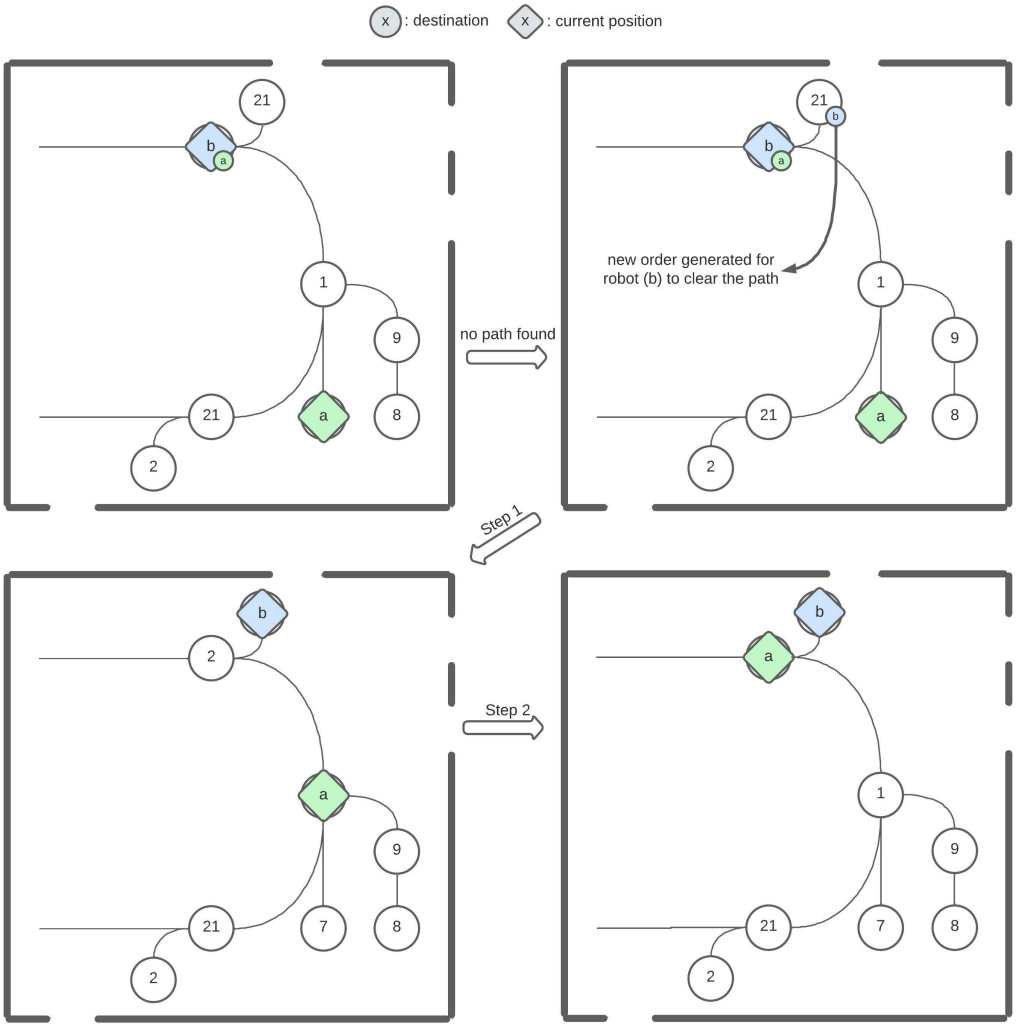



Figure 5.6: An example of the creation of an order for an idle robot to unblock the way for another robot

Other tests were done to examine this priority swapping feature. For example, following Figure 5.7, a lower priority robot is stationed at node 9 and has an order to go to node 2, and there is a higher priority robot in node 7 whose goal is to go to node 1. This case blocks the lower priority robot from ever completing its order, which leads to an attempt to re-order the priorities between the two robots, allowing the new higher priority robot to go to node 2.

The movement of the new higher priority robot blocks the new lower priority robot for a short time but, in the end, allows both robots to complete their orders, proving advantageous. This

situation can be seen in Figure 5.7.

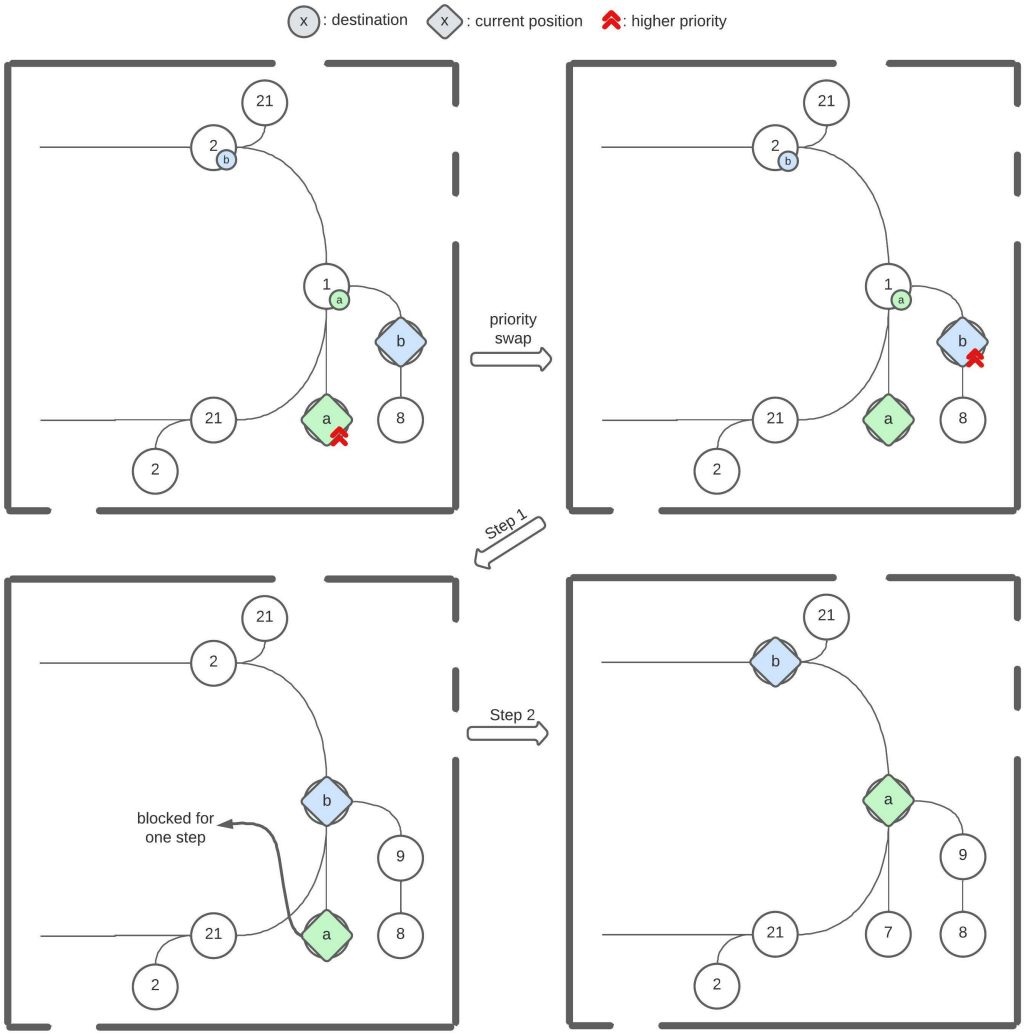



Figure 5.7: An example of a priority swap situation

Regarding the smaller map, one final test was done. This test evaluated the graph decomposition performance by comparing the system's response with and without the graph decomposed. Furthermore, the tests also measured the time taken to plan a path for all robots to estimate the time it takes for a path with a certain number of nodes and for a specific number of robots to be processed and calculated using the TEA* algorithm.

### 5.2.2 Big Map

The bigger map of the two was used to thoroughly test the capabilities of the path planning algorithm, the graph decomposition algorithm and the supervisor's response to the system. The tests

are done using different combinations of the number of robots in the system, the path length to be planned for each robot, and even the robots' heterogeneity. It is possible to evaluate the graph decomposition algorithm's performance using various tests by checking how often the supervisor acts to prevent a possible collision. Furthermore, these various tests are also used to evaluate whether the path planning algorithm and supervisor are working as intended.

The first tests can be summed up by having a specific amount of robots spread across the map and having them follow a complex path, defined by a pre-determined number of nodes, and checking how many times, if any, the supervisor triggers and generates a re-planning of all the robots' paths. These tests were also repeated by simulating the same paths using the graph before applying the decomposition algorithm as a point for comparison. Finally, the time taken to plan the paths for all the robots is also measured for extra conclusions regarding the processing time for path planning.

Because these first tests aim to test the graph decomposition algorithm, the supervisor feature of checking if two robots' paths intercept is disabled, which means that no matter where the robots are if the step difference is higher than 1.0, it will trigger a new path planning. The disabling of this feature is crucial because the goal is to check whether the robots keep up with all the other robots in the system to test the graph decomposition's performance in which their current relative position is irrelevant.

Because these tests were done to test the graph decomposition algorithm's performance thoroughly, they were also used to decide on the best method of obtaining an optimal step used for decomposition. The two methods are the minimization of the maximum error of decomposition and the minimization of the average error of decomposition. Both these methods are a good choice, but only through testing a conclusion can be obtained about the best one.

An example of this complex path for the robots is seen in Figure 5.8.

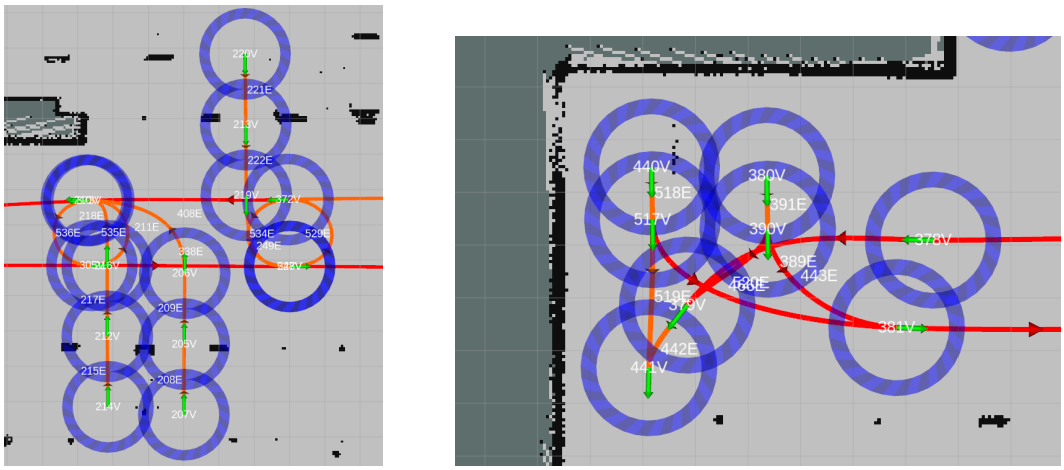



Figure 5.8: Two examples of the complexity of the big map's graph

These tests were repeated at least ten times each using the following parameters:

- Long Path with 2 Robots: 12/56 nodes (12 for undecomposed and 56 for decomposed)

- Long Path with 4 Robots: 12/56 nodes
- Short Path with 4 Robots: 6/25 nodes

Finally, four robots were taken and set on a path successively in a straight line to see the effects of the supervisor triggering a new re-planning in the middle of a path. By having these robots stay at a distance of 1 node between each other and, then, by manually stopping the movement of a robot in that line, the process of the supervisor acting could be seen as the other robots would back off and be blocked by the robot that was interfered. This test was mainly done to demonstrate the supervisor working and ensure it was working as intended. Some videos of this demonstration working can be seen in Annex B along with examples of previous tests.

### 5.2.3 Heterogeneity of the Robots

The last thing to test is the response of the path planning system when coordinating a group of heterogeneous robots. In order to test these, the four robots used were changed to differ in their structure, and traction type and the "long path" tests made previously with homogeneous robots were repeated. Again, these tests' main goal is to test the capabilities of the graph decomposition algorithm when subjected to a group of heterogeneous robots. To accomplish this goal, when decomposing the graph, some parameters had to be changed, namely, the minimum size of the length of an edge.

Because the sizes of the robots change, the minimum length of an edge also has to change to be higher than the new maximum size of a robot in the system. This parameter change can lead to the overall error of decomposition being more significant than the one used in homogeneous robot coordination since the lengthier the minimum length of a link, the higher the error is. This error increase comes from the fact that the more it is possible to divide a link, the lower the error of the decomposition of all links around that value will be and vice versa.

Some videos of these tests can be seen in the Annex B.

## 5.3 Results Obtained and Conclusions

This section will showcase the most prominent results obtained with both the small map and the bigger map, as well as conclusions regarding these results.

### 5.3.1 Small Map

The small map tests mainly tested the features of the TEA* path planning algorithm and the algorithm itself. After various tests, the algorithm implemented could complete all the tasks without failing any of the implemented extra or core features. Some examples of these tests can be seen in Annex B.

After all the features of the path planner were working, some tests were done regarding the decomposition of the graph used in path planning and the time it took to plan a path for all the

robots. Because the small map has very few nodes to travel onto, even when decomposed, only shorter paths were used. These paths are composed of 5 nodes when the graph was undecomposed and 16 when decomposed. The paths are equal in origin and destination nodes but differ in the number of nodes between these two nodes because of the decomposition. The results from this tests can be seen in Tables 5.1 and 5.2. Note that the "xx/yy" values correspond to the "xx" number of nodes used in the path without decomposition and the "yy" number of nodes used in the path with decomposition.

| | 5/16 Nodes - Short Path with 2 Robots | | | |
|---|---|---|---|---|
| | Without Decomposition | | With Decomposition | |
| Test nº | Supervisor Triggers | Time Taken to Plan (ms) | Supervisor Triggers | Time Taken to Plan (ms) |
| 1 | 1 | 2.16 | 0 | 6.86 |
| 2 | 1 | 3.45 | 0 | 6.18 |
| 3 | 1 | 2.23 | 0 | 6.67 |
| 4 | 1 | 2.30 | 0 | 7.33 |
| 5 | 1 | 2.30 | 0 | 5.58 |
| 6 | 1 | 1.60 | 0 | 5.05 |
| 7 | 1 | 1.74 | 0 | 5.49 |
| 8 | 1 | 4.99 | 0 | 5.64 |
| 9 | 1 | 2.40 | 0 | 7.18 |
| 10 | 1 | 4.42 | 0 | 6.13 |
| Average | 1 | 2.76 | 0 | 6.21 |

Table 5.1: Results obtained while testing the path planning system with two robots following a short path composed of 5 nodes when not decomposed and 16 nodes when decomposed

| | 3/5 Nodes - Short Path with 4 Robots | | | |
|---|---|---|---|---|
| | Without Decomposition | | With Decomposition | |
| Test nº | Supervisor Triggers | Time Taken to Plan (ms) | Supervisor Triggers | Time Taken to Plan (ms) |
| 1 | 1 | 2.81 | 0 | 5.96 |
| 2 | 1 | 2.98 | 0 | 12.05 |
| 3 | 1 | 3.01 | 0 | 7.29 |
| 4 | 1 | 2.30 | 0 | 7.15 |
| 5 | 1 | 2.30 | 0 | 9.79 |
| 6 | 1 | 1.60 | 0 | 8.47 |
| 7 | 1 | 1.74 | 0 | 9.42 |
| 8 | 1 | 4.99 | 0 | 8.60 |
| 9 | 1 | 2.40 | 0 | 7.18 |
| 10 | 1 | 4.42 | 0 | 6.50 |
| Average | 1 | 2.85 | 0 | 8.24 |

Table 5.2: Results obtained while testing the path planning system with four robots following a short path composed of 3 nodes when not decomposed and 5 nodes when decomposed

Looking at these two tables, it is possible to conclude that, on a basic level, the decomposition

of the graph used in path planning helps a lot regarding the safety of the robots as it further guarantees that these robots follow the temporal path as expected by the TEA* algorithm. This safety is seen in the number of times the supervisor needs to trigger a new re-planning of the paths for all robots, which is zero for the decomposed map, meaning it is safer than the undecomposed map where the supervisor needs to act in such a small path. However, due to the compact nature of this map, no further conclusions can be made about this, and a larger map is needed to evaluate the decomposition algorithm's capabilities further.

Furthermore, regarding the times taken for planning the paths for all robots, it is possible to conclude that each robot takes about 0.4 *ms* maximum time to explore each node in the path. This maximum time is obtained by dividing the total time needed to obtain the path by the number of robots and nodes used. As such, this calculation does not consider other factors that may increase the time taken to plan all paths.

This time taken to plan the path for various robots varies depending on different factors. For example, suppose two paths are very close to being the most optimal. In that case, the algorithm will run through each node of both paths sequentially until the destination is reached, increasing the time taken to plan that path substantially. Another example can be the effect of any of the extra implemented features, such as the paths needing to be re-explored entirely more than once to find a path by switching the priorities or having to plan the path for a different robot that is idle, blocking the path for another robot. Regardless of these situations, the main goal of this time is to have an idea of the time it takes to plan a path in a normal situation since these special situations do not always happen and can be pretty rare, even though it is still essential that they are kept in mind.

### 5.3.2 Big Map

Regarding the bigger map, this map was mainly used to thoroughly test the graph decomposition algorithm by checking the system's response to it in a more complex and irregular map. Furthermore, because of the extended dimension of the map, it is also possible to take better conclusions regarding the time taken for path planning and the response of the robots regarding the path planned. These better conclusions are mainly due to having paths with a substantially higher number of nodes, even with an increased number of robots in the system.

The first test was regarding which algorithm to decompose the graph used to plan the robot paths. There were two main approaches for this algorithm: the minimization of the average error of decomposition of the graph and the minimization of the maximum error of decomposition of the graph. After several tests, it was concluded that the minimization of the average decomposition error was the better option. This conclusion is expected since minimizing the average will more often reduce the probability of desynchronization between the robot's current step and the planned one and have specific links where the decomposition is a bit worse, which correspond to the links increasing the maximum error of decomposition, leading to possible problems local to some specific links but having a better response in general. On the other hand, minimizing the maximum error would reduce the errors in those specific links but would, in general, make the decomposition

have an overall worse response. With this, the following tests all use the minimization of the average error of decomposition of the graph method.

The results of the tests done with the bigger map are summarized in the Tables 5.3, 5.4 and 5.5.

| | 12/56 Nodes - Long Path with 2 Robots | | | |
|---|---|---|---|---|
| | Without Decomposition | | With Decomposition | |
| Test nº | Supervisor Triggers | Time Taken to Plan (ms) | Supervisor Triggers | Time Taken to Plan (ms) |
| 1 | 0 | 18.56 | 0 | 52.38 |
| 2 | 0 | 20.32 | 0 | 57.24 |
| 3 | 0 | 14.28 | 0 | 58.96 |
| 4 | 0 | 12.67 | 0 | 57.06 |
| 5 | 0 | 21.70 | 0 | 57.47 |
| 6 | 0 | 22.49 | 0 | 47.44 |
| 7 | 0 | 15.45 | 0 | 54.27 |
| 8 | 0 | 14.13 | 0 | 46.70 |
| 9 | 0 | 15.42 | 0 | 58.46 |
| 10 | 0 | 23.11 | 0 | 51.84 |
| Average | 0 | 17.81 | 0 | 54.18 |

Table 5.3: Results obtained while testing the path planning system using the bigger map and a long path with 2 robots

| | 6/25 Nodes - Short Path with 4 Robots | | | |
|---|---|---|---|---|
| | Without Decomposition | | With Decomposition | |
| Test nº | Supervisor Triggers | Time Taken to Plan (ms) | Supervisor Triggers | Time Taken to Plan (ms) |
| 1 | 1 | 27.69 | 0 | 86.24 |
| 2 | 1 | 30.65 | 0 | 66.17 |
| 3 | 1 | 35.90 | 0 | 99.23 |
| 4 | 1 | 22.95 | 0 | 134.95 |
| 5 | 1 | 35.80 | 0 | 106.93 |
| 6 | 1 | 35.64 | 0 | 92.04 |
| 7 | 1 | 30.02 | 0 | 78.16 |
| 8 | 1 | 35.88 | 0 | 117.98 |
| 9 | 1 | 27.66 | 0 | 90.98 |
| 10 | 1 | 31.53 | 0 | 99.70 |
| Average | 1 | 31.37 | 0 | 97.24 |

Table 5.4: Results obtained while testing the path planning system using the bigger map and a short path with 4 robots

| | 12/56 Nodes - Long Path with 4 Robots | | | |
|---|---|---|---|---|
| | Without Decomposition | | With Decomposition | |
| Test nº | Supervisor Triggers | Time Taken to Plan (ms) | Supervisor Triggers | Time Taken to Plan (ms) |
| 1 | 4 | 48.17 | 1 | 146.62 |
| 2 | 4 | 43.56 | 1 | 106.44 |
| 3 | 4 | 34.02 | 1 | 101.48 |
| 4 | 5 | 38.15 | 1 | 135.54 |
| 5 | 5 | 51.55 | 1 | 119.56 |
| 6 | 5 | 34.49 | 1 | 107.05 |
| 7 | 4 | 39.28 | 1 | 113.87 |
| 8 | 4 | 27.40 | 1 | 116.76 |
| 9 | 5 | 30.71 | 1 | 136.60 |
| 10 | 4 | 31.03 | 1 | 137.56 |
| Average | 4,4 | 37.84 | 1 | 122.15 |

Table 5.5: Results obtained while testing the path planning system using the bigger map and a long path with 4 robots

Looking at the tables above, the first conclusion that is possible to have is that, again, the behaviour of the robots regarding the path planned using graphs that are decomposed compared to graphs that are not decomposed is much safer. The tests where the graphs were not decomposed made the supervisor trigger a re-planning of their paths several times. This point further proves that the TEA* path planning algorithm, without decomposing the graph, can be unsafe for the environment in which the system is implemented by leading to problems such as collisions or deadlocks. Although, it is essential to note that even though the supervisor had to trigger new re-plannings several times, all the paths were still able to reach the end-point, concluding that the supervisor was working as intended. Still, regarding the decomposition, it is also possible to conclude that the implemented graph decomposition algorithm could successfully improve the safety and even efficiency of the system to a very stable point. This improvement comes from only having to re-plan the paths once when the path has a substantial amount of nodes and with quite a few robots is a good result.

Regarding the times, using the same calculations in the smaller map, the maximum time to plan a path per node per robot is about 1.3 *ms*. This value significantly increases compared to the value obtained with the short map. This increase is likely due to the increase in the number of nodes' neighbour nodes when they are being explored. Furthermore, the time taken per node per robot is higher in the long path using four robots. This increase could be because of the more complex paths used having an even bigger increase in the number of neighbour nodes per node, along with the increase in the memory used and accessed, which increases the overall time taken to add and remove data from storage containers such as the min-heap used in the path calculation and accessing data from the graph maps.

Still, even though there is an increase in the time taken for path planning, this time is relatively low and very reasonable. With this time, the path planner should be able to re-plan the paths for

all the robots while they are in movement without causing substantial pauses in the movement of the robots, which could be something to consider when re-planning because of the trigger of the supervisor.

Another test was done where 4 robots were set on a path to follow each other in a straight line to verify the system's response when subjected to a supervisor re-planning of the paths. The robots were set in a specific order where the blue robot was at the front, followed by the red, yellow and orange robots in this order. When in movement, one of the robots was manually blocked from moving so it could trigger a new path planning and verify its effects. The test results are shown in the Table 5.6.

| Line Test Blocking Yellow Robot | | | | |
|---|---|---|---|---|
| Distance traveled by: | Blue Robot (m) | Red Robot (m) | Yellow Robot (m) | Orange Robot (m) |
| Decomposed | 22.27 | 22.34 | 23.85 | 24.53 |
| Undecomposed | 22.41 | 22.39 | 24.01 | 26.16 |

| Line Test Blocking Red Robot | | | | |
|---|---|---|---|---|
| Distance traveled by: | Blue Robot (m) | Red Robot (m) | Yellow Robot (m) | Orange Robot (m) |
| Decomposed | 22.29 | 22.91 | 24.22 | 25.09 |
| Undecomposed | 22.56 | 23.13 | 27.54 | 31.00 |

Table 5.6: Effects of supervisor actuation in decomposed vs undecomposed graphs

Looking at these results, it is possible to conclude that the decomposition of the graph not only makes the system safer but also faster since the distance travelled by the robots affected by the supervisor's re-planning of the paths is lower when the graph is decomposed. This is because the robots affected have to take much longer routes if their paths are re-planned when the graph is not decomposed and, as such, the orders take quite a bit longer to complete.

#### 5.3.2.1 Heterogeneous Robots

To further test the capability of the graph decomposition and, in specific, its response with a system composed of heterogeneous robots, extra tests were made. These tests are composed of the same previous tests done in the big map but now with robots with different footprints from 3 $m^2$ to about 0.25 $m^2$, morphologies and even traction types such as differential, tricycle and omnidirectional. The parameters used for the four robots were:

- Robot 1: Length = 0.5 $m$, Width = 1.0 $m$, Traction Type = Differential
- Robot 2: Length = 0.5 $m$, Width = 0.5 $m$, Traction Type = Tricycle
- Robot 3: Length = 1.0 $m$, Width = 1.0 $m$, Traction Type = Differential
- Robot 4: Length = 2.0 $m$, Width = 0.5 $m$, Traction Type = Omnidirectional

In sum, the heterogeneity of the robots affects the system in two major aspects: the size of the robots and their morphologies affect the decomposition parameters, which affects the whole decomposition, and the traction type affects the irregularity of the velocity parameters, making them

have different behaviours when moving, such as slippage or different behaviours when turning. While the first problem can be solved by re-decomposing the graph with the different parameters, the second one cannot be directly solved the same way and must depend on the supervisor to act when needed to ensure that everything is kept safe. With this in mind, the results of the tests done can be seen in Tables 5.7 and 5.8 and some video demonstrations can be seen in the Annex B.

| | 6/25 Nodes - Short Path with 4 Heterogeneous Robots | |
|---|---|---|
| | Without Decomposition | With Decomposition |
| Test nº | Supervisor Triggers | |
| 1 | 2 | 0 |
| 2 | 2 | 0 |
| 3 | 2 | 0 |
| 4 | 2 | 1 |
| 5 | 2 | 0 |
| 6 | 2 | 0 |
| 7 | 2 | 0 |
| 8 | 2 | 0 |
| 9 | 2 | 0 |
| 10 | 2 | 0 |
| Average | 2 | 0,1 |

Table 5.7: Results obtained while testing the path planning system using the bigger map and a short path with 4 heterogeneous robots

| | 12/56 Nodes - Long Path with 4 Heterogeneous Robots | |
|---|---|---|
| | Without Decomposition | With Decomposition |
| Test nº | Supervisor Triggers | |
| 1 | 6 | 2 |
| 2 | 5 | 2 |
| 3 | 5 | 2 |
| 4 | 5 | 2 |
| 5 | 5 | 2 |
| 6 | 5 | 2 |
| 7 | 6 | 1 |
| 8 | 5 | 1 |
| 9 | 5 | 2 |
| 10 | 5 | 2 |
| Average | 5,2 | 1,8 |

Table 5.8: Results obtained while testing the path planning system using the bigger map and a long path with 4 heterogeneous robots

As expected, the system's response is worse when using heterogeneous robots. This response is primarily due to some of the robots being lengthier than the homogeneous ones, which causes the optimal step that the decomposition algorithm uses to compute to be larger, leading to a higher error on the links decomposed. This higher error exists because the bigger the optimal step size,

the worse the decomposition. This is because the more we can decompose a link into smaller links, the lower the error of decomposition will be, and, when we have a larger robot, the minimum step changes to ensure that it is bigger than the length of the robot and, as such, there is an increase of the decomposition error.

These results, however, are still overall good. Keeping in mind that a *SAFETY_GAP* value of 1 was used and that the check for the robots' proximity to be considered in the supervision was removed for these tests, this system, if implemented in a real scenario, would be very likely that with those tests there would be no triggers from the supervisor and the robots would have been kept in a safe temporal step compared to the planned one.

## 5.4 Conclusion

The overall conclusion is that the implemented system successfully coordinated both homogeneous and heterogeneous groups of robots, which was the project's primary goal. The results are pretty positive. Even though the algorithm for the path planning of the robots usually takes a lot of processing time and memory, the system's response was relatively fast. Furthermore, the supervisor and the stability of the implemented graph decomposition algorithm kept the system safe at all times, especially the supervisor, who, even if needed to force a re-planning for the robot's paths, never failed to process the order to the end.

Regarding the number of times that the supervisor had to act, the number itself was relatively low, even taking into consideration that in all these tests, a *SAFETY_GAP* of 1 was used and that the feature of the supervisor only re-planning when two robots are close by is disabled. Using both an increased *SAFETY_GAP* (which is very likely in a real scenario where other problems may arise such as communication faults) and the proximity feature, there would likely be close to no triggers to re-plan paths using the implemented decomposition algorithm.

# Chapter 6

# Conclusions and Future Work

The present chapter will present a brief review of the goals and the results obtained, along with possible future work to be done following the work done regarding this dissertation.

## 6.1 Goals Accomplishment and Result Analysis

The main objective of this dissertation was to develop a system capable of coordinating a group of heterogeneous robots by planning the movement of these robots with safe and efficient paths. A supervisor had to be built to ensure all the robots' movements were synchronized to maintain a temporal efficiency expected by the path planning algorithm and assure safety in their movement. Furthermore, a graph decomposition algorithm had to be implemented to ensure the least possible deviation of the robots' movement by maintaining an equal time taken to travel between each node so that the temporal stability expected when planning the paths using the path planning algorithm was kept as constant as possible. The system also had to use the TEA* graph search algorithm, proposed by [3], to conduct all the path planning necessary to coordinate the robots and use the ROS framework and the C++ programming language to implement all the components required. In addition to the base TEA* algorithm, a few extra features were also implemented to improve the system and move the system closer to the necessities of a real industrial environment.

Looking at the results obtained during the project's test phase, it is possible to conclude various aspects regarding the accomplishment of various goals. The first one is the implementation of the TEA* algorithm, and its features were an overall success. The implemented path planning system was able to coordinate the various robots safely and efficiently. This implementation also proved to be quite fast concerning the time taken to plan the robot paths. Regarding the implemented supervisor, it was able to keep the robots from deviating too much from the expected temporal planned path by the TEA* algorithm by triggering a path re-planning for all the robots in the system to prevent extreme deviations that could lead to collisions or deadlocks.

Regarding the graph decomposition algorithm, the results proved to be quite favourable. The algorithm was able to decompose the graph's links leading to the robot movement deviating way less from the expected path generated by the path planning system. The supervisor had to act

a considerably low amount of times when the graph used to plan the paths of the robots was decomposed. Considering the extreme case the supervisor was subjected to act by removing the feature that only has the supervisor work on a tight step difference when the paths of two robots intercept, these results were pretty good.

These results together make the system quite robust to the heterogeneity of the group of robots. Although the system proved quite effective in a simulation environment, it was impossible to test it in a real scenario due to time constraints. As such, it is hard to verify that the system would be able to work as expected using real robots. Furthermore, although considerably efficient and safe, the paths obtained by the implemented system are still not fully optimal. There are still some minor safety concerns, as shown in the Chapter 3. Finally, the system also suffered from some common vulnerabilities of a centralized system, such as having a single point of failure, possible communication failures and a significant lack of scalability to an extensive group of robots.

## 6.2 Future Work

As mentioned, there are still some factors that can badly affect the capabilities of the implemented system to be completely safe, such as having paths from two graphs that intercept or run very close to each other, which the TEA* algorithm does not take into account. Some approaches for this problem are in current development by analyzing the robot's footprint and an exception table to be used when planning the paths.

Additionally, it was not possible to test the system in a real environment using real robots. It would be interesting to check how the graph decomposition algorithm performs when presented with the instability of real robots in real environments. These could change some aspects of the graph decomposition or even the path planning.

Another relevant point that could be explored is that the paths generated, although quite efficient, are not entirely optimal, especially if the goal is that all the robots complete their paths as fast as possible. One of the reasons is that using a single priority for a robot from the beginning to the end of a path could lead to less optimal paths being found. This is because the higher priority robot would block the other robot's path or even make it take a longer route in which, if the priorities were swapped, the paths obtained would be optimal. An idea to optimize these paths could be finding contention path sections and testing the swap of priorities or all robots contesting for that section until paths that minimize the total time taken are obtained.

# Appendix A

# Class Diagram

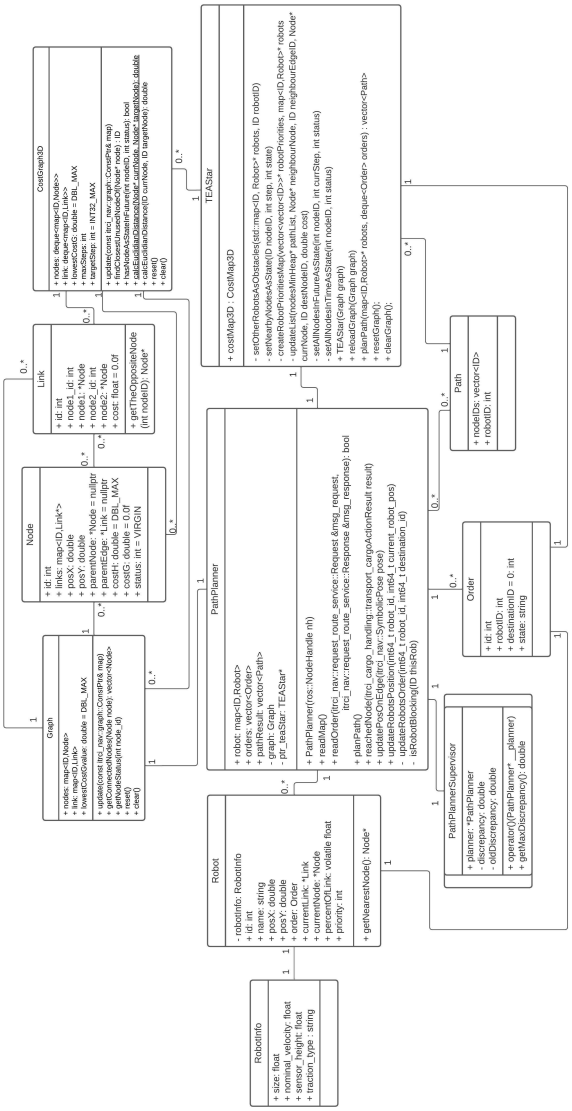



Figure A.1: The path planning central system's class diagram

# Appendix B

# Videos Taken

Video of basic robot coordination tests done:

- Link: https://youtu.be/EEKb_r0efJY

Video of supervisor tests done:

- Link: https://youtu.be/8qBK5eCUi4o

Video of heterogeneous robot coordination tests done:

- Link: https://youtu.be/RDjN2HEko40

# References


[1] Pedro Costa. Planeamento cooperativo de tarefas e trajectórias em múltiplos robôs, 2011. URL: https://hdl.handle.net/10216/62107.

[2] Jérôme Barraquand and Jean Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10, 1991. doi:10.1177/027836499101000604.

[3] Joana Santos, Pedro Costa, Luís F. Rocha, A. Paulo Moreira, and Germano Veiga. Time enhanced A* : Towards the development of a new approach for multi-robot coordination. volume 2015-June, 2015. doi:10.1109/ICIT.2015.7125589.

[4] Roland Geraerts and Mark H. Overmars. A comparative study of probabilistic roadmap planners. volume 7 STAR, 2004. doi:10.1007/978-3-540-45058-0_4.

[5] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. URL: https://books.google.pt/books?id=-PwLBAAAQBAJ&lpg=PT9&ots=0iCy2srnlt&lr&pg=PT785#v=onepage&q&f=false.

[6] Pedro Moura, Pedro Costa, José Lima, and Paulo Costa. A temporal optimization applied to time enhanced A*. volume 2116, page 220007, 2019. URL: http://aip.scitation.org/doi/abs/10.1063/1.5114225, doi:10.1063/1.5114225.

[7] Ángel Madridano, Abdulla Al-Kaff, David Martín, and Arturo de la Escalera. Trajectory planning for multi-robot systems: Methods and applications, 2021. doi:10.1016/j.eswa.2021.114660.

[8] Diogo Matos. Multi agv coordination tolerant to communication failures. *Robotics*, 10, 2021. doi:10.3390/robotics10020055.

[9] Cláudia Rocha, Ivo Sousa, Francisco Ferreira, Héber Sobreira, José Lima, Germano Veiga, and A. Paulo Moreira. Development of an autonomous mobile towing vehicle for logistic tasks. In *4th Iberian Robotics Conference, ROBOT 2019*, pages 669– 681, 2020. doi:10.1007/978-3-030-35990-4_54.

[10] Peter Stone and Manuela Veloso. Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots*, 8, 2000. doi:10.1023/A:1008942012299.

[11] Gregory Dudek, Michael R.M. Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3, 1996. doi:10.1007/BF00240651.

[12] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10, 2013. doi:10.5772/57313.

[13] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 1, 2000.

[14] Jens Wawerla and Richard T. Vaughan. A fast and frugal method for team-task allocation in a multi-robot transportation system. 2010. doi:10.1109/ROBOT.2010.5509865.

[15] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. Probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8, 2000. doi:10.1023/A:1008937911390.

[16] Amanda Prorok, Alexander Bahr, and Alcherio Martinoli. Low-cost collaborative localization for large-scale multi-robot systems. 2012. doi:10.1109/ICRA.2012.6225016.

[17] Ana Cruz. Multi agv communication failure tolerant industrial supervisory system, 2021.

[18] Pedro Moura. Coordenação de multi-robots num ambiente industrial, 2018.

[19] Qin Yang and Ramviyas Parasuraman. Needs-driven heterogeneous multi-robot cooperation in rescue missions. 2020. doi:10.1109/SSRR50563.2020.9292570.

[20] Y. Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4, 1997. doi:10.1023/A:1008855018923.

[21] Changyun Wei, Koen V. Hindriks, and Catholijn M. Jonker. Altruistic coordination for multi-robot cooperative pathfinding. *Applied Intelligence*, 44, 2016. doi:10.1007/s10489-015-0660-3.

[22] Chengbao Liu, Jie Tan, Hongsheng Zhao, Yaning Li, and Xiwei Bai. Path planning and intelligent scheduling of multi-agv systems in workshop. 2017. doi:10.23919/ChiCC.2017.8027778.

[23] P. Trodden. Robust distributed control of constrained linear systems., 2009.

[24] Michael Erdmann and Tomás Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2, 1987. doi:10.1007/BF01840371.

[25] James Stephan, Jonathan R. Fink, Benjamin Charrow, and Alejandro Ribeiro. Hybrid decentralized control system for communication aware mobile robotic teams. 2014.

[26] Wenju Mao, Zhijie Liu, Heng Liu, Fuzeng Yang, and Meirong Wang. Research progress on synergistic technologies of agricultural multi-robots. *Applied Sciences (Switzerland)*, 11, 2021. doi:10.3390/app11041448.

[27] Mostafa Moussid, Adil Sayouti, and Hicham Medromi. Development of intelligent hybrid architecture for autonomous uav. *Int. J. of Adv. Res*, 2017. doi:10.21474/IJAR01/3196.

[28] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 57, 2010. doi:10.1007/s10846-009-9383-1.

[29] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20, 2001. doi:10.1177/02783640122067453.

[30] S. M. Yang and Y. A. Lin. Development of an improved rapidly exploring random trees algorithm for static obstacle avoidance in autonomous vehicles. *Sensors*, 21, 2021. doi:10.3390/s21062244.

[31] Luca Iocchi, Daniele Nardi, Maurizio Piaggio, and Antonio Sgorbissa. Distributed coordination in heterogeneous multi-robot systems. *Autonomous Robots*, 15:155–168, 01 2003. doi:10.1023/A:1025589008533.

[32] P. Caloud, Wonyun Choi, J.-C. Latombe, C. Le Pape, and M. Yim. Indoor automation with many mobile robots. In *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, pages 67–72 vol.1, 1990. doi:10.1109/IROS.1990.262370.

[33] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12):399, 2013. doi:10.5772/57313.

[34] Wenjie Wang and Wooi-Boon Goh. Multi-robot path planning with the spatio-temporal a* algorithm and its variants. In Francien Dechesne, Hiromitsu Hattori, Adriaan ter Mors, Jose Miguel Such, Danny Weyns, and Frank Dignum, editors, *Advanced Agent Technology*, pages 313–329, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:0.1007/978-3-642-27216-5_22.

[35] K. Petrinec and Z. Kovacic. The application of spline functions and bezier curves to agv path planning. In *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, volume 4, pages 1453–1458, 2005. doi:10.1109/ISIE.2005.1529146.