FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Multi AGV Communication Failure Tolerant Industrial Supervisory System

Ana Sofia Poças da Silva Cruz

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Pedro Gomes da Costa

Co-supervisor: Paulo Gomes da Costa

June 30, 2021

Resumo

Automated Guided Vehicles (AGV) são robôs que têm vindo a assistir cada vez mais operações em ambientes fabris. Utilizados tanto no transporte de materiais em zonas industriais como no transporte em grandes armazéns, o uso destes robôs revela-se uma vantagem relativamente ao aumento de produtividade e eficiência.

Desta forma, o trabalho desenvolvido visa a implementação de um sistema capaz de coordenar uma frota de robôs AGV, planeando e coordenando os caminhos de cada robô, mas apresentando tolerância a situações críticas, capazes de desencadear colisões e *deadlocks*.

A abordagem escolhida baseia-se num trabalho previamente desenvolvido que foi testado e melhorado conforme a dinâmica de um ambiente real.

O algoritmo responsável pelo planeamento dos caminhos dos robôs, evitando colisões e *deadlocks*, é denominado de *Time Enhanced A** (TEA*). Como apresentado na revisão bibliográfica, é, normalmente, utilizado como um método online, re-planeando os caminhos dos robôs constantemente, de forma a possuir sempre informação atualizada relativa ao ambiente.

Para evitar o re-planeamento constante, visto que também se revela computacionalmente pesado, foi proposto um sistema supervisor. Este sistema supervisor modular é responsável por detetar alterações no ambiente dos robôs e permitir o re-planeamento quando necessário. O sistema supervisor implementado foca-se nas situações de atrasos e dessincronizações entre os robôs e em falhas de comunicação.

Para os dois tipos de situações mais críticas, o módulo de supervisão está dividido em dois submódulos: *Planning Sub-Module*, responsável por verificar se os robôs seguem o caminho planeado pelo TEA* nos tempos corretos (procurando assim atrasos e dessincronizações) e o *Communication Sub-Module*, responsável por detetar e gerir falhas de comunicações.

O trabalho desenvolvido previamente foi testado em robôs AGV de pequena escala. O comportamento do *Planning Sub-Module* foi avaliado e adaptado às condições reais. O *Communication Sub-Module* foi também testado induzindo, artificialmente, falhas de comunicação em zonas específicas e analisando a deteção, localização e a estimação das falhas pelo supervisor.

Através da implementação do módulo supervisor, é permitida uma segurança extra nestas situações descritas. No entanto, as falhas de comunicação continuam a representar uma ameaça à segurança das trajetórias dos AGV. Assim, foi avaliado que as zonas de falhas deveriam de ser evitadas depois de detetadas. Para cumprir este objectivo, o algoritmo TEA* teve de ser modificado de forma a escolher, como caminho ótimo, um caminho sem falhas de comunicação, sempre que possível.

Por fim, foi avaliado um possível caso crítico em relação a falhas de comunicação. Uma vez que, no fim das suas missões, os AGV deslocam-se para uma estação de descanso préviamente atribuída, no pior dos casos, essa estação pode se apresentar numa zona sem comunicação. Desta forma, o robô ficaria preso nessa estação indefinidamente, pois a central de controlo não conseguiria enviar novos comandos. Esta situação foi estudada e uma solução foi desenvolvida e testada.

Abstract

Automated Guided Vehicles (AGV) are robots that have been growing in industrial environments. Used to assist the movement and transportation of items in manufacturing facilities or in warehouses and distribution centres, these systems represent a significant increase in efficiency and productivity.

Thus, the developed work aims to implement a system capable of coordinating a fleet of AGV, planning and coordinating each robot's path but exhibiting tolerance in critical situations that can lead to collisions and deadlocks.

The chosen approach is based on previously developed work that was tested and improved and adapted to a real environment's dynamic.

The algorithm chosen, responsible for the planning of the robot's path, avoiding collisions and deadlocks, was the Time Enhanced A* (TEA*). As presented in the literature, it is usually implemented as an online method, re-planning the paths every cycle to be able to have updated paths even with changes in the environment.

To avoid the constant re-calculation of the paths, since it can reveal to be computationally heavy, a supervisory system was proposed. This modular supervisory system is responsible for detecting changes in the robot's environment and allow the re-calculation of the paths, if necessary. The supervisor implemented searches for delays and desynchronisations on the robots and communication failures.

For these two critical situations, the supervisor is divided into two sub-modules: Planning Sub-Module, which is responsible for checking if the robots follow the computed trajectory, by the TEA*, within the correct time (searching for delays and desynchronisations) and the Communication Sub-Module, responsible for detecting and dealing with communication faults.

The work developed previously was tested in small-scale AGV robots. The behaviour of the Planning Sub-Module was evaluated and adapted to the real world's conditions. The Communication Sub-Module was also assessed by inducing, artificially, communication faults in specific areas and analysing their detection, localisation and estimation (a procedure done by the supervisor).

Through the implementation of the supervisor module, it is granted additional security in the situations described. However, communication faults still reveal to be a threat to the safety of the AGV's trajectories. Therefore, it was assessed that communication faults' areas should be always avoided after being detected. To reach that goal, the TEA* algorithm was modified to choose, as the optimal path, a path free of communication faults, every time it is possible.

Lastly, it was analysed a possible critical scenario concerning communication faults. When a robot completes its mission, the AGV retreats into their designated rest station. In the worst-case scenario, that rest station is inside a communication fault's area and the system is not aware of it. Therefore, the robot would be stuck in its rest station, indefinitely, since the central control would not be able to communicate and send new commands to the robot. This situation was studied and

a solution was developed and tested.

Acknowledgements

The success of this project would have not been possible without all the support and help from my supervisor, professor Pedro Costa, my co-supervisor, professor Paulo Costa, professor José Lima and PhD student Diogo Matos. The regular monitoring and encouragement were crucial to motivate me and help this project go further.

I am deeply grateful for all the patience and comprehension and, above all, all the love given by my parents, not only shown during this intense semester but also during all my academic path. All my success is theirs. It was with their love and approval that they saw their house becoming a laboratory to allow this project to be developed in the middle of a pandemic. I will be forever thankful for that. And mostly, I want to thank them for believing in me even when I did not believe in myself.

I would like to give special thanks to my sisters, Rita and Mariana, for all the support and humorous moments and nicknames given to my robots. I will always cherish those moments.

I would also like to thank Luís Pires for all the love and support shown, not only in the good moments but also during the bad times. Thank you for always listening to me and believing in me.

Finally, I would like to thank all my friends, especially Bruno Gonçalves and Hélder Pereira, for all the tough moments shared and mutual support, encouraging words and laughter shared. Without them, this path would have been lonelier and tougher.

Ana Cruz

"Efforts and courage are not enough without purpose and direction"

John F. Kennedy

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AGV	Automated Guided Vehicle
AGVS	Automated Guided Vehicle System
D*	Dynamic A*
D*LR	D* Lite with Reset
LPA*	Lifelong Planning A*
TEA*	Time Enhanced A*
UDP	User Datagram Protocol
VD	Voronoi Diagram
VFF	Virtual Force Field
VG	Visibility Graph
VFH	Vector Field Histogram

Chapter 1

Introduction

The present document explores the development of the dissertation in the scope of the master's degree in Electrical and Computer Engineering, in the field of Robotics and Systems.

In this first chapter, the context of the problem and what motivated the work developed are described for a better understanding of the environment in which all the work was developed.

It is also explained the main objectives proposed and studied throughout the development of this work.

Lastly, the structure of the dissertation is briefly explained for a better comprehension of it.

The majority of the work was developed through the development environment, Lazarus in Free Pascal language. The computer vision algorithm was developed using Python and tested using a Raspberry Pi.

1.1 Context

Industries of all branches and areas of production share the goal of increasing workflow efficiency on factory floors and in distribution centres. To fulfil this demand, automated guided vehicle systems (AGVS), a flexible automatic means of conveyance, have become a key element in industries' intralogistics, with more industries making use of AGVS since the mid-1990s [10].

With automated guided vehicles (AGV), almost any load can be transported. These systems can assist to move and transport items in manufacturing facilities, warehouses, and distribution centres without any permanent conveying system or manual intervention. It follows configurable guide paths for optimisation of storage, picking, and transport in the environment of premium space [11].

The use of AGV represents a significant reduction of labour cost, an increase in safety, and a sought-after increase in efficiency [12]. These systems are also programmed to take over repetitive and fatiguing tasks that could diminish the human worker attention and lead to possible accidents. Therefore, industries that use AGV can reduce these accidents significantly and, with a 24 hours per day and 7 days per week operability and high production output, AGV ensure worker safety while maximizing production [13].

Even though AGV are commonly found in industrial environments, other sectors (such as hospitals) have been taking advantage of these systems to automate their processes and tasks. One of the main advantages of AGVS is their modularity, meaning that easily more AGV can be added to the fleet: this is sometimes called a “modular system element” [12]. This also represents an advantage when it comes to the initial investment.

Multiple robot systems can accomplish tasks that no single robot can accomplish, since a single robot, no matter how capable it is, is spatially limited [14]. Nevertheless, a multi-AGV environment requires special attention.

Coordinating a fleet of AGV is already a complex task and restrict environments with the possibility of exposing the AGV to delays in the trajectory and communication faults can represent a threat, compromising the safety, productivity and efficiency of these systems. To solve this, trajectory planning algorithms allied with supervisory systems with communication faults detection and mitigation have been studied and developed.

1.2 Motivation

The use of multi AGV implies optimisation of traffic control. Several approaches focus on a trajectory planning method that guarantees efficient and safe coordination of multi AGV. However, many fail to detect, treat and prevent the possible failure and delay in the communication between the AGV and the control platform. These faults can result in possible deadlock situations and collisions.

In environments where communication faults are common, a decrease in efficiency may happen. Thus, this document intends to address the communication problems resorting the Time Enhanced A* (TEA*) algorithm to avoid deadlocks and collisions and optimise environments with communication faults.

1.3 Objectives

The aim of this project is to implement, test and further develop the work initiated by [9]. Therefore, the focus of this thesis is a supervisory system that is able to control the traffic of a fleet of AGV by detecting communication faults, delays in communication, deviations in the routes calculated by the TEA* algorithm and triggering the re-calculation of the trajectories, if necessary.

To implement and test the work of [9], it is intended to develop and assemble small scale AGV to study the dynamic of the environment and how the developed supervisor reacts to the changes of the environment. Based on those results, it is desired to adapt the supervisory system to a real environment and improve its performance.

To obtain the optimal paths for the robots to execute their missions, the system relies on the TEA* algorithm, a graph search algorithm based on the A* with time notion. This algorithm will be used to keep the efficiency and allow time optimisations.

When communication faults are detected by the supervisor, those areas are located and estimated. To increase the safety of the routes chosen, it is planned to reduce the number of times a robot crosses the communication fault's area. Even if those paths may reveal to be the fastest, they cannot be considered optimal if communication between the central module and the robot is lost. Loss of communication reduces the safety and the efficiency of the paths and may expose the robots to deadlocks and collisions.

To achieve that feature, modifications on the TEA* algorithm may be considered so that the resulted paths only cross faulty areas as the last available option.

This work is also focused on studying communication faults (if they are temporary or persistent, how two different faults may be connected, etc) and assessing extreme situations such as having a robot completing its mission inside a fault and not being able to reestablish communication with the central module.

1.4 Document's Structure

The body of this dissertation is constituted by the current introductory chapter and other five chapters.

The following chapter, Chapter 2 contains a compilation of work developed in the scope of the area of this dissertation. The work reviewed in that chapter revealed to be important for the development of this project.

In Chapter 3, it is presented the proposed approach with a description of the modules that constitute the system and how they interact with each other. It is also presented the key elements to be able to implement the work, initially developed by [9] in a simulator, in a real environment. Lastly, the low level control is tested and the results are analysed and discussed.

The next chapter, Chapter 4, focuses on the development of the Supervisor Sub-Module: Planning Sub-Module and Communication Sub-Module. It is described, for each sub-module, their development, the tests performed and the conditions in which they were performed, the results obtained and the discussion and interpretation of them.

Chapter 5 presents an approach for a specific but critical problem that may happen when the system is dealing with communication faults. That approach is explained in detail and tested in multiple situations. The results are analysed and discussed.

Lastly, Chapter 6 concludes this document by summarizing the work developed, the results accomplished and some domains that should be explored and addressed in the continuation of this work.

Chapter 2

Literature Review

Automated Guided Vehicles have to satisfy all safety requirements while working reliably to be cost-effective and gain industry acceptance [15]. To be considered a viable option, these systems need to present a high communication fault tolerance to promote cooperative behaviour in a multi-robot environment.

2.1 Control Architectures

When it comes to the structure of a multi AGV system, the control architecture is the primary element and crucial to determine the capabilities, as well as the limitations, of the system. The mechanism of cooperation of the AGV fleet is directly linked with the design of the control and communication structure [14].

One of the key features of a group architecture lays in the decision of it being a centralised or decentralised model. The main divergent point of these two architectures is the level of autonomy granted to the AGV to choose its route.

2.1.1 Centralised Control Architecture

Most practical multi AGV rely on the centralised control architecture where one central unit concentrates the planning of the strategies of the AGV (task scheduling, path planning, and motion coordination) and communication between robots is non-existent [16]. The central unit is responsible for communicating with each AGV, observing their positions, calculating motion plans, and transmitting control actions.

This architecture can be divided into coupled or decoupled approaches. Coupled approaches treat the whole system as a composite system. In this way, a single-vehicle motion planning algorithms can be applied. This has the propriety of completeness and the possibility of calculating optimal motion plans but is highly demanding in terms of computational resources such as the time required to solve a coordination problem (as the number of vehicles increases the computational load increases exponentially) [17].

Decoupled approaches reduce computational complexity and solve the coordination problem by dividing it into two phases: path planning and motion coordination. Typical approaches are prioritised planning and path coordination [17].

Prioritised planning is based on a sequential calculation of paths in priority order, considering higher priority AGV moving along their path as moving obstacles that must be avoided. Oppositely, path coordination methods, firstly, calculate independent paths for each robot, without considering the others and then adjust each AGV to avoid collisions. Decoupled planners are generally faster than coupled planners, but cannot guarantee completeness and may be sub-optimal. Other centralised coordination methods classified as decoupled include methods based on zone control [18], time windows [19], and multi-agent systems [20].

2.1.2 Decentralised Control Architecture

Considering that centralised control has computational limitations, high communication demands, and low tolerance to faults, there is a high demand for efficient decentralised control methods. In this type of control (also referred to as distributed control), each robot is an agent that can independently plan its path, make its own motion decisions and communicate with other robots [17]. This grants flexibility, scalability and fault tolerance, but cannot guarantee optimal performance [16]. The design of decentralised algorithms must also ensure collision avoidance and deadlock prevention.

In decentralised control architecture, the strategies used can also be divided into prioritised planning and path coordination.

An example of decentralised control architecture using the priority system is described in [21]. Each vehicle plans its trajectory, dynamically, considering other vehicles as static obstacles and a reactive method for obstacle avoidance is used: the information about its state in the environment is broadcast, and by combining all these states locally, each AGV decides which action to take. However, the resulted paths can be highly sub-optimal. It is also needed to take into account that the efficiency of prioritised planning based algorithms depends mainly on the order in which the priorities are defined. So, to increase efficiency, adaptive priority reassignment can be implemented. This can be particularly useful when a robot cannot find a path to complete its task. An increase of priority of said robot could solve the problem.

Concerning the path coordination method, as mentioned before, each robot calculates independently its path without considering the others, and to avoid collisions many strategies can be adopted. When two or more robots overlap trajectories, it is important to detect when and where the conflicts will occur.

To achieve this, the velocity profile of the robots is considered and a temporal based scheduling of the robot position is obtained. Consequently, the locations and time instances of the collisions can be calculated [22]. After this, it is possible to implement prevention strategies. One of them is the implementation of priorities on the crossing points between paths. When defining priorities, a common solution is a negotiation between the robots or the intervention of an external entity. Another strategy is adapting the robot's velocity to avoid collision.

However, in practice, many systems do not conform to a strict centralised/decentralised dichotomy [14] mainly because the control architecture implies a certain path planning technique.

2.1.3 Control Architecture and Path Planning

Taking into consideration the type of control architecture, centralised, decentralised or hybrid, the chosen path planning method needs to be suitable. Therefore, the path planning adopted by the system can be divided into three types: global path planning, local path planning and hybrid path planning.

The global path planning aims for a globally optimal and collision-free path in many types of "map" (Voronoi diagram [23], regular grid, Quad-tree, Roadmap [23], Visibility Graph [1]) which are extracted from the environment space, C_{space}, or other environmental information.

Based on the map, several path finding algorithms can be employed to find the path. Typically, the search algorithm includes graph searching algorithms (such as A* [24], D* [25], D* lite [26]), intelligent optimisation algorithms (such as Genetic algorithm [27], Ant Colony system [28]) and sampling-based algorithms (such as [23] and [29]).

However, most of them are time-consuming and full knowledge of the environment is needed. Consequently, when executed offline (which most of them are due to the computational power required), their adaptability to the changing of environment's information and the motion error of the robot is limited [30].

Local path planning methods, in general, are efficient enough to be executed online and are capable of dealing with both the motion error and the change of environment's information. However, they cannot guarantee that the destination is reached due to the lack of global information. Some of these methods are the Virtual Force Field (VFF) [31] and Vector Field Histogram (VFH) [32] (which combine the potential field method [33] and certainty grid-based world modelling), fuzzy logical control methods (such as [34]), that guide the robot according to human driving experience and are easier to comprehend, and the dynamic window approach [35], that controls the motion with both obstacles and the driving ability of the robot considered, being capable of guaranteeing a feasible and safe motion of the robot [30].

The hybrid path planning approaches combine both types of approaches to avoid their drawbacks while keeping the advantages. They can be further divided into two types based on the combination method: sub-target based approaches, where local path planning is guided by the sub-target chosen from the global path (for example, [35]); behaviour based approaches, where the global behaviour incorporates the local behaviour into a final behaviour (for example, [36]) [30].

Hybrid path planning algorithms obtain moderate adaptability from the local path planning part. However, they are restrained on local scale because of their offline planning model of the global path planning part [30].

All three types of path planning approaches may be used in a partially unknown environment. For such environment, because of the limited dependence on environmental information and its online planning model, local path planning provides good adaptability. In the case that either

global environmental information and local information are not available, local path planning is an appropriate choice [30].

On the other hand, most global path planning methods suffer from their limited adaptability in partly unknown environments. It is mainly because they need to be able to re-plan as fast as the environment changes and that is difficult to achieve. To solve this issue, some algorithms have been developed such as D* lite, whose path is based on a previous planning result instead of starting a new planning process, which accelerates the re-planning. Hybrid path planning is also an attractive manner to solve the planning problem in such environment [30].

2.2 Path Planning

The system performance is highly connected with path planning and trajectory planning. Path planning describes geometrically and mathematically the way from the starting point to the destination point, avoiding collisions with obstacles.

On the other hand, trajectory planning is that path as a function of time: for each time instance, it is defined where the robot must be positioned [4].

To describe the path for each robot to follow, firstly, it is necessary to define the environment space where the robot is: C_{space}. The free space from C_{space} (the space that is not occupied by obstacles) is called C_{free} and it is the area where the safe paths will be created. In opposition, the space unavailable due to obstacles is referred to as $C_{obstacle}$ [4]. It should also be taken into consideration that $C_{obstacle}$ may also contain a margin of free space around the obstacles to create a safety gap.

2.2.1 Path Planning Methods

When choosing a path planning method, several aspects have to be considered. For example, the type of intended optimisation: it might be intended to optimise the path length or the time it takes to perform the trajectory (the trajectory execution time), the energy consumed or other aspects [4].

The computational complexity is another main aspect to take into account as many of the methods are purely theoretically and not possible to implement due to insufficient memory or because the execution time is extremely high [4].

The method can also be complete, if it always finds a solution when it exists, in full resolution, if there is a solution to a particular discretisation of the environment, or probabilistically complete if the probability of finding a solution converges to 1 as the time tends to infinity [4].

Due to the nature of the work to develop, the following sections will only focus on global algorithms that result in graphs.

2.2.1.1 Roadmap

The Roadmap algorithm attempts to capture the free space connectivity with a graph where nodes and links may have physical meaning (the nodes may represent a location and the links the path between these locations) [4]. This method boils down to a research problem using a graph.

Some popular approaches to build the roadmap are based on computational geometry structures. The Visibility Graph (VG) is used to obtain the shortest path and the Voronoi Diagram (VD) is preferred for a maximum clearance path [23].

The Visibility Graph method is applied to a 2 dimensional C_{space} with polygonal $C_{obstacle}$ spaces. The roadmap is built by connecting every pair of vertices in the set of $C_{obstacle}$ spaces by a line segment that cannot intersect any $C_{obstacle}$. The main idea is to construct a path as a group of polygonal lines (the VG) connecting the robot's initial point to its destination through vertices of the $C_{obstacle}$ spaces.

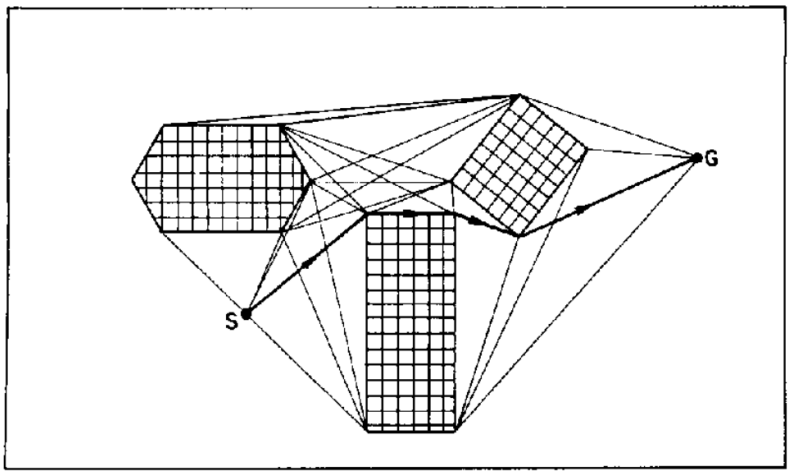


Figure 2.1: Example of a Visibility Graph [1]

In the Figure 2.1, it is considered a moving point A from position S to position G while avoiding the obstacles (shaded areas). It is also possible to observe the shortest collision-free path from S to G. Through the figure, it is possible to verify the important property described previously: that the path is composed only by straight lines joining the initial point to the destination via a possibly empty sequence of obstacle's vertices.

This methodology is one of the earliest path planning methods and it has been widely used to implement path planners for mobile robots since it has been proved that VG guarantees the shortest path [37]. However, it is difficult to compute an efficient path planning for an environment with complicated obstacles (the computation time increases with the number of obstacles' vertices) [38] [1].

On the other hand, the Voronoi Diagram is composed of a set of points that are equidistant from two or more $C_{obstacle}$. The C_{space} is divided into regions and in each region, there is only one obstacle. Inside that region, any point is closer to the inside obstacle than to any other [4].

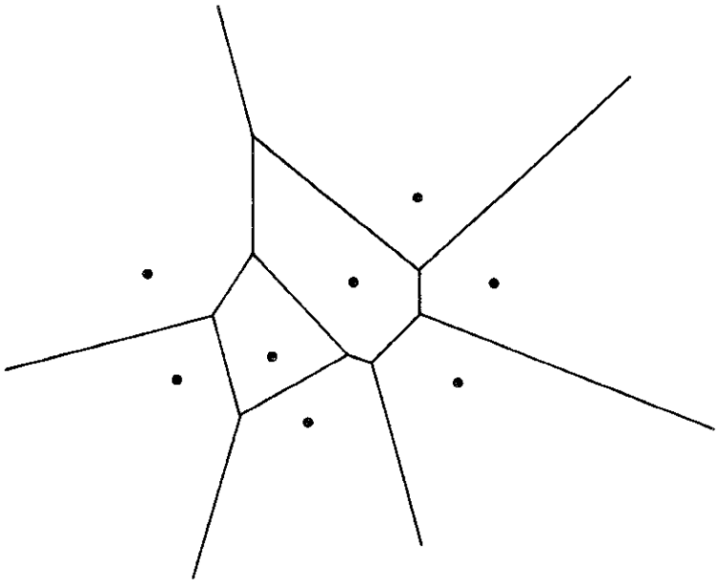


Figure 2.2: Example of a Voronoi Diagram [2]

To find the path between the initial and final points, firstly it is needed to connect the initial point to the VD (Figure 2.2) and then find in the diagram the best path. Lastly, it is necessary to connect the diagram to the final point.

Unlike the Visibility Graph, any path generated by the Voronoi Diagram is too far away from obstacles. This also prevents Voronoi Diagrams from giving minimal paths between points.

2.2.1.2 Cell Decomposition

In this approach, the C_{space} is divided into cells and it is followed by the computation to identify if the cell is free or not and connect it to neighbouring cells. Through this procedure, a graph is created. The path is found through graph search.

The steps of this algorithm consist of allocating the initial point to a cell and the destination point to another. Afterwards, a sequence of cells is found so that the initial and destination points are connected.

Two approaches of this method are:

- Exact Cell Decomposition
- Approximated Cell Decomposition

Exact Cell Decomposition The Exact Cell Decomposition describes accurately the real world by clearly distinguish cells that represent C_{free} (free space) and $C_{obstacle}$ (occupied space). The cells generated by the decomposition should have a simple geometry so that a path between any two configurations in the cell can be easily computed. Also, it should not be difficult either to test

the adjacency of any two cells or to find a path crossing the frontier shared by two adjacent cells [39].

When it comes to the shape of the cells, the most common are the decomposition using convex polygons and the decomposition based on trapezoids.

In the decomposition using cells shaped as convex polygons, the cells' vertexes correspond to the obstacles' vertexes. The links represent adjacent cells and intermediate points of the cells' frontiers are used for path planning [4].

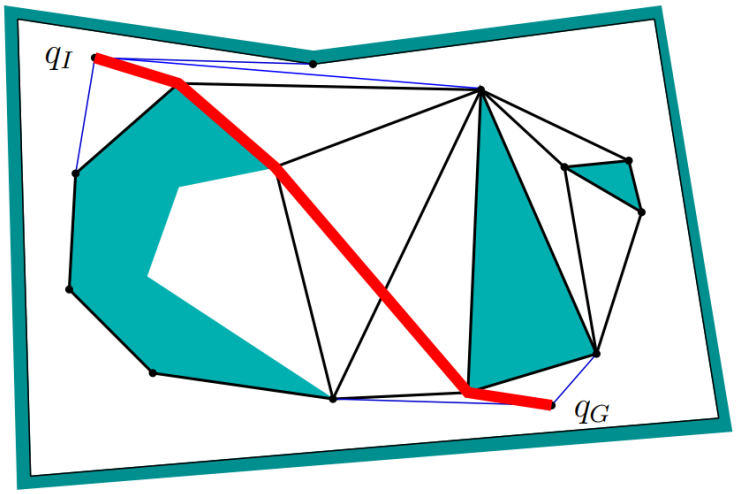


Figure 2.3: Example of Convex Polygons Decomposition [3]

In Figure 2.3, it is possible to observe the shortest path between point q_I and point q_G using the Convex Polygons Decomposition.

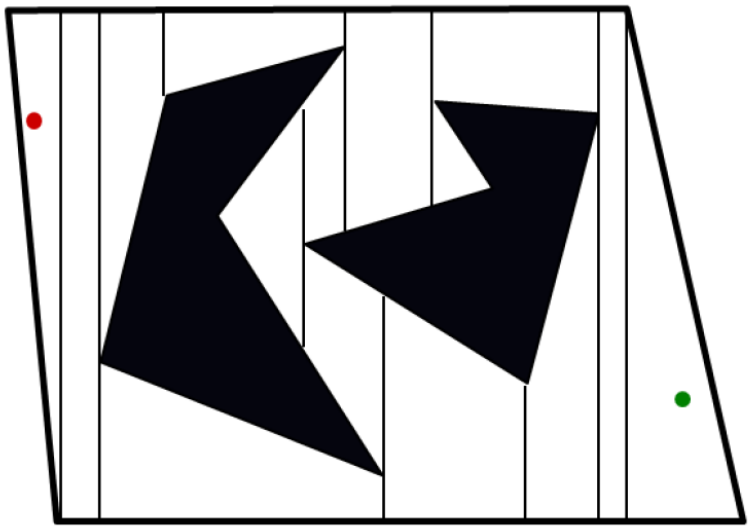


Figure 2.4: Example of a Trapezoidal Decomposition [4]

In trapezoidal decomposition, each obstacle's vertex contains a vertical line. Those vertical lines compose the cells' edges. Figure 2.4 illustrates the decomposition with two obstacles.

Approximated Cell Decomposition Approximated Cell Decomposition relies on cells that can be identified as free (inserted in C_{free}), occupied (inserted in $C_{obstacle}$) or semi-occupied (inserted in both C_{free} and $C_{obstacle}$). Because of this nomenclature, this method does not represent the exact real space. The cells tend to have simpler shapes, such as square, that allows being faster and simpler to configure the C_{space}. However, one of the drawbacks is that the path may not be found even if it exists. It all depends on the chosen cells' size. Overall, the error between the real world and the decomposition lies in the cells' size [4].

In 2D spaces, the most used forms of this decomposition are Fixed Size Cell and Quadtree.

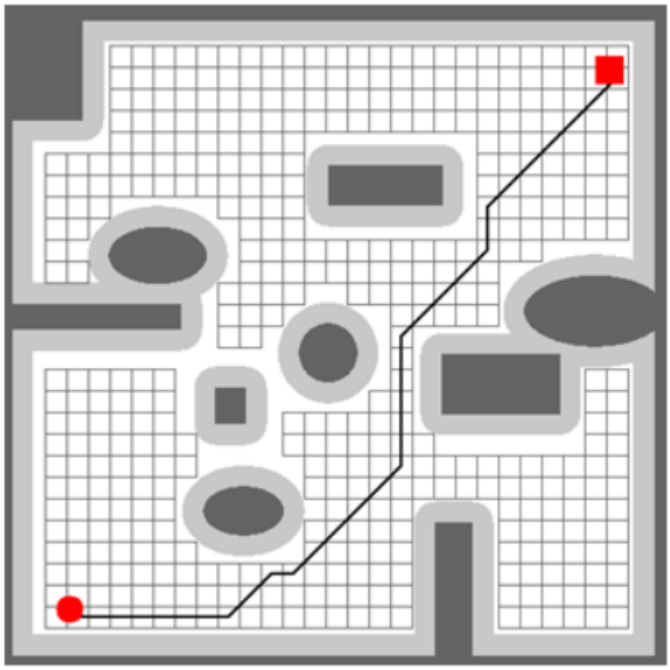


Figure 2.5: Example of Fixed Size Cell Decomposition [4]

With the Fixed Size Cell approach, the cells have always a predefined size and the C_{space} is divided into squares of that size as shown in Figure 2.5. To obtain high precision with this method, especially in areas near the obstacle, the size of the squares should decrease, increasing the number of cells and, consequently, the processing time [4].

In the Approximated Cell Decomposition using Quadtree, firstly, the C_{space} is divided into four equal cells and every time a cell does not belong to C_{free}, the cell is divided in four. This procedure is recursive and only stops when minimal predefined cell size is reached or all space belongs to C_{free}. Thus, the C_{space} around the obstacles is defined with higher precision. An illustrative example is shown in Figure 2.6.

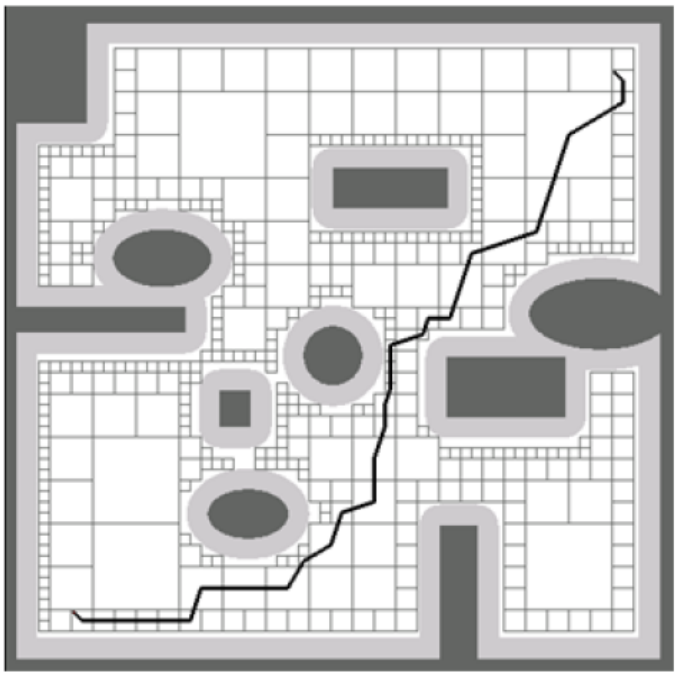


Figure 2.6: Example of Quadtree Decomposition [4]

The limitations of the Quadtree is that the representation requires very high resolution to model the obstacles accurately and it cannot guarantee to find the shortest path (since the position of a path graph node is always the centre of the cell).

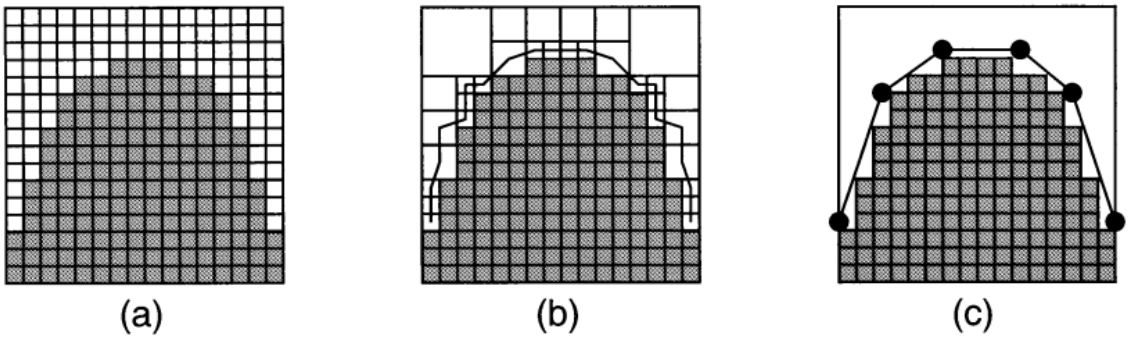


Figure 2.7: Comparison of obstacle boundary modeling using the Quadtree method and the proposed Mesh method. (a) Example obstacle. (b) Quadtree requires 24 vertices to model the obstacle. (c) Mesh requires six vertices to model the obstacle with the nearly equal resolution of (b). [5]

To solve these two limitations of the Quadtree method, [5] proposes a new Compact Mesh (Mesh is a triangular decomposition of a world space) representation for path planning. The vertexes and edges of the compact mesh are determined to optimally represent the curvatures of the obstacles with much fewer number of vertexes and edges than the Quadtree method. A base

mesh is generated by the triangulation and it is simplified to a compact mesh with small error. The simplification algorithm optimises the position of the vertices to better represent the obstacles.

To address the second limitation, the simplification algorithm spreads vertexes in free space in a balanced way (since the density of vertexes is controllable by an edge length threshold). The path graph is extracted from the Compact Mesh and it is possible to verify that that the proposed method generated paths as short as the Quadtree using a much simpler path graph [5]. Through Figure 2.7, it is possible to observe the differences between Quadree and the proposed compact Mesh related to obstacle boundary modelling.

2.2.2 Graph Search Algorithms

In the previous subsection, some methods for path planning were shown. All of the methods mentioned resulted in a graph of the environment space, C_{space}. After obtaining the path graph, to find the best path that connects the initial point to the destination, it is necessary to resort to graph search algorithms. For that matter, the performance of the graph search algorithm determines the efficiency of the system and the cooperation between all robots. These algorithms can be divided into two groups: algorithms without information and algorithms with heuristic.

2.2.2.1 Algorithms Without Information

These algorithms are referred to as "without information" because they do not invoke any other information besides the initial and destination point. They are based on exhaustive search and only when the destination is reached, the algorithm recognises that solution was found. Otherwise, the algorithm does not have the perception if the solution is close or not [4].

In this category, the most known algorithms are of search by depth, search by width, limited depth search and iterative deepening search. What distinguishes them is the search direction, making their efficiency dependant on the graph's format and the localisation of the initial and destination node.

Depth Search Depth-first Search is a depth search algorithm that explores the deepest nodes before retreating. The search is done from bottom to top and left to right. For each node, it only explores the first link, successively, until it finds the solution or reaches a node without any links. If it reaches a node without any links, it retreats until it finds a node with links to explore [4].

Width Search The width search algorithm, Breadth-first search, explores all links from one node before moving to another. This results in a search done by layers [4].

The main differences between the depth search and the width search are shown very clearly in Figure 2.8.

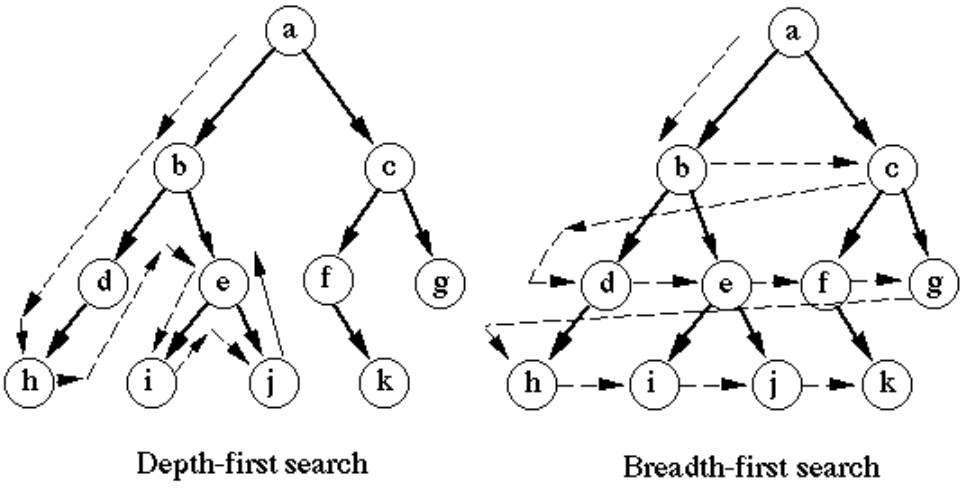


Figure 2.8: Comparison of the Depth-first Search method (depth search) and the Breadth-first Search method (width search) [6]

Limited Depth Search Depth–limited Search algorithm executes the Depth-first Search algorithm until it reaches the limit imposed. Even if there are more nodes with links to explore, the algorithm does not expand and retreats [4].

Iterative Deepening Search Iterative Deepening Depth-first search is an algorithm that applies the Depth-Limited Search algorithm multiple times, increasing each time the depth limit until a solution is found [4].

2.2.2.2 Algorithms With Heuristic

Algorithms with heuristic search attempt to optimise the problem by iteratively improving the solution based on a given heuristic function or a cost measure. A heuristic search method does not always guarantee an optimal or the best solution but instead finds an acceptable solution within a reasonable amount of time and memory space [40].

In a multi-AGV environment, the cost used in a heuristic algorithm could be the time it would take the AGV to cross from one node to another in the physical space. The cost can also represent how difficult it is going from one node to another or even the distance it would take to traverse [9].

The most recognised algorithms with heuristic are the Dijkstra's algorithm, the Greedy algorithm and A* family algorithms.

Dijkstra's Algorithm The Dijkstra's Algorithm is an algorithm conceived by Dijkstra in 1956. To find the shortest path, the algorithm chooses an unvisited vertex with the lowest distance and the distance through it to each unvisited neighbour is calculated. If the resulted distance is smaller, the neighbour's distance is updated. After all nodes are visited, the path with the smallest travelled distance is the one chosen.

The algorithm is complete if the maximum number of links of a node is finite.

Greedy Algorithm The Greedy Algorithm, in each iteration, analyses the closest node to the destination and selects the locally optimal choice. For most problems, the Greedy algorithm does not return an optimal solution. Nonetheless, it provides a locally optimal solution close to a globally optimal solution in a reasonable amount of time [41].

The algorithm is complete if the maximum number of links of a node is finite.

A* Algorithm The A* algorithm uses a heuristic function $f(n)$ to search through a graph for an optimal path from an initial node to a destination node.

The heuristic function $f(n)$, that determines in which order to explore the nodes to find the optimal path in the least amount of time, is calculate through Equation 2.1. The function $h(n)$ represents the current cost from the initial node to the n node. The function $g(n)$ represents the cost from going from the n node to the destination node [4].

$$f(n) = h(n) + g(n) \tag{2.1}$$

The main steps of the A* algorithm consist in [4] [42] :

1. Mark the initial node i as "open" (store the node in the open list, O-list, where all candidate nodes to be explored are stored) and calculate $f(i)$;
2. Select the open node n from O-list whose value of f is the smallest. This node is now considered explored so it is removed from the O-list and moved into the C-list (closed list: the list with all explored nodes);
3. Calculate the cost of all the adjacent nodes of node n, which can be one of the following cases:
   (a) Be inserted on the O-list, if it is neither there nor on the C-list, with the information of the f value and its predecessor, known as parent (node where it came from);
   (b) If it already exists in the O-list, confirm if the function f is smaller than the value previously stored with the node. If so, the parent node is changed and the new value of the f function is stored;
   (c) If it belongs in the C-list, verify if the function f is lower. If so, this node is moved again to the O-list.
4. Go to step 2 until the destination node is reached or the O-list is empty (in this case there is no solution).

The optimal path is obtained by starting on the destination node and following each parent node until arriving at the initial node.

This algorithm possesses several properties including completeness, admissibility, and optimality. The A* algorithm converges to an optimal solution whenever it exists [42]. Nevertheless, its performance can be significantly affected by the computational power, memory limitations, the type of data structure used for the open and closed lists and the heuristics used [4].

TEA* Algorithm Recently, a new graph search algorithm, based on the A* algorithm, was developed by [43]. This new approach aims to fulfil industrial needs by creating routes that minimise the time allocated for each task, avoid collisions between AGV and prevent the occurrence of deadlocks. It is described as a multi-mission algorithm that incrementally builds the path of each vehicle considering the movements of the others, revealing to be very fitted for a multi-AGV environment [43].

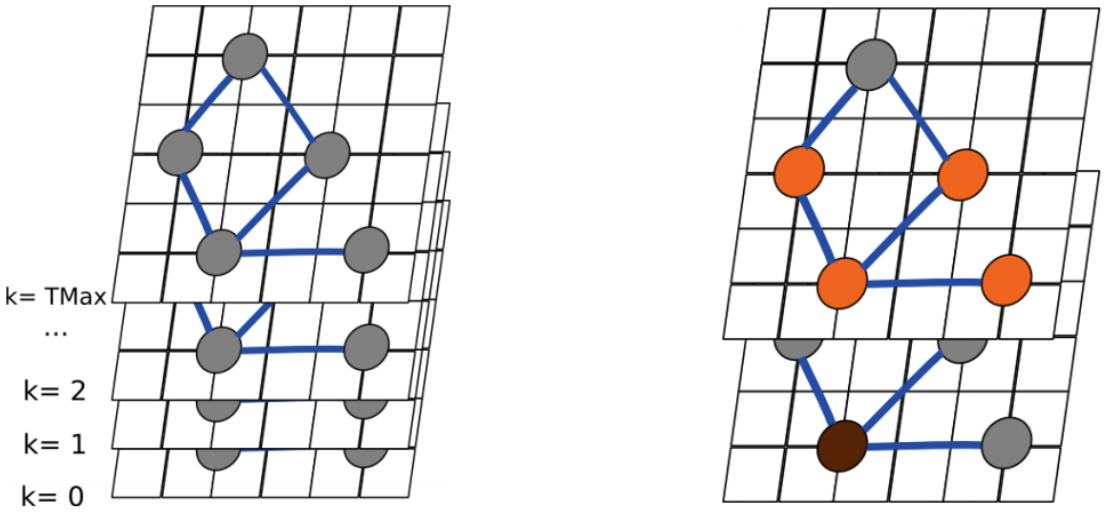


Figure 2.9: TEA* input map and analysed neighbour cells focusing on the cell with the AGV's position [7]

Considering a graph G with a set of vertexes V and edges E (links between the vertexes), with a representation of the time $k = [0; T_{Max}]$, each AGV can only start and stop in vertexes and a vertex can only be occupied by one vehicle at a temporal layer [7]. As it can be seen in Figure 2.9, the analysed neighbour cells belong to the next temporal layer and include the cell containing the AGV's current position.

The number of layers of the map depends on the number of necessary iterations to achieve the final mission and on the map's dimension. The larger the map, the more temporal layers are required. In addition, the more obstacles there are, the longer it will take to find the final point (since the obstacles must be all avoided) [43].

To find the minimal path, the algorithm starts by calculating the position of each robot in each temporal layer. Hence, in the future, possible collisions can be identified and avoided in the beginning ($k = 0$) of the paths' calculation. This contributes to a collisions free initial path's calculation.

Similarly to the A* Algorithm, during the path calculation, the next analysed neighbour cell is dependent on a cost function. In the TEA* approach, the heuristic function used is the euclidean distance [43].

The main differences between TEA* and A* algorithms reside in the features considering the time domain. In TEA*, the analysed neighbour cells belong to the next temporal layer and include the cell containing the AGV's current position, as it is shown in Figure 2.9. This late property allows the AGV to maintain its position between consecutive time instants, if any neighbour cell is free [43].

When applying this algorithm to a multi-AGV environment, the first step is to convert the AGV's current positions in obstacles. As already mentioned, this will allow that a vehicle considers the other's position as occupied cells. To avoid deadlocks these cells are placed as obstacles only in the temporal layer $k = \{0, 1\}$. Thus, each vehicle knows that the corridor is occupied in the first instant but can calculate an alternative collision-free path in the following instants, allowing it to recover from a deadlock situation [43].

The next step of the control loop consists in analysing the list of missions (the tasks assigned to each vehicle) and calculating the path for each AGV. Before moving to the next mission on the list, the calculated path is converted to a moving obstacle for the other robots [43].

Since the coordination between robots is essential to avoid collisions and to guarantee the correct execution of the missions, this approach ensures it since the previously calculated paths become moving obstacles [43].

Dynamic A* Algorithm The Dynamic A* (D*) is an algorithm similar to A* except that the links' cost can change as the algorithm runs [44].

With this algorithm, the robot aims to navigate to the goal coordinates in an unknown environment. Assumptions are made about the unknown part of the environment (for example: that it contains no obstacles) and the shortest path from its current coordinates to the goal coordinates is calculated under these assumptions [44].

When it is observed new map information (such as previously unknown obstacles), it is added to the map and, if necessary, the shortest path from its current coordinates to the given goal is re-planned. The process is repeated until the goal is reached or the goal coordinates cannot be reached [44].

Lifelong Planning A* Algorithm The Lifelong Planning A* (LPA*) algorithm is an incremental version of A*. However, this algorithm can adapt to changes in the path without re-calculating every paths by updating the g-values (cost from going from the n node to the destination node) from the previous search during the current search to correct them, when necessary [45].

When the cost of a node's predecessor or the edge linking it to a predecessor changes, the node becomes locally inconsistent and it is placed in a priority queue for re-evaluation. The nodes are expanded and analysed in the priority queue [45].

D* Lite Algorithm D* Lite is a fast re-planning method for robot navigation in an unknown environment, built on Lifelong Planning A* (LPA*) algorithm (an incremental heuristic search algorithm based on A*). It also implements the same navigation strategies as D* Algorithm [46].

Both algorithms, D* Lite and D*, search from the goal vertex towards the current vertex of the robot, use heuristics to focus the search, and use similar ways to minimize having to re-order the priority queue. However, since D* Lite does not expand any vertices whose g-values (cost from going from the n node to the destination node) were already equal to their respective goal distances, D* Lite is algorithmically different from D* [46].

The D* Lite Algorithm is described as easy to understand and extend while being as efficient as D* Algorithm [46].

D*LR Algorithm When it comes to the D* Lite algorithm, if the robot encounters an environment change when the old path is almost completed and the new path is much longer than the old one, the D* Lite will waste a lot of computing resources to correct the inconsistent nodes (nodes become inconsistent because the cost of its predecessor or the edge linking it to a predecessor has changed). If the total number of these nodes is greater than the number of already-consistent nodes, it is more profitable to reset the D* Lite and recompute the old data instead of reusing it while correcting the false one [47].

Therefore, D* Lite with Reset (D*LR) is a modified version of the D* lite that aimed to correct the mentioned problem. This new algorithm conducts a test whenever the environment changes. If the test is passed (or rejected), it will reset the D* Lite, discarding old data and starting a whole new search [47].

2.2.3 Deadlocks, Livelocks and Semaphores

Two of the main situations that we look forward to avoiding are deadlocks and livelocks. These situations are a common problem concerning path planning and can destabilise the system, decrease the efficiency of the paths planned and avoid substantial production interruptions. For better utilisation of the workspace, flexible deadlock and congestion avoidance mechanisms need to be implemented [48].

A deadlock is described as a situation when a resource is assigned to various vehicles compromising the flow of the system. This problem is a logical one that can emerge in different contexts. However, when deadlock situations arise the following conditions are always observed [49]:

- Tasks claim exclusive control of the resources they require ("mutual exclusion" condition);
- Tasks hold resources already allocated to them while waiting for additional resources ("wait for" condition);
- Resources cannot be forcibly removed from the tasks holding them until the resources are used to completion ("no preemption" condition);

- A circular chain of tasks exists, such that each task holds one or more resources that are being requested by the next task in the chain ("circular wait" condition).

To prevent deadlock situations, at least one of these conditions needs to be avoided.

On the other hand, systems can also be confronted with the livelock situation. This situation appears when a vehicle is trying to access a resource that is continuously occupied by others.

In multi AGV environments, the resources shared are usually physical spaces that the vehicles must access to fulfil their tasks. To manage these resources, some algorithms use the priorities established to each robot to decide which robot must access the resource. However, that does not always guarantee the prevention of deadlocks and livelocks [9].

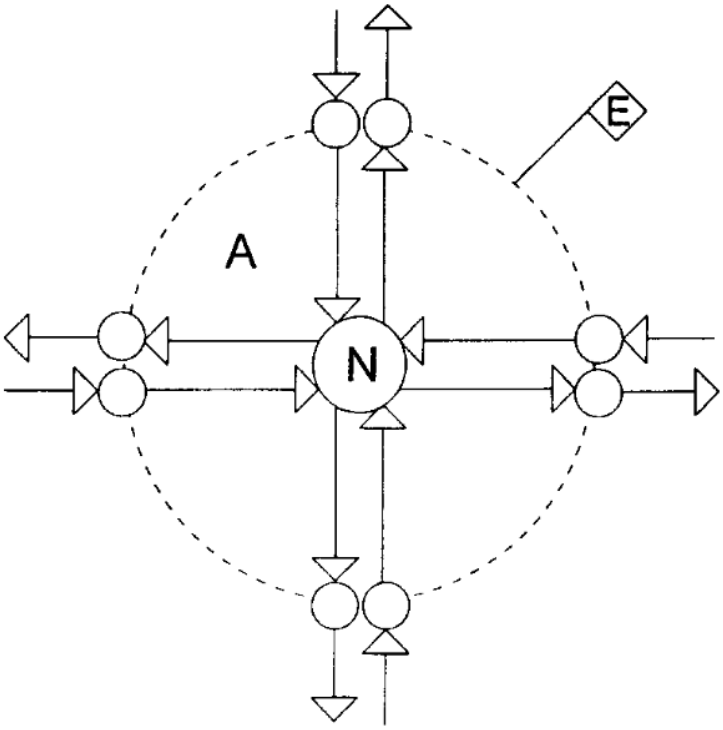


Figure 2.10: Area A composed of an elementary node N (shared resource) controlled by semaphore E [8]

Some synchronization methods can be used to help to control the vehicles' traffic. In the example developed by [8] (Figure 2.10), the AGV's traffic was controlled through the use of binary semaphores (a variable whose value varies between 0 and 1). Whenever a vehicle approached the resource N, the vehicle stopped: if the semaphore E=0 then N was occupied and the vehicle had to wait, else if N was free, the vehicle could proceed and the semaphore switch from 1 to 0 (N was now occupied). When the vehicle left, E switched values to 1 to allow another vehicle to access the resource N. When two or more vehicles wanted to access N, one of them had to be selected, so it was necessary to introduce an access protocol on E. Various rules could decide which one to access. The theory of production control suggests the following [8] :

- Random;
- First-In-First-Out;
- Priority to a vehicle in the same direction as the predecessor;
- Priority to the vehicle with the earliest due time;
- Priority to vehicles with a direction with the smallest queue on the following semaphore;
- A combination of the rules mentioned above.

The semaphores can also be counters and be applied to resources that allow multiple instances at the same time.

Dealing with deadlock avoidance for relatively large AGV systems (a large number of AGV and/or zones) is quite challenging [48]. Wherefore, a supervisory system that can identify these situations and prevent them would have a significant impact on the system's performance.

2.3 Communication Faults: System Supervisor

The work developed by [9] contemplates some solutions for communication failures in a multi-AGV environment. The methods implemented can be divided into three:

1. Detection and tolerance of communication faults inside the robot control cycle;
2. Adjustment of path planning taking into consideration the nodes where faults were detected (modified TEA*);
3. Detection of communication faults and association with the graph's nodes.

When it comes to the TEA* algorithm's adaptations, [9] modified the TEA* library to indicate the nodes associated with active faults as immovable obstacles until the affected robot exited the communication fault area and reestablished the communication with the central module. This alteration prevented any other robot from entering the communication fault zone, regulating the traffic. In the example shown in Figure 2.11, both blue and orange robots must enter a critical zone (the link delimited by the two red nodes) where no communication can be established. When the blue robot reaches the communication fault zone, the orange robot is forced to wait until communication is reestablished with the blue robot (since it is the only possible path for the orange robot, otherwise the path would be re-planned) [9].

To detect communication faults, a specific module was created to control when the robots' paths needed to be re-planned and to detect, measure and handle communication faults. This module is composed of two sub-modules: one responsible for controlling when the robots' path need to be re-planned in situations where no communication faults are detected (Planning Supervision Sub-Module) and another sub-module to intervene whenever the communication with a robot is lost (Communication Supervision Sub-Module).

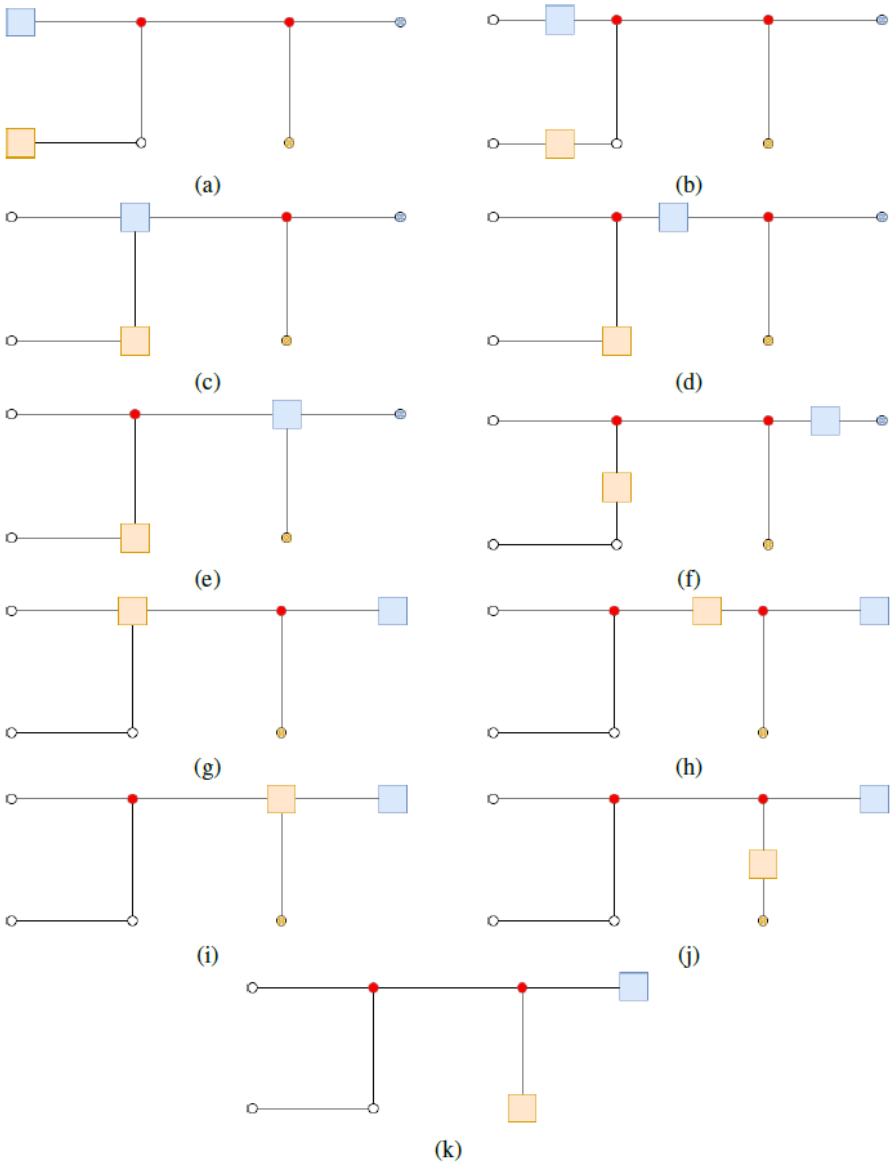


Figure 2.11: Example of a Communication Fault Zone [9]

The Planning Supervision Sub-Module detects when one of the robots is delayed or ahead of its time and if a robot already completed the current step of its path in order to increment it safely. In the TEA* algorithm, when all the robots are synchronised, it means that all robots are in the step assigned to the current temporal layer. If a robot is delayed or ahead of time, it means that it is not on the same temporal layer as the rest of the fleet, possibly leading to collisions and deadlocks.

The other sub-module, the Communication Supervision Sub-Module, is responsible for detecting communication faults, calculating their size and forcing path re-planning considering the detected fault [9]. There are two types of faults: areas in the factory floor map that consistently have no communication with the Central Control unit and areas that face a temporary loss of communication.

Since the system has no prior knowledge of the existence and localisation of these areas, it is necessary to map the factory floor at the same time the robots execute their tasks. When a new robot enters a fault zone, the nodes where it was still possible to communicate with the robot are marked as unflawed and are used to delimit the critical area. Therefore, when a robot enters a zone with no communication, it is possible to estimate when it will exit the zone and re-establish communication with the central unit.

When dealing with sporadic faults, if a zone was already mapped as critical (with communication faults) and a robot is capable of transverse it without experience any loss in communication, the whole fault is discarded, leading to a remapping of the zone.

Chapter 3

Implemented System

Following the work developed by [9], it was intended to coordinate a fleet of three small scale AGV in a real environment using a global path planning with the TEA* graph search algorithm combined with a supervisory system.

Executing the TEA* algorithm with a supervisory system revealed to be a more efficient approach to avoid collisions and deadlocks. Instead of keeping the TEA* methodology an online method, where the paths are re-calculated every cycle (a procedure that is computationally heavy), the supervisory system was responsible for detecting critical situations.

The supervisor detected when delays in the communication, deviations in the routes of the robots and communication faults happened and triggered the re-calculation of the paths when needed. Overall, the supervisor was responsible to keep the robots synchronised because, if the robots are not synchronised, it means that each one is at a different step of the planned path, which can lead to collisions and deadlocks.

Therefore, [9] proposed a supervisory system consisting of two sub-modules, Planning Supervision Sub-Module and Communication Supervision Sub-Module, that were tested in a real implementation and further developed to fit a real environment's dynamic.

3.1 System Architecture

To implement and test the work developed by [9] and further modifications, it was set a shop floor map and developed a fleet of three robots and their control module, a localisation system based on the pose estimation of fiducial markers and a central control module.

This model does not represent a real life situation (since the localization of the robots is given by the Robot Localisation Module and not by the robots themselves) but it is suitable to test and develop the supervisory system.

The architecture of the system developed is represented in Figure 3.1. The communication between the different modules is established via Wi-Fi.

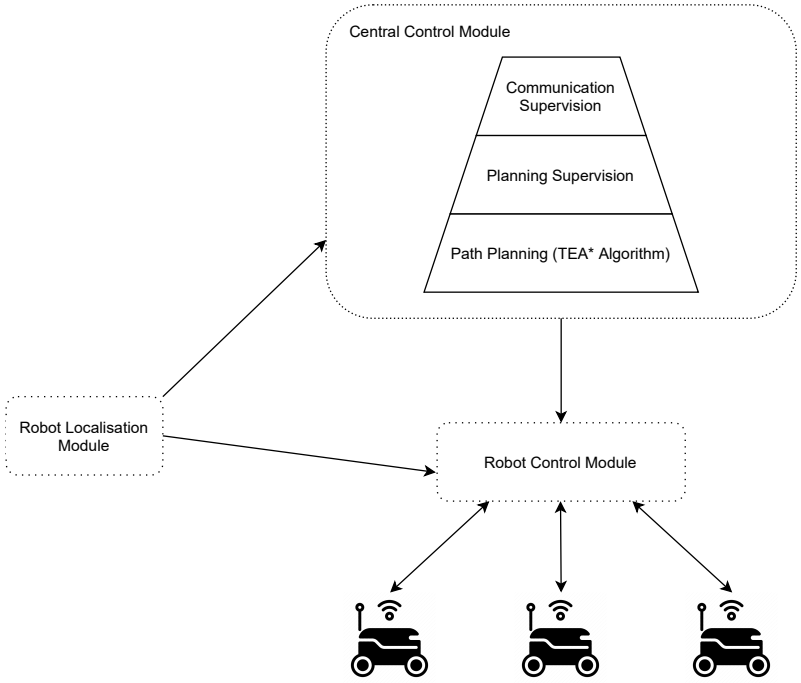


Figure 3.1: System architecture and relations between the different modules

The Central Control Module of the system maintains the structure initially developed by [9]. Therefore, it is composed of three hierarchical units: the Path Planning Sub-Module (TEA* Algorithm) and the Supervisor: Planning Supervision Sub-Module and Communication Supervision Sub-Module.

The Robot Localisation Module locates and estimates the robots' coordinates. It then communicates, through User Datagram Protocol (UDP) messages, simultaneously, with the Central Control Module and the Robot Control Module.

Lastly, the Robot Control Module is responsible for calculating and delivering through UDP packets the most suitable velocities for each robots' wheel, depending on the destination point/node indicated by the Central Control Module.

The system was configured with a centralised control architecture since the graph search algorithm used, the TEA* algorithm, selects the optimal paths from a graph of the global environment space, C_{space}, and communication between robots is non-existent. However, instead of using the TEA* algorithm as an online method, the Supervisory Module is responsible for detecting the environment changes and triggering the re-calculation of the paths, providing good adaptability to the overall system.

3.2 Implementation

The main elements, that were necessary to introduce to the work already developed, were a shop floor map and a fleet of three robots.

The robots developed represented a small scale AGV and the map designed had the dimension of 173 cm per 123 cm.

3.2.1 Robot Architecture

To validate the implemented system, three differential mobile robots were developed, as a small scale of an industrial Automated Guided Vehicle, AGV. Each one measures 11 cm length by 9 cm width, as presented in Figure 3.2. Each robot was produced through additive manufacturing using a 3D printer.

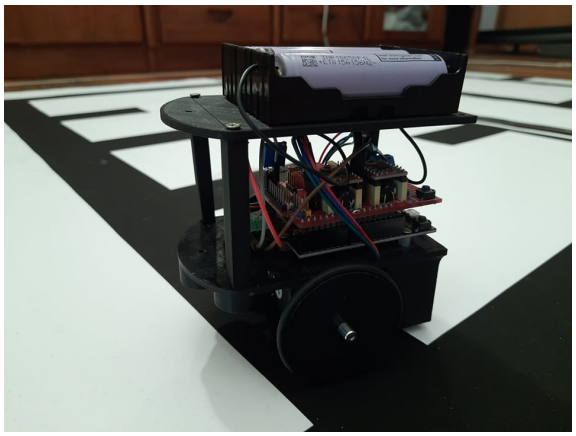


Figure 3.2: Example of the robot developed to perform tests

Each robot is powered by three Li-Ion MR18650 batteries, placed on the top of the robot, that supply the main controller board based on an ESP32 microcontroller and the stepper motor drivers DRV8825.

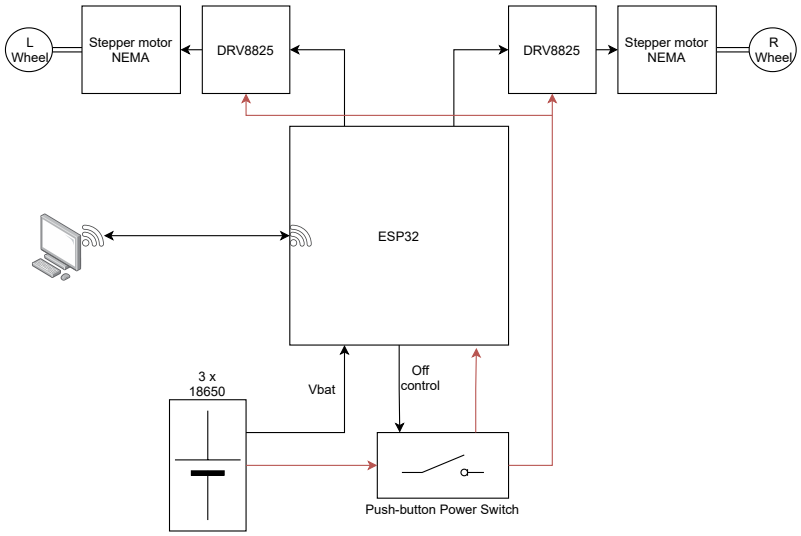


Figure 3.3: Robot's diagram: Black lines represent control signals and red ones represent power

The ESP32 owns Wi-Fi connection that allows it to communicate with the Robot Control Module. The main architecture of the small scale AGV is presented in Figure 3.3. A push-button Power switch controller is used to turn the system off when batteries are discharged, avoiding damaging them. A voltage divider was applied so that the microcontroller is able to measure the battery voltage.

The communication between the Robot Control Module and each robot is also done using UDP packets. Commands are sent to the motors while the Robot Control Module receives information from the robots about the odometry (steps).

To avoid slippage, a circular rubber was placed on each wheel. A free contact support of Teflon with low friction was also used to support the robot.

3.2.2 Shop Floor Map

The shop floor map designed for testing is presented in Figure 3.4.

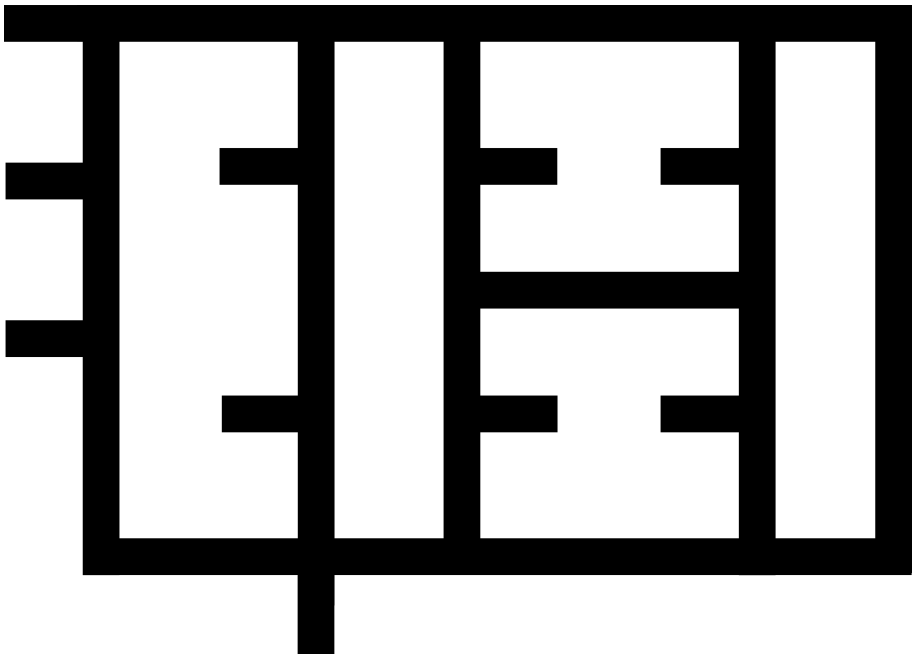


Figure 3.4: Shop floor map

The map in Figure 3.4 was turned into a graph with links around the size of the robot plus a safety margin, in this case, it was intended for the links to measure, approximately, 15 cm.

The decomposition of the map was done using the Map Decomposition Module, developed by [9], and incorporated in the Central Control Module through an XML file. Through Figure 3.5, it is possible to observe how to map was decomposed in graph nodes. It is also possible to observe that the distance between two nodes is not always the same.

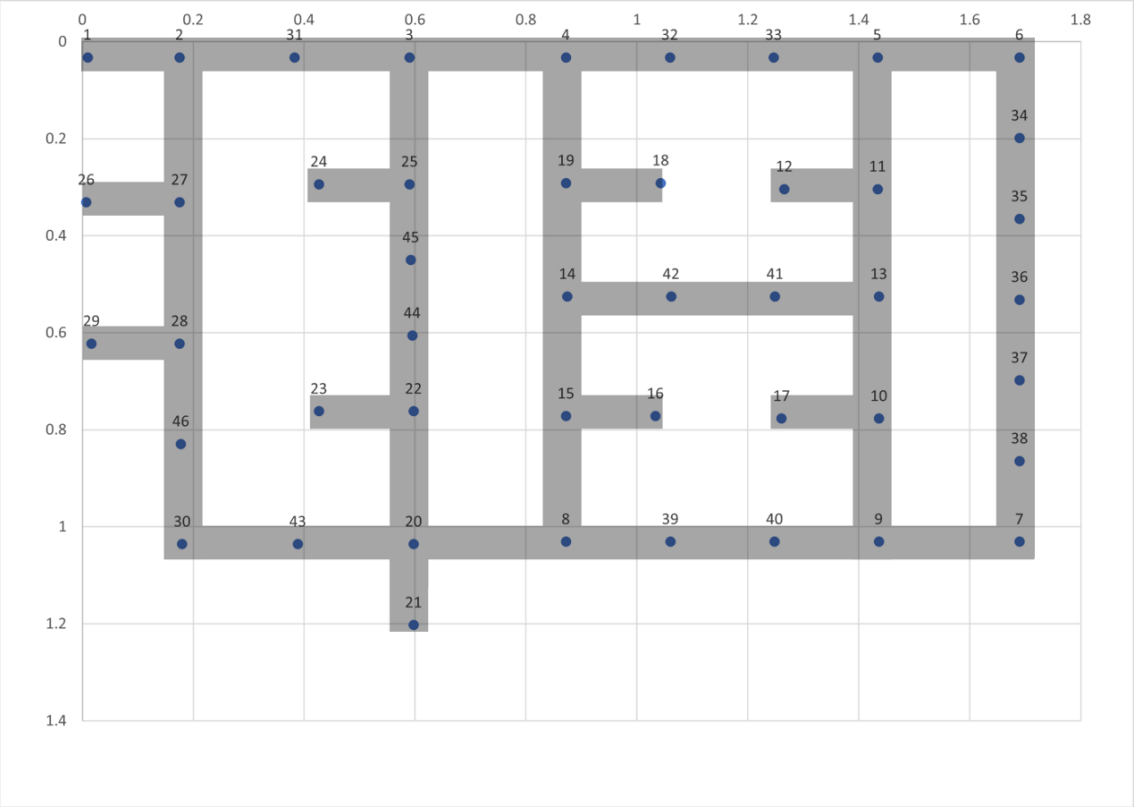


Figure 3.5: Nodes from the graph resulted from the decomposition of the map and their location (meters)

The Map Decomposition Module aimed to divide the links between the main nodes (the nodes on crossroads, path terminations and path direction changes) into smaller links of 15 cm by adding new nodes.

However, this was only possible if the link was superior to 30 cm, resulting in two links of 15 cm and a new node. Otherwise, the link would not be divided, resulting in links of different dimensions.

The smallest link measures, approximately, 15.6 cm and the longest, approximately, 25.15 cm. This difference affects the coordination of the robots, as will be further explained.

3.3 Robot Localisation Module

To control the robots and run the system, the current coordinates, *X* and *Y*, and orientation, *Theta*, of the robots were crucial. To obtain these values, approaches such as odometry or even an Extended Kalman Filter localisation using beacons [50] would be suitable. However, for pose estimation, the Robot Localisation Module applies a computer vision algorithm.

For this purpose, a camera was set above the centre of the shop floor map and the robots were identified with ArUco markers [51] [52], as represented in Figure 3.6.

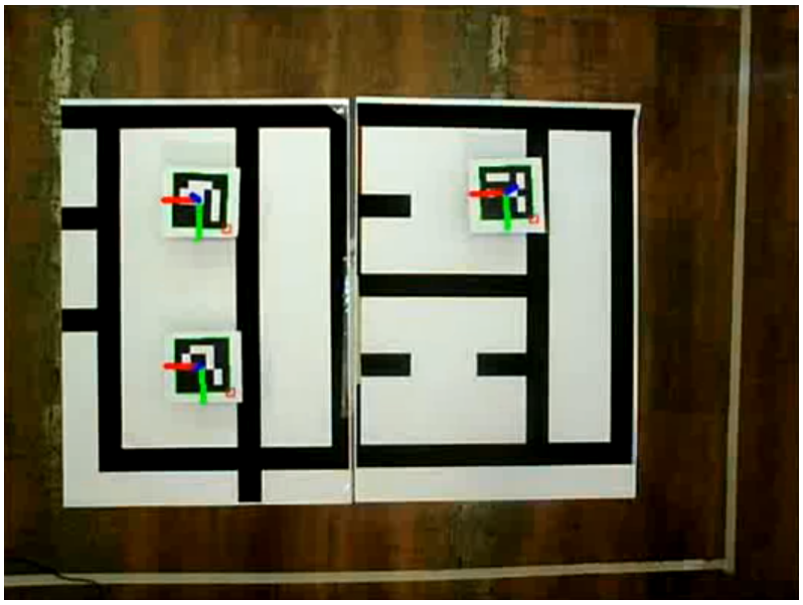


Figure 3.6: Pose estimation through the detection and identification of ArUco markers (Robot Localisation Module)

The camera distortion was corrected through its calibration using the OpenCV library [53]. Through this procedure, it was possible to obtain the camera matrix, the distortion coefficients and the rotation and translation vectors. These variables were incorporated in the ArUco functions to obtain the coordinates X , Y and *Theta* in the real world frame.

(a) First marker (ID: 23) (b) Second marker (ID: 33) (c) Third marker (ID: 43)

Figure 3.7: ArUco markers used to identify the robots

The ArUco markers used are square markers characterised by a wide black border and an inner binary matrix that determines their identifier. Using the ArUco library developed by [51] and [52] and associating an ArUco tag id to each robot (Figure 3.7), it was possible to obtain the coordinates of the robots' positions.

However, the vision algorithm revealed to be very sensitive to poor illumination conditions and unstable in regard to non-homogeneous lighting. The size of the tag used was 10 per 10 cm with a white border of 1.6 cm.

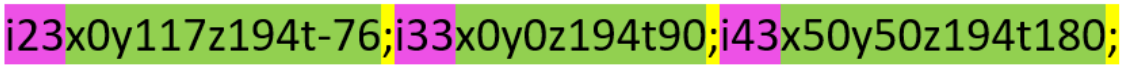


Figure 3.8: UDP packet sent from Robot Localisation Module to both Central Control Module and Robot Control Module

The Robot Localisation Module is able to communicate with the Central Control Module and the Robot Control Module through UDP messages. Each UDP message carries the positions of all robots detected and one message is sent every 170 ms (camera frame rate plus vision algorithm processing time).

In Figure 3.8 is represented a UDP message sent from the Robot Localisation Module to both Central Control Module and Robot Control Module. Highlighted in pink is the ArUco tag identified by the Localisation Module. The match between the ArUco marker and the robot's ID and IP address is done only in the Central Control Module and Robot Control Module. Highlighted in green is the information concerning the robot previously identified. The computer vision algorithm identified the coordinates X , Y and Z and the orientation *Theta* of the robot. However, since the Z coordinate is not needed, both Control Modules discarded it. To separate the robots' information, it is used a termination character, highlighted in yellow. The coordinates are sent in centimetres and the orientation is sent in degrees.

3.4 Robot Control Module

The Robot Control Module is in charge of leading each robot towards the desire nodes indicated by the Central Control Module.

After the TEA* algorithm calculates each robot's path, a command list stores the nodes that the robot has to cross to complete its mission.

Every 80 ms, the Central Control Module sends, to each robot, their updated command list and the Robot Control Module assures the execution of it.

3.4.1 Velocity Control

The velocity of the robots is controlled through the calculation of the most suitable velocity for each wheel for each step's X and Y coordinates and direction/*Theta* angle. That operation is done through the function represented by the diagram in Figure 3.9.

In each stage of the diagram, the linear and angular velocities are calculated accordingly. The velocity for each wheel/motor (*M0* represents the motor of the left and *M1* the right motor) is calculated through Equation 3.1 and 3.2, where b represents the distance between both wheels.

$$M0Speed = linearvelocity + \frac{angularvelocity \cdot b}{2} \tag{3.1}$$

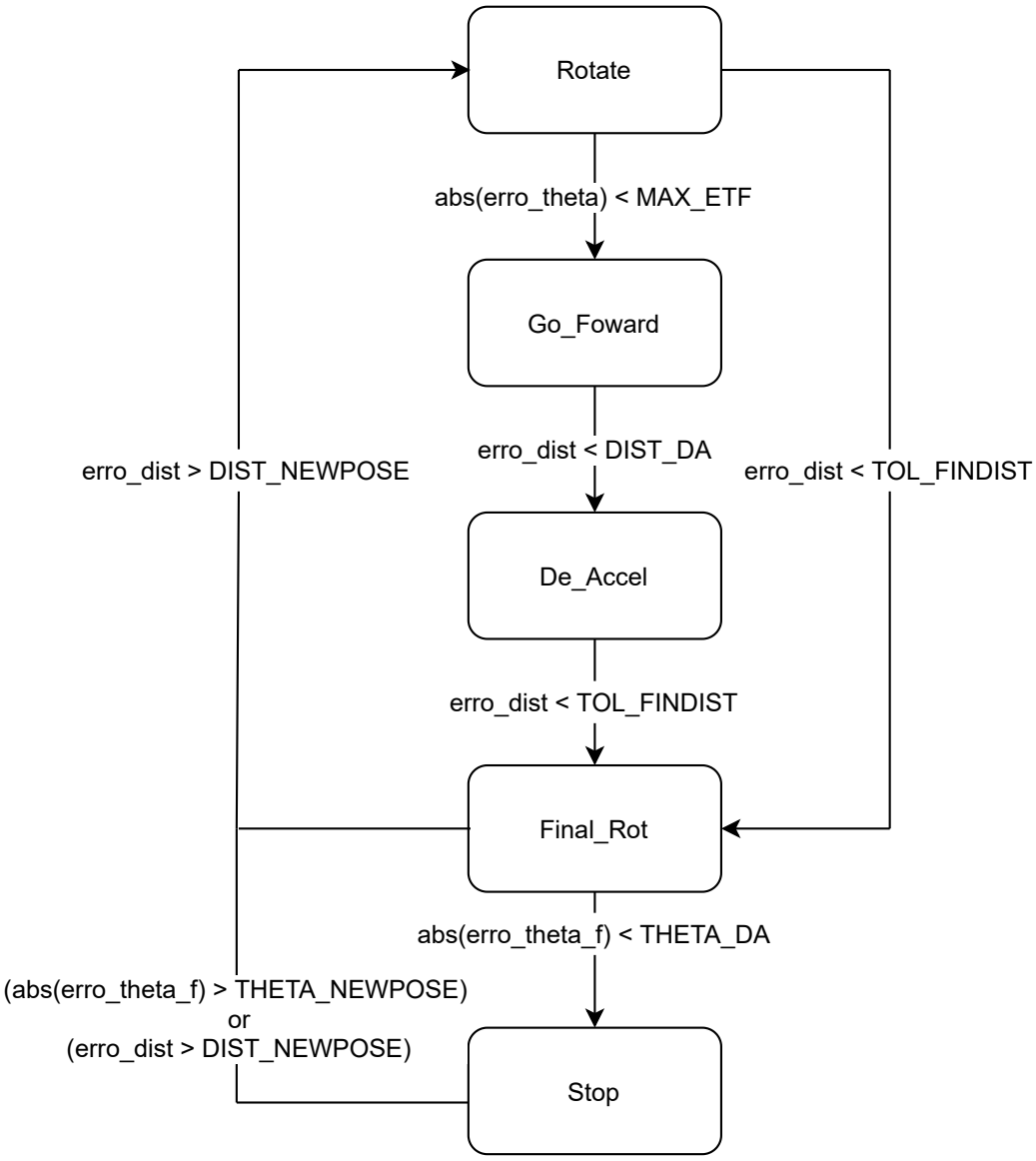


Figure 3.9: Velocity Control Function

$$M1Speed = linearvelocity - \frac{angularvelocity \cdot b}{2} \tag{3.2}$$

To understand which movement the robot needs to complete, in order to move towards the desired point, three types of error are calculated. These errors are the *erro_theta*, Equation 3.3, which represents the error between the current robot's orientation (Theta) and the orientation that the robot needs to have to travel towards the destination; the *erro_dist*, Equation 3.4, that represents the distance between the current robot's position and the target position; the *erro_theta_f*, Equation 3.5, which represents the error between the robot's current orientation and the target/final orientation given by the Central Control Module.

$$erro\_theta = thetarobot - Atan2(ytarget - yrobot, xtarget - xrobot) \tag{3.3}$$

$$erro\_dist = \sqrt{(xrobot - xtarget)^2 + (yrobot - ytarget)^2} \tag{3.4}$$

$$erro\_theta\_f = thetatarget - thetarobot \tag{3.5}$$

By comparing these errors with the threshold values, the linear and angular velocities are calculated to fit the type of movement needed.

When a new target position is indicated, to rotate the robot towards it, the Rotate step is selected. The linear and angular velocities are calculated accordingly to Equation 3.6 and 3.7, where *sign* determines the fastest rotation (positive or negative rotation) and *WNOM* represents the nominal angular velocity.

$$linear\_vel = 0 \tag{3.6}$$

$$angular\_vel = sign \cdot WNOM \tag{3.7}$$

After the robot acquires the correct orientation (the module of the *erro_theta* is smaller than a certain *MAX_ETF* threshold), the robot begins its movement forward, state Go_Forward. The linear velocity maintains its nominal value and the angular velocity is calculated through a proportional controller that tries to correct any angle deviation *erro_theta*, Equations 3.8 and 3.9.

$$linear\_vel = LINVELNOM \tag{3.8}$$

$$angular\_vel = -GAIN\_FWD \cdot erro\_theta \tag{3.9}$$

When the robot is within a certain distance, *DIST_DA*, the function enters a deceleration step in order to stop the robot with the smallest error. To reduce its velocity, the linear velocity takes into account the distance error, *erro_dist*, normalised. Reaching a distance error smaller than the

threshold *TOL_FINDIST*, the function looks forward to correct the robot's orientation to the target orientation.

$$linear\_vel = LINVELNOM \cdot \frac{erro\_dist}{DIST\_DA} \tag{3.10}$$

$$angular\_vel = -GAIN\_DA \cdot erro\_theta \tag{3.11}$$

To correct the robot's orientation, the function enters a Final_Rot state that calculates the angular velocity by relying on the final angle error, Equations 3.12 and 3.13.

$$linear\_vel = 0 \tag{3.12}$$

$$angular\_vel = GAIN\_DA \cdot erro\_theta\_f \tag{3.13}$$

After completing the final rotation, the robot enters a rest state, Stop step. This state is only changed if a new target is received by the velocity control function.

3.4.2 Online Mode

The communication between the Central Control Module, constituted by the Path Planning Module (TEA* Algorithm) and the Supervisory Module, and the Robot Control Module is established through UDP messages. After the paths are planned, the information is sent to the Robot Control Module through UDP packets. Each packet has the structure represented in Figure 3.10.

Figure 3.10: UDP packet sent from Central Control Module to the Robot Control Module

The heading of the packet, highlighted in yellow, carries the robot id (*N*), the priority of the robot (*P*) and the number of steps that the packet contains (*S*). Consequently, the body of the packet, highlighted in green, contains the information of the *S* steps. This information is organised by the number of the step (*I*), the coordinates *X* and *Y* and the direction *D* (that is translated into an angle in radians inside the trajectory control function). The ending of the packet, highlighted in blue, indicates if the robot has reached the destination and carries a termination character. The indicator *T0* or *T1* informs if the robot's current position is the final destination (*T1* if yes, *T0* if not) as a form of reassurance. When a robot finishes its mission, the UDP packet sent would not contain any steps (*S0*), therefore it does not contain any information on its body. The character *F* was the termination character chosen to indicate the end of the packet.

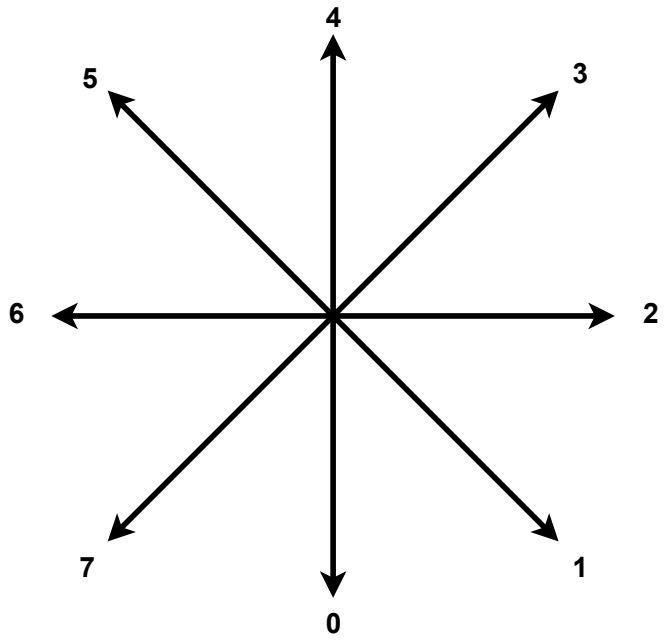


Figure 3.11: Orientation Compass Card

To include the orientation of the robot in the TEA* algorithm, it was needed to be represented by a discrete value. Hence, the robot's direction was translated to integers from 0 to 7, where each integer represented a direction of the compass card, Figure 3.11.

Due to the map's frame having its origin in the superior left corner, the X axis pointing to the right and the Y axis pointing down (Figure 3.5), the orientation of the robots is as described in Figure 3.11. The -90 degrees are represented by 0 since it is the direction for the robot to move in the positive Y axis. The integers are translated into radians inside the Velocity Control Function.

After decoding the packet received from the Robot Localisation Module, the suitable velocity for each wheel is calculated through the Velocity Control Function.

3.4.3 Offline Mode

The Robot Control Module needed to be able to detect when the Central Control Module could not establish communication with a robot. Hence, message control was needed. After four cycles of not receiving new UDP messages from the Central Control Modules, concerning a certain robot, an offline mode takes over.

When it comes to the Central Control Module, it is considered that after 3 cycles without any message concerning the robot's position, the robot is inside a communication fault area. Since it is received a UDP message from the Robot Localisation Module every 170 ms, the Central Control Module only associates the robot as in a communication fault after 3×170 ms = 510 ms. Each cycle in the Robot Control Module takes 80 ms and 4 cycles correspond to 4×80 ms = 320 ms. Being the worst case the robot moving forward at the nominal linear velocity, and since the linear

velocity was 800 steps/s (4.09 cm/s), in the worst-case scenario, the robot would move 3.39 cm without any control.

Reaching the fourth cycle with no messages, an offline mode is activated. This mode guarantees the accomplishment of the last command list received while trying to avoid any delays, collisions and deadlocks.

While the robot is not able to establish communication with the Central Control Module, a timer is set to force the lost robot to respect its steps within their time. This operation revealed to be very important, especially for the waiting steps (steps where the robot awaits for its turn to move because its path is temporary blocked, usually by other robot). It also allowed for more than one robot to cross the faulty area, simultaneously, with more assurance of safety.

For the offline mode's timer, an estimation of each step's time was made. For this estimation, some tests were performed and it was analysed if the different steps (movement and rotation) were taking, approximately, the same execution time.

3.4.4 Velocity Control Tests

To validate the Velocity Control Function and analyse one step's execution time, tests were performed.

The tests consisted of commanding a robot to go from an initial position and orientation to a target X , Y and *Theta* values. For each test, it was aimed to analyse the following:

- The error between the robot's final X coordinate and the target X coordinate;
- The error between the robot's final Y coordinate and the target Y coordinate;
- The error between the robot's final orientation and the target orientation;
- The time taken by the robot to perform the trajectory.

To perform the following tests, the distance threshold *TOL_FINDIST* was set to 3 cm and the orientation threshold *THETA_DA* was set to 0.087 radians (5 degrees). The nominal linear velocity was set at 800 steps/sec (4.09 cm/s) and 125 rad/sec.

As mentioned before, in Section 3.3 and observed in Figure 3.5, the links between nodes do not have all the same length. However, in the TEA* algorithm, one step is equivalent to crossing one link or to a rotation of 90 degrees. So, the time taken by a robot to perform the rotation or to cross one link needs to be, approximately, the same. These tests also aimed to verify this situation.

3.4.4.1 Test I: Movement forward with no rotation

The first situation that was evaluated was the movement of the robot that was equivalent to crossing one link. Since the links do not have the same length, for this test, an average distance of 0.2 meters was used. Therefore, it was expected a straight trajectory of 0.2 meters in the X axis (to avoid any initial rotation) and a final orientation equal to the initial one. This test was performed 10 times and the results are described in Table 3.1.

- Average *X* error: 0.001 m;
- Average *Y* error: -0.003 m;
- Average *Theta* error: 0.8 degrees;
- Average execution time: 4967 ms.

Table 3.1: Results from Test I (coordinates in meters and orientation in degrees)

#	X_{ini}	Y_{ini}	$Theta_{ini}$	X_{final}	Y_{final}	$Theta_{final}$	*X* error	*Y* error	*Theta* error	Time (ms)
1	0.40	0.28	0	0.58	0.29	3	-0.02	0.01	3	5625
2	0.58	0.29	3	0.76	0.30	4	-0.02	0.01	1	4391
3	0.76	0.30	2	0.94	0.31	3	-0.02	0.01	1	4578
4	0.94	0.31	3	1.13	0.31	6	-0.01	0	3	4531
5	1.50	0.34	180	1.31	0.33	-176	0.01	-0.01	4	5609
6	1.31	0.33	-176	1.13	0.32	-176	0.02	-0.01	0	4531
7	1.13	0.32	-176	0.95	0.32	-178	0.02	0	-2	4625
8	0.95	0.32	-178	0.76	0.31	-174	0.01	-0.01	4	4938
9	0.76	0.31	-174	0.58	0.29	-177	0.02	-0.02	-3	6094
10	0.58	0.29	-177	0.38	0.28	180	0	-0.01	-3	4750

3.4.4.2 Test II: Rotation of 90 degrees

The TEA* algorithm also assigns a step to a rotation of, at least, 90 degrees. To avoid delays between robots, it is important that the average execution time of this test is similar to the previous one.

In this test, the robot kept the initial *X* and *Y* coordinates and was expected to rotate 90 degrees. This test was performed 10 times and the results are described in Table 3.2.

- Average *X* error: -0.003 m;
- Average *Y* error: -0.001 m;
- Average *Theta* error: -1.8 degrees;
- Average execution time: 4286.1 ms.

Table 3.2: Results from Test II (coordinates in meters and orientation in degrees)

#	X_{ini}	Y_{ini}	$Theta_{ini}$	X_{final}	Y_{final}	$Theta_{final}$	*X* error	*Y* error	*Theta* error	Time (ms)
1	0.38	0.28	-180	0.37	0.28	-91	-0.01	0	-1	5016
2	0.37	0.28	-91	0.38	0.29	-3	0.01	0.01	-2	4468
3	0.38	0.29	89	0.38	0.28	177	0	-0.01	-2	4391
4	0.38	0.28	177	0.39	0.29	91	0.01	0.01	4	4203
5	0.39	0.29	91	0.38	0.29	3	-0.01	0	2	3860
6	0.39	0.30	8	0.38	0.29	94	-0.01	-0.01	-4	4328
7	0.38	0.29	95	0.39	0.29	-178	0.01	0	-3	4094
8	0.39	0.29	-178	0.37	0.28	-92	-0.02	-0.01	-4	4610
9	0.39	0.29	-91	0.37	0.28	-5	-0.02	-0.01	-4	3703
10	0.37	0.28	-6	0.38	0.29	80	0.01	0.01	-4	4188

3.4.4.3 Results' Discussion

Analysing the results from both Table 3.1 and 3.2, it is possible to confirm that the threshold values were respected. However, the orientation presented a bigger error than the coordinates X and Y. The X and Y errors revealed to be very small and insignificant.

Analysing the execution time, the average values of both tests were similar. This means that, since one step corresponded to both crossing one link and rotating, at least, 90 degrees, delays caused by the execution of these commands should not be common. However, since the links do not all measure the same, some delays are still expected.

To minimise delays on the steps during communication faults, the timer used, in the offline mode of the Robot Control Module, was of 5000 ms. The value took into consideration longer links and situations where a rotation smaller than 90 degrees can be needed before moving forward.

3.5 Conclusion

During this chapter, it was presented a summary of how the system was implemented and some context for the following chapters. The chosen system's architecture was explained, as well as how the modules are connected and communicated between them.

The main adjustments that were applied to the previous work were also discussed. These modifications aimed to replace the simulation platform with a real implementation. Therefore, the shop floor map and the robots used were presented and described.

Another important feature added was a module responsible for locating and estimating the robots' position through computer vision. The technology used was the ArUco markers and library.

It was also detailed how the Robot Control Module is composed and how it runs. It was described how the velocity control of the robots is executed and how the communication faults, between the robots and the Central Control Module, affect the robot's control.

The velocity control is done through a specific Velocity Control Function. This function calculates the velocity of each wheel taking into account the necessary movement for the robot to reach the desired position. The function's behaviour was explained in deep detail during this chapter.

Concerning communication faults, it was explained how the Robot Control Module is divided in two modes. When the Central Control Module can locate the robots, messages are sent from the central module to the Robot Control Module. Otherwise, since the robots can not be located, no messages are sent to the Robot Control Module.

When the Robot Control Module detects that packets from the Central Control Module are not arriving (4 execution cycles), an offline mode is activated. It was described how this offline mode helps to avoid deadlocks and collisions.

The Robot Control Module was also tested to validate the Velocity Control Function. Analysing the execution time of both tests, it was also possible to estimate the time taken by the robots to execute one step of their paths. This time was incorporated in the offline mode of the Robot Control Module.

Chapter 4

Supervisory System

The supervisory system proposed, based on the work developed by [9], is responsible for deciding when to re-plan the robots' paths and for detecting and handling communications faults.

As mentioned before, the supervisory system implemented consists of two sub-modules hierarchically related with each other: Planning Supervision Sub-Module and Communication Supervision Sub-Module.

In environments where no communication faults are present, the Planning Supervision Sub-Module is responsible for determining when one of the robots is delayed or ahead of time (according to the path planned by the TEA* algorithm) and also detect when a robot completes the current step of its path.

On the other hand, the Communication Supervision Sub-Module is only responsible for detecting communication faults. When a communication fault is detected, the Planning Supervision Sub-Module is overruled and the re-calculation of the robots' paths is controlled by the Communication Supervision Sub-Module. Once the communication is reestablished, the supervision is returned to the Planning Supervision Sub-Module.

4.1 Planning Supervision

In the Central Control Module, after the initial calculation of the robot's paths, done by the Path Planning Module, [9] proposed three critical situations for the Planning Supervision Sub-Module to check:

- If a robot is too distant from its planned position;
- If the maximum difference between steps is 1;
- If there is a robot moving into a position currently occupied by another.

These situations are described as critical since they can lead to collisions and/or deadlocks.

The first situation: verifying if a robot is too distant from its planned position, aims to verify the distance between the most advanced robot's position and the position it would have if it was at the lowest step.

This procedure is done by, firstly, analysing the type of movement of each robot. If the robot is crossing a link, the current steps of every robot are taken into consideration and the lowest step is obtained.

Comparing the coordinates of each robot, at their current step, with the ones they would have if they were executing the lowest step obtained, a difference is computed. If that distance is superior to a certain threshold, the paths are re-planned because the robot was too advanced in its path.

In Figure 4.1 is represented a situation example analysed by the first condition of the Planning Supervision Sub-Module. In the example, being the lowest current step of the fleet step 1, each robot has its current coordinates compared with the ones they would have if they were in step 1. The maximum distance can only be the one defined by the threshold. Therefore, in Figure 4.1, the coordinates indicated by this condition have to correspond, at maximum, with the coordinates of the dashed robot. Since it is expected that, at step one, the robot is on the second node, in this case, this robot would not trigger the supervisor.

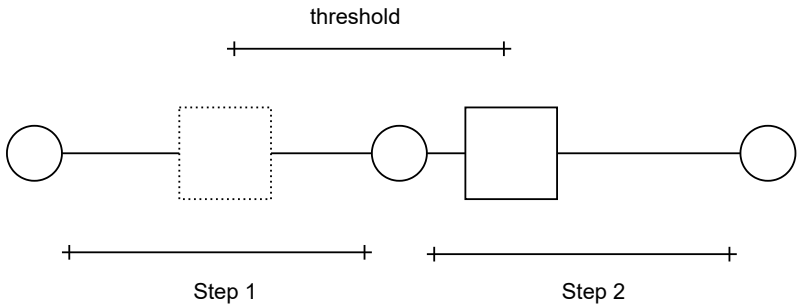
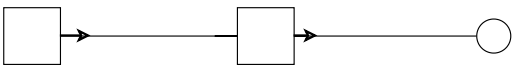


Figure 4.1: Example of the situation that the first condition of the Planning Supervision Sub-Module analyses: In this case, the distance threshold is smaller than the length of a link

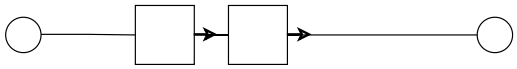
However, this function assumes that all links are superior to the threshold defined and only checks robots that are moving between nodes.

To take into consideration delays during rotations or even stopped robots, condition number two is used. By checking the maximum difference between steps and setting the threshold to one, it is possible to detect delays in robots that are at waiting steps or rotating, and also compensate the previous function if there are links smaller than its threshold.

To detect specific cases of small but dangerous delays, the last condition checks if the node that corresponds to the current step of a robot also corresponds to the current position of another robot. An example of a small delay is represented in Figure 4.2.



(a) Robots are synchronised and Planning Supervision Sub-Module is not triggered



(b) Small delay happened and both robots can collide

Figure 4.2: Example of a small delay that leads to a collision: The last condition of the Planning Supervision Sub-Module aims to detect these situations

All these situations can be covered in only one: verifying when the robots become unsynchronised. If the robots are not synchronised, it means that each robot is at a different step of the planned path, which can lead to collisions and deadlocks. However, triggering the supervisory system by this criterion may lead to an ineffective system where the paths are constantly being re-calculated.

In the next sub-section are described the tests performed to study and understand better how the re-planning of the robot's path affects the execution's time and the number of tasks completed, and which factors influenced the intervention of the supervisor: what caused the delays.

4.1.1 Experiments and Results

To test the Planning Supervision Sub-Module and its intervention, it was assigned a workstation to each robot as written in Table 4.1. After reaching the workstation, each robot moves towards the nearest rest station, which is any other station that is not chosen as a workstation. In Figure 4.3, it is possible to observe the chosen workstations in blue and the available rest stations in red.

Table 4.1: Assigned tasks for Test A

Robot ID	Robot Station	Orientation	Workstation	Rest Station
1	No 8	180 °	No 6	No 7
2	No 2	180 °	No 4	No 3
3	No 7	180 °	No 9	No 10

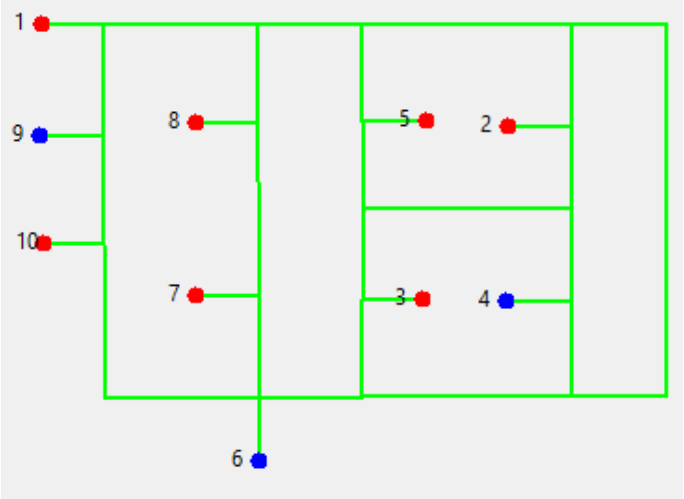
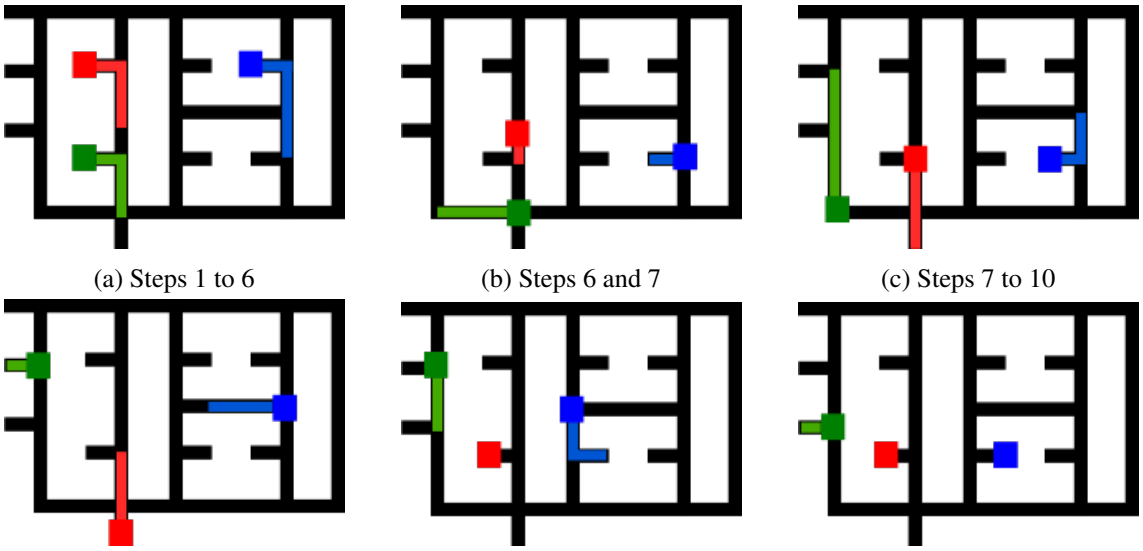


Figure 4.3: Shop floor skeleton graph with the workstations selected in blue

For this mission, the trajectory that each robot should have followed, according to the initial calculation done by the TEA* algorithm, is represented in Figure 4.4 (rotations are not represented). Robot number 1 is coloured in red, robot number 2 is coloured in blue and robot number 3 is coloured in green.



(a) Steps 1 to 6 (b) Steps 6 and 7 (c) Steps 7 to 10

(d) Steps 10 to 14 (e) Steps 14 to 17 (f) Steps 17 and 18

Figure 4.4: Initial path schematic calculated by the Path Planning Module (rotations are not represented)

The following tests aimed to evaluate how many times the supervisor had to intervene and which situations triggered it the most. It was also studied what may have caused the delays and what could influence those situations.

It was expected that the paths were re-calculated more times when the Planning Supervision Sub-Module was triggered by every delay (Test B), instead of re-calculating the paths only on the critical situations (Test A).

Initially, the nominal linear velocity and the nominal angular velocity set in the Robot Control Module was of 800 steps/s (corresponds to 4.09 cm/s) and 100 rad/s, respectively.

Before running the algorithm, it was already possible to predict a critical situation. Since robot number 1 and number 3 have part of their paths in common, the TEA* algorithm defined steps of waiting to prevent the collision of robot number 1 with robot number 3. However, if robot number 3 suffers any delay, robot number 1 can still collide with it.

4.1.1.1 Test A

During this test, the Planning Supervision Sub-Module only searched for the three situations previously mentioned to re-plan the paths.

This mission was executed five times and the results of each sample and the average results are registered in Table 4.2 and Table 4.3, respectively.

The situations that triggered the Planning Supervision Sub-Module the most were a robot being in a step ahead of time (the robot was too distant from its planned position) and the maximum step difference between all robots being superior to one.

Table 4.2: Execution of Test A: Planning Supervision Sub-Module followed the criteria defined previously

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	22	1:50	0	100%
2	23	1:51	0	100%
3	23	1:54	0	100%
4	32	1:47	0	100%
5	21	2:10	0	100%

Table 4.3: Average values of Test A: Planning Supervision Sub-Module followed the criteria defined previously

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	24.2	2:02	0	100%

4.1.1.2 Test B

The mission previously described was also tested, with the same velocity conditions, with the Supervisory Sub-Module analysing and acting whenever any robot became out of sync, instead of waiting for any of the other situations to happen. The results of each sample and the average results are registered in Table 4.4 and Table 4.5, respectively.

Table 4.4: Execution of Test B: Planning Supervision Sub-Module acts upon any delay

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	34	1:47	0	100%
2	37	1:52	0	100%
3	37	1:49	0	100%
4	39	1:45	0	100%
5	36	1:50	0	100%

Table 4.5: Average values of Test B: Planning Supervision Sub-Module acts upon any delay

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	36.6	1:49	0	100%

Comparing with the previous results, the supervisor was called an average of 36.6 times, 12.4 times more. However, since the paths were planed again at the slightest delay, robot number 1 and number 3 never got as close as in the first performance. Therefore, this criterion was able to correct the issues before the other situations triggered the supervisor.

4.1.1.3 Test C

Analysing the number of times that the supervisor intervened in the previous tests, it was analysed what may have caused the delays between the robots.

The situations proposed were the time that it took for a robot to rotate not corresponding to the time that it took to cross a link (since one step corresponds to crossing from one node to the other but corresponds also to a rotation of 90 degrees) or the links (the distance between two nodes) not having a similar size. This last situation was already analysed and verified in Sub-Section 3.2.2.

Due to the topology of the map, it is not possible to change the size of the links. So, to try to reduce the number of delays, it is proposed an adjustment of velocity to try to correspond the time taken by a rotation with the time taken by crossing a link.

Therefore, focusing on the supervisor tested in Test A, the nominal linear velocity of the robots was set at 800 steps/s and the nominal angular velocity at 125 rad/s (increased by 25%).

The results are in Table 4.6 and Table 4.7.

Table 4.6: Execution of Test C: Nominal linear velocity at 800 steps/s and nominal angular velocity at 125 rad/s

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	19	1:47	0	100%
2	21	1:44	0	100%
3	24	1:43	0	100%
4	24	1:42	0	100%
5	24	1:46	0	100%

Table 4.7: Average values of Test C: Nominal linear velocity at 800 steps/s and nominal angular velocity at 125 rad/s

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	22.4	1:44	0	100%

Comparing the results with the ones in Test A (Table 4.2 and Table 4.3), the average number of times that the supervisor re-calculated the path is lower when the angular velocity is 25% superior.

However, increasing too much this value can cause new delays (because the time taken by one step of rotating would be less than the time of one step of going forward). The average execution time is also lower, as expected.

4.1.1.4 Test D

Even though, the average value of the supervisor intervention of Test C, Table 4.7, was lower than the average value of Test A, Table 4.3, having to re-plan the paths more than twenty times represented an exaggerated value.

To solve this issue, the threshold values, that evaluated and triggered any of those three conditions, were reconsidered and amended. The main condition reconsidered was the situation of a robot being too distant from its planned position.

Initially (in Tests A, B and C), the threshold used was 10 cm. This value seemed adequate due to the fact that it was determined that the links should measure, approximately, 15 cm. However,

most of the links measure more than 15 cm. So, a new threshold was set to test how much this factor influenced the re-calculation of the paths.

To test a new threshold, the same mission was performed and it was considered a threshold distance of 15 cm.

The results of this test are presented in Table 4.8 and Table 4.9.

Table 4.8: Execution of Test D

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	5	1:38	0	100%
2	4	1:34	0	100%
3	2	1:38	0	100%
4	4	1:35	0	100%
5	4	1:52	0	100%

Table 4.9: Average values of Test D

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	3.8	1:39	0	100%

As it is possible to observe, the average value of the re-calculations is much lower when combined the velocity adjustment with the threshold adjustment.

However, during the execution of the samples, it was noticed that the threshold distance was too big and was allowing the robots to get too close before triggering the re-calculation. An example of this was, on sample number 4, robot number 1 getting too close to robot number 3 and, when the re-calculation was triggered, robot number 1 was sent back to its previous node instead of just adding waiting steps on its current node/position.

Another test was performed to verify that hypothesis. In this test, the threshold was set at 13 cm. The results are described in Table 4.10 and Table 4.11.

Table 4.10: Execution of Test D'

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	2	1:30	0	100%
2	1	1:29	0	100%
3	1	1:35	0	100%
4	6	1:31	0	100%
5	4	1:33	0	100%

Table 4.11: Average values of Test D'

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	2.8	1:32	0	100%

As it is possible to observe in the results, the execution time between Tests D and D' does not vary significantly, but the threshold of 13 cm reveals to be more adequate and the re-calculation of the paths is done less frequently.

4.1.1.5 Results' Discussion

The time taken by the robots to perform their tasks is slightly longer in Test A. This can be justified due to the situations where the supervisor acts on being more critical and resulting in different paths instead of extra steps on the same node (the robot waiting in the same position).

For example, in Table 4.2, the execution time of sample number 5[1] stands out from the others. This longer performance is justified by the alteration of the path of robot number 1. The addition of extra steps in station number 7, for robot number 1 to wait until robot number 3 has cleared the section, was a consequence of the re-planning done by the supervisor. Even though, in the other samples, it was possible to observe robot number 1 starting to rotate in the direction of station number 7, the robot never moved into that node/station, resulting in a shorter execution time.

In sample number 4 of Test A[2], it was also possible to observe robot number 2 taking an alternative path to reach its rest station. The execution time was not significantly affected but the supervisor had to re-calculate the paths 32 times.

When it comes to the results of Test B, sample number 4[3] is also distinct. Even though the supervisor re-calculates the paths more frequently, the execution time is a little shorter than on the other samples. This can be a result of the supervisor being called on simpler situations that do not involve a change of the path and are quickly sorted.

With Tests A and B, it is possible to verify that there is a trade-off between the number of times that the path is calculated (re-calculating the paths frequently can be a computationally heavy procedure) and the mission's execution time.

If the supervisor is only called on critical situations, the algorithm will be executed faster and it will not be as computationally heavy but the robots' may take longer to perform their missions due to alternative and longer paths.

Contrarily, if the paths are re-calculated every time a robot becomes unsynchronised, the most likely situation to happen is the TEA* algorithm adding extra steps on the current node, making the robots wait for each other and, consequently, avoiding longer paths.

To test the impact of the velocity on the delays and, consequently, on the number of paths' re-calculation, in Test C, the nominal angular velocity was increased. This modification aimed to observe if the rotation's velocity was similar to the velocity of going forward. As expected, the execution time decreased.

Comparing the results from Test C and Test A, the average number of the supervisor interventions was inferior in Test C. This means that the angular velocity in Test C was more adequate. However, the results are not significantly different.

[1] https://www.youtube.com/watch?v=7tiBd8hPfKE
[2] https://www.youtube.com/watch?v=XeMGC1BlOg8
[3] https://www.youtube.com/watch?v=NmdK5b0vj64

Since the situation that triggered the supervisor the most was the robot being too distant from its planned position, that condition was evaluated.

In Tests A, B and C, the threshold used to evaluate this condition was 10 cm. This value seemed adequate due to the fact that it was determined that the links should measure, approximately, 15 cm. However, not all links measure the same. So, a new threshold was set to test how much this factor influenced the re-calculation of the paths.

With a proper velocity adjustment and with a better threshold, the number of re-calculations decreased. In Test D, two threshold values were tested to obtain the best fit, not only to avoid excessive re-planning but also to guarantee that the re-calculation happened at the right moment.

In sample number 5 of Test D, once again, the execution time stood out from the others due to the same reason as sample number 5 of Test A.

Through sample number 4 of Test D[4], it was possible to observe robot number 1 getting too close to robot number 3 before triggering the Planning Supervision Sub-Module (0:18). When the paths were re-calculated, robot number 1 was sent back to its previous node instead of waiting in its current position. This meant that the threshold used for the first condition was too big.

To avoid situations similar to that, a threshold of 13 cm was tested in Test D'. This new value revealed to be the adequate maximum distance between the most advanced robot's current coordinates and the coordinates it would have if it was in the delayed robot's current step. This value translates to less than a step of difference (since one step represents crossing one link-approximately 15 cm) between the most advanced and the slowest robot. With this new threshold, the re-calculation of the paths was trigger adequately, as it is possible to watch in sample number 5 of Test D'[5].

For this mission, the trajectory that each robot should have followed, according to the initial calculation done by the TEA* algorithm, is represented in Figure 4.4 (rotations are not represented). In most cases, the paths were mainly maintained because the re-calculation only forced the robots to stop and wait until all robots were in sync, the only exceptions were the previously mentioned samples.

4.2 Communication Supervision

As explained before, to prevent collisions and deadlocks, is extremely important that the supervisor is tolerant of communication failures. Hence, the Communication Supervision Sub-Module was implemented in order to detect communication faults, determine their size and forcing the re-calculation of the paths, considering the fault and the possible area affected. The type of communication faults that can affect the environment are areas of the shop floor map that consistently have no communication with the Central Control Module and temporary loss of connection. In both cases, the system does not have any prior knowledge of the faults' location.

[4] https://youtu.be/Zi3IVC1MvTc
[5] https://youtu.be/39bkMAfT_fg

During the normal functioning of the system, the Communication Supervision Sub-Module searches for communication faults constantly. When a communication fault is detected, the normal functioning of the Central Control Module is interrupted and the fault and its dimension is studied. After estimating the size of the area with no communication, the re-calculation of the paths is forced in order to prevent any other robot to enter that area. The system then returns to its normal functioning.

The Central Control Module also interrupts its normal functioning when it is detected that a robot exited the area of the communication fault and the system is again capable of locating the robot. In this case, the fault has its size and location adjusted.

4.2.1 Persistent Communication Faults

Initially, when no robot is inside an area without communication, while the robots are executing their missions, the nodes that they cross are mapped and labelled as "unfaulted" nodes (since it is possible to establish communication with the robots in them). These nodes will help estimate the size of a persistent communication fault's area. When it comes to sporadic communication faults, all nodes can lose communication unexpectedly. Therefore, the node status as "unfaulted" may not be permanent.

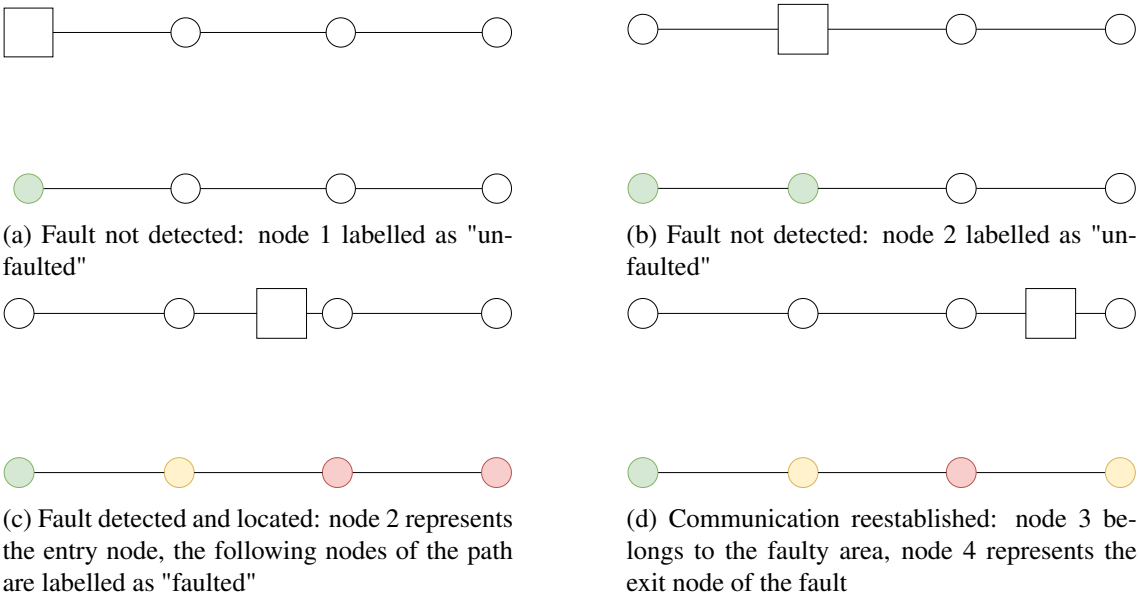
When a communication fault is detected, one of these two situations happened: either the robot entered a fault not mapped yet or the fault has already been mapped. The robot that lost communication is immediately associated with the area of the communication fault and, if the area has not been mapped, the robot is labelled as mapping the faulty area.

In the case of a robot entering a communication fault area that has not been mapped yet, the path of the robot is analysed to determine the most likely exit node and where the communication can be reestablished. To implement this, the nodes of the following steps of that robot are compared with the nodes already mapped as "unfaulted". If the node is not mapped then it is included in the faulty area. This comparison is done until one of the path's node is found to be mapped as "unfaulted" or the path ends. This set of faulted nodes is considered an area where no communication can be established. The possible entry/exit nodes are also associated with each area.

If the robot exits the faulty area before reaching the estimated exit node, the faulty area's dimension is corrected as well as the possible entry/exit nodes. The fault's exit is corrected by locating the current robot's node, and the following nodes, that were previously included in the communication fault, are removed from the fault's structure. The nodes that were removed from the fault are labelled as not mapped and, after exiting the area, the robot is no longer associated with that fault.

In Figure 4.5, it is exemplified the procedure of detecting, locating, estimating and correcting (when communication is reestablished) the area of a communication fault.

On the other hand, if a robot enters a communication fault and the system already has communication faults registered, it is tried to associate the current fault with the ones already mapped.



(a) Fault not detected: node 1 labelled as "unfaulted"

(b) Fault not detected: node 2 labelled as "unfaulted"

(c) Fault detected and located: node 2 represents the entry node, the following nodes of the path are labelled as "faulted"

(d) Communication reestablished: node 3 belongs to the faulty area, node 4 represents the exit node of the fault

Figure 4.5: Schematic of a situation example of the detection, location, estimation and correction of the area of a communication fault

If the entry node corresponds to an entry node of another fault then the fault's location is extended and it is added the not mapped nodes that belong to the robot's path.

If the entry node does not correspond to the entry nodes of any of the mapped faults, the method used tries to recognise if 60% of the path's nodes of the robot are located at the already mapped faults. If at least 60% of the nodes do not belong to any of the mapped faults, it is considered that the robot is mapping a new zone with communication fault and the procedure explained initially is repeated.

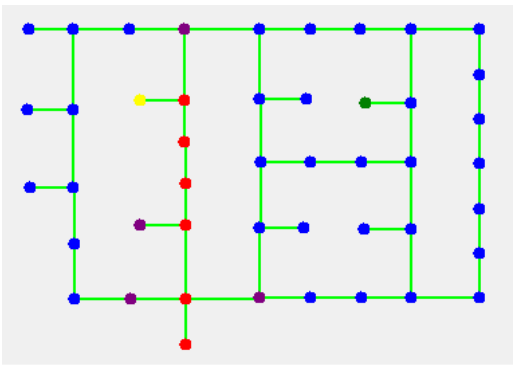
In the case of 60% of the nodes corresponding to an already mapped fault but the robot's entry node does not correspond to the entry nodes of the fault, it is adopted the procedure of merging the two sets of nodes: the nodes from the robot's path that are not mapped and the nodes of the fault already mapped. As a result, the already mapped fault has its dimension extended and the nodes of the robot's path are included.

For the example of Figure 4.6, the chosen stations are in Table 4.12. The area of the fault, since it was induced artificially, is represented in Figure 4.7.

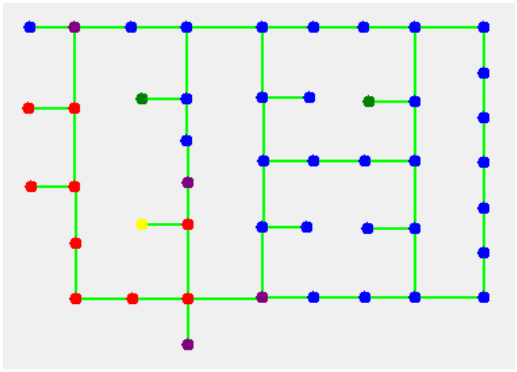
Table 4.12: Mission's assigned tasks

Robot ID	Robot Station	Orientation	Workstation	Rest Station
1	No 8	0 °	No 6	No 7
2	No 2	0 °	No 4	No 3
3	No 7	0 °	No 9	No 10

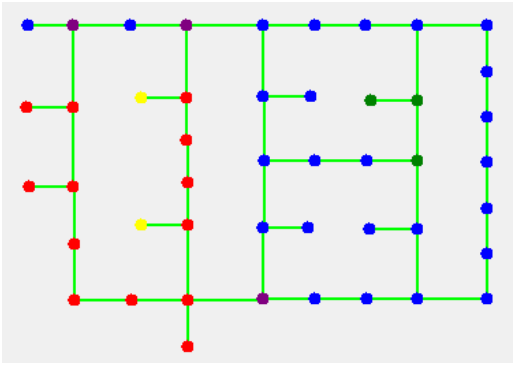
The example in Figure 4.6 only happens if the two robots entered the fault at the exact same time. Otherwise, the TEA* algorithm re-calculates the path and avoids the fault's area by choosing



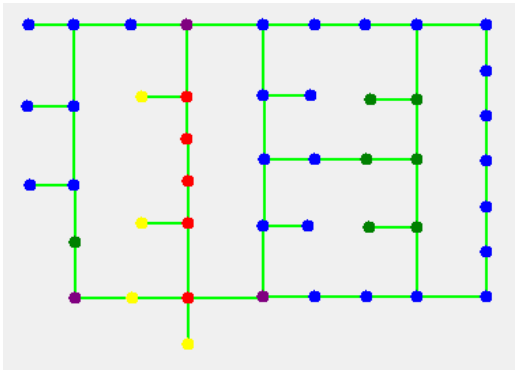
(a) First detected, located and estimated fault



(b) Dimension of the second fault



(c) Combined fault due to merging process



(d) Combined fault adjusted after it is mapped

Figure 4.6: Merge of the faults through common "faulted" nodes: red nodes represent "faulted" nodes, yellow nodes represent entry/exit nodes, purple nodes represent frontier nodes, green nodes represent "unfaulted" nodes and blue nodes represent not mapped nodes

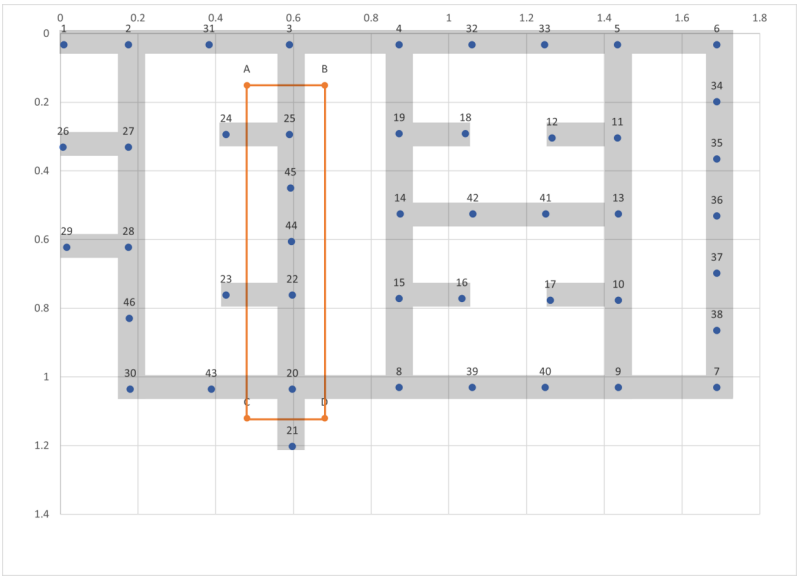


Figure 4.7: Borders of the fault's area (orange) and where it is located in the environment's graph

an alternative path for robot number 1 or by forcing robot number 3 to wait until the first robot reestablished communication.

For the same mission and induced fault, if robot number 1 and 3 do not enter at the same time (if robot number 3 enters the fault first), the fault would be seen as two separate faults.

Therefore, it was tested the same situation but with different initial orientations. The assigned tasks and initial positions are described in Table 4.13 and the fault induced is the same as Figure 4.7.

Table 4.13: Mission's assigned tasks with different initial orientations

Robot ID	Robot Station	Orientation	Workstation	Rest Station
1	No 8	180 °	No 6	No 7
2	No 2	0 °	No 4	No 3
3	No 7	0 °	No 9	No 10

During the execution of this mission[6], robot number 3 stepped into the fault first. Once robot number 3 entered the fault, robot number 1 had its path re-calculated. As robot number one entered the fault, the system compared the new trajectory of robot number one with the fault. Since there are no nodes in common, the system assumed two different faults. The mapping of the shop floor map during this mission is represented in Figure 4.8.

Through the video of the execution of this mission with a fault, it is possible to observe an alternative path chosen for robot number 1 in order to avoid the first estimated fault. As soon as robot number 3 exited the fault, the paths of robot number 1 and 2 were re-planned and the first alternative path of robot number 1 (that was temporary also marked as a fault location and can be seen in Figure 4.8b) was not executed.

Repeating this mission, with the mentioned fault, it was analysed the execution time and the re-calculation of the paths (supervisor intervention: re-calculation called by Planning Supervisor and Communication Supervisor). The results are in Table 4.14 and Table 4.15.

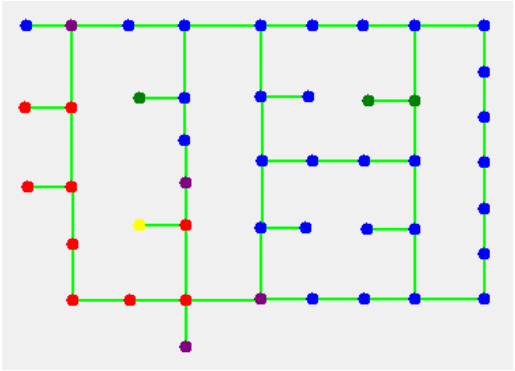
Table 4.14: Execution of Mission with Fault: Robot 3 enters fault before robot 1

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	16	1:51	0	100%
2	15	1:50	0	100%
3	11	1:48	0	100%
4	19	1:49	0	100%
5	18	1:42	0	100%

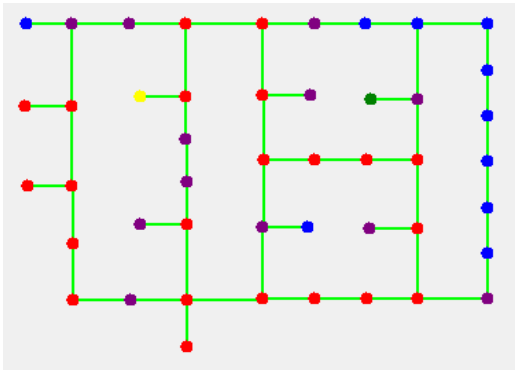
Table 4.15: Average values of Mission with Fault: Robot 3 enters fault before robot 1

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	15.8	1:48	0	100%

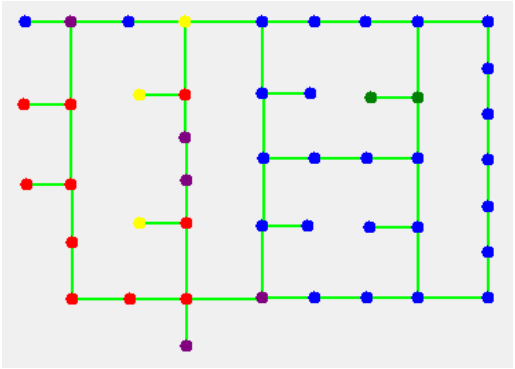
[6] https://youtu.be/NlXUYYPKLvw



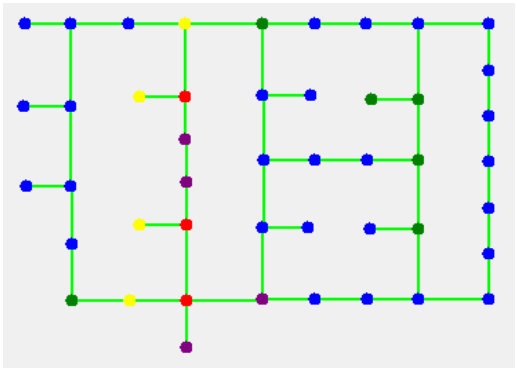
(a) First detected, located and estimated fault



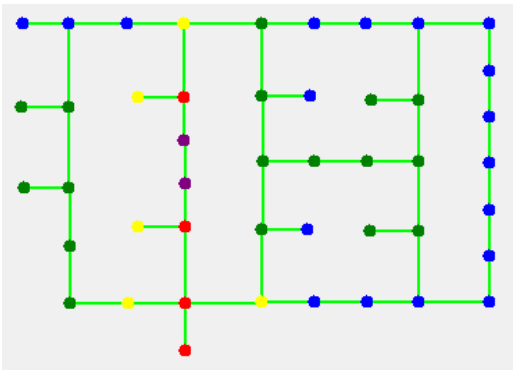
(b) Second detected, located and estimated fault



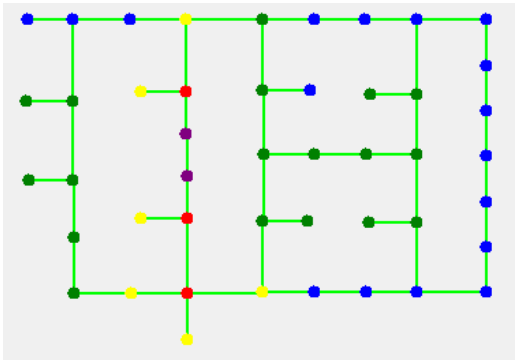
(c) Faults are interpreted as being distinct faults: fault 2 is adjusted



(d) Robot number 1 exits first fault and it is adjusted



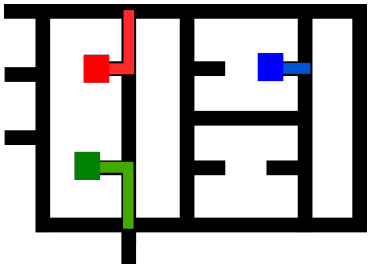
(e) Robot number 2 reentries fault and adjusts it



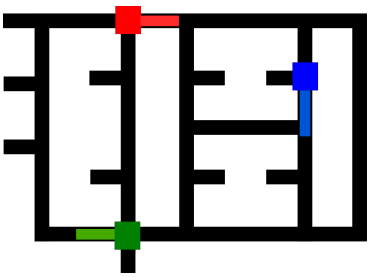
(f) Robot number 2 exits first fault and finishes its mission

Figure 4.8: Mapping of the faults (the induced fault is interpreted as two different faults) and crossed nodes: red nodes represent "faulted" nodes, yellow nodes represent entry/exit nodes, purple nodes represent frontier nodes, green nodes represent "unfaulted" nodes and blue nodes represent not mapped nodes

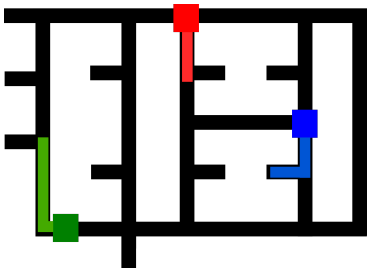
The initial path calculated by the Path Planning Sub-Module is represented in Figure 4.4. However, the paths that were performed are represented in Figure 4.9. During the steps represented in Figure 4.9a, the fault is detected but interpreted as two different communication faults. In steps of Figure 4.9b, both robot number 1 (red robot) and robot number 3 (green robot) exit the fault and the paths are re-calculated resulting in the final trajectories.



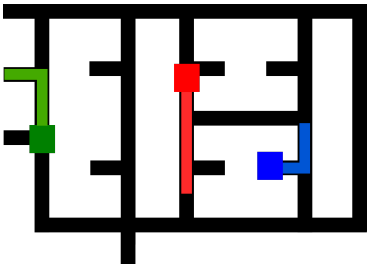
(a) Fault is detected but seen as two different faults



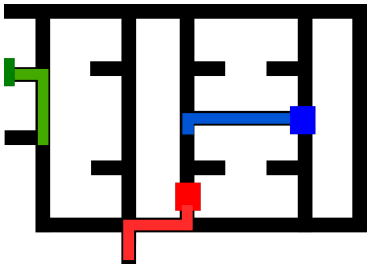
(b) Robot number 1 exits the fault and robot number 2 proceeds its path



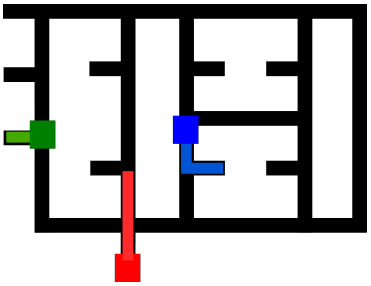
(c) Robot number 3 exits the fault



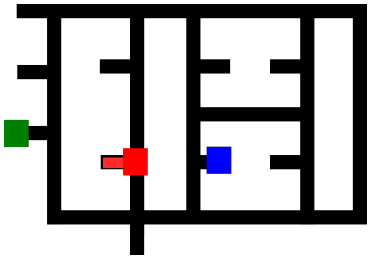
(d) No robots are inside the fault: recalculations are trigger by the Planning Sub-Module



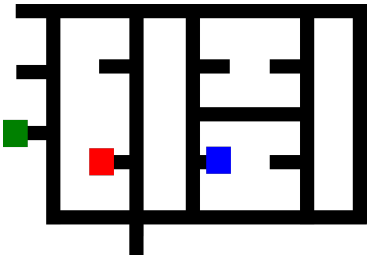
(e) Robot number 1 enters the communication fault again to enter station 6



(f) Robot number 1 reestablishes communication when in station 6 but reenters fault when moving towards its rest station



(g) Robot number 1 is inside the fault



(h) Robot number 1 reestablishes communication once it reaches the rest station

Figure 4.9: Robots' trajectories during mission with two faults

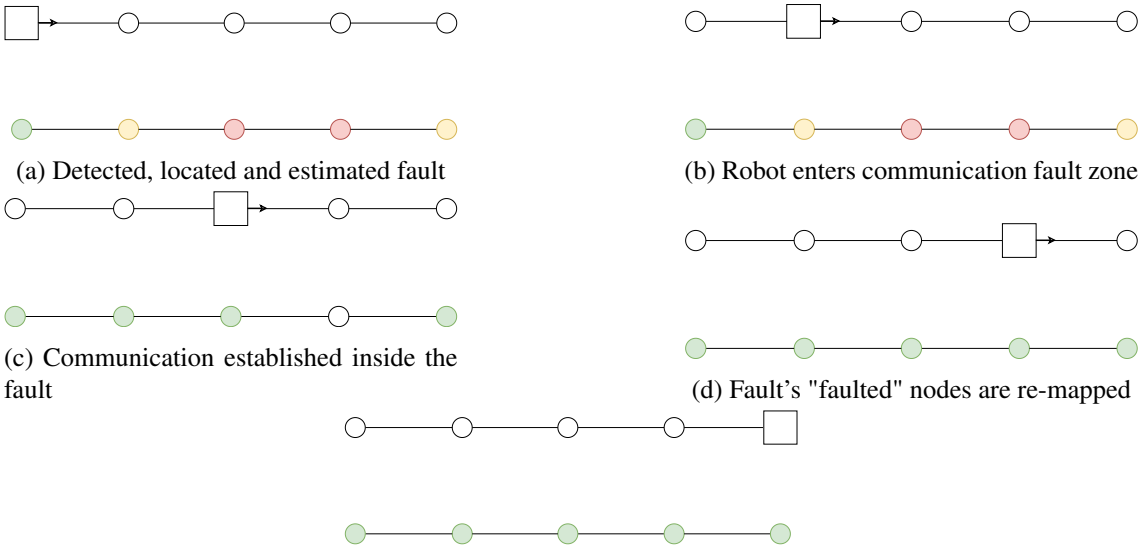
As it is possible to observe, when a robot enters a faulty area, the estimated location of the fault is locked and that area is prevented from other robots to enter and cross it. However, it is impossible to prevent two or more robots to enter the same fault from different entry nodes. When that situation happens, the process of merging may happen and the robots map, simultaneously, the fault's area. Inside the fault, the collision/deadlock prevention is done only through the Robot Control Module, as explained in Sub-Section 3.4.3.

4.2.2 Sporadic Communication Faults

In a real multi-AGV environment, most communications faults that may occur do not represent steady areas where communication is always down. Hence, the system had to be capable of dealing with sporadic faults.

The function implemented starts by analysing, as the robots execute their missions and cross the map's nodes, the nodes in which communication can be established. The node that corresponds to the robot's position is compared to the nodes in the faults' areas. If the node is included in a fault, it means that the node, and probably the rest of the fault's area, is no longer subjected to communication faults. So, the fault associated with that node is removed and all the "faulted" nodes are labelled as "not mapped" except for the node where the communication was established: that node is placed as "unfaulted". The entry/exit nodes of the fault are kept as "unfaulted". An exemplification of this process is represented in Figure 4.10.

Through this method, it is possible to remove any sporadic communication faults that may occur as well as remove inactive persistent communication faults. However, even if the status of the nodes are changed, the Path Planning Sub-Module (TEA* algorithm) still takes into consideration that the nodes were once subjected to communication faults, as will be further explained in Sub-Section 4.2.3.



(a) Detected, located and estimated fault

(b) Robot enters communication fault zone

(c) Communication established inside the fault

(d) Fault's "faulted" nodes are re-mapped

(e) The fault is removed

Figure 4.10: Schematic of fault removal

4.2.3 TEA* with Dynamic Costs

As mentioned before, when a communication fault is detected and estimated, the area is locked and prevented from other robots to enter. However, as soon as the robot exits the area, nothing prevents the other robots from crossing that area.

Even though, as mentioned before, the system implements procedures to avoid collisions and deadlocks during communication failures, it is safer to avoid crossing these areas when there are alternative paths available.

To minimise the paths that cross the areas of the communication faults, it was incorporated a dynamic cost on the nodes on the TEA* algorithm. Therefore, the nodes affected by communication faults present an extra cost when being considered for the optimal path. The extra cost incorporated is proportional to the probability of the node suffering communication faults. Even if a fault is sporadic, the TEA* still takes into account that the node once suffered from communication faults and it is not as reliable as other "unfaulted" nodes.

During the execution of the robots' missions, it is calculated the probability of the crossed node being exposed to communication faults. This reliability factor is then added to each node in the TEA* algorithm.

As mentioned in Sub-Section 2.2.2.2, the TEA* algorithm uses a heuristic function $f(n)$, described by Equation 4.1, to search through the map's graph for an optimal path from an initial node to a destination node.

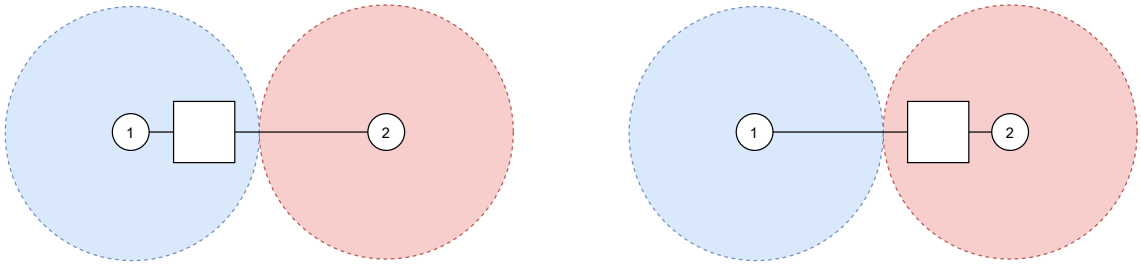
$$f(n) = h(n) + g(n) \tag{4.1}$$

The function $h(n)$ represents the current cost from the initial node to the n node. The function $g(n)$ represents the cost from going from the n node to the destination node. To consider the probability of the node suffering from communication faults, an extra cost is added to the $g(n)$ function, Equation 4.2.

$$g(n) = g(n) + COST \cdot prob \tag{4.2}$$

It is important to understand that, even though a node is described as having only an X and Y coordinate, during the execution of the missions, a node ends up representing a circular area. Analysing the situation in Figure 4.11, while a robot is crossing the link between node 1 and 2, during the first half of the link, the robot is still associated with node 1.

Once it crosses the first half of the link, the system identifies the robot as being on the second node. Therefore, even if a node is included in the location of a communication fault area, it is not guaranteed that the probability of being affected by the communication fault is 100%. This is a common situation for entry/exit nodes.



(a) Robot is associated to node number 1 (b) Robot is associated to node number 2

Figure 4.11: Example of how the nodes represent a circular area and not just a point

To observe the impact of the dynamic cost, it was performed the mission in Table 4.16. The goals of the following tests, Test E and Test F, are analysing the number of times the faulted area is crossed, the number of re-calculations of paths and the execution time. The two tests explore two situations:

- The TEA* algorithm considers an extra cost which is proportional to the probability of the node presenting communication faults;
- The TEA* algorithm considers that every node, with or without communication faults, has the same cost;

It is expected that, when the TEA* algorithm considers an extra cost in nodes with a probability of losing communication with the system, the optimal paths chosen avoid the area unless there are no alternative routes. The execution time is predicted to be longer because of the longer alternative paths.

When it comes to the number of the paths' re-calculations, even though every time a robot enters and exits a communication fault the supervisor re-plans the paths, when a robot takes an alternative longer route, the probability of happening delays is higher.

Table 4.16: Dynamic Cost Tests: Mission's assigned tasks

Robot ID	Robot Station	Orientation	Workstation	Rest Station
1	No 6	90 °	No 4	No 2
2	No 7	180 °	No 3	No 5
3	No 9	180 °	No 10	No 9

The workstations, nodes coloured in blue, and the localisation of the rest stations chosen can be consulted in Figure 4.12.

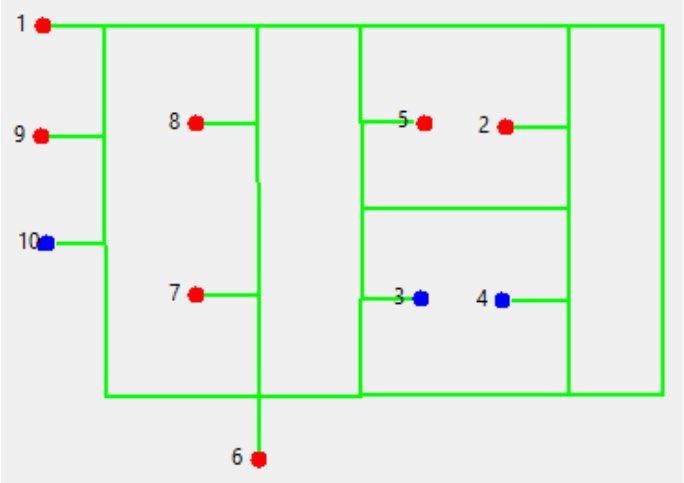


Figure 4.12: Shop floor skeleton graph with the workstations selected for Test E and Test F (blue nodes)

The communication fault induced in Test E and Test F is presented in Figure 4.13, where is possible to identify the nodes that will be affected.

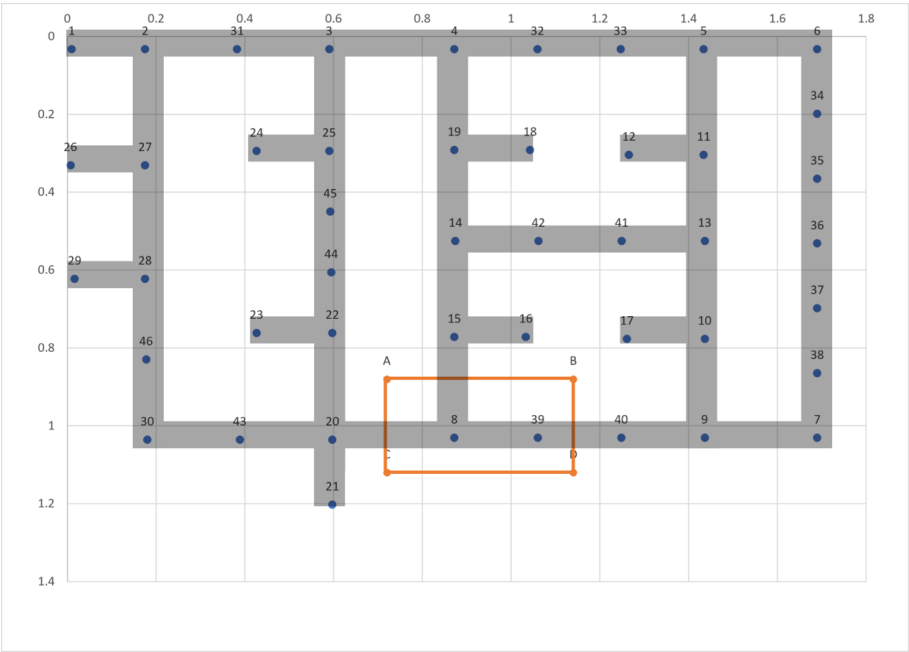
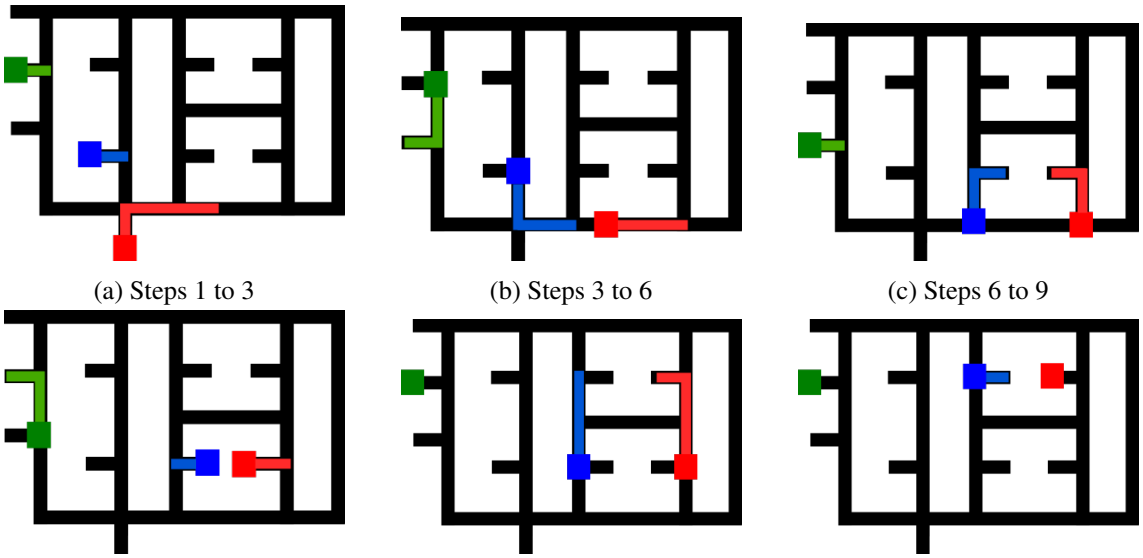


Figure 4.13: Localisation of the induced fault (borders at orange) in Test E and Test F on the map's graph

The initial trajectory calculated by the Path Planning Sub-Module is represented in Figure 4.14, where robot number 1 is coloured in red, robot number 2 is coloured in blue and robot number 3 is coloured in green. However, since a communication fault is going to be induced, it is possible to predict that the paths will not be exactly as described in Figure 4.14.



(a) Steps 1 to 3 (b) Steps 3 to 6 (c) Steps 6 to 9

(d) Steps 9 to 12 (e) Steps 12 to 15 (f) Steps 15 to 17

Figure 4.14: Initial path schematic, calculated by the Path Planning Module (rotations are not represented) for the mission of Test E and Test F

4.2.3.1 Test E: TEA* algorithm without dynamic cost

The execution of five samples of the mission without considering a dynamic cost resulted in the values on Table 4.17 and Table 4.18.

Table 4.17: Execution of Mission with Fault: TEA* algorithm does not distinguish nodes affected by communication faults

#	Supervisor Intervention	Exec. Time (min.)	Fault's Area Crossed	Tasks Completed
1	7	1:31	2	100%
2	8	1:36	2	100%
3	6	1:35	2	100%
4	6	1:35	2	100%
5	7	1:39	2	100%

Table 4.18: Average values of Mission with Fault: TEA* algorithm does not distinguish nodes affected by communication faults

#	Supervisor Intervention	Exec. Time (min.)	Fault's Area Crossed	Tasks Completed
5	6.8	1:35	2	100%

During the execution of Test E[7], it can be observed clearly (0:23) that, after robot number 1 exits the fault, robot number 2 has its path re-calculated and rotates in order to execute the original path because it reveals to be the fastest even if it presents communication faults.

[7] https://youtu.be/8jxzUIGUT8k

4.2.3.2 Test F: TEA* algorithm with dynamic cost

Considering an extra cost, as explained in Equation 4.2, the mission was again executed five times. The results can be consulted in Table 4.19 and Table 4.20.

Table 4.19: Execution of Mission with Fault: TEA* algorithm considers an extra cost

#	Supervisor Intervention	Exec. Time (min.)	Fault's Area Crossed	Tasks Completed
1	6	1:45	1	100%
2	5	1:52	1	100%
3	2	1:46	1	100%
4	6	1:40	1	100%
5	4	1:46	1	100%

Table 4.20: Average values of Mission with Fault: TEA* algorithm considers an extra cost

#	Supervisor Intervention	Exec. Time (min.)	Fault's Area Crossed	Tasks Completed
5	4.6	1:46	1	100%

When executing Test F[8], the fact that robot number 1 exited the communication fault had no impact on the path of robot number 2. The path executed by robot number 2 was longer but avoided crossing a communication fault.

4.2.3.3 Results' Discussion

Observing the average execution time of both tests, through Table 4.18 and Table 4.20, the average execution time on Test F is longer. This is a consequence of avoiding the communication fault that is located on the optimal path of robot number 2.

Even though, on Test E, robot number 2 starts executing the alternative route (because the faulty area is locked and no robot can cross it while robot number 1 is inside it), when robot number 1 leaves the communication fault, the paths are re-planed. At the moment of the re-calculation of the paths, robot number 2 is further away from that path than initially but the optimal path still remains the same as calculate in the beginning. Therefore, robot number 2 inverts its movement and follows the initial path.

On Test F, after it is detected the communication fault, robot number 2 avoids that area and it completes the alternative path even after robot number 1 exits the fault. The initial path still reveals to be the fastest after the faulty area is unoccupied. However, because of the dynamic cost, the TEA* algorithm discards that path unless no alternative routes are available. This reflects on a longer execution time. Nevertheless, it is safer to adopt paths with no communication faults.

The re-calculation of the paths (the intervention of the Supervisor) happens more often in Test E. Considering the average value of 6.8 (Table 4.18), it is possible to affirm that 4 of those re-calculations were due to the entry and exit of robot number 1 and robot number 2 on the communication fault's area. The number of times the Communication Supervision Sub-Module had

[8] https://youtu.be/VSPS3h5hxeY

to intervene is two times (re-plans when a robot enters the fault and when it exits it) multiplied by the number of times a fault's area is crossed.

However, the alternative paths are usually longer, which will cause the mission to take a longer time. Hence, is also probable that delays happen and re-calculations need to be done by the Planning Supervision Sub-Module.

4.3 Conclusion

The supervisor has the responsibility of evaluating potentially dangerous situations that may lead to deadlocks and collisions in the multi-AGV environment. The key moments for preventing those situations are detecting significant changes on the robots' path (calculated by the TEA* algorithm: Path Planning Sub-Module) and allowing their re-calculation, and the detecting and controlling communications faults.

To execute those tasks, the supervisor is divided into two sub-modules: Planning Supervision Sub-Module and the Communication Supervision Sub-Module.

Throughout this chapter, the sub-modules of the supervisor of the system were explained in greater detail.

The Planning Supervision Sub-Module monitors deviations and delays in the routes of the robots by controlling three specific situations. The implementation of this sub-module had the aim of replacing the use of the TEA* algorithm, as an online method that re-plans the paths at every iteration.

The conditions monitored by the supervisor were described and tested in this chapter. It was compared the efficiency of tracking those three conditions to re-calculating the paths every time a delay was detected (when the robots were not all at the same step on the trajectory).

It was also studied the factors that influence the Planning Supervision Sub-Module the most. Two hypotheses were made: the significant variation of the length of the links of the map and an inadequate linear and angular velocity. Due to the map's topology, it was only tested the variation of the velocity, since the length of links could not be changed.

The conditions of the Planning Supervision Sub-Module were also tested in order to find adequate thresholds to the environment's dynamics.

The main goal of the Communication Supervision Sub-Module is to detect communication faults and guarantee secure and efficient paths when stable communication cannot be established. To operate accordingly, this sub-module locates and estimates the dimension of the communication fault to allow the TEA* algorithm to calculate safe routes, avoiding those locations.

The Communication Supervision Sub-Module can register two types of faults: persistent and sporadic. Persistent communication faults often represent bigger areas whereas sporadic faults are temporary losses of communication in a few nodes. However, if a fault becomes inactive, the supervisor proceeds the same way for both cases.

To avoid locations with communication faults, even if no robot is crossing that area, the Path Planning Sub-Module incorporated, in the TEA* algorithm, a reliability factor depending on the

node. As explained in this chapter, an extra cost is added to a certain node depending on its probability to suffer communication faults.

Chapter 5

Communication Fault Recovery

In the previous chapter, it was explored how the Supervisor is composed and how the different sub-modules interact with each other and with the changes in the environment.

One of the major concerns about the coordination of AGV is the possible presence of communication faults. The Communication Supervision Sub-Module revealed to be quite effective in the detection, localisation and estimation of communication faults during the execution of the robots' missions.

However, if a robot finishes its mission while on a communication fault, the system cannot reach out to that robot and it will be permanently stuck on its rest station until communication can be reestablished.

In order to solve this problem, it was developed, as a part of the Communication Supervision Sub-Module, a Recovery Mode.

5.1 Recovery Mode

In the Central Control Module, during the execution of a mission, if it is detected that a robot just entered an area without communication, a timer is set and associated with that robot.

The timer aims to predict when will the robot finish its mission if it does not reestablish communication with the Central Control Module until the end of it. Therefore, until communication is reestablished, the timer is always running and being checked.

The prediction of the timer's value is based on the number of steps that the robot still had to complete and an estimation of the execution time of one step (that estimation was explained in Sub-Section 3.4.4).

Once the value of the timer is reached, the Supervisor Sub-Module activates the Recovery Mode. This mode is activated with the intention of preventing the robot from being stuck, indefinitely, on its rest station (since the robot cannot establish communication with the Central Control).

The Recovery Mode assumes that, approximately after the timer ends, the lost robot, which is aware that the communication is down, will find a path from its current workstation to the nearest

workstation. Therefore, it is necessary to coordinate the rest of the fleet to avoid the lost robot and the possible exit area.

Firstly, the supervisor assesses the nearest rest station to the current rest station of the lost robot. The sub-missions of the robot are registered as completed and another sub-mission is added: shifting of rest stations. This update on the sub-missions will allow the reestablishment of communication with the robot to happen smoothly without incoherent sub-missions and outdated steps. Since the robot is inside a faulty area, the new trajectory is included in the communication fault's data structure. It is expected that, when the robot reestablishes communication, the fault will be updated and corrected.

After obtaining the station and updating the lost robot's mission and the fault's area, it is necessary to coordinate the rest of the AGV fleet so that collisions and deadlocks can be avoided. The Path Planning Sub-Module is called to intervene and re-calculates the paths according to the possible exit path of the lost robot.

To determine new paths, it is given the higher priority to the lost robot and the faults' locations are not taken into consideration when planning the lost robot's path. These two features are important because it is not expected for the Robot Control Module to know the location and the paths of the rest of the fleet and neither to have access to the communication faults' location.

As it was mentioned, ideally, in these situations, the Robot Control Module would have access to the map's graph, the available stations (workstations and rest stations) and to a graph search algorithm to be able to calculate a new trajectory. This new trajectory would move the robot from its current rest station to the nearest one where communication with the Central Control can be established.

However, because of the structure of the Robot Control Module, it is not possible for the robot to obtain a new trajectory for itself. To test the Recovery Mode, and since the communication faults are induced, this process is simulated. After the Path Planning Sub-Module re-plans the new paths, a single UDP message is sent to the Robot Control Module with the lost robot's new trajectory.

With new paths calculated for the connected robots and a predicted path for the lost robot, a new timer is set and the Recovery Mode goes to sleep until activated again. The Supervisor Sub-Module returns to its normal functions.

Another measure taken to prevent these situations is the evaluation of the location of all rest stations after a robot exits a communication fault. When communication is reestablished with a lost robot, not only the fault's area is updated but also the chosen rest stations are compared with the fault's nodes. If any of the rest stations' nodes belong to the fault, the attributed rest stations are changed to the closest with communication.

To observe the performance of the Recovery Mode, different missions were chosen and executed.

5.2 Tests and Results

To test the supervisor response to this critical situation, some tests were performed.

Test G and Test H intend to evaluate not only the new path from the new sub-mission of the lost robot: the movement from the first designated rest station to the nearest one, but also the coordination of the rest of the fleet in order to avoid collisions and deadlocks, specially on the expected exit area of the communication fault.

Test G assigns simple tasks to the robots and it is tested the system's performance with a induced fault. Next, the same mission is tested (Test G') to allow to identify and compare the impact of the fault on the nodes (dynamic cost), on the re-calculation of the paths (supervisor intervention), on the execution time and on the activation of the Recovery Mode.

Test H is then performed to test the same parameters. However, the tasks assigned are more complex and it is expected a more careful coordination on the Central Control Module's behalf. The robots' performance is evaluated as well as the previous mentioned parameters .

5.2.1 Test G

The first mission tested to analyse the Recovery Mode is described in Table 5.1 and in Figure 5.1.

Table 5.1: Assigned tasks for Test G

Robot ID	Robot Station	Orientation	Workstation	Rest Station
1	No 8	180 °	No 6	No 7
2	No 2	180 °	No 4	No 3
3	No 7	180 °	No 1 & 9	No 10

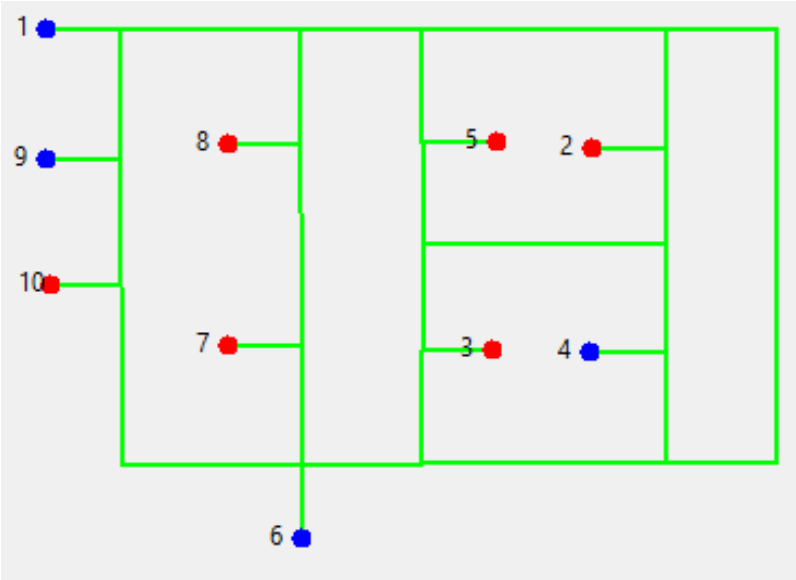


Figure 5.1: Shop floor skeleton graph with the workstations selected for Test G

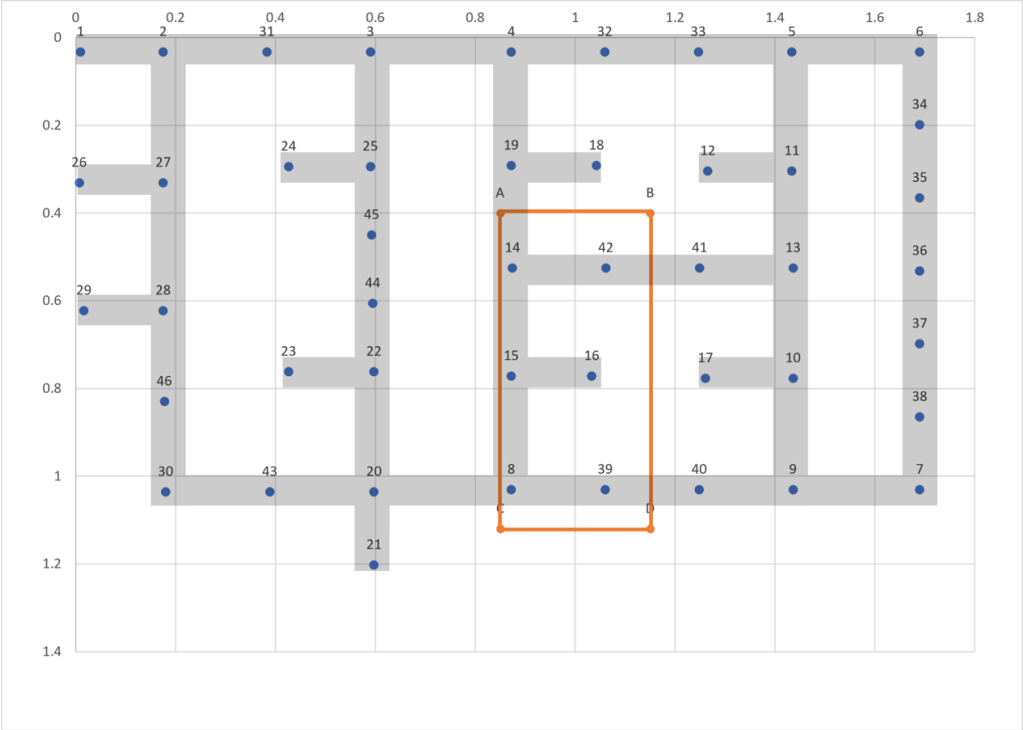


Figure 5.2: Induced fault (borders at orange) for Test G

The location of the induced fault is represented on the map's graph in Figure 5.2. As intended, the rest station of robot number 2 is located inside the fault.

After the Recovery Mode is activated, it is expected for robot number 2 to change its rest station from number 3 (node 16 in Figure 5.2) to rest station number 5 (node 18 in Figure 5.2).

The test was performed five times and the results are described in Table 5.2 and Table 5.3.

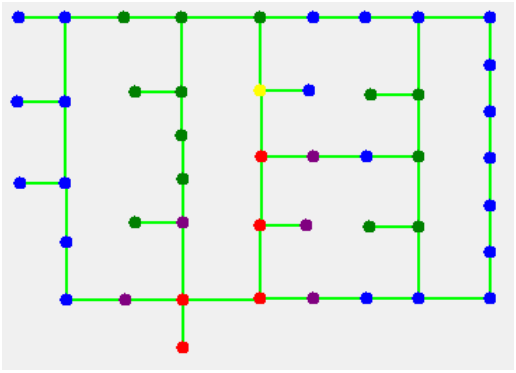
Table 5.2: Execution of Test G

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	10	2:19	0	100%
2	11	2:13	0	100%
3	10	2:16	0	100%
4	8	2:16	0	100%
5	8	2:17	0	100%

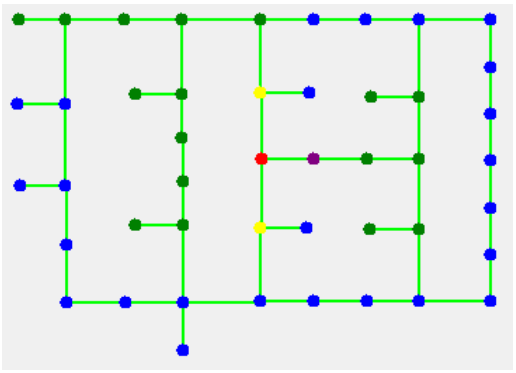
Table 5.3: Average values of Test G

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	9.4	2:16	0	100%

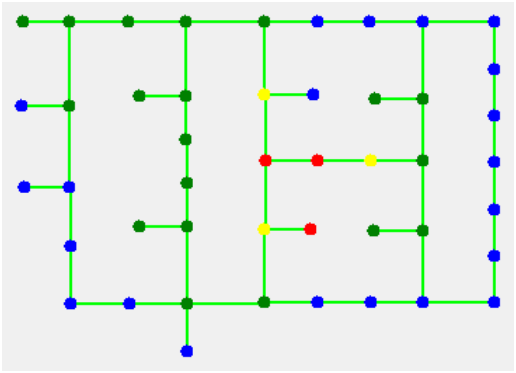
The execution times of the samples are very similar. Since this fault is firstly mapped by robot number 1 (before robot number 2 becomes stuck inside), the paths are re-calculated four times due to the fault. The other re-calculations were cause by the delays, as studied in Section 4.1.



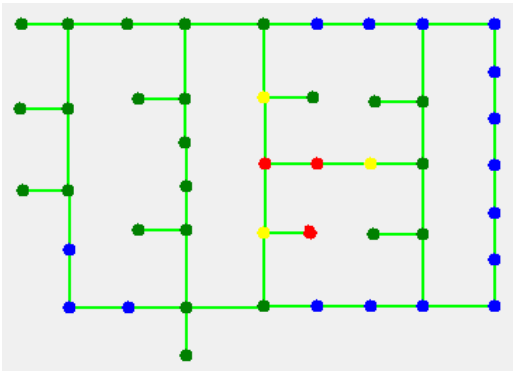
(a) Fault detected, located and estimated by robot number 1



(b) Fault's dimension adjustment after robot number 1 reestablished communication



(c) Robot number 2 entered the fault and its dimension was adjusted



(d) Robot number 2 exited the fault and stopped at rest station number 5

Figure 5.3: Mapping of the fault of Test G: red nodes represent "faulted" nodes, yellow nodes represent entry/exit nodes, purple nodes represent frontier nodes, green nodes represent "unfaulted" nodes and blue nodes represent not mapped nodes

During the execution of Test G[1], the induced fault was mapped initially by robot number 1 and, after robot number 1 exited the fault's area, by robot number 2.

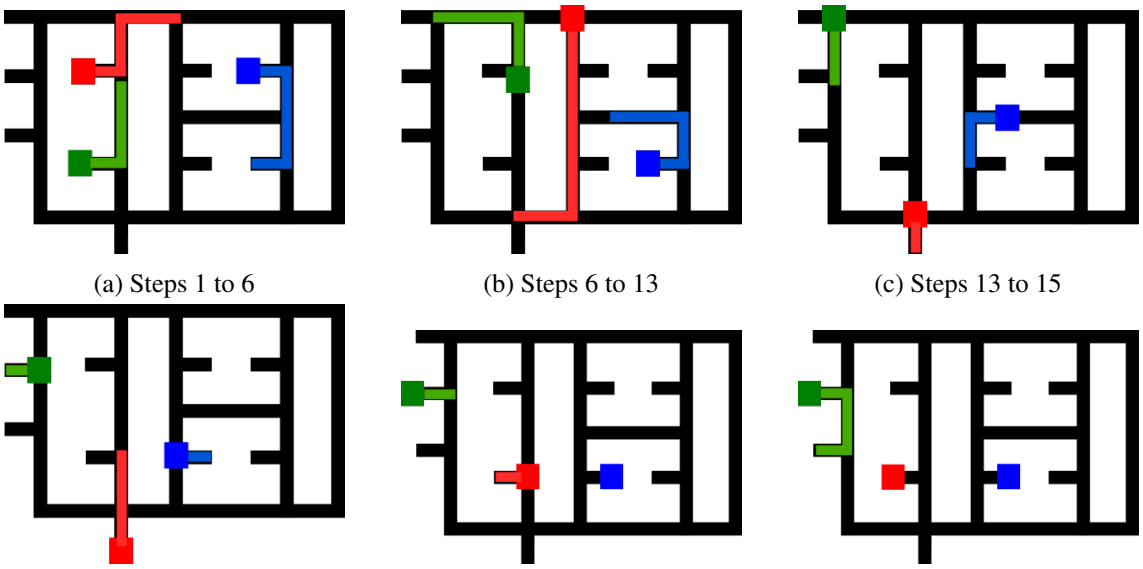
Robot number 2 completed its mission inside the fault and became stuck inside of it. As expected, the Recovery Mode was activated (1:40) and robot number 2 followed a new trajectory towards the nearest rest station. The path towards station number 5 was also included on the fault's area. However, since robot number 1 had already mapped those nodes, no changes were made when it comes to the fault's estimated area.

The schematic of the process of mapping the nodes and the fault is represented in Figure 5.3.

The path initially calculated by the Path Planning Sub-Module is represented in Figure 5.4, where robot number 1 is coloured in red, robot number 2 in blue and robot number 3 in green. However, as it was already explained, the final trajectories performed were not the same trajectories due to the Recovery Mode (there were also some extra waiting steps due to the Planning Supervisor re-calculations).

The performed paths can be consulted through the link on footnote.

[1] https://youtu.be/rCv7Aauoxk0



(a) Steps 1 to 6 (b) Steps 6 to 13 (c) Steps 13 to 15

(d) Steps 15 to 17 (e) Steps 17 to 19 (f) Steps 19 to 23

Figure 5.4: Initial path schematic calculated by the Path Planning Module (rotations are not represented)

If the same mission is repeated without the communication fault, it is possible to compare the execution time and estimate the time added by the Recovery Mode, as well as other parameters.

Therefore, Test G was repeated without the induced fault. In this way, it is also possible to observe the impact of the fault on the re-calculations of the paths.

The results are presented in Table 5.4 and Table 5.5.

Table 5.4: Execution of Test G': Test G without the induced fault

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	6	2:00	0	100%
2	2	1:37	0	100%
3	7	1:40	0	100%
4	7	1:55	0	100%
5	4	2:01	0	100%

Table 5.5: Average values of Test G': Test G without the induced fault

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	5.2	1:50	0	100%

5.2.2 Test H

The previous mission represented simple tasks, since only one of the robots visited more than one work station. In this test, more complex missions were tested. The tasks assigned and their location are described in Table 5.6 and in Figure 5.5.

Table 5.6: Assigned tasks for Test H

Robot ID	Robot Station	Orientation	Workstation	Rest Station
1	No 1	180 °	No 8 & 7 & 3	No 4
2	No 4	180 °	No 3 & 5& 8	No 1
3	No 6	-90 °	No 7 & 5	No 2

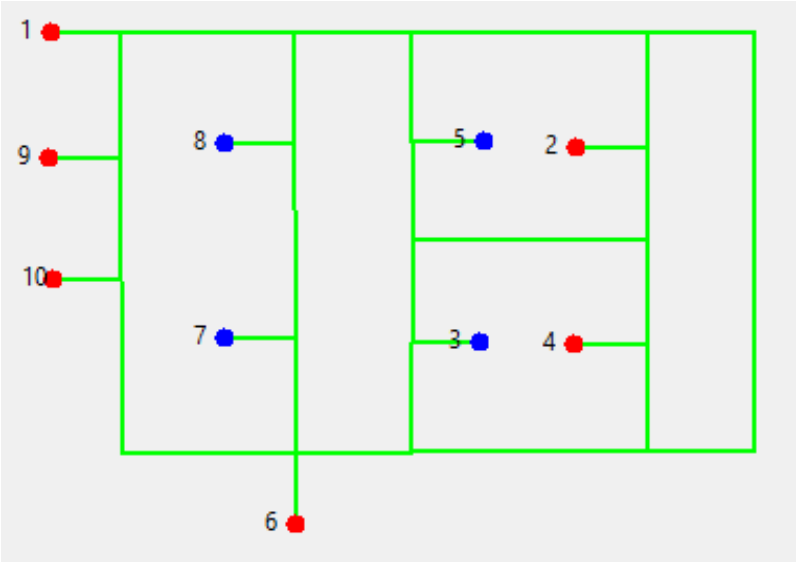


Figure 5.5: Shop floor skeleton graph with the workstations selected for Test H

To observe which nodes will be affected by the fault, the graph of the map and the location of the induced fault are represented in Figure 5.6.

It is expected for robot number 3 to change its rest station from number 2 (node 12 in Figure 5.6) to rest station number 1 (node 1 in Figure 5.6). Observing Figure 5.6, it is possible to confirm that rest station number 2 is inside the communication fault's area. To assess the nearest rest station is necessary to exclude the stations that are workstations (stations number 3 and 5, as shown in and Figure 5.5). The nearest rest station would be station number 4. However, robot number 1 had that station attributed in the beginning of the mission. So, considering the available rest stations, the closest one would be station number 1.

The test was performed five times and the results are described in Table 5.7 and Table 5.8.

Table 5.7: Execution of Test H

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	14	4:14	0	100%
2	22	4:01	0	100%
3	19	3:59	0	100%
4	15	4:04	0	100%
5	15	4:07	0	100%

Table 5.8: Average values of Test H

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	17	4:05	0	100%

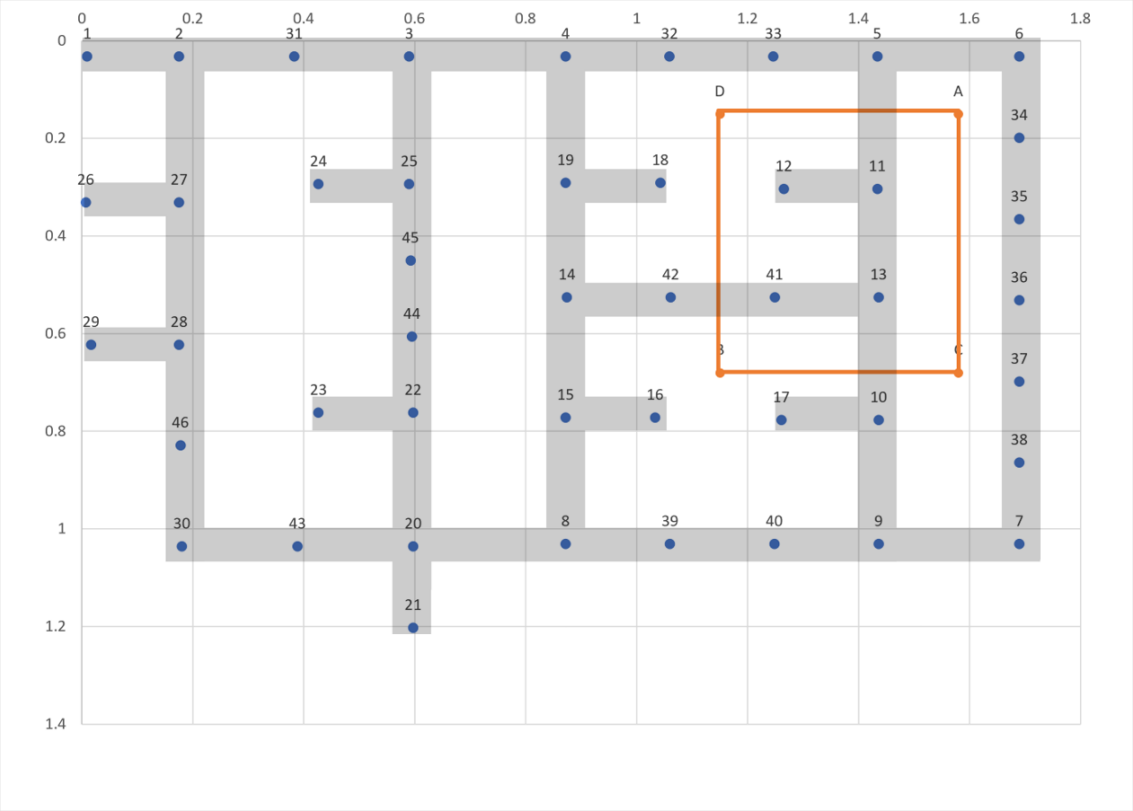


Figure 5.6: Induced fault (borders at orange) for Test H

During the execution of Test H[2], it was possible to observe that this mission required a lot of coordination to avoid deadlocks and collisions. The area where the nodes 4, 19, 14, 15 and 8 belong was accessed multiple times by the three robots, which required a lot of traffic control.

As expected, when the Recovery Mode was activated (3:09), robot number 3 moved to rest station number 1.

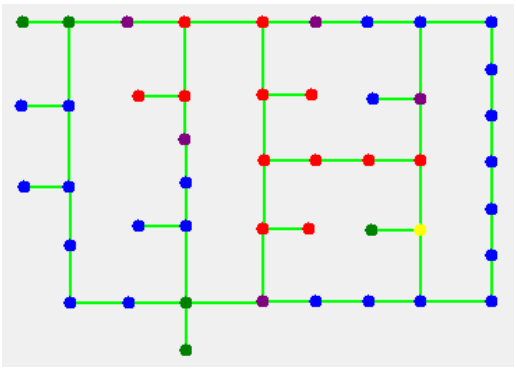
As it is possible to observe, from the results in Table 5.7 and Table 5.8, no collisions or deadlocks happened and the tasks where all completed. However, the multiple access to that area may explain the supervisor intervention.

The schematic of the process of mapping the nodes and the fault is represented in Figure 5.7.

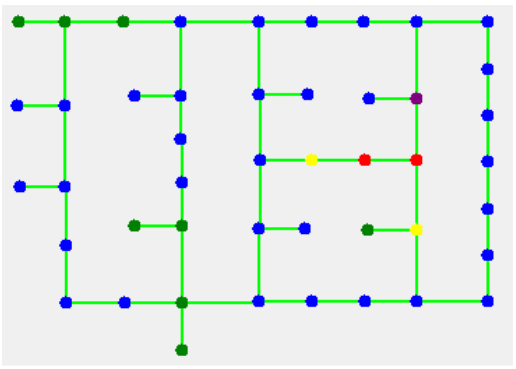
As it was possible to predict, the fault was, firstly, partially mapped by robot number 2 and then mapped by robot number 3 as it went to complete its mission in its rest station.

[2] https://youtu.be/huASuabvJHM

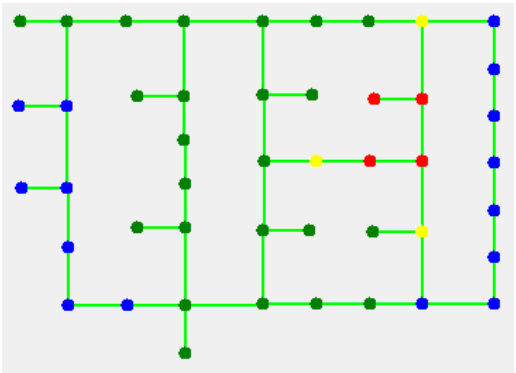
Once again, because the map was almost completely mapped, it is not possible to observe the addition of the area of the trajectory from station 2 to station 1 because all nodes around the fault were already mapped as "unfaulted".



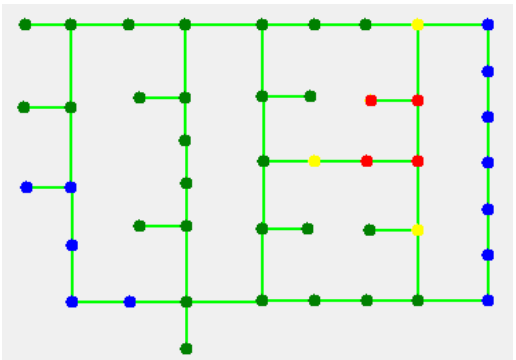
(a) Robot number 2 enters the fault and it is detected, located and estimated



(b) Fault's dimension is adjusted after robot number 2 reestablished communication



(c) Robot number 3 entered the fault and its dimension was extend until its rest station



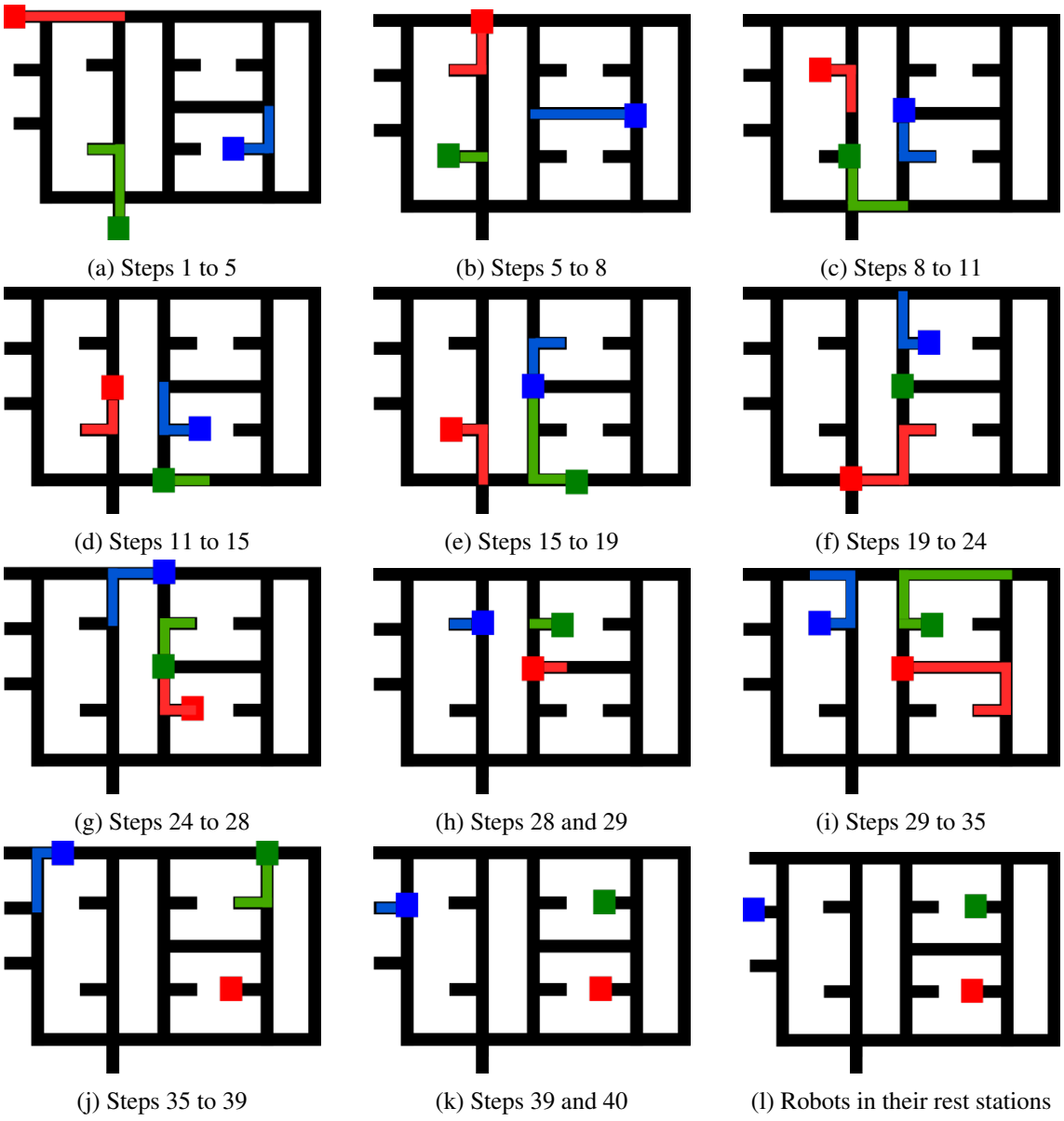
(d) Robot number 3 exited the fault after changing to rest station 1 (Recovery Mode)

Figure 5.7: Mapping of the fault of Test H: red nodes represent "faulted" nodes, yellow nodes represent entry/exit nodes, purple nodes represent frontier nodes, green nodes represent "unfaulted" nodes and blue nodes represent not mapped nodes

The path initially calculated by the Path Planning Sub-Module is represented in Figure 5.8. The actual paths completed can be observed through the link on footnote.

The same mission was repeated without the communication fault, to analyse, approximately, the time added by the Recovery Mode to the execution time of the mission and to observe the impact of the fault on the re-calculations of the paths.

The results are presented in Table 5.9 and Table 5.10.



(a) Steps 1 to 5 (b) Steps 5 to 8 (c) Steps 8 to 11

(d) Steps 11 to 15 (e) Steps 15 to 19 (f) Steps 19 to 24

(g) Steps 24 to 28 (h) Steps 28 and 29 (i) Steps 29 to 35

(j) Steps 35 to 39 (k) Steps 39 and 40 (l) Robots in their rest stations

Figure 5.8: Initial path schematic calculated by the Path Planning Module (rotations are not represented)

Table 5.9: Execution of Test H': Test H without the induced fault

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
1	7	3:00	0	100%
2	15	2:59	0	100%
3	12	3:35	0	100%
4	12	2:58	0	100%
5	8	3:09	0	100%

Table 5.10: Average values of Test H': Test H without the induced fault

#	Supervisor Intervention	Exec. Time (min.)	Deadlocks & Collisions	Tasks Completed
5	10.8	3:08	0	100%

5.2.3 Results' Discussion

Through the implementation of Test G and Test H, it was aimed to observe the response of the system to a critical situation such as having a robot completing their mission inside a communication fault and being stuck inside of it.

The Communication Supervisor Sub-Module guaranteed the prevention of this situation by activating a Recovery Mode, as explained previously.

The first test executed, Test G, assigned simple missions to the robots. It was assessed the rest stations chosen and an induced fault was triggered in order to cover one of the rest stations.

Analysing the resulted data, the supervisor had to re-calculate the paths an average of 9.4 times. As it was already mentioned, since two robots mapped the fault, 4 of those re-calculations were due to the fault. The average execution time was of 2:16.

To be able to compare those results and analyse what was triggered by the fault, the same mission was repeated without the fault. The results in Table 5.4 and Table 5.5, describe an average supervisor intervention of 5.2 times and an execution time of 1:50.

Therefore, it is possible to confirm that, due to the difference between the average re-calculations of both experiments, the fault truly imposed a re-calculation of 4 times.

The difference in the execution time is justified by the activation of the Recovery Mode and the addition of the sub-mission to shift rest stations. Even though the Recovery Mode was activated before robot number 3 finished its mission, robot number 2 only reached its new rest station a few seconds, approximately 26, later.

To analyse the Recovery Mode, the coordination of the fleet and the influence of the fault, in more detail, another mission was tested. Test H aimed to assign more complex tasks. A careful coordination of the fleet was expected and needed to avoid deadlocks and collisions.

An induced fault was also triggered, affecting the rest station of robot 3. It was expected the Recovery Mode to change its rest station, once it was stuck inside the fault, to rest station number 1.

During Test H performance, when robot number 3 was inside the communication fault, in its rest station, robot number 1 would not go inside its rest station because it would have to cross an entry/exit of the fault. That node can be observe in Figure 5.7, in the colour yellow, at the entrance of station number 4 (node 17 in Figure 5.6). Since a robot was inside the fault, the area of the fault was blocked and robot number 1 had to wait (on the video it is possible to observe the robot moving from node 9 to node 40 because node 40 was the only available node connected to node 9) until robot number 3 exited it to cross node number 10 (Figure 5.6).

Because of the Recovery Mode, this situation did not reveal to be a problem. However, alternative procedures could be considered to avoid this situation such as a temporary window to block the entry/exit nodes (accordingly to the expected exit area of the lost robot).

As studied in Sub-Section 4.2.3, the dynamic cost incorporated on the Path Planning algorithm, TEA* algorithm, intended to avoid optimal paths that crossed faulty nodes.

That implementation was possible to observe in the execution of Test H, after robot number 3 completed its task in workstation number 5, it moved towards its initial rest station, station number 2, through the "unfaulted" nodes (2:04).

In some situations, this could have not happened because there were only two nodes marked as "faulted" and two entry/exit nodes with a probability of being faulty superior to 0 but inferior to 1. These nodes and their cost could have not been sufficient to balance a longer alternative path.

Comparing the results of Test H with the ones of Test H', the difference between the average execution time was of 1:13. In Test H, the supervisor also had to re-calculate the paths 6.2 times more.

The difference in the number of re-calculations (6.2 times) is superior to the predicted (4 times). However, that difference does not have a significant meaning.

When it comes to the time difference, 1:13 reveals a significant addition on the duration of the mission. However, comparing this result with the obtained in Test G (26 seconds), in Test G not all robots had completed their mission before the Recovery Mode was activated, existing an overlap between the execution time of the mission and the execution time of the Recovery Mode. In Test H, all robots had finished their missions, being the resulted time difference, between Test H and Test H', exclusively from the Recovery Mode.

It is also important to mention that the new rest station chosen in Test H was located further away from the current rest station, due to the unavailability of the closest rest stations.

Therefore, the distance difference, between the current rest station and the new rest station, and the execution of the Recovery Mode not overlapping the execution of the assigned mission, resulted in the longer execution time of the Recovery Mode in Test H.

5.3 Conclusion

After the implemented system and its most important features have being described in previous chapters, Chapter 3 and Chapter 4, in this chapter a specific situation was evaluated, resorting all the features previously implemented and explained.

Therefore, throughout this chapter, it was presented the approach of the Communication Supervisor Sub-Module to a critical situation.

The situation studied concerned the completion of a robot's mission inside a communication fault, resulting in that lost robot being stuck inside the fault until, eventually, communication could be established.

To solve this situation, a solution denominated Recovery Mode was proposed. This feature of the Communication Supervisor Sub-Module would be activated after the Central Control Module determined that all steps of the lost robot had, probably, been completed. This assessment is done through the estimation of the time taken to complete the number of missing steps, to complete the mission, at the time that the connection with the robot was lost.

It is expected that the lost robot, after completing its mission inside a fault, realises that it is inside a fault and removes itself from the fault to the nearest rest station. When the Recovery Mode is activated, it aims to predict the exit area and the lost robot's new path (to a new rest station) and coordinate the rest of the fleet to avoid collisions and deadlocks.

This approach was implemented and tested through two particular tests: Test G and Test H.

Both tests intended to not only evaluate the execution of the Recovery Mode and its effectiveness in solving the described problem, but also to analyse all the features implemented previously.

Therefore, both tests were also performed without contemplating the induced fault. The results were analysed and discussed.

The Recovery Mode revealed to have an effectiveness of 100%, taking into account the tests executed.

Chapter 6

Conclusions and Future Work

In this chapter, a brief summary of the objectives is presented as well as the implemented approaches and obtained results. Based on the obtained results, modifications and additional features to be developed in future work are also proposed.

6.1 Accomplishment of the Described Objectives

The main objective of this dissertation was the implementation of a system capable of planning and controlling, safely and effectively, a fleet of AGV. To reach this objective, it was needed a multi-AGV supervisory system capable of being communication failure tolerant, since communication failures represented a big threat.

By continuing the work previously developed by [9], it was possible to test the developed approaches in a real environment, since it was only tested in a simulation platform, and adapt them to the real world's dynamics. Further developments were also made in order to increase the safety of the planned paths.

To implement the work previously developed, some modules were added and small scale AGV were developed.

The overall system comprised different modules. The Central Control Module comprised the Path Planning Sub-Module and the Supervisor Sub-Module. The implemented system had to rely on the TEA* algorithm (in the Path Planning Sub-Module) to plan efficient paths for the robots to accomplish the designated missions and be implemented in Free Pascal language in the development environment Lazarus. Once the paths were calculated, any alteration needed would be triggered by the Supervisor Sub-Module.

The supervisor implemented searched for communication faults, delays in the execution of the missions and deviations in the routes, as initially proposed. In this way, the execution of the paths was monitored to maintain the temporal efficiency granted by the TEA* and to incorporate strategies to safely deal with communication faults.

The Planning Supervisor Sub-Module, initially developed by [9], was tested in several situations in order to explore what triggered the re-calculation of the paths and which factors could be adjusted to adapt the supervisor to the dynamic of a real environment.

As already mentioned, the main goal of the Communication Supervision Sub-Module was to detect communication faults and guarantee secure and efficient paths when stable communication could not be established. To operate accordingly, several situations were tested to observe how the communication faults were located and their dimension estimated.

An approach was proposed to try to prevent the robots from crossing mapped areas with communication faults. That approach consisted of a dynamic cost on the TEA* to allow the algorithm to calculate safe routes, avoiding the locations of communication faults. To avoid those locations, it was incorporated, in the TEA* algorithm, a reliability factor depending on the node. An extra cost was added to certain nodes, depending on their probability to suffer communication faults.

Through the study of the communication faults' detection and their impact on the robots' trajectories, a new feature was added to the Communication Supervision Sub-Module to prevent robots to be stuck inside communication faults after completing their mission. That feature, Recovery Mode, was implemented and tested and revealed to be effective when that problem happened.

Other critical situations described by [9] were also fixed. The synchronisation of two or more robots when inside the same fault, at the same time, was guaranteed through the obligation of executing one step within the time defined. This procedure was applied in the Robot Control Module and was only followed when communication with the Central Control Module was off.

Although the system exhibits good tolerance to communication faults, further testing is still needed to discover critical situations and implement additional features.

Some of the results present in this dissertation have been included in a paper accepted in the *"OL2A: International Conference on Optimization, Learning Algorithms and Applications 2021"* conference.

Another paper is already being developed with the obtained results from the validation and testing of the present work.

6.2 Future Work

During the development of this project, some critical situations, usually caused by communication faults, were evaluated and approaches were proposed to help the system tolerant those. However, further testing is needed.

Firstly, it was noticed that the different size of the links affected significantly the synchronisation of the robots. Different link sizes would not allow some of the triggered conditions of the supervisor to be fair criteria. Therefore, the conception of another map, with strategically located crossroads, would be a factor to test and evaluate how it would impact the re-calculations of the paths.

Also, other critical situations, such as having a robot starting its mission inside a communication fault, have not been considered yet.

It would also be beneficial to test the already implemented features in an already mapped graph. For example, the situation of having the system changing an assigned rest station, that was found to be inside a fault, to another was not tested yet. A feature to solve this problem was already implemented but it was not possible to test it because rest stations are not mapped unless a robot completes its mission.

During the test and validation of the Recovery Mode, it was observed a situation where a robot could not move inside its rest station because it needed to cross an entry/exit node. Since a robot was inside that fault, all area concerning the fault was locked. To prevent situations like that, instead of applying a binary semaphore to control the access to faults, a semaphore based on a time window should be tested. This approach would increase the effectiveness of the system because it would allow the entry/exit nodes to be unlocked when the probability of the lost robot being near that area would be insignificant.

References

[1] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, October 1979. URL: https://doi.org/10.1145/359156.359164, doi:10.1145/359156.359164.

[2] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991. URL: https://doi.org/10.1145/116873.116880, doi:10.1145/116873.116880.

[3] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. doi:10.1017/CBO9780511546877.

[4] Pedro L. C. Gomes da Costa. *Planeamento Cooperativo de Tarefas e Trajectórias em Múltiplos Robôs*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2011.

[5] Joo Young Hwang, Jun Song Kim, Sang Seok Lim, and Kyu Ho Park. A fast path planning by path graph optimization. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 33(1):121–129, 2003. doi:10.1109/TSMCA.2003.812599.

[6] UNSW Computing. Methods of Search. URL: http://www.cse.unsw.edu.au/~billw/Justsearch.html [last accessed 2021-01-29].

[7] Joana Santos, Pedro Costa, Luís Rocha, Kelen Vivaldini, A. Paulo Moreira, and Germano Veiga. Validation of a time based routing algorithm using a realistic automatic warehouse scenario. In Luís Paulo Reis, António Paulo Moreira, Pedro U. Lima, Luis Montano, and Victor Muñoz-Martinez, editors, *Robot 2015: Second Iberian Robotics Conference*, pages 81–92, Cham, 2016. Springer International Publishing.

[8] Joseph J.M. Evers and Stijn A.J. Koppers. Automated guided vehicle traffic control at a container terminal. *Transportation Research Part A: Policy and Practice*, 30(1):21 – 34, 1996. URL: http://www.sciencedirect.com/science/article/pii/0965856495000119, doi:https://doi.org/10.1016/0965-8564(95)00011-9.

[9] Diogo M. R. Matos. *Multi AGV Coordination Tolerant to Communication Failures*. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2020.

[10] G. Ullrich. *Automated Guided Vehicle Systems*. Springer, Berlin, Heidelberg, Second edition, 2015. doi:10.1007/978-3-662-44814-4.

[11] Grand View Research. Automated Guided Vehicle Market Size, Share & Trends Analysis Report By Vehicle Type, By Navigation Technology, By Application, By End-use

Industry, By Component, By Battery Type, By Region, And Segment Forecasts, February 2020. URL: https://www.grandviewresearch.com/industry-analysis/automated-guided-vehicle-agv-market [last accessed 2021-01-21].

[12] Chris Benevides. The Advantages and Disadvantages of Automated Guided Vehicles (AGVs), October 2020. Updated on January 5th, 2021. URL: https://www.conveyco.com/advantages-disadvantages-automated-guided-vehicles-agvs/ [last accessed 2021-01-21].

[13] Andre Clayton. 4 Benefits of Industrial AGVs in Manufacturing, December 2018. URL: https://blog.pepperl-fuchs.us/4-benefits-of-industrial-agvs-in-manufacturing [last accessed 2021-01-21].

[14] Alex S. Fukunaga Y. Uny Cao and Andrew Kahng. Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, 4:7–24, 1997. doi:10.1023/A:1008855018923.

[15] M. Bader, A. Richtsfeld, W. Holl, M. Suchi, G. Todoran, and M. Vincze. Balancing Centralised Control with Vehicle Autonomy in AGV Systems for Industrial Acceptance. *Proceedings 11th International Conference on Autonomic and Autonomous Systems (ICAS)*, May 2015.

[16] C. Liu, J. Tan, H. Zhao, Y. Li, and X. Bai. Path planning and intelligent scheduling of multi-agv systems in workshop. In *2017 36th Chinese Control Conference (CCC)*, pages 2735–2739, 2017. doi:10.23919/ChiCC.2017.8027778.

[17] I. Draganjac, D. Miklić, Z. Kovačić, G. Vasiljević, and S. Bogdan. Decentralized control of multi-agv systems in autonomous warehousing applications. *IEEE Transactions on Automation Science and Engineering*, 13(4):1433–1447, 2016. doi:10.1109/TASE.2016.2603781.

[18] Ying-Chin Ho and Ta-Wei Liao. Zone design and control for vehicle collision prevention and load balancing in a zone control agv system. *Computers & Industrial Engineering*, 56(1):417 – 432, 2009. URL: http://www.sciencedirect.com/science/article/pii/S036083520800137X, doi:https://doi.org/10.1016/j.cie.2008.07.007.

[19] N. Smolic-Rocak, S. Bogdan, Z. Kovacic, and T. Petrovic. Time windows based dynamic routing in multi-agv systems. *IEEE Transactions on Automation Science and Engineering*, 7(1):151–155, 2010. doi:10.1109/TASE.2009.2016350.

[20] Raffaello D'Andrea Peter Wurman and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29:9–20, 03 2008.

[21] D. Herrero-Perez and H. Martinez-Barbera. Decentralized coordination of automated guided vehicles. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '08, page 1195–1198, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[22] M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1419–1424, 1986. doi:10.1109/ROBOT.1986.1087401.

[23] P. Bhattacharya and M. L. Gavrilova. Roadmap-based path planning - using the voronoi diagram for a clearance-based shortest path. *IEEE Robotics Automation Magazine*, 15(2):58–66, 2008. doi:10.1109/MRA.2008.921540.

[24] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

[25] Anthony Stentz. The focussed d* algorithm for real-time replanning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1652–1659, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[26] Xiaoxun Sun, William Yeoh, and Sven Koenig. Moving target d* lite*. volume 1, pages 67–74, 01 2010. doi:10.1145/1838206.1838216.

[27] Theodore Manikas, K. Ashenayi, and R.L. Wainwright. Genetic algorithms for autonomous robot navigation. *Instrumentation & Measurement Magazine, IEEE*, 10:26 – 31, 01 2008. doi:10.1109/MIM.2007.4428579.

[28] Miguel Porta-Garcia, Oscar Montiel Ross, Oscar Castillo, Roberto Sepúlveda, and Patricia Melin. Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. *Appl. Soft Comput.*, Vol. 9:1102–1110, 06 2009.

[29] Robert Bohlin and Lydia Kavraki. Path planning using lazy prm. volume 1, pages 521 – 528 vol.1, 02 2000. doi:10.1109/ROBOT.2000.844107.

[30] Yang Gao, Shu-dong Sun, Da-wei Hu, and Lai-jun Wang. An online path planning approach of mobile robot based on particle filter. *Industrial Robot: An International Journal*, 40, 06 2013. doi:10.1108/01439911311320813.

[31] Saurabh Sarkar and Ernest Hall. Virtual force field based obstacle avoidance and agent based intelligent mobile robot. volume 6764, 09 2007. doi:10.1117/12.734691.

[32] Andrej Babinec, František Duchoň, Martin Dekan, Zuzana Mikulová, and Ladislav Jurišica. Vector field histogram* with look-ahead tree extension dependent on time variable environment. *Transactions of the Institute of Measurement and Control*, 40(4):1250–1264, 2018. doi:10.1177/0142331216678062.

[33] Oussama Khatib. *The Potential Field Approach And Operational Space Formulation In Robot Control*, pages 367–377. Springer US, Boston, MA, 1986. URL: https://doi.org/10.1007/978-1-4757-1895-9_26, doi:10.1007/978-1-4757-1895-9_26.

[34] Meng Wang and James N.K. Liu. Fuzzy logic-based real-time robot navigation in unknown environment with dead ends. *Robotics and Autonomous Systems*, 56(7):625–643, 2008. URL: https://www.sciencedirect.com/science/article/pii/S0921889007001467, doi:https://doi.org/10.1016/j.robot.2007.10.002.

[35] Dieter Fox and Wolfram Burgard. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4:23 – 33, 04 1997. doi:10.1109/100.580977.

[36] M. Tarokh. Hybrid intelligent path planning for articulated rovers in rough terrain. *Fuzzy Sets and Systems*, 159:2927–2937, 11 2008. doi:10.1016/j.fss.2008.01.029.

[37] L. Lulu and A. Elnagar. A comparative study between visibility-based roadmap path planning algorithms. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3263–3268, 2005. doi:10.1109/IROS.2005.1545545.

[38] Han-Pang Huang and Shu-Yun Chung. Dynamic visibility graph for path planning. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2813–2818 vol.3, 2004. doi:10.1109/IROS.2004.1389835.

[39] Jean-Claude Latombe. *Exact Cell Decomposition*, pages 200–247. Springer US, Boston, MA, 1991. URL: https://doi.org/10.1007/978-1-4615-4022-9_5, doi:10.1007/978-1-4615-4022-9_5.

[40] Jason Jason Lu and Minlu Zhang. *Heuristic Search*, pages 885–886. Springer New York, New York, NY, 2013. URL: https://doi.org/10.1007/978-1-4419-9863-7_875, doi:10.1007/978-1-4419-9863-7_875.

[41] UMSL University of Missouri–St. Louis. Greedy Algorithms. URL: http://umsl.edu/~adhikarib/cs4130-fall2017/slides/03%20-%20Greedy%20Algorithms.pdf [last accessed 2021-01-29].

[42] J. L. Bander and C. C. White. A heuristic search algorithm for path determination with learning. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 28(1):131–134, 1998. doi:10.1109/3468.650331.

[43] J. Santos, P. Costa, L. F. Rocha, A. P. Moreira, and G. Veiga. Time Enhanced A*: Towards the Development of a New Approach for Multi-Robot Coordination. In *2015 IEEE International Conference on Industrial Technology (ICIT)*, pages 3314–3319, 2015. doi:10.1109/ICIT.2015.7125589.

[44] Anthony Stentz. The focussed d* algorithm for real-time replanning. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.

[45] Sven Koenig and Maxim Likhachev. Incremental a*. In *Proceedings of (NeurIPS) Neural Information Processing Systems*, pages 1539 – 1546, December 2001.

[46] Sven Koenig and Maxim Likhachev. D*lite. pages 476–483, 01 2002.

[47] An T. Le, Minh Q. Bui, Than D. Le, and Nauth Peter. D* lite with reset: Improved version of d* lite for complex environment. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 160–163, 2017. doi:10.1109/IRC.2017.52.

[48] Qin Li, Alexander Pogromsky, Teun Adriaansen, and Jan Udding. A control of collision and deadlock avoidance for automated guided vehicles with a fault-tolerance capability. *International Journal of Advanced Robotic Systems*, 13:1, 04 2016. doi:10.5772/62685.

[49] E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, 3(2):67–78, June 1971. URL: https://doi.org/10.1145/356586.356588, doi:10.1145/356586.356588.

[50] Oliver Bittel and Michael Blaich. Mobile robot localization using beacons and the kalman filter technique for the eurobot competition. In David Obdržálek and Achim Gottscheber, editors, *Research and Education in Robotics - EUROBOT 2011*, pages 55–67, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[51] Sergio Garrido-Jurado, Rafael Munoz-Salinas, Francisco Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51, 10 2015. doi:10.1016/j.patcog.2015.09.023.

[52] Francisco Romero-Ramirez, Rafael Munoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 06 2018. doi:10.1016/j.imavis.2018.05.004.

[53] Y. M. Wang, Y. Li, and J. B. Zheng. A camera calibration technique based on opencv. In *The 3rd International Conference on Information Sciences and Interaction Sciences*, pages 403–406, 2010. doi:10.1109/ICICIS.2010.5534797.