Path Planning and Optimization of Autonomous Mobile Robots in a Logistics Warehouse Scenario

Gabriel Henrique Silva Cangussu - a52574

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master's Degree in Mechanical Engineering.

Work oriented by:

Professor José Luís Sousa de Magalhães Lima

Professor Ana Isabel Pinheiro Nunes Pereira

Bragança

2024

Acknowledgement and Dedication

I want to thank God first of all for allowing me to have the opportunity to do a school exchange, being in Portugal, and being able to study at the Polytechnic Institute of Bragança; I am from a small town in the interior of Goiás state, Brazil.

I would also like to thank my parents, Jusley and Giselle, who never spared any effort to help me or help with anything that was needed. Even though our family didn't have the best conditions, They always helped me and cared about me financially, physically, and academically. Thanks to my brother Marcos, who always saw me arrive home at dawn in Brazil, tired from working during the day, traveling every day about 75 kilometers to Anápolis, where the college is located, and returning every night, often tired or angry.

I would also like to thank my supervisors, Prof. Dr. José Luís Sousa de Magalhães Lima and Prof. Dr. Ana Isabel Pinheiro Nunes Pereira, who accompanied and guided me all the way from the elaboration and idealization of the project to its final phase. Also, I have a special thanks to researcher João Braun, one of those responsible for the project I worked on, who always helped me with everything I needed, helped with the changes resulting from the project, the development of the algorithm, gave feedback on changes and improvements and has always been helping me through every step.

I thank the professors of both universities, Evangelical University of Goiás (Unievangélica) and Polytechnic Institute of Bragança (IPB), for all the classes and content taught that enabled me to defend my thesis.

And finally, I thank everyone who directly or indirectly supported or helped me with everything that was necessary during my undergraduate degree and now during my master's degree.

Abstract

The use of robots in manufacturing and storage environments is increasing every day. Their benefits within a company's logistics and their speed in performing tasks make them more productive in terms of carrying out tasks directed towards a common goal. With this, the main objective of this work is to optimize the path planning of autonomous mobile robots (AMRs) for later use in various applications, focusing on logistics warehouse scenarios. The work involves the analysis and improvement of algorithms (and their variants) for control and movement of AMRs, focusing on reducing the execution time of the algorithm itself. Variants of the A* algorithm were implemented and used in path planning problems, aiming to find the shortest path for robots, even in environments with obstacles.

The adopted methodology includes a literature review of algorithm models for use in the context of this work, as well as the analysis of algorithm variants found for adaptation and use in the proposed scenario. After analyzing them, the obtained results were checked and compared for two different map models.

The performance of the algorithm depends on the functions programmed into it and the steps it must go through to get from the start point to the end point. The greater the number of functions in the algorithm, the longer its execution time will be. The generation of graphs to visualize the algorithm's result requires most of this execution time, as it uses visual resources to show the user the proper functioning of the tested algorithm.

The analyses and tests conducted revealed that, although both algorithms have quick response times suitable for use in scenarios with entry warehouses, exit warehouses, and

obstacles along the route, the Sakai A* algorithm variant proved to be more efficient due to its simple structure, fewer steps within the algorithm, and agile response time in generating optimal solutions. This had a shorter execution time compared to the Mehdi A* algorithm variant, which, although good, proved to be slower in obtaining the optimal path.

Key-words: A*; AMRs; Path planning.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The use of Autonomous Mobile Robots (AMRs) in logistics warehouses is increasing every day. The possibilities they offer to transport materials and perform functions such as transportation, storage, item relocation, inventory organization, and others, quickly, make their choice accelerate the performance and production of storage and warehousing sectors.

However, the use of these robots requires them to be well programmed for the context in which they are placed, with specific functions to move and perform actions related to the objective of their work (picking up boxes, pushing objects, or even lifting materials).

With that, it is necessary for the robot to have a well-structured operating algorithm, with the dimensions of the location where it will operate, the obstacles it cannot collide with, the actions it will perform, and the planning of the path it must take to complete the journey from the starting point to the endpoint. These stages will be analyzed and identified in this work.

1.1 Objectives

The main objectives of this work are:

- Verify the historical context of the use of robots in manufacturing or storage environments;
- Create maps where robots can be inserted, checking their response to the presence of obstacles along the route;
- Explore algorithms and their variants that enable path planning for an AMR robot so that it can follow a predetermined route from a start point to an end point in an optimal manner, in the shortest possible time.
- Test the analyzed algorithms and verify their responses on the created maps, for various points;
- Analyze the results of the tests of the algorithms and their variants, highlighting their strengths and weaknesses;
- Compare the results obtained between the variants, identifying possible problems or failures in their use;
- Choose the best algorithm among those analyzed for this work.

1.2 Document Structure

In sequence, this report contains chapters 2 to 5. Chapter 2 consists of a conceptual framework that presents the historical data on trajectory planning in the robotic environment and the programming languages used in this work. Chapter 3 is dedicated to the methodology used in the algorithm, as well as its computational language. It also shows the approach chosen during the analysis of the work and the variants of the path planning algorithm used, as well as their analyses. Next, in Chapter 4, the results and

discussions conducted during the implementation of the algorithm are presented, as well as the complete comparison between the analyses of the two variants of the algorithm. Finally, in Chapter 5, the conclusions observed at the end of the work, the choice of the best variant, the next steps and the future paths to be followed are described.

Chapter 2

Conceptual Framework

2.1 Robotics in the manufacturing environment

The implementation of automation systems began in the last century. One of the first signs of automation in the automotive industry is from Henry Ford [1], who used assembly lines to build cars and their components. The idea that each person would perform only a specific number of tasks in a controlled and schematized environment, following a single order of steps, revolutionized the industrial sector at the time.

Over the decades, the number of machines in industries, whether autonomous or not, has increased. With the third industrial revolution, the number of robots used could bring benefits such as cost reduction, improvements in the quality of services performed, reduction in manufacturing time, and the possibility of being used in exhaustive work or with excessive load [2].

Automated Guided Vehicle (AGV) technology is becoming entirely inserted in this sense, and with this, the use of robots is more flexible. Among the AGV technologies implemented in the market, Autonomous Mobile Robots (AMRs) are considered leaders and the highest level of automation [3].

In general, AMRs are considered devices or equipment that continue to perform specific predefined tasks without the need for human intervention. This makes the unitary

transport of materials and devices optimized. Wicher et al. [3] classify AMRs as a subgroup of AGVs, and other types are also included, such as Autonomous in Movement Vehicles (AMV) and Mobile Autonomous Units (MAU), all of which are just divisions of the same system. Alatise et al. [4] classifies AMRs as mobile robots with the freedom and ability to move almost completely while performing a specific task. Because it is an area of vast and comprehensive study, it cannot be attributed to only one definition. Since the first AGV was introduced in 1955 [5], these have undergone changes, adaptations, and improvements for each case where they are needed. In most cases, they need optical systems or similar to those used to analyze the environment in which it is inserted and detect the presence of objects and control points, which are used to mark the location and route where it should pass and stop.

Fragapane et al. [6] quickly explain that the main difference between AGV and the AMR is in the mode these move. While AGVs can only move along pre-established points and paths, AMRs can, within a specific mined area to, move without running the risk of collision. From this perspective, AGVs are unable, if necessary, to change their operating mode, route, dimensions of the transported object, or any data they need.

AMRs have greater flexibility in this regard, as they can make certain decisions autonomously according to the programme inserted into them, so that you can monitor them yourself to analyse problems, failures or necessary modifications. This possibility of operating without the almost total presence of a human being is still a challenge when it comes to implementing this trajectory planning model, as it is necessary to create specific algorithms and programmes for each environment in which the robot is deployed.

In view of this, due to growing technological and industrial advances and the tests carried out every day, there is a need for systems that have degrees of freedom for autonomous movement and processing according to programming and need, thus reducing the response time to current needs in the manufacturing environment. Using systems that analyse their own context, they look for alternative routes and ways to solve existing problems on the production line, providing a way of optimising the shop floor, reducing the time needed to discover errors, without having to send an employee to their own

station to discover the problem, so that fault corrections can be made in real time.

2.2 Path planning

When researching historically, authors such as Baker [7], Fox [8], Sadeh [9] and Chase and Aquilano [10], summarise the meaning of path planning as the good use of available resources within the timeframe because this planning can be defined, roughly speaking, as a set of decisions to be made, taking into account possible changes in all aspects and prior preparation for these events, without affecting the mandatory requirements for each case.

The management of time, space, and resources is essential for any organization. Optimization and control models have been created over the years in an attempt to correctly manage all functions within a factory without losses, such as Charles Babbage (18th century) and Taylor (19th century) [11], some of the pioneers in using path planning models in the field of economics, employing techniques to analyze industrial processes.

With the advent of World War II, concerns about production planning and using available resources initiated what was later called the science of Operational Research [11]. Extremely complex calculations were required, considering that several possibilities should be analyzed and different types of errors and problems that could occur during production and transportation.

Path planning, calendaring, scheduling, or programming are good resources for executing a set of processes and operations in a given period [7]. Path planning differs from Stock Management because it does not focus solely on resources available but on all processes carried out from the selection of the raw material until the finished product reaches its final consumption or until the result is obtained desired, being related to the allocation of resources [12].

In general, path plannings [13] analyzes all existing variables in the production process (steps to be followed, path options, priority order for products or steps, processes to be completed, among others), with the aim of meeting deadlines, supplying high quality

products, and reduce costs for the manufacturer and consumer. Currently, one of the main problems found in all areas of activity is the failure to meet delivery deadlines and the excessive amount of work that requires qualified or specialized labour [14].

Nowadays the world requires mass customization, where the end consumer chooses several details and changes for their product, the industry finds it difficult to meet these requirements. Each product is unique, with characteristics that differ from others of the same model, such as colors, accessories, height, density, and other factors that vary from product to product [15].

Figure 2.1 is an example of a generic trajectory planning model, highlighting the steps that the product must take according to the type of load. (simple-load ou multi-load). In it, we see that each load first leaves the BUFs (loading position of loads) for its type of load, passes through specific points on the line (C1, C2, and C3), goes to the waiting area of its type of load before making the trip again. It is also possible to observe that the line models are different for single-loads and multi-loads, but they repeat so that the AGVs do not collide with each other during material transport within the cycle shown in the diagram of this image.

The most common uses of AGV technologies are in control systems for warehouses and manufacturing, i.e. stock organisation and the like [6], [17]. Although it is more widespread today, the concept of AMR was originally created in 1987 [18].

Currently, there are various uses for these systems, which can be in the areas of health, hospitality, industry, administration and others. We have examples of the use of AMRs in manufacturing, where robots can help transport materials, load or lift equipment, or even, in a collaborative way with the employee, carry out tasks together with the human being. Within this environment we also see robots with articulated arms for transporting long or heavy materials. In warehouses we see AMRs sorting goods, using programmes that make it identify the product and know where to store it. Within this same environment we also have robots for organising stock, locating inventories, collaborative robots for transporting and archiving materials, and robots that carry out all the fetching and carrying of stored

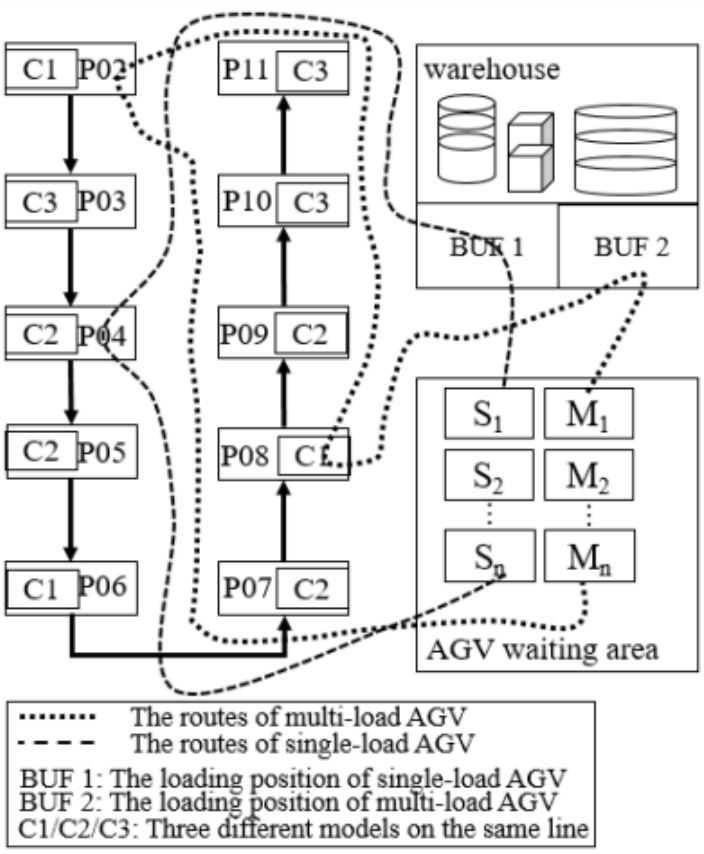

Figure 2.1: Schematic diagram of an assembly line and material transport process (S are single-load AGVs and M are multi-load AGVs) [16]

products by themselves.

Still in the context of uses for AMR robots, we can find them in hospitals transporting medicines to the various sectors, while other models can make video calls with the doctor for patients in a state of observation. In addition, we see them in hotels and aviation transporting luggage. Jeon and Lee [19], analyzed the use of a fleet of AMR robots that would be implemented in the hospital to perform various tasks within the delivery service and others. These would follow marked routes to transport objects, surgical materials, and whatever else was necessary, and could perform various tasks in sequence using a system that is easy for hospital staff to handle. Just like in manufacturing environments with separate sectors, each focused on specialization, in a hospital there are areas where each professional works differently. This fleet of robots would be responsible for serving

all the professionals along their route. AMRs can also be used in agriculture to find the best and fastest route to spray a greenhouse or a plantation [20]. Figure 2.2 shows the procedure necessary to deliver this robot. In the figure, it can be seen that as soon as the user requests the delivery of an item, the server analyzes the entire task queue, that is, the entire order of requests made by all users of the system, to create the delivery priority schedule. After that, it allocates tasks to the robot so that it can fulfill them in the established order, and at the end, updates the status of the requested task to the user who requested it.

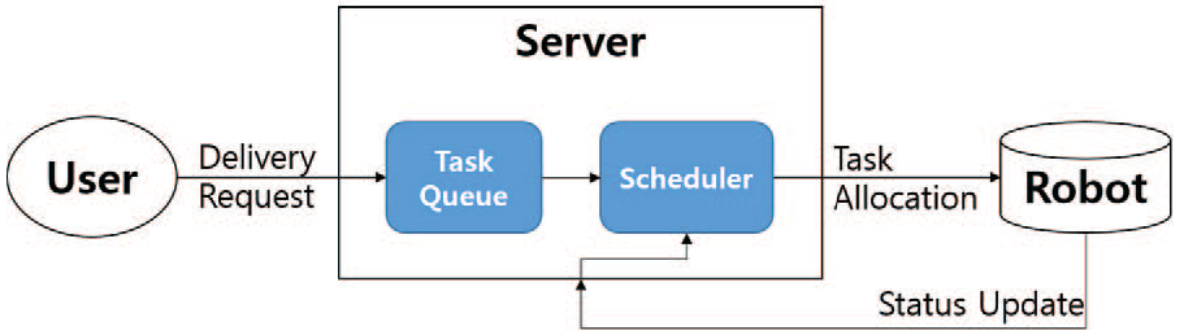

Figure 2.2: Steps for carrying out robot deliveries in a hospital environment [19]

For use in restaurants, where robots deliver food to the customer's table, path planning depends on the order in which the food is ready and the position of the tables. By following the numerical order of the tables and the pre-established positions, the robot is able to travel the route without colliding with obstacles or people, until it reaches the table where the order came from. Figure 2.3 shows some of these possible uses through path planning.

Each type of operation requires different route planning. There is no generic model that is used in all sectors, but rather a sequence of stages that changes according to demand. In other words, there may be preferences for each type of sequence. For example: for manufacturing companies, it is necessary to analyse the entire process, defining the paths and stages through which the material passes so that time is optimised, ensuring that the part is manufactured in the shortest possible time, without errors or delays between stages. For use in restaurants, where robots deliver food to the customer's table,

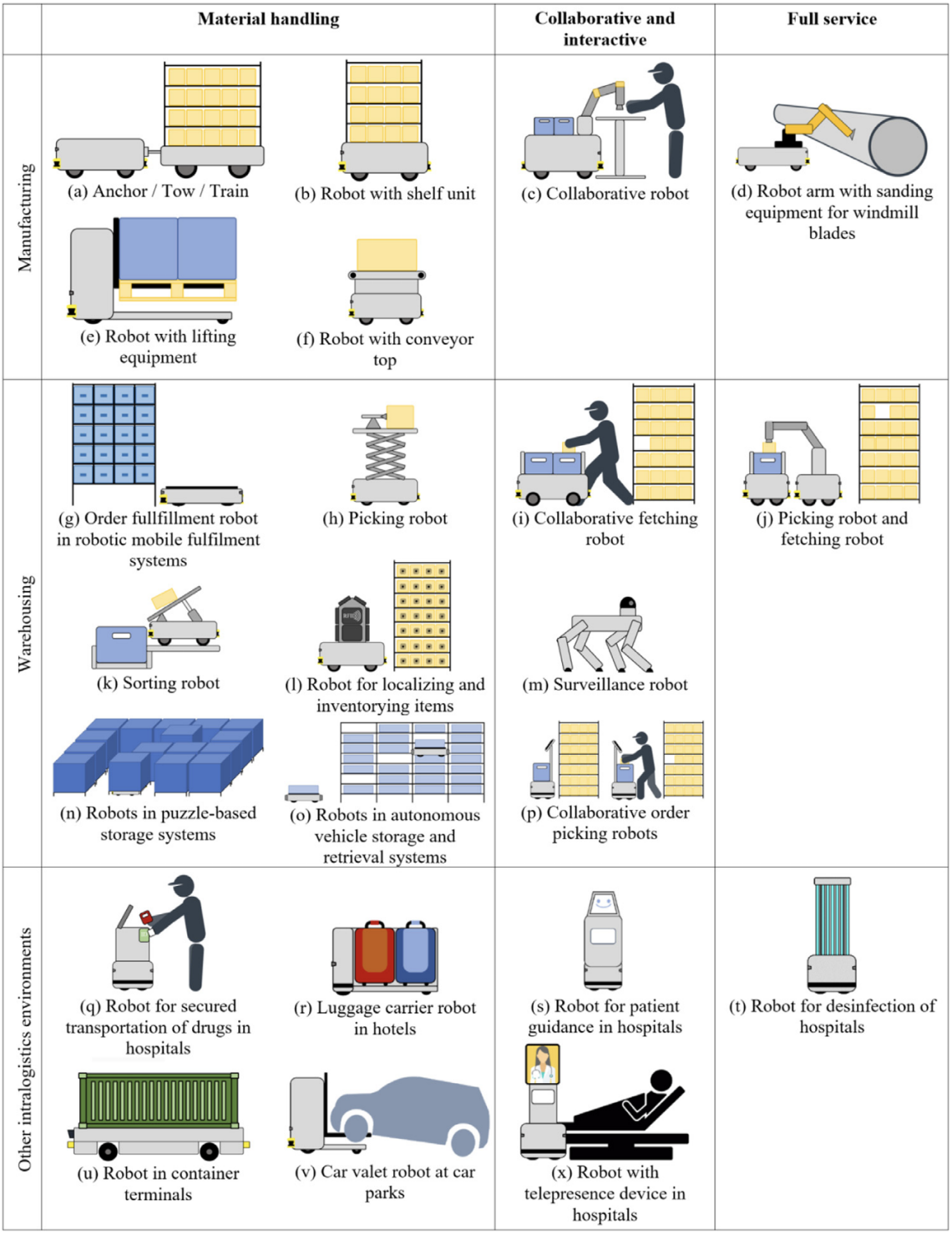

Figure 2.3: Demonstration AMRs types and their applications [6]

path planning depends on the order in which the food is ready and the position of the tables. By following the numerical order of the tables and the pre-established positions, the robot is able to travel the route without colliding with obstacles or people, until it reaches the table where the order came from.

Mohan [13] presents models of methods that can be used to try to reach a consensus on general path planning, regardless of the product to be referenced. Among the possible types of path planning, some examples will be cited here. One can use exact methods, where optimized models of operational functioning are constructed to obtain a exact solution that transforms the problems encountered into types of future restrictions, or approximate methods, which will be explained below.

2.3 Path planning for robots

It was proposed to create an algorithm, or modify an existing algorithm, that would be applicable to robots in a logistics warehousing environment using AMRs. This algorithm would check the map into which it would be inserted, analyse the point from which it should depart and the point to which it should arrive (start point and end point), the obstacles along the way and then identify the ideal path between the points, avoiding collisions with obstacles and taking this path as quickly as possible, also checking the distance needed to travel it.

Alves et al. [21] analyzed online routes using pre-instructed paths. In this path planning analyzed by them, it is essential to calculate the best route that the robot can follow to cover the entire path, as well as to evaluate different situations and ways to transport the object so that the total distance and time are as short as possible. This route can be seen in Figure 2.4 below, where the positions of the start point, end point, and intermediate points through which the robot must pass before reaching the final point are initially defined. Once defined, the path planning begins, checking the distance between all points, the possible routes, and obstructions within the map. With the path planned, the best route found is scheduled in the robot, and the function is executed so

that it performs all the necessary steps, passing through all the points.

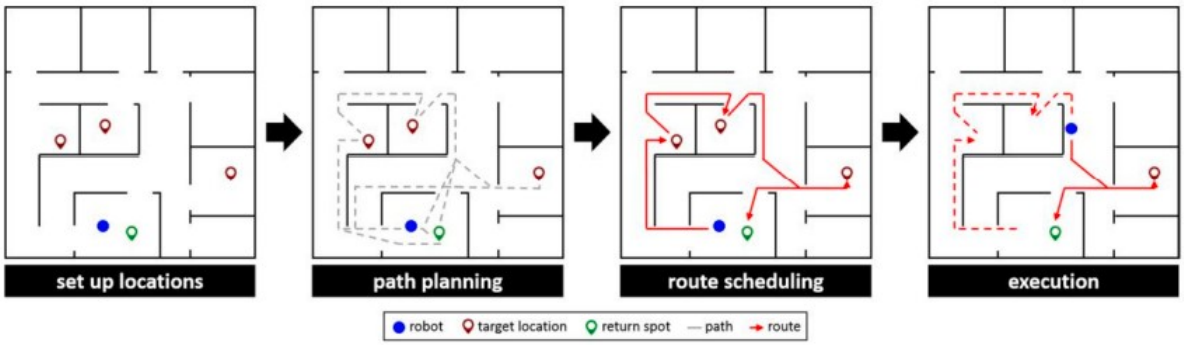

Figure 2.4: Steps of route path planning [21]

An example is the study by Zhang et al [22], that analyzes the use of AMRs for order separation. Aiming to optimize path planning to order, catalog, and separate orders and goods, the authors indicate using robots as a possible improvement in the Picker-to-Parts system within a warehouse or stock. Knowing the exact location of the object to be separated, the robot can automatically and efficiently collect the products and separate them for sale, delivery, and other activities. Taking this to our object of study, the robot knowing the exact initial and final location of the objects to be transported, as well as the route and the steps that need to be completed, one can verify which would be the most efficient way to complete the route.

One of the problems in executing any path planning for robots using routes is the need to identify the perception rates of the computational environment. The decision about the type of wireless network to be used is one of the initial points in the program design. It is assumed that the robot, which uses degrees of freedom in various axes (x, y, or z), cannot have wires or any type of equipment that hinders its mobility. Moreover, when creating the algorithm, it must be efficient and quick in reading and executing commands, because the longer the algorithm takes to function, the longer the robot will take to receive and execute the obtained data [23].

To calculate all existing routes and determine which would be the best, it is necessary to have an ideal computational environment to perform the equations and initiate the algorithm to be used. It is also necessary to know the number of robots that will use

this algorithm. Calculating the processing time requires knowing how many devices will be used. AMRs used in industries are programmed to always be available, or to operate for the longest possible time. Even when using more than one robot simultaneously, it is necessary to keep in mind that they will never be working 100% of the time, which, in practice, leads to errors in route planning. It is challenging to create a programming environment where all existing factors that can alter the data or the programmed routine can be considered. For this reason, route planning applicable to factories or automated environments encounters difficulties in its effectiveness, as it requires long periods of data collection to achieve better performance and analyze all possibilities [24].

Even though it is a separate case, the procedure adopted in this case is similar to the project's objective since with all the variant data entered into the program, the user starts the ideal path planning for the specific case. The server performs the analysis and, after path planning all the steps, sends the data to the robot, which performs the tasks of picking up the objects, going through the mandatory steps, and placing them in the desired location, all of this in the locations marked by the TAGS existing in the environment where the AMR will work.

Even though robots are not collaborative types that work together with humans in the same environment, the way in which human-robot interaction (HRI) occurs must be considered. The robot system must be partly intuitive and easy to operate to ensure that the person responsible for interacting with it can carry out all the processes with the minimum delay, performing all the tasks effectively and presenting solutions more quickly. Its interface must, therefore, be similar to the human system [25].

One of the challenges in creating path plannings for robots is the need to meet the stipulated schedule using the minimum amount of resources and respecting the inter-coupled restrictions that require distinct steps during the process. Designing perfect heuristics that act quickly and effectively requires collecting a large amount of data and information since an almost instantaneous response time must be achieved using the analysis of what is collected. It is therefore necessary to extract as much knowledge as possible about the entire scope of the project [26], [27].

2.3.1 Exact Methods

The planning of a robot's path within a manufacturing or storage environment is important for the management of these services. During the operation of a production system, it is inevitable that there will be a need to organize a dynamic route planning to ensure the quality and efficiency of the entire process.

Knowledge of the environment is also a complexity linked to the problem. A navigation strategy is crucial to obtain the shortest path based on the environment's configurations and obstacle detection [28].

The Dijkstra algorithm calculates the lowest cost from the starting node to the origin node, checking the weight of the neighboring nodes and analyzing if the lowest cost path is known [29]. The Djikstra algorithm performs this task by creating an auxiliary vector between the start and end points, checking the nodes near this vector and which of them are part of the optimal path [30]. This is one of the most commonly used algorithm models for path planning today.

2.3.2 Approximate Methods

Some approximation algorithms include First Fit analysis (capacity analysis to allocate the next item to the first compartment, basing this classification on its size), Next Fit (if the capacity of the previous compartment is not sufficient, a new compartment is opened), and Best Fit (allocation of the next item to the compartment with the smallest capacity that is still sufficient) [31], [32].

Approximate methods can be used in industrial environments, but also in various other sectors where it is necessary to plan the path to be taken. Dexter and Traub [33] evaluated heuristics to achieve a more dynamic allocation of operating rooms with the aim of maximizing the utilization and organization of surgeries within a hospital. They also analyzed the Early Start Time heuristic, which verifies the earliest start time available to allocate a surgery, as well as the Latest Start Time heuristic, which verifies the shortest surgery duration, the available time in the room and allocates it to the earliest possible

time, thus speeding up space for the rest of the day.

The analysis presented in Lenstra et al. [34] considers the use of parallel machines that perform different tasks. It compares the processing time of each machine and the deadline of the products to be manufactured on them, one independent of the other, thus obtaining a polynomial approximation algorithm. Hochbaum and Shmoys [35] verified a bin-packing problem also using a polynomial time algorithm in which the maximum number of bins is used in the construction of the packaging, since in some practical applications this "overflow" indicates the natural flexibility of the bin.

The D* (Focussed Dynamic A*) algorithm [36] and its variations (D* Lite, Space D*) combine the heuristic calculation of A* with incremental searches, allowing the optimization of A* to be adapted to a dynamic environment. This has wide application in multi-robot environments since, depending on how it is operated, it can smooth the path to be taken, increasing the possibility of transition between cells [28].

The A* Algorithm combines conventional calculations with heuristic analysis. It has the formula:

$$f(n) = g(n) + h(n)$$

The term $f(n)$ is the total calculated weight (cost) of the equation, where $g(n)$ indicates the weight cost that has been accumulated from the initial node to the current node (n), and $h(n)$ is the heuristic calculation of the remaining cost needed to go from the current node (n) to the end point. In other words, it stores the cost of the current point by adding it to the cost of the next point to be analysed in order to find the optimal path [37]. This cost, for example, could be the distance between cells (or nodes), which allows it to predict the distance between nodes and calculate the length of nodes that are irrelevant to the optimal path.

2.4 Computer language

In the 1950s, programs were greatly hindered by the available computer resources, primitive computing systems, and the need for several people to create an efficient program since programming a computer was highly complex. At the time, the idea of programs that worked automatically was a series of decimal numbers where the orders to be sent to the machine were recorded, which were seen as machine language. However, in early 1954, the creation of the A-2 compiler changed the way programming was done [38]. By sending and receiving pseudo-instructions, this compiler allowed compilation instructions to have various forms without being tied to decimal numbers. The commands were abbreviated and translated for machine reading.

The problem with automatic programming languages at the time was the high cost of use, the slowdown, and the low speed at which they read and sent algorithms. They also needed operators with high computational skills to understand and execute the programs and meet the users' needs. The cost of the programmers was much higher than that of the computer itself due to the high level of education required. Much of the programming time was spent decoding and translating the program to the computer. This problem was also seen with the algebraic system used by Laning and Zierler from MIT [38].

The A-2 compiler and the Laning and Zierler system served as the basis for creating the FORTRAN programming language (IBM Mathematical Formula Translation System), created by John Backus and IBM (International Business Machines Corporation). This, officially launched in 1957, was the precursor of programming languages [38]. Over the decades, the various programs created were developed, modified, and updated until today; wherein the digital age, there have several programming languages, such as Java, C++, and Python, the last two of which were used to develop the algorithm for this work.

The Python programming language, renowned for its ease of construction, empowers programmers of all levels to create both simple and complex programs with confidence. Its good overall performance and support for various types of programming (functional, object-oriented, imperative, and others) further enhance its appeal.

To create an intuitive computational language that could solve the problems and flaws of the C language, Guido Van Hossum (Amsterdam, Netherlands) [39], one of the developers of the ABC language and the CWI amoeba system, created the Python programming language in 1982. The name given by Hossum, G. came from his favorite television program, *Monty Python's Flying Circus* [39], with no relation to the name of the snake.

For the correct use of Python, it is necessary to import external libraries into the algorithm that will allow the use of external functions that are not integrated within it. Below are some of the libraries which were used in this work.

Time library

The Time library has the function of allowing Python to perform time calculations, which can be the manipulation of this time, pauses, and specific intervals within the code. It even allows you to see the current time, among other functionalities [40], [41].

Numpy library

The NumPy array incorporates several fundamental concepts into the Python library. It is a structure for efficiently storing and accessing multidimensional data. It consists of a memory that can store shapes, strides, data types, and other types of arrays. In addition to creating new arrays, it also allows existing arrays to be reshaped, filled, searched, and counted and also allows the reading and writing of files [41].

Matplotlib library

Matplotlib is a library used to create graphs, histograms, explore and visualize data. It is a plotting library for Python, and can be used in conjunction with the NumPy library [42].

Math library

The Math library allows Python to calculate simple and advanced mathematical functions, such as trigonometry, arithmetic calculations, and heuristics. It serves as the basis for all functions used within the Python program [43].

Chapter 3

Methodology

In this work, after verifying the approximation method of the A* algorithm and the exact method of Dijkstra's algorithm, it was decided to use the approximation method of the A* algorithm, as it proved to be slightly faster in performing the calculations and returning the desired result. The original A* algorithm was not used in this work, but rather some of its variations, which will be mentioned later. To calculate the trajectory planning, the Python programming language was used.

3.0.1 Maps used

For both variants studied, two different maps were analyzed. The two maps were made in two-dimensional grids to obtain precise data on the robot's movement along the x and y axes.

Map I in Figure 3.1 is a map with dimensions 70 x 70 meters, starting at point -10 on both axes, which contains only two barriers as obstacles throughout the entire course. This is a simple map, aimed at verifying the robot's movement and the proper functioning of the algorithm in environments with few obstacles and a low degree of movement difficulty.

Map II (Figure 3.2) was created based on the map used in the Robot@Factory competition environment [44] (Figure 3.3), which has four entry warehouses (boxes in the upper left) and four exit warehouses (boxes in the lower right), where all start points

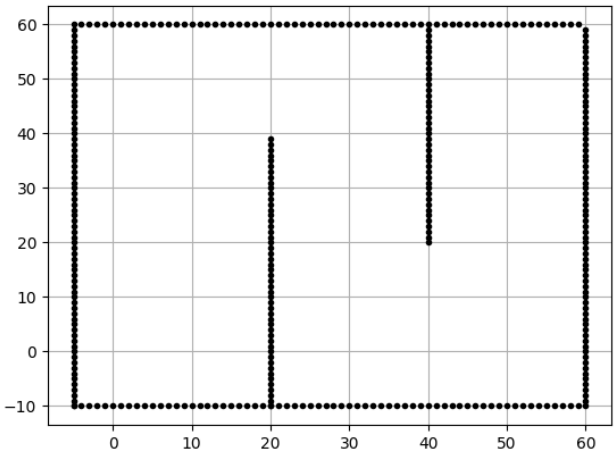

Figure 3.1: Visual representation of Map I

must have routes to all end points, one path at a time. In this map, there are two large central obstacles, which simulate the presence of machines or shelves, for example, in a warehouse environment.

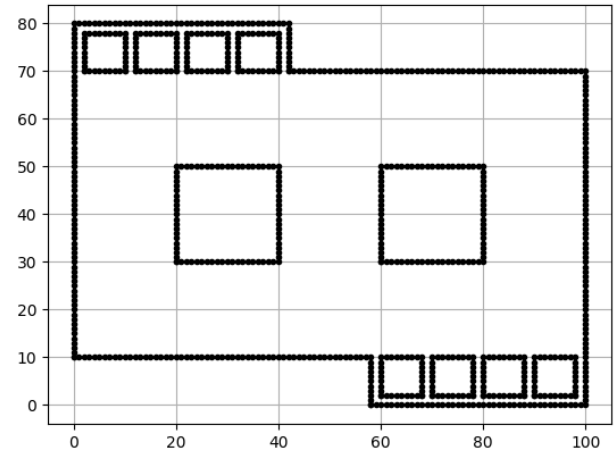

Figure 3.2: Visual representation of Map II

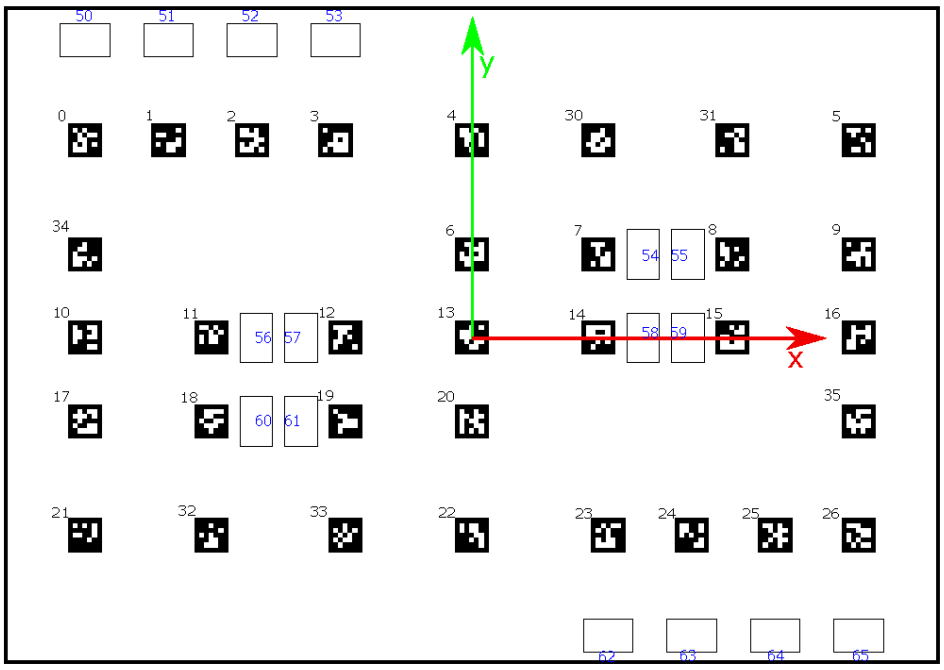

Figure 3.3: The Shop Floor used in Robot@Factory competition [44]

3.1 A* algorithm and variants

It is possible to observe the basic operation of A* in Figure 3.4, where we have the Start Point in green, the Destination Point in red, and the points located around these points, thus seeing how the algorithm analyses the map to calculate the best route between the defined points. The algorithm checks all the points around the Start Point, checks the next points it can use to move around and the existence of obstacles along the way, and with the result of calculating the formula mentioned above, it analyses which points will cost the least to move around until it reaches the Destination Point.

Figure 3.5 demonstrates an example of how A* algorithms work. We see a starting point (the red star) and an endpoint (X on the right side of the image). The algorithm will perform a guided search for the optimal route through the map it is on. When analyzing all possible routes, it will return the best path found, so that the journey can be made with the highest possible accuracy, without passing through locations irrelevant to the optimal path. When analyzing the route, the A* algorithm checks the direction where

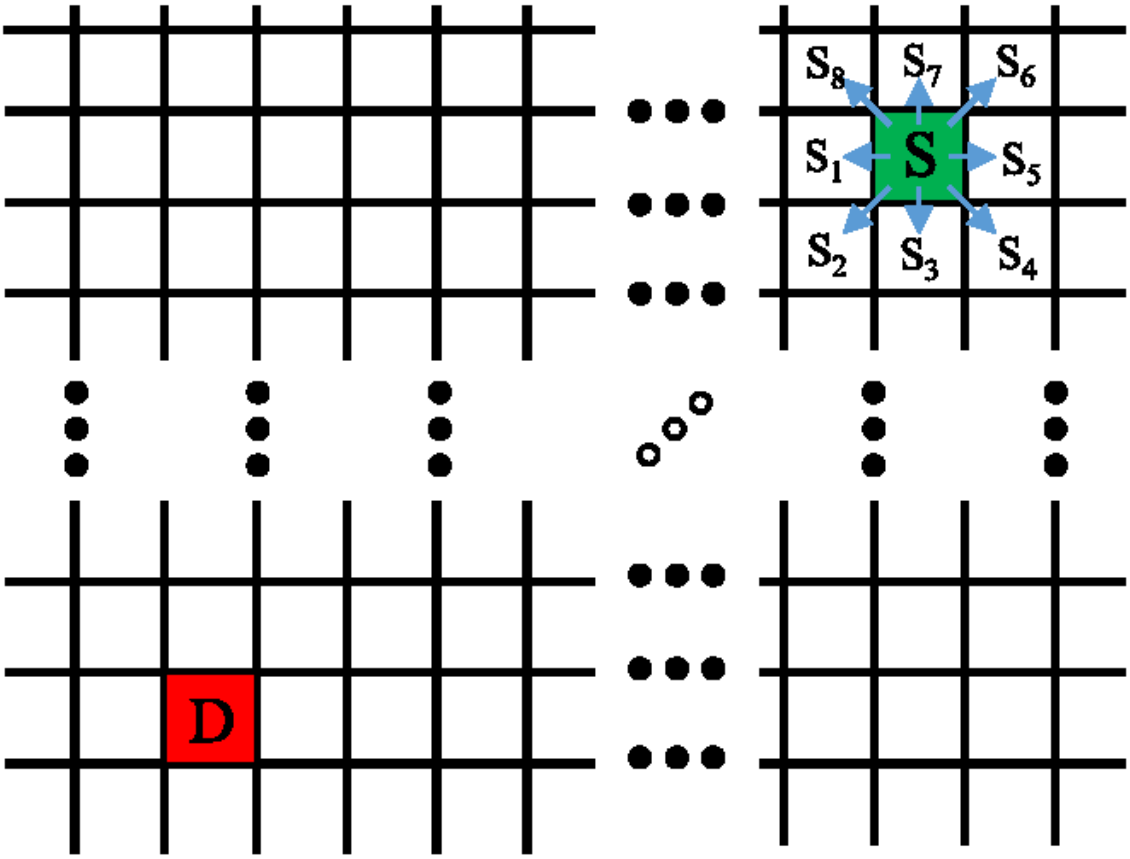

Figure 3.4: Basic operating diagram of A* [45]

the endpoint is located and bases its path planning on it, making the route shorter due to its objectivity in finding only the best path to reach the endpoint.

This algorithm is designed to optimize the transportation routes of objects. The use of a simulation environment, even if only computer-based, during planning allows for the exploration of different strategies and approaches, as it reduces the time spent in this stage.

The definition of the map where the robot will be located is one of the first steps, as after it the initial, final, obstacle points, and points along the route are defined. With this information, it is possible to analyze how many possible routes exist and the total number of available routes, how many steps in total are necessary to complete the entire route, and whether there will be changes between the stages, such as adding an extra

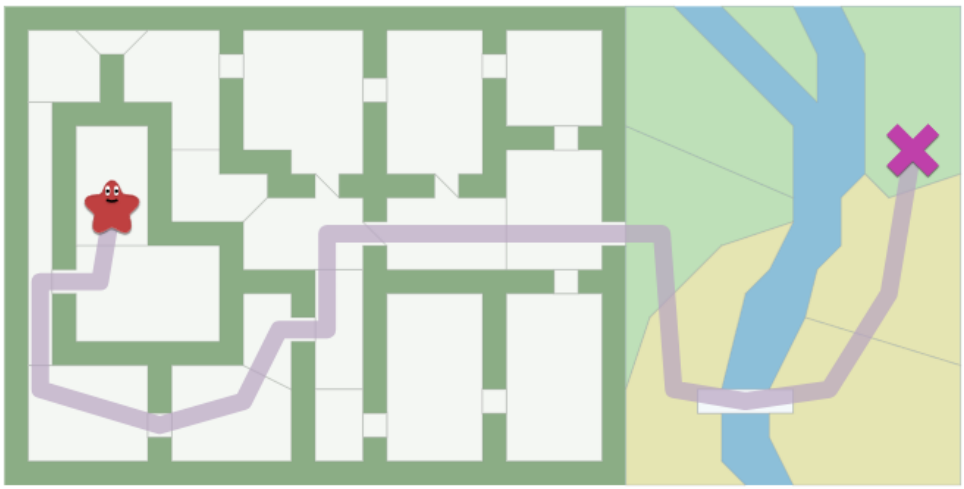

Figure 3.5: Example of how an A* algorithm works [46]

stage where the robot must pause or perform a secondary function.

This work does not analyze the use of the original A* algorithm. The original A* is not suitable when it is necessary to make turns, move diagonally, or perform a rotation during the path. Moreover, it also encounters difficulties in parameter selection [47]. Some initial parameters must be configured before the algorithm starts, such as the size of the robot, the distance between each point, definitions of the functions at each stage (rotate, move forward, move backward, etc.), and because of this, the original A* is inadequate for this occasion. Therefore, two variants of this algorithm were used: Sakai A* variant proposed by Sakai and Kanargias [48] and the Sakai A* variant proposed by Sarim Mehdi [49].To evaluate the response effectiveness of the two analyzed variants of the A* algorithm, the execution time of the variants (total time required to calculate the optimal path between the starting point and the endpoint) and the length of the found path were checked. (the shortest distance between the start point and the end point). For the calculation of the path length, a standard length of 1 meter between one point and another was used.

3.1.1 Analysis of Sakai A* variant

Sakai A* variant is based on the implementation proposed by Sakai and Kanargias [48]. After importing the necessary libraries (math, matplotlib, and time) and defining the animation control variable, which will give us the graphical result for the map, the algorithm inserts its functions inside the **AStarPlanner** class:

- **___init___:** starts the A* planner, defining the size of the grid, the coordinates of the obstacles on the map, and the radius of the robot so that it is possible to check for collisions with obstacles.
- **get_motion_model:** defines the possible movements of the robot about the nodes around it.
- **calc_obstacle_map:** generates the obstacle map and defines their positions on the x and y axes.
- **Class Node:** represents the nodes on the map, containing information about the position of this node on the grid, the cost required to reach this node by accumulating the costs of the nodes along the path to it, and finally visualizes it by printing its information.
- **calc_heuristic:** it calculates the heuristic for A*, the Euclidean distance between two nodes.
- **verify_algorithm:** it checks for the existence of any node that is not within the created map, or that collides with an obstacle.
- **calc_grid_position, calc_xy_index, and calc_grid_index:** these functions help to convert between real-world coordinates and the index in the grid.
- **calc_obstacle_map:** generates the obstacles on the map based on the x and y coordinates provided earlier in the algorithm.

- **get_motion_model:** defines the possible movements of the robot within the grid (up, down, left, right, diagonals).
- **calculate_path_length:** calculates the total length of the final path by summing the Euclidean distance between each point on the path.
- **execution_time:** by recording the initial time and the time it takes to find each node, this function calculates the execution time of the algorithm until it finds the destination, subtracting the initial time from the final time.

In the end, the algorithm provides a graphical visualization of the path, representing the positions of obstacles, the starting point and the ending point, as well as the final path considered as the best path.

Essentially, this algorithm leverages the power of the A* algorithm, a highly efficient path planning method. It meticulously checks for collisions with obstacles, expands nodes based on a motion model, calculates heuristics and costs to find the shortest path, and provides a real-time graphical visualization of the entire process.

As a versatile tool, this algorithm can be applied in a multitude of situations where finding the best routes in an obstacle-filled space is a necessity. Its adaptability makes it a valuable resource in a variety of scenarios.

Sakai A* Variant on Map I

The A* algorithm of the Sakai variant generates a simple grid, with two walls as obstacles and fixed start and end points in a similar manner, in addition to showing the shortest path between them. Its operation is simple, as the grid always has exact measurements and predefined obstacles. Therefore, its simplicity makes the compilation faster, requiring fewer indentations and fewer functions programmed in the algorithm, as it only needs to move from the start point to the end point while avoiding the two existing obstacle barriers, as seen in Figure 3.6.

To verify the effectiveness of this map, 10 different locations were selected and tested for the start point and the end point, in order to analyze how the algorithm behaves

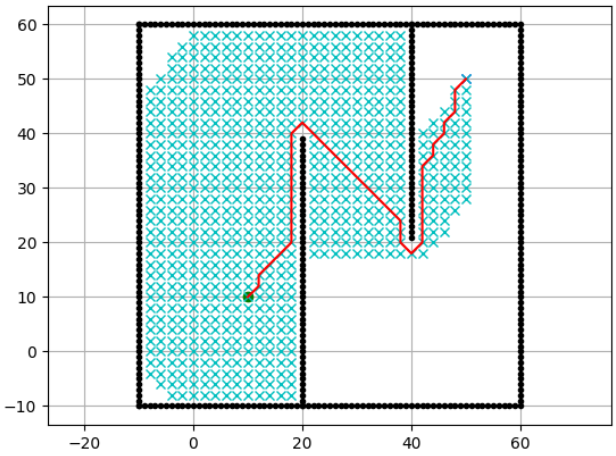

Figure 3.6: Visual result of Sakai A* variant algorithm [48]

when it has to find the optimal path at different distances. The results of these tests are available in the Table 3.1, and the graphically generated results of two of these tests (test #3 and test #4, chosen randomly) can be seen in Figures 3.7 and 3.8.

Start's point X position	**Start's point Y position**	**End's point X position**	**End's point Y position**	**Execution time (s)**	**Path length (m)**
10	10	10	0	0.1829	10.0000
1	-5	50	50	3.8222	123.8823
10	10	30	30	0.9222	52.2843
10	10	50	0	2.0139	90.5685
22	10	50	30	0.3570	37.4558
0	0	50	50	7.5176	118.7107
-8	-5	50	30	2.7278	108.0244
10	-5	38	30	1.3290	75.1127
-5	58	40	-5	2.1086	88.9117
-9	-9	59	59	4.3093	144.1665

Table 3.1: Analysis of results based on Sakai * variant algorithm, with changes in the start and end positions

After these 10 tests, a graph was made to compare the relationship between execution time and path length (Figure 3.9), when the algorithm needs to generate the result graphically.

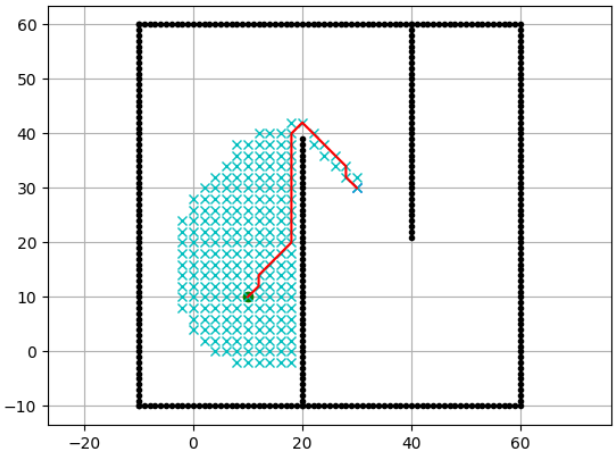

Figure 3.7: Test #3 Graphical interface - Sakai A* variant on map I

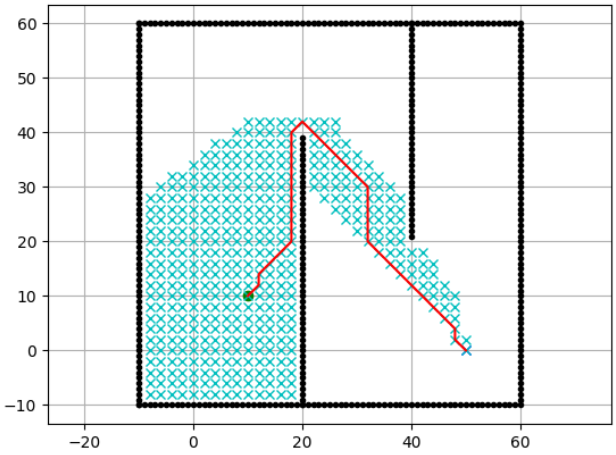

Figure 3.8: Test #4 Graphical interface - Sakai A* variant on map I

To analyze whether the algorithm's functionality and the calculation of the distance between the points are correct, test #1, available in Figure 3.10), examines the distance between these two points by keeping the x position for both points constant and altering the y distance by 10 meters. Thus, it was observed that the distance between the start and end point was 10 meters, confirming the proper functioning of the algorithm.

The last test of the Table 3.1 (test #10) considered the greatest possible distance

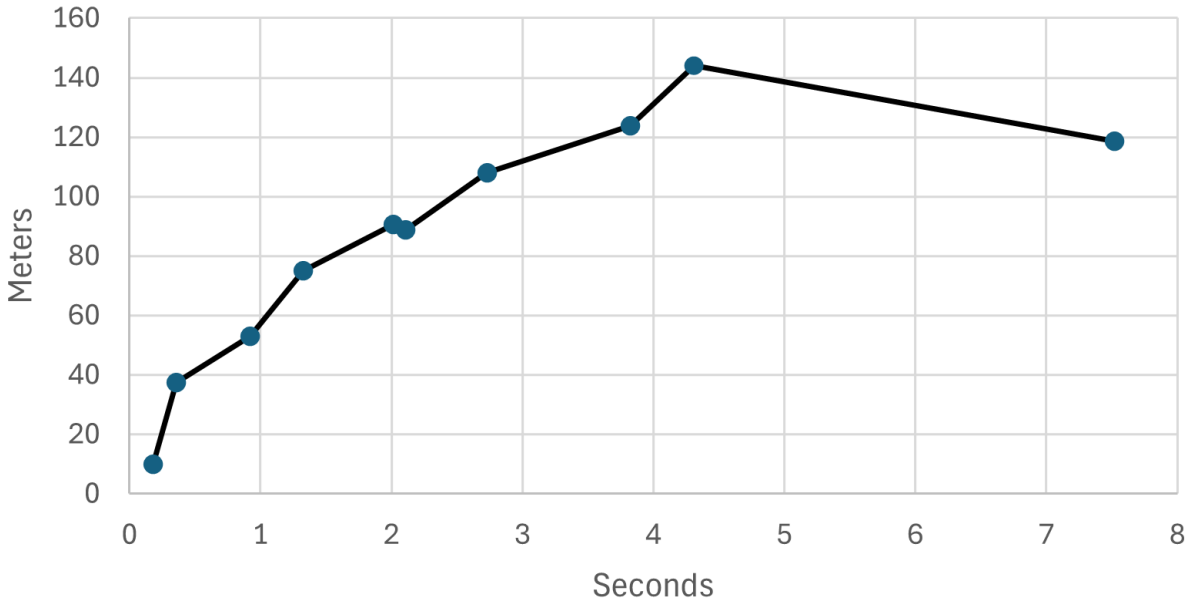

Figure 3.9: Comparison graphic for Sakai A* Variant on Map I

between two points on the created map, whose result indicates the longest execution time and path length that would be covered within this map.

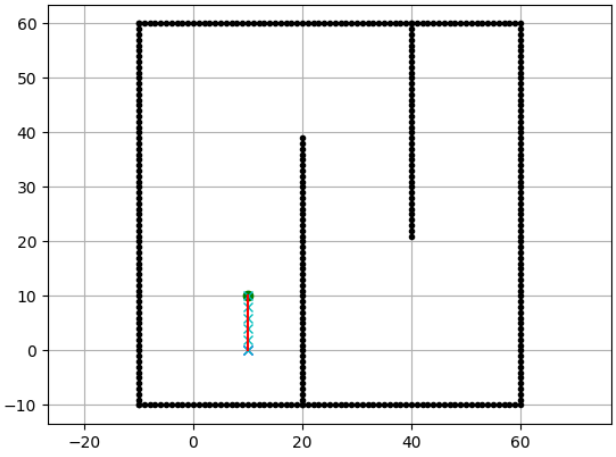

Figure 3.10: Sakai A* Variant algorithm test with 10 meter start-to-end distance

After performing all the time calculations related to the algorithm with the generation

of visual graphs, the same comparative analyses were conducted, but without the computational graphs of the generated algorithm. Therefore, the execution time of this step is verified using the same algorithm but disabling the graphical generation of the map images. The results of these 10 tests are in Table 3.2, where they are organized from the shortest to the longest execution time.

Start's point X position	**Start's point Y position**	**End's point X position**	**End's point Y position**	**Execution time (s)**	**Path length (m)**
10	10	10	0	0.0009	10.0000
22	10	50	30	0.0050	37.4558
10	10	30	30	0.0170	52.2843
10	-5	38	30	0.0244	75.1127
10	10	50	0	0.0345	90.5685
-8	-5	50	30	0.0431	108.0244
-5	58	40	-5	0.0492	88.9117
-9	-9	59	59	0.0620	144.1665
1	-5	50	50	0.0671	123.8823
0	0	50	50	0.0698	118.7107

Table 3.2: Results obtained with Sakai A* variant on Map I

It is possible to verify that the increase in the length of the path is directly related to the increase in execution time, since according to what can be observed, the greater the distance between the two points, the longer the algorithm will need to find the best path, as can be seen in the graph in Figure 3.11.

A comparison was made between the execution time results with and without graphs to verify the difference in seconds and percentage in Table 3.3. In it, we have the calculation of the delta time (execution time with graphs - execution time without graphs) to obtain the time used by the algorithm to generate the graphs, excluding the time used for the algorithm's operation. We also have the percentage variation between the execution time with graphics and without graphics. As a result, it is observed that the difference in execution time is greater when there are graphs and visual results within the algorithm. The order of the tests in this table is the same as in the previous tables.

Analyzing the results of Table 3.3, it was possible to observe that the median value

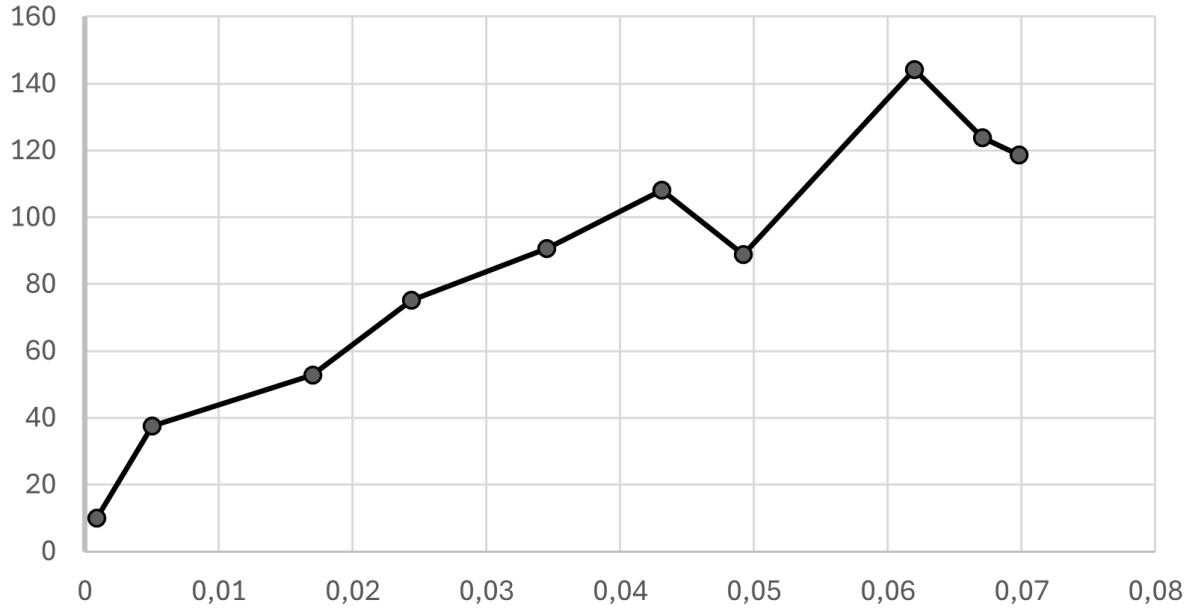

Figure 3.11: Sakai A* Variant on Map I Time (seconds) vs Length (meters) comparison

Execution time with graphics (s)	**Execution time without graphics (s)**	**Delta of the time (s)**	**Variation of time (%)**
0.1829	0.009	0.1739	95.08
3.8222	0.0671	3.7551	98.24
0.9222	0.0170	0.9052	98.16
2.0139	0.0345	1.9794	98.29
0.3570	0.0050	0.3520	98.60
7.5176	0.0698	7.4478	99.07
2,7278	0.0431	2.6847	98.42
1.3290	0.0244	1.3046	98.16
2.1086	0.0492	2.0594	97.67
4.3093	0.0620	4.2473	98.56

Table 3.3: Comparison of execution times for Sakai A* variant

among all the percentages of time variation was 98.26%, with this being the percentage of time used to create the graphs, and only the remaining 1.74% as the actual execution time of the algorithm.

Sakai A* Variant on Map II

In this step, a map similar to the map used by the competition Robot@Factory [44] was created, with the graphical representation of the central obstacles, the start point and end point, and the limits of the map itself, as shown in Figure 3.12.

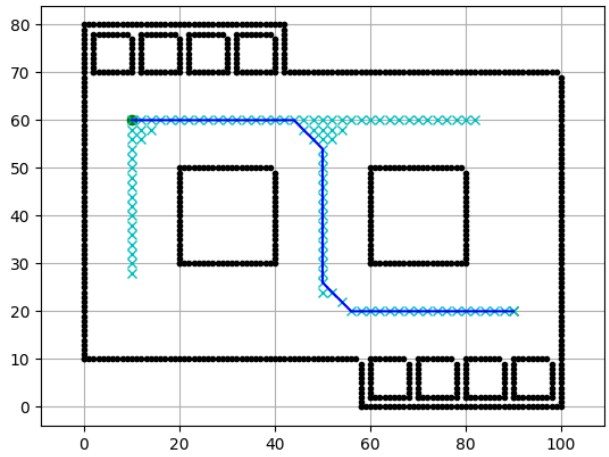

Figure 3.12: Sakai A* Variant algorithm applied to the Robot@Factory map

A new comparison was made using this new map, marking the distance between start and end points 1, 2, 3, and 4, as well as the length of the path (Table 3.4). To do this, it was necessary to create positions for all points about the x-axis, so the start points 1, 2, 3, and 4 were in positions 8, 16, 26, and 36, respectively, and the end points were in positions 64, 74, 84, and 90, respectively.

Through the analyses conducted, it was possible to observe that, for this last map created, the variation between the execution time of the algorithm with and without the generation of graphs is, on average, 99.4%, where the 0.6% represents the execution time of the algorithm itself.

Finally, to verify the execution time of the algorithm at different stages, it was analyzed at each step. Using as a basis the path from entry warehouse 1 to exit warehouse 4, where the path would be the longest possible within the route to be developed, 0.1751 seconds are used to generate the map, the obstacles, the warehouses, and all defined points (start and

Start point	End point	Path length (m)	Time with graphics (s)	Time without graphics (s)	Variation of time (%)
1	1	88.9706	0.6159	0.0030	99.5
1	2	98.9706	0.7047	0.0040	99.4
1	3	108.9706	0.7537	0.0050	99.3
1	4	114.9706	0.8459	0.0050	99.4
2	1	80.9706	0.6034	0.0030	99.5
2	2	90.9706	0.6152	0.0040	99.3
2	3	100.9706	0.7488	0.0040	99.5
2	4	106.9706	0.7825	0.0050	99.4
3	1	70.9706	0.4606	0.0020	99.6
3	2	80.9706	0.5808	0.0030	99.5
3	3	90.9706	0.6569	0.0030	99.5
3	4	96.9706	0.6216	0.0040	99.4
4	1	60.9706	0.5347	0.0030	99.4
4	2	70.9706	0.4985	0.0030	99.4
4	3	80.9706	0.5927	0.0034	99.4
4	4	86.9706	0.6299	0.0040	99.4

Table 3.4: Comparison of Sakai A* variant with Robot@Factory-like map

end), and 0.2921 seconds to define the coordinates within the variant of the A* algorithm. These calculations refer to the case where the graphical animation is generated. When it is not generated, the time to set the coordinates within the planner is 0.2760 seconds.

3.1.2 Analysis of Mehdi A* variant

Mehdi A* Variant is based on the implementation proposed by Sarim Mehdi [49]. With the definition of the libraries to be imported (time, numpy and matplotlib), the insertion of the functions related to A* began:

- **draw_horizontal_line and draw_vertical_line:** starts drawing the horizontal and vertical lines of the grid. Within it, the algorithm will go through the points along the path, mark each position as an obstacle, and add them to the lists of obstacle coordinates in x and y.

- **in_line_of_sight:** checks if there is a direct line of sight between two points without obstructions. By marking the obstacle grid, the coordinates of the starting and ending points are calculated, represented by x1 and y1 and x2 and y2, respectively, begin. In this calculation, a linear interpolation is also made between the points to check if there are obstacles between them.

- **key_points:** identifies strategic points on the map to optimize the search parameters, in addition to points irrelevant to the optimized search, such as corners and intersections. It also goes through all the points on the grid, checking whether they are obstacles or surrounded by empty spaces.

- **SearchAlgo:** implements A* within the algorithm. It records the initial and final coordinates (start and end points) and the dimensions of the grid and analyzes the list of key points. By calculating the cost of the nodes (from the initial to the current node), it also discovers the estimated heuristic to the end, thus creating a dictionary of nodes that contains all the nodes on the grid.

- **get_hval:** calculates the heuristic between two points, estimating the cost to move the robot from the current point to the final end, considering all movement possibilities (diagonal and orthogonal).

- **get_farthest_point:** used to find the furthest point in the given direction without obstacles.

- **jump_point:** explores only the corners to analyze and update the node costs, checks whether the algorithm has reached its end, and reconstructs the final path considered the best.

- **a_star:** implements the A* variant. While the set is empty, it selects the node with the lowest cost, explores its neighbors and calculates the costs for them, updates all nodes if a better path is found, and adds nodes with a good probability to the open list. It also changes the weight of the heuristic during the search.

- update_node_cost: updates the costs of neighboring nodes and decides whether or not they should be added to the open list.

When creating the obstacle grid, the starting points and the goal are defined, marking their positions in x and y; the map edges and internal walls are created, and the entire algorithm is visually plotted, defining the colors of the open and closed lists, as well as the color of the obstacles.

The modifications made to this algorithm were the additions of execution time calculations and path length, important factors to know how and when the algorithm communicates well with the robot.

Mehdi A* Variant on Map I

Based on the changes made to the algorithm to incorporate the necessary evaluation measures, ten tests were conducted with the Mehdi A* variant, as well as with the Sakai A* variant, using Map I (Figure 3.1), thus generating the results of these tests. At the end of the simulations, a comparison was made between the number of nodes the robot passed through to go from the start point to the end point. These data are presented in Table 3.5. Due to the fact that this variant cannot analyze points located at any position before zero or at zero on both axes, it was necessary to alter the start points and end points to obtain the analysis, with test #3 being the only one that maintained the same positions for both variants on Map I.

The graphical results of two of the tests conducted in Table 3.5, tests #3 and #4 as well as in the previous variant, are shown in Figures 3.13 and 3.14.

In this tests, it was possible to observe the algorithm's behavior when the start and end points are inserted and how it checks the path between these two points. The comparative graph between the execution time and the number of points between the start point and the end point referring to the last table is in Figure 3.15.

As in the previous variant, in this one the algorithm's responses were also analyzed in relation to the same points but without printing graphs, as shown in Table 3.6.

Start's point X position	Start's point Y position	End's point X position	End's point Y position	Execution time (s)	Path length (m)
5	5	35	45	0.9906	50
2	2	50	50	4.2806	90
10	10	30	30	1.0012	40
9	10	50	2	1.6328	68
22	10	50	10	0.0715	28
2	2	50	50	3.8916	90
2	30	50	10	1.1187	49
10	50	38	30	0.3024	28
2	48	40	2	1.4348	56
10	25	25	30	0.6100	25

Table 3.5: Analysis of results based on Mehdi A* variant algorithm on Map I

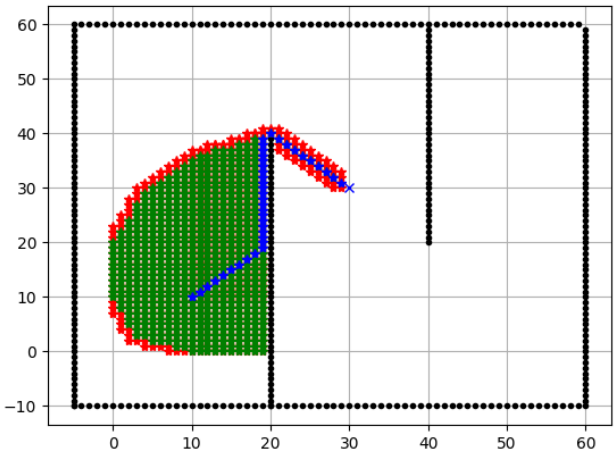

Figure 3.13: Test #3 graphical interface - Mehdi A* variant on map I

The comparative analysis of the number of points in relation to the execution time, present in Table 3.6 without printing graphs, is in Figure 3.16.

With the results obtained in Tables 3.5 and 3.6, it was possible to compare the execution time data with and without graphical generation, calculating the time delta (execution time with graphs - execution time without graphs) and the percentage variation between the obtained data, as shown in Table 3.7. In it, it is possible to observe that the

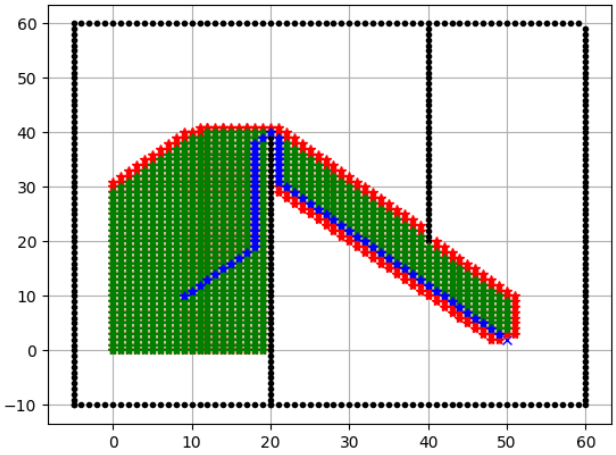

Figure 3.14: Test #4 graphical interface - Mehdi A* variant on map I

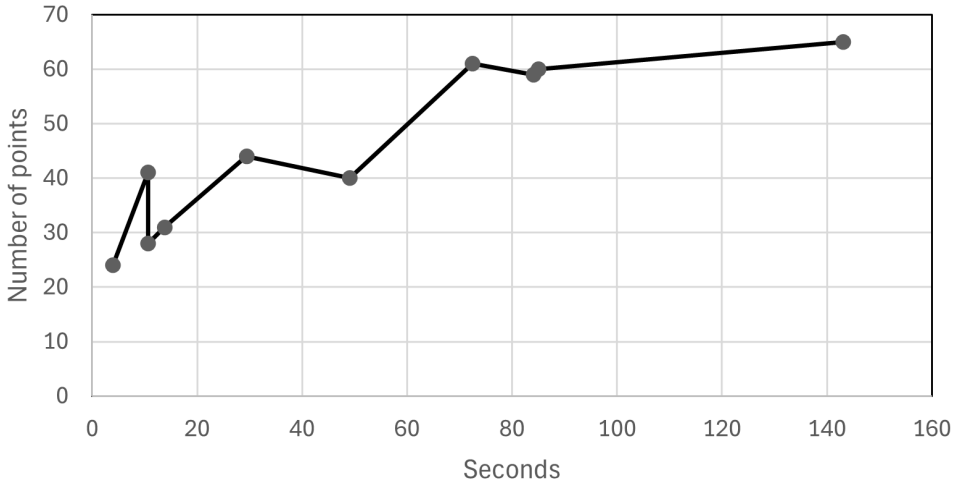

Figure 3.15: Comparision of Mehdi A* variant on Map I

median of the percentage time variation is 97.19%, therefore this percentage refers to the time used to generate only the graphical and visual part.

Start's point X position	Start's point Y position	End's point X position	End's point Y position	Execution time (s)	Path length (m)
5	5	35	45	0.0280	50
2	2	50	50	0.1155	90
10	10	30	30	0.0320	40
9	10	50	2	0.0450	68
22	10	50	10	0.0020	28
2	2	50	50	0.1110	90
2	30	50	10	0.0320	49
10	50	38	30	0.0080	28
2	48	40	2	0.0545	56
10	25	25	30	0.0150	25

Table 3.6: Analysis of results based on Mehdi A* variant algorithm on Map I without graphics

Execution time with graphics (s)	Execution time without graphics (s)	Delta of the time (s)	Variation of time (%)
0.9906	0.0280	0.9626	97.17
4.2806	0.1155	4.1651	97.30
1.0012	0.0320	0.9692	96.80
1.6328	0.0450	1.5878	97.24
0.0715	0.0020	0.0695	97.20
3.8916	0.1110	3.7806	97.15
1.1187	0.0320	1.0867	97.14
0.3024	0.0080	0.2944	97.35
1.4348	0.0545	1.3803	96.20
0.6100	0.0150	0.5950	97.54

Table 3.7: Comparison of execution times for Mehdi A* variant algorithm

Mehdi A* Variant on Map II

After checking the results obtained with the assignment of random start and end points within Map I, a comparison was made between the distances of the start point and the end point within Map II. The positions of these points are the same as those used in Map II of Sakai A* Variant (Figure 3.17).

Observing Table 3.8, one can see that the number of points present in the path considered optimal is always the same for outputs 1 and 2, regardless of which input is started from.

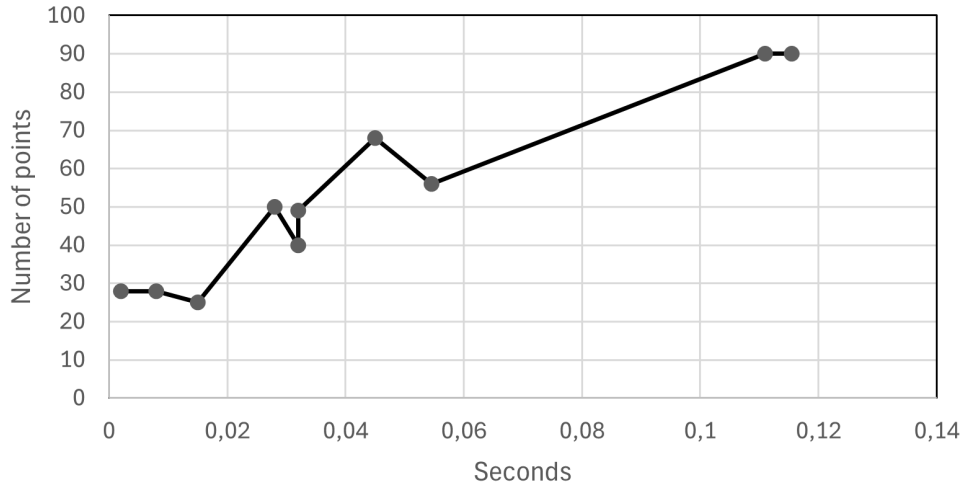

Figure 3.16: Comparision of Mehdi A* variant on map I without graphics

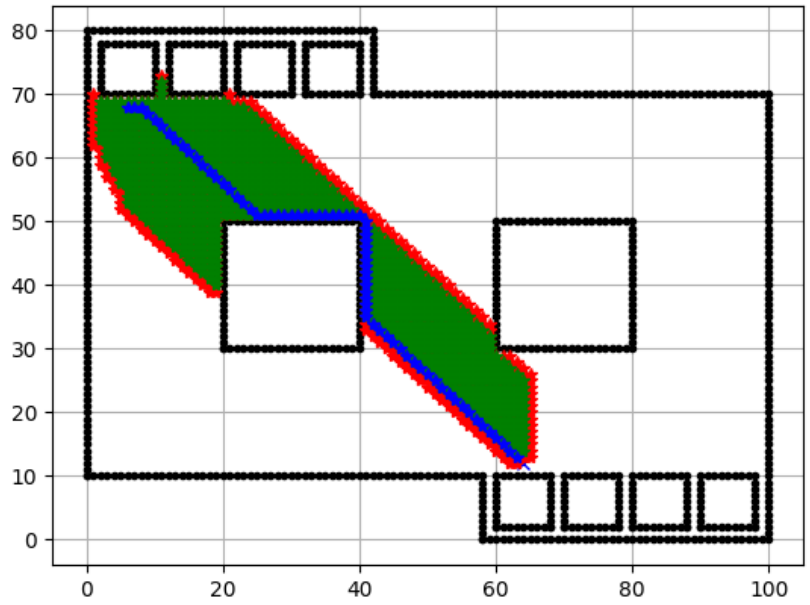

Figure 3.17: Mehdi A* Variant on Map II

At the end of the tests, it was possible to observe that, in addition to being slower to find the best path, the percentage of time variation with and without graphics is much higher, with the average of these percentages being 96.4%, that is, only 3.6% of the time is used by the algorithm and the remaining time is only for the animations.

Start point	**End point**	**Path length**	**Time with graphics (s)**	**Time without graphics (s)**	**Length Variation (%)**
1	1	71	1.4987	0.0525	96.5
1	2	71	0.9151	0.0300	96.7
1	3	78	1.1640	0.0475	95.9
1	4	84	1.4543	0.0660	95.5
2	1	63	1.1025	0.0400	96.4
2	2	70	0.7176	0.0230	96.8
2	3	76	1.0004	0.0390	96.1
2	4	56	1.3562	0.0570	95.8
3	1	56	0.8481	0.0305	96.4
3	2	2563	0.5571	0.0200	96.4
3	3	63	0.9831	0.0390	96.0
3	4	69	1.2847	0.0540	95.8
4	1	56	0.9587	0.0380	96.0
4	2	56	0.6871	0.0240	96.5
4	3	63	1.2092	0.0420	96.5
4	4	69	1.5956	0.0575	96.4

Table 3.8: Comparison of Mehdi A* variant with map II

Chapter 4

Results and Discussions

The evaluation measures reveal the performance disparities between the A* algorithm variants, emphasizing the need to balance efficiency and operational cost. The A* algorithm's heuristic function, with its dynamic adjustment to the search depth within the map grid, is a prime example of this balance. It ensures an equilibrium between path optimization and computational time, a critical factor in environments requiring rapid path processing and precise cost control. When running the algorithm in both variants with the same data repeatedly, the time variation between executions was approximately 0.1 seconds, demonstrating a minimal fluctuation in response time. It was noted that the greater the distance between the starting point and the destination, the longer the algorithm needs to calculate the best path. This relationship between distance and execution time becomes even more evident in complex maps, as they cause the algorithm to explore additional points in the search for the ideal path. Furthermore, when analyzing the visual result generated by the algorithm, it was noticeable that variant I does not present a visual distinction between the points in the open and closed lists, making it difficult to see the points already analyzed compared to those still under analysis.

Map I has limits of 70 meters for both the x and y axes (from -10 to 60 meters), with each mark representing a unit of 1 meter. Due to the grid resolution limitations and the graphic components' size, there may be discrepancies in the manual counting of the points when the map is enlarged, which may affect the accuracy of the user's visual analysis.

This simple map containing only two long obstacles was created to test how the algorithm would find the end point with two barriers along the way.

Map II resembles the map used in the Robot@Factory competition scenario of the National Robotics Festival of Portugal [44], allowing us to conduct tests in a controlled environment, with specific points and objectives distributed along the grid. However, its implementation in the variants presented difficulties, mainly due to the limitation of movements. Since the robot must follow fixed markings, diagonal movement is restricted. When reaching an axis change point, the robot must rotate 90 degrees to continue its path. Even so, obstacles and markings facilitate implementation within the map. The dimensions of 100 meters wide by 80 meters long were defined to avoid restrictions on the robot's movement and code execution. When trying to implement a reduced version, with a scale of 10% of the initial value, Variant II could not perform the calculations, as the points became too small to be recognized as a viable path.

4.1 Results of analysis of Sakai A* variant

Sakai A* variant [48] stands out for its simplicity of implementation and modification. Its accuracy is verified by calculating the distance between the start and end points. Analyzing the execution time, this variant presents a fast resolution of the best path, regardless of the need to visualize the results, with execution times in the order of hundredths or thousandths of a second.

Although it has a reduced number of functions defined in the code, Sakai A* variant has proven effective in finding the ideal solution. The visual representation of the best path and the grid of analyzed points facilitates the interpretation of the results, creating a comparative environment between the verified points. In both tested maps, Sakai A* variant achieved excellent verification results.

During the tests, the possibility of adjusting the dimensions of the robot that follows the path was essential to bring the responses closer to the desired scenario. As the robot's size increased, the movements on the x and y axes were better verified, restricting the

code to consider fewer diagonal trajectories. This behavior was even more evident when using map II since, with the larger robot, the axes change became more precise, resulting in fewer points calculated diagonally than the previous map. These results represent an essential factor to be considered in selecting the best variant of the A* algorithm.

4.2 Analysis results of Mehdi A* variant

Compared to the Mehdi A* variant, the Sakai A* variant algorithm contains fewer defined functions, making it simpler. These, along with the heuristic calculation method, make the Sakai A* variant easier to adapt to different scenarios. When testing both variants with random x and y positions, it was discovered that the Mehdi A* variant fails to calculate negative positions on one of the axes, such as -5 on the y-axis. The negative positions were changed to positive to overcome this limitation, allowing for a correct analysis. However, the Mehdi A* variant ignores the points positioned before zero when forming a point cloud for path analysis.

The detailed analysis of the points in the Sakai A* variant also indicated a quick and objective calculation of the positions on the grid, prioritizing the nodes with the highest probability of belonging to the optimal path. This prioritization is observed in the agility with which the algorithm quickly compares the surrounding points, standing out in comparison to the Mehdi A* variant. The total execution time of the Sakai A* variant is mainly used for graphic animation (99.4%), while the algorithm creation takes only 0.6% of the time. Therefore, it is the fastest algorithm for calculations without graphics, with the best path calculated in milliseconds. Even when generating graphs, the increase in time is minimal, whereas the Mehdi A* variant requires considerably more time to execute the algorithm without graphical animation; creating graphs increases the time to tenths of a second or even seconds. These values are low compared to algorithms that require more time to perform complex calculations, and this speed becomes a characteristic that sets it apart from others.

Defining the grid size and the robot in the algorithm improves the results, allowing

the configuration of dimensions similar to the robot used in the Robot@Factory competition, or in real cases of logistical warehouses. This facilitates the simulation of how the algorithm behaves in an environment with pre-programmed displacement points and less freedom of movement. The Sakai A* variant showed better results in practical tests, with minimal variation in the time to find the best solution, regardless of the creation of graphic animations.

A possible improvement for the Sakai A* variant would be to enhance the definition of obstacles on the map, as currently each obstacle must be defined individually, increasing the computational load. This means that there is a possibility for the algorithm to take longer to execute, especially on larger maps with multiple obstacles. The amount of information, although useful, can make it difficult for the user to inspect the search points.

Among the tests conducted on Map I, due to the Mehdi A* variant not reading locations at zero or below zero (negative) on the maps, only test #3 maintained the same positions in both variants. With this, it was possible to compare this point in both variants, showing that, according to Table 4.1, the Sakai A* variant achieved the best path result in less time than the Mehdi A* variant.

Number test	**Sakai Variant Execution time with graphics (s)**	**Mehdi Variant Execution time with graphics (s)**	**Sakai Variant Execution time without graphics (s)**	**Mehdi Variant Execution time without graphics (s)**
3	0.9222	1.0012	0.0170	0.0320

Table 4.1: Variants comparison on Map I

Já no Mapa II, como a localização de todos os armazéns de entrada e saída são as mesmas para as duas variantes, quatro dos testes foram escolhidos aleatoriamente para efetuar o comparativo do tempo de execução para ambas as variantes, conforme a Tabela 4.2. Podemos ver então que, para o Mapa II, a Sakai A* variant também encontrou o melhor caminho em menos tempo que a Mehdi A* variant.

Mehdi A* Variant stands out for its additional functions and visual differentiation

Point of the test	**Sakai Variant Execution time with graphics (s)**	**MehdiVariant Execution time with graphics (s)**	**Sakai Variant Execution time without graphics (s)**	**Mehdi Variant Execution time without graphics (s)**
1 - 1	0.6159	1.4987	0.0030	0.0525
1 - 4	0.8459	1.4543	0.0050	0.0660
2 - 2	0.6152	0.7176	0.0040	0.0230
3 - 1	0.4606	0.8481	0.0020	0.0305
4 - 4	0.6299	1.5956	0.0040	0.0575

Table 4.2: Variants comparison on Map II

between the open and closed lists, which facilitate verifying the analyzed points. However, these extra functions increase the execution time, especially on larger maps with more obstacles. In addition, the complexity of modifications makes Mehdi A* variant less practical for quick adaptations during competitions or scenarios in which it is necessary to change some aspects. The advanced heuristics of this variant, incorporating calculations of jump points and dynamic weights, are advantageous for more precise inspections on large maps. The direct relationship between execution time and path length observed in Sakai A* variant does not occur in variant II, presenting differences when comparing time and length data and not allowing changing the robot's radius, leading most of its movements to be diagonal. Sakai A* variant corrects this limitation by restricting movements within the grid and improving calculation accuracy by changing the robot's size.

In short, Mehdi A* variant is superior in complex and large-scale scenarios, but the simplicity and agility of Sakai A* variant make it more practical for quick adaptations and scenarios with less detailed requirements, such as simple path planning tasks or scenarios with fewer obstacles.

Chapter 5

Conclusions

After all the analyses, the Sakai A* variant algorithm proved to be more efficient for use in warehouse scenarios, as it is simpler to modify, has fewer steps in its composition, which influences the decision, if necessary, during the competition stages, and has a much faster response time for the best solution compared to the Mehdi A* variant algorithm.

It is important to note that the algorithm's response time strongly depends on the device's computing settings. Therefore, its role in ensuring the use of a computer with a fast processor is fundamental, as it significantly influences the obtained result.

Keeping this in mind, it is worth noting that, for the perfect implementation of the algorithm in the robot that will be the object of the competition, the following steps must be followed. First, a complete and accurate definition of all the pre-established positions of the arcs and machines must be made. Next, the movements that the robot will make must be defined, using Python algorithms similar to those that were analyzed. Next, the time required for the robot to move and turn (90^{o} and 180^{o}) must be verified, as this factor is critical to ensure it makes the best and fastest journey possible. The influence of the rotation time and the speed at which it moves forward and backward can possibly cause it to complete the journey more slowly due to the low speed at which it moves backward. Therefore, when entering this information in the future, it is imperative to analyze all aspects and functions that will be included so that the result presented by the robot within the competitive map is satisfactory.

The next steps to be taken are: the implementation of the algorithm for the robot with the exact positions and measurements of the factory floor, inserting functions that allow the robot to move and rotate in the desired directions, at 90^{o}, 180^{o}, or other necessary degrees of rotation, defining the robot's movement speed to advance and retreat, and the time it will take to pass through possible intermediate points along the route, in case it is necessary for the product to perform various stages passing through other machines before reaching the final warehouse.

Bibliography

[1] A. Royston, *Henry Ford and the Assembly Line*. The Rosen Publishing Group, Inc, 2015.

[2] P. B. e Silva, "Collective behavior on multi-agent robotic systems using virtual sensors," M.S. thesis, PUC-Rio, 2012.

[3] M. Čech, P. Wicher, R. Lenort, *et al.*, "Autonomous mobile robot technology for supplying assembly lines in the automotive industry," *Acta logistica*, vol. 7, no. 2, pp. 103–109, 2020.

[4] M. B. Alatise and G. P. Hancke, "A review on challenges of autonomous mobile robot and sensor fusion methods," *IEEE Access*, vol. 8, pp. 39 830–39 846, 2020.

[5] T. Muller, "Automated guided vehicles," *IFS Ltd., Kempston England*, 1983.

[6] G. Fragapane, R. De Koster, F. Sgarbossa, and J. O. Strandhagen, "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda," *European Journal of Operational Research*, vol. 294, no. 2, pp. 405–426, 2021. DOI: https://doi.org/10.1016/j.ejor.2021.01.019.

[7] K. R. Baker, *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[8] M. Zweben and M. Fox, *Intelligent scheduling*. Morgan Kaufmann Publishers Inc., 1994.

[9] N. Sadeh, "Micro-opportunistic scheduling: The micro-boss factory scheduler," Carnegie-Mellon Univ Pittsburgh PA Robotics Inst, Tech. Rep., 1994.

[10] R. Chase, N. J. Aquilano, and S. d'Espiney, *Gestão da produção e das operações: perspectiva do ciclo de vida*. 1995, Monitor, Lisboa.

[11] J. Reis, *Uma introdução ao scheduling*, http://hdl.handle.net/10071/169, 2006.

[12] S. C. Graves, "A review of production scheduling," *Operations research*, vol. 29, no. 4, pp. 646–675, 1981.

[13] J. Mohan, K. Lanka, and A. N. Rao, "A review of dynamic job shop scheduling techniques," *Procedia Manufacturing*, vol. 30, pp. 34–39, 2019. DOI: `https://doi.org/10.1016/j.promfg.2019.02.006`.

[14] V. S. Roldão, *Planeamento e programação da produção*, Monitor, Lisboa, 1995.

[15] İ. Kabasakal, F. D. Keskin, K. Ventura, and H. Soyuer, "From mass customization to product personalization in automotive industry: Potentials of industry 4.0," *Journal of Management Marketing and Logistics*, vol. 4, no. 3, pp. 244–250, 2017.

[16] L. Zhang, Y. Hu, and Y. Guan, "Research on hybrid-load AGV dispatching problem for mixed-model automobile assembly line," *Procedia CIRP*, vol. 81, pp. 1059–1064, 2019.

[17] T. Le-Anh and M. De Koster, "A review of design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 171, no. 1, pp. 1–23, 2006.

[18] P. Mattaboni, "Autonomous mobile robot," Tech. Rep., 1987.

[19] S. Jeon and J. Lee, "Performance analysis of scheduling multiple robots for hospital logistics," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, IEEE, 2017, pp. 937–940.

[20] M. S. A. Mahmud, M. S. Z. Abidin, and Z. Mohamed, "Solving an zagricultural robot routing problem with binary particle swarm optimization and a genetic algorithm," *International Journal of Mechanical Engineering and Robotics Research*, vol. 7, no. 5, pp. 521–527, 2018.

[21] R. Alves, J. S. de Morais, and K. Yamanaka, "Speeding up on-line route scheduling for an autonomous robot through pre-built paths," *Robotica*, vol. 41, no. 1, pp. 3–15, 2023.

[22] S. Zhang, D. Zhuge, Z. Tan, and L. Zhen, "Order picking optimization in a robotic mobile fulfillment system," *Expert Systems with Applications*, vol. 209, p. 118 338, 2022.

[23] M. Eisen, S. Sudhakaran, V. Mageshkumar, A. Baxi, and D. Cavalcanti, "Joint Resource Scheduling for AMR Navigation Over Wireless Edge Networks," *IEEE Open Journal of Vehicular Technology*, vol. 4, pp. 36–47, 2022.

[24] L. Hidri, K. Alqahtani, A. Gazdar, and A. Badwelan, "Integrated scheduling of tasks and preventive maintenance periods in a parallel machine environment with single robot server," *IEEE Access*, vol. 9, pp. 74 454–74 470, 2021.

[25] J.-H. Lee and J.-M. Park, "User-centric real time service scheduling for robots," in *2011 IEEE International Conference on Robotics and Biomimetics*, IEEE, 2011, pp. 1024–1028.

[26] Z. Wang and M. Gombolay, "Learning scheduling policies for multi-robot coordination with graph attention networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4509–4516, 2020.

[27] P. L. Mareddy, S. R. Narapureddy, V. R. Dwivedula, and P. R. Karanam, "Development of scheduling methodology in a multi-machine flexible manufacturing system without tool delay employing flower pollination algorithm," *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105 275, 2022.

[28] R. Maffei, S. Botelho, L. Silveira, *et al.*, "Space d*: Um algoritmo para path-planning multi-robôs," *Proceedings of the VIII ENIA/CSBC*, pp. 607–618, 2011.

[29] H. I. Kang, B. Lee, and K. Kim, "Path planning algorithm using the particle swarm optimization and the improved dijkstra algorithm," in *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, IEEE, vol. 2, 2008, pp. 1002–1004.

[30] H. Wang, Y. Yu, and Q. Yuan, "Application of dijkstra algorithm in robot path-planning," in *2011 second international conference on mechanic automation and control engineering*, IEEE, 2011, pp. 1067–1069.

[31] J. D. Ullman, "The performance of a memory allocation algorithm," *NSF Grant GJ-1052*, 1971.

[32] A. Habibi, "New approximation method for structural optimization," *Journal of computing in civil engineering*, vol. 26, no. 2, pp. 236–247, 2012.

[33] F. Dexter and R. D. Traub, "How to schedule elective surgical cases into specific operating rooms to maximize the efficiency of use of operating room time," *Anesthesia & Analgesia*, vol. 94, no. 4, pp. 933–942, 2002.

[34] J. K. Lenstra, D. B. Shmoys, and É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical programming*, vol. 46, pp. 259–271, 1990.

[35] D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems theoretical and practical results," *Journal of the ACM (JACM)*, vol. 34, no. 1, pp. 144–162, 1987.

[36] A. Stentz *et al.*, "The focussed dˆ* algorithm for real-time replanning," in *IJCAI*, vol. 95, 1995, pp. 1652–1659.

[37] W. Yodianto, H. L. H. S. Warnars, L. L. H. S. Warnars, A. Ramadhan, T. Siswanto, *et al.*, "Searching routing using a-star (a*) search algorithm," in *2024 3rd International Conference on Creative Communication and Innovative Technology (ICCIT)*, IEEE, 2024, pp. 1–7. DOI: http://doi.org/10.1109/ICCIT62134.2024.10701177.

[38] J. Backus, "The history of Fortran I, II, and III," *ACM Sigplan Notices*, vol. 13, no. 8, pp. 165–180, 1978.

[39] L. Tulchak and. rchuk, "History of python," Ph.D. dissertation, 2016.

[40] M. Hoffmann, M. Scherer, T. Hempel, *et al.*, "Deeptime: A python library for machine learning dynamical models from time series data," *Machine Learning: Science and Technology*, vol. 3, no. 1, p. 015 009, 2021.

[41] C. R. Harris, K. J. Millman, S. J. Van Der Walt, *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[42] S. Gholizadeh, "Top popular python libraries in research," *Authorea Preprints*, 2022.

[43] S. Morley, *Applying Math with Python: Over 70 practical recipes for solving real-world computational math problems*. Packt Publishing Ltd, 2022.

[44] *National Robotics Festival - Major Competitions Rules*. [Online]. Available: `https://www.festivalnacionalrobotica.pt/2024-1/?page_id=1228`.

[45] G. Dong, F. Yang, K.-L. Tsui, and C. Zou, "Active balancing of lithium-ion batteries using graph theory and a-star search algorithm," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2587–2599, 2020.

[46] A. F. Pereira, *Uma introdução ao algoritmo a**. [Online]. Available: `https://sweet.ua.pt/antoniop/SearchAlgorithms/pathfinding/a-star/introduction.html`.

[47] S. Erke, D. Bin, N. Yiming, Z. Qi, X. Liang, and Z. Dawei, "An improved a-star based path planning algorithm for autonomous land vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 5, p. 1 729 881 420 962 263, 2020.

[48] A. Sakai and N. Kanargias, *A* grid planning*. [Online]. Available: `https://github.com/AtsushiSakai/PythonRobotics/blob/master/PathPlanning/AStar/a_star.py`.

[49] S. Mehdi, *A* variants code*. [Online]. Available: `https://github.com/AtsushiSakai/PythonRobotics/blob/master/PathPlanning/AStar/a_star_variants.py`.