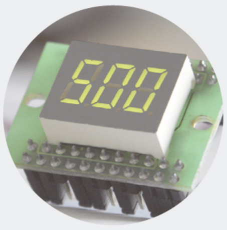
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Implementação e Teste de Algoritmos de Planeamento e Escalonamento para Frotas AGV

MÁRCIA MARTINS COSTA

julho de 2018

POLITÉCNICO
DO PORTO

Implementation and Testing of Planning and Scheduling Algorithms for AGV Fleets

Márcia Martins Costa

Supervisors
Manuel Fernando dos Santos Silva
Pedro Luís Cerqueira Gomes da Costa

Mestrado em Engenharia Eletrotécnica e de Computadores
Instituto Superior de Engenharia do Porto
Departamento de Engenharia Electrotécnica
Rua Dr. António Bernardino de Almeida, 431, P-4200-072 Porto
July 2018

Resumo

Os Automatic Guided Vehicles (AGV) são veículos de transporte sem condutor e a sua incorporação nas empresas, para transporte de materiais e apoio à produção, está a ser cada vez mais frequente, pois cada vez mais se pretende ter eficácia e eficiência na logística das fábricas e na entrega da quantidade necessária de materiais na produção, na altura certa e pela ordem correta, obtendo-se maiores lucros e rendimentos de produção nas indústrias. Recorrendo apenas a colaboradores e transportadores, como empilhadores, é provocado na fábrica bastante tráfego e muitas das vezes ocorrem velocidades elevadas dos transportadores. Isto pode causar problemas, tais como, danificação de materiais (levando a desperdício e à necessidade de produzir novos materiais) e acidentes. Recorrendo aos AGV é possível evitar estes problemas, uma vez que conseguem transportar os materiais com fiabilidade, trabalhar junto dos colaboradores em segurança e, ainda, garantir uma cadência de ritmo das suas tarefas.

O objetivo deste trabalho foi implementar e testar algoritmos de planeamento de trajetórias e escalonamento de tarefas para frotas de AGV. Neste sentido, foram criados *layouts* fabris onde os AGV percorrem caminhos com menor custo e realizam as tarefas pretendidas. Os vários modelos foram analisados e foram também apresentadas as características de cada um.

Este projeto proporcionou a oportunidade de aprofundar conhecimentos na área da modelação e simulação de AGV, e em algoritmos de planeamento de trajetórias e escalonamento de tarefas.

Palavras-Chave: AGV, Simulação, Algoritmos de planeamento de trajetórias e Algoritmos de escalonamento de tarefas.

Abstract

AGV are driverless vehicles for transportation and their incorporation in companies, for transportation of materials and production support, is becoming more frequent, because more and more is intended to have effectiveness and efficiency in the factory logistics and the delivery of the quantity of materials in production, at the right time and in the right order, resulting in higher profits and production yields for industrial companies. Thus, using only employees and transportation vehicles, such as forklifts, there is a lot of traffic in the factory floor and many times high vehicles speeds. This can cause problems, such as, damage to materials (leading to waste and the need to produce new materials) and accidents. Using AGV, is possible to avoid these problems, since they can transport the materials with reliability, work safely with the employees and still ensure a cadence of their tasks.

The objective of this work was the development and testing of models that simulate factory layouts with AGV fleets. For that purpose were implemented planning and scheduling algorithms for controlling the displacement and task allocation for the different vehicles. The implemented models were analyzed and its main characteristics were also presented.

This project provided the opportunity to deepen knowledge in the area of AGV modeling and simulation, and in trajectory planning and scheduling algorithms.

Keywords: AGV, Simulation, Trajectory planning algorithms and Task scheduling algorithms.

Summary

List of Figures

List of Tables

Glossary

Abbreviation	Description	Page
AGV	Automatic Guided Vehicles	iii
ISEP	Instituto Superior de Engenharia do Porto	xi
VC	Visual Components	2
3D	Three Dimensional	5
CAD	Computer-Aided Design	5
IDE	Integrated Development Environment	6
API	Application Programming Interface	6
ROS	Robot Operating System	6
VR	Virtual Reality	7
2D	Two Dimensional	7
HHLA	Hamburger Hafen und Logistik AG	8
CTA	Container Terminal Altenwerder	8
A*	A star	17
TEA*	Time Enhanced A*	18
RRT	Rapidly Exploring Random Tree	20
RDT	Rapidly Exploring Dense Trees	20
GA	Genetic Algorithms	23
ID	Identify	34

Acknowledgements

This project would not have been possible without the collaboration of those which I will now refer. To all my sincere thanks.

To Professor Doctor Manuel Fernando dos Santos Silva and to Professor Doctor Pedro Luís Cerqueira Gomes da Costa, advisor and co advisor, respectively, of this project, a very special thanks for all the support and encouragement they have given me, and for having infected me with their good disposition at work. Thank you, yet, for having played the role, which I think is the one that is expected of the advisores, that is, they were a “ pillar ”.

To Eng. Fernando Rocha for having made some of his time available to explain me Python and for helping in some problems I had during the development of the algorithms programming.

To Instituto Superior de Engenharia do Porto (ISEP) for making available study workplaces and having accompanied all my academic journey.

Chapter 1

Introduction

This chapter presents the project contextualization, objectives, planning, and, finally, the structure of this document.

1.1 Contextualization

AGV are transport vehicles without driver, *i.e.*, they are mobile robots that can follow markers, wires on the ground, use vision, magnets or lasers to carry out their routes. They can safely transport various types of products without human intervention, within production, logistics, warehouse and distribution environments, thereby reducing costs and increasing the efficiency and profitability of a company.

With this project it was intend to model, analyze and test, through 3D simulation, planning and scheduling algorithms for AGV fleets, trying to determine their relative advantages and disadvantages, taking into account several aspects, such as avoid deadlocks, minimize the number of AGV required and minimize response times.

1.2 Motivation

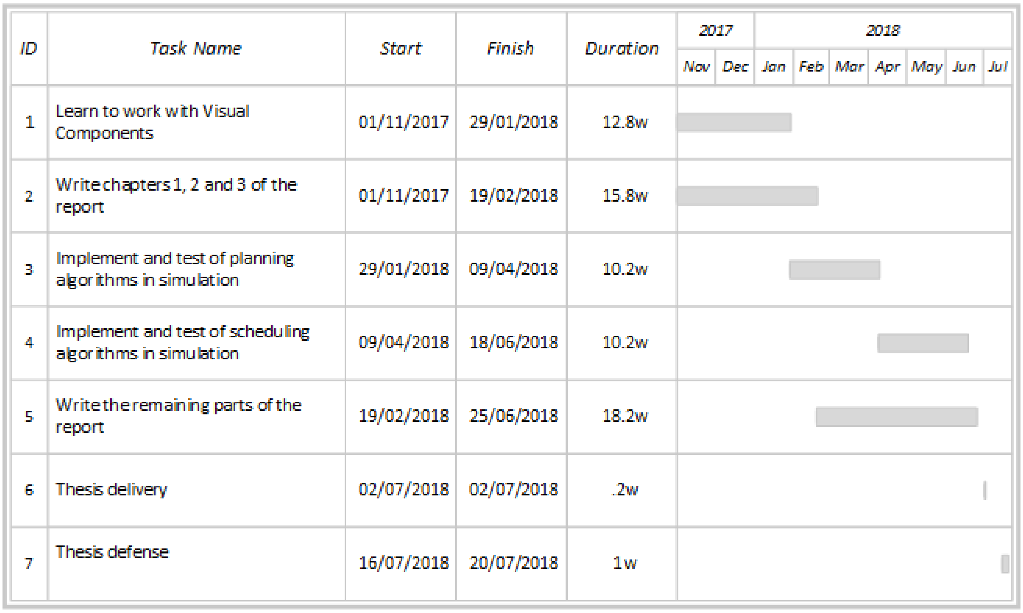
The motivation to carry out this project had two aspects: (*i*) the study of AGV fleets and its trajectory planning and scheduling algorithms; and (*ii*) an acquisition of skills in the area of industrial processes simulation.

1.3 Objectives

The main objective was to apply fleet planning and scheduling algorithms in a simulation environment, in this case using the Visual Components (VC) software, trying to determine the advantages and disadvantages related to the mentioned algorithms, taking into account aspects as, avoid deadlocks, minimize the number of AGV required and minimize response times.

1.4 Work Plan

The Gantt chart depicted in Figure 1.1 summarizes the work plan for this project.



ID	Task Name	Start	Finish	Duration
1	Learn to work with Visual Components	01/11/2017	29/01/2018	12.8w
2	Write chapters 1, 2 and 3 of the report	01/11/2017	19/02/2018	15.8w
3	Implement and test of planning algorithms in simulation	29/01/2018	09/04/2018	10.2w
4	Implement and test of scheduling algorithms in simulation	09/04/2018	18/06/2018	10.2w
5	Write the remaining parts of the report	19/02/2018	25/06/2018	18.2w
6	Thesis delivery	02/07/2018	02/07/2018	.2w
7	Thesis defense	16/07/2018	20/07/2018	1w

Figure 1.1: MSc. Thesis work plan Gantt chart

The project will begin with the writing of chapters 1, 2 and 3 of the report and with the learning of the operation of VC software. The learning of the simulator will be based on a document of initiation and through small files of examples available in the VC forum [13, 14]. Learning to work with VC will take about thirteen weeks and writing about sixteen weeks. The next step will be to implement and test the trajectory planning algorithms for approximately ten weeks. After completing the previous task, the work will proceed to the implementation and testing of the task scheduling algorithms, which will last ten weeks. The implementation and testing of the algorithms will be largely accompanied by the writing of the full report. It is estimated that the thesis will be delivered on July 2, 2018 and its presentation and discussion in the week of July 16 to 20, 2018.

1.5 Structure of the Dissertation

This document is structured in six chapters.

In Chapter 1 is made an “Introduction” to the work developed. The contextualization, motivation, objectives and task plan of this project are discussed.

Chapter 2 is devoted to “Simulation Environments and AGV” and presents simulation software for mobile robots, namely its main features, advantages and disadvantages, and navigation methods and steering control of AGV.

Chapter 3 describes the “Planning and Scheduling Algorithms for AGV”, in particular detailing the ones that were later implemented and tested during the work.

Chapter 4 introduces the “Work developed” in that it allows the reader to know the problem and the proposed solution to develop the project, the architecture of the system and, finally, the practical implementation of the algorithms.

In Chapter 5, named “Tests and Results”, are mentioned which tests were performed, the results obtained and if these are in accordance with what was expected to be obtained.

Finally, on Chapter 6 (“Conclusions”) are analyzed the results of the work developed and presented suggestions for improvements that can be performed to the project in the future.

Chapter 2

Simulation Environments and AGV

In this chapter a reference is made to the simulation software used, as well as to other examples of simulators for mobile robots. Here are described types, advantages, disadvantages, navigation methods and steering control of the AGV.

2.1 Simulation Applications

In this section will be mentioned some examples of software applications that make it possible to perform simulations of industrial environments which use mobile or static robots.

2.1.1 Visual Components

Visual Components software allows to create, configure, and simulate three dimensional (3D) factory layouts, program in Python language or Dot.NET, view, edit component properties, as well model them and create components from scratch. It is compatible with Computer-Aided Design (CAD) and contains ready-to-use component/equipment library. It has a forum where users of this software can place questions and get answers from other users, and where they can share small projects they have developed. Additionally, it has a support service that allows users to place questions/doubts or improvement suggestions, via email, directly to the software developers and get answers [15].

This software contains many simulation resources that may help companies: (*i*) in its sales, since it allows to create and present 3D layouts according to the needs/requirements of the clients so that these can visualize the whole process before buying some equipment or before implementing a new manufacturing process [16]; and (*ii*) in its management, since, for instance, before changing the layout or a production processes, it is possible to test the changes in VC to detect problems and analyze results, thus helping to reduce the downtime of manufacturing companies and to accelerate the implementation of new processes, because if the simulation accomplishes the requirements, the implementation becomes fast, for it is enough to implement in reality what was simulated [17].

It should be noted that VC contains several resources for simulation with industrial robot arms, but there is still little information, and few resources, for simulation with mobile robots, since this functionality is in an early stage of development.

2.1.2 Webots

Webots was developed by the Swiss Federal Institute of Technology in Lausanne and is a development environment used to model, program and simulate mobile robots. The user is allowed to design robotic configurations with one or several, similar or different, robots in a shared environment, choose the properties of each object (shape, color, texture, friction, among others) and choose sensors and actuators for each robot. Robot controllers can be programmed with the Integrated Development Environment (IDE) or third-party development environments. The behavior of the robot can be tested in worlds physically very upcoming to reality. Controller programs can arbitrarily be transferred to real robots [18].

This simulator runs on Windows, Mac and Linux. Both global files and Application Programming Interface (API) functions are cross-platform, since they can be shared by people using different operating systems. Just as Visual Components, it contains a forum and a support service and a library of robots, sensors and other objects. It provides online a user guide, a reference manual and a robot curriculum. Programming can be done in C, C++, Java, Python or Matlab. It provides dynamic simulation, interface with Matlab and Robot Operating System (ROS) and has an interactive 3D simulation window [19].

2.1.3 V-REP

The V-REP simulator, used for simulations of factory automation, development of robot prototypes, among others, incorporates an IDE and is based on a distributed control architecture, that is, each object/model can be controlled individually through an embedded script, a plugin, a ROS node, a remote API client, or a custom solution. This makes this simulator suitable for multi-robot applications. Controllers can be written in C/C ++, Python, Java, Lua, Matlab or Octave. Such as the previously discussed simulators, it also contains a component/equipment library and a forum [20, 21, 22].

2.1.4 FlexSim

FlexSim software enables performing simulations in 3D, optimize industry systems and, like the previous simulators, contains an equipment/component library and a forum. It has a FlexSim AGV simulation software module that is used to model systems that use AGV [23, 24, 25].

This software has Virtual Reality (VR) resources, containing demo templates that are compatible with Oculus Rift, HTC Vive and OpenVR technologies. Two of the models use touch controllers to add more realism to the models [26]. Finally, it is still possible to have the resources to run simulations in the area of health care. According to the case studies, it was possible to [27]: (*i*) redesign the general stores department of an hospital, with the purpose of obtaining more space in the supply reception area and avoiding congestion of pallets in the corridor that hinder the movement of employees; (*ii*) manage a hemorrhagic emergency in labour, in order to reduce the time of ordering and administration of blood supplies and medication; (*iii*) help determine optimal beds capacity for an outpatient surgery center to improve patient workflow and experience; (*iv*) obtain data on patient assistance, including arrivals and discharges and use of the staff utilization; (*v*) study the large-scale rapid transport of hospitalized children, including estimate travel times and develop contingency plans (evacuation priorities, recovery analysis and prevention/mitigation); and (*vi*) help in the preparation of a possible emergency evacuation in an hospital facilities, with the purpose of evacuating in the best way and in safety. With the simulation it is tried to predict the evacuation time taking into account the different time periods of the day.

2.1.5 Simio

The Simio software provides an object-based 3D modeling environment which lets construct 3D models from a top-down (2D) view, and then instantly switch to a 3D view of the system, that is, from a more general 2D to a more detailed (3D) view [28]. It also provides a detailed production schedule that records performance measures that encompass the probability of reaching a target (e.g. due date), the expected milestone completion date, and the optimistic and pessimistic completion times (percentile estimates). The goals show the probability of the tasks being met and codify the tasks with colors based on the level of risk, as can be seen in Figure 2.1. In this figure it is possible to verify that jobs 3 and 4 are in order, while job 2 is at high risk and job 1 is at moderate risk [1].

All Simio model-building products directly integrate with Google Warehouse to allow the quickly download from a massive library of freely available 3D symbols, to make models more realistic [28].

Simio also provides a forum and technical support for users [29].

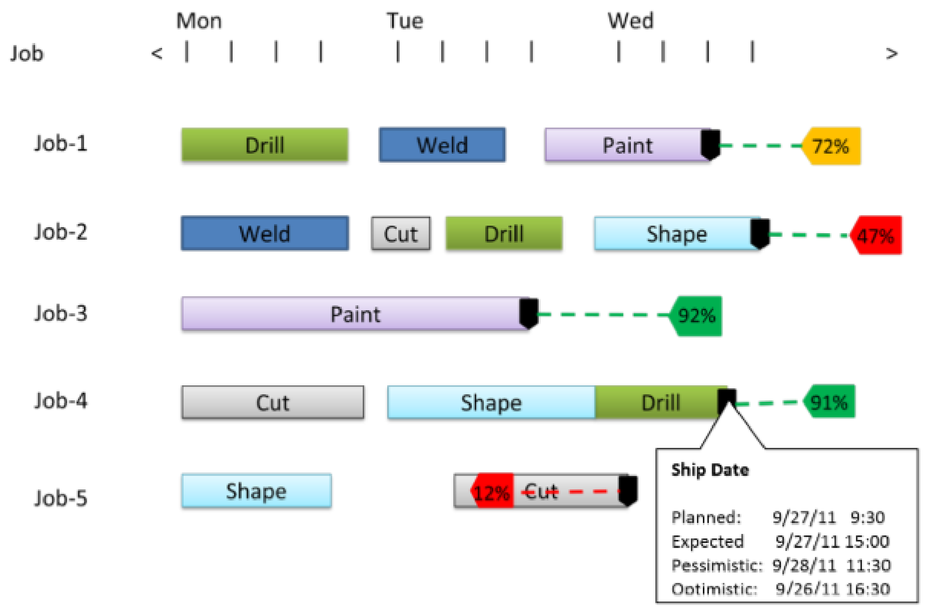


Figure 2.1: Example of a detailed production scheduling Gantt chart [1]

Like VC, all of these simulators are intended to help companies optimize their manufacturing processes and factory layouts (resulting in higher profits), reducing errors in the real world implementation, and minimizing the downtime of production, as well as presenting simulations that allow customers to see if the required requirements by them are fulfilled when these intend to buy some equipment.

2.2 AGV

An AGV is an Automated Guided Vehicle, usually battery-powered, and that has an associated controller to control its fleets or interactions with other AGV. The batteries can be of lead-acid, lithium-ion, among others. Lithium-ion batteries, used in Hamburger Hafen und Logistik AG (HHLA) Container Terminal Altenwerder (CTA), provide an alternative to diesel-electric drive units. Lead-acid batteries also provide that alternative [30].

AGV are used in internal or external environments where it is necessary to transport products in the production, warehouses, containers terminals and external (underground) transport systems. They are programmed to transport materials through defined routes of collection and delivery of products. This type of vehicles emerged as an alternative to the use of forklifts operated by drivers [7, 31]. The AGV can be guided by means of an electric and inductive signal (filo-guided system), magnetic or optical (band and marker system) and laser (trilateration and triangulation system) as it will be described later [32].

2.2.1 Types of AGV

The AGV normally used can be classified into tugs, unit load and forklifts.

The tugs AGV are used to transport one or more wagons along a fixed route with pre-established loading and unloading points, where the wagons can be uncoupled/unattended to the AGV automatically and/or manually, the latter using people. These AGV present high traction and the path curves present minimum radius limitation to be able to transport the wagons. They are well known for transporting parts kits in the automobile industry. The application of this type of AGV in the logistics supply of the production lines becomes more efficient than the systems that use forklifts for the unit transport of products to the points of use [2, 3]. Two examples of this type of AGV can be seen in Figure 2.2.

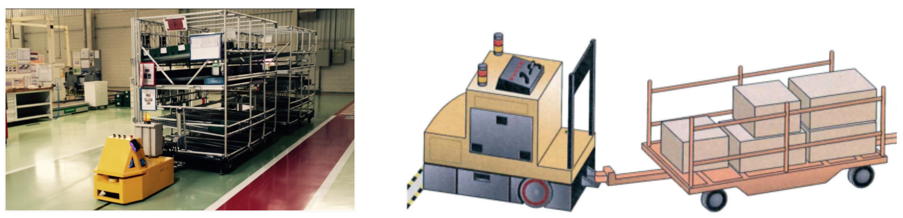


Figure 2.2: Examples of tugs AGV [2, 3]

The unit load AGV have the possibility to move in places with little space for maneuvers, since they are capable of making curves with very small radii or even turning around its own axis. For this type of vehicle to carry a load, it must be on a rolling base or a table type support, since the AGV will enter under the rolling base and will raise it sufficiently so that no point touches the surface of the floor. Afterwards it will start moving. The AGV platform may have an elevator to help lift the material to be loaded. These AGV have application in industry, hospitals and laundries [2, 4]. Two examples of this type of AGV are illustrated in Figure 2.3.

The forklifts AGV operate in the same way as forklifts, with the difference that they do not require a driver. Fork vehicles are the most common because they are more flexible. They can interact with unit load AGV and also collect or deposit a load on the floor. These vehicles are designed to move fairly high loads, such as a steel coil or coach bodies, or else fragile products, such as glass plates, or yet different types of pallets or different sizes of boxes [2, 5]. Two examples of this type of AGV are presented in Figure 2.4.

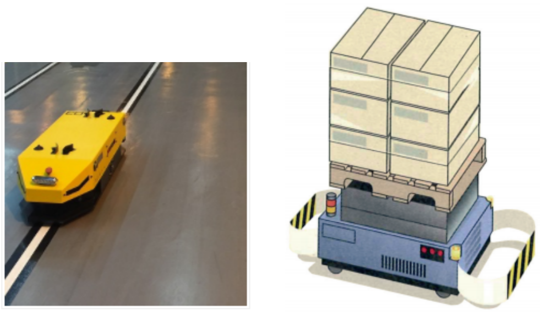


Figure 2.3: Examples of unit load AGV [2, 4]

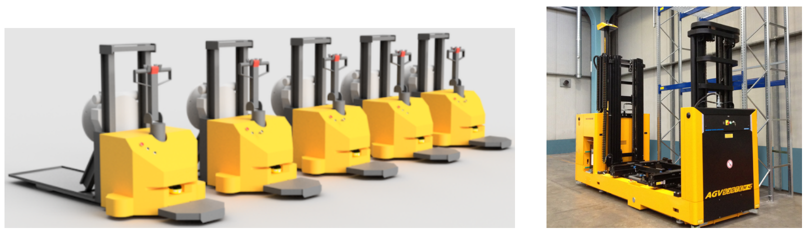


Figure 2.4: Examples of forklifts AGV [5, 6]

2.2.2 Advantages

The use of AGV has benefits, such as [7, 33]:

- Flexibility of adaptation - these vehicles can be inserted in factories, requiring little or no modification in the structure of the factory floor. They allow quick and easy adjustment to changes in the operating environment and changes in the transport sequence of the materials and can be customized to facilitate their adaptation to the particular functions of each type of factory.

- Efficiency - the use of AGV makes the operations more efficient, since depending on the technology involved in the AGV development, these vehicles can be monitored and controlled, which allows to locate the products at any time and, if needed to collect products for manufacturing or deliveries, the vehicle is never lost.

- Reduced costs - AGV reduce operating costs, because they can work continuously (without interruptions) 24 hours a day and with little or no supervision, unlike people who need rest and supervision at the job they perform.

- Safety - AGV have cameras, lasers, and other sensors that allow them to operate safely in environments with people and built structures. On the other hand, people-operated equipment, for example, forklift trucks, do not have as many safety mechanisms, since they rely on operators to avoid accidents. However, due to distractions or fatigue, accidents are frequent. With the use of AGV these concerns disappear, since they do not get distracted or tired. In addition, AGV can operate under conditions that humans do not tolerate or in conditions where humans can not work optimally, such as in extreme heat or cold, and/or around hazardous materials. In this way, AGV increase safety and decrease operational downtime, which helps to increase the efficiency and effectiveness of operations and production [33].

The AGV are built according to the European Standard for the safety of self-contained vehicles (EN1525 - Driverless industrial trucks and their systems [34]). The safety devices include: (*i*) collision detection because they have sensors that detect obstacles in their range and calculate the distance to them. In this way, the AGV slows down or stop, in order to avoid collisions. In addition, bumpers can be incorporated which, when triggered, ensure instantaneous stoppage of the vehicle if the obstacle sensor(s) has not been actuated; (*ii*) audio or visual signals due to the existence of signaling devices, such as flashing light signals or audible alerts that indicate the state of the AGV (loaded, unloaded, etc.) or warn workers of their presence; and (*iii*) manual control and emergency buttons to stop the AGV abruptly and instantaneously [7].

2.2.3 Disadvantages

Beyond the benefits presented when using AGV, these vehicles also present some drawbacks, such as [33]:

- Initial investment - is potentially high because buying an AGV will probably be more expensive than hiring employees or using other equipment, for example forklifts. This purchase is also associated with periodic maintenance costs and occasional repairs. Although AGV are not directly operated by people, there is some operational downtime required as people are trained and vehicles are deployed. However, this is not properly a disadvantage, but it can lead to one punctual expenditure.

- Flexibility of operations - is reduced since AGV work according to predefined systems and processes. Thus, it is more advantageous to have collaborators because they can “jump” between tasks, for example in a situation where a collaborator needs to stop what he is doing and replace a colleague in a completely different task. However, collaborators with experience in diverse tasks are necessary, so that they are able to replace or to help where necessary, whereas an AGV is not able of doing this.

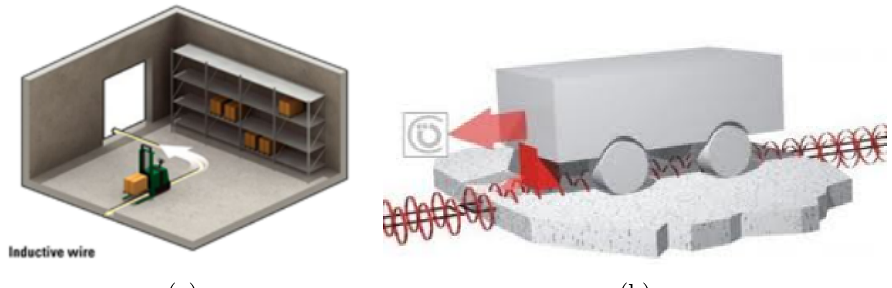
2.2.4 Navigation Methods

There are several navigation methods that encompass fixed or dynamic trajectories that an AGV can travel. The option of implementing fixed or dynamic paths will depend on the installation costs, flexibility requirements and the need or not for future system expansion. Fixed paths are the most used solution in the market, implying lower costs, however have less flexibility preventing layout changes [35].

2.2.4.1 Fixed Trajectories

Fixed trajectories encompass filo-guided and bands systems.

The filo-guided system, presented in Figure 2.5(a), translates into the definition of the AGV path by means of electric conductors incorporated in the ground, through which flows an alternating electric current, as depicted in Figure 2.5(b). These conductors generate a magnetic field due to the sinusoidal current that flows through them. This field is detected by an antenna placed on the AGV. It is a non-flexible system, as it does not allow routes to be changed easily. In order to make changes it is necessary to re-implant conductors on the floor, entailing high costs, which is why it is not used in industries that need to reconfigure the layout many times. However, it is widely used because of its simplicity and robustness [7, 35].



(a) (b)

Figure 2.5: Navigation method (a) and Magnetic field used (b) [7]

The bands system defines the trajectory using a magnetic tape glued to the floor or using lines that are painted on the floor, as shown in Figure 2.6. Its operation is identical to the filo-guided system, incorporating, however, a sensor suitable to detect the tracks. The main advantage of this system in relation to the filo-guided system lies in the routes, since these can be exchanged more easily and quickly when compared with filo-guided systems and with lower costs and in less time, thus becoming a more flexible navigation method. The main disadvantage is that if the tape gets damaged or dirty due to the movement of people or objects on it, the AGV may fail to detect it and, as a result, will not be able to continue the journey.

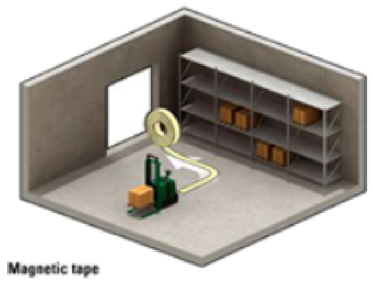


Figure 2.6: Band system [7]

2.2.4.2 Dynamic Trajectories

The dynamic trajectories include the system of trilateration and laser triangulation and the system of markers.

In the system of trilateration and laser triangulation there are emitter or reflector posts or headlights (ultrasound, laser, infrared, color code, among others) in columns, walls and/or other high places and of easy access to the laser installed in the AGV. This laser is usually of the scanner type, so that it can perform rotating scans, as depicted on Figure 2.7. The laser rotates in search of reference points (posts or headlights) that help the AGV to locate itself. However, in order to obtain the location, the vehicle needs to detect at least three of these reference points (advisable five or more), which means that there must be a good/strategic planning in the arrangement of these points, *i.e.*, one must know, a priori, the position of the points, using trilateration techniques, which make it possible to measure distances or differences of distances to points, or triangulation techniques that allow to measure angles between the AGV and the points. The mapped area is stored in the AGV memory. This type of navigation allows a high speed of movement of the vehicle, and is a more flexible, reliable and safe form of navigation that guarantees greater precision in the position of the AGV; however it is the most expensive because it may be necessary to prepare the surrounding AGV work area, if there are no adequate reference points already installed [7, 35].

The marking system is based on marking small magnetic disks spaced from each other on the shop floor, as shown in Figure 2.8. In the case of simpler magnetic markers, it is not possible to know the absolute position of the AGV, however, it is possible to previously store the coordinates of the markers in a database giving to the AGV the information of its location when passing through a marker. The markers may also indicate the next path segment in order to direct the vehicle to the next desired marker. If the AGV deviates from the planned

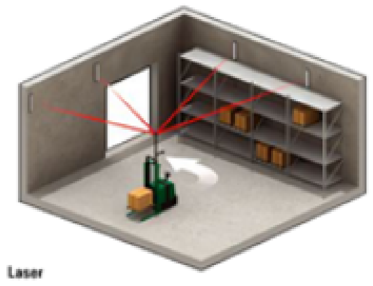


Figure 2.7: Laser triangulation system [7]

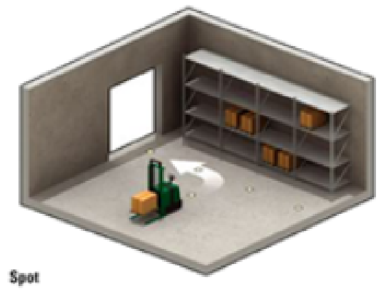


Figure 2.8: Marking system [7]

trajectory by accumulating errors, it will not find the next marker, getting lost. In this way, this system is usually used together with a gyroscope that analyzes the variations in direction. It is a flexible solution allowing routes to be changed easy and quickly. The costs are higher than the magnetic tape solution, but it is the type of dynamic trajectory with the lowest price [7].

2.2.5 Steering Control

The direction control depends of the type of wheels used and their geometric configuration on the AGV chassis. Some possibilities are shown in Figures 2.9 and 2.10. More alternatives can be found in [8].

2.3 Conclusion

This chapter described simulation software for mobile robots, including the one used in this project. Types, advantages, disadvantages, navigation methods and control of AGV direction were also described. The objective was to present simulation software for mobile robots and the main AGV characteristics.

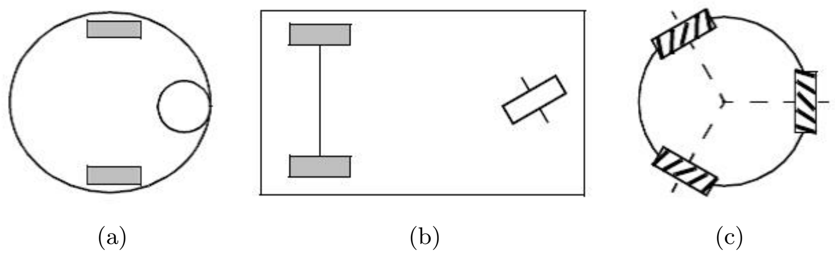


Figure 2.9: Two-wheel centered differential drive with a third point of contact (a), two connected traction wheels (differential) in rear, 1 steered free wheel in front (b) and three motorized Swedish or spherical wheels arranged in a triangle; omnidirectional movement is possible (c) [8]

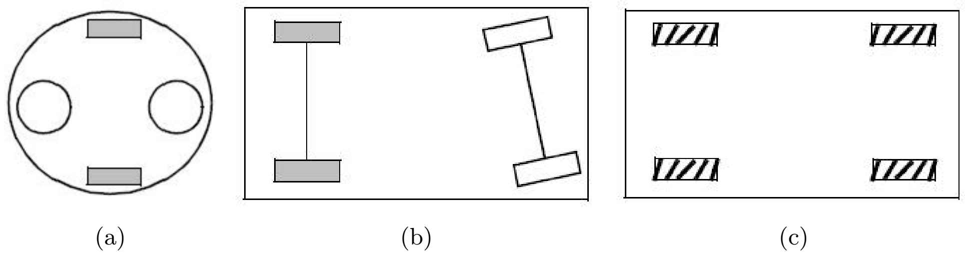


Figure 2.10: Two-wheel differential drive with 2 additional points of contact (a), two motorized wheels in the rear, 2 steered wheels in the front; steering has to be different for the 2 wheels to avoid slipping/skidding (b) and four omnidirectional wheels (c) [8]

Chapter 3

Planning and Scheduling Algorithms for AGV

In this chapter are described planning and scheduling algorithms for AGV fleets. The planning algorithms allow AGV to travel the most efficient path, that is, at a lower cost and without collisions, while the scheduling algorithms allow the AGV to perform the intended tasks in the most adequate order.

3.1 Trajectory Planning Algorithms

The trajectory planning can include aspects such as the planning of movements between obstacles and the coordination of movement with other AGV. Thus, aims at choosing the route that usually takes less time and that presents less costs for the AGV to accomplish the intended tasks.

3.1.1 A star

The A star (A*) search method is an extension of the Dijkstra method; however, as it uses heuristics, it can achieve better performance. This method evaluates the node to be expanded, encompassing the cost of arriving from the initial node to node n ($g(n)$) and the estimated cost of moving from that node to the destination node ($h(n)$ - heuristic), resulting in Expression 3.1, where $f(n)$ is the estimated cost of the best route to the destination node, passing through node n [36, 37]:

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

This search method can be considered optimal if both of the following conditions are satisfied:

- Admissibility - utilization of an allowable (optimistic) heuristic, but h must not overestimate the real cost to reach the solution. Thus, straight line distance can be used, because the shortest distance between two points is a straight line. That is, verify Equation 3.2 [36, 37]:

$$h(n) \leqslant h^*(n), \text{ where } h^*(n) \text{ is the real cost from } n \tag{3.2}$$

- Consistency - according to [36], "... is a necessary condition for the A* method to be optimal, but only if this method is used in graph searches. The heuristic $h(n)$ is consistent if for all n nodes and their successors n' generated by an action a, the estimated cost of the displacement from the n node to the end node is not greater than the cost of moving from the n node to the n' node plus the estimated cost of displacement from node n' to the destination node", as it can be seen in Equation 3.3.

$$h(n) \leqslant c(n, a, n') + h(n') \tag{3.3}$$

3.1.2 Time Enhanced A*

The Time Enhanced A* (TEA*) algorithm consists of an incremental algorithm that builds the path of each vehicle taking into account the movements of other AGV. This feature allows the algorithm to produce conflict free routes and, at the same time, deal with deadlock situations, since the paths are constantly recalculated and the map information is updated at each iteration, that is, the unpredictable events are considered in the input map allowing to avoid the main challenges of any multi-AGV approach such as collisions and deadlocks [9].

The algorithm in question is based on the A* heuristic search, previously mentioned, however it contains an additional component, the time. This component is important to allow a better prediction of the vehicles' movements during run time. Each node on the map has three dimensions: the cartesian coordinates (x, y) and a representation of the discrete time. The time is represented through temporal layers, $k = [0; T_{Max}]$, on which T_{Max} represents the maximum number of layers. Each temporal graph is composed of a set of free and occupied/obstacles nodes, as it can be can seen in Figure 3.1 [9].

The path for each AGV is calculated during the temporal layers. In each temporal layer, the position of each vehicle is known and shared with the other vehicles. In this way, it is possible to detect future possible collisions and avoid them at the beginning of the paths' calculation. Each AGV can only start and stop in nodes and a node can only be occupied by one vehicle on each temporal layer [9].

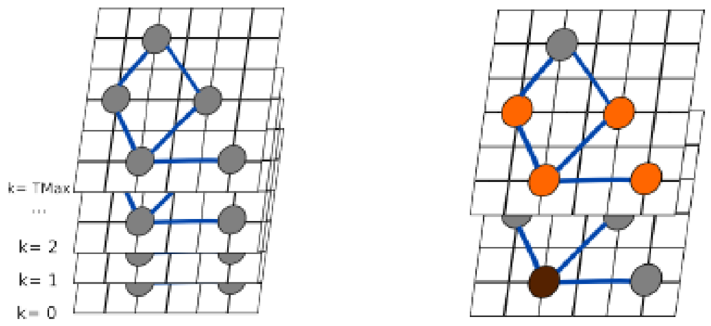


Figure 3.1: Temporal graphs (left) and AGV positions in each graph (right) [9]

The operation of the TEA* algorithm is similar to A*, as already mentioned, since for each AGV, during the path calculation, the next neighbor node analyzed depends on a cost function, $f(n)$, given by the sum of two terms: the distance to the initial vertex, $g(n)$, and the distance to the end point, $h(n)$. The main difference between these two algorithms is that in TEA* time is considered, resulting in two definitions, according to [9]:

- "Definition 1: The neighbor vertices of a vertex j in the temporal layer k belong to the next temporal layer given by $k + 1$" (Figure 3.1), that is, the total number of temporal layers depends of the number of iterations required to reach the intended destination. The more complex the map, the more iterations are needed.

- "Definition 2: The neighbor vertices of vertex j (v^j_{adj}) include the vertex containing the AGV current position, and all adjacent vertices in the next time component.", that is, the set of neighboring nodes includes not only the adjacent nodes, but also the node corresponding to the position in analysis. This property allows a vehicle to maintain its position between consecutive time instants if any neighbour node is free, as can be seen in Figure 3.1. In this case, it is not considered a zero value for the euclidean distance; instead a constant heuristic value corresponding to the stopped movement is assigned.

In Figure 3.2 is depicted the control diagram of the TEA* operation, for each iteration of the algorithm in a multi-AGV context. The initial positions of the AGV are placed as obstacles, in the first time layer ($k = 0$), allowing a vehicle to consider the positions of the other vehicles as nodes occupied. In order to avoid

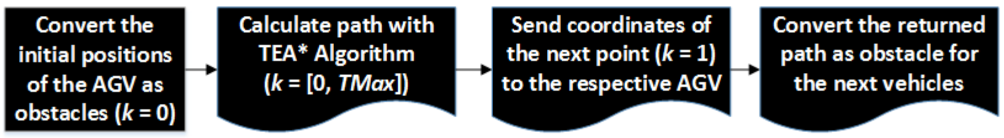

Convert the initial positions of the AGV as obstacles (k = 0)
Calculate path with TEA* Algorithm (k = [0, TMax])
Send coordinates of the next point (k = 1) to the respective AGV
Convert the returned path as obstacle for the next vehicles

Figure 3.2: Control diagram for each algorithm iteration in a Multi-AGV situation [9]

deadlocks, those nodes are placed as obstacles only in $k = 0$, 1, that is, in the first two time layers. Next, is analyzed what the AGV has to do (missions) and the path for each of the vehicles is calculated using the TEA*. The coordinates of the next node, in the second time layer ($k = 1$), are transmitted to the respective AGV. Before moving to the next mission, the full path is converted as obstacle to the following missions and respective AGV. With this in mind, it is possible to coordinate the vehicles, while avoiding collisions [9].

3.1.3 Rapidly Exploring Dense Trees

This method of trajectory planning follows an approach of sampling and incremental search that allows a good performance of the algorithm without any parameter adjustment. The idea is to incrementally construct a search tree that gradually improves the resolution, without having to explicitly define some resolution parameter. At the limit, the tree covers all space. Thus, it has properties similar to the space filling curves, but instead of a long path, there are shorter paths that are organized into a tree. A dense sequence of samples is used as a guide in incremental tree construction. If it is a random sequence, the resulting tree is called Rapidly Exploring Random Tree (RRT). Typically, this family of trees, whether random or deterministic, will be referred to as Rapidly Exploring Dense Trees (RDT) to indicate that a dense cover of space is obtained [10, 36].

The algorithm exploration begins with the creation of an initial vertex, q_0, which will then result in ramifications, as can be seen in Figure 3.3(a), where there are three edges and four vertices. That is, for k iterations, a tree is created iteratively by connecting $\alpha(i)$ to its nearest point. The connection is usually made along the shortest possible path. At each iteration, $\alpha(i)$ becomes a vertex, resulting a dense tree. If the closest point, q_n, for $\alpha(i)$ is a vertex, as shown in Figure 3.3(b), an edge is constructed from q_n to $\alpha(i)$. However, if the nearest point is inside an edge, as shown in Figure 3.3(c), the existing edge is divided so that q_n appears as a new vertex and an edge is constructed from q_n to $\alpha(i)$. Note that the total number of edges may increase by one or two in each iteration [10].

Observing Figure 3.4(a) it is possible to verify that in the initial iterations, the RRT quickly reached unexplored parts, but it is dense in the limit (with

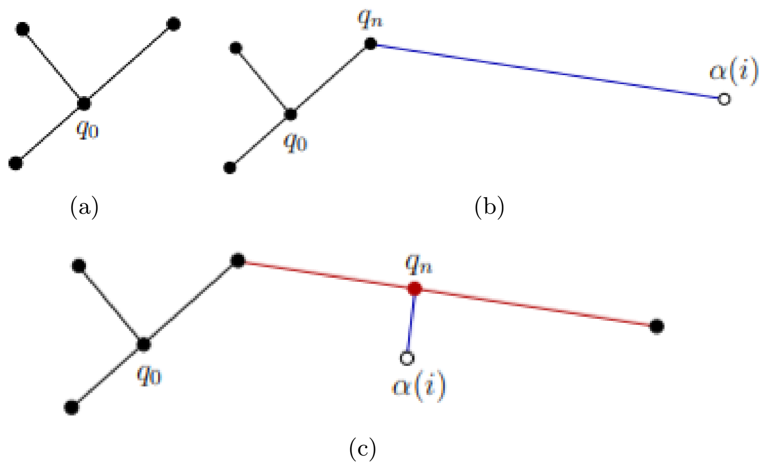


Figure 3.3: Tree constructed so far (a), addition of new edge and connection from the sample $\alpha(i)$ to the nearest point, which is the vertex q_n (b) and division of the edge into two and insertion of new vertex (c) [10]

probability one), which means that it gets arbitrarily close to any point in the space. Several main branches were first constructed because it rapidly reached the far corners of the space. Progressively, more and more area was filled by smaller branches, as can be seen in Figure 3.4(b). From the images, it is noticed that in the limit, the tree fills densely the space. In this way, it is verified that the tree gradually improves the resolution (or dispersion) as the number of iterations increases [10].

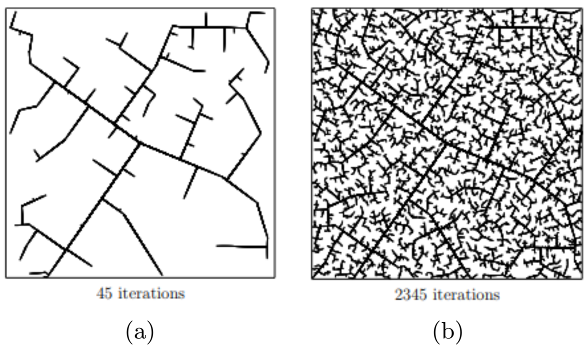


Figure 3.4: Tree obtained for 45 iterations (a) and tree obtained for 2345 iterations (b) [10]

In sampling-based motion planning, the obstacle region (C_{obs}), present in Figure 3.5, is not explicitly represented, hence it must be taken into account when building the tree. The procedure for solving the mentioned situation allows to obtain the closest possible configuration of the C_{free} boundary (free space), along the direction toward $\alpha(i)$. The nearest point, q_n, is set to be the same (obstacles are ignored), however the new edge may not reach $\alpha(i)$. In this case, an edge is constructed from q_n to q_s (last possible point before reaching the obstacle). The proximity of the edge to the boundary of the obstacle depends on the method used to check the collision, as explained in [10] (section 5.3.4.). Sometimes it may happen that q_n is already as close as possible to the C_{free} limit in the direction of $\alpha(i)$. In this case, no new edge or vertex is added for that iteration [10].

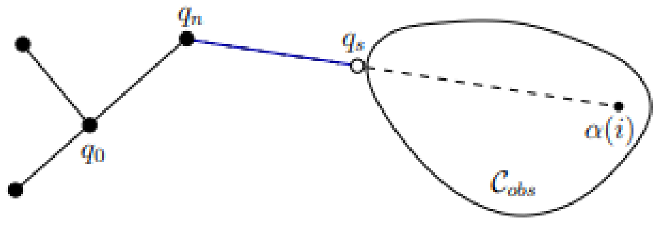


Figure 3.5: Presence of an obstacle in the construction of the tree [10]

3.1.4 Time Windows

In dynamic routing a calculated path depends on the number of currently active AGV' missions and their priorities. The Time Windows method allows to determine the shortest path using time windows (Figure 3.6). This method checks the mission candidate paths by using the time windows to verify if certain paths are feasible. Viability of a particular path is evaluated by a time windows insertion followed by a time windows overlap (conflict) test. In the case of conflict, the algorithm iteratively reinserts time windows until conflicts disappear or an overlap is present only on the path's origin arc, indicating that the candidate path is not feasible. The procedure is repeated for all candidate paths and the result is a set of executable paths. The final task of the algorithm is to choose the shortest one among executable paths in terms of a time required for a vehicle in mission to get from the origin to the destination arc. When a new mission is requested at a given time, is searched a idle vehicle to assign it to that mission (with initial mission priority). As a goal of dynamic routing is to determine the shortest path for certain mission under current state of the system, all candidate paths should be checked for feasibility [11].

In this method each task is answered within a certain time interval. Thus, when an AGV arrives at a work center, it can only perform the task of the same, if the time it arrived is greater than or equal to the beginning of the time window. If a vehicle arrives before the time window starts, has to wait. On the other hand, after answering the task of a work center, the time at which the vehicle leaves

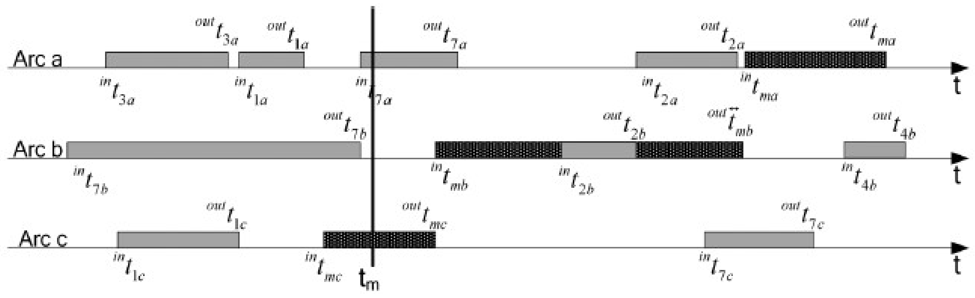


Figure 3.6: Example of time windows in a vector form [11]

the center must be less than or equal to the end of the time interval given by the work center [38].

The purpose of this method is to help find routes, for an AGV fleet, without collisions and that minimize the total distance travelled, so that all tasks are satisfied within the defined time interval for each work center [38].

3.1.5 Genetic Algorithms

Genetic Algorithms (GA) are a parallel and global search technique that emulates natural genetic operators. As it simultaneously evaluates many points in the parameter space, it is more likely to converge to the global optimal [12].

Many path planning methods use a grid-based model to represent the environment space. The grid-based environment space can be represented in two ways, by the way of an orderly numbered grid, as shown in Figure 3.7, or by the way of (x,y) coordinates plane [12].

A chromosome represents a candidate solution for the path planning problem. A chromosome or a path consists of a starting node, a target node and the nodes though which the mobile robot travels. These nodes or steps in the path are called genes of the chromosome. A valid path consists of a sequence of grid labels which begins from starting node and ends at the target node, as shown in Figure 3.8 [12].

The initial population is generally generated randomly. Thus, some of the generated chromosomes may include infeasible paths which intersect an obstacle. An optimal or near optimal solution can be found by genetic operators, even though the initial population includes infeasible paths. However, this process reduces the search capability of the algorithm and increases the time to find an optimal solution. Furthermore, crossover of two infeasible chromosomes may generate new infeasible paths. In order to solve this problem, each chromosome must be checked whether it intersects an obstacle, when generating the initial popu-

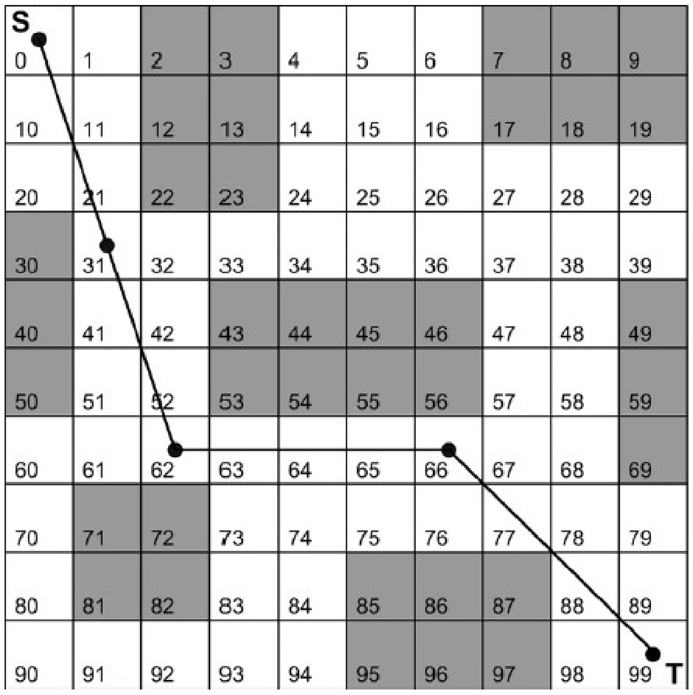


Figure 3.7: Example of the orderly numbered grid environment representation [12]

lation. If it does, the intersected gene on the chromosome is changed randomly with a feasible one [12].

The optimal path may be the shortest, the one requiring the least time or less energy to be traversed. Generally, in the path planning problems, the objective function is considered as the shortest path. In the performed study in [12], the objective function value of a chromosome used in GA is given by equations 3.4 and 3.5:

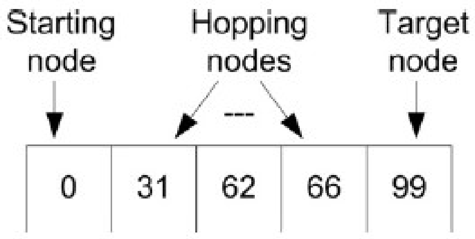


Figure 3.8: Decimal coded genes of a chromosome [12]

$$f = \begin{cases} \sum_{i=1}^{n-1} d(p_i, p_{i+1}), & \text{for feasible paths} \\ \sum_{i=1}^{n-1} d(p_i, p_{i+1}) + penalty, & \text{for infeasible paths} \end{cases} \tag{3.4}$$

$$d(p_i, p_{i+1}) = \sqrt{(x_{(i+1)} - x_i)^2 + (y_{(i+1)} - y_i)^2} \tag{3.5}$$

In these equations, f is the fitness function, p_i is the ith gene of chromosome, n is the length of the chromosome, d is the distance between two nodes, x_i and y_i are the robot's current horizontal and vertical positions, $x_{(i+1)}$ and $y_{(i+1)}$ are the robot's next horizontal and vertical positions. The direction of the robot is determined by the equation 3.6 [12]:

$$\alpha = \tan^{-1} \frac{(y_{i+1} - y_i)}{(x_{i+1} - x_i)} \tag{3.6}$$

The objective function value is defined as the sum of distances between each node in a path. If there is an obstacle in the direction of the robot, a penalty is added to the objective function value. The penalty value should be greater than the maximum path length on the environment. In order to find an optimal path, the algorithm searches for a chromosome whose penalty is eliminated [12].

The main principle of the GA is that the best genes on the chromosomes should survive and be transferred to new generations. A selection procedure needs to be done to determine the best chromosomes. The selection process consists in the following three steps [12]:

1. Objective function values of all chromosomes are found.
2. Fitness values are assigned to chromosomes according to their objective function values. In the study performed in [12], the rank based fitness assignment is used instead of the proportional assignment method. This prevents a few better chromosomes to be dominant in the population.
3. Chromosomes are selected according to their fitness values and then put into a mating pool to produce new chromosomes.

Normally, crossover combines the features of two parent chromosomes to form two offsprings. In Figure 3.9, single-point crossover operator is used, and the genes of the two "parent" chromosomes after the crossover point are changed [12].

All candidate chromosomes in the population are subjected to the random mutation after the crossover operation. This is a random bit-wise binary complement operation or a random small change in a gene, depending on the coding of chromosomes, applied uniformly to all genes of all individuals in the population

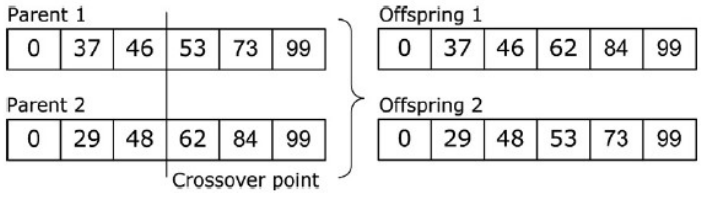


Figure 3.9: Single-point crossover [12]

with a probability equal to the mutation rate. The mutation operation expands the search space to regions that may not be close to the current population, thus ensuring a global search. The mutation operation increases the diversity of the population and avoids the premature convergence [12].

In conventional GA, random mutation is the most commonly used operator. However, random mutation can cause generation of infeasible paths. Even though a chromosome is feasible before the mutation operation, the new node changed by the mutation may have an obstacle and therefore it constitutes an infeasible path, as shown in Figure 3.10. This makes the optimization slower and increases the number of generations [12].

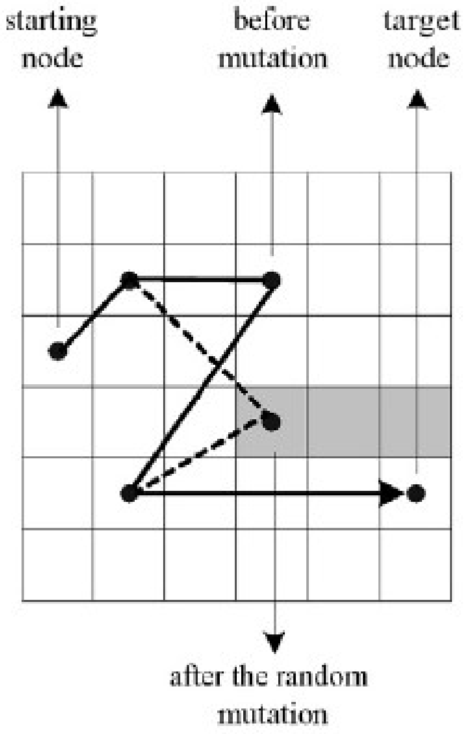


Figure 3.10: Infeasible path [12]

In order to overcome this problem, several studies concerned with the improvement of mutation operation have been done in the literature. The authors of those studies, as well as the method proposed by each author, are described in [12].

3.2 Task Scheduling Algorithms

Scheduling tasks allow to order and sequence the tasks to be performed. Scheduling algorithms can be seen as rules to follow for scheduling tasks. Some rules will be mentioned, as well as an algorithm that helps to solve AGV affectation problems to tasks.

3.2.1 Shorter Distance Rule

This algorithm is ruled by the nearest task rule. Three strategies can be adopted for this rule: (*i*) each AGV has its task list, does not share it with the other AGV and always executes which is closest; (*ii*) the AGV share the task list. The task is verified and associated with the closest AGV. In this way, the situation may be that the AGV is always closer to the tasks and the others do not perform any tasks; and (*iii*) the AGV share the task list. For each AGV is checked which is the closest task.

3.2.2 Longest Idle Vehicle Rule

This rule gives the highest priority to the AGV which has remained available for a longer time, in the aggregate of all available vehicles. That is, the AGV used is that whose difference between the current time and the time the AGV became available is maximum [39].

3.2.3 Least Utilized Vehicle Rule

This rule needs to have access to time statistics of the AGV in order to determine the rate of use of each vehicle. The vehicle chosen shall be the one with the lowest rate of use, up to the moment when the expedition decision has to be taken [39].

Like the longest idle vehicle rule, the least utilized vehicle rule allows to balance the workload of the various AGV of a fleet [39].

3.2.4 Modified First Come-First Serve Rule

This rule seeks to grant AGV to work centers in chronological order of reception of pending transport tasks [39].

When a work center issues a new task and it can not be immediately answered, the time at which the task is issued is saved. If there are issued new tasks by the same work center before the first pending task is assigned to a vehicle, its issued time is not saved. Thus, no work center can have more than one task to compete for the use of AGV [39].

When a vehicle becomes available, it is assigned to the work center whose first pending task has the shortest issuing time. When the pending task of a work center is attended, the transportation needs of the center are evaluated and updated in one of two possible ways [39]:

- if there are no more pending tasks, the work center does not need any other AGV;
- if more than one AGV is required, the oldest new pending task is assigned the time the last AGV was assigned to a task in the work center in cause (the time the last pending task was served). That is, if there are, for example, three work centers, the time at which the last AGV was assigned to a task of the center 1, is assigned to the next task to be taken care of from that center. Thus, the AGV can perform tasks of the remaining work centers.

This rule does not directly take into account any exhaustion of the capacity of the work center queues, however, it assures the reduction of the attending time of a task. The number of assignments made to a work center is related with its intensity of manufacturing operations achievement [39].

3.2.5 Hungarian Algorithm

The "classic" problem consists in affecting n individuals to n tasks, reducing the time spent in tasks achievement. Thus, is presented the Model exposed in 3.7, where j represents the tasks and i the vehicles, c_{ij} is the cost of affecting vehicle i to task j, and n is the number of vehicles (in the case of vehicle sum) or number of tasks (in the case of the task sum) and x_{ij} the coefficients of the Efficiencies Matrix [40, 39].

$$\begin{gathered} max \quad Z = \sum_{j=1}^{n}\sum_{i=1}^{n} c_{i_j} x_{ij} \\ \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, ..., n \\ \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, ..., n \\ x_{ij} \in \{0, 1\}, \quad \forall \ i, j \end{gathered} \tag{3.7}$$

In the affectation problems each AGV is affected to a single task and each task is performed by a single AGV. The matrix of the coefficients c_{ij} is called the

Efficiencies Matrix, in which if a constant is added or subtracted to each row or column of the matrix, the optimal affectation is not changed, but its cost [40].

Assumptions

The assumptions for the Hungarian algorithm are [40]:

- $c_{ij} \geqslant 0$
- Minimization problem. That is, when the problem is of maximization, the complement for the maximum of all the elements of the matrix is calculated and the problem is solved as if it were of minimization.

Procedure

The objective of the algorithm is to obtain, by successive manipulation, an equivalent matrix to the efficiencies in which a null cost affectation can be defined. The steps mentioned below serve to implement the algorithm [40].

1. Subtract to each line of the efficiencies matrix its smallest element.
2. Subtract to each column of the efficiencies matrix its smallest element.
3. Cover with the fewest traces all zeros of the matrix.
4. If that number is equal to n, was found the optimal affectation. If not, select the smallest number not traced, subtract it from all the elements not traced and add it to the tracings twice. Go back to point 3.

The efficiencies matrix has to be square, that is, the number of AGV to be affected must be equal to the number of tasks to which they should be affected. When this is not verified, it is necessary to convert the problem into another equivalent, whose efficiencies matrix is square. To this, AGV or fictitious tasks are created, in order to be possible to have a problem in which the number of AGV and tasks is equal. The costs to be introduced are worth infinite for the affectation of a real AGV to a fictitious task or for the affectation of a fictitious AGV to a real task, and are worth zero for the affectation of a fictitious AGV to a fictitious task. This procedure aims to ensure that AGV and fictitious tasks are affected among themselves. The value of the matrix size (n) must be equal to, or greater than, the number of system AGV, so that AGV do not stay unnecessarily free when there are enough tasks to occupy all vehicles. Tasks that can not be considered by the algorithm, or that are attributed to fictitious AGV, stay waiting in queue. AGV attributed to fictitious tasks remain available [39].

By using the hungarian algorithm it is possible to implement a dynamic reaffectation mechanism. This type of mechanism has the objective of better optimizing the affectation of AGV to the tasks, at the expense of a greater number of tasks and of AGV involved in the scheduling process. Without this mechanism, when the scheduling process is called, only tasks in waiting queue intervene, as well as AGV that are stopped and available. With the mechanism, also intervene the tasks and the AGV affected to them that are in the first step of their execution, that is, in which the vehicle is still moving to the point of loading. In this way, these tasks can be perfectly affected in a different way to the vehicles, if the new affectation allows a better optimization of the global cost of executing the tasks [39].

The scheduling process is called whenever an AGV completes the execution of one task and becomes available for another, provided that there is at least one task in queue or in the first phase of its execution. It may happen that the vehicle that becomes available is better placed to perform a currently running task than the vehicle that is currently allocated to the task. In this type of situations, the dynamic reallocation allows that the AGV that becomes available performs a task that was already allocated to another AGV, resulting in a better optimization [39].

3.3 Conclusion

This chapter presented planning and scheduling algorithms for AGV fleets which allow AGV to carry out the intended tasks and move to these tasks according to the best route.

Chapter 4

Work Developed

In this chapter is described the proposed problem to be addressed in this work, as well as the way it was solved, being also presented the architecture of each simulated model (components used and its functions) and the functionalities of the implemented algorithms.

4.1 Problem and Solution

This project consisted in the creation of layouts with paths for AGV to move between places of loading and unloading of material, obeying algorithms of planning and scheduling. Therefore, models were created, in which AGV, paths, loading and unloading sites, loading station and controller for the vehicles were inserted and, still a component to program the algorithms. With this in mind, the most advantageous way of representing the “map” of each model layout and implementing the algorithms was missing. Given this, a solution arose consisting of the map representation using a graph. With this, pillars were inserted to define the points (nodes) of the graph.

4.2 Model Architecture

The simulations performed involved components of the Visual Components software library, such as AGV (Adept Lynx), AGV Controller, AGV Task Caller, AGV Charging Station, AGV Pick Locations, Shape Feeders, AGV Drop Locations, AGV Pathways, AGV Crossings and Pillars (Round). All these components are displayed in Figure 4.1.

Each model can contain more than one AGV (and of different types), with only one AGV Controller to manage/control its fleet, one AGV Task Caller to program

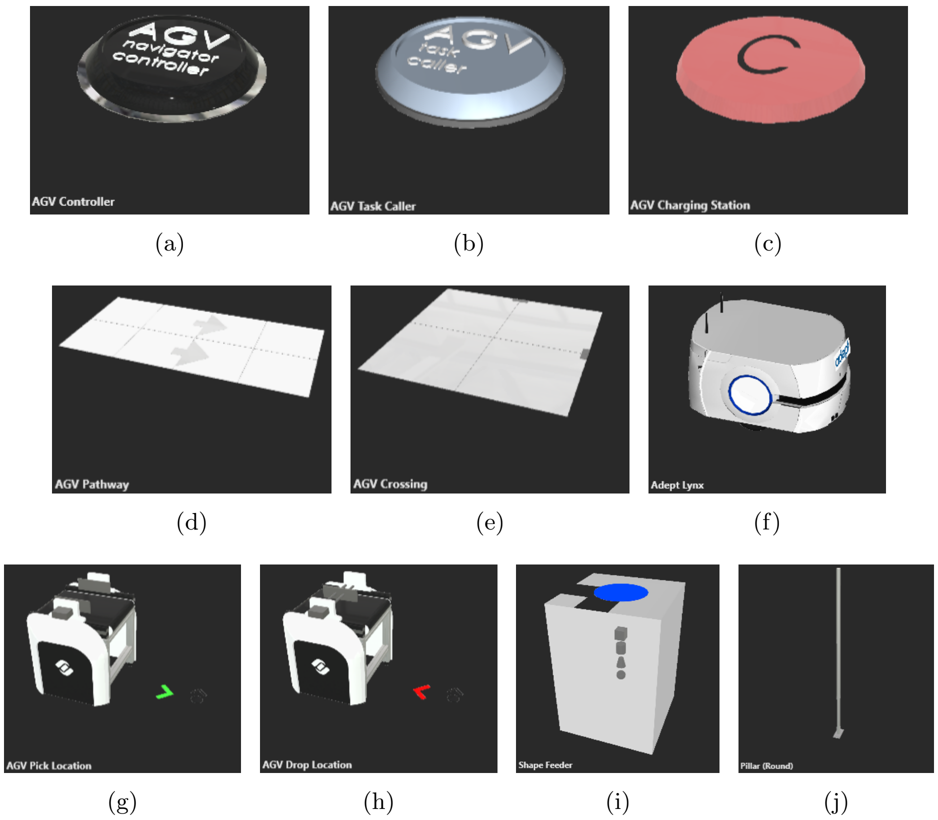


Figure 4.1: AGV Controller (a), AGV Task Caller (b), AGV Charging Station (c), AGV Pathway (d), AGV Crossing (e), Adept Lynx (f), AGV Pick Location (g), AGV Drop Location (h), Shape Feeder (i) and Pillar (Round) (j)

the algorithms and one AGV Charging Station where the AGV can charge its battery. For each model there may also be several Pick and Drop Locations. The AGV loads the materials in the Pick Location and unloads them in the Drop Location. Each Pick Location has to be connected to a Shape Feeder. The Shape Feeder serves to deliver products to the Pick Location. The AGV Crossings (crosses that allow the AGV to change direction) and AGV Pathways (paths that allow the AGV to circulate in various directions and tracks) are necessary in order for the vehicles, through their AGV Controller, to be able to travel on the defined tracks and in the directions of the tracks. The AGV Pathways also serve to connect the Pick and Drop Locations, so that the AGV can interact with them. For this end, the Pick Location has to have its green arrow and the Drop the red arrow on the pathway. The black arrow of these components relates to the direction with which the AGV will load the parts. Figure 4.2 illustrates the positioning of the mentioned arrows. Lastly, the Pillars were used to define the coordinates of the points (nodes) belonging to the graph created for the AGV so that the points are easily identified. These have alphabet letter names.

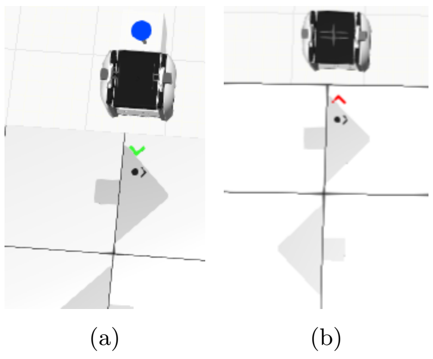


Figure 4.2: Position of the arrows of the AGV Pick Location (a) and position of the arrows of the AGV Drop Location (b)

4.3 Practical Implementation

This section presents the algorithms that were implemented, including the components used, the configurations of the components and the operating mode of each algorithm.

4.3.1 Trajectory Planning Algorithms

The planning algorithms implemented were A* and TEA*. The A* was first implemented, and later the TEA*, since the latter includes the former.

4.3.1.1 A*

The present algorithm is indicated to plan the displacement of only one AGV, because when there are several, it can not detect and avoid collisions. Hence the TEA* has been implemented in the sequel.

Components Used

The A* algorithm implementation was started by creating the AGV track, connecting the AGV Pathways with the AGV Crossings. Afterwards, the Pillars were placed in specific places of the track, with the purpose of forming a graph of nodes through which the AGV would move. Subsequently, an AGV Pick and an AGV Drop Location, a Shape Feeder, an AGV (Adept Lynx), an AGV Controller, an AGV Task Caller and an AGV Charging Station were added. All this can be seen looking at Figure 4.3.

Settings in the Components Properties

After several experiments performed in the software, the following conclusions were withdrawn.

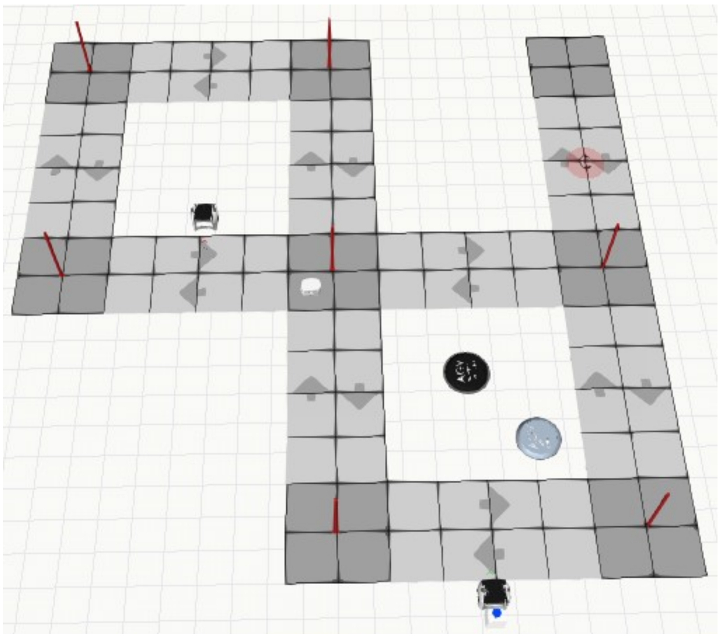
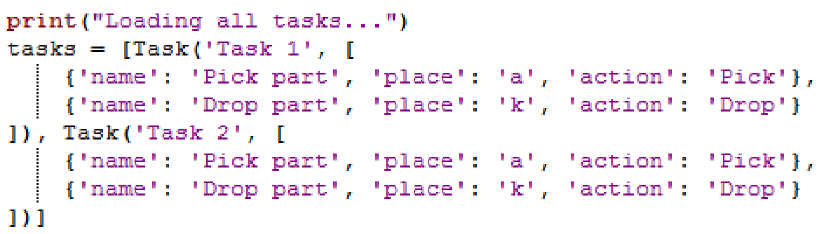


Figure 4.3: A* simulation layout

AGV Pick Location

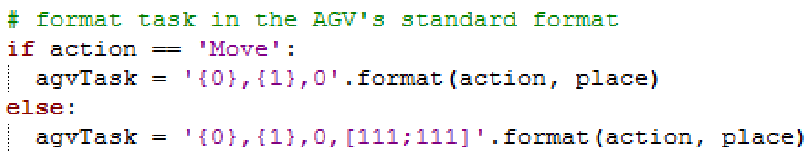
- Disable the `UseLocalCall` option because the AGV Task Caller was used. In this way, the AGV can interact with the AGV Pick Location.
- Using the AGV Task Caller it is not necessary to fill in the Default section: (*i*) the name of the AGV Drop Location where the AGV is intended to download the material in the `AGV_DropLocationName` field, since the drop actions are given in the form of tasks that already include the name of the AGV Drop Location (in this case the name is "k"), as can be seen from the example of the Figure 4.4; (*ii*) the AgvID, in the `AGV_ID` field, because in the `OnRun()` function is the name of the AGV (Adept Lynx) that is intended to perform the defined tasks; and (*iii*) the number of parts to be loaded by the AGV in the `AGV_PickCount` field, nor the wagon where the parts are intended to be loaded, in the `AGV_Wagon` field, because the number of parts to be loaded and the wagon are included in the `OnRun()` function. The parts are identified by their identifier (ID) ([111 ; 111]). The field of the wagon, in this case has the value zero, that is, the parts are loaded in the AGV. The information of this last point can be confirmed looking at Figure 4.5.

The above information is summarized in Table 4.1 and illustrated in Figure A.1 of the Appendix A.



```
print("Loading all tasks...")
tasks = [Task('Task 1', [
    {'name': 'Pick part', 'place': 'a', 'action': 'Pick'},
    {'name': 'Drop part', 'place': 'k', 'action': 'Drop'}
]), Task('Task 2', [
    {'name': 'Pick part', 'place': 'a', 'action': 'Pick'},
    {'name': 'Drop part', 'place': 'k', 'action': 'Drop'}
])]
```

Figure 4.4: Tasks example



```
# format task in the AGV's standard format
if action == 'Move':
    agvTask = '{0},{1},0'.format(action, place)
else:
    agvTask = '{0},{1},0,[111;111]'.format(action, place)
```

Figure 4.5: Example of standard task format

Table 4.1: Pick Location Settings

Component	Options/Fields				
	`UseLocalCall`	`AGV_DropLocationName`	`AGV_ID`	`AGV_PickCount`	`AGV_Wagon`
AGV Pick Location	x	x	x	x	x

✓- Yes
x - No

Shape Feeder

- The parts ID is in the `ProdID` (Figure A.2, Appendix A), field of the `ProductParams` section.

Adept Lynx

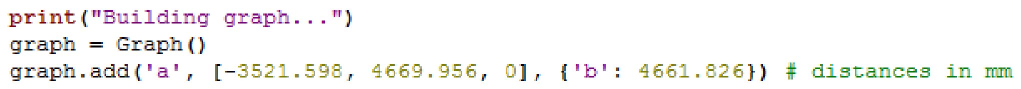
- The acceleration and deceleration were respectively set in the `AGVAcceleration` and `AGVDesaceleration` fields of the `Default` section, with the value zero, so that the AGV will not slowdown on arrival or accelerate on leaving at the points identified by the Pillars. Thus, in the AGV Pick and Drop Location also does not slowdown or accelerate. The velocity is constant throughout the simulation (1000 mm/s).

- In the Stations field of the `ReCharge` section was added the name of the AGV Charging Station, present in the properties of the AGV Charging Station, where the AGV can go to charge its battery. Here, too, the charge capacity of the AGV, its charging time and its initial charge capacity are present.

The settings mentioned for Adept Lynx are shown in Figure A.3 of the Appendix A.

Functioning of the Algorithm

The implementation of the algorithm started with the creation of a graph, as shown in Figure 4.6. This graph is constituted by nodes (Pillars) and each node contains a name, its coordinates and its neighbours, with the respective distances among them. These distances were obtained using the properties of the AGV (`Default` section, `TravelledDistance` field), and the task format (already seen previously in Figure 4.4), while the vehicle was forced to travel from each node to its neighbours nodes, thus obtaining the value of the respective distances, in millimeters. These measurements were taken before the implementation of the A* algorithm. Initially, random values were entered for the distances among nodes, so that the program was able to run, and the AGV could move to the desired nodes. Later, after the program was executed for the first time, and the correct distances determined, these were introduced in the A* algorithm.



```
print("Building graph...")
graph = Graph()
graph.add('a', [-3521.598, 4669.956, 0], {'b': 4661.826}) # distances in mm
```

Figure 4.6: Example of insertion of a node in the A* graph

The operation of the algorithm is explained through the flow chart depicted in Figure 4.7 and included the following steps:

1. Detection of the AGV location node.

2. Verification of which is the first step of the task that the vehicle will have to perform.

3. Addition of the location node of the AGV to the closed list[1].

4. Verification of which are the neighbours of that node.

5. For each neighbour node it is checked whether its name matches the name of the destination node:

   - if yes: (*i*) the AGV moves to the destination node; (*ii*) the origin and destination of the vehicle, as well as the way made and its cost are presented; and (*iii*) if there is another step to be performed on the task, this it is checked, it is also verified which is the location node of the AGV and is repeated everything from step 3, inclusive. If there is no next step, the AGV checks if there is another task to perform and, if it exists, repeats

[1]List containing the nodes already visited by the vehicle.

everything again from step 1, inclusive. If there isn't another task, the algorithm ends, the origin and destination of the vehicle, the path made and its cost, as well as the total distances (costs) of each task are presented;

- if not, the value of f is calculated and the node is added to the open list[2].

6. Empty path verification and open list not empty:

   - if yes: (*i*) is removed from the open list the node with the lowest value of f and added to the closed list; (*ii*) the neighbours of that node are checked; (*iii*) for each neighbour: is calculated the value of g (sum of the g of the current node with the g of the node added to the closed list) and f and is verified if the name of the neighbour corresponds to the name of the destination node. If the name is the same, is repeated the true condition of step 5. If it is different, it is checked if the neighbour in study is already on the closed list. If it is already, advances to the next neighbour if it exists, if it does not exist re-check this step (6). If it is not already, it is checked if the neighbour in study is already on the open list. If yes, compares the value of calculated f with what is in the list and if the calculated is smaller, eliminates the largest and puts the lowest one in the list. If not, it is added to the open list;

   - if not: (*i*) the AGV stays where it is and the points (*ii*) and (*iii*) of the true condition of step 5 are executed.

4.3.1.2 TEA*

The TEA* algorithm was created based on the A* algorithm, with the objective of being used for situations in which there are several AGV navigating the layout and it is necessary to anticipate collisions, in order to avoid them, and to solve deadlock situations, that is, to recalculate the vehicle's path so as not to collide with other vehicles.

The implementation of the TEA* algorithm was very similar to that of the A*, since the components used were the same, adding only more AGV, AGV Pick and Drop Locations for these new vehicles. The configurations in the components properties also remained the same and, finally, the operation of the TEA* encompassed the A*; however, were added to the graph, for each node, the times to each neighbour, as shown in Figure 4.8, and the detection of collisions between AGV.

[2]List that contains the nodes that have not yet been visited by the vehicle.

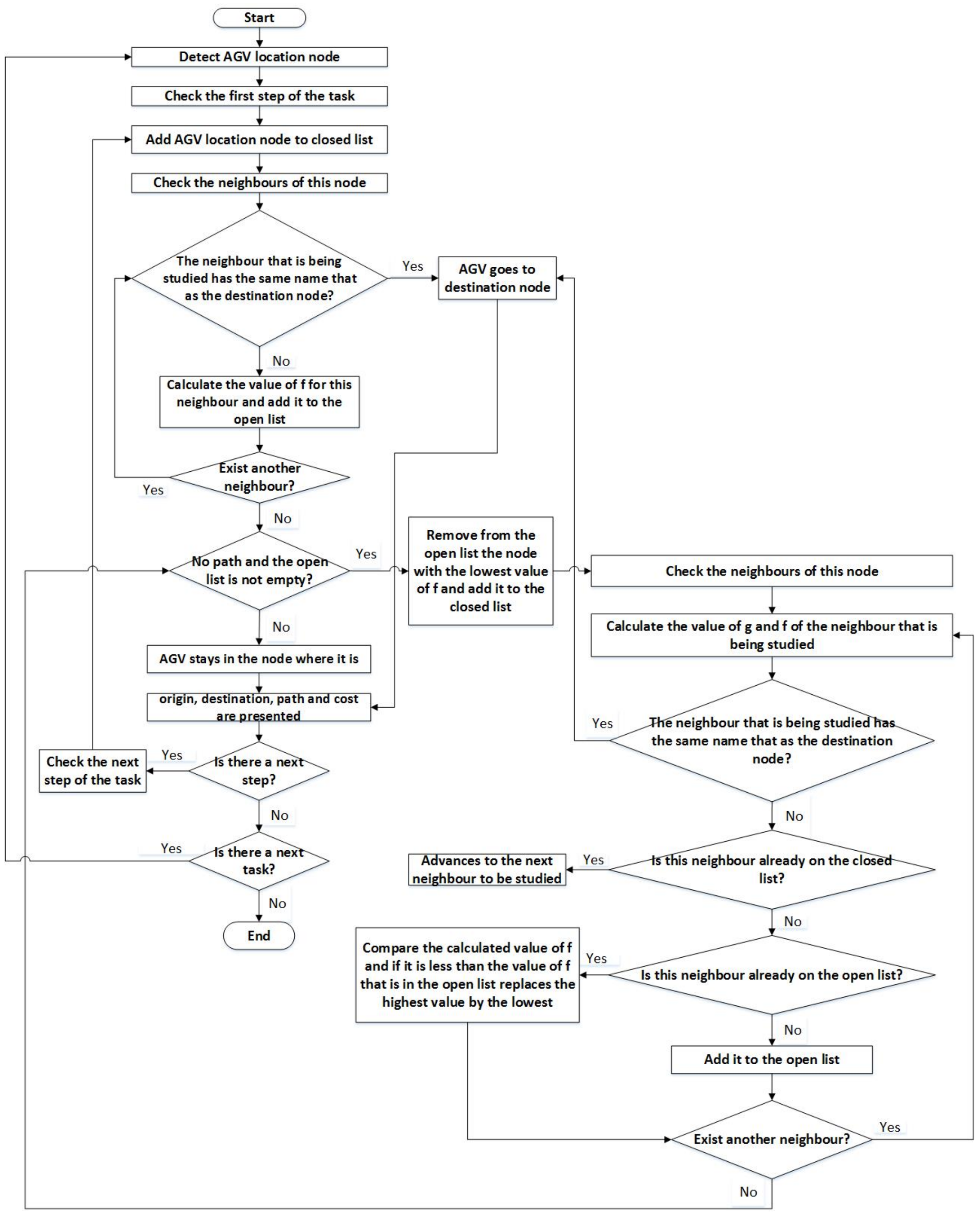
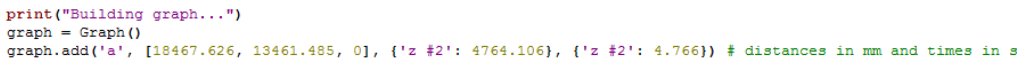

Start
Detect AGV location node
Check the first step of the task
Add AGV location node to closed list
Check the neighbours of this node
The neighbour that is being studied has the same name that as the destination node?
Yes
AGV goes to destination node
No
Calculate the value of f for this neighbour and add it to the open list
Exist another neighbour?
Yes
No
No path and the open list is not empty?
Yes
Remove from the open list the node with the lowest value of f and add it to the closed list
Check the neighbours of this node
No
AGV stays in the node where it is
Calculate the value of g and f of the neighbour that is being studied
origin, destination, path and cost are presented
Yes
The neighbour that is being studied has the same name that as the destination node?
Check the next step of the task
Yes
Is there a next step?
No
No
Yes
Is there a next task?
Yes
Advances to the next neighbour to be studied
Is this neighbour already on the closed list?
No
No
End
Compare the calculated value of f and if it is less than the value of f that is in the open list replaces the highest value by the lowest
Yes
Is this neighbour already on the open list?
No
Add it to the open list
Exist another neighbour?
Yes
No

Figure 4.7: Flow chart of A* algorithm



```
print("Building graph...")
graph = Graph()
graph.add('a', [18467.626, 13461.485, 0], {'z #2': 4764.106}, {'z #2': 4.766}) # distances in mm and times in s
```

Figure 4.8: Example of insertion of a node in the TEA* graph

The operation of the TEA* algorithm included the following steps:

1. Execution of the two first steps of the A* algorithm previously mentioned.
2. Presentation of all AGV maps until now.
3. Addition of the location node of the AGV to the closed list[3].
4. Verification of which are the neighbours of that node.
5. Verification of which are the times of those neighbours.
6. For each neighbour node:
   - it is verified, in the maps of the other AGV, if in the time in study the node itself is occupied by some of the other AGV:

     - if yes: (*i*) says which is the AGV that is occupying that node at that time; (*ii*) shows the way up to the moment and a message that refers between which AGV there will be collision, in which node and at what time; and (*iii*) advances to the study of the next neighbour's time, if it exists. If there is no next neighbour advances to step 7.

     - if not: (*i*) is returned false; (*ii*) it is checked whether at that time the origin node of the respective AGV is occupied by some of the other AGV. If it is busy, it says by which AGV and it presents the way up to the moment and a message that refers between which vehicles there will be the collision, in what node and at what time and what was the way that the vehicle was trying to follow. Advances to the next neighbour, if it exists. If there is no other neighbour advances to step 7. If it is not busy, it returns false and goes to the next verification.
   - it is checked whether its name matches the name of the destination node:

     - if yes: (*i*) the AGV moves to the destination node; (*ii*) the origin and destination of the vehicle, the route made and its cost and also the times of the nodes that form the path are presented; and (*iii*) if there is another step to be performed on the task, it is verified, it is also checked which is the location node of the AGV and repeats everything again from step 2, inclusive. If there is no next step, the AGV checks if there is another task to perform and, if it exists, repeats everything again from step 1, inclusive. If there isn't another task, the algorithm ends, the origin and destination of the vehicle, the route made and its cost, the times of the nodes that form the path, and also the total distances (costs) of each task are presented;

[3]In the case of TEA*, besides containing the nodes already visited by the vehicle, also contains the nodes that belong to dead ends.

- if not, the value of f is calculated and the node is added to the open list.

7. Empty path verification and open list not empty:

   - if yes: (*i*) is removed the node with the lowest value of f from the open list and add it to the closed list; (*ii*) check which are the neighbours of that node and their times; (*iii*) for each neighbour:

     a) check if the neighbour in study is already on the closed list. If it is, it advances to the next neighbour, in case it exists; if it does not exist, it will check this step again (7). If it is not already, is calculated the value of g (sum of the current node g with the g of the node added to the closed list), the time (sum of the current node time with the time of the node added to the closed list) and the f;

     b) it is verified if in the study time the AGV in study will collide with the other vehicles. If there is a collision, it returns which is the AGV that is occupying that node at that time, it presents the path up to the moment and the times of the nodes that compose this path and still, a message that indicates between which AGV there will be collision, in which node and at what time. Advances to the study of the next neighbour's time, if it exists. If there is no next neighbour, checks this step again (7). If there is no collision, it is returned false and checked if at that time the node that was added to the closed list is occupied by some of the other AGV. If it is busy, it returns by which AGV, it presents the path up to the moment and the times of the nodes that compose this path and still, a message that refers between which vehicles will exist the collision, in which node and in what time and what was the way which the vehicle was trying to follow. Advances to the next neighbour, if it exists. If there is no next neighbour, checks this step again (7). If it is not busy, it returns false;

     c) verifies if the name of the node in study corresponds to the name of the destination node. If it corresponds, the true condition of the second point of the step 6 is performed. If it does not correspond, it is checked whether the neighbour in study is already on the open list. If yes, compares the f calculated value with what is in the list and if the calculated is smaller, it eliminates the larger one and places the smallest one in the list. If not, it is added to the open list.

   - if not: (*i*) the AGV stays where it is and the points (*ii*) and (*iii*) of the true condition of the second point of the step 6 are executed.

When it is not possible to move the AGV due the collisions without resolution, an error message is displayed indicating that a path without collisions can not be calculated.

In view of the examples of possible AGV maps, presented in Table 4.2, it is considered: (*i*) that the AGV 1 occupies the **d** node between [0 ; 9.359[s, the **c** between [9.359 ; 18.807[s and so on. According to this, it was decided to make the detection of the collisions first by the node that is being studied and then by the node that has already been added to the closed list, because is always necessary the next node to be able to make the intervals in which each node is busy. For example, if the last node added to the closed list was the **d** node, when examining the time of the first neighbour of **d**, example **c: 9.359**, it is checked whether the **c** node is occupied by some of the other AGV at this time, and if it is, a collision message is presented and the vehicle does not go through this node. If it is not, it is checked whether the **d** node is occupied by any of the other AGV at the same time. If it is, it also presents a collision message and the vehicle does not go though this node; and (*ii*) that when the path of AGV 1 is being studied, AGV 2 occupies the **z #2** node and AGV 3 the **g** node until the time of the **c** node (AGV 1 map), inclusive.

Table 4.2: Example of AGV maps (time units in seconds)

AGV 1	AGV 2	AGV 3
d: 0	z #2: 0	g: 0
c: 9.359	z: 9.636	h: 2.743
b: 18.807	y: 19.277	q: 12.198
a: 28.727	w: 28.990	r: 21.579

4.3.2 Task Scheduling Algorithms

In the previous chapter three strategies for this purpose were mentioned. The operation of each strategy is explained below.

Strategy 1

In this strategy each AGV has its task list, not sharing it with the other AGV, and always performs the task that is closest to it. To check which task is closest, the euclidean distance heuristic is used. That is, the distance between the position of the AGV and the position of each task is calculated. For the AGV to travel though the path with less cost, do not pass by already visited nodes and to avoid collisions, was included the TEA* planning algorithm in this strategy.

Strategy 2

In this strategy the AGV share the task list. The task is verified and associated to the closest AGV. In order to verify the closest AGV, the euclidean distance heuristic is used, as in the previous strategy. This heuristic allows to calculate the distance between the position of the task and the position of each AGV. For the AGV to travel through the path with less cost, do not pass by already visited nodes and to avoid collisions, was included the TEA* planning algorithm in this strategy.

Strategy 3

This strategy is similar to strategy 1, with the only difference being that the AGV here share the task list.

As in the three strategies in each task, it is necessary to first go to the AGV Pick Location, which already has an AGV Drop Location associated with it, and the heuristic is only applied between the AGV and the AGV Pick Location in strategies 1 and 3, or vice versa in strategy 2.

4.4 Conclusion

This chapter presented the problem addressed in this work and how it can solved, the model architecture and, finally, described some algorithms for AGV fleet trajectory planning and scheduling and detailed how these were implemented.

Chapter 5

Tests and Results

In this chapter are presented the operational tests performed and the results obtained, illustrating the operation of the various simulated models.

5.1 Trajectory Planning Algorithms

In this section is presented the simulated model with the A* algorithm and the models with the TEA* algorithm.

5.1.1 A* Model

The A* model (file `Astar_Simulation.vcmx`) was the first to be developed and tested. This model is composed by the layout shown in Figure 5.1. In this figure are the names of the Pillars (nodes) and of the AGV Pick (**a**) and AGV Drop Location (**k**), as well as the following components:

- 1 AGV Adept Lynx
- 1 AGV Pick Location
- 1 AGV Drop Location
- 1 AGV Charging Station
- 1 AGV Controller
- 1 AGV Task Caller
- 7 Pillars (nodes)

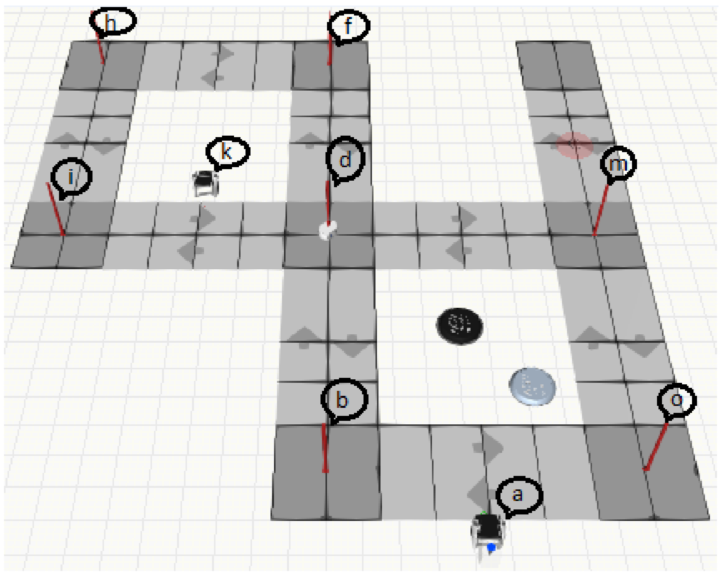


Figure 5.1: A* model layout and components

The objective of this model was to simulate the A* algorithm with the execution of two equal tasks. In each task the AGV has to go load two boxes to the AGV Pick Location and unload them in the AGV Drop Location, always traveling the smallest path and not passing in points that were already visited. The tests performed and the results obtained are presented below.

First step of the first task:

- origin: node **d**

- destination: AGV Pick Location **a**

The path, of the first step of the first task, performed by the AGV is represented in Figure 5.2. During the execution of this step, it was detected that the **o** node was already in the open list, because starting from **d** and with objective to go to **a**, analyzing the path through **b** (that has as neighbour **o**), and the path through **m** (that also has **o** as neighbour), resulted the warning shown in Figure 5.3. Looking to this figure it is possible to verify that the one that has lower cost is the path option [d, m, o]. The results obtained were:

- Path: [m, o, a]
- Travelled distance (cost - f): 24 083 mm

Second step of the first task:

- origin: AGV Pick Location **a**

- destination: AGV Drop Location **k**

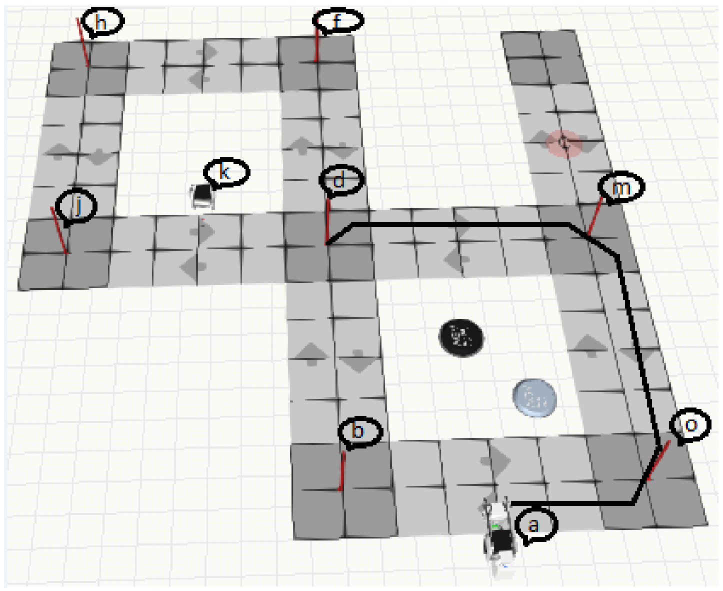
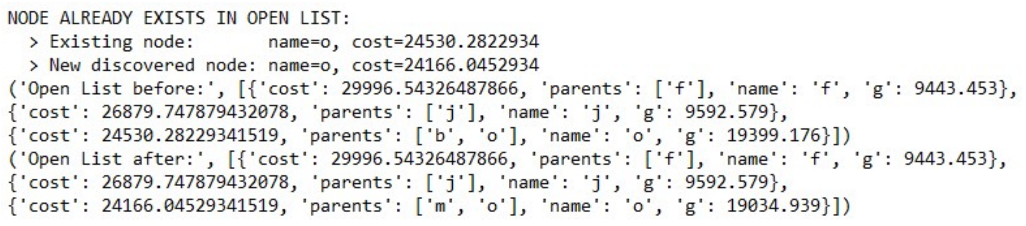


Figure 5.2: Travelled path between **d** and **a**



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=o, cost=24530.2822934
  > New discovered node: name=o, cost=24166.0452934
('Open List before:', [{'cost': 29996.54326487866, 'parents': ['f'], 'name': 'f', 'g': 9443.453},
{'cost': 26879.747879432078, 'parents': ['j'], 'name': 'j', 'g': 9592.579},
{'cost': 24530.28229341519, 'parents': ['b', 'o'], 'name': 'o', 'g': 19399.176}])
('Open List after:', [{'cost': 29996.54326487866, 'parents': ['f'], 'name': 'f', 'g': 9443.453},
{'cost': 26879.747879432078, 'parents': ['j'], 'name': 'j', 'g': 9592.579},
{'cost': 24166.04529341519, 'parents': ['m', 'o'], 'name': 'o', 'g': 19034.939}])
```

Figure 5.3: **o** node already in the open list (A* model)

The path, of the second step of the first task, performed by the AGV is represented in Figure 5.4. The results obtained were:

- Path: [b, d, j, k]
- Travelled distance (cost - f): 28 926 mm

First step of the second task:

- origin: AGV Drop Location **k**

- destination: AGV Pick Location **a**

The path, of the first step of the second task, performed by the AGV is represented in Figure 5.5. During the execution of this step, it was detected that the **o** node was already in the open list, for the same reason as explained in the execution of the first step of the first task. The presented warning is displayed in Figure 5.6. The results obtained were:

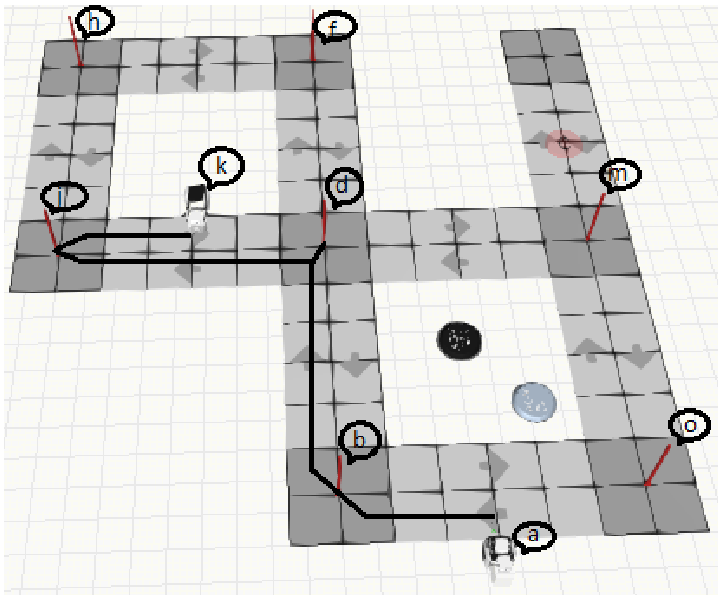


Figure 5.4: Travelled path between **a** and **k**

- Path: [d, m, o, a]
- Travelled distance (cost - f): 28 534 mm

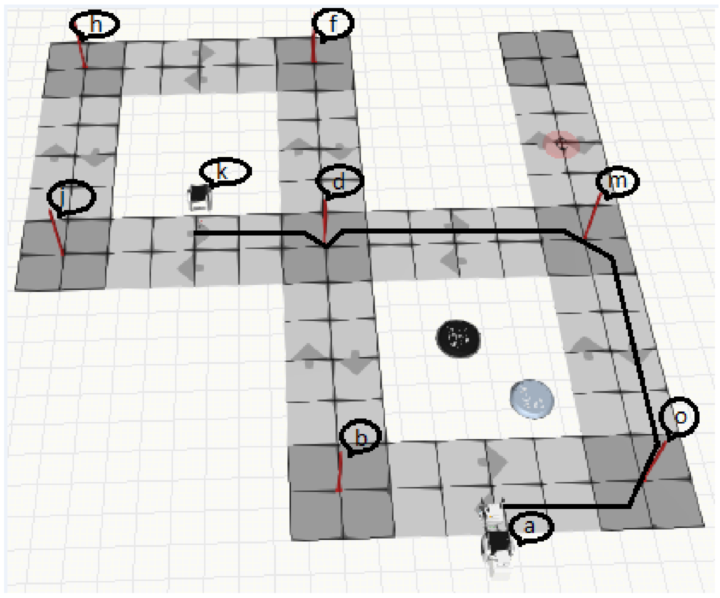
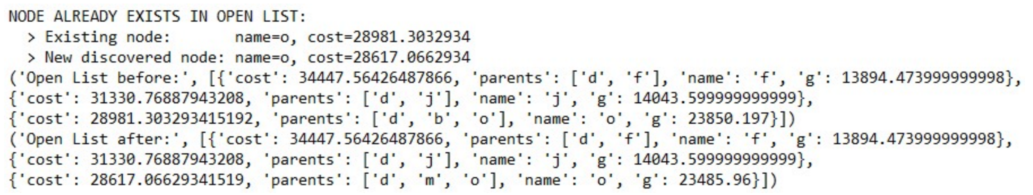


Figure 5.5: Travelled path between **k** and **a**



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=o, cost=28981.3032934
  > New discovered node: name=o, cost=28617.0662934
('Open List before:', [{'cost': 34447.56426487866, 'parents': ['d', 'f'], 'name': 'f', 'g': 13894.473999999998},
{'cost': 31330.76887943208, 'parents': ['d', 'j'], 'name': 'j', 'g': 14043.599999999999},
{'cost': 28981.303293415192, 'parents': ['d', 'b', 'o'], 'name': 'o', 'g': 23850.197}])
('Open List after:', [{'cost': 34447.56426487866, 'parents': ['d', 'f'], 'name': 'f', 'g': 13894.473999999998},
{'cost': 31330.76887943208, 'parents': ['d', 'j'], 'name': 'j', 'g': 14043.599999999999},
{'cost': 28617.06629341519, 'parents': ['d', 'm', 'o'], 'name': 'o', 'g': 23485.96}])
```

Figure 5.6: **o** node again already in the open list (A* model)

Second step of the second task:

- origin: AGV Pick Location **a**

- destination: AGV Drop Location **k**

The path, of the second step of the second task, performed by the AGV is the same as the one of Figure 5.4. The results obtained were the same as those obtained in the second step of the first task.

The total costs of each task were:

- First task: 53 009 mm
- Second task: 57 460 mm

In view of the obtained results it is concluded that the objectives were fulfilled and the task with lower cost was the first one, according to the presented values and with the Figures 5.2, 5.4, 5.5 that illustrate the routes that the AGV made.

5.1.2 TEA* Model

The TEA* model (file `TEAstar_Simulation.vcmx`) was the second to be developed and tested. For this model a more complex layout was created than that of A* and more AGV were added in order to detect possible errors in the TEA* functionalities and to originate more possibilities of collisions detection. The layout is shown in Figure 5.7. In this figure are the names of the Pillars (nodes), the AGV Pick (**a**, **l**, **s #2**) and the AGV Drop Locations (**g**, k, **x**), as well as the following components:

- 3 AGV Adept Lynx (Adept Lynx #1 (yellow), Adept Lynx #2 (green) and Adept Lynx #3 (blue))
- 3 AGV Pick Locations
- 3 AGV Drop Locations
- 1 AGV Charging Station

- 1 AGV Controller
- 1 AGV Task Caller
- 22 Pillars (nodes)

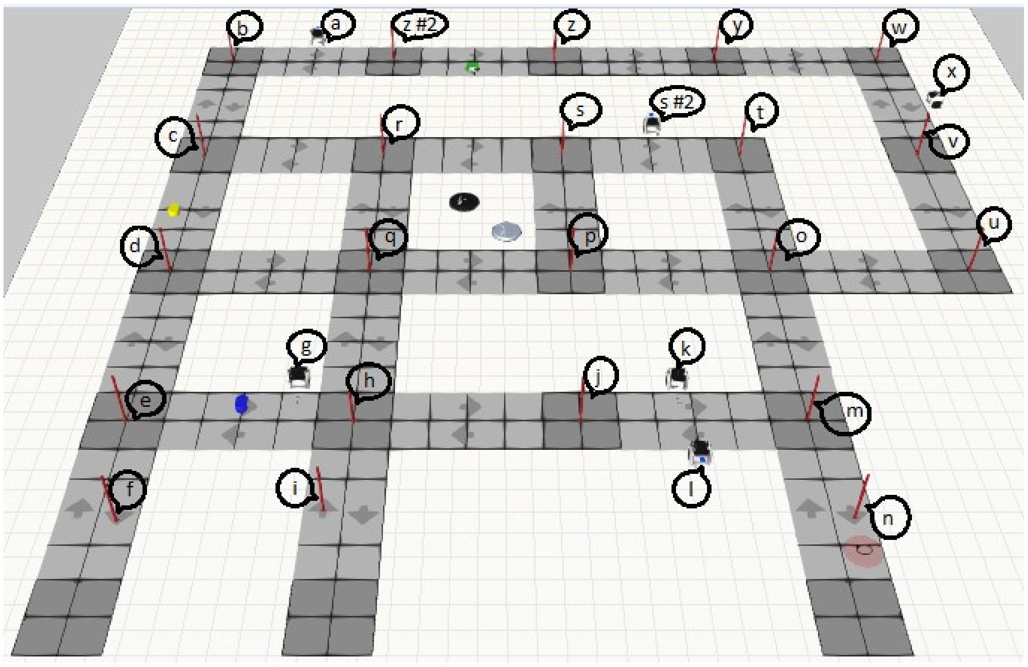


Figure 5.7: TEA* model layout and components

The objective of this model was to simulate the TEA* algorithm with the execution of three tasks. The tasks of each AGV are:

- AGV 1: load a box in the AGV Pick Location (**a**) and unload it in the AGV Drop Location (**g**);
- AGV 2: load a box in the AGV Pick Location (**l**) and unload it in the AGV Drop Location (**k**);
- AGV 3: load a box in the AGV Pick Location (**s #2**) and unload it in the AGV Drop Location (**x**);

Each AGV has to travel the smallest distance, not to pass in points that has already visited and nor in the points **f**, **i**, **n** (dead ends) and to avoid collisions with the other vehicles. The tests performed are presented below.

AGV 1 (yellow)

First step of its task:

- origin: **d** node

- destination: AGV Pick Location **a**

Second step of its task:

- origin: AGV Pick Location **a**

- destination: AGV Drop Location **g**

AGV 2 (green)

First step of its task:

- origin: **z #2** node

- destination: AGV Pick Location **l**

Second step of its task:

- origin: AGV Pick Location **l**

- destination: AGV Drop Location **k**

AGV 3 (blue)

First step of its task:

- origin: **g** node

- destination: AGV Pick Location **s #2**

Second step of its task:

- origin: AGV Pick Location **s #2**

- destination: AGV Drop Location **x**

The paths obtained for the three AGV are illustrated in Figure 5.8. It is possible to verify that AGV 3 made parts of the path of AGV 1 and 2 and that the AGV 1 and AGV 2 had the **z #2** node in common, however as the passages by the common parts occurred at different times, the vehicles did not collide.

The detailed maps obtained for each AGV are found in Table 5.1.

In the second step of the AGV 1 task, it was detected that the **q** node was already in the open list, because starting from **a** and with objective to go to **g**, analyzing the path through [**c**, **d**] the **d** has as its neighbour the **q** and analyzing the path by [**c**, **r**] the **r** also has the neighbour **q**, hence the warning shown in Figure 5.9. Looking to this figure it is possible to verify that the one that has lower cost is the path [**z #2**, **b**, **c**, **d**, **q**]. However, when the costs were analyzed with all neighbours of **d**, it was verified that it is more advantageous to go to **e** than to **q**, that is [**z #2**, **b**, **c** , **d**, **e**].

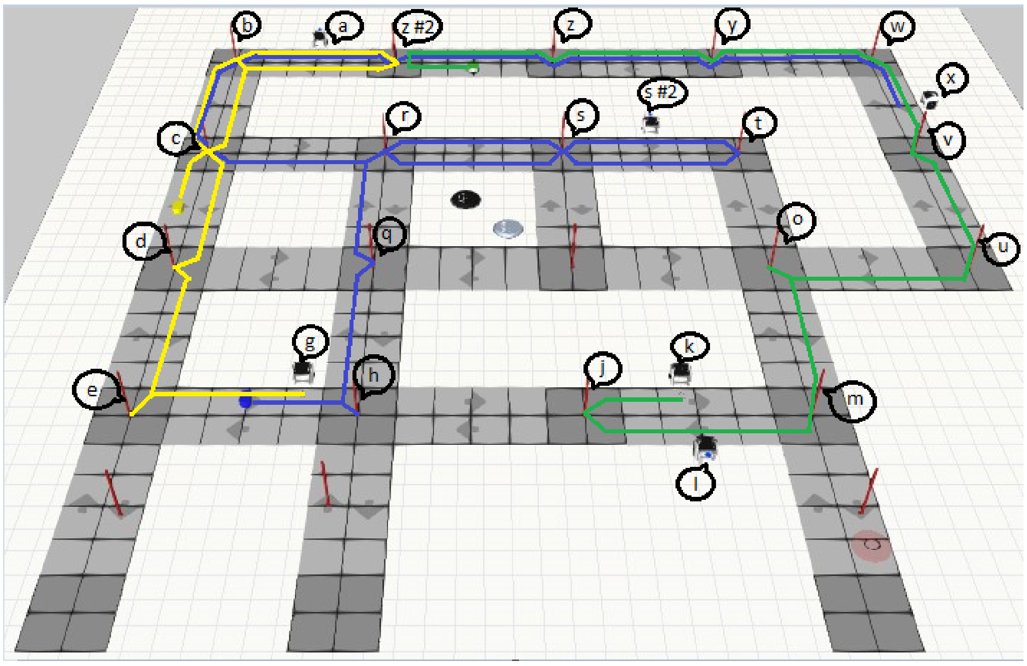
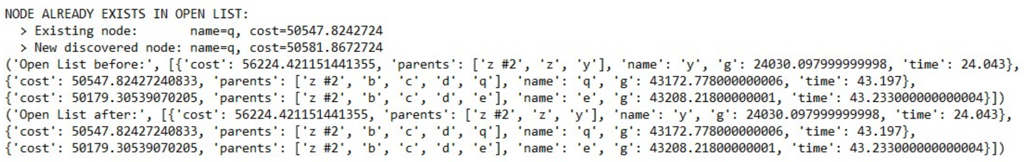


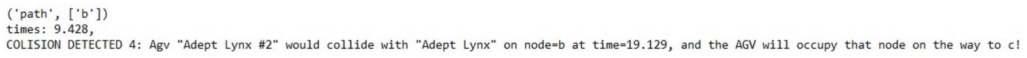
Figure 5.8: Travelled paths by the three AGV (TEA* model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=q, cost=50547.8242724
  > New discovered node: name=q, cost=50581.8672724
('Open List before:', [{'cost': 56224.421151441355, 'parents': ['z #2', 'z', 'y'], 'name': 'y', 'g': 24030.097999999998, 'time': 24.043},
{'cost': 50547.82427240833, 'parents': ['z #2', 'b', 'c', 'd', 'q'], 'name': 'q', 'g': 43172.778000000006, 'time': 43.197},
{'cost': 50179.30539070205, 'parents': ['z #2', 'b', 'c', 'd', 'e'], 'name': 'e', 'g': 43208.21800000001, 'time': 43.233000000000004}])
('Open List after:', [{'cost': 56224.421151441355, 'parents': ['z #2', 'z', 'y'], 'name': 'y', 'g': 24030.097999999998, 'time': 24.043},
{'cost': 50547.82427240833, 'parents': ['z #2', 'b', 'c', 'd', 'q'], 'name': 'q', 'g': 43172.778000000006, 'time': 43.197},
{'cost': 50179.30539070205, 'parents': ['z #2', 'b', 'c', 'd', 'e'], 'name': 'e', 'g': 43208.21800000001, 'time': 43.233000000000004}])
```

Figure 5.9: **q** node already in the open list (TEA* model)

In the first step of the AGV 2 task it was detected a collision with the AGV 1 at the node **b** at 19.129 s, because the AGV 2 would occupy that node in its way to the **c** node. The collision message is displayed in Figure 5.10. This collision was correctly detected, since the AGV 1 occupies the **b** node between [18.807 ; 28.727[s. Thus, AGV 2 re-planned its path to avoid this collision and instead of going through the **b** node went through the **z** node, which is also a neighbour of **z #2**.



```
('path', ['b'])
times: 9.428,
COLISION DETECTED 4: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=b at time=19.129, and the AGV will occupy that node on the way to c!
```

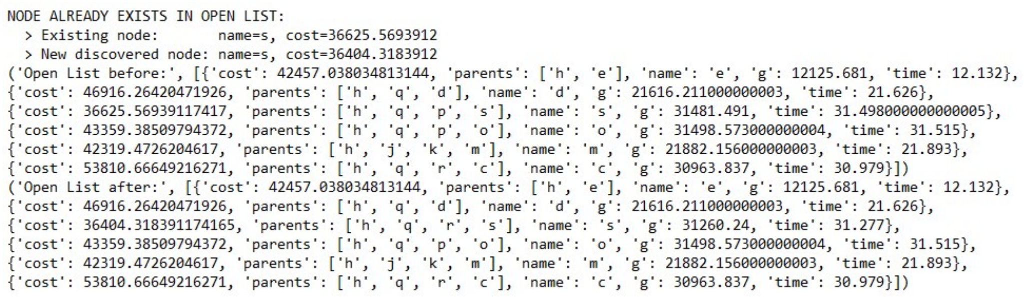
Figure 5.10: Collision message in the first step of the AGV 2 task

In the first step of the AGV 3 task, it was detected that the **s** node was already in the open list, because starting from **g** and with the objective to go to **s #2**,

Table 5.1: Maps obtained for the three AGV (TEA* model) (time units in seconds)

Steps	AGV 1	AGV 2	AGV 3
1	d: 0	z #2: 0	g: 0
	c: 9.359	z: 9.636	h: 2.743
	b: 18.807	y: 19.277	q: 12.198
	a: 28.727	w: 28.990	r: 21.579
		x: 33.549	s: 31.277
		v: 38.477	s #2: 41.374
		u: 48.129	
		o: 57.350	
		m: 66.722	
		l: 76.676	
2	z #2: 33.493	j: 81.425	t: 46.012
	b: 42.921	k: 90.730	s: 55.41
	c: 52.622		r: 64.989
	d: 62.269		c: 74.389
	e: 71.960		b: 83.837
	g: 83.962		a: 88.757
			z #2: 93.523
			z: 103.159
			y: 112.800
			w: 122.513
			x: 132.072

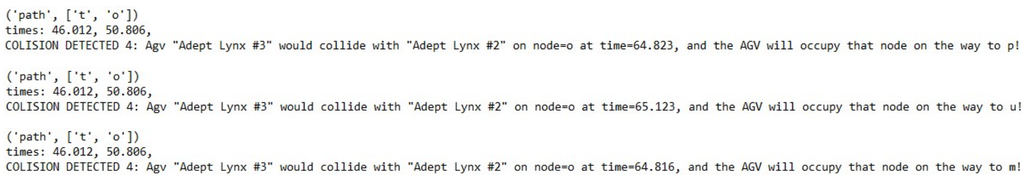
analyzing the path through [**q**, **p**] the **p** has as neighbour the **s** and analyzing the path through [**q**, **r**] the **r** also has the neighbour **s**, hence the warning shown in Figure 5.11. Looking to this figure it is possible to verify that the one that has lower cost is the path [**h**, **q**, **r**, **s**].



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=s, cost=36625.5693912
  > New discovered node: name=s, cost=36404.3183912
('Open List before:', [{'cost': 42457.038034813144, 'parents': ['h', 'e'], 'name': 'e', 'g': 12125.681, 'time': 12.132},
{'cost': 46916.26420471926, 'parents': ['h', 'q', 'd'], 'name': 'd', 'g': 21616.211000000003, 'time': 21.626},
{'cost': 36625.56939117417, 'parents': ['h', 'q', 'p', 's'], 'name': 's', 'g': 31481.491, 'time': 31.498000000000005},
{'cost': 43359.38509794372, 'parents': ['h', 'q', 'p', 'o'], 'name': 'o', 'g': 31498.573000000004, 'time': 31.515},
{'cost': 42319.4726204617, 'parents': ['h', 'j', 'k', 'm'], 'name': 'm', 'g': 21882.156000000003, 'time': 21.893},
{'cost': 53810.66649216271, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979}])
('Open List after:', [{'cost': 42457.038034813144, 'parents': ['h', 'e'], 'name': 'e', 'g': 12125.681, 'time': 12.132},
{'cost': 46916.26420471926, 'parents': ['h', 'q', 'd'], 'name': 'd', 'g': 21616.211000000003, 'time': 21.626},
{'cost': 36404.318391174165, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277},
{'cost': 43359.38509794372, 'parents': ['h', 'q', 'p', 'o'], 'name': 'o', 'g': 31498.573000000004, 'time': 31.515},
{'cost': 42319.4726204617, 'parents': ['h', 'j', 'k', 'm'], 'name': 'm', 'g': 21882.156000000003, 'time': 21.893},
{'cost': 53810.66649216271, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979}])
```

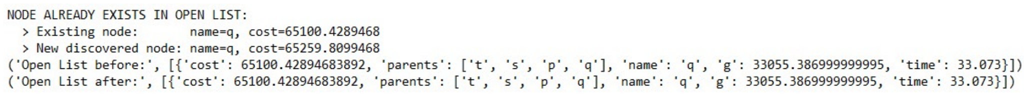
Figure 5.11: **s** node already in the open list (TEA* model)

In the second step of the AGV 3 task, three collisions were detected with AGV 2 at **o** node: (*i*) at 64.823 s, because AGV 3 would occupy that node in the path to **p**; (*ii*) at 65.123 s, because AGV 3 would occupy that node in the path to **u**; and (*iii*) at 64.816 s, because AGV 3 would occupy that node in the path to **m**. The collision messages are displayed in Figure 5.12. The three collisions were correctly detected, since AGV 2 occupies the **o** node between [57.350 ; 66.722[s. According to the observed collisions, the AGV 3 re-planned its path to avoid these collisions, and instead of going through the **o** node it went by the **s** node that is also the neighbour of **t**. In this second step of the AGV 3 task was also detected: (*i*) that the **q** node was already in the open list, because starting from **s #2** and with the objective to go to **x**, analyzing the path through [**t**, **s**, **p**] (and not for [**t**, **o**], due to the mentioned collision), the **p** has as neighbour the **q** and analyzing the path through [**t**, **s**, **r**] the **r** also has the neighbour **q**, hence the warning shown in Figure 5.13. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**, **q**]. However, when the costs were analyzed with all neighbours of **r**, it was verified that it is more advantageous to go to **c** than to do [**t**, **s**, **p**, **q**]; and (*ii*) that the node **d** was already in the open list, because starting from **s #2** and with the objective to go to **x**, analyzing the path through [**t**, **s**, **p**, **q**] (and not for [**t**, **o**], due to the mentioned collision), the **q** has as neighbour the **d**, and analyzing the path through [**t**, **s**, **r**, **c**] the **c** also has the neighbour **d**, hence the warning shown in Figure 5.14. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**, **q**, **d**]. However, when the costs were analyzed with all neighbours of **r**, it was verified that it is more advantageous to go to **c** than to do [**t**, **s**, **p**, **q**, **d**].



```
('path', ['t', 'o'])
times: 46.012, 50.806,
COLISION DETECTED 4: Agv "Adept Lynx #3" would collide with "Adept Lynx #2" on node=o at time=64.823, and the AGV will occupy that node on the way to p!

('path', ['t', 'o'])
times: 46.012, 50.806,
COLISION DETECTED 4: Agv "Adept Lynx #3" would collide with "Adept Lynx #2" on node=o at time=65.123, and the AGV will occupy that node on the way to u!

('path', ['t', 'o'])
times: 46.012, 50.806,
COLISION DETECTED 4: Agv "Adept Lynx #3" would collide with "Adept Lynx #2" on node=o at time=64.816, and the AGV will occupy that node on the way to m!
```

Figure 5.12: Collision messages in the second step of the AGV 3 task

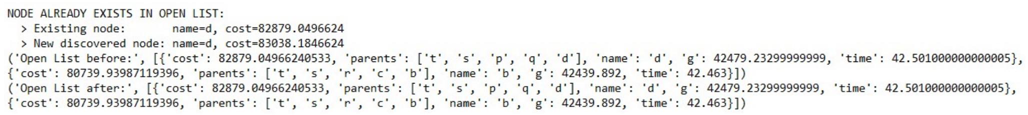


```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=q, cost=65100.4289468
  > New discovered node: name=q, cost=65259.8099468
('Open List before:', [{'cost': 65100.42894683892, 'parents': ['t', 's', 'p', 'q'], 'name': 'q', 'g': 33055.386999999995, 'time': 33.073}])
('Open List after:', [{'cost': 65100.42894683892, 'parents': ['t', 's', 'p', 'q'], 'name': 'q', 'g': 33055.386999999995, 'time': 33.073}])
```

Figure 5.13: **q** node again already in the open list (TEA* model)

The total costs of the task of each AGV were:

- AGV 1: 73 920 mm
- AGV 2: 80 685 mm



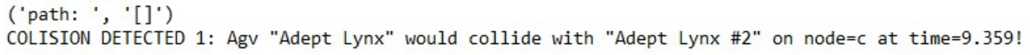
```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=d, cost=82879.0496624
  > New discovered node: name=d, cost=83038.1846624
('Open List before:', [{'cost': 82879.04966240533, 'parents': ['t', 's', 'p', 'q', 'd'], 'name': 'd', 'g': 42479.23299999999, 'time': 42.501000000000005},
{'cost': 80739.93987119396, 'parents': ['t', 's', 'r', 'c', 'b'], 'name': 'b', 'g': 42439.892, 'time': 42.463}])
('Open List after:', [{'cost': 82879.04966240533, 'parents': ['t', 's', 'p', 'q', 'd'], 'name': 'd', 'g': 42479.23299999999, 'time': 42.501000000000005},
{'cost': 80739.93987119396, 'parents': ['t', 's', 'r', 'c', 'b'], 'name': 'b', 'g': 42439.892, 'time': 42.463}])
```

Figure 5.14: **d** node already in the open list (TEA* model)

- AGV 3: 122 007 mm

In view of the results obtained, it is concluded that the objectives were fulfilled and the task with lower cost corresponds to AGV 1, according to the presented values and with Figure 5.8 that illustrates the paths that the AGV made.

A characteristic of this algorithm, mentioned in Chapter 3, refers that when is executed the path of the first AGV, in order to avoid deadlocks, the initial position nodes of the other AGV are placed as obstacles only in the first two temporal layers. Thus, confronting the map of Table 5.1, two simulations were performed to test this condition: (*i*) simulation (file `TEAstar_Simulation_v1.vcmx`) where AGV 1 was placed in the **d** node, AGV 2 in **c** node and AGV 3 in **g** node. This way, a collision of the AGV 1 with the 2 at node **c** must be detected; and (*ii*) simulation (file `TEAstar_Simulation_v2.vcmx`) where AGV 1 was placed in **d** node, AGV 2 in **b** node and AGV 3 in **g** node. Therefore, no collision of the AGV 1 with the 2 in the **b** node must be detected. The results obtained were as expected, since in the first simulation the collision was detected, as can be confirmed by Figure 5.15, and in the second not. Figure 5.16 presents the layouts of the simulated models.



```
('path: ', '[]')
COLISION DETECTED 1: Agv "Adept Lynx" would collide with "Adept Lynx #2" on node=c at time=9.359!
```

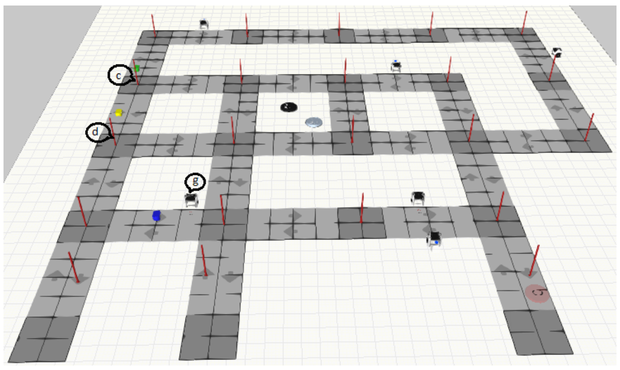
Figure 5.15: Collision message in the first step of the AGV 1 task (TEA* model)

Finally, a simulation (file `TEAstar_Simulation_v3.vcmx`) was performed, with the layout shown in Figure 5.17, in which the AGV can not move due to collisions impossible to solve, that is, it is impossible to re-plan paths to the vehicles. Collision messages are displayed in Figure 5.18. Because it was not possible to re-plan paths without collisions, the message of Figure 5.19 was presented.

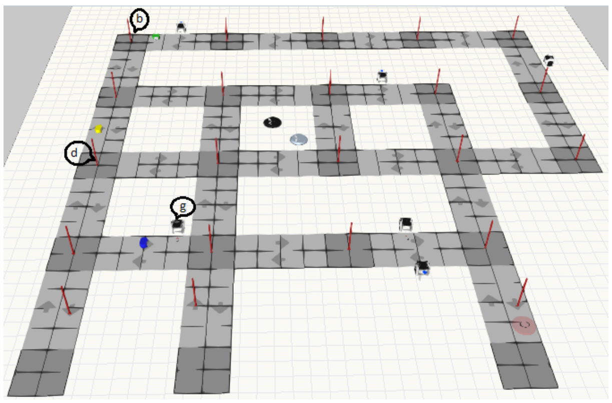
The maps obtained for each AGV are found in Table 5.2.

In the first step of the AGV 1 task a collision with the AGV 2 at the **c** node was detected at 9.359 s. This collision was correctly detected, since AGV 2 occupies the **c** node between [0 ; 9.448[s.

In the first step of the AGV 2 task three collisions were detected with AGV 1: (*i*) in **d** node at 9.647 s. This collision was correctly detected, since AGV 1



(a)



(b)

Figure 5.16: Layout of the simulation that detects the collision (a) and layout of the simulation that does not detect the collision (b)

occupies the **d** node between [0 ; 9.655[s; (*ii*) in the **r** node at 19.337 s, because AGV 2 would occupy that node in the path to the **q** node. This collision was correctly detected, since AGV 1 occupies the **r** node between [19.036 ; 28.436[s; and (*iii*) in the **r** node at 19.416 s, because AGV 2 would occupy that node in the path to the **s** node. This collision was correctly detected, since AGV 1 occupies the node **r** between [19.036 ; 28.436[s.

In the first step of the AGV 3 task two collisions were detected with AGV 1: (*i*) in **q** node at 9.655 s. This collision was correctly detected, since AGV 1 occupies the **q** node between [9.655 ; 19.036[s; (*ii*) in **r** node at 19.209 s. This collision was correctly detected, since AGV 1 occupies the **r** node between [19.036 ; 28.436[s;

In the second step of the AGV 3 task two collisions were detected: (*i*) with AGV 2 at **v** node at 62.553 s, because AGV 3 would occupy that node in the path to **w** node. This collision was correctly detected, since AGV 2 occupies the **v** node of [57.611 ; 67.263[s; (*ii*) with AGV 1 at **b** node at 62.190 s. This

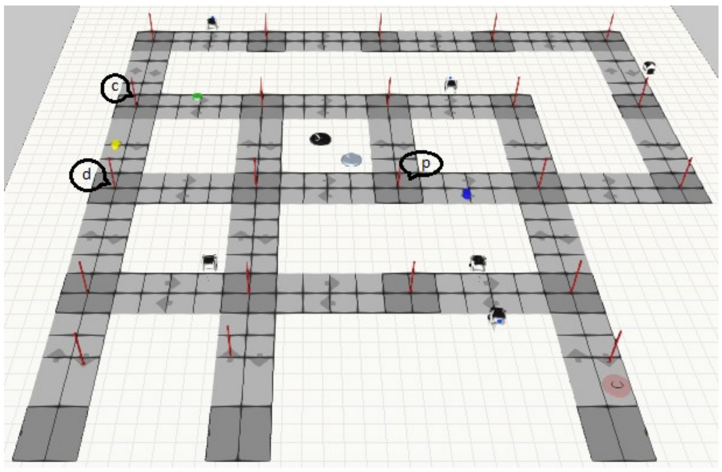
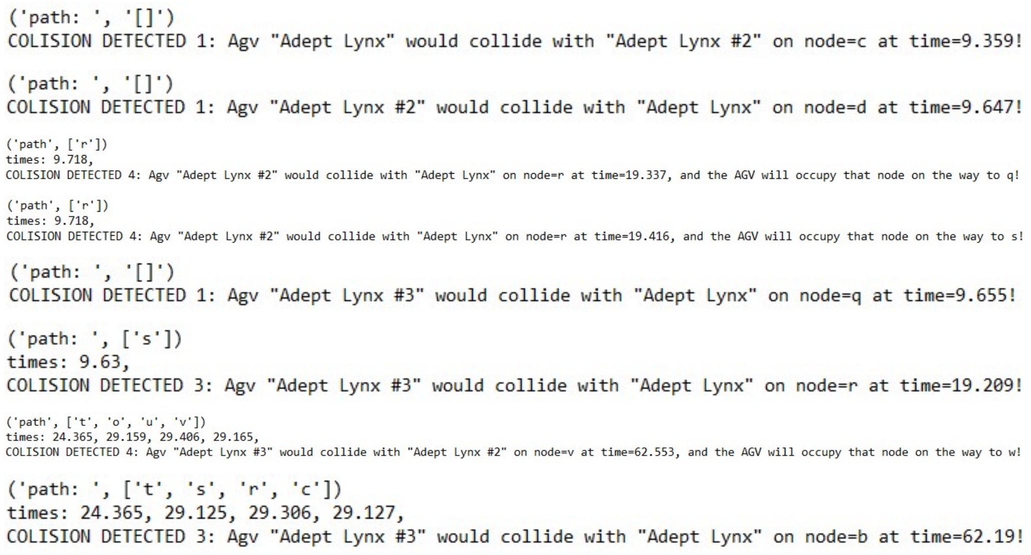
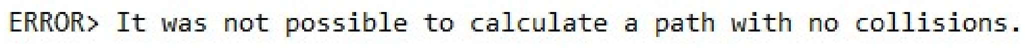


Figure 5.17: Layout and AGV positions



```
('path: ', '[]')
COLISION DETECTED 1: Agv "Adept Lynx" would collide with "Adept Lynx #2" on node=c at time=9.359!

('path: ', '[]')
COLISION DETECTED 1: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=d at time=9.647!

('path', ['r'])
times: 9.718,
COLISION DETECTED 4: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=r at time=19.337, and the AGV will occupy that node on the way to q!

('path', ['r'])
times: 9.718,
COLISION DETECTED 4: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=r at time=19.416, and the AGV will occupy that node on the way to s!

('path: ', '[]')
COLISION DETECTED 1: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=q at time=9.655!

('path: ', ['s'])
times: 9.63,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=r at time=19.209!

('path', ['t', 'o', 'u', 'v'])
times: 24.365, 29.159, 29.406, 29.165,
COLISION DETECTED 4: Agv "Adept Lynx #3" would collide with "Adept Lynx #2" on node=v at time=62.553, and the AGV will occupy that node on the way to w!

('path: ', ['t', 's', 'r', 'c'])
times: 24.365, 29.125, 29.306, 29.127,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=b at time=62.19!
```

Figure 5.18: Collision messages (TEA* model)



```
ERROR> It was not possible to calculate a path with no collisions.
```

Figure 5.19: Error message

collision was correctly detected, since AGV 1 occupies the **b** node between [61.998 ; 71.699[s in the second step of its task.

The AGV 1 had to replan and checked that the most advantageous was to go by **q** (neighbour of **d**). The AGV 2 had to replan and due to collisions, verified that it could only go through **b**. The AGV 3 had to replan and verified that the

Table 5.2: Maps obtained for the AGV, with collisions impossible to solve (TEA* model) (time units in seconds)

Steps	AGV 1	AGV 2	AGV 3
1	d: 0 q: 9.655 r: 19.036 c: 28.436 b: 37.884 a: 47.804	c: 0 b: 9.448 a: 14.368 z #2: 19.134 z: 28.770 y: 38.411 w: 48.124 x: 52.683 v: 57.611 u: 67.263 o: 76.484 m: 85.856 l: 95.810	p: 0 s: 9.630 s #2: 19.727
2	z #2: 52.570 b: 61.998 c: 71.699 d: 81.346 e: 91.037 g: 103.039	j: 100.559 k: 109.864	

solution it had was to go through [**p**, **s**, **s #2**]. The AGV 3, in the second step of its task, tried to replan its route, however, it could not find an alternative route, because all possible hypotheses did not allow it to reach the destination. The hypotheses studied after the collisions were: [**t**, **o**, **m**, **l**, **j**, **h**], [**t**, **s**, **p**, **q**, **d**], [**t**, **o**, **m**, **l**, **j**, **h**, **e**] and [**t**, **o**, **m**, **l**, **j**, **h**, **e**, **g**].

5.2 Task Scheduling Algorithms

In this section the simulated models encompass the TEA* trajectory planning algorithm and its respective scheduling strategy. The objective was to allow the AGV to perform the intended tasks, traveling the lowest possible distance in its journeys.

5.2.1 Shorter Distance Rule

In the previous chapter three strategies for this purpose were mentioned. The models of these strategies are composed by the same layout, as depicted in Figure 5.7, and contain the same components as the TEA* model. Next, the tests performed and the results obtained for each strategy are presented.

Strategy 1

The objective of the model (file `TEAstar_Simulation_strategy1.vcmx`) was to simulate the TEA* algorithm with the scheduling algorithm of this strategy. Thus, the following tasks were attributed to the AGV:

- AGV 1: (*i*) load a box in the AGV Pick Location (**a**) and unload it in the AGV Drop Location (**g**); (*ii*) load a box in the AGV Pick Location (**l**) and unload it in the AGV Drop Location (**k**); and (*iii*) load a box in the AGV Pick Location (**s #2**) and unload it in the AGV Drop Location (**x**).
- AGV 2: load a box in the AGV Pick Location (**l**) and unload it in the AGV Drop Location (**k**).
- AGV 3: load a box in the AGV Pick Location (**s #2**) and unload it in the AGV Drop Location (**x**).

Each AGV has to perform all the tasks on its list and always choose the one that is closest. In all its journeys has to travel the smallest possible distance, do not pass in points that has already visited and in the points **f**, **i**, **n** and avoid collisions with the other vehicles.

Through Figure 5.20 it is possible to verify that AGV 1 starts by performing the task (*iii*), then do (*ii*), however when performing task (*ii*) encounters AGV 2 in the AGV Drop Location **k**, because AGV 2 has finished its task here. Thus, AGV 1 stops there because it has detected AGV 2 and the TEA* algorithm is not developed to place the node where each AGV ends as occupied. If it was, it would give a collision message of AGV 1 with AGV 2 in node **k**. AGV 2 and 3 will just perform the only task assigned to them.

<u>AGV 1</u> (yellow)

First step of the first task:

- origin: node **t**

- destination: AGV Pick Location **s #2**

Second step of the first task:

- origin: AGV Pick Location **s #2**

- destination: AGV Drop Location **x**

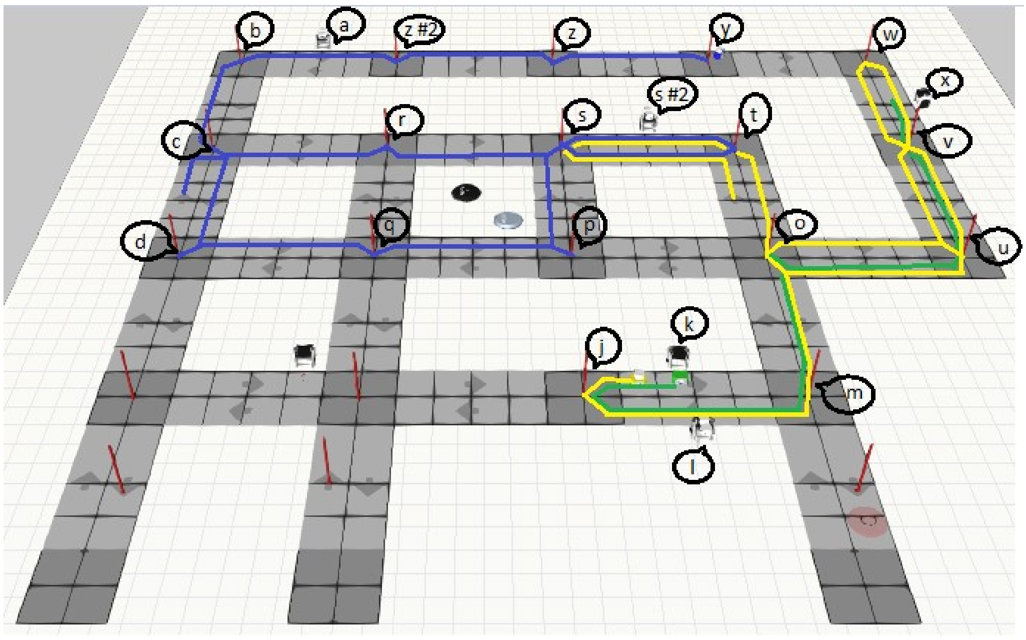


Figure 5.20: Travelled paths by the three AGV when the AGV 1 encounter the 2 (strategy 1 model)

First step of the second task:

- origin: node **x**

- destination: AGV Pick Location **l**

Second step of the second task:

- origin: AGV Pick Location **l**

- destination: AGV Drop Location **k**

First step of the third task:

- origin: node **k**

- destination: AGV Pick Location **a**

Second step of the third task:

- origin: AGV Pick Location **a**

- destination: AGV Drop Location **g**

<u>AGV 2</u> (green)

First step of its task:

- origin: node **x**

- destination: AGV Pick Location **l**

Second step of its task:

- origin: AGV Pick Location **l**

- destination: AGV Drop Location **k**

<u>AGV 3</u> (blue)

First step of its task:

- origin: node **c**

- destination: AGV Pick Location **s #2**

Second step of its task:

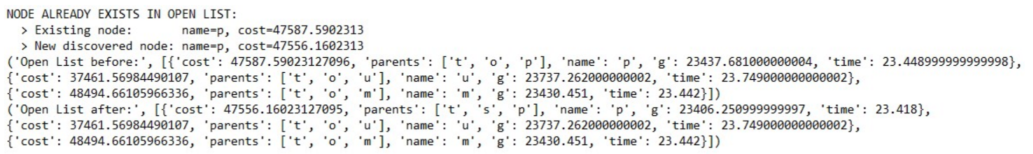
- origin: AGV Pick Location **s #2**

- destination: AGV Drop Location **x**

The paths obtained are also depicted in Figure 5.20, and it can be seen that AGV 3 made parts of the path of AGV 1 and that AGV 1 made parts of the path of AGV 2, but since the passages through the common segments occurred at different times, the vehicles did not collide.

The maps obtained for each AGV are found in Table 5.3.

In the second step of the AGV 1 first task it was detected that the **p** node was already in the open list, because starting from **s #2** and with objective to go to **x**, analyzing the path through [**t**, **o**] the **o** has as its neighbour **p** and analyzing the path through [**t**, **s**] o **s** also has the neighbour **p**, hence the warning shown in Figure 5.21. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**]. However, when the costs were analyzed with all neighbours of **t**, it was verified that it is more advantageous to do [**t**, **o**, **u**] than [**t**, **s**, **p**].



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=p, cost=47587.5902313
  > New discovered node: name=p, cost=47556.1602313
('Open List before:', [{'cost': 47587.59023127096, 'parents': ['t', 'o', 'p'], 'name': 'p', 'g': 23437.681000000004, 'time': 23.448999999999998},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
('Open List after:', [{'cost': 47556.16023127095, 'parents': ['t', 's', 'p'], 'name': 'p', 'g': 23406.250999999997, 'time': 23.418},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
```

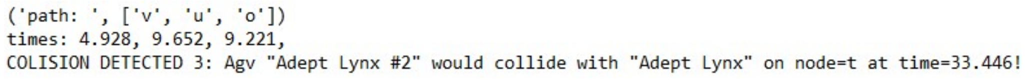
Figure 5.21: **p** node already in the open list (strategy 1 model)

In the first step of the AGV 2 task it was detected a collision of AGV 2 with AGV 1 at the **t** node at 33.446 s. The collision message is displayed in Figure 5.22. This collision was correctly detected, since AGV 1 occupies the node **t** between [24.133 ; 33.565[s in the second step of its first task. Thus, AGV 2 re-planned its path to avoid this collision and instead of going through the **t** node, analyzed the costs of **o** neighbours and chose to go through the **m** node.

Table 5.3: AGV maps of the case when the AGV 1 encounters the AGV 2 (strategy 1 model) (time units in seconds)

Tasks	Steps	AGV 1	AGV 2	AGV 3
1	1	t: 0 s: 9.398 s #2: 19.495	x: 0 v: 4.928 u: 14.580 o: 23.801 m: 33.173 l: 43.127	c: 0 d: 9.647 q: 19.302 p: 28.972 s: 38.602 s #2: 48.699
	2	t: 24.133 o: 33.565 u: 43.244 v: 52.682 w: 62.321 x: 71.880	j: 47.876 k: 57.181	t: 53.337 s: 62.735 r: 72.314 c: 81.714 b: 91.162 a: 96.082 z #2: 100.848 z: 110.484 y: 120.125
2	1	v: 76.808 u: 86.460 o: 95.681 m: 105.053 l: 115.007		
	2	j: 119.756		

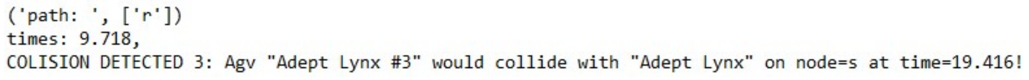
In the first step of the AGV 3 task two collisions of AGV 3 were detected with AGV 1: (*i*) at **s** node at 19.416 s. The collision message is displayed in Figure 5.23. This collision was correctly detected, since AGV 1 occupies the node **s** between [9.398 ; 19.495[s. Thus, AGV 3 re-planned its path to avoid this collision and instead of doing [**c**, **r**, **s**], verified that it was more advantageous to choose the path [**c** , **d**]; (*ii*) at **o** node at 38.619 s. The collision message is displayed in Figure 5.24. This collision was correctly detected, since AGV 1 occupies the **o** node between [33.565 ; 43.244[s. Thus, AGV 3 re-planned its path to avoid this collision and instead of going through the **o** node, analyzed the costs of **p** neighbours and chose to go through the **s** node. In this first step of the AGV



```
('path: ', ['v', 'u', 'o'])
times: 4.928, 9.652, 9.221,
COLISION DETECTED 3: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=t at time=33.446!
```

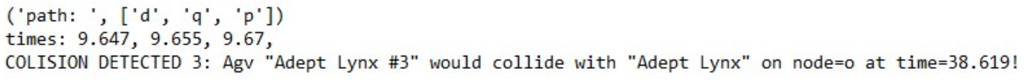
Figure 5.22: Collision message in the first step of the AGV 2 task (strategy 1 model)

3 task it was also detected that the **q** node was already in the open list, because starting from **c** and with objective to go to **s #2**, analyzing the path through **r** (that has as neighbour **q**), and analyzing the path through **d** (that also has **q** as neighbour), hence the warning shown in Figure 5.25. Looking to this figure it is possible to verify that the one that has lower cost is the path [**d**, **q**].



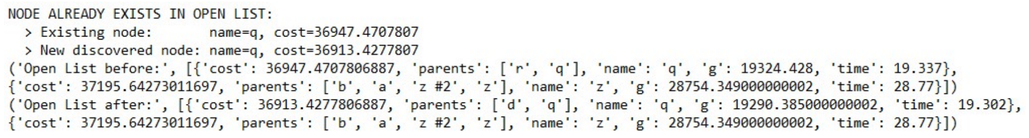
```
('path: ', ['r'])
times: 9.718,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=s at time=19.416!
```

Figure 5.23: First collision message in the first step of the AGV 3 task



```
('path: ', ['d', 'q', 'p'])
times: 9.647, 9.655, 9.67,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=o at time=38.619!
```

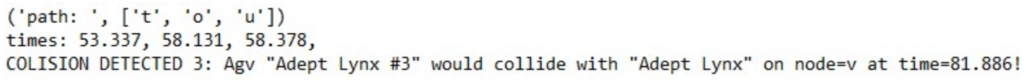
Figure 5.24: Second collision message in the first step of the AGV 3 task



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=q, cost=36947.4707807
  > New discovered node: name=q, cost=36913.4277807
('Open List before:', [{'cost': 36947.4707806887, 'parents': ['r', 'q'], 'name': 'q', 'g': 19324.428, 'time': 19.337},
{'cost': 37195.64273011697, 'parents': ['b', 'a', 'z #2', 'z'], 'name': 'z', 'g': 28754.349000000002, 'time': 28.77}])
('Open List after:', [{'cost': 36913.4277806887, 'parents': ['d', 'q'], 'name': 'q', 'g': 19290.385000000002, 'time': 19.302},
{'cost': 37195.64273011697, 'parents': ['b', 'a', 'z #2', 'z'], 'name': 'z', 'g': 28754.349000000002, 'time': 28.77}])
```

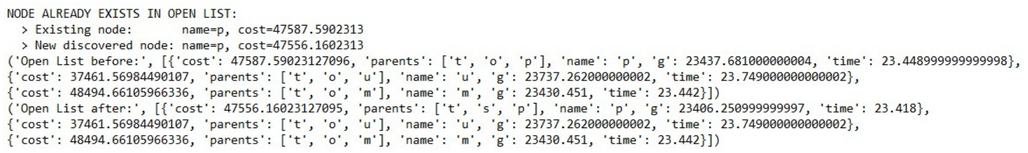
Figure 5.25: **q** node already in the open list (strategy 1 model)

In the second step of the AGV 3 task a collision of the AGV 3 with AGV 1 at the **v** node was detected at 81.886 s. The collision message is displayed in Figure 5.26. This collision was correctly detected, since AGV 1 occupies the node **v** between [52.682 ; 62.321[s. Thus, AGV 3 re-planned its path to avoid this collision and instead of doing [**t**, **o**, **u**, **v**], verified that it was more advantageous to choose the path [**t**, **s**]. In this second step of the AGV 3 task it was also detected that: (*i*) the **p** node was already in the open list, because starting from **s #2** and with objective to go to **x**, analyzing the path through [**t**, **o**] the **o** has as its neighbour the **p** and analyzing the path through [**t**, **s**] the **s** also has the neighbour **p**, hence the warning shown in Figure 5.27. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**]. However, when the costs were analyzed with all neighbours of **t**, it was verified that it is more advantageous to do [**t**, **s**, **r**] than [**t**, **s**, **p**]; (*ii*) the **q** node was already in the open list, because starting from **s # 2** and with objective to go to **x**, analyzing the path through [**t**, **s**, **p**] the **p** has as its neighbour the **q** and analyzing the path through [**t**, **s**, **r**] the **r** also has the neighbour **q**, hence the warning shown in Figure 5.28. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**, **q**]. However, when the costs were analyzed with all neighbours of **t**, it was verified that it is more advantageous to do [**t**, **s**, **r**, **c**] than [**t**, **s**, **p**, **q**]; (*iii*) the **d** node was already in the open list, because starting from **s #2** and with objective to go to **x**, analyzing the path through [**t**, **s**, **p**, **q**] the **q** has as its neighbour the **d** and analyzing the path through [**t**, **s**, **r**, **c**] the **c** also has the neighbour **d**, hence the warning shown in Figure 5.29. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**, **q**, **d**]. However, when the costs were analyzed with all neighbours of **t**, it was verified that it is more advantageous to do [**t**, **s**, **r**] than [**t**, **s**, **p**, **q**, **d**]; and (*iv*) the **e** node was already in the open list, because starting from **s #2** and with objective to go to **x**, analyzing the path through [**t**, **o**, **m**, **l**, **j**, **h**] the **h** has as its neighbour the **e** and analyzing the path through [**t**, **s**, **p**, **q**, **d**] the **d** also has the neighbour **e**, hence the warning shown in Figure 5.30. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **o**, **m**, **l**, **j**, **h**, **e**]. However, when the costs were analyzed with all neighbours of **t**, it was verified that it is more advantageous to do [**t**, **s**, **r**] than [**t**, **o**, **m**, **l**, **j**, **h**, **e**].



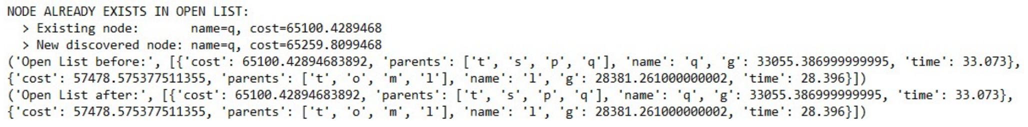
```
('path: ', ['t', 'o', 'u'])
times: 53.337, 58.131, 58.378,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=v at time=81.886!
```

Figure 5.26: Collision message in the second step of the AGV 3 task



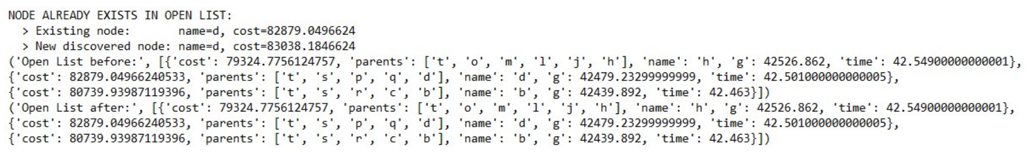
```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=p, cost=47587.5902313
  > New discovered node: name=p, cost=47556.1602313
('Open List before:', [{'cost': 47587.59023127096, 'parents': ['t', 'o', 'p'], 'name': 'p', 'g': 23437.681000000004, 'time': 23.448999999999998},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
('Open List after:', [{'cost': 47556.16023127095, 'parents': ['t', 's', 'p'], 'name': 'p', 'g': 23406.250999999997, 'time': 23.418},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
```

Figure 5.27: **p** node again already in the open list (strategy 1 model)



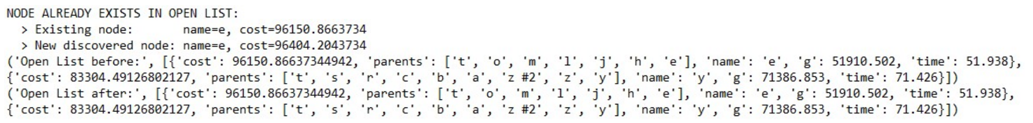
```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=q, cost=65100.4289468
  > New discovered node: name=q, cost=65259.8099468
('Open List before:', [{'cost': 65100.42894683892, 'parents': ['t', 's', 'p', 'q'], 'name': 'q', 'g': 33055.386999999995, 'time': 33.073},
{'cost': 57478.575377511355, 'parents': ['t', 'o', 'm', 'l'], 'name': 'l', 'g': 28381.261000000002, 'time': 28.396}])
('Open List after:', [{'cost': 65100.42894683892, 'parents': ['t', 's', 'p', 'q'], 'name': 'q', 'g': 33055.386999999995, 'time': 33.073},
{'cost': 57478.575377511355, 'parents': ['t', 'o', 'm', 'l'], 'name': 'l', 'g': 28381.261000000002, 'time': 28.396}])
```

Figure 5.28: **q** node again already in the open list (strategy 1 model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=d, cost=82879.0496624
  > New discovered node: name=d, cost=83038.1846624
('Open List before:', [{'cost': 79324.7756124757, 'parents': ['t', 'o', 'm', 'l', 'j', 'h'], 'name': 'h', 'g': 42526.862, 'time': 42.5490000000001},
{'cost': 82879.04966240533, 'parents': ['t', 's', 'p', 'q', 'd'], 'name': 'd', 'g': 42479.23299999999, 'time': 42.501000000000005},
{'cost': 80739.93987119396, 'parents': ['t', 's', 'r', 'c', 'b'], 'name': 'b', 'g': 42439.892, 'time': 42.463}])
('Open List after:', [{'cost': 79324.7756124757, 'parents': ['t', 'o', 'm', 'l', 'j', 'h'], 'name': 'h', 'g': 42526.862, 'time': 42.5490000000001},
{'cost': 82879.04966240533, 'parents': ['t', 's', 'p', 'q', 'd'], 'name': 'd', 'g': 42479.23299999999, 'time': 42.501000000000005},
{'cost': 80739.93987119396, 'parents': ['t', 's', 'r', 'c', 'b'], 'name': 'b', 'g': 42439.892, 'time': 42.463}])
```

Figure 5.29: **d** node already in the open list (strategy 1 model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=e, cost=96150.8663734
  > New discovered node: name=e, cost=96404.2043734
('Open List before:', [{'cost': 96150.86637344942, 'parents': ['t', 'o', 'm', 'l', 'j', 'h', 'e'], 'name': 'e', 'g': 51910.502, 'time': 51.938},
{'cost': 83304.49126802127, 'parents': ['t', 's', 'r', 'c', 'b', 'a', 'z #2', 'z', 'y'], 'name': 'y', 'g': 71386.853, 'time': 71.426}])
('Open List after:', [{'cost': 96150.86637344942, 'parents': ['t', 'o', 'm', 'l', 'j', 'h', 'e'], 'name': 'e', 'g': 51910.502, 'time': 51.938},
{'cost': 83304.49126802127, 'parents': ['t', 's', 'r', 'c', 'b', 'a', 'z #2', 'z', 'y'], 'name': 'y', 'g': 71386.853, 'time': 71.426}])
```

Figure 5.30: **e** node already in the open list (strategy 1 model)

In another model tested (file `TEAstar_Simulation_strategy1_v1.vcmx`), also with the same objective of the previous one (to simulate the TEA* algorithm with the algorithm of this scheduling strategy), the same layout was used and the AGV also were placed in the same positions, but with different tasks, in order to avoid that some AGV collide with others who had already finished all its tasks. That is, to avoid similar collisions to those mentioned in the previous test, where AGV 1 collided with AGV 2 at the AGV 2 stopping place. This time, the assigned tasks were:

- AGV 1: (*i*) load a box in the AGV Pick Location (**a**) and unload it in the AGV Drop Location (**x**); (*ii*) load a box in the AGV Pick Location (**l**) and unload it in the AGV Drop Location (**g**); and (*iii*) load a box in the AGV Pick Location (**s #2**) and unload it in the AGV Drop Location (**x**).
- AGV 2: (*i*) load a box in the AGV Pick Location (**a**) and unload it in the AGV Drop Location (**g**); and (*ii*) load a box in the AGV Pick Location (**l**) and unload it in the AGV Drop Location (**g**).
- AGV 3: load a box in the AGV Pick Location (**s #2**) and unload it in the AGV Drop Location (**k**).

Once again, each AGV has to perform all the tasks on its list and always choose for the one that is closest. In all its journeys has to travel the lowest possible displacement, do not pass in points that has already visited nor in points **f**, **i**, **n** and avoid collisions with the other vehicles.

Through Figure 5.31 it is possible to verify that AGV 1 begins by performing the task (*iii*), then does (*ii*) and finally (*i*). The AGV 2 starts with the task (*ii*) and ends with the task (*i*). AGV 3 performs just the only task assigned to it.

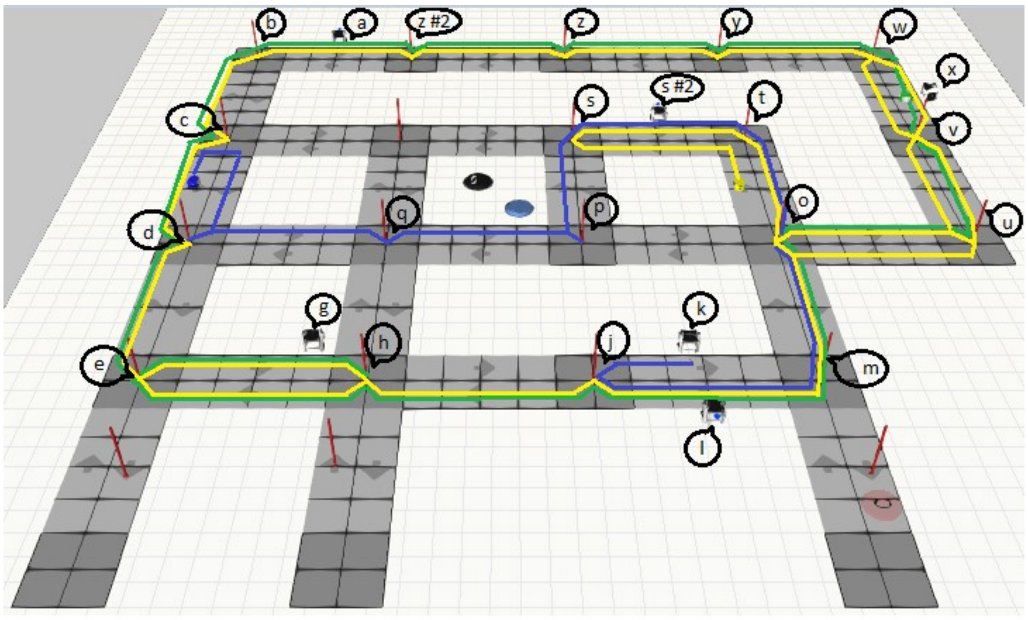


Figure 5.31: Travelled paths by the three AGV (strategy 1 of the second model)

<u>AGV 1</u> (yellow)

First step of the first task:

- origin: node **t**

- destination: AGV Pick Location **s #2**

Second step of the first task:

- origin: AGV Pick Location **s #2**

- destination: AGV Drop Location **x**

First step of the second task:

- origin: node **x**

- destination: AGV Pick Location **l**

Second step of the second task:

- origin: AGV Pick Location **l**

- destination: AGV Drop Location **g**

First step of the third task:

- origin: node **g**

- destination: AGV Pick Location **a**

Second step of the third task:

- origin: AGV Pick Location **a**

- destination: AGV Drop Location **x**

<u>AGV 2</u> (green)

First step of the first task:

- origin: node **x**

- destination: AGV Pick Location **l**

Second step of the first task:

- origin: AGV Pick Location **l**

- destination: AGV Drop Location **g**

First step of the second task:

- origin: nó **g**

- destination: AGV Pick Location **a**

Second step of the second task:

- origin: AGV Pick Location **a**

- destination: AGV Drop Location **g**

<u>AGV 3</u> (blue)

First step of its task:

- origin: node **c**

- destination: AGV Pick Location **s #2**

Second step of its task:

- origin: AGV Pick Location **s #2**

- destination: AGV Drop Location **k**

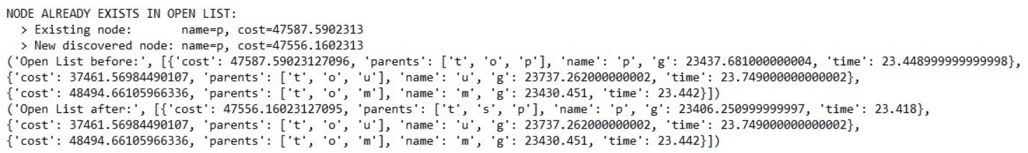
The paths obtained are also represented in Figure 5.31 and it is possible to verify that the three AGV shared parts of their paths, but as the passages through the common segments occurred at different times, the vehicles did not collide.

The maps obtained for each AGV are found in Table 5.4.

In the second step of the AGV 1 first task it was detected that the **p** node was already in the open list, because starting from **s #2** and with objective to go to **x**, analyzing the path through [**t**, **o**] the **o** has as its neighbour the **p** and analyzing the path through [**t**, **s**] the **s** also has the neighbour **p**, hence the warning shown in Figure 5.32. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**]. However, when the costs were analyzed with all

Table 5.4: AGV maps (strategy 1 of the second model) (time units in seconds)

Tasks	Steps	AGV 1	AGV 2	AGV 3
1	1	t: 0	x: 0	c: 0
		s: 9.398	v: 4.928	d: 9.647
		s #2: 19.495	u: 14.580	q: 19.302
			o: 23.801	p: 28.972
			m: 33.173	s: 38.602
			l: 43.127	s #2: 48.699
	2	t: 24.133	j: 47.876	t: 53.337
		o: 33.565	h: 57.280	o: 62.769
		u: 43.244	e: 66.669	m: 72.141
		v: 52.682	g: 78.671	l: 77.095
		w: 62.321		j: 81.844
		x: 71.880		k: 91.149
2	1	v: 76.808	h: 81.414	
		u: 86.460	e: 90.803	
		o: 95.681	d: 100.223	
		m: 105.053	c: 109.582	
		l: 115.007	b: 119.030	
			a: 128.950	
	2	j: 119.756	z #2: 133.716	
		h: 129.160	z: 143.352	
		e: 138.549	y: 152.993	
		g: 150.551	w: 162.706	
			x: 167.265	
			v: 172.193	
			u: 181.845	
			o: 191.066	
			m: 200.438	
			l: 205.392	
			j: 210.141	
			h: 219.545	
			e: 228.934	
			g: 240.936	
3	1	h: 153.294		
		e: 162.683		
		d: 172.103		
		c: 181.462		
		b: 190.910		
		a: 200.830		
	2	z #2: 205.596		
		z: 215.232		
		y: 224.873		
		w: 234.586		
		x: 244.145		

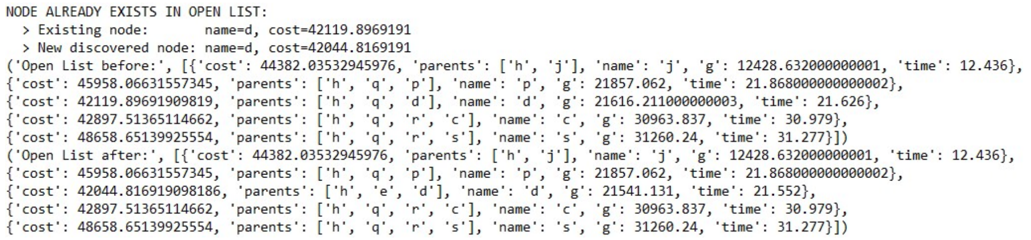


```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=p, cost=47587.5902313
  > New discovered node: name=p, cost=47556.1602313
('Open List before:', [{'cost': 47587.59023127096, 'parents': ['t', 'o', 'p'], 'name': 'p', 'g': 23437.681000000004, 'time': 23.448999999999998},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
('Open List after:', [{'cost': 47556.16023127095, 'parents': ['t', 's', 'p'], 'name': 'p', 'g': 23406.250999999997, 'time': 23.418},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
```

Figure 5.32: **p** node already in the open list (strategy 1 of the second model)

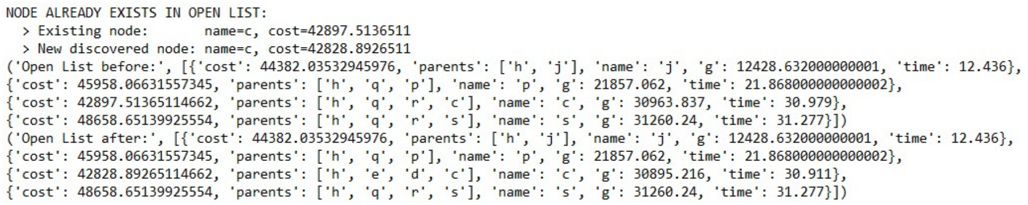
neighbours of **t**, it was verified that it is more advantageous to do [**t**, **o**, **u**] than [**t**, **s**, **p**].

In the first step of the AGV 1 third task it was detected that: (*i*) the **d** node was already in the open list, because starting from **g** and with objective to go to **a**, analyzing the path through [**h**, **q**] the **q** has as its neighbour the **d** and analyzing the path through [**h**, **e**] the **e** also has the neighbour **d**, hence the warning shown in Figure 5.33. Looking to this figure it is possible to verify that the one that has lower cost is the path [**h**, **e**, **d**]; and (*ii*) that the **c** node was also already in the open list, because starting from **g** and with objective to go to **a**, analyzing the path through [**h**, **q**, **r**], the **r** has as its neighbour the **c** and analyzing the path through [**h**, **e**, **d**] the **d** also has the neighbour **c**, hence the warning shown in Figure 5.34. Looking to this figure it is possible to verify that the one that has lower cost is the path [**h**, **e**, **d**, **c**].



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=d, cost=42119.8969191
  > New discovered node: name=d, cost=42044.8169191
('Open List before:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42119.89691909819, 'parents': ['h', 'q', 'd'], 'name': 'd', 'g': 21616.211000000003, 'time': 21.626},
{'cost': 42897.51365114662, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
('Open List after:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42044.816919098186, 'parents': ['h', 'e', 'd'], 'name': 'd', 'g': 21541.131, 'time': 21.552},
{'cost': 42897.51365114662, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
```

Figure 5.33: **d** node already in the open list (strategy 1 of the second model)

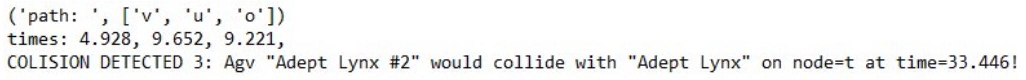


```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=c, cost=42897.5136511
  > New discovered node: name=c, cost=42828.8926511
('Open List before:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42897.51365114662, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
('Open List after:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42828.89265114662, 'parents': ['h', 'e', 'd', 'c'], 'name': 'c', 'g': 30895.216, 'time': 30.911},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
```

Figure 5.34: **c** node already in the open list (strategy 1 of the second model)

In the first step of the AGV 2 first task a collision of AGV 2 with AGV 1 at the **t** node at 33.446 s was detected. The collision message is displayed in Figure 5.35. This collision was correctly detected, since AGV 1 occupies the node **t** between

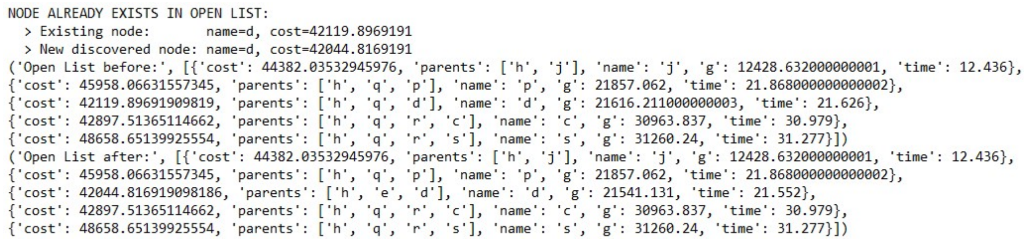
[24.133 ; 33.565[s in the second step of its first task. Thus, AGV 2 re-planned its way to avoid this collision and instead of going through the **t** node, analyzed the costs of **o** neighbours and chose to go through the **m** node.



```
('path: ', ['v', 'u', 'o'])
times: 4.928, 9.652, 9.221,
COLISION DETECTED 3: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=t at time=33.446!
```

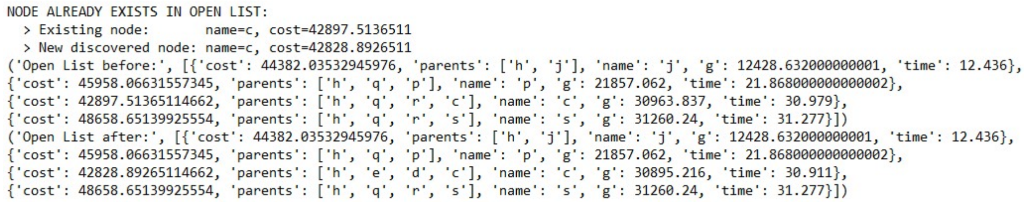
Figure 5.35: Collision message in the first step of the AGV 2 first task

In the first step of the AGV 2 second task it was detected that: (*i*) the **d** node was already in the open list, because starting from **g** and with objective to go to **a**, analyzing the path through [**h**, **q**] the **q** has as its neighbour the **d** and analyzing the path through [**h**, **e**] the **e** also has the neighbour **d**, hence the warning shown in Figure 5.36. Looking to this figure it is possible to verify that the one that has lower cost is the path [**h**, **e**, **d**]; and (*ii*) that the **c** node was also already in the open list, because starting from **g** and with objective to go to **a**, analyzing the path through [**h**, **q**, **r**], the **r** has as its neighbour the **c** and analyzing the path through [**h**, **e**, **d**] the **d** also has the neighbour **c**, hence the warning shown in Figure 5.37. Looking to this figure it is possible to verify that the one that has lower cost is the path [**h**, **e**, **d**, **c**].



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=d, cost=42119.8969191
  > New discovered node: name=d, cost=42044.8169191
('Open List before:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42119.89691909819, 'parents': ['h', 'q', 'd'], 'name': 'd', 'g': 21616.211000000003, 'time': 21.626},
{'cost': 42897.51365114662, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
('Open List after:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42044.816919098186, 'parents': ['h', 'e', 'd'], 'name': 'd', 'g': 21541.131, 'time': 21.552},
{'cost': 42897.51365114662, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
```

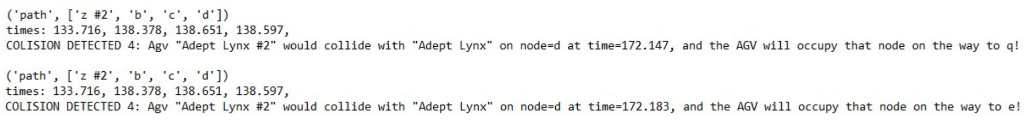
Figure 5.36: **d** node already again in the open list (strategy 1 of the second model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=c, cost=42897.5136511
  > New discovered node: name=c, cost=42828.8926511
('Open List before:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42897.51365114662, 'parents': ['h', 'q', 'r', 'c'], 'name': 'c', 'g': 30963.837, 'time': 30.979},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
('Open List after:', [{'cost': 44382.03532945976, 'parents': ['h', 'j'], 'name': 'j', 'g': 12428.632000000001, 'time': 12.436},
{'cost': 45958.06631557345, 'parents': ['h', 'q', 'p'], 'name': 'p', 'g': 21857.062, 'time': 21.868000000000002},
{'cost': 42828.89265114662, 'parents': ['h', 'e', 'd', 'c'], 'name': 'c', 'g': 30895.216, 'time': 30.911},
{'cost': 48658.65139925554, 'parents': ['h', 'q', 'r', 's'], 'name': 's', 'g': 31260.24, 'time': 31.277}])
```

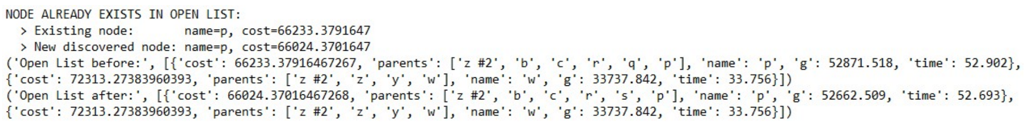
Figure 5.37: **c** node again already in the open list (strategy 1 of the second model))

In the second step of the AGV 2 second task two collisions of the AGV 2 with AGV 1 were detected at node **d**: (*i*) at 172.147 s, because AGV 2 would occupy that node in the path to **q**; (*ii*) at 172.183 s, because AGV 2 would occupy that node in the path to **e**. The collision messages are displayed in Figure 5.38. The two collisions were correctly detected, since AGV 1 occupies the node **d** between [172.103 ; 181.462[s in the first step of its third task. Thus, AGV 2 re-planned its path to avoid this collision and instead of doing [**z #2**, **b**, **c**, **d**], verified that it was more advantageous to choose the path [**z #2**, **z**]. In this second step of the AGV 2 second task it was also detected that: (*i*) the **p** node was already in the open list, because starting from **a** and with objective to go to **g**, analyzing the path through [**z #2**, **b**, **c**, **r**, **q**] the **q** has as its neighbour the **p** and analyzing the path through [**z #2**, **b**, **c**, **r**, **s**] the **s** also has the neighbour **p**, hence the warning shown in Figure 5.39. Looking to this figure it is possible to verify that the one that has lower cost is the path [**z #2**, **b**, **c**, **r**, **s**, **p**]. However, when the costs were analyzed with all neighbours of **z #2**, it was verified that it is more advantageous to go to **z** than to do [**z #2**, **b**, **c**, **r**, **s**, **p**]; (*ii*) that the **o** node was already in the open list, because starting from **a** and with objective to go to **g**, analyzing the path through [**z #2**, **b**, **c**, **r**, **s**, **p**] the **p** has as its neighbour the **o** and analyzing the path through [**z #2**, **b**, **c**, **r**, **s**, **s #2**, **t**] the **t** also has the neighbour **o**, hence the warning shown in Figure 5.40. Looking to this figure it is possible to verify that the one that has lower cost is the path [**z #2**, **b**, **c**, **r**, **s**, **p**, **o**]. However, when the costs were analyzed with all neighbours of **z #2**, it was verified that it is more advantageous to go to **z** than to do [**z #2**, **b**, **c**, **r**, **s**, **p**, **o**]; (*iii*) again, the **o** node was already in the open list, because starting from **a** and with objective to go to **g**, analyzing the path through [**z #2**, **b**, **c**, **r**, **s**, **s #2**, **t**] the **t** has as its neighbour the **o** and analyzing the path through [**z #2**, **z**, **y**, **w**, **x**, **v**, **u**] the **u** also has the neighbour **o**, hence the warning shown in Figure 5.41. Looking to this figure it is possible to verify that the one that has lower cost is the path [**z #2**, **z**, **y**, **w**, **x**, **v**, **u**, **o**].



```
('path', ['z #2', 'b', 'c', 'd'])
times: 133.716, 138.378, 138.651, 138.597,
COLISION DETECTED 4: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=d at time=172.147, and the AGV will occupy that node on the way to q!

('path', ['z #2', 'b', 'c', 'd'])
times: 133.716, 138.378, 138.651, 138.597,
COLISION DETECTED 4: Agv "Adept Lynx #2" would collide with "Adept Lynx" on node=d at time=172.183, and the AGV will occupy that node on the way to e!
```

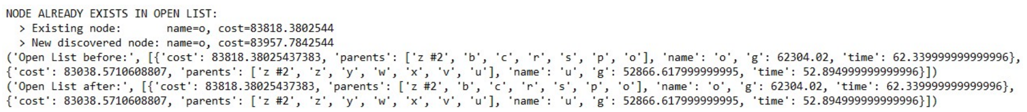
Figure 5.38: Collision message in the second step of the AGV 2 second task



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=p, cost=66233.3791647
  > New discovered node: name=p, cost=66024.3701647
('Open List before:', [{'cost': 66233.37916467267, 'parents': ['z #2', 'b', 'c', 'r', 'q', 'p'], 'name': 'p', 'g': 52871.518, 'time': 52.902},
{'cost': 72313.27383960393, 'parents': ['z #2', 'z', 'y', 'w'], 'name': 'w', 'g': 33737.842, 'time': 33.756}])
('Open List after:', [{'cost': 66024.37016467268, 'parents': ['z #2', 'b', 'c', 'r', 's', 'p'], 'name': 'p', 'g': 52662.509, 'time': 52.693},
{'cost': 72313.27383960393, 'parents': ['z #2', 'z', 'y', 'w'], 'name': 'w', 'g': 33737.842, 'time': 33.756}])
```

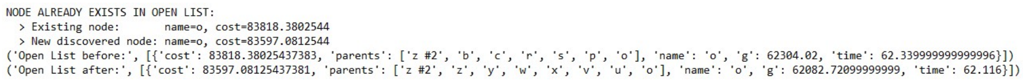
Figure 5.39: **p** node again already in the open list (strategy 1 of the second model)

In the first step of the AGV 3 task two collisions of the AGV 3 were detected with AGV 1: (*i*) at **s** node at 19.416 s. The collision message is displayed in



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=o, cost=83818.3802544
  > New discovered node: name=o, cost=83957.7842544
('Open List before:', [{'cost': 83818.38025437383, 'parents': ['z #2', 'b', 'c', 'r', 's', 'p', 'o'], 'name': 'o', 'g': 62304.02, 'time': 62.339999999999996},
{'cost': 83038.5710608807, 'parents': ['z #2', 'z', 'y', 'w', 'x', 'v', 'u'], 'name': 'u', 'g': 52866.617999999995, 'time': 52.894999999999996}])
('Open List after:', [{'cost': 83818.38025437383, 'parents': ['z #2', 'b', 'c', 'r', 's', 'p', 'o'], 'name': 'o', 'g': 62304.02, 'time': 62.339999999999996},
{'cost': 83038.5710608807, 'parents': ['z #2', 'z', 'y', 'w', 'x', 'v', 'u'], 'name': 'u', 'g': 52866.617999999995, 'time': 52.894999999999996}])
```

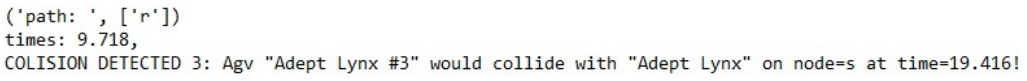
Figure 5.40: **o** node already in the open list (strategy 1 of the second model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=o, cost=83818.3802544
  > New discovered node: name=o, cost=83597.0812544
('Open List before:', [{'cost': 83818.38025437383, 'parents': ['z #2', 'b', 'c', 'r', 's', 'p', 'o'], 'name': 'o', 'g': 62304.02, 'time': 62.339999999999996}])
('Open List after:', [{'cost': 83597.08125437381, 'parents': ['z #2', 'z', 'y', 'w', 'x', 'v', 'u', 'o'], 'name': 'o', 'g': 62082.72099999999, 'time': 62.116}])
```

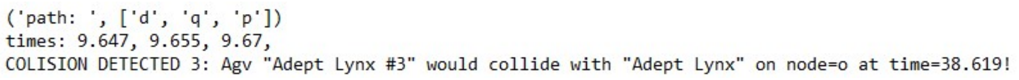
Figure 5.41: **o** node already again in the open list (strategy 1 of the second model)

Figure 5.42. This collision was correctly detected, since AGV 1 occupies the **s** node between [9.398 ; 19.495[s. Thus, AGV 3 re-planned its way to avoid this collision and instead of doing [**c**, **r**], did [**c**, **d**]; (*ii*) in the **o** node at 38.619 s. The collision message is displayed in Figure 5.43. This collision was also correctly detected, since AGV 1 occupies the **o** node of [33.565 ; 43.244[s. Thus, AGV 3 re-planned its way to avoid this collision and instead of going through the **o** node, checked that it was more advantageous to go through the node **s**, which is also neighbour of **p**. In this first step of the AGV 3 task it was also detected that the **q** node was already in the open list, because starting from **c** and with objective to go to **s #2**, analyzing the path through **r** (that has as neighbour **q**), and analyzing the path through **d** (that also has **q** as neighbour), hence the warning shown in Figure 5.44. Looking to this figure it is possible to verify that the one that has lower cost is the path [**d**, **q**].



```
('path: ', ['r'])
times: 9.718,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=s at time=19.416!
```

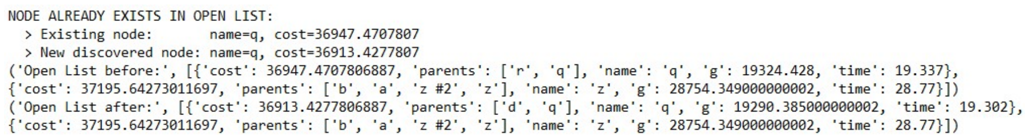
Figure 5.42: Collision message in the first step of the AGV 3 task



```
('path: ', ['d', 'q', 'p'])
times: 9.647, 9.655, 9.67,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=o at time=38.619!
```

Figure 5.43: Second collision message in the first step of the AGV 3 task

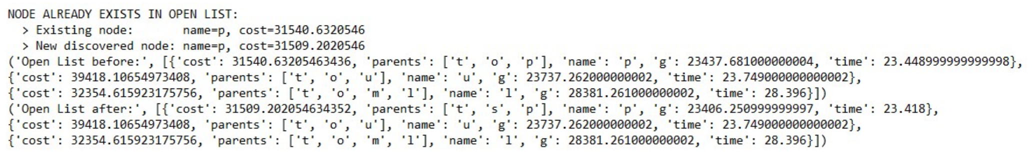
In the second step of the AGV 3 task it was detected that the **p** node was already in the open list, because starting from **s #2** and with objective to go to **k**, analyzing the path through [**t**, **o**] the **o** has as its neighbour the **p** and analyzing the path through [**t**, **s**] the **s** also has the neighbour **p**, hence the warning shown in Figure 5.45. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**]. However, when the costs were analyzed with all neighbors of **t**, it was verified that it is more advantageous to do [**t**, **o**, **m**] than



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=q, cost=36947.4707807
  > New discovered node: name=q, cost=36913.4277807
('Open List before:', [{'cost': 36947.4707806887, 'parents': ['r', 'q'], 'name': 'q', 'g': 19324.428, 'time': 19.337},
{'cost': 37195.64273011697, 'parents': ['b', 'a', 'z #2', 'z'], 'name': 'z', 'g': 28754.349000000002, 'time': 28.77}])
('Open List after:', [{'cost': 36913.4277806887, 'parents': ['d', 'q'], 'name': 'q', 'g': 19290.385000000002, 'time': 19.302},
{'cost': 37195.64273011697, 'parents': ['b', 'a', 'z #2', 'z'], 'name': 'z', 'g': 28754.349000000002, 'time': 28.77}])
```

Figure 5.44: **q** node already in the open list (strategy 1 of the second model)

[**t**, **s**, **p**].



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=p, cost=31540.6320546
  > New discovered node: name=p, cost=31509.2020546
('Open List before:', [{'cost': 31540.63205463436, 'parents': ['t', 'o', 'p'], 'name': 'p', 'g': 23437.681000000004, 'time': 23.448999999999998},
{'cost': 39418.10654973408, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 32354.615923175756, 'parents': ['t', 'o', 'm', 'l'], 'name': 'l', 'g': 28381.261000000002, 'time': 28.396}])
('Open List after:', [{'cost': 31509.202054634352, 'parents': ['t', 's', 'p'], 'name': 'p', 'g': 23406.250999999997, 'time': 23.418},
{'cost': 39418.10654973408, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 32354.615923175756, 'parents': ['t', 'o', 'm', 'l'], 'name': 'l', 'g': 28381.261000000002, 'time': 28.396}])
```

Figure 5.45: **p** node already in the open list (strategy 1 of the second model

The total costs of each task were:

- Task: Transport parts with AGV 1: 61 848 mm
- Task: Transport parts with AGV 1: 68 631 mm
- Task: Transport parts with AGV 1: 83 551 mm
- Task: Transport parts with AGV 2: 68 631 mm
- Task: Transport parts with AGV 2: 152 182 mm
- Task: Transport parts with AGV 3: 81 104 mm

In view of the obtained results, it is concluded that the objectives were fulfilled.

Strategy 2

In this strategy (file `TEAstar_Simulation_strategy2.vcmx`) there is a list with three tasks. It is verified which is the first task of the list and that it is associated to the nearest AGV. Next, it is verified which is the next task and is checked again which is the nearest AGV, and so on. The objective of the model of this strategy was to ensure that each task always choose the closest AGV and that the chosen AGV uses the TEA* planning algorithm to move to it and execute the entire path involved in performing the same. In conclusion, each task chooses the closest AGV, and the AGV performs the entire task going through the smallest path, not passing in points that it has already visited nor in the points **f**, **i**, **n**, while avoiding collisions with other vehicles.

Task list

Task 1:

- first step: move the AGV from its position to the AGV Pick Location **a**

- second step: move the AGV from the AGV Pick Location **a** to the AGV Drop Location **g**

Task 2:

- first step: move the AGV from its position to the AGV Pick Location **l**

- second step: move the AGV from the AGV Pick Location **l** to the AGV Drop Location **k**

Task 3:

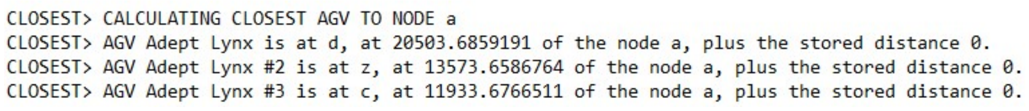
- first step: move the AGV from its position to the AGV Pick Location **s #2**

- second step: move the AGV from the AGV Pick Location **s #2** to the AGV Drop Location **x**

Comparing Figure 5.46 and Figure 5.47, it is verified that the closest AGV of task 1 is AGV 3. When this AGV finishes the execution of the first step of the task, a message is displayed showing the distance travelled until that moment (14359.605 mm). This message is shown in Figure 5.48. When the AGV finishes executing the second step of the task, the messages shown in Figure 5.49 are displayed. These messages report the distance travelled in the second step of task 1 (50206.537 mm) and the distances of each AGV to the AGV Pick Location **l**. It is verified that the closest AGV to task 2 is AGV 1. When this AGV finishes the execution of the first step of the task, a message is displayed showing the distance travelled until that moment (43272.629 mm). This message is shown in Figure 5.50. When the AGV finishes executing the second step of the task, the messages shown in Figure 5.51 are displayed. These messages inform the distance travelled in the second step of task 2 (9049.303 mm) and display the distances of each AGV to the AGV Pick Location **s # 2**. it is verified that the closest AGV to task 3 is AGV 2. When this AGV finishes the execution of the first step of the task, a message is displayed showing the distance traveled until that moment (71779.891 mm). This message is shown in Figure 5.52. When the AGV finishes the execution of the second step of the task, the messages shown in Figure 5.53 are displayed. These messages inform about the distance travelled in the second step of task 3 (47361.386 mm) and show the distances travelled (costs) in the execution of each task.

Through Figure 5.47 it is possible to verify that AGV 1 and 2 share parts of their paths and that AGV 1 and 3 share the **d** node, but since the passages through the common parts occur at different times, the vehicles did not collide.

The maps obtained for each AGV are found in Table 5.5.



```
CLOSEST> CALCULATING CLOSEST AGV TO NODE a
CLOSEST> AGV Adept Lynx is at d, at 20503.6859191 of the node a, plus the stored distance 0.
CLOSEST> AGV Adept Lynx #2 is at z, at 13573.6586764 of the node a, plus the stored distance 0.
CLOSEST> AGV Adept Lynx #3 is at c, at 11933.6766511 of the node a, plus the stored distance 0.
```

Figure 5.46: Results for the first step of task 1

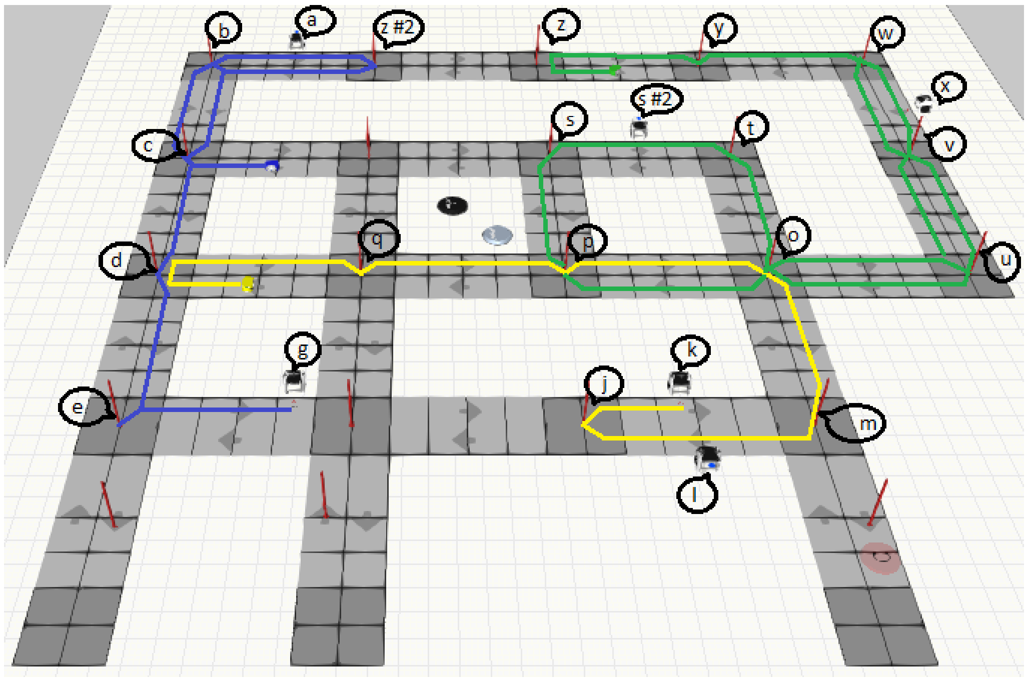
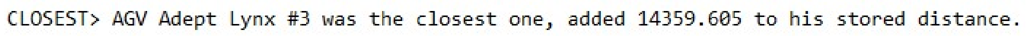
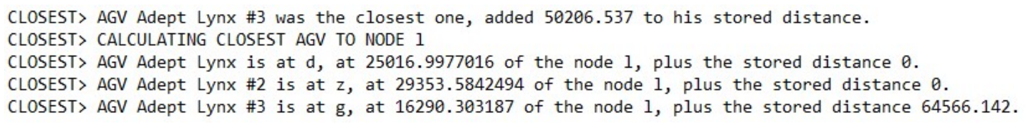


Figure 5.47: Travelled paths by the three AGV



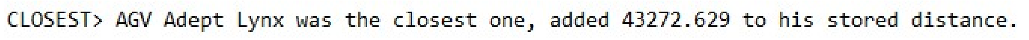
```
CLOSEST> AGV Adept Lynx #3 was the closest one, added 14359.605 to his stored distance.
```

Figure 5.48: Results when AGV 3 finished the first step of the task



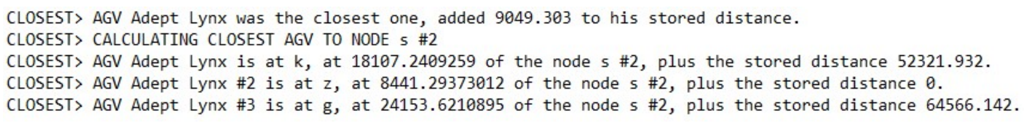
```
CLOSEST> AGV Adept Lynx #3 was the closest one, added 50206.537 to his stored distance.
CLOSEST> CALCULATING CLOSEST AGV TO NODE l
CLOSEST> AGV Adept Lynx is at d, at 25016.9977016 of the node l, plus the stored distance 0.
CLOSEST> AGV Adept Lynx #2 is at z, at 29353.5842494 of the node l, plus the stored distance 0.
CLOSEST> AGV Adept Lynx #3 is at g, at 16290.303187 of the node l, plus the stored distance 64566.142.
```

Figure 5.49: Results when AGV 3 finished the second step of the task



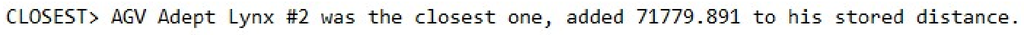
```
CLOSEST> AGV Adept Lynx was the closest one, added 43272.629 to his stored distance.
```

Figure 5.50: Results when AGV 1 finished the first step of the task



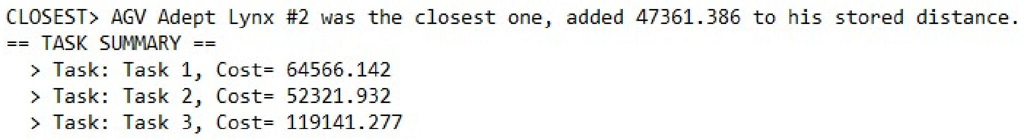
```
CLOSEST> AGV Adept Lynx was the closest one, added 9049.303 to his stored distance.
CLOSEST> CALCULATING CLOSEST AGV TO NODE s #2
CLOSEST> AGV Adept Lynx is at k, at 18107.2409259 of the node s #2, plus the stored distance 52321.932.
CLOSEST> AGV Adept Lynx #2 is at z, at 8441.29373012 of the node s #2, plus the stored distance 0.
CLOSEST> AGV Adept Lynx #3 is at g, at 24153.6210895 of the node s #2, plus the stored distance 64566.142.
```

Figure 5.51: Results when AGV 1 finished the second step of the task



```
CLOSEST> AGV Adept Lynx #2 was the closest one, added 71779.891 to his stored distance.
```

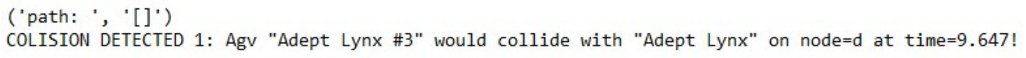
Figure 5.52: Results when AGV 2 finished the first step of the task



```
CLOSEST> AGV Adept Lynx #2 was the closest one, added 47361.386 to his stored distance.
== TASK SUMMARY ==
  > Task: Task 1, Cost= 64566.142
  > Task: Task 2, Cost= 52321.932
  > Task: Task 3, Cost= 119141.277
```

Figure 5.53: Results when AGV 2 finished the second step of the task

In the first step of the first task a collision of the AGV 3 with AGV 1 at **d** node was detected at 9.647 s. The collision message is displayed in Figure 5.54. This collision was correctly detected, since AGV 1 occupies the **d** node between [0 ; 9,655[s. Thus, AGV 3 re-planned its path to avoid this collision and instead of going through the **d** node went through the **b** node, also neighbour of **c**.



```
('path: ', '[]')
COLISION DETECTED 1: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=d at time=9.647!
```

Figure 5.54: Collision message in the first step of the first task (strategy 2 model)

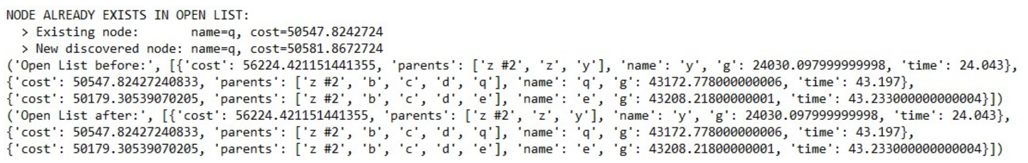
In the second step of the first task it was detected that the **q** node was already in the open list, because starting from **a** and with objective to go to **g**, analyzing the path through [**z #2**, **b**, **c**, **d**] the **d** has as its neighbour the **q** and analyzing the path through [**z #2**, **b**, **c**, **r**] the **r** also has the neighbour **q**, hence the warning shown in Figure 5.55. Looking to this figure it is possible to verify that the one that has lower cost is the path [**z #2**, **b**, **c**, **d**, **q**]. However, when the costs were analyzed with all neighbours of **d**, it was verified that it is more advantageous to do [**z #2**, **b**, **c**, **d**, **e**] than [**z #2**, **b**, **c**, **d**, **q**].

In the first step of the second task a collision of the AGV 1 with the AGV 3 at the **c** node was detected at 9.359 s. The collision message is displayed in Figure 5.56. This collision was correctly detected, since the AGV 3 occupies the node **c** between [0 ; 9.448[s. Thus, AGV 1 re-planned its path to avoid this collision and instead of going through the **c** node went through the **q** node, also neighbour of **d**. In this first step of this task it was also detected that: (*i*) the **m**

Table 5.5: Maps of the three AGV (strategy 2 model) (time units in seconds)

Tasks	Steps	AGV 1	AGV 2	AGV 3
1	1			c: 0 b: 9.448 a: 19.368
	2			z #2: 24.134 b: 33.562 c: 43.263 d: 52.910 e: 62.601 g: 74.603
2	1	d: 0 q: 9.655 p: 19.325 o: 28.972 m: 38.344 l: 48.298		
	2	j: 53.047 k: 62.352		
3	1		z: 0 y: 9.641 w: 19.354 x: 23.913 v: 28.841 u: 38.493 o: 47.714 p: 57.093 s: 66.723 s #2: 76.82	
	2		t: 81.458 o: 90.890 u: 100.569 v: 110.007 w: 119.646 x: 129.205	

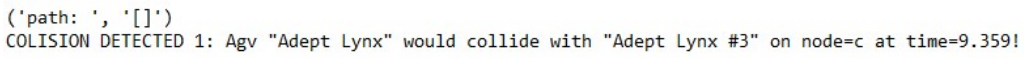
node was already in the open list, because starting from **d** and with objective to go to **l**, analyzing the path through [**e**, **g**, **h**, **j**, **k**] the **k** has as its neighbour the **m** and analyzing the path through [**q**, **p**, **o**] the **o** also has the neighbour **m**, hence the warning shown in Figure 5.57. Looking to this figure it is possible to verify that the one that has lower cost is the path [**q**, **p**, **o**, **m**]; and (*ii*) the **s** node was already in the open list, because starting from **d** and with objective to go to **l**,



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=q, cost=50547.8242724
  > New discovered node: name=q, cost=50581.8672724
('Open List before:', [{'cost': 56224.421151441355, 'parents': ['z #2', 'z', 'y'], 'name': 'y', 'g': 24030.097999999998, 'time': 24.043},
{'cost': 50547.82427240833, 'parents': ['z #2', 'b', 'c', 'd', 'q'], 'name': 'q', 'g': 43172.778000000006, 'time': 43.197},
{'cost': 50179.30539070205, 'parents': ['z #2', 'b', 'c', 'd', 'e'], 'name': 'e', 'g': 43208.21800000001, 'time': 43.233000000000004}])
('Open List after:', [{'cost': 56224.421151441355, 'parents': ['z #2', 'z', 'y'], 'name': 'y', 'g': 24030.097999999998, 'time': 24.043},
{'cost': 50547.82427240833, 'parents': ['z #2', 'b', 'c', 'd', 'q'], 'name': 'q', 'g': 43172.778000000006, 'time': 43.197},
{'cost': 50179.30539070205, 'parents': ['z #2', 'b', 'c', 'd', 'e'], 'name': 'e', 'g': 43208.21800000001, 'time': 43.233000000000004}])
```

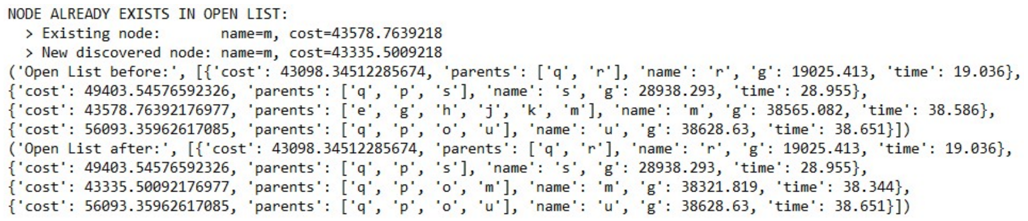
Figure 5.55: **q** node already in the open list (strategy 2 model)

analyzing the path through [**q**, **p**] the **p** has as its neighbour the **s** and analyzing the path through [**q**, **r**] the **r** also has the neighbour **s**, hence the warning shown in Figure 5.58. Looking to this figure it is possible to verify that the one that has lower cost is the path [**q**, **r**, **s**]. However, when the costs were analyzed with all neighbours of **q**, it was verified that it is more advantageous to do [**q**, **p**, **o**] than [**q**, **r**, **s**].



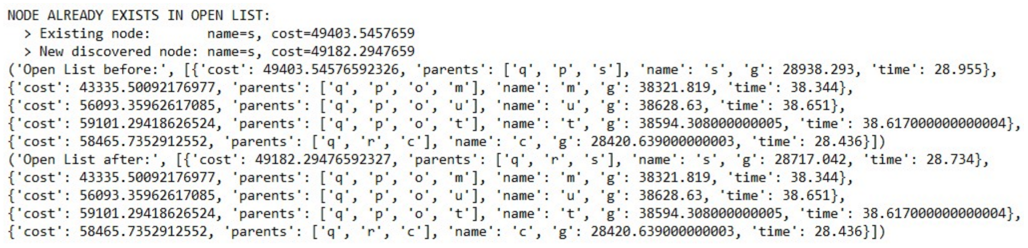
```
('path: ', '[]')
COLISION DETECTED 1: Agv "Adept Lynx" would collide with "Adept Lynx #3" on node=c at time=9.359!
```

Figure 5.56: Collision message in the first step of the second task (strategy 2 model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=m, cost=43578.7639218
  > New discovered node: name=m, cost=43335.5009218
('Open List before:', [{'cost': 43098.34512285674, 'parents': ['q', 'r'], 'name': 'r', 'g': 19025.413, 'time': 19.036},
{'cost': 49403.54576592326, 'parents': ['q', 'p', 's'], 'name': 's', 'g': 28938.293, 'time': 28.955},
{'cost': 43578.76392176977, 'parents': ['e', 'g', 'h', 'j', 'k', 'm'], 'name': 'm', 'g': 38565.082, 'time': 38.586},
{'cost': 56093.35962617085, 'parents': ['q', 'p', 'o', 'u'], 'name': 'u', 'g': 38628.63, 'time': 38.651}])
('Open List after:', [{'cost': 43098.34512285674, 'parents': ['q', 'r'], 'name': 'r', 'g': 19025.413, 'time': 19.036},
{'cost': 49403.54576592326, 'parents': ['q', 'p', 's'], 'name': 's', 'g': 28938.293, 'time': 28.955},
{'cost': 43335.50092176977, 'parents': ['q', 'p', 'o', 'm'], 'name': 'm', 'g': 38321.819, 'time': 38.344},
{'cost': 56093.35962617085, 'parents': ['q', 'p', 'o', 'u'], 'name': 'u', 'g': 38628.63, 'time': 38.651}])
```

Figure 5.57: **m** node already in the open list (strategy 2 model)

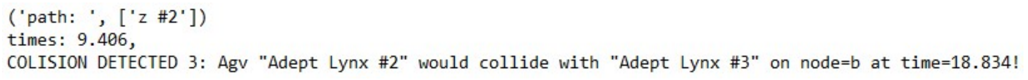


```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=s, cost=49403.5457659
  > New discovered node: name=s, cost=49182.2947659
('Open List before:', [{'cost': 49403.54576592326, 'parents': ['q', 'p', 's'], 'name': 's', 'g': 28938.293, 'time': 28.955},
{'cost': 43335.50092176977, 'parents': ['q', 'p', 'o', 'm'], 'name': 'm', 'g': 38321.819, 'time': 38.344},
{'cost': 56093.35962617085, 'parents': ['q', 'p', 'o', 'u'], 'name': 'u', 'g': 38628.63, 'time': 38.651},
{'cost': 59101.29418626524, 'parents': ['q', 'p', 'o', 't'], 'name': 't', 'g': 38594.308000000005, 'time': 38.617000000000004},
{'cost': 58465.7352912552, 'parents': ['q', 'r', 'c'], 'name': 'c', 'g': 28420.639000000003, 'time': 28.436}])
('Open List after:', [{'cost': 49182.29476592327, 'parents': ['q', 'r', 's'], 'name': 's', 'g': 28717.042, 'time': 28.734},
{'cost': 43335.50092176977, 'parents': ['q', 'p', 'o', 'm'], 'name': 'm', 'g': 38321.819, 'time': 38.344},
{'cost': 56093.35962617085, 'parents': ['q', 'p', 'o', 'u'], 'name': 'u', 'g': 38628.63, 'time': 38.651},
{'cost': 59101.29418626524, 'parents': ['q', 'p', 'o', 't'], 'name': 't', 'g': 38594.308000000005, 'time': 38.617000000000004},
{'cost': 58465.7352912552, 'parents': ['q', 'r', 'c'], 'name': 'c', 'g': 28420.639000000003, 'time': 28.436}])
```

Figure 5.58: **s** node already in the open list (strategy 2 model)

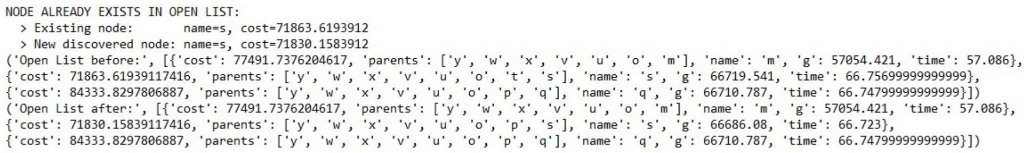
In the first step of the third task a collision of the AGV 2 with AGV 3 at **b** node was detected at 18.834 s. The collision message is displayed in Figure 5.59. This collision was correctly detected, since the AGV 3 occupies the **b** node between

[9.448 ; 19.368[s. Thus, AGV 2 re-planned its path to avoid this collision and instead of doing [**z**, **z #2**, **b**] did [**z**, **y**]. In this first step of the third task it was also detected that the **s** node was already in the open list, because starting from **z** and with objective to go to **s #2**, analyzing the path through [**y**, **w**, **x**, **v**, **u**, **o**, **t**] the **t** has as its neighbour the **s** and analyzing the path through [**y**, **w**, **x**, **v**, **u**, **o**, **p**] the **p** also has the neighbour **s**, hence the warning shown in Figure 5.60. Looking to this figure it is possible to verify that the one that has lower cost is the path [**y**, **w**, **x**, **v**, **u**, **o**, **p**].



```
('path: ', ['z #2'])
times: 9.406,
COLISION DETECTED 3: Agv "Adept Lynx #2" would collide with "Adept Lynx #3" on node=b at time=18.834!
```

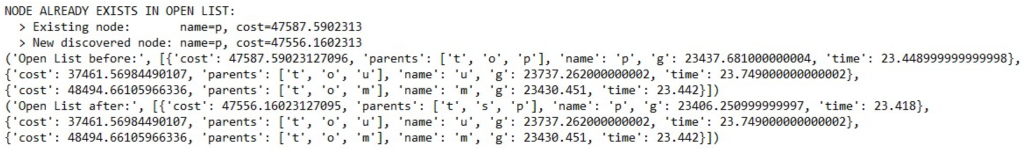
Figure 5.59: Collision message in the first step of the third task (strategy 2 model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=s, cost=71863.6193912
  > New discovered node: name=s, cost=71830.1583912
('Open List before:', [{'cost': 77491.7376204617, 'parents': ['y', 'w', 'x', 'v', 'u', 'o', 'm'], 'name': 'm', 'g': 57054.421, 'time': 57.086},
{'cost': 71863.61939117416, 'parents': ['y', 'w', 'x', 'v', 'u', 'o', 't', 's'], 'name': 's', 'g': 66719.541, 'time': 66.75699999999999},
{'cost': 84333.8297806887, 'parents': ['y', 'w', 'x', 'v', 'u', 'o', 'p', 'q'], 'name': 'q', 'g': 66710.787, 'time': 66.74799999999999}])
('Open List after:', [{'cost': 77491.7376204617, 'parents': ['y', 'w', 'x', 'v', 'u', 'o', 'm'], 'name': 'm', 'g': 57054.421, 'time': 57.086},
{'cost': 71830.15839117416, 'parents': ['y', 'w', 'x', 'v', 'u', 'o', 'p', 's'], 'name': 's', 'g': 66686.08, 'time': 66.723},
{'cost': 84333.8297806887, 'parents': ['y', 'w', 'x', 'v', 'u', 'o', 'p', 'q'], 'name': 'q', 'g': 66710.787, 'time': 66.74799999999999}])
```

Figure 5.60: **s** node again already in the open list (strategy 2 model)

In the second step of the third task it was detected that the **p** node was already in the open list, because starting from **s #2** and with objective to go to **x**, analyzing the path through [**t**, **o**] the **o** has as its neighbour the **p** and analyzing the path through [**t**, **s**] the **s** also has the neighbour **p**, hence the warning shown in Figure 5.61. Looking to this figure it is possible to verify that the one that has lower cost is the path [**t**, **s**, **p**]. However, when the costs were analyzed with all neighbours of **t**, it was verified that it is more advantageous to do [**t**, **o**, **u**] than [**t**, **s**, **p**].



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=p, cost=47587.5902313
  > New discovered node: name=p, cost=47556.1602313
('Open List before:', [{'cost': 47587.59023127096, 'parents': ['t', 'o', 'p'], 'name': 'p', 'g': 23437.681000000004, 'time': 23.448999999999998},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
('Open List after:', [{'cost': 47556.16023127095, 'parents': ['t', 's', 'p'], 'name': 'p', 'g': 23406.250999999997, 'time': 23.418},
{'cost': 37461.56984490107, 'parents': ['t', 'o', 'u'], 'name': 'u', 'g': 23737.262000000002, 'time': 23.749000000000002},
{'cost': 48494.66105966336, 'parents': ['t', 'o', 'm'], 'name': 'm', 'g': 23430.451, 'time': 23.442}])
```

Figure 5.61: **p** node already in the open list (strategy 2 model)

The total costs of each task were:

- Task 1: 64 566 mm
- Task 2: 52 322 mm

- Task 3: 119 141 mm

In view of the obtained results it is concluded that the objectives were fulfilled.

Strategy 3

In this scheduling strategy (file `TEAstar_Simulation_strategy3.vcmx`), there is also a list with three tasks. For each vehicle the closest task is assigned. The objective of the model of this strategy was to make each vehicle perform the closest task and use the TEA* planning algorithm to move to it and execute the entire path involved in performing the same. That is, each AGV has to perform the closest task, execute the complete task going through the smallest path, do not pass through points that it has already visited nor in points **f**, **i**, **n** and avoid collisions with other vehicles.

Task list

Task 1:

- first step: move the AGV from its position to the AGV Pick Location **a**

- second step: move the AGV from the AGV Pick Location **a** to the AGV Drop Location **x**

Task 2:

- first step: move the AGV from its position to the AGV Pick Location **l**

- second step: move the AGV from the AGV Pick Location **l** to the AGV Drop Location **g**

Task 3:

- first step: move the AGV from its position to the AGV Pick Location **l**

- second step: move the AGV from the AGV Pick Location **l** to the AGV Drop Location **k**

Looking at Figure 5.62 it is possible to verify that AGV 1 (yellow) is closer to task 2, AGV 2 (green) of task 1 and AGV 3 (blue) of task 3. In this figure are also found the paths performed by the AGV and it is verified that the AGV 1 and the AGV 3 share parts of its route, however as the passages by the common segments happened at different moments, the vehicles did not collide.

The maps obtained for each AGV are found in Table 5.6.

When AGV 3 was performing the first step of its task, it detected two collisions: (*i*) with AGV 1 at **h** node at 29.129 s, because AGV 3 would occupy that node in the path to **j**. The collision message is displayed in Figure 5.63. This collision was correctly detected, since AGV 1 occupies the node **h** between [24.107 ; 33.496[s. Thus, AGV 3 re-planned its path to avoid this collision, and instead of doing [**d**, **e**, **g**, **h**], verified that it was more advantageous to do [**d**,

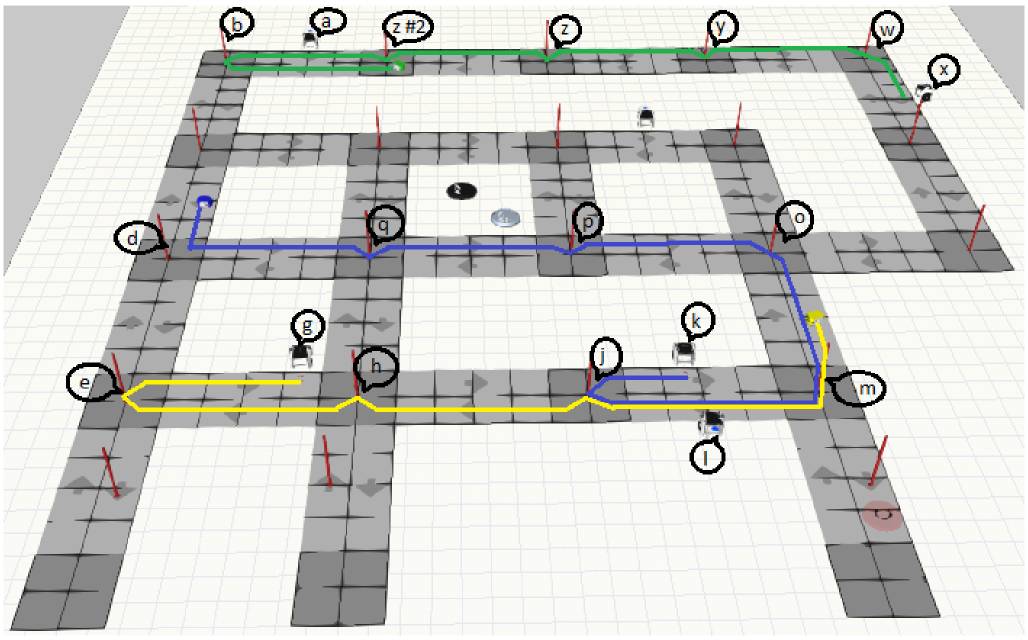
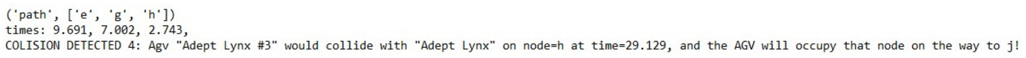


Figure 5.62: Travelled paths by the three AGV (strategy 3 model)

q]; (*ii*) with AGV 2 at **b** node at 18.807 s. The collision message is displayed in Figure 5.64. This collision was correctly detected, since AGV 2 occupies the **b** node of [9.428 ; 19.348[s. Thus, AGV 3 re-planned its way to avoid this collision, and instead of doing [**d**, **c**, **b**], checked that it was it is more advantageous to do [**d**, **q**]. In this first step of this task it was also detected that: (*i*) the **r** node was already in the open list, because starting from **d** and with objective to go to **l**, analyzing the path through **q** (that has as neighbour **r**) and analyzing the path through **c** (that also has **r** as neighbour), hence the warning shown in Figure 5.65. Looking to this figure it is possible to verify that the one that has lower cost is the path [**q**, **r**]. However, when the costs were analyzed with all neighbours of **d**, it was verified that it is more advantageous to do [**q**, **p**] than [**q**, **r**]; and (*ii*) the **s** node was already in the open list, because starting from **d** and with objective to go to **l**, analyzing the path through [**q**, **p**] the **p** has as its neighbour the **s** and analyzing the path through [**q**, **r**] the **r** also has the neighbour **s**, hence the warning shown in Figure 5.66. Looking to this figure it is possible to verify that the one that has lower cost is the path [**q**, **r**, **s**]. However, when the costs were analyzed with all neighbours of **q**, it was verified that it is more advantageous to do [**q**, **p**] than [**q**, **r**, **s**].

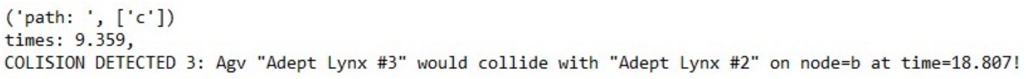
Table 5.6: Maps of the three AGV (time units in seconds)

Steps	AGV 1	AGV 2	AGV 3
1	m: 0 l: 9.954	z #2: 0 b: 9.428 a: 19.348	d: 0 q: 9.655 p: 19.325 o: 28.972 m: 38.344 l: 48.298
2	j: 14.703 h: 24.107 e: 33.496 g: 45.498	z #2: 24.114 z: 33.750 y: 43.391 w: 53.104 x: 62.663	j: 53.047 k: 62.352



```
('path', ['e', 'g', 'h'])
times: 9.691, 7.002, 2.743,
COLISION DETECTED 4: Agv "Adept Lynx #3" would collide with "Adept Lynx" on node=h at time=29.129, and the AGV will occupy that node on the way to j!
```

Figure 5.63: Collision message in the first step of the AGV 3 task (strategy 3 model)

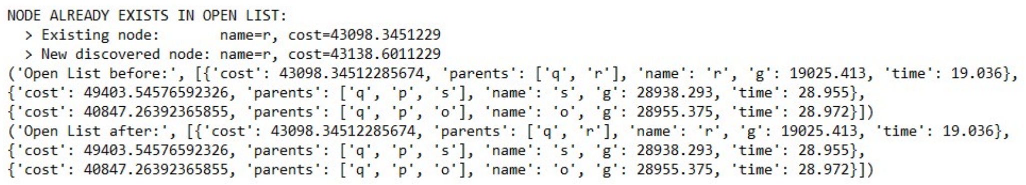


```
('path: ', ['c'])
times: 9.359,
COLISION DETECTED 3: Agv "Adept Lynx #3" would collide with "Adept Lynx #2" on node=b at time=18.807!
```

Figure 5.64: Second collision message in the first step of the AGV 3 task (strategy 3 model)

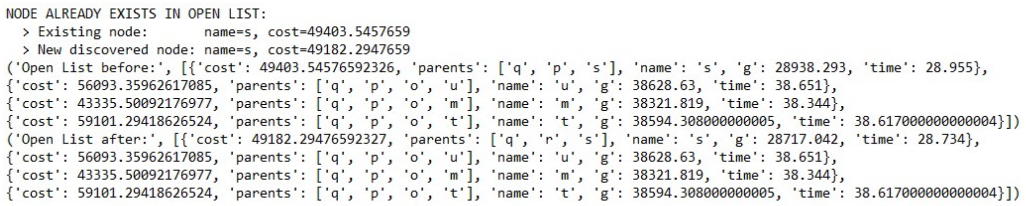
The total costs of the task of each AGV were:

- Task 1: 35 478 mm
- Task 2: 52 636 mm
- Task 3: 52 322 mm



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=r, cost=43098.3451229
  > New discovered node: name=r, cost=43138.6011229
('Open List before:', [{'cost': 43098.34512285674, 'parents': ['q', 'r'], 'name': 'r', 'g': 19025.413, 'time': 19.036},
{'cost': 49403.54576592326, 'parents': ['q', 'p', 's'], 'name': 's', 'g': 28938.293, 'time': 28.955},
{'cost': 40847.26392365855, 'parents': ['q', 'p', 'o'], 'name': 'o', 'g': 28955.375, 'time': 28.972}])
('Open List after:', [{'cost': 43098.34512285674, 'parents': ['q', 'r'], 'name': 'r', 'g': 19025.413, 'time': 19.036},
{'cost': 49403.54576592326, 'parents': ['q', 'p', 's'], 'name': 's', 'g': 28938.293, 'time': 28.955},
{'cost': 40847.26392365855, 'parents': ['q', 'p', 'o'], 'name': 'o', 'g': 28955.375, 'time': 28.972}])
```

Figure 5.65: **r** node already in the open list in the first step of the AGV 3 task (strategy 3 model)



```
NODE ALREADY EXISTS IN OPEN LIST:
  > Existing node:       name=s, cost=49403.5457659
  > New discovered node: name=s, cost=49182.2947659
('Open List before:', [{'cost': 49403.54576592326, 'parents': ['q', 'p', 's'], 'name': 's', 'g': 28938.293, 'time': 28.955},
{'cost': 56093.35962617085, 'parents': ['q', 'p', 'o', 'u'], 'name': 'u', 'g': 38628.63, 'time': 38.651},
{'cost': 43335.50092176977, 'parents': ['q', 'p', 'o', 'm'], 'name': 'm', 'g': 38321.819, 'time': 38.344},
{'cost': 59101.29418626524, 'parents': ['q', 'p', 'o', 't'], 'name': 't', 'g': 38594.308000000005, 'time': 38.617000000000004}])
('Open List after:', [{'cost': 49182.29476592327, 'parents': ['q', 'r', 's'], 'name': 's', 'g': 28717.042, 'time': 28.734},
{'cost': 56093.35962617085, 'parents': ['q', 'p', 'o', 'u'], 'name': 'u', 'g': 38628.63, 'time': 38.651},
{'cost': 43335.50092176977, 'parents': ['q', 'p', 'o', 'm'], 'name': 'm', 'g': 38321.819, 'time': 38.344},
{'cost': 59101.29418626524, 'parents': ['q', 'p', 'o', 't'], 'name': 't', 'g': 38594.308000000005, 'time': 38.617000000000004}])
```

Figure 5.66: **s** node already in the open list in the first step of the AGV 3 task (strategy 3 model)

In view of the obtained results, it is concluded that the objectives were fulfilled.

In all analyzed strategies, when it is not possible to obtain paths without collisions, the error message of the Figure 5.19 is also displayed, as seen in the model of the `TEAstar_Simulation_v3.vcmx` file.

The subject mentioned in the model of the `TEAstar_Simulation_strategy1.vcmx` file, of the TEA* algorithm not being prepared to place the node where each AGV ends its trajectory as occupied, is common to the remaining strategies.

All files referred throughout this chapter are included in CD that is attached to this document.

5.3 Conclusion

The tests accomplished allowed to confirm that the AGV performed the desired tasks according to the intended algorithm, travelled the smallest path, avoided collisions, did not pass in nodes that it had already visited, and furthermore, whenever detected the same node with different costs, placed the ones which had the lowest cost on the open list.

Chapter 6

Conclusions

In this chapter are drawn the main conclusions of the work developed and are presented some ideas, possible to be implemented, in order to improve the presented work.

6.1 Final Outcomes of the Work

According to the initial objectives mentioned in Chapter 1, and analyzing the tests performed and the results obtained, it can be concluded that there is a clear positive outcome of all the work developed, since two trajectory planning algorithms were implemented in the VC software, and one scheduling algorithm with three distinct types of approaches.

Regarding the determination of the advantages and disadvantages between the different algorithms, it was only possible to compare the planning algorithms (A* and TEA*) and not the of the scheduling ones. It was verified that the TEA* algorithm allowed to achieve the goal of avoiding deadlocks in the trajectory planning of AGV fleets. In relation to the three approaches of the scheduling algorithm, the characteristics of each one were described.

Comparing the results attained with the initial objectives, there was a difference that is mainly due to the fact that the initial learning of the Visual Components software operation and programming took longer than expected, mainly due to the absence of technical information available.

6.2 Improvement Suggestions

Firstly, it is suggested to implement more planning and scheduling algorithms in order to determine the advantages and disadvantages between algorithms. Secondly, it is suggested to incorporate other functionalities, among which: (*i*) in the planning algorithms the condition that if any AGV has to go to charge the battery to the AGV Charging Station, it has to plan the route until this one and, after load, to plan the route to the destination that it intends to reach; (*ii*) add wagons to the AGV and be able to detect the position of the AGV and its last wagon, in order to avoid collisions with other vehicles; (*iii*) to use the nodes (mentioned in the chapters 4 and 5), that belong to dead ends, to direct some AGV to there in situations in which it is intended to leave the way free for another AGV, thus avoiding a collision; (*iv*) in the scheduling algorithms, to ensure that the AGV also move to the nearest AGV Drop Locations; and (*v*) if any AGV is not able to go to the nearest task due to collisions, make it replan and choose the nearest next task.

Bibliography

[1] Simio LLC, "Business Benefits of Simio's Production Scheduling | Simio." `https://www.simio.com/resources/white-papers/Benefits-of-Production-Scheduling/index.php`, Last access: June 14, 2018. [cited on p. iii, 7, 8]

[2] IMAM Consultoria, "Sem título-2." `https://www.imam.com.br/consultoria/artigo/pdf/fundamentos-veiculos-automaticamente-guiados.pdf`, Last access: February 7, 2018. [cited on p. iii, 9, 10]

[3] AGVs, "AGVs :: Automated Guided Vehicle Solutions." `http://www.agvs.com.br/eng-agv-rebocador.html`, Last access: February 7, 2018. [cited on p. iii, 9]

[4] AGVs, "AGVs :: Automated Guided Vehicle Solutions." `http://www.agvs.com.br/eng-agv-carregador.html`, Last access: February 7, 2018. [cited on p. iii, 9, 10]

[5] AGVs, "AGVs :: Automated Guided Vehicle Solutions." `http://www.agvs.com.br/eng-agv-especial.html`, Last access: February 8, 2018. [cited on p. iii, 9, 10]

[6] System Logistics Spa, "www.system-agv.com/por/veículos-quad-com-eixo-de-rotação-livre." `http://www.system-agv.com/por/veículos-quad-com-eixo-de-rotaç~{a}o-livre`, Last access: February 8, 2018. [cited on p. iii, 10]

[7] Eduardo António da Silva Santos, "Logística baseada em AGVs," June 2013. [cited on p. iii, 8, 10, 11, 12, 13, 14]

[8] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots.* MIT Press, 2004. [cited on p. iii, 14, 15]

[9] J. Santos, P. Costa, L. Rocha, K. Vivaldini, A. P. Moreira, and G. Veiga, "Validation of a time based routing algorithm using a realistic automatic warehouse scenario," in *Robot 2015: Second Iberian Robotics Conference* (L. P. Reis, A. P. Moreira, P. U. Lima, L. Montano, and V. Muñoz-Martinez, eds.), (Cham), pp. 81–92, Springer International Publishing, 2016. [cited on p. iii, 18, 19, 20]

[10] S. M. LaValle, *PLANNING ALGORITHMS*. Cambridge University Press, 2006. [cited on p. iii, 20, 21, 22]

[11] 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*, vol. 7, pp. 151–155, Jan 2010. [cited on p. iii, 22, 23]

[12] A. Tuncer and M. Yildirim, "Dynamic path planning of mobile robots with improved genetic algorithm," *Computers Electrical Engineering*, vol. 38, no. 6, pp. 1564 – 1572, 2012. [cited on p. iii, iv, 23, 24, 25, 26, 27]

[13] VISUAL COMPONENTS, "Quick Start for AGV Library - Quick-Start-for-AGV-Library-2.pdf." `http://academy.visualcomponents.com/wp-content/plugins/pdf-viewer/stable/web/viewer.html?file=http://academy.visualcomponents.com/wp-content/uploads/2017/01/Quick-Start-for-AGV-Library-2.pdf`, Last access: November 3, 2017. [cited on p. 2]

[14] Visual Components Community, "Visual Components - The Simulation Community." `http://forum.visualcomponents.com/`, Last access: December 9, 2017. [cited on p. 2]

[15] VISUAL COMPONENTS, "Visual Components 4.0 - The next generation of 3D manufacturing simulation." `https://www.visualcomponents.com/products/visual-components-4-0/`, Last access: January 22, 2018. [cited on p. 5]

[16] VISUAL COMPONENTS, "Accelerate the sales of your production solutions with Visual Components." `https://www.visualcomponents.com/solutions/sales-acceleration/`, Last access: June 13, 2018. [cited on p. 6]

[17] VISUAL COMPONENTS, "Design the factories of the future with Visual Components." `https://www.visualcomponents.com/solutions/manufacturing-design-planning/`, Last access: June 13, 2018. [cited on p. 6]

[18] Cyberbotics Ltd., "Webots: webots." `https://www.cyberbotics.com/webots.php`, Last access: January 25, 2018. [cited on p. 6]

[19] Cyberbotics Ltd., “Webots: features.” https://www.cyberbotics.com/features, Last access: January 25, 2018. [cited on p. 6]

[20] COPPELIA ROBOTICS, “Coppelia Robotics V-REP: Create. Compose. Simulate. Any Robot..” http://www.coppeliarobotics.com/index.html, Last access: January 25, 2018. [cited on p. 6]

[21] COPPELIA ROBOTICS, “Coppelia Robotics V-REP: Create. Compose. Simulate. Any Robot..” http://www.coppeliarobotics.com/features.html, Last access: January 25, 2018. [cited on p. 6]

[22] COPPELIA ROBOTICS, “V-REP Forum - Index page.” http://www.forum.coppeliarobotics.com/, Last access: January 25, 2018. [cited on p. 6]

[23] FlexSim Software Products, Inc., “Software de simulação para a fabricação, manuseio de materiais, de saúde, etc. - Flexsim Simulação Software.” https://www.flexsim.com/pt/, Last access: January 26, 2018. [cited on p. 7]

[24] FlexSim Software Products, Inc., “Flexsim - Flexsim Simulation Software.” https://www.flexsim.com/pt/flexsim/, Last access: January 26, 2018. [cited on p. 7]

[25] Talumis, “FlexSim AGV Simulation Software - Talumis.” https://www.talumis.com/agv-simulation-software/, Last access: January 26, 2018. [cited on p. 7]

[26] FlexSim Software Products, Inc., “Simulation modeling meets virtual reality, Oculus Rift | FlexSim.” https://www.flexsim.com/pt/simulation-modeling-meets-virtual-reality-oculus-rift/, Last access: June 13, 2018. [cited on p. 7]

[27] FlexSim Software Products, Inc., “Case Studies Other Resources | FlexSim Healthcare.” https://healthcare.flexsim.com/case-studies/, Last access: June 14, 2018. [cited on p. 7]

[28] Simio LLC, “Simio Simulation and Scheduling Software: Product Family | Simio.” https://www.simio.com/products/, Last access: May 24, 2018. [cited on p. 7]

[29] Simio LLC, “Simio Simulation Software Resources | Simio.” https://www.simio.com/resources/, Last access: May 24, 2018. [cited on p. 7]

[30] Konecranes, “Automated Guided Vehicles | Konecranes.com.” http://www.konecranes.com/equipment/container-handling-equipment/automated-guided-vehicles, Last access: April 9, 2018. [cited on p. 8]

[31] I. F. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 170, no. 3, pp. 677 – 709, 2006. [cited on p. 8]

[32] Translift, "Translift." `http://www.transliftbr.com/site/produtos-servicos/transportadores/sistema-agv`, Last access: February 7, 2018. [cited on p. 9]

[33] Conveyco, "The Advantages and Disadvantages of Automated Guided Vehicles (AGVs) | Conveyco." `http://www.conveyco.com/advantages-disadvantages-automated-guided-vehicles-agvs/`, Last access: November 26, 2017. [cited on p. 10, 11]

[34] Klas Hedenberg, "Obstacle Detection for Driverless Trucks in Industrial Environments," 2014. [cited on p. 11]

[35] André dos Santos Granjo Oliveira, "Localização e deteção de peças utilizando lasers de segurança," February 2013. [cited on p. 12, 13]

[36] Filipe Manuel Costa Ferreira, "Desenvolvimento de módulos para planeamento e controlo de execução de missões de veículos aéreos não tripulados," October 2012. [cited on p. 17, 18, 20]

[37] Leliane Nunes de Barros, "Microsoft PowerPoint - Aula5-Astar-2006." `https://www.ime.usp.br/~leliane/IAcurso2006/slides/Aula5-Astar-2006.pdf`, Last access: January 23, 2018. [cited on p. 17, 18]

[38] Sónia Raquel de Sousa Neves Cardoso, "Optimização de Rotas e da Frota Associada," November 2009. [cited on p. 23]

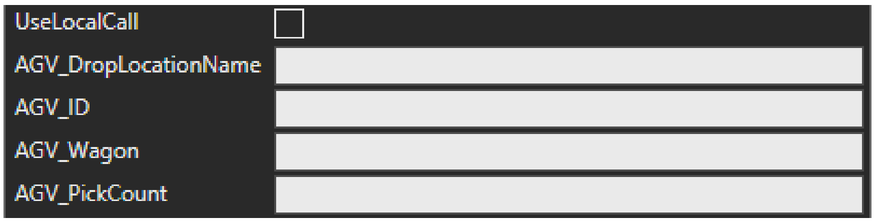
[39] Rui Paulo Pinto da Rocha, "Desenvolvimento de um Sistema de Gestão de AGVs," September 1998. [cited on p. 27, 28, 29, 30]

[40] Ana Viana, "O que é a Investigação Operacional?." `https://moodle.isep.ipp.pt/pluginfile.php/120719/mod_folder/content/0/T_METAD_A.pdf?forcedownload=1`, Last access: May 23, 2018. [cited on p. 28, 29]

Appendix A

Settings in the Components Properties

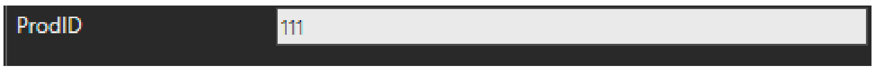
AGV Pick Location



UseLocalCall	☐
AGV_DropLocationName	
AGV_ID	
AGV_Wagon	
AGV_PickCount	

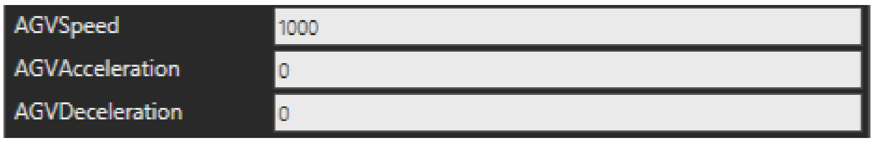
Figure A.1: AGV Pick Location properties

Shape Feeder



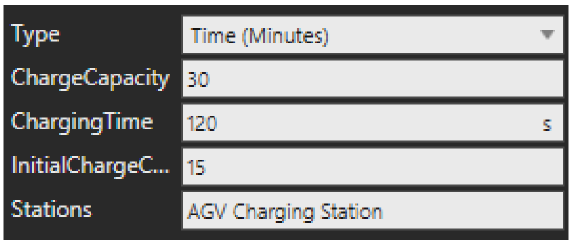
ProdID	111

Figure A.2: Shape Feeder properties



AGVSpeed	1000
AGVAcceleration	0
AGVDeceleration	0

(a)



Type	Time (Minutes)
ChargeCapacity	30
ChargingTime	120 s
InitialChargeC...	15
Stations	AGV Charging Station

(b)

Figure A.3: AGV Default properties (a) and AGV ReCharge properties (b)