FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Analysis and simulation of AGVS routing strategies using V-REP

Luís Filipe de Lemos Miranda

Mestrado Integrado em Engenharia Mecânica

Supervisor: Manuel Romano dos Santos Pinto Barbosa

Co-Supervisor: António Manuel Ferreira Mendes Lopes

July 19, 2017

Analysis and simulation of AGVS routing strategies using V-REP

Luís Filipe de Lemos Miranda

Mestrado Integrado em Engenharia Mecânica

July 19, 2017

Abstract

An Automated Guided Vehicle System (AGVS) is a modern solution for material transportation in a industrial context, where vehicles, with varying degrees of autonomy, can independently move around the layout. The routing problem, a part of fleet management, concerns specifically the issue of how a vehicle can get from a point of origin to a destination and when multiple vehicles are present, it becomes quite complex to solve. Due to the highly dynamic nature of several vehicles moving simultaneously, the wrong routing strategy can lead to much higher delivery times and distances and even deadlocks. Due to its complexity, usually there is not a single solution to the problem, and thus, simulations become very important. For this purpose, V-REP, a robotic systems simulator, is used.

The first step towards a complete simulation is the creation of a layout. For this dissertation, free range vehicles, which move freely inside their confined spaces, are used. Using the provided 3D modeller, the size, objects, obstacles and nodes are defined, as well as the vehicle, a simple round shape. To control all aspects of the simulation, three scripts were created: *manager script* (assigns tasks to the vehicles), *vehicle script* (controls the vehicle movements, one for each) and the *totals script* (gathers all data to produce the outputs).

Since the vehicles are free range, a special focus was given to strategies where the paths are found using path finding algorithms, using the already included in V-REP, Open Motion Planning Library (OMPL), which allows quick and easy testing of many algorithms. To better understand V-REP and the OMPL, a first case with a single vehicle is created, after which a transition to a multi-vehicle situation is made.

Finally, the multi-vehicle routing strategies are implemented and tested. Most of the strategies use mainly the OMPL, while one of them uses a exhaustive node search approach. Each strategy must complete a given task list and the number of vehicles increases from one to six vehicles. From the data obtained, which concerns both the in-simulation parameters (time and distance) and also how V-REP performs while executing them, each strategy is analysed along with an overall comparison.

In general, it was possible to implement different routing strategies in a multi-vehicle type problem, taking advantage of V-REP functionalities for modelling the various elements, including layout facilities, vehicles, path planning and routing strategies. The main limitation is related to the simulation speed performance, which can be expected to degrade with the size and complexity of the system. Taking into account the extensive capabilities of V-REP, it might be more appropriate to use it to simulate more localized navigation problems regarding selected paths (incorporating the vehicle's dynamic and kinematic models), rather than complex routing problems with a high number of vehicles.

Resumo

Um AGVS (Sistema de Veículos Guiados Autonomamente) é uma solução moderna para o transporte de materiais num contexto industrial, onde os veículos, com diferentes níveis de autonomia se movem independentemente no espaço. O problema de *routing*, uma parte da gestão de frota, relaciona-se diretamente com a questão de como deve um veículo ir desde um ponto de origem até um de destino e quando múltiplos veículos estão presentes, torna-se um problema bastante complexo. Devido à natureza altamente dinâmica de ter vários veículos em movimento simultaneamente, a estratégia de *routing* errada pode levar a tempos e distâncias de entrega elevados e até bloqueios do sistema. Devido à sua complexidade, não existe normalmente uma solução única para este problema e, portanto, as simulações tornam-se bastante importantes. Para este propósito, o V-REP, um simulador de sistemas robóticos, foi usado.

O primeiro passo para uma simulação completa é a criação do espaço de trabalho. Para esta dissertação, foi escolhido um espaço onde os veículos se movem livremente dentro das suas áreas delimitadas. Usando o modelador 3D fornecido, o tamanho, objetos, obstáculos e nodos são definidos, tal como o veículo, uma simples forma redonda. Para controlar todos os aspetos da simulação, são criados três scripts: *manager script* (onde as tarefas são atribuídas), *vehicle script* (controla os movimentos de veículo, um para cada) e o *totals scripts* (que reúne toda a informação para produzir os *outputs*).

Sendo o espaço de movimento livre, foi dado um foco especial a estratégias onde os caminhos são encontrados usando algoritmos de pesquisa de caminhos, usando a Open Motion Planning Library (OMPL), já incluída no V-REP, que permite testar e alterar facilmente vários algoritmos. Para entender melhor o V-REP e a OMPL, é criado um primeiro caso com apenas um veículo, após o qual é feita uma transição para uma situação de múltiplos veículos.

Finalmente, as estratégias de *routing* de múltiplos veículos são implementadas e testadas. A maioria das estratégias usa principalmente a OMPL, enquanto uma delas usa uma técnica de pesquisa exaustiva de nodos. Cada estratégia tem de completar uma lista de tarefas, com o número de veículos a variar de um a seis. Os dados obtidos referem não só os parâmetros de simulação (tempo e distância), mas também o próprio desempenho do V-REP ao executar as estratégias. É feita uma análise para cada estratégia, tal como uma comparação entre todas.

De modo geral, foi possível implementar diferentes estratégias de *routing* com múltiplos veículos, aproveitando as funcionalidades do V-REP para modelar diferentes elementos, tais como o espaço de trabalho, veículos, planeamento de rotas e estratégias de *routing*. A principal limitação encontrada está relacionada com a velocidade de execução das estratégias, que, tal como esperado, piora com o aumento do tamanho e complexidade do sistema. Tendo em conta as muitas capacidades do V-REP, talvez seja mais apropriado para a simulação de problemas de navegação local (incorporando os modelos dinâmicos e cinemáticos dos veículos), do que para problemas complexos de *routing* com elevado número de veículos.

Agradecimentos

Aos meus orientadores, Professor Manuel Romano Barbosa e Professor António Mendes Lopes, pela ajuda e orientação dada ao longo deste trabalho.

A todos os colegas e professores desta faculdade que me acompanharam ao longo de todo o curso.

Aos meus amigos, que sempre me incentivaram e me ajudaram a completar esta dissertação.

Aos meus pais, por sempre me apoiarem e me permitirem chegar a este ponto na minha vida e a quem dedico esta dissertação.

À minha namorada, por sempre me aturar e encorajar, sem nunca me deixar desistir, numa altura tão difícil, como foi este trabalho.

Luís Filipe Miranda

“Whenever you feel like criticizing any one," he told me,
"just remember that all the people in this world haven’t had the advantages that you’ve had.”

F. Scott Fitzgerald, *The Great Gatsby*

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AGV	Automated Guided Vehicle
AGVS	Automated Guided Vehicle System
RRT	Rapidly-exploring Random Tree
RRT*	Rapidly-exploring Random Tree star
PRM	Probabilistic RoadMap
PRM*	Probabilistic RoadMap star
Lazy PRM*	Lazy Probabilistic RoadMap star
OMPL	Open Motion Planning Library
ROS	Robot Operating System
CAD	Computer-Aided Design
V-REP	Virtual Robot Experimentation Platform

Chapter 1

Introduction

1.1 The Routing Problem

The routing problem is found in the context of vehicle and traffic management and it specifically concerns the issue of how to go from a point of origin to a destination. This problem can be found in a multiple vehicle situation, where the complexity increases as it has to guarantee a successful path, one that connects the origin and destination, for each of the vehicles in a efficient and timely manner. While it may be applied to a multitude of situations, such as urban traffic or delivery trucks, in this case it concerns the routing of multiple automated guided vehicles in an industrial setting. An automated guided vehicle system (AGVS) is comprised not only by the vehicles but also by the fleet management, control architecture and all other elements necessary for a fully working system [1].

The vehicles, which are a type of mobile robots, can vary according to the navigation and localization methods employed and also the type of task they are designed for. These vehicles are integrated in a larger network that allows communication between the vehicles as well as to a central computer system that receives and provides the necessary information for the vehicles to complete the orders. The management system also plays a major part in the larger AGVS and is responsible for, among other things, the dispatching, allocation and scheduling of vehicles, as well as the routing and path selection. When there is a high number of vehicles involved, the management problems become more dynamic and complex, and thus, a proper strategy can have a considerable impact on the system's performance.

A solution to the routing problem is dependent on many factors, e.g. the number of vehicles, the autonomy and movement capabilities of the vehicles, the layout, including the load and unloading areas, occupied space and free workspace available for the vehicles to move, among others. All of these variables can combine in innumerable ways, making the solution to the routing problem not a single one, but instead one that depends on the specific case. The difficulties of a multi-vehicle scenario arise from the highly dynamic behaviour caused by the ever changing position of the vehicles and other obstacles. Due to this, the solutions usually rely on a previously

chosen set of rules that provide a route for each vehicle from start to finish and possible solutions to any problems that can arise during movement, such as blocked areas or traffic congestions [2].

Due to the complexity and high cost of implementing or changing an AGVS in a factory or other industrial setting, system simulations are an important and valuable step, both before its creation and during its use. This can help in determining the most efficient layout scheme, the management and routing strategies that should be used, the number of vehicles required and other parameters [3]. The software used in this work is V-REP due to its public availability for students and its capabilities for creating graphic environments, calculation modules and programming abilities, that allow for the implementation, testing and visualisation of different routing strategies.

1.2 Goals

The main objective for this work is to create a model of an automated guided vehicle system, capable of working under different routing strategies and to determine how adequate is V-REP for this goal. For this purpose, the V-REP software must be explored and analysed as to determine how to create an environment that allows for the analysis of the various routing strategies. The goals for these models are:

- Creation of various layouts, including the occupied and the free space for the vehicle to move, as well as the location of the stations;
- Selection of the number of vehicles to be used in the simulation;
- Specification of the transport tasks to be completed;
- Implementation of different routing strategies;
- Extraction of data from the simulations, to further analyse the advantages and disadvantages of the different strategies.

With the creation and combination of these features in V-REP scenes, a multitude of possibilities can be created to compare routing strategies under different conditions.

1.3 Dissertation Structure

This dissertation is structured in five chapters. Subsequent to this introduction comes Chapter 2: *AGV Systems*, where it is presented what an automated guided vehicle system is and how it works, from the vehicles themselves to fleet management and the control structure. What the routing problem is and its possible solutions, are addressed in more detail. In conclusion, it is given an overview of some commercially available simulators, including V-REP, and a comparison between them.

The third chapter, Chapter 3: *Single Vehicle Routing Problem in V-REP*, concerns, in the first section, the functionalities and possibilities of V-REP, with a special focus on the elements that can

assist the creation of the system and are used throughout this work, such as *paths* and embedded scripts. A model of a single vehicle system is created, along with a delineation of the composing elements, particularly the scripts. This system is then tested to validate the elements created and to define a structure for modelling a multiple vehicle routing problem.

Chapter 4: *Modelling a Multi-vehicle Problem in V-REP*, explores the possibilities of introducing more vehicles to the system and how this impacts the routing problem. The different layouts used for the tests are described, as well as the new routing strategies implemented. Additional scripts and elements added to the scene are also specified, along with the data produced from the test cases. The information obtained is then analysed and the results from different strategies are compared for a conclusive evaluation.

Finally, in Chapter 5: *Conclusions*, the results and conclusions from the work are summarized and presented in a global context, focusing on the capabilities of using V-REP for the specific purpose of simulating an automated guided vehicle system and applying different routing strategies to it. Possible future work and expansions to the system are also discussed.

Chapter 2

AGV Systems

This chapter addresses what an automated guided vehicle system is and how it works. The vehicles, possible layouts, path planning algorithms and the system in which they are integrated in are presented. Then, the routing problem is presented along with some common solutions. Lastly, a brief comparison between simulators is made.

2.1 Automated Guided Vehicle System

Robotics is currently one of the most appealing areas in the industrial world and a big part of it are mobile robots. Of these, automatic guided vehicles take a special interest by being a flexible and modern solution for material flow, offering new possibilites in many industrial applications, which has been used for more than fifty years. While usually an AGV is seen as a lesser mobile robot, in terms of abilities and independence, nowadays AGVs are becoming smarter and more autonomous, as they benefit from the technical developments of modern robots. Due to the advancements in the vehicles technology, more manufacturers worldwide are opting to implement AGVS in their factories. By 2008, over 27,500 vehicles in 3,300 AGV systems were installed in Europe and the global number is expected to continue rising, in part due to a growth in interest from China [4]. While this number may be small when compared to other robotic systems components, manipulators or industrial robots for example, it is partly explained by the relatively high cost and difficulty to implement.

The most common use of an AGV is in a factory environment and the most usual task is material handling and transportation from one place to another. Moving products is a task that adds no value to them and where a lot of resources are spent. Because of this, it is very important for the transportation to be done in the most efficient and economical way possible, and thus, it is necessary to use the best strategy in navigation and routing.

An AGVS is composed not only by its tangible parts, such as the vehicles and the layout, but also by its intangible elements, for instance, the control architecture and fleet management. The vehicles and the layout are what defines which tasks can be performed by the system, while control

architecture and fleet management are in charge of assigning the orders to the vehicle and making sure the paths are successfully completed.

2.1.1 Vehicles

The vehicles used in AGV systems can be of many types and the choice depends on the kind of transportation task to be completed. Some commonly available vehicles are:

- Forklifts;
- Tow vehicles;
- Unit load vehicles (figure 2.1);
- Special application vehicles (figure 2.2).

Figure 2.1: Unit load AGV for transporting containers from Konecranes Gottwald [5]

The methods used to allow an AGV to navigate autonomously have changed throughout the years, as the sensor technology used in the vehicles and environments has improved. In earlier versions, a more rigid approach was used, such as using a conducting wire embedded in the floor or reflective paint, which the vehicles would follow [6]. These methods had the problem of requiring alterations to the environment and were more intrusive and difficult to modify.

A modern alternative is to use wireless navigation. Initially this was done by using markers, such as a laser scan on top of the vehicle and reflective markers. By triangulating the positions of the markers and comparing it with the positions stored in the memory it can localise where the vehicle is [7]. The current version of this wireless technology is inertial navigation, which uses a gyroscope and an accelerometer for each axis of a three coordinate system. By starting from

a known position and measuring the internal movements, it can get an accurate reading of the current position. It can also be aided by floor markers, such as QR codes [8] or magnets, to assure a correction of the positioning errors within a much more precise range.

The kinematic design found in these vehicles are usually of three basic types [9]:

- **Differential control**: two drive wheels vary their speed independently, in order to turn and move the vehicle, supported by swivel caster wheels;
- **Steered wheel control**: turns one or more guiding wheels along with other traction or support wheels, similar to normal cars;
- **Omnidirectional vehicles**: uses wheels that allow movement in all directions and function in a way similar to differential control. This allows the vehicle to rotate around its center and move sideways. An example can be seen in figure in figure 2.2.

Figure 2.2: Omnidirectional AGV used to move shelves in a warehouse, lifting them from below, from Amazon Robotics [8]

2.1.2 Layouts

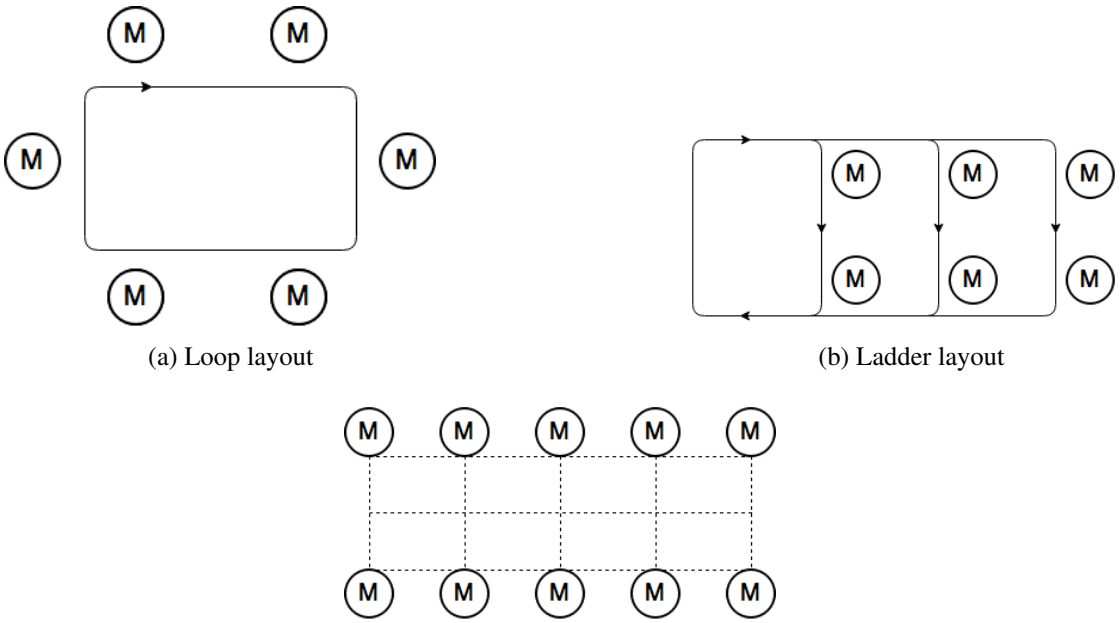
The layout of a factory is how the workspace is defined, not only by its size, but also by the location of pickup/drop-off points, static obstacles, area limitations and, in case they exist, the pre-defined paths.

The first step in a layout is defining the location of the load/unload stations. Some common arrangements are [10]:

- Straight line;
- Loop (figure 2.3a);
- Ladder type (figure 2.3b);

- Custom design (figure 2.3c).

A possible custom design is similar to the one found in ports, where the stations are located in two rows opposite from each other for loading and unloading. Another layout is a grid, where the stations are located in the middle of the branches.



(a) Loop layout

(b) Ladder layout

(c) Port layout

Figure 2.3: A few common layouts, where M represents a machine or load/unload station

After placing the stations, a decision regarding how the vehicles move must be made. Two possible approaches are using physically pre-defined paths (using tape-guided or wire-guided navigation) or free range (requires the use of smarter vehicles). If the movement is done in free range, this means that the vehicles can take any possible path connecting the two stations, using different routing rules.

When the paths are pre-defined, then the routes must be carefully chosen. The layout is usually the biggest influence on what paths can be created. For more complex layouts, the paths between stations must be designed taking into account travel distance, intersections created and possible traffic occurrences, for example. The lanes can also have one or two directions.

A viable approach is what is called a tandem configuration, where a layout consists of two or more loops. Each of these loops contain specific stations where the material can be transferred from one loop to the other and usually a single vehicle per loop. This brings the simplicity and advantages of loop layouts, such as reducing or eliminating traffic control (depending if there is more than one vehicle), to more complex layouts [11].

A feature present in some layouts are buffer areas [12]. These areas exist to help traffic and avoid deadlocks. The vehicles can move into these spaces while stopped at a station or next to an intersection so the section does not get blocked.

2.1.3 AGVS Control Architecture

An automatic guided vehicle must be able to work autonomously but also along with other vehicles, persons and computer controlled equipment in a cooperative environment. For that, it must be fully integrated into a factory system and so it has to communicate with a manager or dispatcher system.

A common way of doing this is by having a central computer manage all the vehicles and where some decisions are made and sent to each individual one. This computer is usually connected to a larger company wide system, such as an WMS (Warehouse Management System) or an ERP (Enterprise Resource Planning System), and so it will receive information on a higher level, such as which products are necessary or where they can be found, but also lower level equipments such as PLCs (Programmable Logic Controller) (figure 2.4) . This central controller will also gather all information regarding the AGV's status such as location and availability. In some cases the central computer that receives the higher level information may be connected to other computers that each then control a part of the machines or vehicles in an hierarchical form.

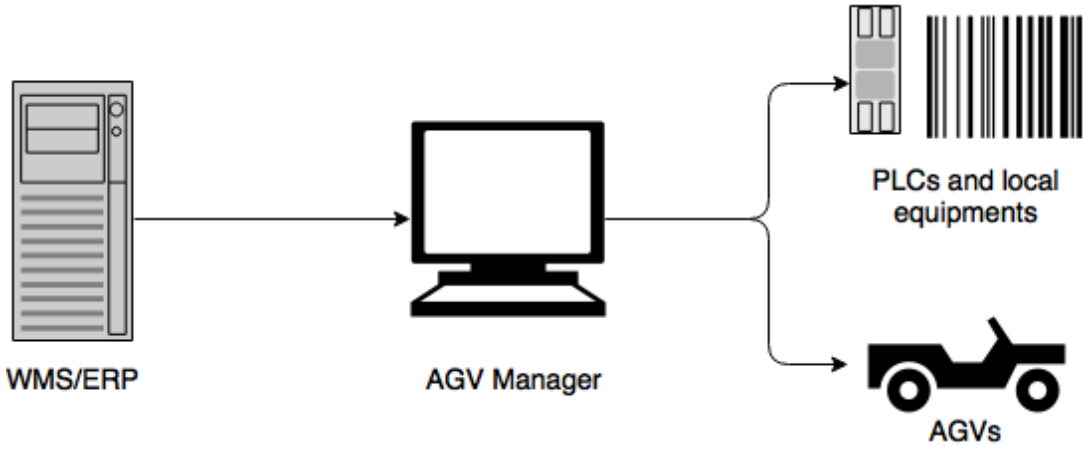


Figure 2.4: Example of a common AGV control architecture

With all this information, the system can then make all the necessary decisions, regarding vehicle allocation, path planning and others. Often, the system is also able to control other parts of the environment such as doors or traffic lights. The communication between the computer and the vehicle is usually done wirelessly using Radio Frequencies or an 802.11n Wireless network (Wi-Fi) [9].

The information can be visualised in a system monitor, which can gather information about the system operation and make it easily available for the user in real time or record it for external use. The computer can display the current location of the vehicles, their path, destinations or status conditions. It can also generate performance reports regarding each AGV, station, path or whatever information is desired. Any error that occurs is also logged in the system so that it can be more easily identified and solved. With the recordings of previous operations, a more careful analysis can be done at a later date.

Companies like Savant [13] or Egemin [14] offer software capable of performing these tasks (figure 2.5), which can be customised, as no two factories have the same needs and specifications. Some softwares also provide functionalities to help create the working layout by using a CAD file of the factory and placing stations and pathways [15].

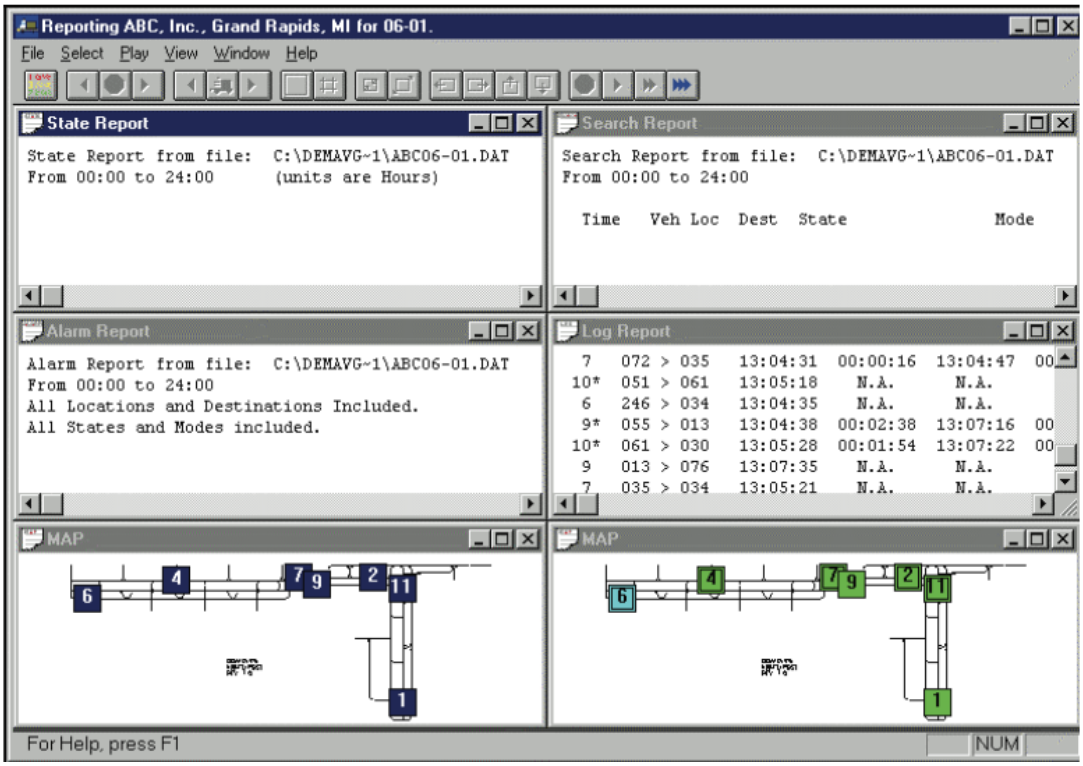


Figure 2.5: Example of a system monitor software, Q-View by Savant [13]

2.1.4 AGV Management

When managing a fleet of AGVs there are considerations that need to be done regarding when and how they complete the given tasks. The three main functions are [16]:

- **Dispatching:** dispatching is the allocation of a specific vehicle to perform a particular task;
- **Routing:** routing is the selection of the path each vehicle should take from a starting position to a destination;
- **Scheduling:** scheduling is the determination of arrival and departure times for all vehicles, taking into account the need to avoid blocking and satisfy overall production objectives.

All these aspects of an AGV system must be taken into consideration when building the simulation. In fact, they are interconnected problems and must all be dealt with cooperatively to ensure a smooth operation with maximum efficiency.

Dispatching

The assignment of a vehicle to a task is the first step in vehicle management, and thus, must be done carefully and in consideration of the chosen performance measures. At the start of the operation or after finishing a task, that is, when the vehicle is free of any responsibility, a task must be assigned to it.

There are two ways of looking at this problem [17]:

- The vehicle initiates the assignment: this happens when there is more than one station awaiting a vehicle. In this case, the vehicle chooses a station following a specific heuristic, such as shortest travel distance, maximum queue size. A common rule is what is called the First Come-First Serve rule, where the vehicles are assigned to the stations in chronological order of when the call was placed;

- The station initiates the assignment: this occurs when there is more than one available idle vehicle. The vehicle is again chosen by a set heuristic, such as nearest vehicle or least utilised.

The measures used to evaluate performance are not always the same and it can be, for example, distance travelled by the vehicles or machine utilisation. In general, not a single dispatching rule is going to please all performance evaluators. One possibility is to consider multi-attribute models, for example, combining both distance of vehicle to station as well as the station output queue [18].

2.2 Routing Problem and Path Planning Algorithms

The routing problem is a very important one when managing an AGV fleet. If poorly conceived, the vehicles take a much longer path than necessary, or never even arrive, which makes the system not efficient at all. The routing issue is deeply connected to the layout of the work space, as it directly affects the number of possible paths and greatly influences the decision regarding which routing strategy to adopt.

A possible approach is to send all vehicles in a loop. This eliminates the existence of intersections and, consequently, the possible collisions and blocked zones. In the case of only one vehicle per loop there is very little path planning, as there is only one possibility, entering the loop, which can move in one or both directions. If multiple vehicles are used in a loop, then, traffic problems may arise, even though at a lower complexity level than other configurations. While there is less planning, for both paths and traffic control, the distances travelled by the vehicle will, in most cases, be longer than necessary, which makes this strategy less interesting. Also, it is only possible if the layout supports it. If the stations are not laid out in a way that makes a loop feasible, such as having a station in the middle of others, this strategy becomes even less efficient.

A different method to routing is one where the stations are connected with pre-defined paths. By connecting the links in the mesh and moving from link to link, a path from one station to the other can be found. The rules for connecting the paths can vary, but the goal is to usually find the

shortest distance. In this scenario, the possibility of collisions, blocking and other traffic problems is much higher, such as two vehicles trying to enter the same branch or what link to choose in alternative. For these cases, a set of heuristics regarding traffic control and zone blocking must be applied. Almost any layout can support this strategy as it only requires paths connecting the stations.

The routing can also be done in a layout with no previously and phisically defined paths and in a work space that is free or with obstacles, besides other vehicles. In this case the route taken will usually be a path connecting the two stations and since the only thing to avoid is other vehicles, recalculation will only be necessary if there is another vehicle in the way. There can also be no recalculation of the paths but instead an adjustment of the times in which the vehicles leave the stations, so there are no collisions or even avoidances [19].

When conflicts arise, such as deadlocks, rules must be in place to detect and deal with them or even to avoid them altogether [2]. Intersections can be a problematic area and methods for how to deal with situations when more than one vehicle wants to cross can be implemented, such as a buffer area or something similar to a traffic light. A possible approach is called forward sensing and it uses sensors that detect the distance between the vehicle and its surroundings and stop or redirect the vehicle when the distance reaches a defined threshold [20]. This method allows the vehicles to be closer to each other and a greater density in an area, but has the drawback of usually only looking in front of the vehicle and so it is only more useful in straight paths.

Zone blocking is a technique that can help avoid deadlocks. The path is first segmented into different zones and the rule is that only one vehicle at a time can be inside the zone. A vehicle that happens to be following another must wait in the previous zone until the other vehicle has left its desired area before it can move. In static zone planning, when a vehicle reaches a zone it plans to enter, it first checks if there is already another vehicle there and if there is, then it must wait for it to leave or find a new path. If the zone planning is dynamic, then the zone can change according to flow of the traffic or where the vehicle is. The control can be done by a central station that tracks each vehicle and communicates when can they enter a zone or it can be done by the vehicles themselves. The more zones there are in the work space then the more mobility there is [2].

Path Planning Algorithms

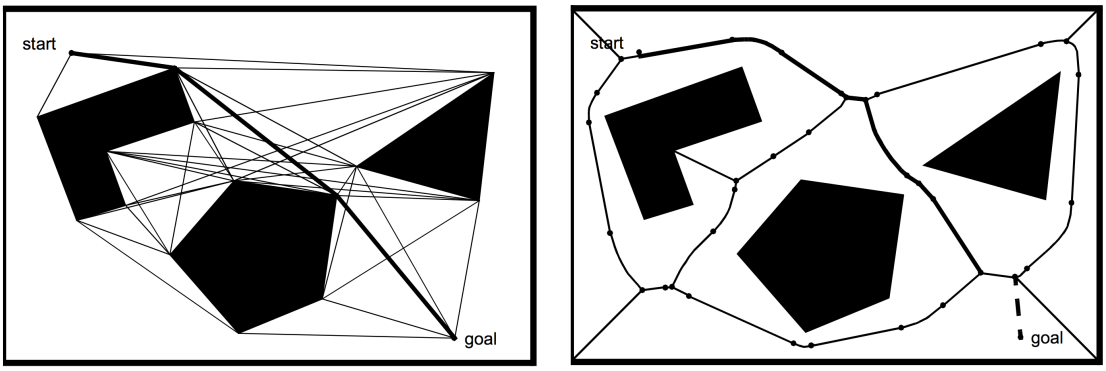
For the vehicle to navigate in an open space configuration it must be able to find a path connecting the origin and the destination and this is where path finding algorithms prove useful. While this subject relates to any sort of map and path finding task, it can also be applied to the specific problem of AGV routing.

The first step of any path planning process is the map, a representation of the environment, which can be pre-existing or built in real time. For the algorithms to be applied, the map, possibly continuous, must first be discretised. The way in which these maps are used is what differs between strategies.

Using graph search techniques, a connectivity graph in free space is built and then searched for the best solution. The graph construction starts with a representation of both occupied and free space, which is then decomposed into a map where it is possible to apply the algorithms to [21].

The **visibility graph** (figure 2.6a) is a method in which the obstacle's vertices are connected to all other vertices that are visible, including the start and destination vertex as well. These lines are always the shortest distance possible between vertices and the algorithm's task will be to connect the start and goal positions, using these roads, in the shortest distance possible. The **Voronoi diagram** (figure 2.6b) takes an opposite approach and roads are placed as far as possible from the obstructions. While it is more easily executable it is also less optimal as to path length.

Exact cell decomposition is an approach where the cells, or nodes, are divided by geometric features, making it that each cell is either completely free or occupied. What matters then is the ability of the vehicle to move from one cell to the other, not where it is on the cell. Due to its complexity to implement, it is not often used in mobile robotics. On the other hand, **approximate cell decomposition** is one of the most popular techniques, due to its grid based representation and ease to implement. In this case, the decomposition is done by recursively decreasing the cell size, generating four new rectangles for each free cell and considering free only the ones that are fully unoccupied. This method is also known as quadtree.



(a) Visibility graph (b) Voronoi diagram

Figure 2.6: Comparison between visibility graph and Voronoi diagram of the same space [21]

The next step after building the graph is to explore it and connect the start and the goal in the best way, usually the shortest path, but other criteria can be used, such as time, energy or risk. The performance of these algorithms is also affected by the graph construction and search technique used. Since these algorithms always guarantee an optimal solution, due to exhaustively searching all possibilities, the difference between performances is how long it takes to complete the search and in what context can it be used. They can either be deterministic or randomised.

Breadth-first search begins with the start node and then explores all of the subsequent nodes by visiting all its neighbours until it reaches the destination node. The best path will be the one with the least amount of edges leading to the goal, assuming all edges are of equal length. Similar

to this algorithm is the **depth-first search**, which instead of exploring the neighbours, goes to the successor node first until it reaches the deepest level [22].

Dijkstra's algorithm [23] is nearly identical to breadth-first, but with the important consideration that edges may take any positive value, making the optimal path not one with the least edges but one with the least total cost. The resulting optimal path to the goal is not only valid from the start but also from any other node, allowing the vehicle to reach the goal without recomputing the solution, as long as the environment stays unaltered. A flowchart describing this algorithm can be seen in figure 2.7.

The **A* algorithm** builds off the Dijkstra algorithm with the addition of heuristics and is mainly used in grids. The heuristic function gives more information about the cost of the movement. Beginning at the start cell, each neighbouring cell is ordered by lowest total cost, defined by travel plus heuristic cost, the travel cost being the distance from the start cell to the next cell and the heuristic cost the distance from any cell to the goal cell. The lowest cost node is then expanded and explored until it reaches the goal node [24].

An existing variation of A* is the **D* algorithm**, where the algorithm reuses previous searches into the new iterations. The advantage is if there are any changes to the environment observed by the vehicle, an entirely new solution is not necessary (as it would be with A*), only cells that were altered need to be recomputed, largely decreasing the computation time required. There is also a variation where the replanning is done anytime for both A* and D* [25].

A possible approach to path planning is sampling-based, a concept that allows for quicker and more efficient answers to planning queries, as well as being able to take into account many degrees of freedom and differential constraints. It also uses less memory than other approaches. This approach, unlike graph search techniques, does not require a special map to be previously built. The algorithms only need to know what space is free and what space is occupied, and so, no previous map building is required. The process consists of randomly placing valid vehicle configurations along the state space and then connecting these with collision free paths. This means that a larger amount of samples lead to a better solution, and as the number of samples approaches infinity a valid solution is guaranteed to be found, as long as one exists.

A **probabilistic roadmap (PRM)** [26] is one of the earliest sampling-based motion planners and more appropriate for multi-query planning. It works by randomly and uniformly placing nodes on the free space and then connecting them, thus forming a graph. Since the free space is not actually known beforehand, when a configuration is placed on the workspace it is then checked if it is valid or not, and then retained or rejected. The manner in which the state space is sampled can also be changed to find the most appropriate strategy. When the pretended number of free samples has been found, the algorithm will then try to connect each sample to a chosen number of the nearest samples. Using interpolation, if a collision free path is found, the edge is added to the roadmap. Once the graph is completed the start and goal states are connected to the nearest sample and then searched for the best solution. An example of how this works can be seen in figure 2.8

Tree-based planners are another type of sampling-based planners and it is an approach many algorithms use, with **rapidly-exploring random tree (RRT)** being one of the more commonly

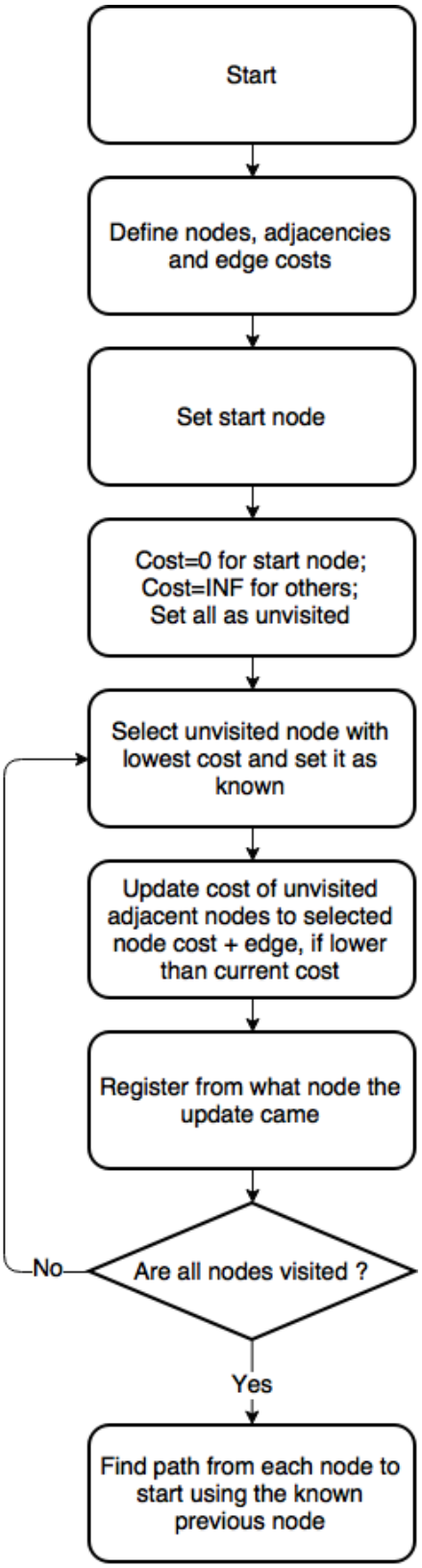


Figure 2.7: Flowchart describing how Dijkstra's algorithm explores a graph

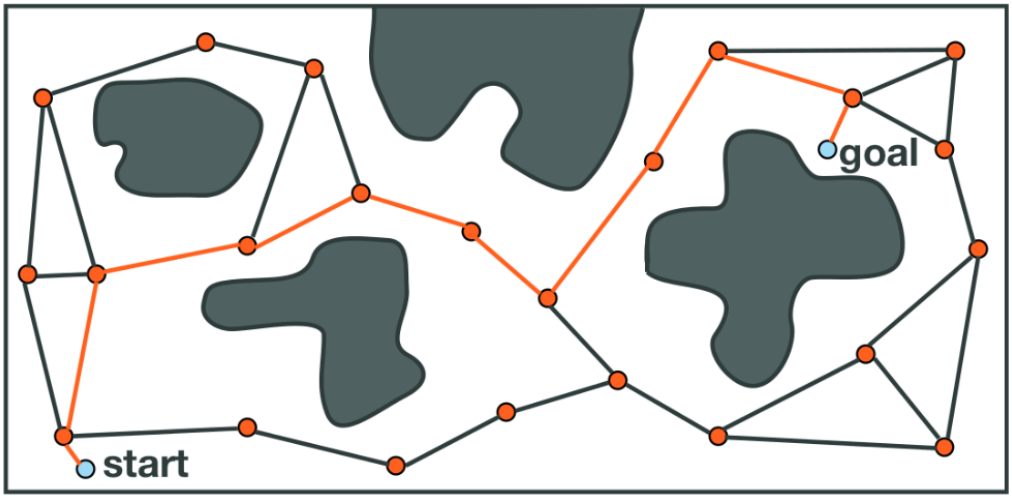


Figure 2.8: Example of a path finding task using a PRM algorithm, in [27]

found [28]. They are more suited for single-query planning. The planning starts by placing a node at the start state and from there expanding a tree by sampling the free space around it. The way this is done is what differentiates the many algorithms, each using a different heuristic. The expansion can also be biased towards the goal state for a better and quicker solution. This expanding tree method can be visualised in figure 2.9

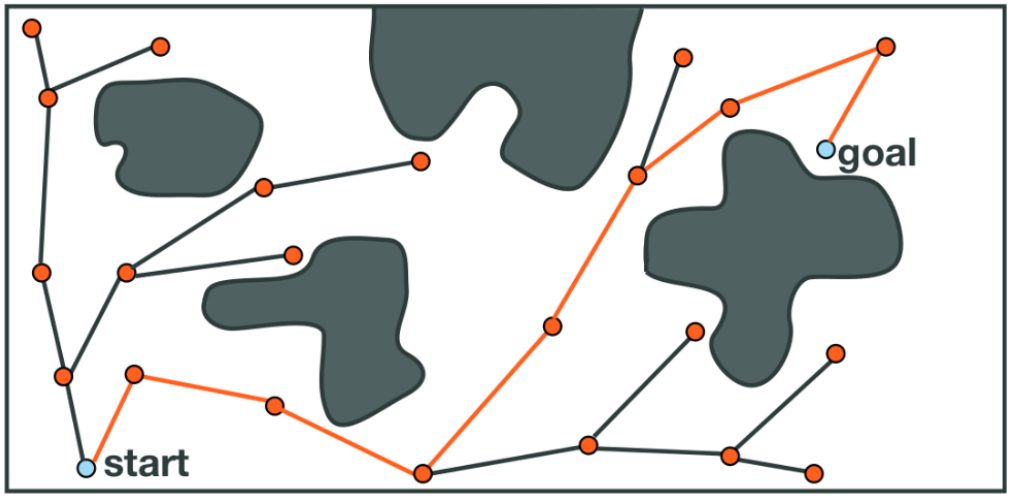


Figure 2.9: Example of a path finding task using a RRT algorithm, in [27]

A variation on some algorithms such as the RRT and PRM is the **RRT*** and **PRM***. The appended star means that the algorithm presents optimal solutions. Due to the random nature of the sampling based algorithms, the nodes end up not being placed in a way that create the shortest possible path, but instead in a more spread out way. This star variations try to smooth out the graph by connecting nodes in shorter distances. If the number of nodes approaches infinity then

the solution is guaranteed to be the shortest one possible. Other variations are possible, such as the **RRT-Connect** [29], where two trees are built simultaneously, one from the start and other from the goal and advance towards each other or the **Lazy-PRM** [30] where the collision checking is reduced to a minimum to save computation time.

Potential field planning is a completely different method of path planning, where a mathematical function is applied to the free space. The algorithm treats the vehicle as a point under the influence of a potential field, where the obstacles are repulsive forces and the goal is an attractive force. While simple and intuitive, due to its limitations, such as the vehicle getting into trap situations and difficulty with narrow passages, it is not as commonly used [31].

2.3 Simulators

Simulators are programs that allow the user to test and create behaviours for robots and other components, without having to physically interact with them. Depending on the software and robot, some may allow for the applications created on the simulator to be transferred to the physical robot, possible for example, through ROS (Robot Operating System) [32].

Most modern simulators include a variety of features that enable realistic and useful simulation such as the ability to display 3D models of robots and environments, either with included modelling tools or the capability to import external models, physics engines, calculation modules and scripting of behaviours with commonly used coding languages.

The benefits of using a simulator are immense, in the way that it reduces costs, saving time and money, allowing various alternatives to be tested with no costs or risks and no down-time. It also allows the user to optimize the number of vehicles needed, the most problematic areas or what management strategies to use.

2.3.1 Webots

The software Webots was created in 1998 by Dr. Olivier Michael and is used mainly for educational purposes [33]. Included in the program is a wide collection of robots, sensors and actuators.

It is one of the most commonly used simulation softwares in many fields, from research on robot locomotion and simulation of adaptive behaviour to teaching and robot programming contests.

The software is cross-platform and supports languages like C/C++, Java, Python and MATLAB. The rendering is made using the OGRE engine and uses a custom version of ODE as its physics engine. It also includes an internal 3D modeller, as well as supporting ROS.

The software is proprietary and can be downloaded for free with limited functionality. To access the full version it is necessary to purchase a professional or educational license. An example of a Webots scene can be seen in figure 2.10

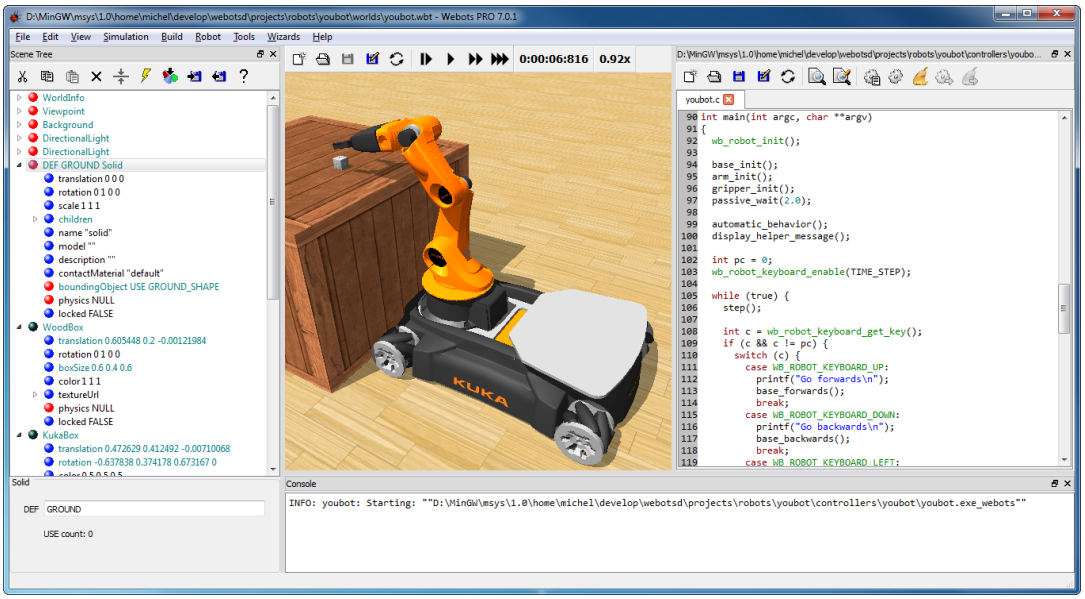


Figure 2.10: Webots graphical environment [33]

2.3.2 Gazebo

Gazebo is a robotic simulator released in 2002, developed by Open Source Robotics Foundation [34]. It is a multi-robot simulator with dynamics, with the ability to simulate multiple robots, objects and sensors in complex environments. It is also able to generate realistic sensor feedback and interactions between objects as well as simulate rigid-body physics.

The graphics are rendered using the OGRE engine. The dynamic simulations can be made using one of the four included physics engines: ODE, Bullet, Simbody and DART. The main programming language is C++ and plugins can be developed using its own API. The simulations can be run on remote servers using TCP/IP or on a cloud.

Gazebo is open-source, available for all platforms and has a very active community, with an on-line simulation model repository, forum, wiki and library for robot applications. The same company also developed ROS, a framework for writing robot software. An example of a mobile robot in Gazebo can be seen in figure 2.11.

2.3.3 V-REP

V-REP (Virtual Robot Experimentation Platform) was created by Marc Freese and first launched in 2010, making it one of the most modern simulators available [35]. It can be used for many applications such as fast prototyping, simulation of automation systems and teaching.

The software offers several calculation modules for object interaction. For dynamic interactions, four different physics engines are available, Bullet, ODE, Vortex and Newton. Other calculations modules are: kinematics, collision detection, mesh-mesh distance and path/motion planning.

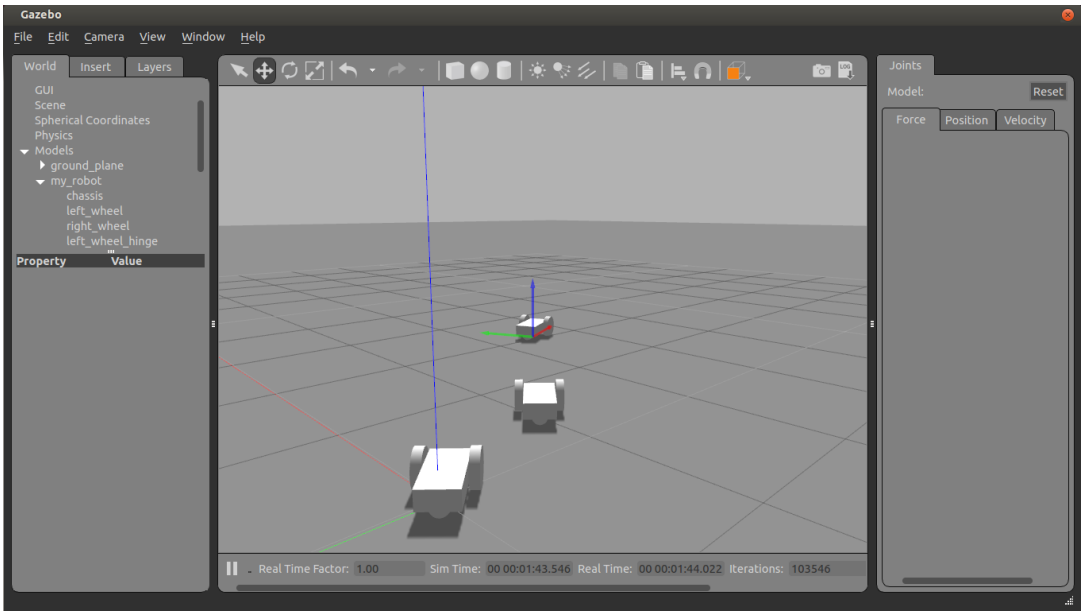


Figure 2.11: Gazebo graphical environment [34]

V-Rep allows the user to choose among various programming techniques, such as:

- Embedded script, the main feature of V-REP, coded in Lua;
- Add-on, also in Lua;
- Plugin, using C/C++;
- Remote API client, in C/C++, Python, Java, Matlab and Urbi.

Included in the program is a large collection of commercially available robots and sensors, as well as the ability to import new models or create them using the integrated modelling abilities. Using ROS it can also connect to actual robots.

The program is cross-platform and available for free to students but must be purchased for commercial purposes. Available as well is a free player with limited capabilities. A public forum and help system are accessible on their website as well as an active on-line community. A scene containing mobile robots can be seen in figure 2.12.

2.3.4 Simulators Comparison

The robotics simulators previously mentioned are some of the most commonly used and advanced available on the market. While there are other programs that can do similar tasks they have not been considered, as they are not as accessible or interesting as the others presented. While many AGVS companies offer their own simulators and managers, these are not available for public use.

As to programming, Webots allows the user to choose from four different programming languages, while Gazebo and V-REP mainly use only one language. Even with this limitation, V-Rep still allows for the user to employ different programming techniques.

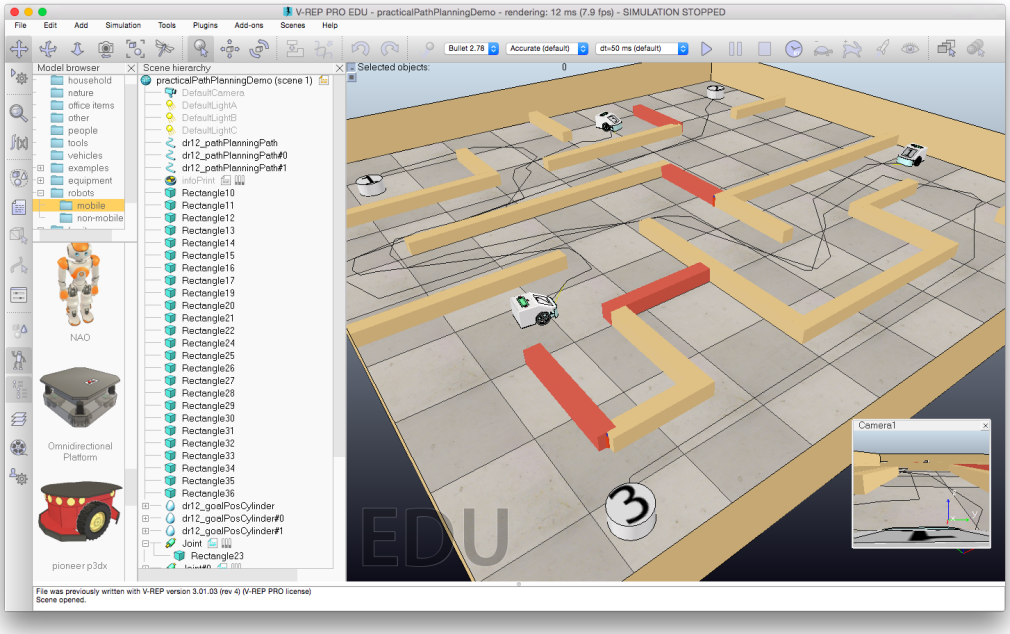


Figure 2.12: Example of a V-REP scene

V-Rep and Gazebo have four different physics engines available and when compared to Webots, that only has one, it shows one of the limitations of Webots.

Webots is also the only that requires a paid license, while the other two programs have full versions available for free to students.

Considering the available physics engines and calculations modules, programming techniques, versatility of both environments and robots and availability, V-REP has advantages over the other simulators. It is also more user-friendly while still including more features than the others as well as being less hardware demanding [36].

Chapter 3

Single Vehicle Routing Problem in V-REP

In this chapter it is discussed how the V-REP software works and how it can be used to model a single vehicle system. The different components of V-REP, as well as the Open Motion Planning Library (OMPL) tool, are analysed as to how they work and can be used to the advantage of modelling any subsequent system. The single vehicle system is then built, detailing the process from the construction of the layout and vehicles to the creation of the scripts that control the model. This example is then used to test and characterize a few aspects of both V-REP and the scripts created.

3.1 V-REP

V-REP, or Virtual Robot Experimentation Platform, is a very versatile software capable of many different things within the world of robotics and due to the freedom of use given by the scripts and other programming features, almost everything you would want to do, can be done. An important aspect of V-REP is also its many calculation modules.

For clarity, the version of V-REP used in this dissertation is 3.3.2 and all simulations are executed on a computer running OS X, with 8 GB of RAM and an SSD.

V-REP is composed of three central elements that together control all aspects, graphical and logical, of a scene. The components, which are subsequently detailed, are:

- Scene objects;
- Calculation modules;
- Control mechanisms.

All the information in this section, and all others regarding V-REP, come from the user manual [37], the official website [35] and official forum [38].

3.1.1 Scene Objects

Scene objects are the basic building blocks of V-REP and from which other capabilities are built upon. Everything that can be visualised in a scene is an object and they can either be static or, with the help of scripts, dynamic and can be used to read data, interact with each other and many other functionalities. In a scene, objects can be in a hierarchy (e.g. figure 3.1), which can influence their behaviour, or independent of other objects.

The fourteen scene objects available on V-REP are:

- **Shapes:** shapes are made up of triangular meshes and used for both visualisation and rigid body dynamics. Through V-REP, only primitive shapes, such as cuboids and cylinders, can be created and along with convex shapes these are optimal for a faster dynamic collision response. Shapes created using an external 3D CAD software can be imported into V-REP, first being converted into a triangular mesh. All shapes, including *heightfield* and *random meshes*, can be grouped and combined to create more complex shapes, such as vehicles and manipulators. Calculation modules, such as distance measurement and collisions, depend on shapes, and how they are modelled, to work properly. Shapes, and other objects with a physical presence on the scene, have special properties that can be customised, such as being collidable, measurable, renderable, detectable or visible. The graphical aspect of a shape can also be altered, for instance, its color or texture. The shapes can be created in the user interface, but may also be generated, and altered, during a simulation using commands like *simCreatePureShape* in the script;

- **Joints:** joints can be of four types, revolute, prismatic, spherical or screw. Joints link two or more objects together, with varying degrees of freedom, and can operate in different modes, such as force or inverse kinematics;

- **Proximity sensors:** proximity sensors measure the minimum distance from the sensor, which is usually attached to a different object, to any other viewable object contained within the configurable detection volume. This leads to a more realistic and continuous detection, unlike ray-type sensors;

- **Vision sensors:** vision sensors permit the extraction of complex data regarding an image, such as size, colors or depth. In conjunction with a plugin and integrated image processing, it is possible to filter and analyse the image;

- **Force sensors:** force sensors measure force and torque and are represented by rigid links.The links, depending on conditions, can break apart;

- **Graphs:** graphs are able to record and register a variety of data, such as time, X/Y and 3D curves;

- **Lights:** lights illuminate a scene, or even individual objects, and can be spotlight, directional or omnidirectional;

- **Cameras:** cameras allow the visualisation of scene from a specific angle and position. They can also track an object or auto-fit a scene;
- **Mirrors:** mirrors function as real mirrors and have a clipping function;
- **Dummies:** dummies are used as a reference frame and commonly attached to other scene objects. Usually, scripts are attached to them as dummies have no function by themselves;
- **Mills:** mills are customizable convex volumes that are used to simulate cutting and milling operations;
- **Paths:** paths allow other objects (mainly shapes) to perform a defined movement in space (orientation and position). Can be used, for example, for conveyor belts or vehicles;
- **Octrees:** octrees are a voxel based spatial partitioning of a shape. It can be used for faster calculations or a simplified representation of a complex mesh;
- **Point clouds:** point clouds are point containers and similar to octrees. Also used for faster calculations.

Paths

Paths are a very important, and often used, scene object in all test cases created along this dissertation and so, deserve a longer explanation.

The path object is a continuous line shaped by its *control points* and the successive linking of them. Each *control point* has certain properties, besides the more basic position and orientation, that can be altered to influence the shape of the path. One of the more important properties are the Bezier points, which dictate how the path shape behaves when connecting non straight *control points*, that is, the interpolation between points, in both translation and rotation, as shown in figure 3.2. Besides path shaping properties, the *control point* also contains information regarding virtual distance (can be useful to create a pause point along the path), relative velocity and auxiliary flags and channels.

The path can be edited manually (point by point), imported (also exported) from an external file or generated from the edge of a shape. The manual edition can be done before the simulation, using the path edition mode, or during the simulation in a script using *simCreatePath* and then *simInsertPathCtrlPoints* to insert each individual *control point* that form the desired path.

The only scene object that can actually perform any movement along a path is a dummy, and so, for a shape, or other object, to move along it, it must be attached to the dummy. The commands used to follow a path, such as *simFollowPath* or *simMoveToObject*, also allow the user to define a velocity and acceleration, which are associated to the command and not the path itself. The commands allow the user to choose how much of the path is completed, in a percentage, and it is not required to fully complete the path. It should also be noted that these path following commands are blocking operations, which means that while they are running, the child script is stopped and

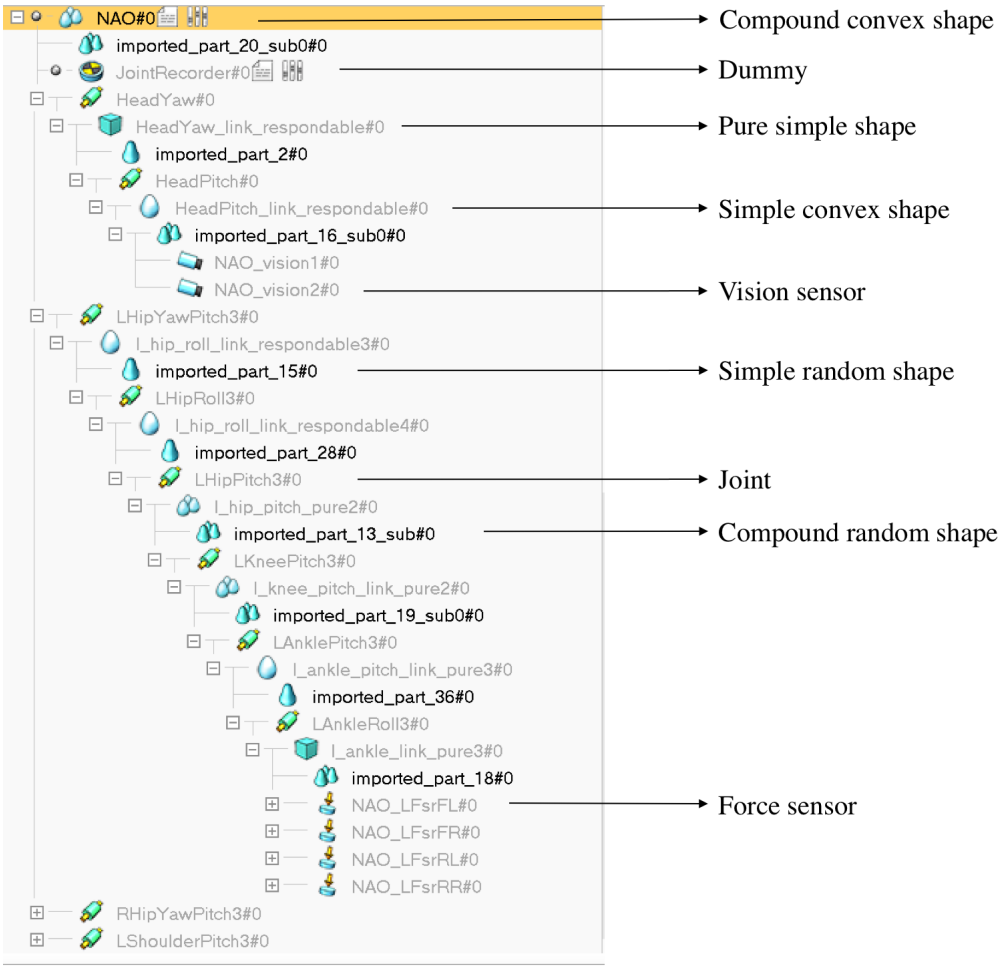


Figure 3.1: Hierarchy of the NAO robot implementation in V-REP, composed of shapes, dummies, joints, force sensors and vision sensors

can't complete any other operations such as collision checking or register information. A possible way to get around this limitation is to make the dummy move along the path in small increments and between motions perform any other function desired.

In V-REP, scene objects can also be models, either the object by itself or the objects below it in its hierarchy tree. If an object is flagged as being a model base, which can be done in the object dialogue box or in the script using *simSetModelProperty*, then it can be saved as an external file with the extension *.ttm*, using either the user interface or the *simSaveModel* command. The models may then be loaded and used in a different scene. Applying this to *paths* means that they may be calculated in a scene and saved for future use. Each path model is quite small in size and takes up about 50 to 70 kB.

The length of the path can be calculated in seven distinct ways, which are different combinations of the sum of the linear and angular variations between *control points*. The method in which the distance is calculated can influence the performance of the path following commands.

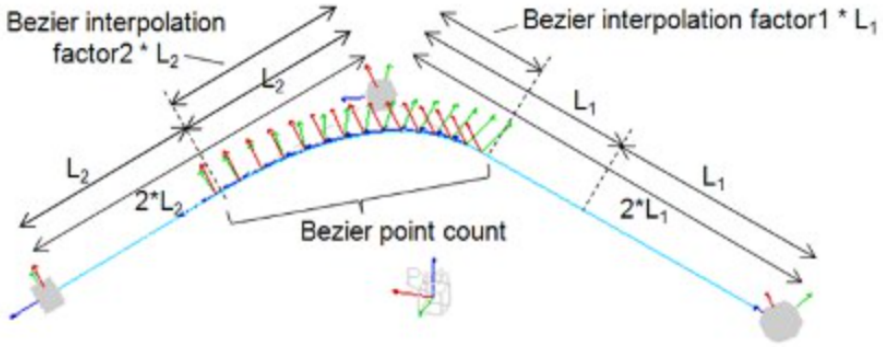


Figure 3.2: Demonstration of how bezier interpolation works in V-REP [37]

3.1.2 Calculation Modules

A scene object by itself does not have much use and therefore needs an algorithm or command to make it interact or move. The calculation modules are no more than algorithms that come pre-built in V-REP and can be modified and implemented through a user interface window, as seen in figure 3.3. The advantage of having these modules already built-in is that these basic functions, which are often used in any simulation, require no external libraries, plugins or even coding and so makes any scene portable and easy to share. The calculations modules are also accessible by the scripts, and while they may be used exclusively through the user window, for a more complete and dynamic usage the commands available through the scripts are necessary.

The five available calculation modules are:

- **Collision detection module:** the collision detection module works by checking for interference between shapes or collection of shapes, that is, a collision exists when there is an overlap of shapes, octrees or point clouds. This collision module is independent of collision responses from the dynamic engines;

- **Kinematics module:** the kinematics module works for both inverse and forward kinematics of any mechanism, such as redundant, branched or closed. It supports conditional, damped/undamped and weighted resolutions, as well as collision avoidance. Also available is a geometric constraint solver, that performs the same tasks as the kinematics module in a more intuitive way for the user, but is slower and less precise;

- **Distance calculation module:** the distance calculation module can quickly measure the minimum distance between any two meshes or collection of shapes;

- **Path/Motion planning module:** the path and motion planning module can handle holonomic and non-holonomic tasks and uses an RRT approach. While it is still available, V-REP developers no longer recommend it as it has been replaced by the OMPL plugin, which is further detailed below;

- **Dynamics module:** the dynamics and physics module is what allows the realistic handling of rigid body interactions, such as collisions and grasping. Four different physics engines can be used for this purpose, the *Bullet Physics*, the *Open Dynamics Engine*, the *Vortex Dynamics* and the *Newton Dynamics*. Since physics engines work mainly by using approximations, it is good to use more than one engine to confirm results.

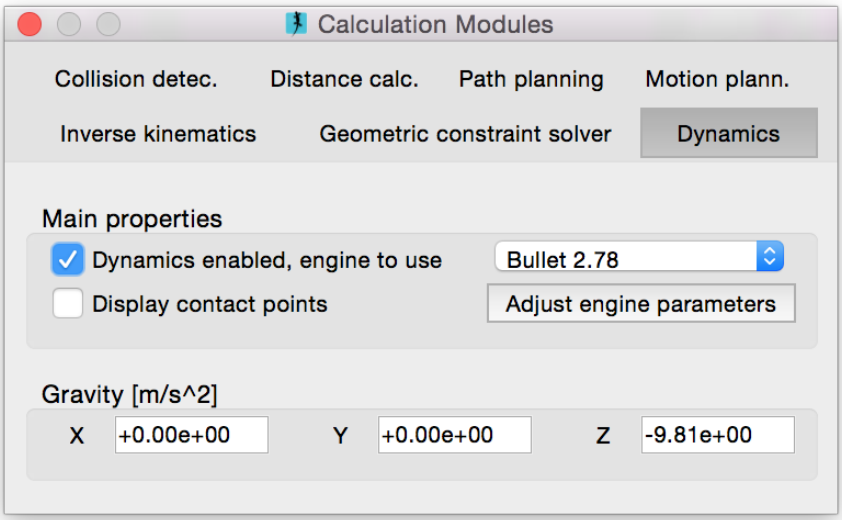


Figure 3.3: User window for the calculation modules, with the dynamics module selected

Open Motion Planning Library

The Open Motion Planning Library (OMPL) is a library that includes a variety of sampling based motion planning algorithms [39]. The library itself consists of only the algorithms with no collision checking or user interface, so it can be more easily implemented with other programs, such as V-REP. While it is not a calculation module, it replaces the previously existing path and motion calculation module.

The implementation of OMPL in V-REP is done with a plugin wrapping the library and is available natively in the most recent versions of V-REP. The library can only be used inside the scripts using functions during simulations and no user interface window for the library is available. To complete a path or motion planning task using this functionality, a few steps have to be taken into account.

The first step is to set a state space. The state space is the space where the algorithm will search for possible configurations. The lower and upper bounds of the search area (or volume) are set, as well as the type of state space, which can be in two or three dimensions, position or pose (position and orientation). In V-REP there are twenty five available sampling algorithms from the OMPL and although any of them can be chosen for any situation, some are more appropriate for

certain tasks than others. No parameters associated with the algorithms, such as goal bias or how neighbours are connected, can be altered through V-REP, therefore, the algorithms must be used as provided.

The next step is to specify which pair of entities are not allowed to collide, such as the vehicle against itself or the vehicle against the environment for example. The start state and goal state also have to be defined. Depending on the type of state space chosen this can be only the position coordinates or the orientation as well. All these parameters are associated to a path planning task, meaning that they can be changed individually at any time, or even have different tasks in the same script. For example, the origin or destination position may be altered and then a new path calculated without having to redo all the steps, since each parameter only has to be defined once.

When all of this is specified the last step is reached, which is the computation of the task itself. The computation is made up of three parts: the solving, simplifying and interpolation of the path. The maximum time allowed for each of the searching and the simplification procedures can be specified, in seconds, as well as the number of states to be returned, that is, how many individual coordinates that make up the path. The result is a vector containing the coordinates of each state in the path in a sequential manner. Using this vector, each point can be inserted as a *control point* in a path object and thus making the result a continuous path that can be followed by a vehicle. An example of how these steps are implemented in code in an embedded script, using regular API functions, can be seen in figure 3.4.

In case a solution is not found, the returned vector is smaller than the set number of states. To make sure a solution is found, a possible fix is to repeat the computation in a loop, until the number of states returned is the desired and the last state is the same as the goal state.

In most situations there may be an infinity of resulting paths connecting the start and goal state, and the outcome of the computation will be only one of them. This, coupled with the fact that sampling algorithms have a degree of randomness associated to them, means that the returned path does not guarantee any optimality.

The above process is the minimum necessary to complete an OMPL task, but further customisation and flexibility may be added through the use of callback functions. The three callback functions available regard state validation, project evaluation and goal state. For example, the state validation callback may be used to force each state to be within a minimum and maximum distance from the obstacles.

While V-REP is completing the search procedure, the simulation clock is paused. This means that from the point of view of the vehicle, and in-simulation parameters, the search is instantaneous, despite of how much real time it takes to calculate. Also important to note, is that the position of the collidable objects that the algorithm must avoid are identified at the moment the path calculation begins. This means that for the OMPL, the obstacle are always static and never dynamic, since if even an object is moving the algorithm is only going to take into account its position at the time it started calculating. The OMPL is a very helpful tool for testing routing strategies that require the vehicles to find new valid paths, and it being already implemented in V-REP means that changing path finding algorithms is easy and fast.

```
t=simExtOMPL_createTask('t') --task that contains all information
ss={simExtOMPL_createStateSpace('2d',sim_ompl_statespacetype_pose2d,robotHandle,{-30,-30},{30,30},1)}
simExtOMPL_setStateSpace(t,ss) -- associate created state space, 2d in pose mode, to task
simExtOMPL_setAlgorithm(t,sim_ompl_algorithm_RRTstar) -- associate algorithm to task
simExtOMPL_setCollisionPairs(t,{lcube,allobst}) -- set collision pairs
startpos=simGetObjectPosition(robotHandle,-1)
startorient=simGetObjectOrientation(targetHandle,-1)
startposition={startpos[1],startpos[2],startorient[3]}
simExtOMPL_setStartState(t,startposition) -- set start state
goalpos=simGetObjectPosition(targetHandle,-1)
goalorient=simGetObjectOrientation(targetHandle,-1)
goalposition={goalpos[1],goalpos[2],goalorient[3]}
simExtOMPL_setGoalState(t,goalposition) -- set goal state
r,path=simExtOMPL_compute(t,5,5,1000)  -- compute path, 5 seconds search and simplification time
                                       -- result in table with 1000 sequential poses
```

Figure 3.4: Example of an OMPL task in an embedded script

3.1.3 Control Mechanisms

To build any complex scene, one that involves movement, calculations or interaction, it requires more than scene objects and the basic calculation modules. To control all aspects of a simulation, a number of approaches can be chosen, each offering their own advantages. For a better user experience, V-REP tries to make any scene as flexible, portable and scalable as possible, so that it can be shared, run on a different computer or platform without any problems [40]. When running a simulation, the code can be executed in three different ways: different machine, computer or robot, connected to the computer where the simulation is being performed; same machine but in a separate process than the simulation loop; code and simulation executed on the same computer. For this dissertation, the third option was selected, for both ease of use and better synchronization between threads.

There are four ways to implement the code and program a model in V-REP and they may be combined, used simultaneously or independently. The way these mechanisms interact is shown in figure 3.5. These methods are:

- **Embedded scripts**: embedded scripts are the main control mechanism and one of the most powerful, allowing the user to program directly on the model, without any external software. The coding language used here is Lua [41];
- **Plugins**: plugins are used to further customize a scene, extend functionalities or register new script commands. Some functionalities, such as the OMPL or the ROS interface, are implemented through plugins. Uses a C/C++ interface;
- **Add-ons**: add-ons can be independent functions or be used alongside regularly executed code. It also uses a Lua interface and can customise the simulator;
- **Remote API**: the remote API facilitates the interaction between external software or machines, such as real robots, and V-REP. This can be done using five different coding languages and on V-REP the functions are performed as regular, but are triggered by the external clients. The external client can also read data streams.

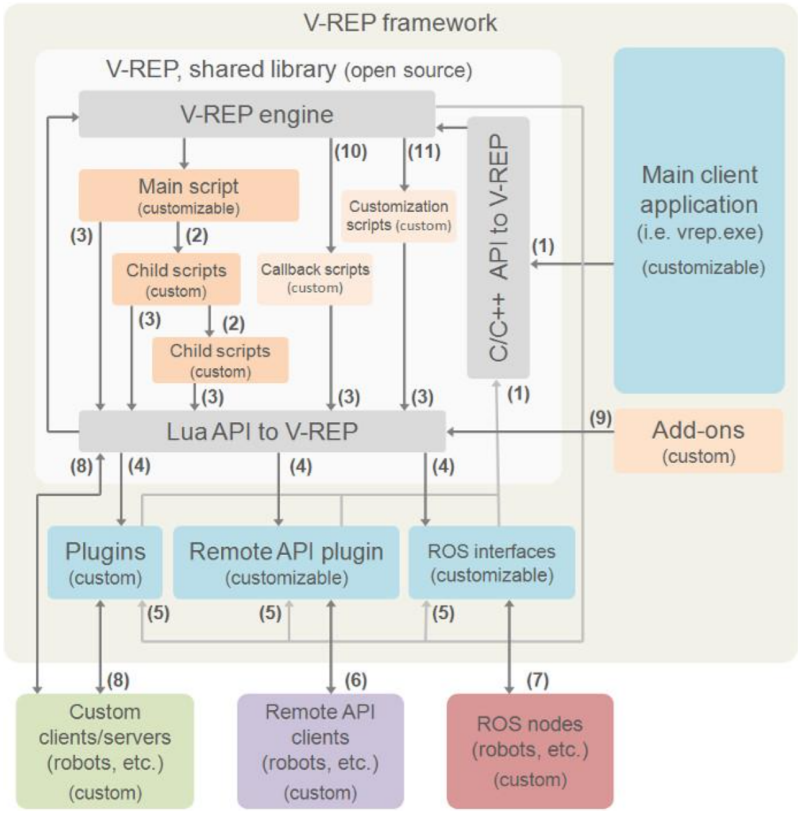


Figure 3.5: V-REP control architecture demonstrating how each control mechanism integrates with the others [37]

Embedded Scripts

Embedded scripts are the main control mechanism used in this dissertation and in V-REP. This strategy has the advantage of easier integration, inherent scalability, no conflicts with different versions, less effort to create, modify and maintain, uncomplicated portability and no problems regarding synchronization. The embedded scripts allow the use of over five hundred API functions, which cover almost any programming possibility, along with the ability to further expand this number using plugins. Embedded scripts are where all other control mechanisms are connected, therefore, are fundamental in any scene.

In any V-REP scene there is always a main script that handles the more basic functionalities (calls all sensing and actuation modules) and also has the function of calling the child scripts. This main script is often left unaltered, as its modification can severely affect the performance of the entire scene, and so, the customisation of the scene should be done in child scripts only. A child script, unlike the main script, has to be attached to an object, usually a dummy, and controls a specific part of the simulation. Each child script is fully self contained and can be easily duplicated or altered, and there can be an unlimited number of them.

There are two types of child scripts, threaded and non-threaded, and they differ from each other in timing. A non-threaded child script is launched by the main script and at each simulation

step is called again. Every time they are called, they perform a task, such as updating parameters or reading sensors, and return the control to the main script. They are usually divided in three parts: initialization, actual task and restoration. The task performing part is repeated every simulation step, while the others only on the first and last pass, respectively. In the case of the non-threaded scripts, the task performing part is usually divided in actuation and sensing.

Threaded scripts are also launched from the main script, but once they start executing the code and haven't finished, the script is not started a second time. This means that the code is executed once from top to bottom per simulation. Threaded scripts can be seen as co-routines which allow them to function alongside each other and supports a greater programming flexibility. Blocking functions can also be used in this kind of script, since it is running separately from the other scripts. Each type of child script offers their own advantage and the choice between them is crucial to the proper implementation of the intended functionalities.

To communicate between scripts, or any other control mechanism, there are specific methods depending on what is pretended. Since in this dissertation the main technique used are embedded scripts, the best way to communicate between two, or more, are the *signals*. The *signals* work as global variables, that can be read by any script. They can be integers, float or strings, and are defined, or cleared, using script functions, such as *simSetIntegerSignal*. At the end of the simulation all *signals* are cleared.

Almost any function available in the user interface window is also accessible from the scripts using commands as well as more functions that are not found in the user interface. This means that a scene could be fully constructed from the scripts alone.

For this dissertation, all child scripts created are threaded scripts. This is due to the ease of programming offered by them and since none of them are a single repetitive task, threaded scripts are more appropriate.

3.1.4 Simulation

When every component of the scene is finally built, then the more interesting part can begin, the simulations. Before running the simulation, a few parameters have to be defined as to better fit the intended results. One of the main parameters when running a simulation is the time step. The time step is the time elapsed between each time the main script is executed. This means that a smaller time step leads to a more precise simulation but will also take more time. The simulation passes per frame, that is, the rate at which the screen is refreshed compared to the main script, can also be altered. The maximum selectable passes per frame is 200.

There is also a real-time simulation mode, where the the software tries to keep the simulation time and real time synchronized by dynamically altering the time step. Also available is a threaded rendering mode, where rendering is done separately from the main script, and while this speeds up the simulation, it can lead to unexpected results of some operations.

Also available are two buttons to adjust (speed up or slow down) the simulation speed. These buttons work by changing the time step and passes per frame, the two parameters that influence

the simulation speed, but never increasing the time step, as that could lead to undesirable consequences. Through these buttons, the passes per frame can be increased to 12800. This will help V-REP execute the simulation faster, since it calculates less frames, while not affecting the scripts performance.

As the goal in the models that will be constructed is a fast and precise simulation and considering the graphical aspect is not as important, then the parameters chosen will be towards a lower time step and more accuracy in results.

3.2 Single Vehicle System Model

The single vehicle system is used to test and better understand how the software and the scripts work together. It comprises only one vehicle but it is fully fledged and contains most features necessary for further expanding the system.

When considering a single vehicle model, the routing problems are not so apparent, as there is no risk of collision between vehicles, traffic blocking or other occurrences. This case tests instead other characteristics of the model that will be useful in more complex systems. In a free range layout, where the vehicles have the freedom to choose any path connecting the origin and destination, the path finding algorithms can take a major relevance in the performance of the system. The performance of each algorithm in equal situations will be evaluated in the following tests. This will reveal which ones perform better in different situations. In the strategy used for the following tests, each time the vehicle has to reach a new destination, a new path is calculated using a path finding algorithm. This means that no paths are previously calculated and that the vehicle is free to use the entire unoccupied workspace.

Also tested in this model are other performance issues related to V-REP, such as how the time step impacts the results, and OMPL parameters, like search time.

For this system, a specific layout and a vehicle are created, along with scripts to control them. The created components of the system are shown in figure 3.6. The relationship between the scripts can be seen in figure 3.7. Provided to the model is the information concerning the location of the stations, number of vehicles to be used and the tasks to be completed. The data obtained concerns mainly the length of the paths obtained and the time to calculate them.

3.2.1 Layout

The layout used in this first case is a flat grid like system and it serves the purpose of testing and better understanding how the model works.

The work space is a 27 m x 27 m area, delimited by solid walls. In this space there are nine locations, or stations, that can be seen as pick up/drop off or load/unload multiple points and their coordinates are the ones the vehicles will be sent to. Each station is spaced 10 m from the adjacent ones and the outermost stations are 3.5 m away from the walls and so forming a 20 by 20 m uniform grid. They are numbered from one to nine, with one being on the bottom left corner, $(0\,,0)$ in x-y coordinates and nine, $(20\,,20)$ in x-y coordinates.

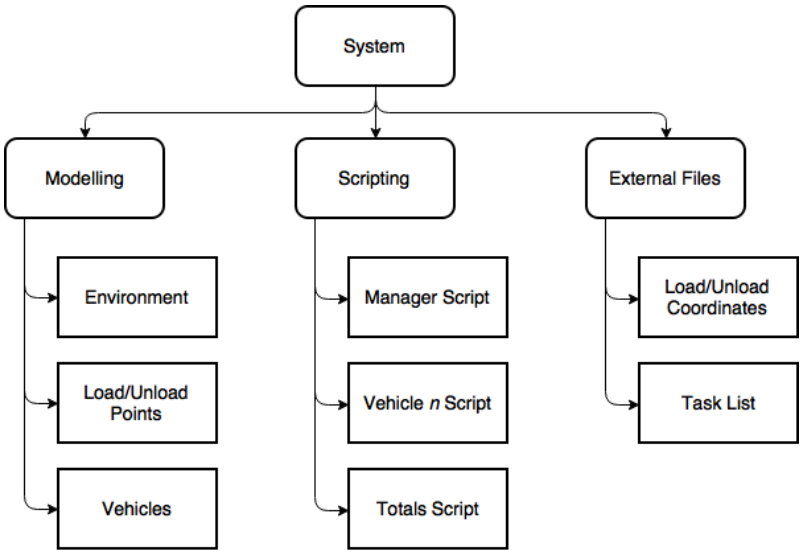


Figure 3.6: The created components that make up the system (*scene*)

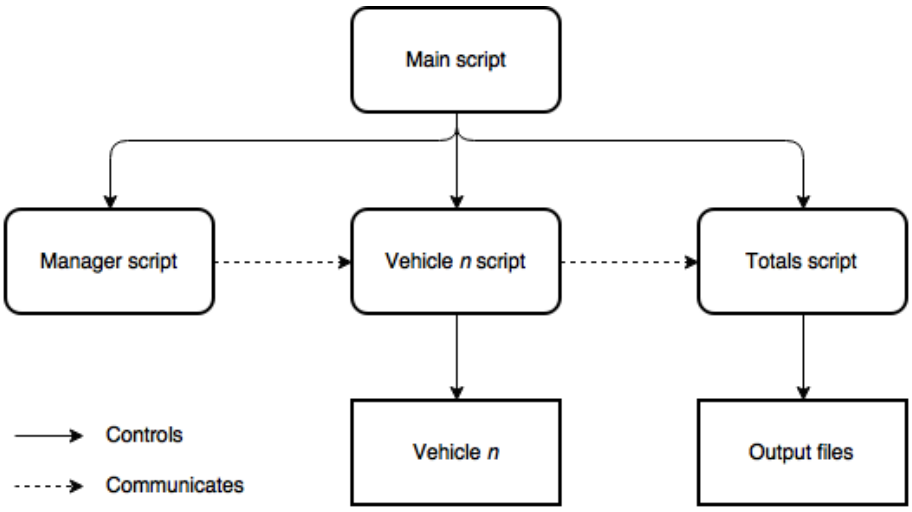


Figure 3.7: Relationship between the scripts, vehicles and output files. The communication between scripts is done using *signals*

Adjacent to the location of each of these stations, except one, is a block, or machine, all of the same shape, measuring 2.5 m by 1.5 m and 1.25 m away from the station. They are placed between the station and the wall, except in the middle. These machines are an obstacle that the vehicle must avoid. In a real world case these could be a robotic arm or a stack of pallets for example. Other than these machines, there are no other obstacles in the workspace. While this may not be realistic, as in a real world case the floor could be more occupied, it fulfils the desired requirements for this test.

On one side of the environment there is an entrance measuring 4 m wide to a parking area where all the vehicles are initially parked in sequence which is always open throughout the simulation. The configuration of the layout can be seen in figure 3.8.

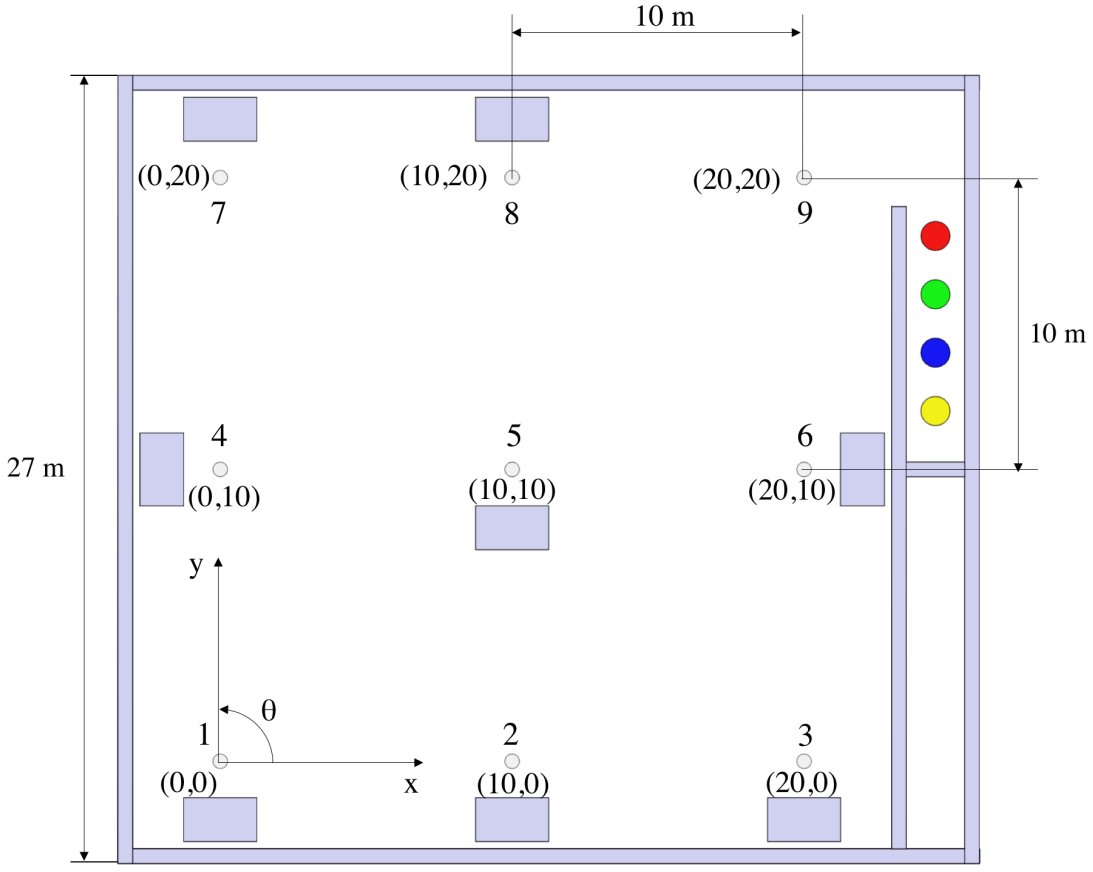


Figure 3.8: Location of the stations in the layout

3.2.2 Vehicles

The vehicles are represented by cylinders with a 1 m diameter and so their orientation is considered irrelevant. Each vehicle behaves as an omnidirectional vehicle and is capable of carrying only a

single load at a time.

Each vehicle has its own colour, which helps identify where which vehicle is during the simulation. A goal shape, equal to the vehicle but with a slightly different colour, is placed on the vehicle current goal station. This helps visualize the path that will be taken by the vehicle, and its current state.

The vehicles use an approach to collision detection similar to forward sensing, presented in section 2.2. In this case the detection area surrounds the entire vehicle, by creating a larger similar shape, with 1.4 m diameter, that moves with the vehicle and is invisible. This larger area is what is detected in the collision module and the OMPL computation.

3.2.3 Manager Script

The *manager script* (figure 3.9) is a threaded script where the inputs are read and the tasks are allocated. There are two kinds of input in the system and this script uses both, where one is on screen and the other is an external file (*.txt* format, in plain text), each containing different information.

The on screen input is a text box where the number of vehicles to be used in the simulation is chosen. The external file contains the coordinates of all the stations, as defined in the layout used, in an $(x; y)$ format, and all the tasks to be completed. Each task is comprised by a pair of stations, one origin and a destination location. The number of tasks has no limit and can be as many as desired. These simple text files can be generated by other programs or integrated into a larger work-flow. For the manipulation of these files, the I/O library existing in Lua was used [42]. These commands are used throughout the scripts when external files are necessary.

The script starts by parsing the external file and storing the coordinates of the stations in a matrix and the origin and destination of each task in a different matrix. After, the number of vehicles to be used is checked and stored. Then it enters a loop where it continually checks if each vehicle is both selected and free. If both conditions are met, it sends a *signal* (a global variable) to the vehicle script containing the coordinates of the station, registers what vehicle completed the task and also signals it to start moving. The loop is only broken when all the tasks are given out. The allocation of tasks to a vehicle is done in a First Come-First Serve basis, which means that as soon as a vehicle completes a task it receives a new task in the chronological order it was placed.

When all tasks are completed, which AGV completed what task is registered on screen and it is signalled that it has reached the end of the script.

3.2.4 Vehicle Script

The *vehicle script* is a threaded script responsible for the movement of the vehicle and also one of the most important. This is where the path planning, blocking and collision detection, for example, are done. Each vehicle has its own script, attached to a dummy that contains, that is, hierarchically above, the vehicle.

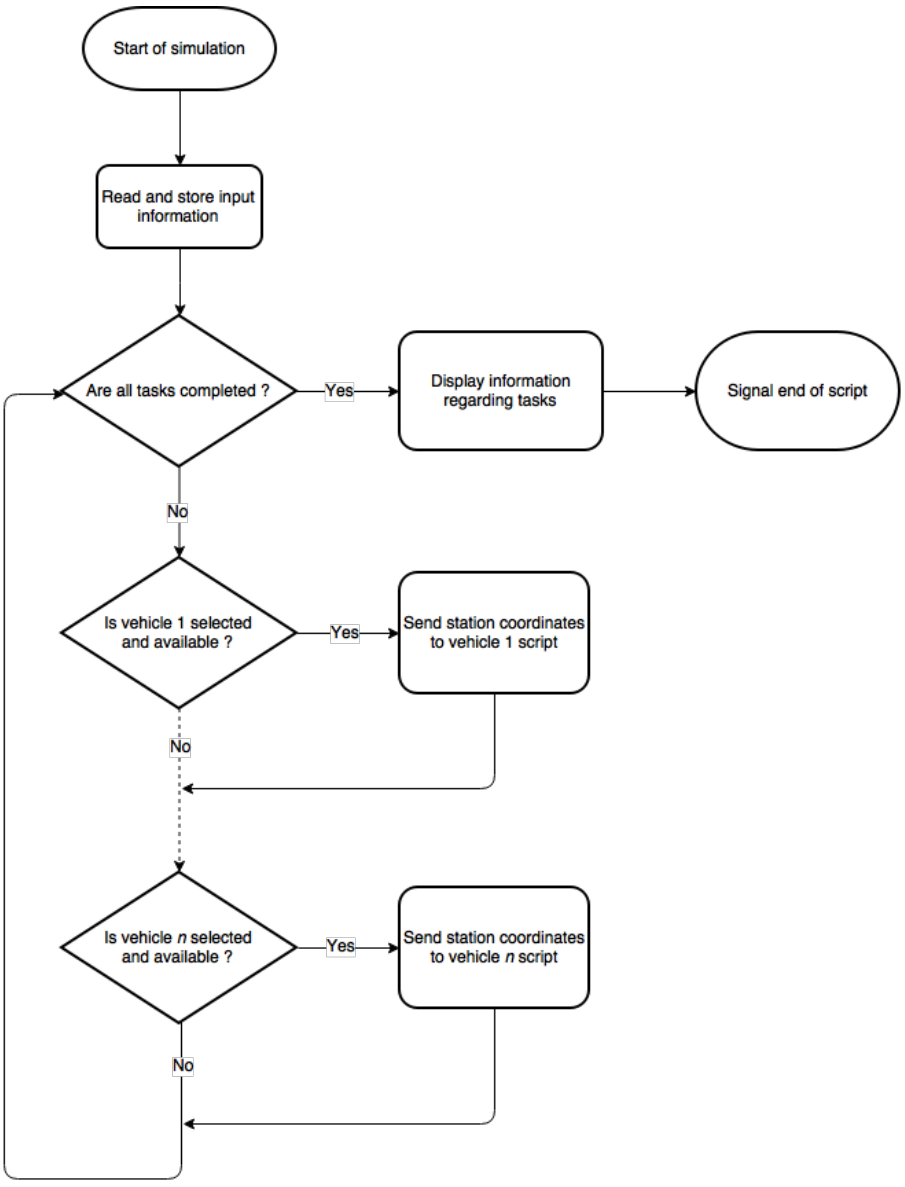


Figure 3.9: Flowchart describing the *manager script* for n vehicles

When the simulation is started, and if the vehicle is selected, it starts by traveling through the pre-defined exit path. This path takes the AGV from the parking area out to the working space and each vehicle has its own path that takes it to its own starting position.

After the initialisation of some necessary coding variables (e.g. time and distance values), the script enters a loop where the work is done and it is broken only when all tasks are completed. It starts by setting a *signal* that it is free and available for work, after which it receives the coordinates of both the origin and destination stations and then again setting the *signal* to busy.

With this coordinates, a path is calculated using the OMPL, as was detailed before in section 3.1.2. If no solution is found, because of blocked paths or inability of the algorithm used for example, it will wait for one second before trying once again, since by then a possible path may exist, as only one vehicle is moving. Since this results in a vector containing a series of positions $(x; y)$, it must be converted into a usable *path* entity. Each position from the vector is inserted into a newly created path as a *control point* using the *simInsertPathCtrlPoints* function, resulting in a continuous line. The same process is repeated but for the same path in reverse, since the *path* object has a defined direction and the path following commands do not allow movement in the reverse direction, which may be necessary in case of collisions.

Now that the paths are created and usable, the script enters another *for loop* where the actual movement is made and ends when the vehicle reaches the destination or a collision occurs. For each increment of this loop, a small percentage of the path is traveled. This is done because of the way the path following command works. This command is a blocking operation, and this means that while it is running, no other parts of the script can run. By moving a small amount each time, the script can now check for collisions or register information during the path. Because it is only a small, constant, percentage, and therefore a small distance, since the total path length possible in this layout is also not a considerably large value, it allows an accurate collision detection. The size of this increment is the resolution at what the vehicle will check for collisions or new orders and so it is desirable to keep it small. It must also be taken into account that the smaller the increment, the longer the simulation takes to execute. The value used in the following models was found through iterative use and experimentation, is in the order of 0.1%.

When a collision is found, detected using the collision module, through the *simCheckCollision* command, the vehicle moves backwards for a short distance, using the previous path in reverse. It then calculates a new trajectory, again using the OMPL, towards the original destination. The current loop is broken and it starts back up from the beginning of the previous loop.

When the destination is reached, the information regarding the task, such as distance and time, is stored in a matrix and it is signalled that the task has been completed and is free and ready to receive a new destination.

When it finishes its current task and receives the *signal* from the *manager script* that all tasks have been assigned, it registers all the information about the orders completed and sends it to the *totals script*. Other output files are produced in the *vehicle script* but are described in detail below in 3.2.5. A flowchart of the script can be seen in figure 3.10.

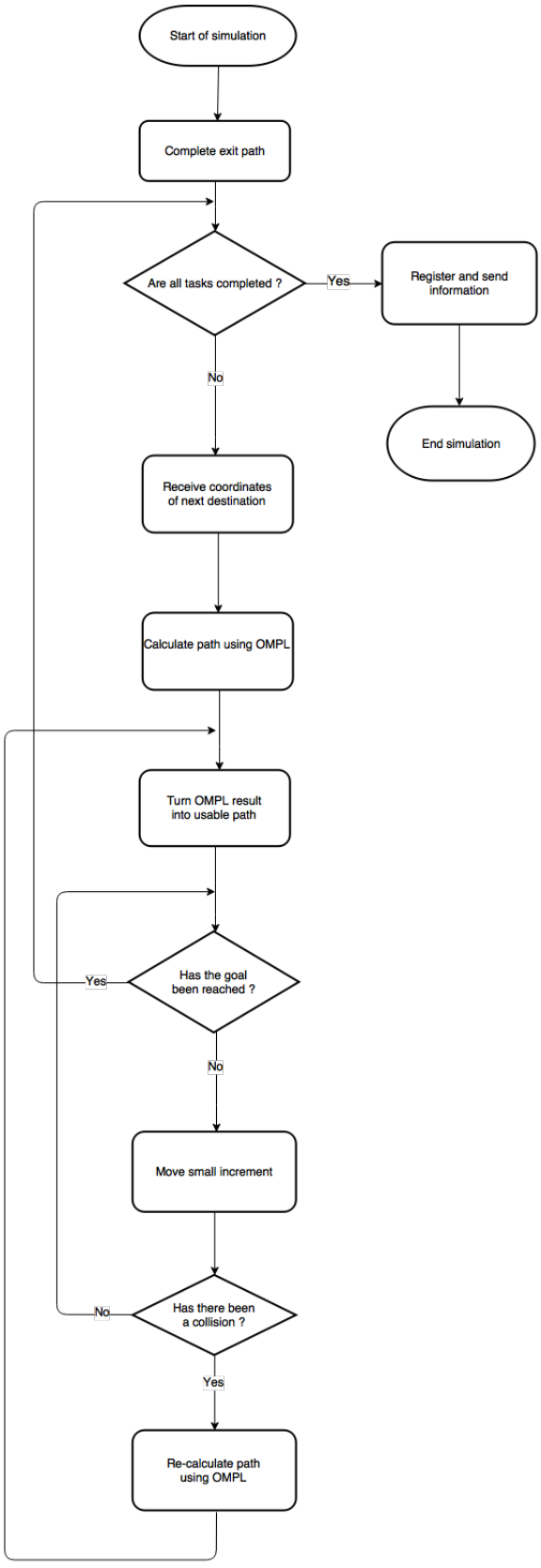


Figure 3.10: Flowchart demonstrating the *vehicle script*

3.2.5 Totals Script and Other Output Files

The *totals script* is a threaded script and is where the results are gathered and some of the output files are produced. Other files are generated in the vehicle script. All of them are in *.txt* format.

When all tasks are completed, the vehicles used send data regarding the tasks each one performed from their own script to the *totals script*. This is done so that all results can be in one organised and readable file. This file contains the time, distance and how many collisions for each individual task, organised by vehicle, as well as the total of each vehicle and what search algorithm was used. In the end of the file there is also the overall information, the sum of all totals, for the simulation. This file is in plain text and easily readable. A different file is also produced containing only the same overall distance, time, collisions and algorithm used in a format that can be easily copied into a spreadsheet, in *Excel* for example, so that further analysis is facilitated.

In each *vehicle script* two other output files are created. One is a file containing the result of all successfully calculated (connecting the origin and destination) paths, consisting of the vector produced by the OMPL, and so is a series of x and y positions. This is formatted in a way that can be easily copied into a spreadsheet. In that spreadsheet, by creating a scatter plot, the path created can be visualised. Before each path, there is information about the destination, origin and task so that it can be conveniently identified.

The other file created in the *vehicle script* contains information about the origin, destination, simulation time, distance, collisions and real time to calculate and simulate each individual movement. This information is, once again, in a table so it can be used in a spreadsheet and more conclusions can be efficiently taken from the data.

All these output files generated by the scripts are in a plain text format and so can be read by any computer without the need of any specific software. At the beginning of all new tasks or simulations it is also registered in all files a time stamp containing the date and hour at which the simulation was started. These files are edited also using the I/O library.

For clarification, when simulation time is mentioned, it means the internal time of the simulation, that starts when the simulation begins and ends when the simulation time is over. Real time means the real world clock, as in the actual time an user has to wait for the simulation to complete. For example, one second of simulation time can take three seconds of real time.

3.3 Single Vehicle System Tests

The first run of tests made has the goal of better understanding how the path finding algorithms behave in V-REP and the model.

The test consists of giving a single vehicle the task of completing both a straight path (station 8 to station 5) and an obstacle avoidance path (station 5 to station 2), as can be seen in figure 3.11. The calculated (graphically) minimum distance for the straight path is 10 m and for the obstacle avoidance path is about 11.53 m.

This will test how the algorithm deals with two kinds of paths that can possibly occur in a more complex situation. It will also give more information on the relation between the calculation time (real time) and the resulting path.

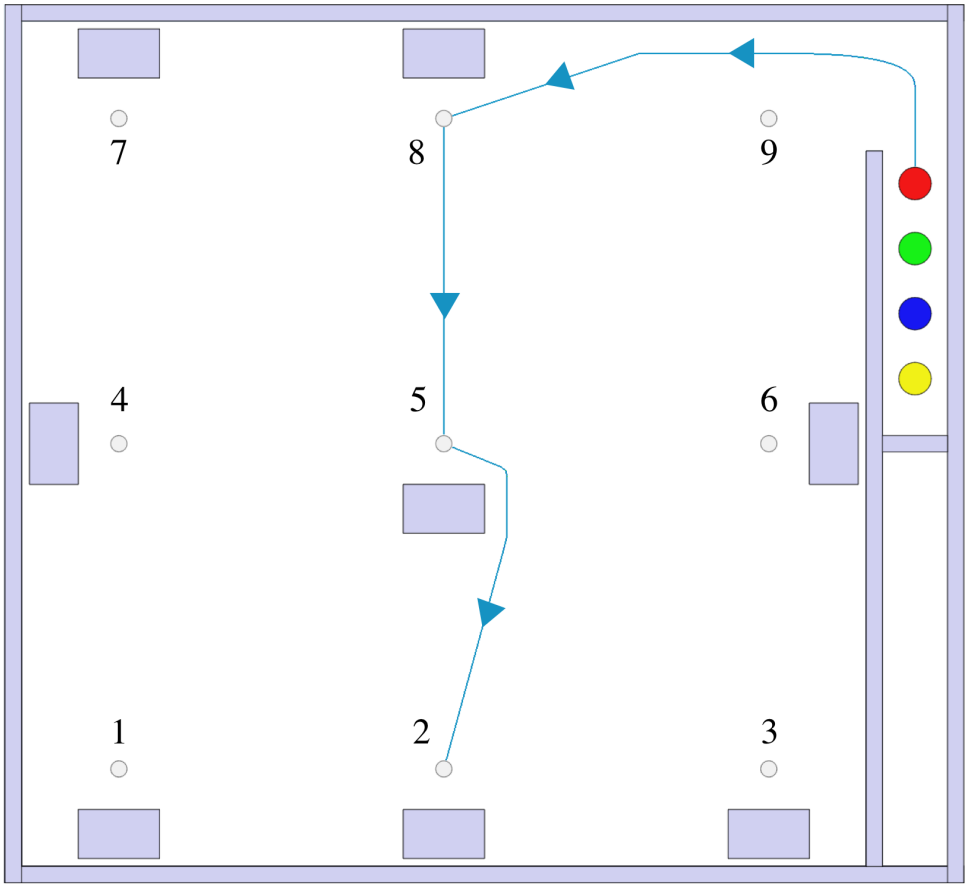


Figure 3.11: Path performed by the vehicle in the first test case

The algorithms tested are:

- RRT*;
- PRM*;
- Lazy PRM*;
- RRT;
- RRT Connect.

These algorithms were chosen because they represent a few different approaches to path planning algorithms, especially ones that attempt to reach optimal solutions. It is expected, due to their optimality, that the star variation of RRT, and all of the first three mentioned, performs better than

the regular RRT and the Connect variant. These two are included as a comparison and to better illustrate the differences between them.

3.3.1 Simulation Time Step

First, a test is made to know what simulation step should be used. The simulation step affects mainly the speed of the vehicles, and makes it more or less accurate. This is due to the fact that the movement is done in increments. The speed parameter of the *simMoveToObject* command is defined as a relation between the increment size and the path's total length.

The target speed chosen for these tests is 1 m/s, as it better reflects the speed of a real AGV. Each test consists of the vehicle going from station 8 to 5 to 2 and this course was repeated five times for each time step. The average speed is calculated by dividing the time by the distance for each path. The results are presented in table 3.1

Table 3.1: Vehicle average speed over five runs for different simulation time steps

Simulation time step	Average speed [m/s]	Standard deviation [m/s]
50ms	0.719	0.032
10ms	0.931	0.006
1ms	0.992	0.000
0.1ms	0.999	0.000

It is clear from the outcome of the experiment that the lower the simulation time step, the more accurate and more consistent the speed becomes. The lowest time step allowed by V-REP is 0.1 ms, but the simulation becomes much more slower to complete. The accuracy for 1 ms is quite good, and with faster simulation times, and so it will be the chosen one.

3.3.2 OMPL Parameters

When using OMPL, the search time for each path is actually made up of two different times, a path search time and a path simplification time. To know what times to use in the test case, and how each one impacts the total OMPL time, an experiment is made to find out. Once again using the same paths, a straight and an obstacle avoidance path and varying the search and simplification time separately. The algorithm used is RRT* and the time step 1 ms. The results, average and standard deviation of the calculation time are presented in table 3.2 for both paths. The first time is the maximum allowed search time and the second is the simplification time.

Looking at the results of the experiment it can be observed that the search time has a greater impact on the total calculation time. For future cases, the simplification time was selected to be the same as the search time, since the total time is mainly dictated by the search time.

The reason why the calculation time is actually higher than the search time by a small margin is partly because of how the time is obtained. In Lua (the programming language used), and the way

Table 3.2: Impact of search and simplification time on the total calculation time for the complete path

Algorithm	Search time [s]	Calculation time [s]	
		Average	St. Dev.
RRT*	(10,5)	10.4	0.49
	(5,10)	5.4	0.49
	(5,60)	5.2	0.40
	(5,0)	5.4	0.49

it is implemented (command *os.time*), the seconds are measured in integers, and due to rounding it can obtain a time one second above or below the actual time, instead of a constant time equal to the search time as would be expected for the (5,0) case. The higher time can also come from the allowed simplification time (which is optional) and since the path is simple, very little time to simplify is used, leading to an increase of less than a second. In a more complex situation, the time taken to simplify could increase.

3.3.3 Straight Line Tests

Each algorithm is run with four different search times and equal simplification times: (60,60), (30,30), (10,10) and (5,5) seconds. Each combination of algorithm and search time is simulated five times. This is done because due to the nature of the search algorithms and their implementation, the results can vary, and this way an average and standard deviation is obtained, which gives a more clear idea of the results. The lower and upper bounds of the search area are set to cover the whole workspace.

All the simulations are ran using the "Fast Simulation Mode", where the screen turns black and the simulation takes less time to run and is implemented through a non-threaded script already present in V-REP and all shapes have the dynamic properties turned off.

For each simulation it is recorded the task, origin and destination coordinates, time, distance, collisions, calculation real time and movement real time. The distances and calculation times obtained for a straight line path are shown in table 3.3.

With these results, a few conclusions can be had. The first is that it is obvious that for any combination of algorithm and search time, for a straight line path with no obstacles, the distance obtained is consistently the same. This can be further proved by comparing the OMPL result output file for any two simulations and checking that, in fact, the outcome is fully identical. It can then be concluded that if the layout had no obstacles, static or dynamic, any algorithm and the lowest search time can be used with no disadvantage as to path length. The only difference comes from how each algorithm utilises the allowed search time, with the star variations taking a lot more than the other two. The last two algorithms, RRT and RRT Connect, lead to the same results while being much faster.

Table 3.3: Distance and calculation time over five runs for different algorithms and varying search times in a straight line (station 8 to 5)

Algorithm	Search time [s]	Distance [m]		Calculation time [s]	
		Average	St. dev.	Average	St. dev.
RRT*	(60,60)	10	0	60.8	0.40
	(30,30)	10	0	33.4	4.80
	(10,10)	10	0	10.4	0.49
	(5,5)	10	0	5.60	0.49
PRM*	(60,60)	10	0	76.0	20.26
	(30,30)	10	0	30.6	0.49
	(10,10)	10	0	10.4	0.49
	(5,5)	10	0	5.60	0.49
Lazy PRM*	(60,60)	10	0	83.2	29.12
	(30,30)	10	0	30.4	0.49
	(10,10)	10	0	10.2	0.40
	(5,5)	10	0	5.60	0.49
RRT	(60,60)	10	0	0.60	0.49
	(30,30)	10	0	0.60	0.49
	(10,10)	10	0	0.60	0.49
	(5,5)	10	0	0.80	0.40
RRT Connect	(60,60)	10	0	0.80	0.40
	(30,30)	10	0	0.20	0.40
	(10,10)	10	0	0.60	0.49
	(5,5)	10	0	0.20	0.40

The calculation times for some algorithms while using 60 seconds of search time is longer than the expected 60 seconds of the previous search time tests. This could mean that in some occasions, the algorithm used some time to simplify the path, despite this not being expected in a simple straight line case.

3.3.4 Obstacle Avoidance Tests

An obstacle avoidance test was also made, using the same parameters as the previous test, presented in table 3.4.

The obstacle avoidance case is what gives the more interesting results, as it is expected that a real life free range layout would have some sort of static or dynamic obstacle in most paths and those would be the ones that present a bigger challenge for the algorithms. As expected, the RRT

Table 3.4: Distance and calculation time over five runs for different algorithms and varying search times in an obstacle avoidance path (station 5 to 2)

Algorithm	Search time [s]	Distance [m]		Calculation time [s]	
		Average	St. dev.	Average	St. dev.
RRT*	(60,60)	12.03	0.23	67.2	13.91
	(30,30)	11.95	0.22	31.0	0.00
	(10,10)	12.00	0.29	10.4	0.49
	(5,5)	**11.76**	0.13	5.40	0.49
PRM*	(60,60)	11.82	0.40	60.4	0.49
	(30,30)	11.94	0.46	30.8	0.40
	(10,10)	**11.76**	0.11	10.6	0.49
	(5,5)	11.85	0.16	5.80	0.40
Lazy PRM*	(60,60)	**11.75**	0.21	76.2	18.67
	(30,30)	11.96	0.54	30.6	0.49
	(10,10)	11.92	0.42	10.4	0.49
	(5,5)	12.20	0.59	5.40	0.49
RRT	(60,60)	14.65	2.51	0.40	0.49
	(30,30)	15.07	1.68	0.40	0.49
	(10,10)	15.05	2.06	0.20	0.40
	(5,5)	14.46	2.20	0.60	0.49
RRT Connect	(60,60)	15.04	3.12	0.80	0.40
	(30,30)	18.28	0.94	0.20	0.40
	(10,10)	19.69	5.55	0.20	0.40
	(5,5)	19.70	1.63	0.40	0.49

and especially RRT Connect are the worst performers, as the distances obtained are much longer than the minimum and are also much more inconsistent with their results. Despite the worse results, in terms of path length, for any maximum search time, it only uses one second at most for any search. This happens because these are not optimising algorithms and so, only try to find a solution, and when it finds one, it stops searching. It can be concluded that these two algorithms have little interest in following simulations, as the quality of the solution is very poor, about 25 to 70 percent worse than the calculated minimal distance compared to the 1 to 5 percent for the better performing algorithms.

It can also be seen that the search time doesn't have the expected impact on the path length for the best performing algorithms. It was anticipated that the higher the search time, the shorter the path, but as it can be seen on the table, this wasn't really the case. In fact, some algorithms

performed better with lower times than with higher allowed times, or just marginally better. This means that, in some cases, it is probably better to use a shorter search, as the result will be almost as good if not better, and in much less calculation time. For a simulation with multiple vehicles and tasks this is important, as at each calculation the full time is used, and so it can take quite a long time to finish the simulation. It must also be taken into account that the obstacle avoidance path was relatively simple and in more complex situations, the longer time may have a larger impact.

Between the first three algorithms, RRT*, PRM* and Lazy PRM*, the ones that were expected to perform better, the difference seems to be very small and maybe negligible, with no clear advantage to any of them. From the results of this test it seems that the choice between them is indifferent. If a distinction has to be made, the RRT* algorithm appears to perform better with shorter times, while PRM* is better for longer search times.

3.3.5 Pose Tests

To further explore the software and the model and the choices to be made, a similar test but with an alteration was made. Now, each station is made up of not only a position but an orientation as well, in an $(x; y; \theta)$ format. For this, the OMPL mode was changed to *pose mode*, as was the input file to include orientation. The orientation for each of the stations that are used in these paths is the same ($\theta = 0^{o}$).

While in the case of an omnidirectional and round vehicle, as used here, the *pose mode* does not have much relevancy, depending on the mobility of the vehicle and if there are any orientation requirements to the stations, it could be essential. While the routing problem does not usually concern orientation, just the paths to be taken, since the option is present in V-REP and the OMPL, it should be explored.

A test under the same conditions as the previous one was executed, but this time using only the better performing algorithms: RRT*, PRM* and Lazy PRM*. The same time step and search times are used once again. The results are in tables 3.5 and 3.6. The results for all algorithms using 60 s and 30 s are also presented in a boxplot format in figure 3.12.

Consulting the table it is visible that, once again, for a straight path the results are always equal and optimal. It should be noted that since both stations have the same orientation, no rotation had to be done by the vehicle.

For the obstacle avoidance path, both stations have the same orientation, but some rotation still occurs by the vehicle, in order to workaround the obstacle. The results for almost all combinations of algorithm and search time are close to the optimal solution and with a low deviation. It can also be checked that unlike the position only test, the length of the paths gets shorter with longer times, as it is expected to happen, but the difference is still small and possibly not worth the longer simulation times. Overall, using the Lazy PRM* algorithm, the best result was obtained. This happened at a 30 s search time and not while using a 60 s search time as expected. With Lazy PRM* the lowest single result was also achieved, at 11.56 m.

Table 3.5: Distance and calculation time over five runs for different algorithms and varying search times in a straight line (station 8 to 5) in *pose mode*

Algorithm	Search time [s]	Distance [m]		Calculation time [s]	
		Average	St. dev.	Average	St. dev.
RRT*	(60,60)	10	0	60.8	0.40
	(30,30)	10	0	30.4	0.48
	(10,10)	10	0	10.8	0.40
	(5,5)	10	0	5.80	0.40
PRM*	(60,60)	10	0	60.6	0.49
	(30,30)	10	0	30.6	0.49
	(10,10)	10	0	10.8	0.40
	(5,5)	10	0	5.40	0.49
Lazy PRM*	(60,60)	10	0	60.6	0.49
	(30,30)	10	0	30.6	0.49
	(10,10)	10	0	10.6	0.49
	(5,5)	10	0	5.80	0.40

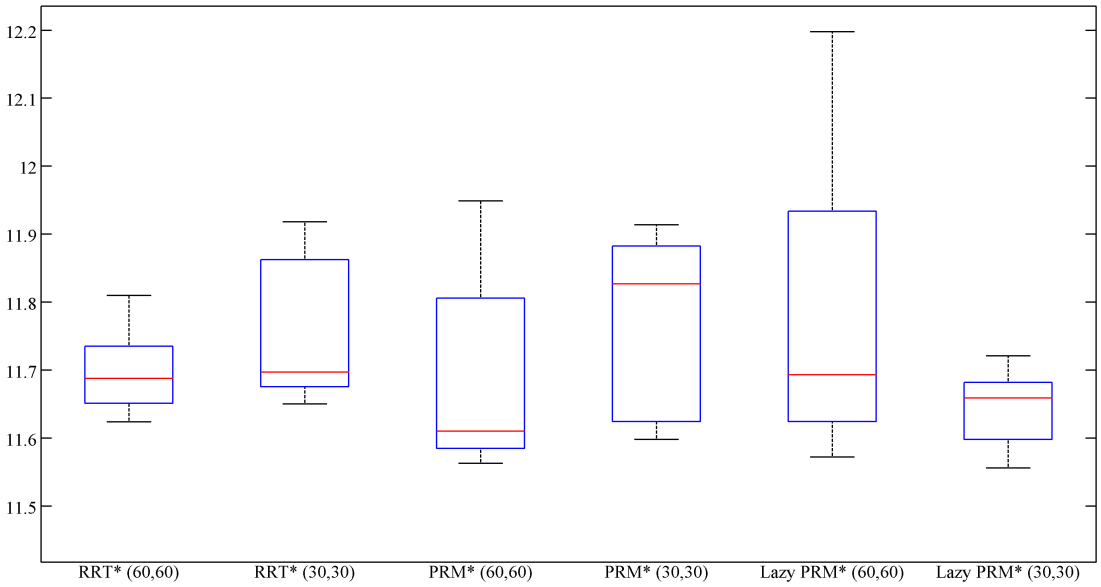


Figure 3.12: Graphical visualisation of the results for all algorithms in pose mode, using the two longest times

The boxplot allows an easier visualisation of the difference between each algorithm and the longer search times, which is no more than a few centimetres. In a path of about 12 m, the

Table 3.6: Distance and calculation time over five runs for different algorithms and varying search times in an obstacle avoidance path (station 5 to 2) in *pose mode*

Algorithm	Search time [s]	Distance [m]		Calculation time [s]	
		Average	St. dev.	Average	St. dev.
RRT*	(60,60)	**11.70**	0.06	60.8	0.40
	(30,30)	11.76	0.10	30.4	0.49
	(10,10)	11.84	0.19	10.4	0.49
	(5,5)	12.26	0.43	5.00	0.00
PRM*	(60,60)	**11.69**	0.14	60.8	0.40
	(30,30)	11.79	0.13	30.6	0.49
	(10,10)	11.79	0.08	10.8	0.40
	(5,5)	12.46	0.61	5.40	0.49
Lazy PRM*	(60,60)	11.79	0.22	60.6	0.49
	(30,30)	**11.64**	0.06	30.8	0.40
	(10,10)	11.80	0.13	10.4	0.49
	(5,5)	11.83	0.15	5.80	0.40

difference is quite small, of about making no algorithm more significantly better than the others.

When using *pose mode*, compared to the previous test, in no case the average time was higher than the search time by more than a second. This gives pose mode an advantage as to time, both by being lower and more consistent.

3.4 Conclusions

Following the preceding sections some conclusions can be drawn from the building and testing of the single vehicle system. The first is that, in fact, V-REP is an appropriate tool to build and model environments, along with behaviour scripting, that allow the testing of routing strategies. The ability to create a graphical setting using shapes, read input files, create output files, OMPL integration and programming freedom become important characteristics of the software. The scene also proved the proper working condition of the constructed scripts and files and their usability in future scenes.

The experiments revealed how the OMPL, and the impact of search time and chosen algorithm, work in different conditions and the best choice for some parameters. Different path finding algorithms were tested for two different configurations, position and pose, and two paths, straight line and obstacle avoidance. The results, judged by length of resulting path, show that the algorithms expected to perform better (RRT*, PRM* and Lazy PRM*), indeed were better. The allowed search time was shown to not have an impact as big as expected and the increased computational

time for each simulation does not lead to clearly better results. This can partly be explained due to the relative simplicity of the tasks given.

The scripts written for this scene, along with layout built, can be expanded with minor alterations to accommodate multiple vehicles and even different layouts due to V-REP's embedded scripts portability. With multiple vehicles in a single work space, the routing problem becomes much more complex and it will allow the implementation of more strategies in which the information obtained from these tests can be helpful in the creation of newer models.

Chapter 4

Modelling a Multi-vehicle Problem in V-REP

In this chapter it is described how a multi-vehicle system, and various routing strategies, were implemented in V-REP and how its performance was evaluated. In the first section, it is discussed how the previous single vehicle system can be used to approach the multi-vehicle system problem. New tests regarding path finding algorithms are carried out on a more complex layout. Next, the multi-vehicle routing strategies are established, implemented and tested. The information obtained from the tests concerns not only the simulation characteristics, such as time and distance, but also how V-REP handles the strategies, measuring the time to complete the simulation.

4.1 From Single Vehicle to Multi-Vehicle

In the previous chapter, all tests were performed considering the routing of a single vehicle in order to evaluate how V-REP executes the created models. The information obtained is required to validate the implementation of the scripts and the paths each vehicle takes.

The dynamics of a single vehicle system are quite simple when compared to a multi-vehicle one. Since no other vehicles are present, the risk of collisions, path blocking and other management complications are nearly non-existent and usually the only concern is finding the shortest path from origin to destination. When multiple vehicles are added to the system, the same concern still exists and efficiency is still the main goal, but now, it is a much more complex problem to solve. The solution to the routing problem in a multi-vehicle system must take into account all possible occurrences and usually multiple routing solutions are possible, which is why simulations are so important. The complexity of the problem means that there is not one solution fit to all situations and only through testing is it possible to find the most appropriate one for each case.

4.1.1 Layout

As previously mentioned in chapter 2, the layout of a multi-vehicle system can be constructed in many different ways. The free space for the vehicles to move, as well as the number of intersec-

tions, have a great impact on the ability of the vehicles to efficiently perform the tasks, since an increased number of vehicles means an increased chance of collisions.

For the multi-vehicle system, a layout similar to the one used for the single vehicle tests is used, regarding the dimensions of the workspace, which remain the same, a 20 m square grid, with the difference being in the placement of the machines and load/unload stations. Now, the stations are placed halfway between the location of the stations in the previous layout and so, each station is 5 m away from its adjacent ones. Since the tasks are from machine to machine, the resulting orders are not from any node to any other, but instead only from, and to, the ones that are adjacent to a machine (load/unload stations). This results in twenty one possible nodes, of which twelve are load/unload stations (figure 4.1).

Present in this configuration are nodes with four (node 11), three (nodes 3, 9, 13, 19) and two (all other nodes) possible directions as well as single width corridors. These features should create a challenge for the strategies and the results obtained from it could be more easily extrapolated and applied to different or larger layouts.

The initial parking area for the vehicles is now wider so that they can take an exit path independently of the vehicles in front being parked.

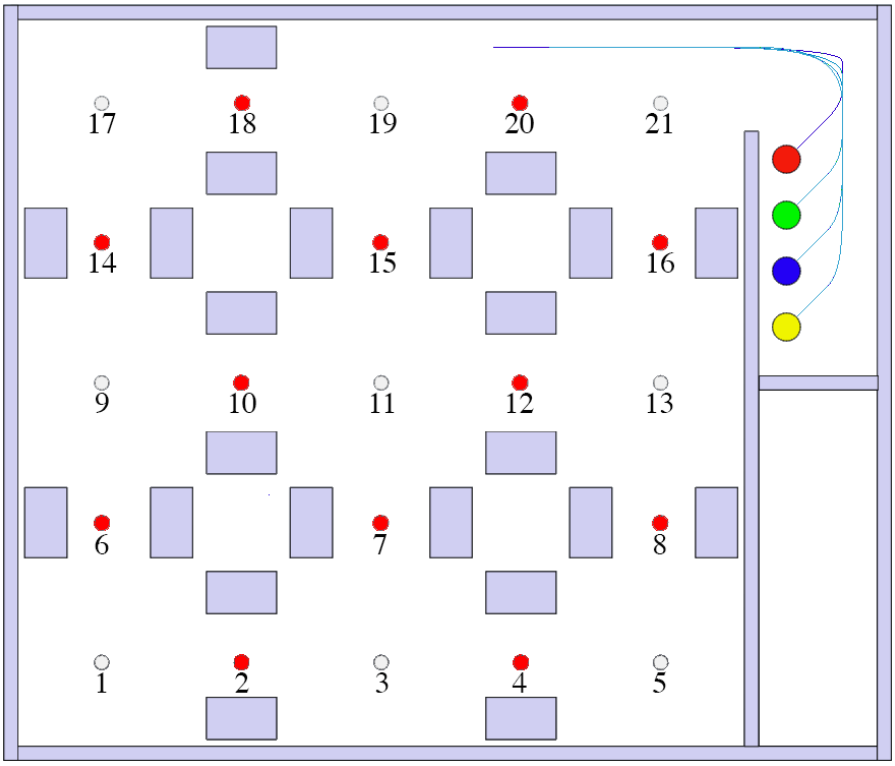


Figure 4.1: Multi-vehicle layout, made up of 21 nodes, including load/unload stations and intersections

4.1.2 Algorithms Performance in a Single Vehicle Case

In an industrial setting, a common approach to how the AGVs move is to have the vehicle movement confined to specific spaces, as to reduce the risk of interfering or colliding with a worker. To represent this, the layout was converted to cells and walls that form corridors where the vehicles can move freely.

Corridor Size Test

The greatly reduced free space, where a new path may be searched, is a major influence on the algorithm path finding capability. The width of the corridors where the vehicles may move must be found and for this purpose a test case was created.

This test case pretends to determine the distance between walls and cells that allows the path search algorithms to find a path connecting the start and goal position. The algorithms are evaluated not only on the ability to find a path, but also by the quality of the paths found.

The machine groups, or cells, are now represented by a square with rounded corners and its dimensions and distance to the walls define the width of the corridors, as seen in figure 4.2. Due to V-REP's limited shape modelling capabilities, the cells and walls were created in an external CAD software, *OnShape*, and imported into V-REP in *.stl* format.

In the first test, the three best performing algorithms in the previous chapter tests, Lazy PRM*, PRM* and RRT*, are given the task of moving from one corner, station number 21, to the opposite corner, station number 1. This test will reveal the minimum corridor width that should be used in future cases. Each algorithm was allowed three different maximum search times for each computation: 5, 10 and 30 seconds. If at the end of the search time no path was found then it would try again under the same conditions. Once a path connecting the stations is found, the test is over. Each combination of algorithm and search time was simulated five times. If after fifty attempts, in a single simulation, no path is found it is considered that it did not finish (DNF).

The scripts and input files used in this test are all similar to the ones presented in the past chapter, with only small changes to accommodate the larger number of stations. The OMPL mode used here is *pose mode*.

Two different widths were tested, 1.5 m and 2 m. Considering the outer bounds of the vehicle safety area are 1.4 m in diameter, the first width gives only a small tolerance, while the second allows easier travelling, but still only one vehicle at a time.

For a corridor width exactly the same as the vehicle safety size, that is, 1.4 m width, in any simulation performed, even when given much longer search times, a solution was never found in any case, as expected (due to no tolerance being given). To further test the algorithms capabilities, only single-width corridors (one vehicle at a time) are tested, but it can be assumed that if the corridor is even larger, the results would be better.

The results for 1.5 m and 2 m width can be seen in table 4.1a and table 4.1b, respectively.

From the results of these test cases a few conclusions can be made. The first is that the PRM* algorithm is clearly the worst performing algorithm of the three, for both widths and all search

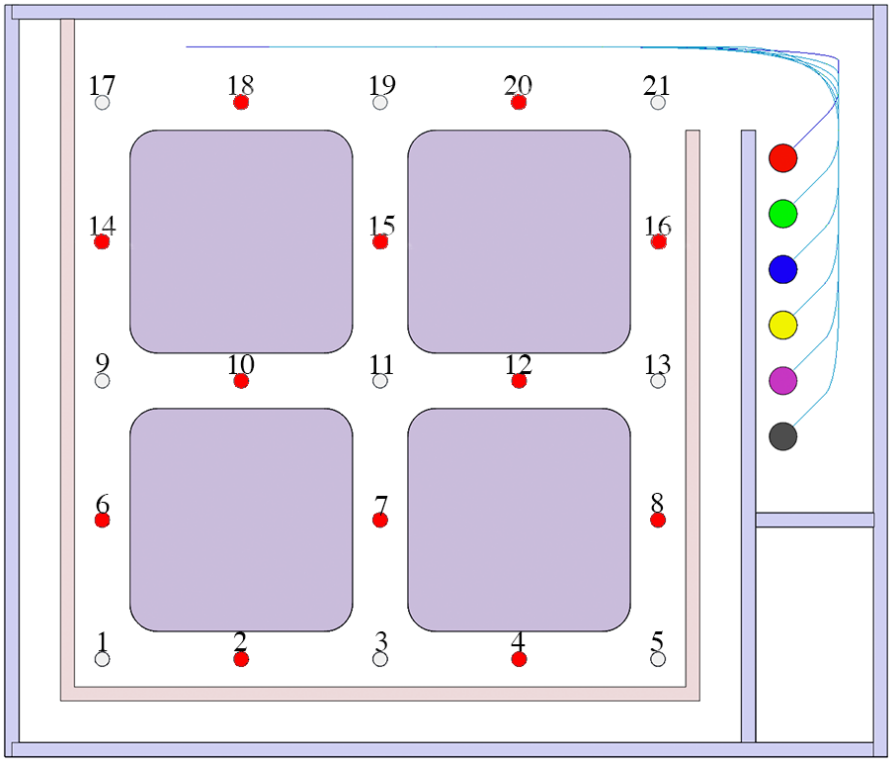


Figure 4.2: Representation of the multi-vehicle layout, with closed walls and cells, forming 2 m corridors

times. Comparing the results from the first table to the second, it shows that the narrowest width, about 7 % larger than the vehicle safety area, is in fact too small and leads to a relatively low success rate for any algorithm and search time and so, was not used. In the case of a 2 m width, about 43 % larger than the vehicle, the paths are found in a much lower number of tries, and except for PRM*, it is successful in the first attempt.

In the first situation, the RRT* algorithm is certainly the best performer, but even then the results are not satisfactory, since in some cases it took a large number of tries to find a path. This reinforces the conclusion that 2 m width corridors should be used over 1.5 m. When considering the wider configuration, the Lazy PRM* algorithm and RRT* algorithm offer similar results and further tests must be done to judge the algorithms not as to their success rate in finding a path, but in the quality of the paths found.

Path Length Test

While the previous test relates to the ability of the algorithms to successfully find a path in relation to the size of the corridor, it does not reflect the quality of the routes, where quality, in this case, pertains to the length of the path. A different test case was created for that purpose. In this new experiment, the best performing algorithms in the previous test, RRT* and Lazy PRM*, with a 10 seconds maximum search time, are given the task of finding a path connecting each of the twelve

Table 4.1: Number of necessary attempts to find a successful path between node 21 and node 1 with 1.5 m and 2 m wide corridors

(a) Results for 1.5 m wide corridors

Algorithm	Search time [s]	Attempts to find path				
		1st	2nd	3rd	4th	5th
RRT*	(5,5)	1	1	12	3	1
	(10,10)	2	2	1	1	1
	(30,30)	1	1	1	5	2
PRM*	(5,5)	DNF	DNF	DNF	38	DNF
	(10,10)	DNF	DNF	DNF	DNF	24
	(30,30)	15	14	16	DNF	DNF
Lazy PRM*	(5,5)	18	37	DNF	16	DNF
	(10,10)	2	26	10	17	7
	(30,30)	3	1	1	6	2

(b) Results for 2 m corridors

Algorithm	Search time [s]	Attempts to find path				
		1st	2nd	3rd	4th	5th
RRT*	(5,5)	1	2	1	1	1
	(10,10)	1	1	1	1	1
	(30,30)	1	1	1	1	1
PRM*	(5,5)	9	5	1	1	10
	(10,10)	3	3	2	1	8
	(30,30)	1	2	1	1	1
Lazy PRM*	(5,5)	1	2	1	1	1
	(10,10)	1	1	1	1	1
	(30,30)	1	1	1	1	1

machine serving stations to the others. In accordance to the findings from the preceding test, the passages are 2 m wide and no obstacles are present.

The length of each calculated path is compared to the minimum possible distance if the paths were made up of straight lines connecting the stations. For each path, the algorithms were allowed to repeat the search until they found a connecting path. In the event of a calculated path being longer than the minimum straight line distance the result is highlighted in the table. It should be noted that for each task more than one optimal route can be available, along with many other possible courses.

The results for the RRT* algorithm can be found in table 4.2 and for the Lazy PRM* algorithm in table 4.3. The format used is [Path Length/Minimum Length] in meters.

Table 4.2: Comparison between the length of calculated paths and minimum straight line distance, connecting the twelve load/unload stations, using RRT* with a 10 s search time

Origin \ Destination	2	4	6	7	8	10	12	14	15	16	18	20
2	0/0	10/10	8.9/10	8.8/10	18.8/20	18/20	17.7/20	18.8/20	18.8/20	28.9/30	27.8/30	27.8/30
4	9.9/10	0/0	18.9/20	8.7/10	8.7/10	17.8/20	17.7/20	28.9/30	19.8/20	18.9/20	27.9/30	28.1/30
6	8.8/10	18.8/20	0/0	18.3/20	27.9/30	9.2/10	19/20	9.9/10	17.9/20	28/30	19.1/20	28.9/30
7	9/10	8.9/10	18/20	0/0	17.8/20	9.7/10	8.9/10	18.7/20	9.9/10	17.8/20	18.7/20	18.9/20
8	18.8/20	8.9/10	28/30	17.8/20	0/0	18.9/20	8.9/10	27.8/30	17.8/20	9.9/10	27/30	19/20
10	17.8/20	17.8/20	9/10	8.7/10	18.9/20	0/0	9.9/10	8.8/10	9.1/10	18.9/20	19.3/20	18/20
12	17.9/20	18.5/20	19.8/20	9.2/10	9/10	9.9/10	0/0	18.8/20	9/10	9/10	18/20	17.6/20
14	19/20	26.7/30	9.9/10	17.9/20	28.1/30	8.8/10	19.7/20	0/0	18/20	27.7/30	8.9/10	18.9/20
15	18.9/20	18.9/20	17.9/20	9.9/10	17.7/20	9.1/10	9.1/10	18.3/20	0/0	18.2/20	8.8/10	8.9/10
16	26.8/30	18.9/20	27.8/30	18.1/20	9.9/10	18.9/20	8.9/10	27.9/30	17.9/20	0/0	18.9/20	8.8/10
18	27.8/30	27.9/30	19/20	19/20	27/30	17.9/20	18.5/20	8.7/10	8.7/10	18.9/20	0/0	9.9/10
20	27.7/30	27.8/30	27.4/30	19.2/20	18.9/20	18.1/20	17.9/20	19.8/20	8.9/10	8.8/10	9.9/10	0/0

Table 4.3: Comparison between the length of calculated paths and minimum straight line distance, connecting the twelve load/unload stations, using Lazy PRM* with a 10 s search time

Origin \ Destination	2	4	6	7	8	10	12	14	15	16	18	20
2	0/0	9.9/10	9.2/10	8.8/10	18.8/20	17.9/20	18/20	19.9/20	18.9/20	26.8/30	27.8/30	27.7/30
4	9.9/10	0/0	19.2/20	9/10	8.7/10	17.7/20	17.8/20	28.1/30	19.7/20	18.8/20	28.3/30	27.9/30
6	27/10	19.1/20	0/0	17.9/20	28.3/30	9.1/10	18.8/20	9.9/10	17.8/20	28.2/30	18.8/20	26.9/30
7	8.8/10	8.7/10	18.2/20	0/0	17.8/20	9.2/10	9/10	18.8/20	9.9/10	17.8/20	18.8/20	18.8/20
8	18.9/20	9.2/10	27.9/30	19/20	0/0	19.1/20	8.8/10	28.2/30	17.9/20	9.9/10	28.9/30	18.9/20
10	17.6/20	17.7/20	9.3/10	9/10	18.9/20	0/0	9.9/10	8.7/10	9.6/10	18.9/20	18.2/20	18.3/20
12	17.9/20	17.8/20	18.9/20	8.8/10	9.3/10	9.9/10	0/0	19.2/20	9.1/10	8.7/10	17.7/20	18.1/20
14	19.1/20	26.8/30	9.9/10	17.8/20	28.2/30	27.7/10	20.3/20	0/0	17.8/20	28/30	9.9/10	18.8/20
15	18.8/20	18.8/20	17.8/20	9.9/10	17.8/20	8.8/10	8.8/10	18/20	0/0	17.8/20	8.8/10	9.3/10
16	29/30	18.8/20	27.9/30	18.1/20	9.9/10	18.8/20	8.9/10	28.2/30	17.8/20	0/0	19/20	8.9/10
18	28/30	28.2/30	18.9/20	19.9/20	28.9/30	18.2/20	17.8/20	8.8/10	9.1/10	19.2/20	0/0	9.9/10
20	28/30	27.9/30	29.6/30	18.8/20	19.1/20	28.3/20	18/20	19/20	8.8/10	9/10	9.9/10	0/0

Considering the outcome of this experiment, it can be determined that the RRT* algorithm performed better than the Lazy PRM*. For the RRT* case, all routes found were optimal, while with Lazy PRM*, in three different paths a non optimal solution was found. In the situations where a non optimal path was taken, the same did not happen in the equal reverse path, which shows that this happens due to the inherent randomness of the algorithm, not a constant incapability to find the best route.

In all solutions where one of the possible optimal routes was calculated, a small decrease in length, of about 10% on average, was found, when compared to the minimum straight line distance. This is mostly due to the fact that the cells have rounded corners, which the algorithms

take advantage of. This can be seen in figure 4.3, which shows all calculated paths using the RRT* algorithm.

This test also served as an opportunity to implement a new feature in the vehicle script. When the vehicle reaches the end of the calculated path, it saves the *path object* as an external file using the *simSaveModel* command. At the end of the simulation a folder containing all calculated paths is obtained. The *paths* created may then be loaded and used in a different simulation and so would not be necessary to calculate them again, saving a large amount of time. The best paths from different algorithms or simulations may even be chosen and combined as to have a database of the preferred routes, which can be used at any time.

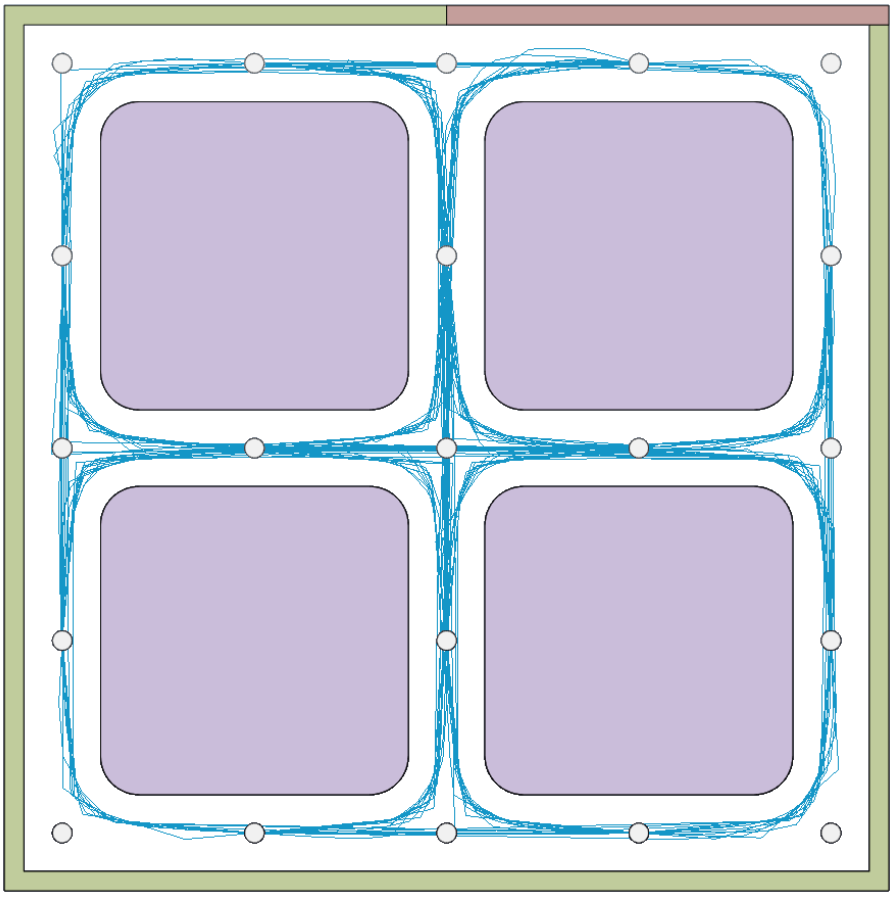


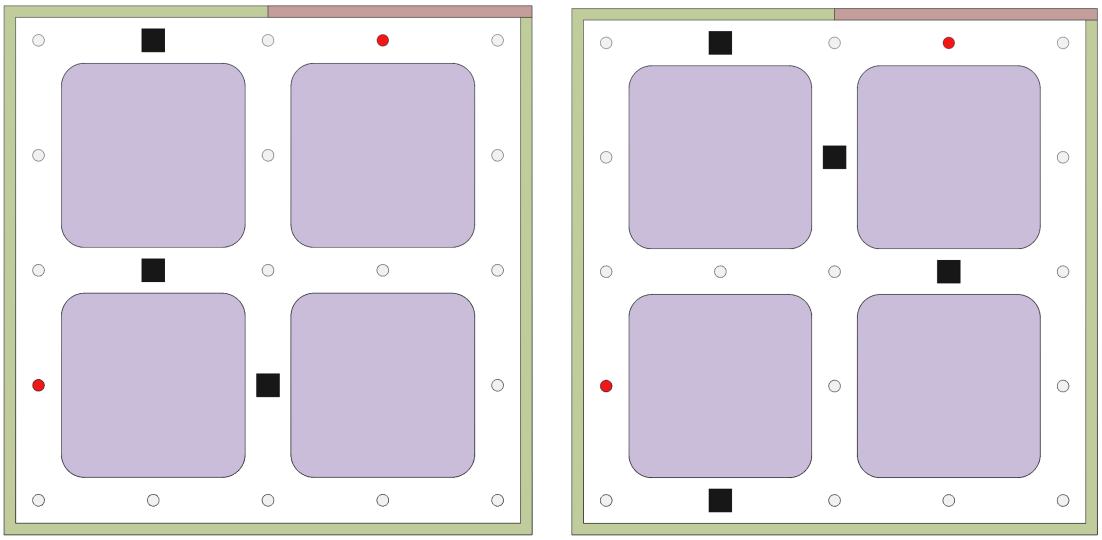
Figure 4.3: All calculated paths using RRT* in the path length comparison test

Blocked Path Test

In a multi-vehicle scenario, the passageways will not always be unobstructed. The obstacle may be static, pallets for example, or dynamic, such as other vehicles. Before moving to multi-vehicle scenes, it is first tested how the algorithms behave in situations where parts of the workspace are blocked. For this purpose, a vehicle is given the task of going from station 20 to station 6. This

path is chosen due to the fact that it presents two optimal alternative paths and other possible routes.

In each situation, a chosen corridor will be blocked, going from one to four obstacles. In the first case, the block will be placed on station 18, leaving an optimal path measuring 30 m (in straight line movements). The second time, the blocks are placed on stations 18 and 10, making the best possible path 40 m in length. In the third case, an additional obstacle is placed on station 7, leaving a single possible optimal path, 50 m long (figure 4.4a). For the last situation, the blocks are placed in stations 18, 15, 12 and 2 (figure 4.4b). This configuration also allows only one possible optimal route, in this case more complex and labyrinthine than the previous one, measuring 60 m, in straight line movements.



(a) Three obstacles configuration (b) Four obstacles configuration

Figure 4.4: Two of the obstacle placement configurations used in the obstacle avoidance test

Once again, the algorithms used are RRT* and Lazy PRM*, both with a 10 s search time, and both are given five attempts for each number of obstacles. In the event of the algorithm not being able to find a solution on the first try, that attempt is highlighted on the table. The results for both algorithms are presented in table 4.4

The results of the experiment reveal that in any case, the solution found was always optimal, along with the usual corner cutting seen before. Only in four of the simulations was more than one try necessary and all of them happened under Lazy PRM*, with one taking three tries (three obstacles case) and the others two. This is the only factor that separates the RRT* and Lazy PRM* performance in this test. This shows that both algorithms are apt to find optimal paths even in situations where one or more passageways are blocked, although Lazy PRM* could take more time.

Considering the outcomes of the last three tests in this chapter, and others made in the one before, it can be concluded that both the RRT* and Lazy PRM* algorithm are adequate in dealing

Table 4.4: Length of path found depending on the number of present obstacles, for both RRT* and Lazy PRM*

Algorithm	Number of obstacles	Path length [m]				
		1st	2nd	3rd	4th	5th
RRT*	1	27.2	26.9	27.2	27.2	27.3
	2	38.2	37.0	36.9	37.2	38.2
	3	47.4	47.1	47.9	47.8	47.2
	4	56.4	56.2	54.8	55.3	55.5
Lazy PRM*	1	27.2	26.9	27.2	26.8	27.1
	2	37.1	38.8	37.6	37.2	37.1
	3	47.7	46.8	46.9	47.2	47.4
	4	58.1	56.9	57.7	56.9	56.6

with obstacle avoidances and narrow corridors and finding optimal routes. Despite the difference not being very large, it can still be determined that the RRT* performed better in almost all cases and especially in the more recent tests. For this reason, RRT*, using a 10 s maximum search time, will be the preferred algorithm in future test cases.

The data obtained from these experiments, as well as the alterations made to the layout and scripts, allow a more informed transition to a multi-vehicle scenario.

4.2 Multi-Vehicle Routing Strategies

In this section, a few multi-vehicle routing strategies will be implemented and tested. The layout shown in the previous section will be the one used in all strategies. The scripts created for the preceding experiments will serve as the basis for any future strategies and any alterations made to them will be detailed.

The strategies created try to demonstrate some of V-REP's potential and a few possibilities of what can be done.

The task list consists of twenty five randomly generated orders and is common to all strategies tested, as to keep uniformity between tests. Since each task is made up of two parts, getting to the origin and then from the origin to the destination, this means that there are fifty paths to be completed. The total distance of all paths is 440 m, besides this distance, the vehicles must also move from the destination of one task to the origin of another (which is a variable distance dependent on the task received), in addition the path that takes them out of the parking area.

The strategies will be judged along two perspectives: 1) evaluating the performance of the strategies as to productivity and 2) evaluating how V-REP executes them. The data presented for each case (table 4.5 and all following tables) is detailed below. For 1), the parameters used are: the simulation time elapsed (*Time [s]*), the distance travelled (*Distance [m]*), the number of collisions

occurred (*Collisions*) and the number of tasks completed (*Tasks completed*). For 2), it includes the real time taken by V-REP to calculate and create the *paths* (*Calculation time [s]*), the real time to perform the paths, namely movement, collision sensing, etc. (*Movement time [s]*) and the sum of both these times (*Total real time [s]*). In addition, a relation between 1) and 2) is made, as the ratio between total real time and simulation time (*Time ratio*). For each table, the overall data (i.e. highest times, sum of all distances, total collisions and tasks completed) is presented in the *Total* row.

The simulations are executed with a 1 ms time step and at the beginning of each simulation, the passes per frame are increased to 12800 (initiated at 200). This leads to the fastest possible simulations, while keeping accuracy. Each strategy is implemented in a separate scene. Due to V-REP's scene objects and scripts portability, the common scripts and shapes can be easily transferred from one scene to the other, with no preparation needed.

The strategies tested can be divided in two groups: the ones using path finding algorithms (OMPL), and one using an exhaustive node search algorithm (graph search). An overview of these strategies can be seen in figure 4.5.

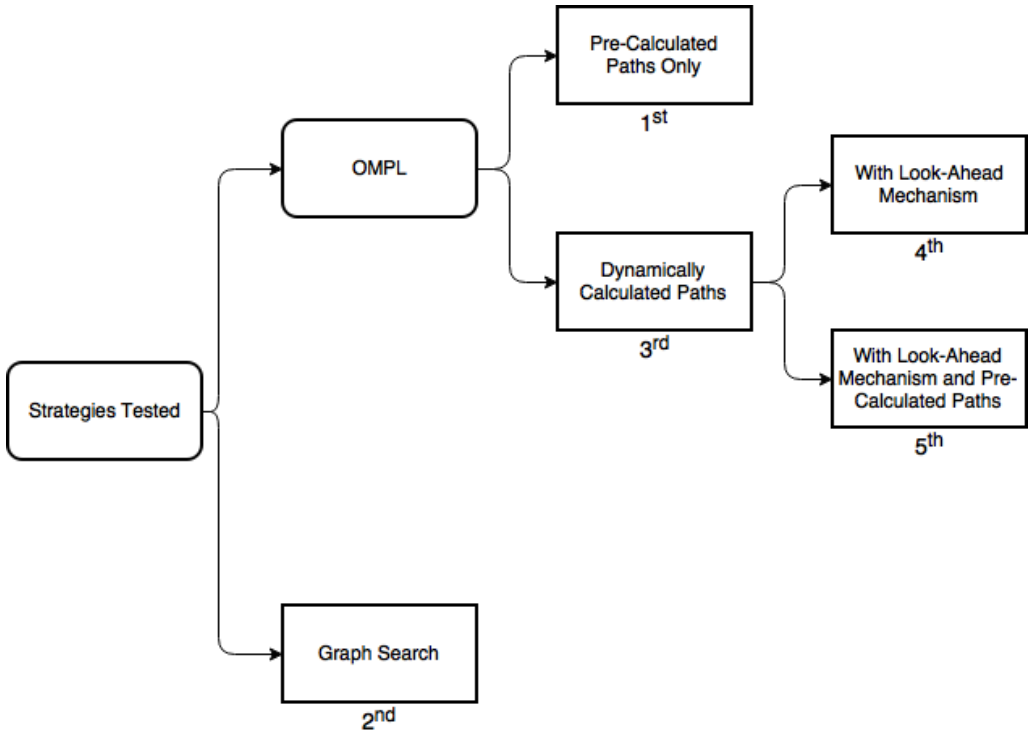


Figure 4.5: Overview of the strategies tested

4.2.1 First Strategy

The first strategy tested aims to be as simple as possible, and will serve as a sort of baseline for future experiments. In this approach the vehicles will use the *paths* generated in the previous

section test (table 4.2), which used RRT* with 10 s search time, and found an optimal path for each route between nodes. The *vehicle script* starts by loading the *paths* stored in an external folder and assigning them a descriptive handle from which they can be called during the simulation. When the vehicle receives a new order from the *manager script*, it will use the stored *path* to try to complete the order. To leave from and return to the parking area (above the layout nodes), the vehicle creates a new path using the OMPL.

During the path, collisions may happen, and in case a vehicle detects a collision, it will move back 2 m on the same path, after which it will wait for some time ,dependent on the vehicle, before moving again on the same path. The time each vehicle waits (wt) is constant and related to its number (n), where $wt = (n-1)*3$. This method was chosen due to ease of programming. By giving each vehicle a different wait time, a priority order between them is established, which, in some cases, helps reduce traffic. Note that when it detects a collision it is not the actual vehicle that is colliding, but instead a surrounding safe area.

This method of dealing with collisions can easily cause a lot of problems. For example if two vehicles are on the same corridor and the path leads them across each other, then they would just keep colliding and retreating over and over again. To deal with this situation, if the task is taking longer than 120 s to be completed, it is considered a deadlock has happened and the vehicle will not be able to complete the current task. After the time exceeds 120 s the vehicle will forget the task and move back to the station where the current path originated and receive a new one. When the vehicle reaches that station it will also communicate with the *manager script* to send the dropped order to the end of the task list.

Since this method has a large tendency to create situations where the vehicles are blocked, a criterion for deciding if the vehicle is blocked is implemented. When a path is taking more than 300 s to complete, then it is considered that the situation is unsolvable. The vehicle that exceeded the allowed time sends a signal to all other vehicles and the manager to stop the simulation and register the data obtained up until that point. These time values were chosen considering the speed of the vehicles and the relatively small size of the layout. Considering the longest possible task, using an optimal path, is 30 m and therefore 30 s to complete, the time limit would allow that task to be completed ten times.

While it is expected that for one vehicle this strategy can be very efficient, when additional vehicles are added, and so the potential for collisions increases, the time and distance necessary to complete all orders will increase greatly or the tasks may not be completed at all.

In the event that not all tasks are completed then the number of tasks is highlighted in the table. The results for all vehicles can be found in table 4.5.

Table 4.5: Results for the first strategy: one to six vehicles

(a) One vehicle

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	931	930	0	25	129	2258	2387	2.56
Total	931	930	0	25	-	-	2387	2.56

(b) Two vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	700	552	8	12	76	1814	1890	2.70
2	640	501	6	13	22	1620	1642	2.57
Total	700	1053	14	25	-	-	1890	2.70

(c) Three vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	591	495	5	10	113	1462	1575	2.66
2	543	366	18	9	21	1314	1335	2.46
3	547	339	6	6	84	1340	1424	2.60
Total	591	1120	29	25	-	-	1575	2.66

(d) Four vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	384	132	43	2	126	1088	1214	3.16
2	402	294	7	5	13	1166	1179	2.93
3	385	137	31	2	11	1121	1132	2.94
4	418	269	10	5	12	1256	1268	3.04
Total	418	832	91	14	-	-	1268	3.04

(e) Five vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	338	156	50	1	141	945	1086	3.21
2	348	204	12	4	12	996	1008	2.89
3	358	141	19	2	11	1046	1057	2.95
4	342	76	23	1	11	1004	1015	2.96
5	338	65	18	1	83	1003	1086	3.21
Total	358	642	122	9	-	-	1086	3.03

(f) Six vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	329	108	12	0	518	2080	2598	7.91
2	337	125	12	2	49	2197	2246	6.67
3	328	38	34	0	28	2209	2237	6.81
4	348	127	10	2	85	2375	2460	7.06
5	333	118	11	2	265	2358	2623	7.88
6	329	60	37	0	69	2417	2486	7.56
Total	348	576	116	6	-	-	2623	7.54

Based on the outcome of the tests, it can be concluded that, as expected, the calculation time is usually small, since the only significant calculations it has to perform is when leaving and returning to the parking spot, using the OMPL. Since the path loading is done previously, the time spent on other paths is mostly just receiving information from the manager script and selecting the correct path, and so, the time it amounts to is negligible.

The time ratio is roughly consistent between tests from one to five vehicles, seeing only a sharp increase when six vehicles are used. The time ratio in a single simulation is consistent between vehicles, with only a small variation occurring. It is also most affected by the vehicles that had trouble leaving or entering the parking area, the only time the OMPL is used, since it only contributes to the real time, not the simulation time.

The time necessary to complete all tasks decreased with the addition of vehicles until a saturation point was reached with four vehicles. When four or more vehicles were present simultaneously in the layout, this strategy was not able to finish all orders.

As expected, for a single vehicle, the distance travelled is low and near optimal. As more vehicles are added, the distance travelled by each vehicle decreases, while the total increases, until the saturation point is reached. A similar behaviour is also found in the time taken by the vehicles to complete the simulations. The number of collisions also increased with the number of vehicles, which in turn lead to the mentioned growth in time and distance, since for each collision the vehicles move back and wait.

It is obvious that this strategy's collision avoidance is not appropriate and easily saturates, as to number of vehicles, the current layout. This is expected, since barely any measures are taken to avoid the collisions and deadlocks.

4.2.2 Second Strategy

The second strategy implemented differs in how the routes are searched. In this strategy, instead of using the OMPL, a graph search approach is taken to find a path between an origin and a destination node, considering that no collisions will occur. The paths connecting adjacent nodes are considered straight line paths.

In the *vehicle script*, each station is introduced in the code as a node, as well as each adjacency between nodes, along with the distance connecting them. When the vehicle receives a new order, it will use Dijkstra's algorithm, as presented in subsection 2.2, to find the shortest path from the current node to the destination and the result will be a sequence of stations that form the path. The movement of the vehicle is then a straight line connecting two adjacent, and sequential, stations.

During a task, when the vehicle reaches a new node, it will run the algorithm towards the destination. If two or more equally optimal nodes are available, one of them will be chosen randomly, as to avoid a tendency for the vehicles to always use the same routes and therefore avoid deadlocks.

Before moving to a new node, the vehicle will check if that connecting path is occupied. This is done by sending a signal when moving to a new station and before moving check the signals to see if any vehicle will block the movement. In this strategy, if it is detected that a path is

occupied, a random, adjacent and unoccupied, node will be chosen as the next move. This search, and subsequent *signal* check, is performed at each new node, as to constantly avoid collisions, while keeping optimality in path distance.

The rules for choosing a new path when the optimal one is blocked can be many, such as next least distance path, or a clockwise movement policy, which induces an overall flow direction of traffic. Only one (random) is tested, as an example of a graph search approach, but other rules could be easily applied. The flowchart for this new behaviour can be visualised in figure 4.6.

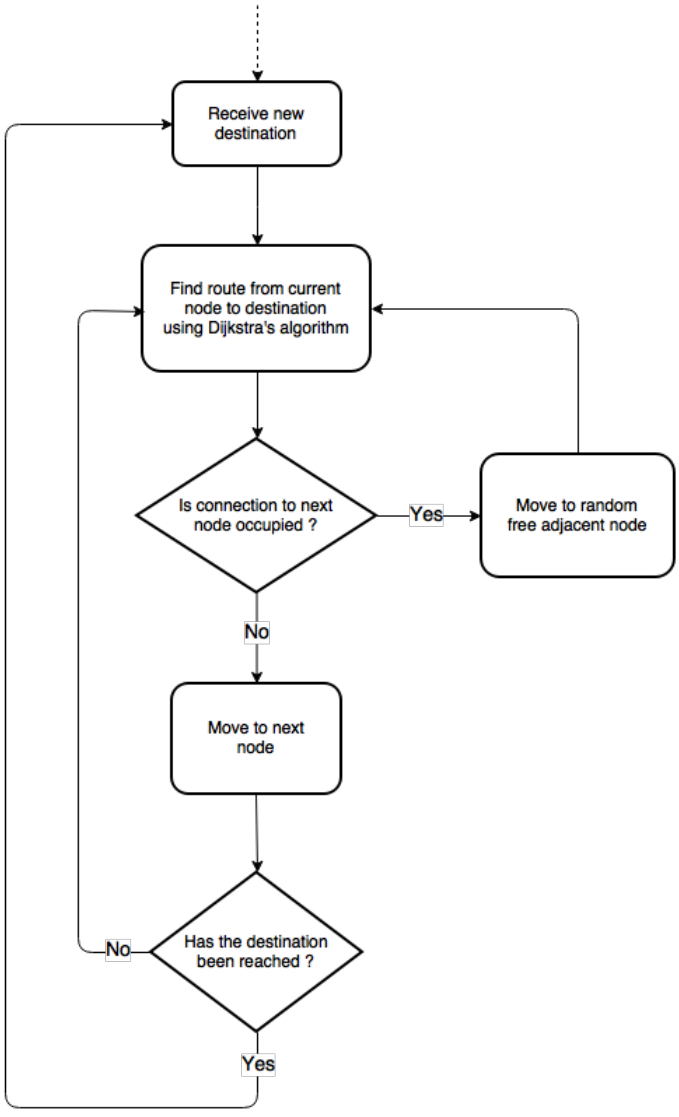


Figure 4.6: Flowchart describing the second strategy vehicle behaviour

With this strategy, collisions may still happen, mostly when two vehicles reach the same station simultaneously, coming from different directions. When this happens, the vehicles will move back a small distance and once again wait for a certain time, dependent on the vehicle.

The rules to decide when the path is unachievable or the vehicle are locked are the same as the previous strategy.

It is anticipated that the calculation time for this strategy, for each individual path, be slightly higher than the previous one, while the total calculation time for a vehicle during the whole simulation be lower, since the OMPL is never used. The number of collisions should also be greatly reduced when compared to the previous strategy.

In the case that not all tasks are finished then the number of tasks is highlighted in the table. The results for all vehicles can be found in table 4.6.

The information gathered from the simulations executed allow some conclusions to be taken. The first is that the calculation time in all cases is always low, which in turn, reduces the time ratio. The time ratio is consistently much lower than the previous strategy, even when using six vehicles, with only a small increase with the addition of more vehicles.

When using multiple vehicles, the time it takes to complete all tasks decreases while the distance travelled increases, until a saturation point at six vehicles is reached. At six vehicles, only a few tasks were completed, with some vehicles not being able to complete a single task. This shows that, for this strategy, five is the maximum number of simultaneous vehicles, while four vehicles took the least time to complete all tasks.

The distance travelled when using a single vehicle is slightly higher than in the previous strategy, which is expected, due to the fact that all paths are now a straight line. The overall distance, unlike the previous strategy, saw only a small increase with the addition of more vehicles. The time taken also decreased with more vehicles, as expected. These two factors show that this strategy is quite efficient as to both time and distance, as well as to V-REP execution.

4.2.3 Third Strategy

The third strategy created relies more heavily in the OMPL. There are no pre-defined routes and all *paths* are created when necessary using the OMPL. Besides the path finding and creation process, the strategy is similar to the first one.

When a new order is received by the vehicle, the OMPL is used to create a new path, with a 10 s search time and the RRT* algorithm. If it can not find a connecting path, it will wait for three seconds before trying again.

When the time to complete a task exceeds 120 s, the vehicle returns the task to the end of the order list and returns to the station where it was initially to receive a new task. In the event of a vehicle colliding five times along a single path, the OMPL is used again to find a new path. When finding this new path, if the number of attempts exceeds fifteen, then the vehicle is told to return to the station in the same way as if 120 s had passed. Between each of the attempts, the vehicle waits in place for three seconds, in order to give time for other vehicles to move. Both these numbers, five collisions and fifteen attempts, are partially arbitrary and adjusted through iterative use. If after being told to return it can not find a path in fifteen attempts, then it is considered that the vehicle is locked, same as if 300 s had passed.

Table 4.6: Results for the second strategy: one to six vehicles

(a) One vehicle

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	990	994	0	25	23	1525	1548	1.56
Total	990	994	0	25	-	-	1548	1.56

(b) Two vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	543	544	0	12	11	829	840	1.55
2	495	494	0	13	14	701	715	1.44
Total	543	1038	0	25	-	-	840	1.55

(c) Three vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	406	395	10	9	10	678	688	1.69
2	336	325	2	8	8	498	506	1.50
3	354	335	3	8	8	548	556	1.57
Total	406	1055	15	25	-	-	688	1.69

(d) Four vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	282	291	20	6	7	455	462	1.63
2	316	233	34	6	13	478	491	1.56
3	272	235	1	5	3	431	434	1.59
4	334	290	2	6	6	598	604	1.81
Total	334	1049	57	25	-	-	604	1.81

(e) Five vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	297	299	3	7	6	538	544	1.83
2	289	264	5	5	9	514	523	1.81
3	337	331	2	6	10	666	676	2.00
4	338	318	2	5	6	674	680	2.01
5	320	211	6	2	6	618	624	1.95
Total	338	1423	18	25	-	-	680	2.01

(f) Six vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	354	244	27	3	21	601	622	1.76
2	353	57	58	2	4	620	624	1.78
3	361	138	7	2	2	195	197	1.78
4	352	31	33	0	2	625	627	1.77
5	355	88	7	2	7	626	633	1.78
6	353	23	20	0	5	625	630	1.78
Total	361	581	152	9	-	-	633	1.73

While it is expected that this strategy has a considerably increase in the calculation time when compared to the preceding strategies, it is also expected a low number of collisions and distance travelled to complete the tasks, when compared to the first strategy. As was shown in previous experiments, the paths found by the OMPL were always optimal, which is once again expected to happen, although it is not guaranteed to always occur. A possible problem with this method, is that when the OMPL calculates a new path, it takes into account the position of the other vehicles at the time. Another vehicle might be blocking a possible optimal route when the path is calculated, forcing the vehicle to take a non-optimal path, even if the distance is enough that no collision would have occurred if the optimal path was taken.

As previously mentioned, while the OMPL is calculating a new path, the simulation clock is stopped, which means that while finding a new path, only the real time keeps counting. Due to this behaviour, the time ratio should be higher than in previous strategies.

As before, if not all tasks given are completed, the result is highlighted in the table. All results are gathered in table 4.7.

From the results displayed in the table, it can be concluded that, as expected, the calculation time is much higher than the previous strategies, which leads to a consistently high time ratio for all vehicles.

The number of collisions is reduced when compared to the first strategy, which was expected, while it is higher than the second strategy. The distance travelled when using a single vehicle is low, as expected, but slightly higher than the first strategy, where all paths are optimal. When further vehicles are added, once again, the distance and time for each vehicle decreased while the total grew. While the results are generally better than the first strategy, they are still inferior to the second strategy.

When using four vehicles, the number of completed tasks was lower than when using five vehicles. While this is contrary to what could normally be expected, that a higher number of vehicles would block the layout more easily, this can be explained by the fact that using OMPL can produce different results every simulation and the additional vehicle also alters the capability to complete tasks of the other vehicles. Even though four more tasks were finished, in the end, both cases were unable to complete all of the given orders. Since the orders are random, by introducing a new vehicle, and therefore altering the assignment of orders to vehicles, the results can change.

It should be noted that, while using the same strategy, completing more or less tasks but not finishing all of them, does not have much significance as to the quality of the strategy. It means that the events that lead to blocking the system appeared later or sooner, but still happened. Only when all tasks are completed, can it be said that the strategy has a proper handling of vehicle collisions and routing problems.

This strategy can only accommodate a smaller number of vehicles compared to the previous one, and the same as the first, as it was unable to finish even with four vehicles at the same time.

Table 4.7: Results for the third strategy: one to six vehicles

(a) One vehicle

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	932	928	0	25	552	1069	1621	1.74
Total	932	928	0	25	-	-	1621	1.74

(b) Two vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	615	483	5	12	332	1560	1892	3.08
2	633	579	2	13	419	1473	1892	2.99
Total	633	1062	7	25	-	-	1892	2.99

(c) Three vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	484	454	1	11	310	1178	1488	3.07
2	456	351	5	8	468	881	1349	2.96
3	455	331	4	6	173	1184	1357	2.98
Total	484	1136	10	25	-	-	1488	3.07

(d) Four vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	562	424	4	9	334	1450	1784	3.17
2	418	194	8	4	476	881	1357	3.25
3	562	174	10	3	374	1356	1730	3.08
4	562	150	12	2	118	1624	1742	3.10
Total	562	942	34	18	-	-	1742	3.10

(e) Five vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	683	425	5	7	661	1715	2376	3.48
2	813	303	18	3	1495	1123	2618	3.22
3	734	375	13	7	693	1778	2471	3.33
4	682	227	11	1	293	2041	2334	3.42
5	772	216	15	4	1170	1420	2590	3.35
Total	813	1546	62	22	-	-	2618	3.22

(f) Six vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	355	89	6	0	134	1027	1161	3.27
2	378	149	4	2	108	1070	1178	3.11
3	355	138	6	3	108	989	1097	3.09
4	364	253	5	5	250	656	906	2.49
5	592	170	14	2	1630	1250	2880	4.86
6	375	192	8	5	557	663	1220	3.25
Total	592	991	43	17	-	-	2880	4.86

4.2.4 Fourth Strategy

One of the problems that causes the most increases in both time and distance travelled, is the occurrence of collisions, that is, situations where the path chosen by the vehicle, which is initially clear, ends up being blocked by some other vehicle. This is the biggest adversity of multi-vehicle routing, where vehicles move freely and at their own time.

The difficulty comes from having to be able to consider not only space, as the OMPL does, but also time. Predicting where future collisions in a path might happen, during the simulation, is not an easy task. To try to minimize this problem, a possible collision avoidance mechanism is presented.

The fourth strategy to be tested is similar to the one before, but with the addition of a new feature. When a new, successful, path is calculated, a vehicle (from here on referred to as *ghost vehicle*) completes the path at a much faster speed, along the same way the vehicle would do. The speed at which it travels can affect its performance. This *ghost vehicle* is seen only by other *ghosts* and does not interfere with the normal vehicles. If during the path, the *ghost vehicle* collides with any other *ghost*, which means that if the vehicle performed the same path it could encounter the same collision, the vehicle waits for a few seconds and sends the *ghost* again. If when sending the *ghost* a second time it encounters another collision, the OMPL is used to find a new path and repeat the process.

This mechanism is expected to work when two, or more, vehicles leave their stations with only a small time difference. One example of a situation where this strategy does not work is when a vehicle is travelling a longer path while the other completes two shorter ones. In this case, since the *ghost* is sent before the movement begins, it would not be able to sense a possible collision with the second vehicle performing a path that started much later.

This new addition to the vehicle script is expected to reduce the amount of collisions and therefore the time and distance necessary to complete the orders. While the in-simulation parameters should improve, the calculation and move time should increase. It is also expected to have a bigger impact on simulations with a larger number of vehicles, since that is where more collisions occur.

The results obtained from this strategy can be seen in table 4.8.

The results for the fourth strategy, which is actually the same as the third with the addition of *ghosts*, allow some conclusions to be drawn.

In comparison to the previous strategy, it was now able to complete all tasks when using both four and five vehicles, although for six it was still not possible.

The presence of the *ghost* for each vehicle increased the time ratio, due to the extra step of checking and avoiding future collisions, which largely increased the calculation times. The abnormally high time ratio for the four vehicles case, while it probably would not always happen it is still a possibility and shows one of V-REP's downsides, its unexpected decreases in performance.

A worse performance, as to time taken, and a similar, one as to distance travelled, can be seen when compared to the third strategy, for up to three vehicles. For all cases, the parameters saw an

Table 4.8: Results for the fourth strategy: one to six vehicles

(a) One vehicle

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	957	927	0	25	596	1020	1616	1.69
Total	957	927	0	25	-	-	1616	1.69

(b) Two vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	717	555	5	14	397	1005	1402	1.96
2	741	586	5	11	295	1136	1431	1.67
Total	741	1141	10	25	-	-	1431	1.67

(c) Three vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	689	455	9	10	638	1650	2288	3.28
2	695	396	9	9	927	1336	2263	3.26
3	723	325	15	6	448	1897	2345	3.24
Total	723	1176	33	25	-	-	2345	3.24

(d) Four vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	522	364	14	8	2271	2743	5014	9.6
2	546	311	8	7	481	4718	5199	9.52
3	669	259	13	5	1703	4306	6009	8.98
4	530	267	10	5	3916	1096	5012	9.46
Total	669	1201	45	25	-	-	6009	8.98

(e) Five vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	545	231	9	5	1669	1223	2892	5.3
2	632	246	14	3	1330	1440	2770	4.38
3	573	306	8	7	530	2254	2784	4.86
4	522	299	8	6	521	2238	2759	5.28
5	612	207	12	4	510	2553	3063	5.01
Total	632	1289	51	25	-	-	3063	4.85

(f) Six vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	398	175	7	2	742	762	1504	3.78
2	383	105	7	1	57	1223	1280	3.34
3	407	151	9	2	839	530	1369	3.36
4	369	127	6	1	602	684	1286	3.48
5	402	69	12	0	932	493	1425	3.54
6	415	218	5	3	135	1293	1428	3.44
Total	415	845	46	9	-	-	1504	3.62

evolution similar to the one found in the previous strategies, a decrease of time and distance for each vehicle, along with an increase in total distance and collisions.

The addition of the *ghost* can be seen as valuable, as it allowed all tasks to be completed with four and five vehicles, although with a decrease in V-REP performance and some simulation parameters.

4.2.5 Fifth Strategy

From the results of the first strategy, it could be observed that the calculation time was always low but the results were poor. And the opposite happened with fourth strategy, the results improved, while the calculation time grew. If both are combined, then the best of both strategies may be achieved in one.

This fifth strategy uses the same system as the fourth strategy (including the *ghost* mechanism), but now, the first path attempted is one of the already pre-existing paths from the first strategy. This means that for most tasks, where no collisions occur, the OMPL will not be needed and so the calculation time will be much lower. The maximum number of collisions before trying to find a new path is reduced from five to three.

The overall time and distance for this strategy is expected to be similar to the fourth strategy, since the newly created path is usually equal to the previously calculated one. The difference is that now, that path may be blocked at that time and the vehicle will still attempt to perform it. If the OMPL was used to create the path, it would avoid the other vehicles, considering their position at that time.

It is expected an improvement as to calculation time, while a slight decline as to simulation parameters (time and distance) may occur. The results can be found in table 4.9.

Consulting the table, it can be concluded that the expected improvement on the time it takes to complete the simulations did not occur. While the calculation time stayed about the same as the previous strategy, the movement time increased.

Using the pre-created *paths*, would have saved about 10 s (the allowed search time for RRT*) of calculation time for each path. When comparing this to the visible increase in movement times, it seems that loading the paths compared to creating them when necessary, is not advantageous.

A possible explanation for this comes from the way the *paths* are loaded into the script. The command to load an external model, *simLoadModel*, only allows the specification of an handle. In the scripts, the handles are how scene objects, collection, etc., are used in commands. In the beginning of the script, an handle has to be attributed to each object, using *simGetObjectHandle*, to be able to use that object.

Because an handle is confined to the script where it was attributed, that is, the same handle can not be used across scripts, each one must individually load all the *paths* they want to use. Since the layout contains 12 work stations, and a *path* connecting each one, this means there are 144 saved paths. If six vehicles are used in a simulation, and each *vehicle script* loads all *paths*, this means that at the start of the script alone, 864 *paths* are introduced in the scene. When also considering

Table 4.9: Results for the fifth strategy: one to six vehicles

(a) One vehicle

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	952	929	0	25	101	1199	1300	1.37
Total	952	929	0	25	-	-	1300	1.37

(b) Two vehicles

	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	779	531	15	13	1006	2161	3167	4.07
2	802	528	12	12	597	2580	3177	3.96
Total	802	1059	27	25	-	-	3177	3.96

(c) Three vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	719	398	14	8	1617	1581	3198	4.44
2	836	381	13	9	1767	1905	3672	4.39
3	721	375	10	8	644	2465	3109	4.31
Total	836	1154	37	25	-	-	3672	4.39

(d) Four vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	576	426	12	11	607	2115	2772	4.73
2	580	329	12	6	1498	1121	2619	4.51
3	598	211	16	5	1311	1418	2729	4.57
4	475	155	11	3	1407	652	2059	4.34
Total	598	1121	51	25	-	-	2729	4.57

(e) Five vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	804	332	6	18	2438	3534	5892	7.33
2	716	278	18	5	4305	905	5210	7.27
3	816	355	17	7	3013	2815	5828	7.14
4	758	215	15	4	3111	2346	5457	7.20
5	822	237	16	3	4728	1255	5983	7.28
Total	822	1417	84	25	-	-	5983	7.28

(f) Six vehicles

Vehicle	Time [s]	Distance [m]	Collisions	Tasks completed	Calculation time [s]	Movement time [s]	Total real time [s]	Time ratio
1	224	63	6	0	334	1036	1370	6.10
2	217	99	5	3	377	808	1185	5.46
3	216	48	5	0	652	541	1193	5.53
4	227	94	5	3	427	849	1276	5.61
5	224	68	4	2	1072	299	1371	6.11
6	227	141	6	1	1053	254	1307	5.75
Total	227	513	31	9	-	-	1371	6.04

the exit *paths* and all other created while completing the tasks, the total number of existing *paths* in a scene can be close to one thousand.

The large number of scene objects present is a possible reason as to why the movement time, and time ratio, for this strategy are worse than in previous ones.

A possible solution to this problem would be to only load each *path* as necessary and deleting them when no longer needed. This would make the scene lighter, although the routes taken by the vehicles during the entire simulation would no longer be visible, which can be interesting in further analysing what areas of the layout are more frequently used.

As to simulation parameters, time, distance and collisions, not much of a difference was observed when compared to the previous strategy. This is expected, as the alterations made for this strategy mainly affect calculation and movement times.

4.2.6 Strategies Comparison

After obtaining the results from all the strategies tested, a few conclusions can be made. First, some issues concerning the results. When the total number of collisions is not even, or the number for each vehicle does not add up, meaning that one vehicle had more collisions than others, it happens due to the fact that when a vehicle collides against another, due to the incremental movement and therefore limited resolution, one vehicle may detect the collision while the other may not. This could be improved by decreasing even further the percentage moved each time, but that would lead to a much longer calculation time. Another possible reason for this might also be poor collision detection by V-REP's collision module or flaws in the code.

Also regarding the results, when not all tasks are completed and a *signal* to stop the simulation is sent, it would be expected that both the simulation and real time be the same for all vehicles. This does not always happen, since there are only a few lines in the code where this signal can be read and the information registered. The time difference comes from the time it takes the script to reach one of these positions.

In some occasions, e.g. the first strategy with one vehicle, the time ratio obtained from the tests is unexpected or inconsistent. A possible reason for this is unrelated computer activity. Even though it was always attempted to be reduced to a minimum, it could still affect V-REP's performance unexpectedly, since V-REP tries to use as much of the computer's memory as it can.

Considering the layout, some new information can be learned from the experiments. The first is that, for the strategies used, the layout can not take more than five vehicles without deadlocks occurring. No strategy used was able to complete all tasks using six vehicles, although it could be possible with other strategies or parameters.

While observing the movements made by the vehicles in each strategy, it seems that the size of the parking area can largely influence the vehicles ability to enter and leave the workspace. It was often a part of the tasks that lead to collisions. The layout used had a 3 m space between each parking spot and 2 m above the upper row of stations and it was sometimes not enough, so a larger space is recommended or at least a different approach.

When comparing the strategies to each other, the stronger points of each are visible. In the first strategy, since the pre-created paths are all optimal, the total distance for a single vehicle case is one of the lowest possible. Due to a poor handling of situations where the path may be blocked, when more vehicles are added it decidedly becomes the worst performing strategy, although the calculation time is always almost non-existent.

The second strategy, which uses a distinctly different approach to the routing issue, shows some of the best results. The graph search technique consistently yields low calculation times and the lowest time ratios. Since all movements are made in straight lines, the distances for the same paths would be marginally longer, which can be seen in the single vehicle test. When more vehicles are added, because the collision avoidance performed better, it leads to superior results regarding both time and distance.

In the third strategy, which uses primarily the OMPL process to find and create new paths, the calculation time increased greatly from almost none of the previous strategies to a considerable number, as expected. It is also a clear improvement from the first strategy, which also uses paths created by the OMPL, in the time and distance necessary to complete the tasks as well as finishing all orders with one more vehicle. When compared to the second strategy, it performs worse in all aspects of all cases except for one vehicle. It can only take two less vehicles, the times and distances were longer and the time ratio and total time to execute the tests were much higher.

For the fourth strategy, and the addition of the *ghost vehicle* as a collision avoidance mechanism, the number of maximum vehicles was increased to five, the highest along with the second strategy. In turn, the time ratio and total time also saw an increase, as the new feature slowed down the simulations a considerable amount.

With the fifth strategy, the first path tested is a pre-created one. This alteration was made with the goal of lowering calculation times, but it lead to a much higher movement time, and therefore, time ratio.

A graphical comparison between all cases where all tasks were completed can be seen in figure 4.7, which helps visualise which strategies perform better. The labels are in format [Strategy] - [Number of vehicles].

From the scatter plot, it can be observed that the distance travelled ranges from 927 m (fourth strategy with one vehicle) to 1423 m (second strategy with five vehicles), a difference of about 53% more. The time it took to complete all tasks is 334 s at a minimum (second strategy with four vehicles) and 990 s at most (second strategy with one vehicle), showing a larger difference of about 196% more.

The time needed to complete all tasks is what most differentiates each case, going from 334 s (second strategy with four vehicles) to 990 s (second strategy with one vehicle). For this parameter, the second strategy clearly stands out as the best performer when multiple vehicles are present, even though it presented the single longest time.

In general, for each strategy, the more vehicles were used, the lower the simulation time became, until it was not able to finish all the tasks given. On the other hand, the distances saw only a small variation, mostly an increase, with the addition of vehicles.

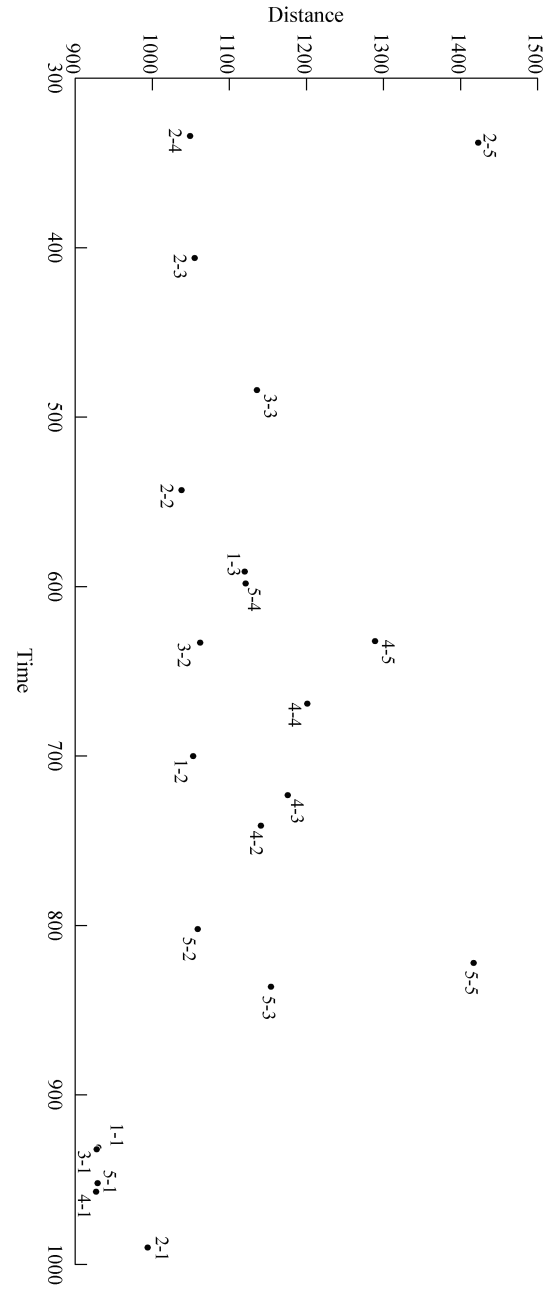


Figure 4.7: Scatter plot showing the time and distance for all successful cases, labels are in format [Strategy] - [Number of vehicles]

Looking at the real time parameters, calculation and movement time, it gives an idea as to how V-REP handles the strategies. These parameters are not important to the strategies themselves, yet are very relevant in judging the software's usefulness as a multi-vehicle routing strategy. To better understand this, the time ratio was calculated for each case.

While the calculation time is strictly dependent of the strategy chosen, the movement time is not. The calculation time in all cases was not much more than whatever time the OMPL took. This is expected, since each OMPL calculation takes 10 s, and any other calculations are quick. The time also increases in strategies where the *ghost vehicle* is present.

The time ratio for the second strategy is far better than any other, making a graph search approach the best option when considering V-REP's performance. The time ratio also increases with complexity of the strategies and as more features or mechanisms are added, the higher it becomes, which is expected.

To obtain the best results it would be necessary to further adjust all parameters, such as the OMPL search time, maximum time elapsed before the vehicle forgets the order or stop the simulation, for example. All these parameters are easily adjusted in the code, as to find the combination that leads to the best results.

4.3 Conclusions

With the implementation and testing of a few strategies, an opinion regarding V-REP's feasibility as a multi-vehicle routing strategy simulator can now be formed.

The implementation here presented for these strategies, or the movement in general, is one created after only a short amount of time using the software. With more experience, and a greater expertise in programming, more complex and better running strategies could possibly be created.

The strategies here presented, show a variety of characteristics that are possible to implement in V-REP and many more could be created. The ones chosen for these tests challenge the software and are capable of showing both the positive, (freedom of programming and versatility) and the negative (e.g. high time ratios and inconsistencies) aspects.

The in-simulation parameters are only indicative of how the strategies perform and parameter tweaking could have been done as to achieve better results. Since the main goal for this work was to test and understand how V-REP could execute routing strategies, the relative better or worse performances of the strategies, as to time and distance, are not as relevant as they would be in a real case scenario.

Chapter 5

Conclusions and Future Work

In this chapter, a few conclusions are made as to how V-REP can be used, not only as a routing strategy simulator, but also as an AGV system simulator in general. Possibilities of further expansion and work are also presented.

5.1 Conclusions

V-REP's biggest quality is the freedom offered by the scripts and its ability to run a large number of them simultaneously. This allows for almost any strategy to be implemented, as long the user is able to program it. Between all the control mechanisms available, there should not be many features that cannot be achieved. It also means that all work must be done by the user, since, excluding the OMPL, nothing else used in this work was previously present in the program.

The scenes can also be easily altered, making it easy to apply the same strategy to a different layout or vehicle, or vice versa, implementing a new strategy to the same layout. This is facilitated by the scripts and the portability of the scene objects and very few changes would have to be made.

The benefit in using V-REP when compared to a more specialized software lies in all the other possibilities it offers, e.g. the routing strategies could be applied to a system where the vehicles are designed in V-REP, taking advantage of the dynamic engine and the cells are made up of models of real manipulators and machines, and actual products are moved in the tasks. This would lead to more realistic simulations, although much slower to complete.

The presence of the OMPL is also very advantageous and something that not all simulators have, which allows the user to quickly implement many different path finding algorithms without having to actually program them. The downside of the current OMPL implementation in V-REP, is that no parameters of the algorithms can be altered and must be used as provided. In some situations, a larger flexibility in their use could have been interesting, and so, programming a new plug-in for OMPL or even the algorithms themselves in V-REP, would have opened up new possibilities.

Since the software is not exclusively a routing or even an AGV simulator, it is not very efficient at these tasks. Since all the elements are present, this sort of behaviour can be programmed, but

for this purpose, other, more specific, programs could possibly be better at it, such as the ones created by AGVS companies and usually not publicly available.

One of the biggest issues with using V-REP as a routing strategy simulator is how it completes the *paths*. Since the code is blocked when performing the path moving commands, such as *simMoveToObject* or *simFollowPath*, no other operations can be executed at the same time. Any strategy where the vehicle is required to stop, change direction or any other process, the movement must be done in increments. This incremental movement is what most affects V-REP's performance and how long the simulations take. If a more accurate simulation is desired, then the distance moved in each loop must be reduced, which would increase the resolution at which other objects can be detected.

Similarly, because of the incremental movement, the time step must be low to achieve accurate and consistent results. The lower the time step is, the longer the simulations take to complete. To assist in speeding up the simulations, the passes per frame can be increased. Another possibility is to use the *Fast Simulation Mode*, which turns the screen black. The side effect of these solutions is that the visual component becomes either very slow or non-existent.

It can then be concluded that a choice must be made between accurate results and fast visuals. Since the choice would usually be accurate results and faster simulation times, it means that the graphical aspect of V-REP becomes irrelevant. V-REP does offer a recording feature that would allow the simulation to be watched in a correct speed later, this however, would further increase the time it takes the simulation to finish.

As for the goals defined in the first chapter, they were all achieved, at least to a pretended level, although some further work could be done, especially in variety of routing strategies and data analysis.

Even though V-REP offers many possibilities and testing routing strategies is completely feasible, especially considering it is a freely available software, it may not always be the best option.

5.2 Future work

There are many possibilities within the context of multiple vehicle routing that were left unexplored or only briefly touched upon.

The routing strategies implemented here are only some of the many possible and more could have been implemented, especially more robust ones. Strategies using conflict-free routing, time windows or dynamic zone planning, would have been interesting, and challenging, strategies to implement, that would further test V-REP's capabilities.

A more realistic vehicle, that more closely resembles a real AGV, would have taken more advantage of V-REP's graphical and dynamic power and perhaps result in more realistic results. The vehicle could be created and programmed taking into account the dynamic and kinematic model, in particular, the steering and drive system. Additional behaviour could also be added, for example, realistic sensors (for the vehicle safety area) or a battery charging behaviour.

Regarding the layouts, more complex features could have been added and programmed, such as buffer areas, uni-directional lanes or double wide lanes. More layout topologies could have also been created, such as a loop layout or more complex and uncommon arrangements.

A complete user interface, where the user could select the layout, number of vehicles, strategy and its parameters to be used in the simulation, would have made the work more complete and user friendly.

Different dispatching and scheduling rules, while not being a direct part of the routing issue, also influence its behaviour and so many other methods to assigning orders could have been chosen. A dynamic task list, one where the machines themselves call the vehicles at different times and orders, would have further tested the strategies in different aspects.

A further and deeper analysis of the data obtained from the multi-vehicle strategies tests could be done, to better understand the strong and weak points of each strategy, both for in-simulation and V-REP performance parameters, not only using the ones already extracted, but also with newer information.

Since the problem of fleet management is so complex, due to being composed of many elements, like dispatching, scheduling, routing as well as the layout used and the vehicles themselves, together with each of these issues being so expansive on their own, individually offering many possibilities each, means that there could have been innumerable ways to combine and adjust each individual parameter of the strategies. Since the routing problem cannot be fully separated from the others, adjusting the other aspects of an AGV fleet management system would affect the routing problem itself, therefore securing a place in this work. Only a few possibilities were implemented, but many others could have been chosen.

References

[1] Russell E King and Carl Wilson. A review of automated guided-vehicle systems design and scheduling. *Production Planning & Control*, 2(1):44–51, 1991.

[2] Tuan Le-Anh and MBM De Koster. A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23, 2006.

[3] Deborah A Davis. Modeling agv systems. In *Proceedings of the 18th conference on Winter simulation*, pages 568–574. ACM, 1986.

[4] Lothar Schulze, Sebastian Behling, and Stefan Buhrs. Automated guided vehicle systems: a driver for increased business performance. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 2, 2008.

[5] 2017. URL: http://www.konecranes.com/equipment/container-handling-equipment/automated-guided-vehicles/agv [last accessed January 2017].

[6] HR Everett. *Sensors for mobile robots: theory and application*. AK Peters, Ltd., 1995.

[7] Johann Borenstein, HR Everett, Liqiang Feng, et al. *"Where am I?" Sensors and methods for mobile robot positioning*. Citeseer, 1996.

[8] 2017. URL: https://www.amazonrobotics.com [last accessed February 2017].

[9] 2017. URL: http://mhi.org [last accessed February 2017].

[10] Tharma Ganesharajah, Nicholas G Hall, and Chelliah Sriskandarajah. Design and operational issues in agv-served manufacturing systems. *Annals of Operations Research*, 76:109–154, 1998.

[11] Yavuz A Bozer and Mandyam M Srinivasan. Tandem configurations for automated guided vehicle systems and the analysis of single vehicle loops. *IIE transactions*, 23(1):72–82, 1991.

[12] Ronald J Mantel and Henri RA Landeweerd. Design and operational control of an agv system. *International Journal of Production Economics*, 41(1-3):257–266, 1995.

[13] 2015. URL: http://www.agvsystems.com/controls-landing/ [last accessed January 2017].

[14] 2017. URL: http://www.egemin-automation.com/en/automation/material-handling-automation_ha-solutions_agv-systems_agv-software-systems/management-software [last accessed May 2017].

[15] 2015. URL: http://www.agvsystems.com/q-can/ [last accessed January 2017].

[16] Christine G Co and José Mario Azaña Tanchoco. A review of research on agvs vehicle management. *Engineering Costs and Production Economics*, 21(1):35–42, 1991.

[17] Pius J Egbelu and José MA Tanchoco. Characterization of automatic guided vehicle dispatching rules. *The International Journal of Production Research*, 22(3):359–374, 1984.

[18] CM Klei and J Kim. Agv dispatching. *International Journal of Production Research*, 34(1):95–110, 1996.

[19] Mark B Duinkerken, Jaap A Ottjes, and Gabriel Lodewijks. Comparison of routing strategies for agv systems using simulation. In *Proceedings of the 38th conference on Winter simulation*, pages 1523–1530. Winter Simulation Conference, 2006.

[20] Laiguang Zeng, Hsu-Pin Wang, and Song Jin. Conflict detection of automated guided vehicles: a petri net approach. *The International Journal of Production Research*, 29(5):866–879, 1991.

[21] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT press, 2011.

[22] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[23] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[24] N Sariff and Norlida Buniyamin. An overview of autonomous mobile robot path planning algorithms. In *2006 4th Student Conference on Research and Development*, pages 183–188. IEEE, 2006.

[25] Radu Robotin, Cosmin Marcu, and Gheorghe Lazea. *Graph search techniques for mobile robot path planning*. INTECH Open Access Publisher, 2010.

[26] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[27] Kavraki Lab. Open motion planning library: A primer, July 2016.

[28] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[29] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[30] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

[31] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.

[32] 2017. URL: http://www.ros.org [last accessed May 2017].

[33] Cyberbotics, 2017. URL: https://www.cyberbotics.com/webots.php [last accessed October 2016].

[34] 2014. URL: http://gazebosim.org [last accessed January 2017].

[35] 2017. URL: http://www.coppeliarobotics.com [last accessed February 2017].

[36] Lucas Nogueira. Comparative analysis between gazebo and v-rep robotic simulators. 2008.

[37] 2017. URL: http://www.coppeliarobotics.com/helpFiles/ [last accessed February 2017].

[38] 2017. URL: http://www.forum.coppeliarobotics.com [last accessed February 2017].

[39] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[40] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1321–1326. IEEE, 2013.

[41] 2017. URL: https://www.lua.org [last accessed May 2017].

[42] 2004. URL: https://www.lua.org/pil/21.html [last accessed February 2017].