NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
NAME OF THE DEPARTMENT

TOMÁS ALEXANDRE DIAS PEDREIRA
BSc in Electrical and Computer Engineering

A VERY LONG AND IMPRESSIVE THESIS TITLE WITH A FORCED LINE BREAK

SOME THOUGHTS ON THE LIFE, THE UNIVERSE, AND EVERYTHING ELSE

MASTER IN STUDY PROGRAM NAME
SPECIALIZATION IN SPECIALIZATION NAME

NOVA University Lisbon
September, 2000

DEPARTMENT OF
NAME OF THE DEPARTMENT

A VERY LONG AND IMPRESSIVE THESIS TITLE WITH A FORCED LINE BREAK

SOME THOUGHTS ON THE LIFE, THE UNIVERSE, AND EVERYTHING ELSE

TOMÁS ALEXANDRE DIAS PEDREIRA

BSc in Electrical and Computer Engineering

Adviser: Ricardo Peres
Associate Professor, NOVA University Lisbon

Co-adviser: Alexandre Mata
Research Engineer, NOVA University Lisbon

Examination Committee

Chair: Name of the committee chairperson
Full Professor, FCT-NOVA

Rapporteur: Name of a rapporteur
Associate Professor, Another University

Members: Another member of the committee
Full Professor, Another University

Yet another member of the committee
Assistant Professor, Another University

MASTER IN STUDY PROGRAM NAME
SPECIALIZATION IN SPECIALIZATION NAME

NOVA University Lisbon
September, 2000

A Very Long and Impressive Thesis Title with a Forced Line Break
Some thoughts on the Life, the Universe, and Everything Else

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.3.9) [1].

Dedicatory lorem ipsum.

Acknowledgements

Acknowledgments are personal text and should be a free expression of the author.

However, without any intention of conditioning the form or content of this text, I would like to add that it usually starts with academic thanks (instructors, etc.); then institutional thanks (Research Center, Department, Faculty, University, FCT / MEC scholarships, etc.) and, finally, the personal ones (friends, family, etc.).

But I insist that there are no fixed rules for this text, and it must, above all, express what the author feels.

"You cannot teach a man anything; you can only help him discover it in himself."

— **Galileo**, Somewhere in a book or speach
(Astronomer, physicist and engineer)

Abstract

Over the years, the world of technology has been expanding in every possible field, and agriculture is no exception. The use of technology in Smart Agriculture has been increasing, with Internet of Things networks, Artificial Intelligence systems for management and forecasting, and robotics to automate processes, do the repetitive jobs humans would rather not do, like applying pesticide, harvesting products, and evaluating the state of the fields, and also some that humans can't do, surveilling large fields. These developments not only come to facilitate tasks but also to battle ongoing problems with the agricultural environment, like the lack of young people willing to do hard work on the fields and the increase of the world population, that will increase the need for food production. This work is focused on the development of a tractor-trailer robot, capable of autonomously navigating through a field while spraying pesticides directly on the products. The focus is on the navigation side of the robot, researching the use of robotics in Smart Agriculture, researching robot path planning methods like Voronoi Graphs, Visibility graphs, Sampling-based approaches like the Rapidly exploring Random Trees and the Probabilistic Roadmap, Bio-heuristic algorithms like the Ant Colony Optimization, Particle Swam Optimization and the Genetic Algorithm, and also mentioning the learning-based methods. This work will also review a few control methods included in the library nav2 of ROS2 like the Dynamic Windows Approach, Pure Pursuit, and the Model Predictive Controller. This task is already a challenging one, however, with the addition of the trailer to the system, the task becomes even more challenging, as the trailer dynamics will need to be accounted for. In the end the chosen methods to be tested will be the Voronoi Graph with the A* algorithm for path planning, and the Pure Pursuit controller for the trajectory tracking. The tractor-trailer system was chosen, despite its complexity and limited documentation, due to its modularity, allowing for independent operation and reusability.

Keywords: Smart Agriculture, tractor-trailer, IoT, Artificial Intelligence, Robotics, Path planning, Robot control, ROS2, Voronoi Graph, A*, Pure Pursuit

Resumo

Ao longo dos anos, o mundo da tecnologia tem-se expandido em todos os possíveis campos, e a agricultura não é exceção. O uso de tecnologia na Agricultura Inteligente tem vindo a aumentar, com redes de Internet of Things, sistemas de Inteligência Artificial para gestão e previsão, e robótica para automatizar processos, realizando as tarefas repetitivas que os humanos preferem não fazer, como aplicar pesticidas, colher produtos e avaliar o estado dos campos, além de algumas que os humanos não conseguem fazer, como a vigilância de grandes áreas agrícolas. Estes desenvolvimentos não só facilitam as tarefas, como também combatem problemas contínuos no ambiente agrícola, como a falta de jovens dispostos a realizar trabalhos duros no campo e o aumento da população mundial, que irá intensificar a necessidade de produção de alimentos. Este trabalho centra-se no desenvolvimento de um trator com reboque, capaz de navegar autonomamente por um campo enquanto aplica pesticidas diretamente sobre os produtos. O foco é na navegação do robô, investigando o uso da robótica na Agricultura de Precisão, explorando métodos de planeamento de trajetórias para robôs, como Voronoi Graphs, Visibility Graphs, abordagens baseadas em amostragem como o Rapidly exploring Random Trees e o Probabilistic Roadmap, algoritmos bio heurísticos como o Ant Colony Optimization, Particle Swarm Optimization e Genetic Algorithm, e mencionando ainda métodos baseados em aprendizagem. Este trabalho também irá rever alguns métodos de controlo incluídos na biblioteca nav2 do ROS2, como o Dynamic Windows Approach, Pure Pursuit e o Model Predictive Controller. Esta tarefa já é desafiante por si só, no entanto, com a adição do reboque ao sistema, a tarefa torna-se ainda mais complexa, pois será necessário considerar as dinâmicas do reboque. No final, os métodos escolhidos para serem testados serão o Voronoi Graphs com o algoritmo A* para o planeamento de trajetórias e o controlador Pure Pursuit para o seguimento da trajetória. O sistema trator-reboque foi escolhido, apesar da sua dificuldade e escassez de documentação, devido à sua modularidade, permitindo que o seu funcionamento seja, por regra, independente e reutilizável.

Palavras-chave: Agricultura Inteligente, trator-reboque, IoT, Inteligência Artificial, Robótica, Planeamento de trajetórias, Controlo, ROS2, Voronoi Graphs, A*, Pure Pursuit

Contents

List of Figures

List of Algorithms

Listings

List of Listings

Glossary

This document is incomplete. The external file associated with the glossary 'main' (which should be called `template.gls`) hasn't been created.

Check the contents of the file `template.glo`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

If you don't want this glossary, add `nomain` to your package option list when you load `glossaries-extra.sty`. For example:

```
\usepackage[nomain]{glossaries-extra}
```

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

  ```
  \usepackage[automake]{glossaries-extra}
  ```

- Run the external (Lua) application:

  ```
  makeglossaries-lite.lua "template"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "template"
  ```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

Acronyms

This document is incomplete. The external file associated with the glossary 'acronym' (which should be called `template.acr`) hasn't been created.

Check the contents of the file `template.acn`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

  ```
  \usepackage[automake]{glossaries-extra}
  ```

- Run the external (Lua) application:

  ```
  makeglossaries-lite.lua "template"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "template"
  ```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

Symbols

This document is incomplete. The external file associated with the glossary 'symbols' (which should be called `template.sls`) hasn't been created.

Check the contents of the file `template.slo`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

  ```
  \usepackage[automake]{glossaries-extra}
  ```

- Run the external (Lua) application:

  ```
  makeglossaries-lite.lua "template"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "template"
  ```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

1 Introduction

In this chapter, there will be a background explanation of the proposed problem, the motivation for a solution and the document structure.

1.1 Background

Agriculture is one of the most essential industries, providing a source of food for the growing population around the globe. However, the industry faces several challenges, such as the need to increase food production, for example, according to the Food and Agrivulture Organization (FAO) of the United Nations, around eight hundred million people are undernourished, and two thirds of them live in Asia. For example, in India, over 70% of the population is dependent on agriculture for their livelihood [2], and, due to lack of the development of the countries' agricultural processes, most farmers use very traditional methods of farming, which are not efficient requiring a lot of time and manual labour. To overcome these issues and to meet the growing demand for food, the agriculture industry is turning to technology, such as robotics, to improve efficiency and productivity.

The use of technologies like AI, Internet of Things, and robotics facilitates the automation of several tasks, such as planting, weeding, harvesting, forecasting, and monitoring which tackles the issues of labour shortages and allowing for easier expansion, solving the need for more food production. One particularly important application of robotics in agriculture is in pesticide spraying. Traditionally, pesticide application has been a labour-intensive and slow task. Autonomous pesticide spraying robots address these issues by applying pesticides more accurately, reducing chemical waste, and minimizing human exposure to harmful substances.

1.2 Motivation

The motivation behind the development of a tractor-trailer pesticide spraying robot is the need to improve agricultural processes to meet the demand for food production and safety. By integrating these robots in Smart Farming, farmers can optimize pesticide usage, reducing the amount of pesticide wasted and overexposure to chemicals. The solution proposed in this work is to develop a trailer-tractor system, where the tractor will be responsible for towing a trailer with a pesticide spraying system. The advantages of this system are its modularity, with each part acting independently, allowing for easier reuse and maintenance, and also the fact that the development of the robot is not constrained by the functionality of the system to be integrated, meaning that the tractor can be developed freely without having to account for the space occupied by the fluid tanks and balancing fluid tanks, leaving more room to develop both systems independently.

The main technical issue with these systems is their non-holonomic characteristics, meaning that the tractor has its movement restricted due to the trailer's dynamics. A good example of a problem these systems can have is when a differential drive robot tries to rotate in place, the trailer will simply be dragged along, drifting sideways, where it might end up in a position that isn't the desired one or even capsizing. This dynamic makes planning and control very challenging since both the planning and control systems will have to account for the constraint on the turning radius of the tractor-trailer system when calculating a path and sending motion commands.

When analysing the advantages and disadvantages of the proposed system, it is clear that, with a good path planning and motion control solution, the advantages far outweigh the disadvantages making the proposed solution a viable approach for increasing safety and efficiency in smart agriculture.

1.3 Document Structure

This document is divided into 4 chapters. The first chapter is the introduction, where the problem is presented along with the motivation for the work and the document structure. The second is the state of the art review, where a contextualization of mobile robotics in agriculture is made along with the most popular methods for path planning and robot control, and some previous work on the trailer-tractor system. The third chapter is the planning and schedule of the work, where the work proposal, schedule and results publishing plan are presented. The last chapter is the conclusion, where a summary of the document is presented.

2

Literature Review

In this chapter, four main topics will be explored: Mobile Robotics in Smart Agriculture, Motion Planning and its Approaches, Robot Control Methods, and Tractor-Trailer-like System Applications, as well as an analysis of the results.

2.1 Mobile Robotics in Smart Agriculture

According to UNESCO, the world population is going to increase by about 30% in the next 25 years and with it, the need for food production. To satisfy this need, there will have to be an upgrade in agricultural production and to do it, there has been a development of IoT and AI technologies to help gather information and make better decisions about the crop fields [3]. These technologies acquire data using several sensors, such as humidity sensors, pH sensors, temperature sensors and substance level sensors, that communicate wirelessly using a variety of case dependant protocols (like ZigBee or LoRa) to make decisions using specific algorithms like PID controllers, Time-Controlled algorithms, Fuzzy-Logic or Machine Learning algorithms like neural networks.

According to [4, 5] there are also other factors to be considered and those are climate change and the heavy use manual labour, which could be scarce if another pandemic or catastrophic event happens, preventing people from working. Agriculture will also need to adapt to this, not only using IoT and AI but also using mobile autonomous robots, alone or as a collaborative system. These robots are used to perform manual labour tasks (seeding, planting, harvesting), acquire information about the fields through physical sensors or cameras built into the robots, and apply pesticides and disease control.

To execute most of these tasks like harvesting or seeding, some sort of Unmanned Ground Vehicle (UGV), like a tractor, is used. For the execution of other tasks like large area monitoring or the broadcasting of pesticides an Unmanned Aerial Vehicle (UAV), like a drone, is often used. These autonomous vehicles can handle tasks that humans would struggle to do more cheaply and quickly. However, these machines have a limited battery capacity and, therefore, need to be designed specifically for the task at hand. For instance, in the case of UAVs, when covering a large area for monitoring, a lot of battery life is required. To facilitate this, the software and overall development of the robot need to be specialized [5, 6].

Going more in-depth into the design features, there are four main issues to consider: locomotion, sensing, planning, and manipulation. Starting with locomotion, there are various types of designs, including wheeled robots, flying robots, railed robots, wire-suspended robots, legged robots, and tracked robots which are used, for example, in wet environments where wheels might get stuck in the mud. When it comes to sensoring, it is also highly dependent on the task at hand. If the goal is to differentiate between two fruits with similar colours, a regular digital camera might not suffice, a more sophisticated camera with advanced image processing software may be needed. Additionally, some characteristics of the environment change, and therefore, the algorithms used must account for factors such as direct sunlight, temperature, and humidity. Moreover, planning is the list of decisions the robot has made to complete its task successfully. There are several algorithms used, however, the most advanced ones are the sample-based ones which have greater performances. Finally, manipulation is the control of manipulators, like robotic arms with grippers, to execute a certain task [7]. The choice of manipulator depends on the task, however, some robots have multiple manipulators, called Multi-functional Intelligent Agricultural Robots (MIARs), to perform multiple tasks, [7] which reduces the number of robots and costs. Per [5, 6] some robots are also programmed to be able to perform multi-objective path planning which increases efficiency of movement in complex environments, increasing overall efficiency of the robot.

As an example, in [8], a robot was created to compensate the lack of labour caused by the aging of the population of Taiwan. The environment characteristics of Penghu, Taiwan, are prone to the overproduction of dragon fruit (Pitaya), and therefore this problem needs to be fixed to fight its production. Since the terrain is very shallow and has a poor water retainability, in addition to droughts, the soil has become arid which might make a wheeled robot bog down. The choice of locomotion for this robot were the tracks. For sensoring, a 9-axis gyroscope, to get the robots deviation, a Webcam (Image Sensor), to get the fruits location, and a Lidar to measure the distance between itself and any obstacle in its way. To harvest the fruit, it uses a e-axis robot arm with a gripper end-effector. Finaly, path-planning wise, it uses the 2-D SLAM algorithm to generate a pixel map and then the Dijkstra algorithm to calculate the fastest route to the target position. The experiments on the field revealed a 97% harvest success rate meaning the method used by the author is appropriate.

Another example of a solution is the robot in [9], a platform robot developed to try to compensate for the aging of the South Korea's population which is causing a lack of young labour. The robot was designed to work on a paprika greenhouse with a 3-meter-wide corridor, where workers traverse, and 0.5-meter-wide ridge between crops where the robot will operate. The ridges between crops have hot water pipping which serve as rails and therefore the robot was designed to be both railed and free driven. Its locomotion system consists of a two Wheeled drive kit with two main traction wheels in the middle, and four auxiliary smaller wheels to level the platform. Two in the front and two in the back. To move autonomously it incorporated a Lidar, to map the environment, a camera, to detect objects, and an encoder, to get the current position of the robot. The chassis was adapted to serve as a vacuum system, a lift table system or as just a manipulator. It had two controllers, one for the movement control system and another for the manipulation system. However, only the movement system was tested. This robot was not tested in the field, but, passed both the maximum load test (withstood 400Kg with no change of the distance between chassis and ground). The autonomous driving had some problems with the steering and network making it not ready to operate, however, it is a good start to solving this agricultural problem.

2.2 Motion Planning

Motion planning is of the most important steps to achieve autonomous movement in robots, and there are several methods that can be implemented to achieve this. The goal is to calculate a path that will get the configuration of the robot from a starting position qstart to a goal configuration qgoal, without colliding with any obstacle, and to achieve this there exist several methods. In [10, 11], the methods are divided into two categories, Classical Approaches, which include Cell Decomposition, Roadmapping, Sampling-based, and Bio-Inspired approaches, Roadmapping approaches, and the Learning Approaches which are the Supervised learning, Unsupervised learning and Reinforcement Learning based approaches. Additionally, according to [11], there are two types of planners, Global and Local planners. Global planning is when a path is planned with prior knowledge of the environment, whereas Local planning is when a path is planned reactively as the knowledge of the environment is being acquired in real time. Global planners focus on modelling method for the environment and path selection where Local planners focus on using data acquisition devices. Both types of planners can use the same approaches for path planning.

To continue there we'll need some definitions:

- C-space (C) are all the possible configurations the robot can have, for example in a robotic arm it is all the angles its joints can be in;

- C_{obs} are all the configurations where the robot is in contact with an obstacle;

- C_{free} are all the configurations possible for the robot to be in, but not in contact with an obstacle $C_{free} = C \setminus U\ C_{obs}$;

2.2.1 Cell Decomposition approaches

These approaches are called Grid-based because they divide the C-space into a grid where each centre point of a cell is a node. The path is chosen as a sequence of these nodes using algorithms like Dijkstra's algorithm or A* for example. There are three main types of Cell Decomposition approaches, the exact, adaptive and approximate cell decompositions [12].

2.2.1.1 Exact Cell Decomposition

The Exact Cell Decomposition approach divides the C-space into small polygonal elements formed by vertical lines created by every vertex of the obstacles. See Figure 2.1 for visual aid.

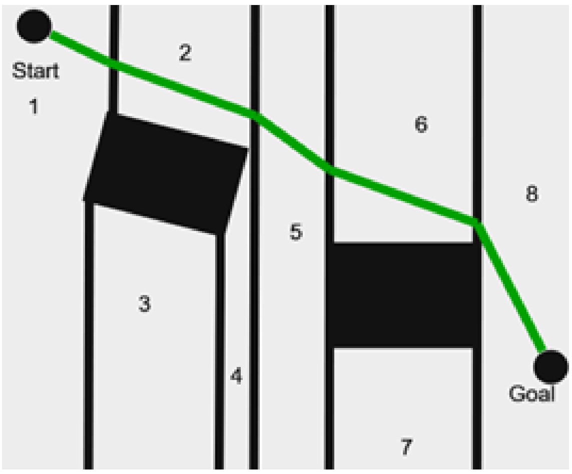

Figure 2.1: Exact Cell Decomposition representation; Green line is a possible chosen path

2.2.1.2 Adaptive Cell Decomposition

The Adaptive or Quadtree Cell Decomposition approach firstly divides the C-space into four cells of the same size. Then every cell that is partially occupied will keep dividing itself until every cell is either free of obstacles or fully occupied. See Figure 2.2 for visual aid.

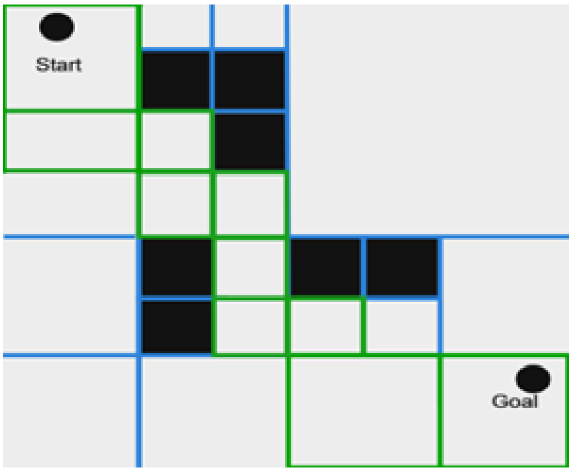

Figure 2.2: Adaptive Cell Decomposition representation; Green Squares are the chosen cells/nodes for the shortest path

2.2.1.3 Approximate Cell Decomposition

Finally, the Approximate Cell Decomposition approach simply divides the C-space into a grid with known cell size. The smaller the cells, better optimised the path, however, more combinations of path exist making computation more time consuming. There are two types of Approximate Cell Decomposition, 8-connected and 4-connected where the former accounts for diagonal movement between nodes and the ladder only vertical and horizontal movement. See Figure 2.3 for 8-connected.

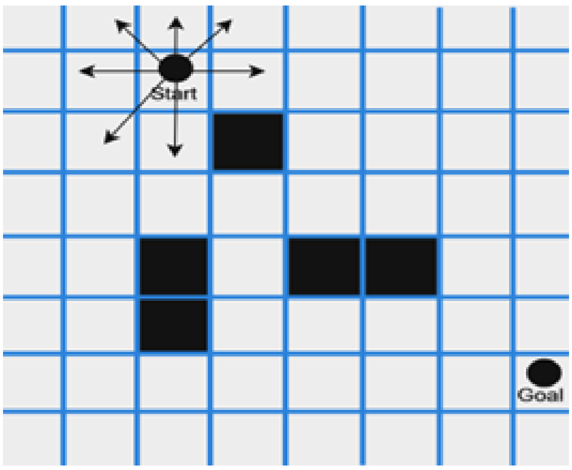

Figure 2.3: Approximate 8-connected Cell Decomposition representation; Arrows are all represent every possible movement the robot can make

2.2.2 Grid/Graph Search algorithms

Most of the motion planning approaches create graphs or grids where nodes are connected forming paths. After the paths are created the best one needs to be chosen. To make this choice, graph search algorithms are used. The most common ones are the Dijkstra's algorithm and the A* algorithm.

2.2.2.1 Dijkstra's Algorithm

Dijkstra's algorithm works by looking at the node with the shortest distance to the root of the graph, which in the beginning is the root itself. Then it calculates the distance to all the nodes connected to the that node (every edge has a weight), and then chooses the node with the shortest distance to the root. Every node keeps track of the parent node and is updated if a shorter path to said node is found. This process is repeated until every node is visited. The path is then found by backtracking from the goal node to the root node.

(a) (b) (c)

Figure 2.7: 8-connected Dijkstra's Algorithm; a) is the starting C-space; b) is a mid point of the algorithm; c) is the end point of the algorithm; The green squares are the searched neighbours, the red ones are the visited nodes and the blue ones are the chosen path.

2.2.2.2 A* Algorithm

The A* is similar to the Dijkstra's algorithm, as it also goes from node to node calculating distances, however, it has a heuristic function that estimates the distance from the current node to the goal node. The value stored in each node instead of being the path distance from the root, is the sum of said distance and the estimated distance towards the goal.

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

where $f(n)$ is the value stored in the node, $g(n)$ is the distance from the root to the node, and $h(n)$ is the estimated distance from the node to the goal. This way the chosen nodes are biased towards the goal, making the algorithm faster than the Dijkstra's.

The algorithm starts in the same way, calculates the f value for all the root adjacent nodes, chooses the smallest value, and then calculates the f value for all the nodes connected to the chosen node. This process is repeated until the goal node is reached. If a value is found that is smaller than the one stored in the node, the value is updated. The path is then backtracked from the goal node to the root node.

As seen in Figure 2.11, the path has more diagonal transitions that the one in Figure 2.7, making it slightly longer. This is due to the heuristic function used in the A* algorithm which makes it go faster towards the goal, but not always in the shortest path.

(a) (b) (c)

Figure 2.11: 8-connected A* Algorithm; a) is the starting C-space; b) is a mid point of the algorithm; c) is the end point of the algorithm; The green squares are the searched neighbours, the red ones are the visited nodes and the blue ones are the chosen path.

2.2.3 Artificial Potential Field

The traditional Artificial Potential Field (APF) approach consists of assigning a positive potential to obstacles and a negative one to the end state. Using the robot as a positive charge, the end goal will have an attractive force on the robot while the obstacles a repulsive force. This way the robot is attracted to the goal configuration while being repelled by the obstacles. This method is advantageous due to its simplicity [13] and due to its speed when compared to the graph search algorithms [14]. However, the main problem with this algorithm is its susceptibility to the local minima problem [13, 14]. If there is ever a configuration where the repulsive forces generated by the obstacles are symmetrical to the attractive one of the goal state, the robot will not be able to move.

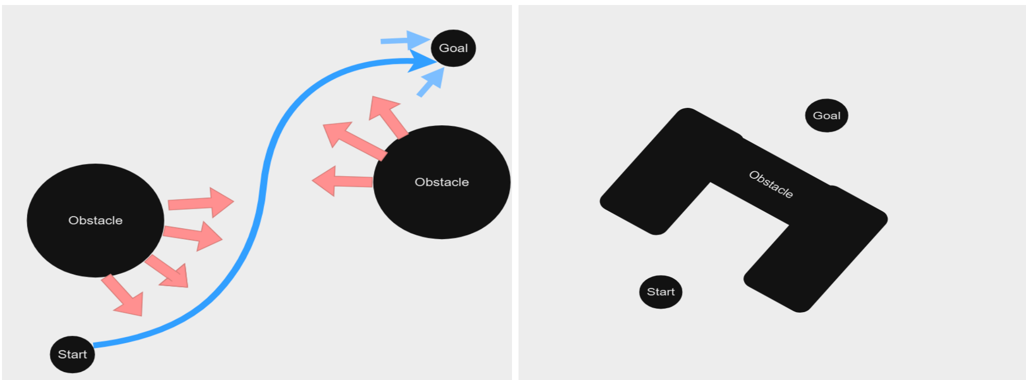

Figure 2.12: APF algorithm; On the left is the normal functioning of the algorithm where the pink arrows represent obstacle repulsion, the smaller blue ones represent the goal attraction and the curvy blue arrow is the chosen path; On the right is the local minima situation.

To solve this local minima problem, the author in [15] proposed a solution where when the robot is detected to be facing this problem, it will move in a random forward direction to get out of that situation. Another approach is to combine multiple methods like the author of [16] proposed. His approach was to use the traditional APF until a local minima position is reached and then switches to using the RRT*, a sampling-based algorithm explained in 2.2.4.2, to escape it.

2.2.4 Sampling-based approaches

There are two main sampling-based approaches, the Probabilistic Roadmap (PRM) and the Rapidly-exploring Random Tree (RRT). These algorithms sample the C-space creating nodes and then connecting them forming a roadmap.

2.2.4.1 Probabilistic Road Map

The PRM is divided into two phases, being the construction and query phase. The construction phase starts by sampling the C_{free} and establishing feasible or non-obstructed connections between the sampled nodes, creating a roadmap R. The query phase joins the initial and goal configurations to the roadmap by connecting them to the closest non-obstructed nodes and then, by using, for example, a distance-reduction algorithm like the A* or Dijkstra, obtaining the desired path. This approach has difficulty in narrow environments because of its probabilistic nature it is not certain that the sampled nodes can make connections.

In [17] this technique is used along with the APF approach and the Approximate Cell Decomposition approach creating the HPPRM or Hybrid Potential Based Probabilistic Roadmap, to avoid the problem mentioned. It first creates a Potential map with the obstacles' information, then it divides the map into a grid with cells with the same size. Afterwards classifies the cells into High or Low potential depending on the repulsive values calculated. Higher potential cells would have more samples than lower potential ones since making connections in the ladder would be easier. After having the nodes, the process is the same as the one in PRM, connections between nodes are made and then the Dijkstra's algorithm is used to obtain the shortest path between initial and goal configurations.

2.2.4.2 Randomly Exploring Random Trees

The other method, the RRT, consists of, with the starting configuration as the root of the tree, randomly sampling the C_{free} and creating a node that connects to the nearest one until the end goal is reached. The creation of the node isn't always on the sample but within a maximum distance in the direction to the nearest node. Once the end goal is reached, all needed to do to get the path is to follow the only path from root to goal state. Despite always being able to find a feasible path, as samples approach infinity, this method as an issue, since the samples are random, the path created might not be optimal in a distance sense. As a solution to this problem, the RRT* algorithm was created. The node creation process is the same as the RRT, a sample is randomly generated, find the nearest node, create a node within the maximum distance from the nearest node and the sample. The difference is in the connection to the tree, instead of connecting it to the nearest node, it searches the nodes around it, within a certain range, and checks if they can be rewired so that the paths are shorter. The sampling ends when a short enough path is found, or the number of samples reaches the desired amount. There is also another variation of RRT which is the bidirectional RRT which works exactly like the RRT but both goal and stating configurations start as a root to a tree, expanding towards each other.

In [18], a combination of variations of the RRT are used. The author mentions the use of the Bi-RRT to find, in less time, a feasible path between the starting and goal configurations. However, due to the algorithm's random nature and robot's constraints, the connections between the two trees are made by solving a 2-point Boundary Value Problem which may take too much time in a practical scenario. To get around this issue, the author proposed the use of a bidirectional-unidirectional-RRT which functions as a regular Bi-RRT with the two trees growing into each other, but, when close enough to connect, it will use a unidirectional search, starting from the initial configuration's tree, with a bias towards the goal state's tree.

2.2.5 Roadmapping approaches

The Roadmapping approaches are divided into main two types, the Visibility Graphs and the Voronoi Diagrams which are motion planning approaches that form connections creating roadmaps.

Note: The PRM and RRT approaches weren't included in this section due to their stochastic nature. For the same C-space there can be multiple roadmaps using PRM and RRT, whereas in the Visibility Graph and Voronoi Diagram approaches there is only one.

2.2.5.1 Visibility Graph

Visibility graphs function by representing every object as a polygon and then connecting every adjacent vertex, including starting and end points, forming straight lines that do not go through objects. Once multiple connections are formed an optimal path is chosen using an algorithm like A* or Dijkstra's. However, this approach has the problem of being time complex and not being dimensionally scalable [19].

An example of the usage of this method can be seen in [20] where the visibility graph approach is used for in real time obstacle avoidance for a UAV. However, due to mechanical restraints the aircraft needed the connections between edges to be smoother, and therefore used the Dubin's curves method, which makes a path from A to B using a sequence of straight and curved lined with a minimal R radius imposed imposed by the necessary restraints [21]. Another example of the use of this approach is in [22] where the author used the Visibility Graph and the Adaptive or Quadtree Cell Decomposition methods to create a path for a marine USV for the coast of South Korea. The map was divided into a grid with four cells, NW, NE, SW, SE, and then would be divided like explained in 2.2.1. After the construction of the quadtree graph, the visibility graph is created using the nodes in the centre of each cell. The shortest path is then chosen using the Dijkstra's algorithm.

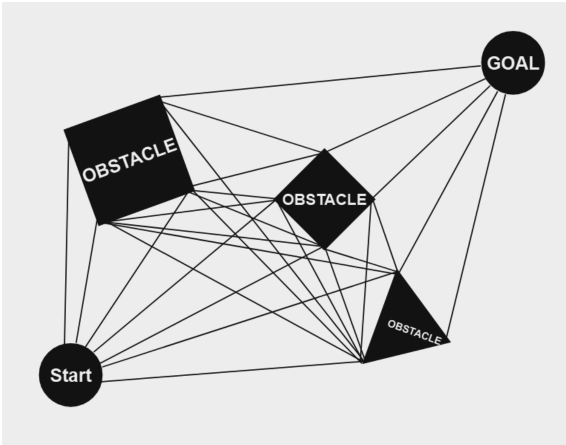

Figure 2.13: Visibility Graph representation; As explained, the black lines are the connections between the vertices that will then be searched for the shortest path

2.2.5.2 Voronoi Diagram

The Voronoi Diagram consists of two sets, the vertex or node set, and the edge set. The edge set is the collection of all the lines which are equidistant to two objects, and the vertex set is the collection of all the points where three or more lines intersect. These objects may be obstacles or just workspace boundaries. Again, like in the Visibility Graphs, after the diagram is created, the best path can be chosen using an algorithm like A* or Dijkstra.

The usage of this approach can be exemplified in [23] where the author used the Voronoi Diagram along with the RRT* and Potential Field approaches to create a path for a UGV in a indoors environment. Firstly, the planner would create a Voronoi Diagram of the C-space, however This would not suffice for a successful path plan due to the robots mechanical constraints. For this, a potential function was used to create a map where the most attractive points would be in the path chosen using the Voronoi Diagram. Afterwards, the RRT* algorithm was used to create the optimal path sampling with a bias towards the attractive field. Despite this approach being successful in most cases, there were still cases where the robot's dynamics would prevent it from following the desired path. The author then suggests that this problem would easily be overcome by making a circle around the robot and if the circle isn't able to move in that position, the position would be discarded, and another path would be chosen in the Voronoi Diagram phase.

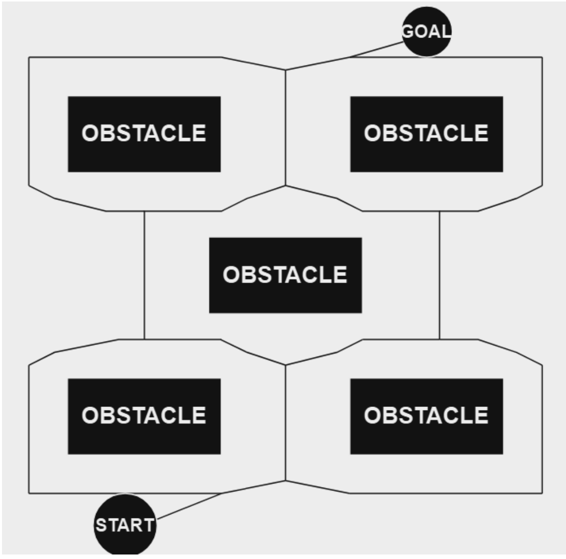

Figure 2.14: Voronoi Diagram representation; The black lines keep the same distance to the obstacles and the map edges

2.2.6 Bio-Inspired approaches

The Bio-Inspired approaches are, as the name would suggest, inspired by natural systems. The main methods are the Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and the Genetic Algorithm (GA). These optimization algorithms minimize a specific Objective or Cost function, which could factor path length, energy spent, or any other example specific characteristic that could impact performance [24].

2.2.6.1 Particle Swarm Optimization

The PSO algorithm starts by creating a swarm of particles, each with a position and velocity. In the motion planning context, each particle represents a possible solution or, in other words, a feasible path. The particles' position and velocity are updated, with each iteration, according to the following equations:

$$v_i^{t+1} = wv_i^t + c_1r_1(pbest_i - x_i^t) + c_2r_2(gbest - x_i^t) \tag{2.2}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.3}$$

where v_i^t is the velocity of particle i at time t, w is the inertia weight, c_1 and c_2 are the cognitive and social coefficients, r_1 and r_2 and randomly generated values between 0 and 1, $pbest_i$ is the best position of particle i, and $gbest$ is the best position of the swarm. The algorithm stops when a maximum amount of iterations K is reached.

This approach was used in [25] the author defined the particles as a set of parameters of cubic splines, which would connect to form a smooth path for the robot. The cost function used considered the path length, the minimum distance between the robot and the closest obstacle, and the Euclidean distance between the robot and the goal configuration. This method was compared to the visibility graph and the APF and found that it was able to find a smoother and more easily executable path (less discontinuities) while maintaining a short path distance. Due to how slow the algorithm is, the author says that it can only be used in a static environment.

2.2.6.2 Ant Colony Optimization

The ACO algorithm is inspired by the behaviour of ants when looking for food. When traveling, ants leave pheromone trails so the other ants can follow them. The algorithm tries to mimic this behaviour by creating a set of artificial ants that will create a path from the starting to the goal configuration. The probability of a k ant of going from node i to node j at time t is given by the following equation:

$$P_{ij}^{k} = \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{k \in N_i} \tau_{ik}^{\alpha}(t)\eta_{ik}^{\beta}(t)} \tag{2.4}$$

where N is the set of nodes connected to i, τ is the pheromone level of the path from i to j, η is the heuristic information or proximity, α is the pheromone incentive factor which means that the larger the α the more wight the pheromone level has on the probability of a path being chosen, and β which acts the same way as the precious coefficient only not for pheromone level but for proximity between nodes [26]. The pheromone level is updated at the end of each iteration using the following equation:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \tag{2.5}$$

where ρ is the pheromone evaporation rate and $\Delta\tau_{ij}$ is the increment amount of pheromone left by the ants in the path from i to j. Given a maximum number of iterations, naturally the shortest path will be the one with the most pheromone left by the ants. Due to its robustness, global optimization ability, and possibly being used in combination with different heuristic algorithms, the ACO is widely used in path planning. However, in complex environments, the algorithm may get stuck in a local minima or get into a Deadlock state where the ants can't escape from a specific node as there are not any not explored ones nearby [27].

In [28] the author proposes an improvement to the ACO algorithm to plan a path in a radioactive environment. The proposed method added a chaos optimization algorithm to create initial random initial paths, it changed the pheromone update equation to include radiation level as a negative trait and also added a local search optimization to avoid the local minima problem. This method proved to be more efficient than the traditional ACO and PSO algorithms in the same scenario.

2.2.6.3 Genetic Algorithm

This method is inspired on the evolution of species. The algorithm starts by creating a random population of chromosomes, each representing a possible solution to the problem, like in the PSO. With every iteration each chromosome is evaluated using a fitness function, which is the cost function in the motion planning context. The better performing chromosomes have increased changes of being selected for reproduction. The reproduction process is done by selecting two chromosomes and crossing them over, creating a new chromosome. This new chromosome can be mutated in order to create different solutions and therefore diversity. The process is repeated until a maximum amount of iterations is reached or a desired fitness level is achieved.

An improved version of this method was used in [29] to plan a path for an Unmanned Surface Vehicle. The chromosome encoding was done using sets of angles and velocities (θ, v) creating one hour steps. In the traditional GA, mutation is done by replacing a parent gene with a random value. The author believes that this approach is not optimal suggesting an alteration to the formula by taking the previous value and adding a value dependant on the gene's position multiplied by a random value and a constant. This change in the approach makes it so the mutated child will be closer to the parent, making convergence faster. The fitness function used is a Monte Carlo approximation of the cumulative detection probability which calculates the efficiency of a path. This method showed better results than the traditional GA as it had faster convergence to the optimal value and was also faster.

2.2.7 Learning-based approaches

There are three main types of learning based approaches, Supervised, Unsupervised and Reinforcement learning [30].

2.2.7.1 Supervised Learning

The Supervised Learning methods, like most ML approaches, are used to predict a value based on a trained model. To train this model a labelled and classified dataset is needed. The model is then trained based on the provided data and is used as necessary. The most commonly used Supervised Learning methods are the artificial neural networks, and the Support Vector Machines (SVM), however, methods can be used like linear regressions, randomforest, logistic regressions, convolutional neural networks, recurrent neural networks, K-nearest neighbour, and Naïve Bayes [30].

2.2.7.2 Unsupervised Learning

The Unsupervised Learning methods function similarly to the Supervised Learning methods, however, there is no labelling or classification of the datasets. The models are trained to find clusters or patterns in the data [31]. Due to this difference the dataset usually needs to be larger than the one used in Supervised Learning methods [30].

2.2.7.3 Reinforcement Learning

The Reinforcement Learning methods, much like the unsupervised ones, do not get a labelled or classified dataset to be trained on. Instead the model is trained by trial and error, where the model is rewarded or punished based on the arrived state [32].

Examples of the usage of Learning-based approaches can be found in [33] where the author used a Reinforcement Learning approach to generate actions the robot should take to reach the desired configuration. Another example can be found in [34] where the author proposed a new method combining Convolutional Neural Networks and the RRT* algorithm, and concluded that the new method outperformed the conventional RRT* algorithm in the same scenario.

2.3 Motion Control

With a path calculated, the next step is to make the robot follow it, for this we use a robot controller or path tracker. In this section, the most common path tracking controllers will be explored, and these controllers are the Pure Pursuit Controller (PP), the Model Predictive Controller (MPC), and the Dynamic Windows Approach (DWA) [35].

2.3.1 Pure Pursuit

The algorithm works by calculating an arc between the robot's nearest waypoint and the one that is at least a lookahead distance L away [35]. Assuming the Path as a set of waypoints $P = p_0, p_1, p_2, ..., p_n$, the point nearest to the robot is chosen as p_r and the lookahead point p_l is chosen as follows:

$$dist(p_i) = \sqrt{(x_r - x_i)^2 + (y_r - y_i)^2} \tag{2.6}$$

$$p_l = p_i \in \mathcal{P}_t, \quad \begin{cases} dist(p_{i-1}) < L \\ dist(p_i) \geq L \end{cases} \tag{2.7}$$

with a chosen p_l it is now possible to calculate the curvature of the arc that will be followed by the robot:

$$\kappa = \frac{2y_l'}{L^2} \tag{2.8}$$

where y_l' is the lookahead's point lateral coordinate and L the desired distance between the robot's point and p_l. This algorithm assumes a constant velocity and lookahead distances and, therefore, to improve performance in more complex scenarios, a new version of this controller was created, the Adaptive Pure Pursuit, where the lookahead distance is proportional to the current velocity:

$$L_t = v_t \cdot l_t \tag{2.9}$$

where L_t is the lookahead distance at time t, v_t is the current velocity, and l_t is the lookahead gain.

2.3.2 Model Predictive Controller

The MPC is a control method that uses the model of the robot system to optimize a cost function. The controller works by, given an Horizon N, predicting the future states of the robot and then minimizing the cost function to obtain the optimal control inputs for the states in the horizon. The controller keeps iterating this process, predicting the future states and optimizing control action until the goal configuration is reached.

The cost function will vary from case to case but in a path tracking problem can be defined as [36]:

$$J(\Delta \mathbf{U}, \mathbf{e}) = \frac{1}{2} \left\{ \sum_{i=k+1}^{k+N_p} \|\mathbf{e}(t_i)\|^2_{\mathbf{Q}} + \sum_{i=k+1}^{k+N_c} \|\Delta \mathbf{u}_e(t_i)\|^2_{\mathbf{R}} \right\} \tag{2.10}$$

where $e(t_i)$ is the error between the reference point and the robot's current position, $\Delta u_e(t_i)$ is the increment of the control input, the N_p and N_c are the prediction and control horizons, and Q and R are the weighting matrices for the error and control input increment, respectively. Additionally, the control action can be constrained by the robot's physical limitations, like maximum and minimum velocity, acceleration, and jerk.

2.3.3 Dynamic Windows Approach

The DWA works by, like in the PP, creating arcs the robot will follow. Firstly, it generates circular trajectories with different pairs of velocities and angular velocities. Then it chooses the only pairs which allows the robot to stop before reaching an obstacle. Finally, it applies the dynamic window where it eliminates the pairs in which the required velocities aren't reachable in the chosen time period. After eliminating all the non usable pairs, the pair with the smallest value when the cost function is applied is chosen as the control action [37]. The cost function can be defined as:

$$J(v, w) = \alpha \cdot \text{heading}(v, w) + \beta \cdot \text{distance} + \gamma \cdot v \tag{2.11}$$

where α, β, and γ are the weighting factors, the heading is cost of the difference between where the robot is facing and the desired node direction, the distance is cost of the distance between the robot and the closest obstacle, and v is the robot's velocity. This process is repeated between nodes until the goal configuration is reached.

2.4 Localization and Mapping

In order to obtain a path for the robot to follow, it is imperative that the robot knows where its initial and goal configurations are, and also the position of the obstacles around the world. To achieve this two things need to happen, Mapping, to get a representation of the world, and Localization, to know where the robot is relative to the obstacles and the goal configuration.

2.4.1 Localization

Localization is the process of determining the robot's position and orientation in the world. This process can be done using different methods, motion sensors that give speeds, accelerations and orientations, Vision sensors that can get information about the robotics surroundings, however, its not possible to use these sensors alone as they can be noisy [38] and, therefore, Kalman Filters are used to fuse the information from the different sensors and get a more accurate position. This method is called Odometry.

Odometry alone cannot be used to get the robot's position as it is prone to drift over time [39], and therefore, other methods are used to correct the accumulated error. One of the most common methods is the Adaptive Monte Carlo Localization (AMCL). This method uses a particle filter to achieve a more accurate position, however,it requires a considerable amount of computational power and memory, as accuracy improves with particle count [40]. To solve this, both Odometry and AMCL are used together, where the Odometry is used to get a continuous estimation of the robot's position and is then, periodically, corrected by the AMCL which uses a map of the world to get a more accurate position.

2.4.2 Mapping

Mapping is the process of creating a 2D representation of the world and can be achieved using algorithms like the Simultaneous Localization and Mapping (SLAM), where both the map and the robot's positions are estimated at the same time, making it a very complex problem [41]. There are a feew different solutions to this problem, the probabilistic approaches, such as the Kalman Filter, the Particle Filter and Information filter [41], that are created from the Baye's Rule. SLAM is a very popular topics in robotics and there has been improvements to the traditional approaches like the more recent Visual-SLAM or VSLAM which also use images to facilitate obstacle detetction and tracking [42].

2.5 Related Work

Since the system that this thesis will be focusing on is a tractor-trailer system, in this section there will be a brief explanation of the system and some ways its been approached.

2.5.1 System Description

The tractor-trailer system is a system where a self-propelled vehicle, the tractor, pulls a non-propelled vehicle, the trailer. This system has been modelled in many papers and generally takes the following form [43, 44]:

$$\dot{x} = v\cos(\theta_0) \tag{2.12}$$

$$\dot{y} = v\sin(\theta_0) \tag{2.13}$$

$$\dot{\theta}_0 = \frac{v}{WB} tan(\phi) \tag{2.14}$$

$$\dot{\theta}_1 = \frac{v}{RTR} sen(\theta_0 - \theta_1) \tag{2.15}$$

where x and y are the hitch position, v is the robot's velocity, θ_0 is the tractor's orientation, θ_1 is the orientation of the trailer, ϕ is the steering angle, and WB and RTR are the are the wheel base distance (distance between axles) and the distance between the hitch and the trailer's rear axle, respectively.

2.5.2 Applications

Some applications of the tractor-trailer system can be found in [43], where the author proposed a Voronoi Hybrid A* for the path-planner and two pure-pursuit controllers as path trackers. The author concluded that the traditional approaches were not feasible due to generating paths that wouldn't account for the trailer's non-holonomic constraints. Therefore, a different version of the A* was used. The Hybrid A* is much like the traditional one, however, instead of only being able to move to the nodes or grid cells, the nodes generated in the Hybrid A* contain the robots dynamics $(x, y, \theta_0, \theta_1, d)$ where d is the direction of motion of said node (forward or reverse), and can reach anywhere in the continuous state space. The cost of each node is calculated penalizes the change of direction, backwards movement, steering input, change of steering input, jack-knifing and path length. Since these nodes have discrete values, it is not possible to always reach the desired nodes, therefore, the author used a Dubin's curve to connect waypoints generated by a traditional A* search on the Voronoi Graph. This approach was compared to the Hybrid A* without the Voronoi Graphs and the Voronoi Hybrid A* was able to create slightly shorter paths a thousand times faster. With a generated path, the author used two pure pursuit controllers that switched depending on the direction d of a node. This approach to the tractor-trailer problem was able to manoeuvre in tight indoor environments which is a necessary feature in the context of this thesis since greenhouse and farm corridors are typically narrow. This approach is also successful because it takes a relatively short time to generate a smooth path, however, it does not include a local planner for dynamic environments which may be crucial in a real-world scenario where workers or other robots may be present.

Another application of the tractor-trailer system can be found in [45] where the author created a path tracker using an mpc controller. The systems model is slightly different from the previous as the author used both the translational velocity and the angular velocity as control inputs $u^T = [v, w]$. The system's dynamics become the following:

$$\dot{x_1} = v\cos(\theta_0) \tag{2.16}$$

$$\dot{y_1} = v\sin(\theta_0) \tag{2.17}$$

$$\dot{\theta_0} = w \tag{2.18}$$

$$\dot{\theta_1} = -v\frac{1}{l_2}sin(\theta_1 - \theta_0) + w\frac{l_1}{l_2}cos(\theta_1 - \theta_0) \tag{2.19}$$

where x_1 and y_1 are the tractors centre position, l_1 is the distance between the tractor's front axle and the hitch, and l_2 is the distance between the hitch and the trailer's rear axle.

The obstacles, walls and the robot were expressed as polygons to facilitate the collision avoidance as this approach has no path planner, only a few hand placed waypoints. With these polygons, the Farkas' lemma to create a constraint for the mpc controller by expressing the polygons as two matrices A and b to discritise the obstacles position and orientation.

The cost function created took into consideration the deviation from the goal state, the deviation from the reference translational velocity and the variation of input control:

$$J = x_e^{goal}(N)^T S x_e^{goal}(N) + (\sum_{i=0}^{N-1} x_e^{goal}(k)^T Q_1 x_e^{goal}(k) + {v_1}_e^{ref}(k)^T Q_2 {v_1}_e^{ref}(k) + \Delta\hat{u}^T(k) Q_3 \Delta\hat{u}(k))\Delta\tau \tag{2.20}$$

where x_e^{goal} is the error between the goal state and the current state, ${v_1}_e^{ref}$ is the error between the current velocity and the reference velocity, $\Delta\hat{u}$ is the control input increment, and $\Delta\tau$ is the time step. The controller was constrained by the robot's physical limitations in $u^m in$ and $u^m ax$ and the obstacle avoidance equation. This cost function is minimised in every iteration until the horizon N is reached.

This approach to path tracking was tested using a narrow environment with four waypoints that would appear one by ne as the robot reached them. The testing occurred in two phases, one being in a static environment and the other in a dynamic one. In the first one the robot was able to reach the goal while managing tight curves and counter curves. However, in the second phase, the robot encountered a problem when the dynamic obstacle was placed in front of the robot, as the robot couldn't avoid it and had to reverse to avoid a collision. This happened because the obstacle wasn't considered dynamic by the model and couldn't predict its movement.

Overall, this approach was able to reach the waypoints without collisions and is very promising, however, it does not have a path planner to place the waypoints and has to be done by hand. It starts to tackle the object avoidance issue to some successful extent but could be improved.

In [46] an exact local planner was used to solve this problem. The tractor-trailer is a non-holonomic system that can't be integrated, however, if the steering angle and velocity were constant, the system would now be integrable. The author used this concept to create rotational and translational paths for this system. Since the steering angle is constant and constrained by the vehicles dynamics it was possible to calculate the angle for each arc by using the following equation:

$$\phi = -arctan(\frac{L_1 sin(\theta_1)}{L_2}) \tag{2.21}$$

where L_1 is the distance between the hitch and the tractor's front axle, L_2 is the distance between the hitch and the trailer's rear axle, and θ_1 is the orientation of the trailer. This approach resembles a PP as it also calculates the curvature of the arc needed for a smooth path. The paths between two nodes are created by connecting two rotational paths and one translational path, this means, two arcs around the nodes and one straight line connecting them. The author mentions that this method can't deal with obstacle avoidance, however, if a global planner was integrated, this wouldn't be a problem since the nodes would be place in a way that connections would be collision free. The testing revealed that this approach could generate a feasible and smooth path in under two seconds, which for the time (1995) was quite a fast time. This approach wouldn't suffice for this thesis' problem as a whole, however could be used as a path tracker instead of a path planner.

2.6 State of the art Summary

To summarize, the tractor-trailer pesticide spraying robot proposed in this dissertation could be of help in the agriculture industry, and, to develop this project there are several approaches to be considered. When it comes to path planning, the Hybrid A* algorithm is a good choice as its nodes take into account the robot's dynamics, this algorithm could be used in combination with the Voronoi Graphs to create a faster path, like in [43]. To control the robot, the most used option would be the MPC, as it can predict future states and optimize the tracking problem, however, this approach could be overkill if the path is already calculated. The next best option would be the PP, as it is simple and can reduce the tractor-trailer problem into just a car problem by having steering and velocity as constant values. For a local planner, in case a dynamic object appears in front of the robot, the MPC could now be used to reach the next node in the path, as it could take too much time for a complete replan of the path.

This project is a very important one, as there isn't a lot of documentation on solutions for this problem, and especially in the context of Smart Agriculture.

3 Architecture

In this chapter there will be a presentation of the architecture of the proposed solution as well as some insights into its implementation.

3.1 System Overview

The proposed solution is designed to address the complexities of navigating a tractor-trailer system in narrow environments, such as agricultural fields. This system integrates advanced path planning and control strategies to ensure safe practices, with collision avoidance, and effective navigation, tending to the specific needs of this type of system.

The system is divided into two main components: a self-propelled tractor and a non-propelled trailer. The tractor is an Agilex Scout V2 which is a four-wheeled differential driven robot that serves as the navigation an propulsion unit. The trailer, designed for modularity can have its own independent functionalities allowing for a simpler maintenance and adaptability to different tasks. This modullarity allows for the independant development of the tractor's navigation and control systems without the need of knowing the trailer's payload or functionalities, with perhaps the exception of the trailer hindering the sensor's view or the trailer's weight exceeding the tractor's capabilities.

The solution proposed in this work is divided into four main modules:

- **Environment**: This module is responsible for the representation of the real world, or actually be the real world. This is where the robot will operate.

- **Sensor perception**: This module is responsible for gathering data from the environment and converting it into usable information like localization and estimation.

- **Path planning**: This module is responsible for generating a feasible path from the current position to the desired goal.

- **Control**: This module is responsible for tracking the generated path and ensuring the tractor-trailer system follows it safely without collisions.

From the modules mentioned above, the ones that are implemented in this work are mainly the path planning and control modules, however, in order to test and develop the system, the environment and sensors were simulated.

The overall functioning of the system is as follows: the tractor is at a point in the environment, simulation or real world, the user then requests a movement to a specific position, the path planning module will then generate a path from the current position, estimated with sensor gathered data, to the desired goal. The path is then forwarded to the controller which will then track it by computing the necessary angular and linear velocities while ensuring the physical constraints of the tractor-trailer system are respected.

3.2 Hardware

The tractor used is an Agilex Scout which is a four wheeled differential driven robot, equipped with a NVIDIA Jetson Agx Xavier as the main computer, a LIDAR and wheel decoders. the NVIDIA Jetson Agx Xavier is a powerful computer capable of running complex algorithms in real time, and is used to run the whole system, from path planning to control and sensor perception. The Lidar and decoders are used to gather data for localization and mapping. This robot can be teleoperated using a controller, or can be programmed to do whatever autonomous task is requiered since all its instruments are accessible in the Jetson Agx Xavier. The tractor is shown in the figure below:

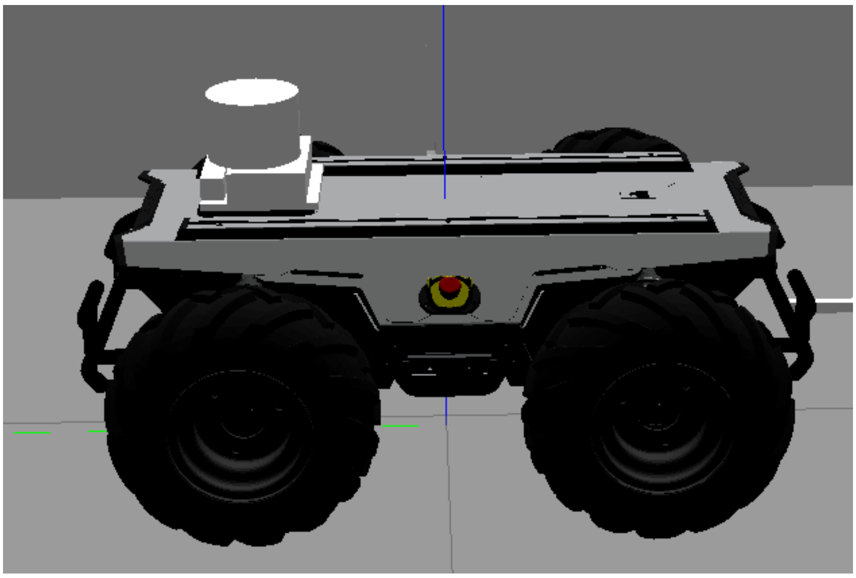

Figure 3.1: Simulated tractor in Gazebo.

There's no representation of the jetson in the simulation since the program is running on the same machine as the simuation. However, it is possible to notice the Lidar rack on top of the other instruments. Once again, this is but a representation for simulation purposes, as the real tractor has a similar configuration, shown in figure 3.2.

Figure 3.2: Real tractor.

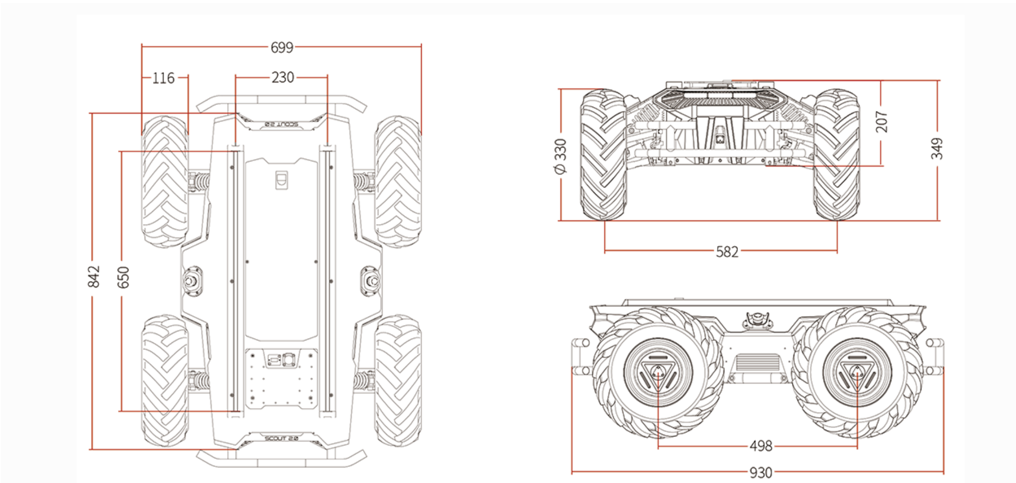

Figure 3.3: Agilex Scout dimensions [47].

The figures 3.3 and 3.2 show the robot, its dimensions, and also the the additional rack, at the top, where the computer, Lidar and all other required electronics are mounted.

Additionally, the tractor is pulling a trailer, which is equipped with a spray system, however for the sake of simplicity, to test the navigation and control algorithms, the tests were be done with the following trailer:

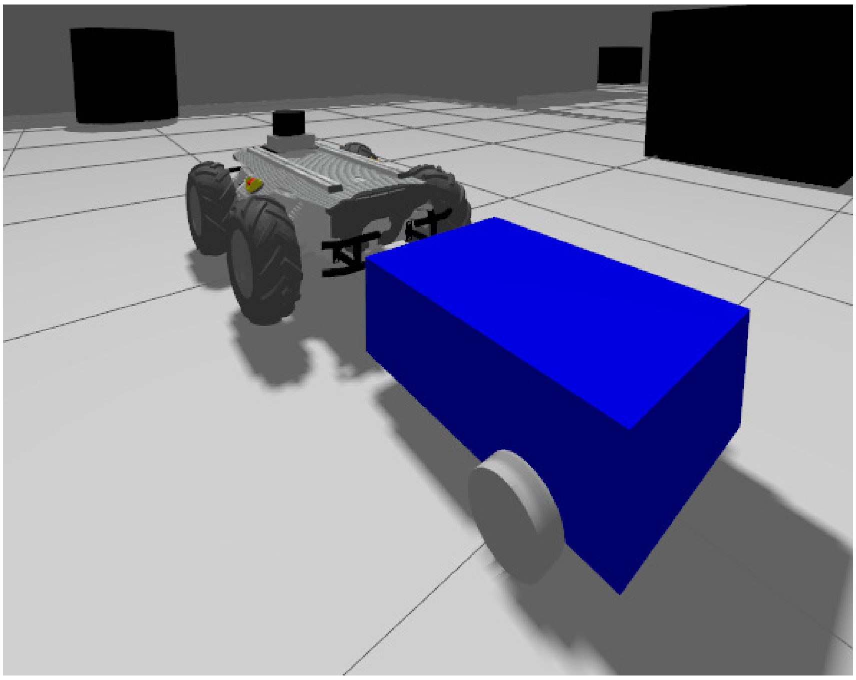

Figure 3.4: Tractor-trailer system simulated in Gazebo Classic.

The trailer shown in figure 3.4, is a representation of a generic trailer, not having any specific functions other than following the systems dynamics, which is the main focus of this work. One particular aspect of this robot is that there's no sensor for trailer position and, therefore, it is estimated in real time using the system's dynamics shown in 2.5.1.

3.3 Planner

The planner is a Voronoi Hybrid A* mixture which takes the ability to generate feasible paths from the Hybrid A* and the added velocity of having a pre-computed Voronoi graph to expand from.

The following flowchart shows the logic of the planner, starting with the request for movement, assigning a goal target. Then, the planner will preform an A* search on the pre computed Voronoi Graph, starting from the current robot positon, to the goal pose. The A* search will return a path which consists of a series of subgoals which will later on be used to speed up the path creation process. Having the subgoals, the next step is to try and find a feasible path from the current position to the subgoal nearest to the goal pose, in this case, since it is the first iteration, the actual goal. If the path is feasible, then the planner will simply return said path, otherwise the planner will try to do the same but with the next subgoal, until it either finds a feasible path or exhausts all the subgoals. If it finds a path, it will repeat the previous process, but instead of using the current position as the starting configuration, it will use the subgoal which a path was able to be computed to. If it exhausts all the subgoals, then it will try and compute a new node, a Hybrid A* node, which consists of a segment, starting from the current algorithmic position (can be the start point or a subgoal depending if a path was found previously), to any point that is reachable by the tractor at a distance d defined by the user, and then assign them as the current algorithmic position to run the Dubins loop again.

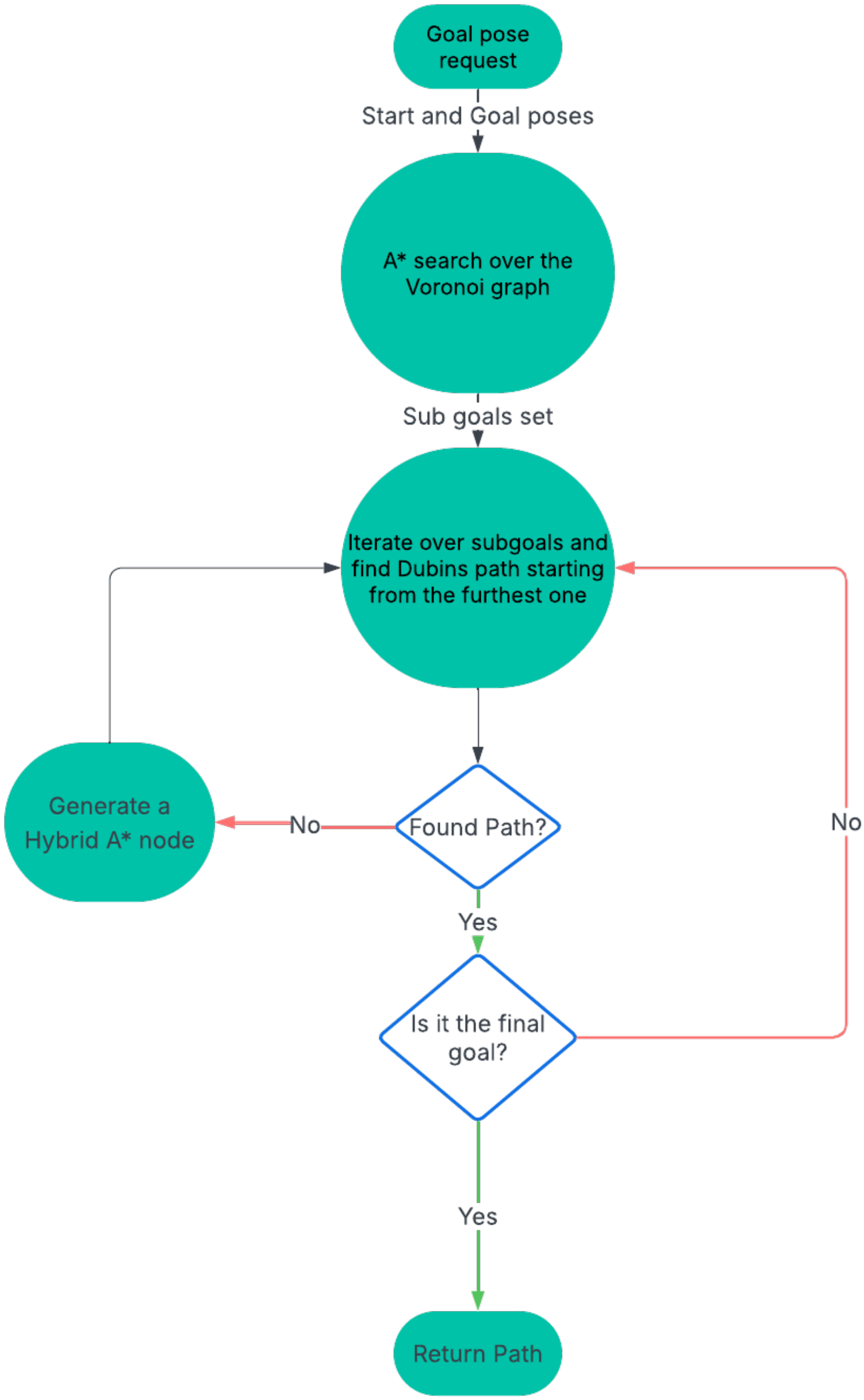

Figure 3.5: Planner logic flowchart

The algorithm 1 shows the detailed pseudocode for the planner. Starting with the

Algorithm 1 Planner Pseudocode

```
function GET_PATH(start, goal)
    subgoals ← get_subgoals(start, goal, voronoi_graph)
    current ← start
    path ← None
    for all subgoal in subgoals do
        path ← get_feasible_path(path, current, subgoal)
        if path ≠ None then
            if subgoal  goal then
                return path
            else
                current ← subgoal
            end if
        end if
    end for
    while True do
        new_node ← expand_hybrid_node(start)
        if check_not_feasible(new_node) then
            return None
        end if
        path ← attatch_node_to_path(path,new_node, goal)
        current ← new_node
    end while
end function
```

subgoals generation, to the looping through them and trying to find a path, to, if needed, the expansion of the hybrid A* nodes.

Algorithm 2 shows the pseudocode for the subgoal generation, which is done by performing a regular A* search on the pre-computed Voronoi graph. It starts by examining the neighbours of the start node and adding them to the open list, it then goes through the open list and preforms the same operation until the goal is found or the open list is empty.

The Dubins path creation is more mathmatically complex to represent simply in pseudocode. It is a calculation of forwards segments that can be of 6 different types, Left Straight Left (LSL), Left Straight Right (LSR), Right Straight Left (RSL), Right Straight Right (RSR), Right Left Right (RLR) and Left Right Left (LRL). These paths are the shortest possible curves connecting two configurations for a non-holonomic vehicle with a fixed minimum radius. The process is devided into six main steps, normalizing the problem, byt calculating the distances between start and goal configurations and also the orientation, the start heading from start to goal and the goal heading relative to the line connecting the start and goal. the next step is to compute the six paths, one of each type and check their feasibility, then, the feasible paths are sorted by length and the shortest is the chosen one. Finally, the path is then segmented into short segments of (x,y and yaw) and then returned.

Algorithm 2 Get_subgoals Pseudocode

```
function GET_SUBGOALS(start, goal, voronoi_graph)
    subgoals ← []
    closed_list ← []
    open_list ← [start]
    found ← False
    while open_list.not_empty() and not found do
        current ← open_list.pop()
        closed_list.append(current)
        neighbours ← current.neighbours
        for all neighbour in neighbours do
            if neighbour not in closed_list then
                if neighbour in open_list then
                    if current.f < open_list(neighbour).f then
                        continue
                    end if
                end if
                current.parent ← current
                current.f ← compute_fcost(current, neighbour, goal)
                open_list.append(neighbour)
            end if
            if neighbour is goal then .parent ← current
                found ← True break
            end if
        end for
    end while
    if found then
        current ← goal
        while current is not start do
            subgoals.append(current)
            current ← current.parent
        end while
    end if
    return subgoals
end function
```

The algorithm 3 shows the pseudocode for the hybrid A* node expansion. It first iterates through all the parameterised directions (steering angles), and creates segments of the tractor's movement, forwards and backwards. All these segments are checked for feasibility, which is done by verifying if the trailer is not colliding with the tractor or with any obstacles in the environment. The feasible segments are then added to open list to be used in the next iteration of the planner and verify if a Dubins path is feasible from that new position.

In figure 3.6, the rectangle represents the tractor and the arrows the segments created, forwards and backwards, representing the recovery process. The amount of nodes, length of the segments and the maximum radius can be parameterised by the user, allowing for

Algorithm 3 Expand_hybrid_node Pseudocode

```
function EXPAND_HYBRID_NODE(node, open_list, directions, node_length)
    for all sign in {-1, 1} do
        for all d in directions do
            segment ← [node.x, node.y, node.yaw]
            for i do in node_length
                new_node ← new_node()
                new_node.yaw ← segment.last.yaw + d
                new_node.x ← sign * segment.last.x + i * cos(new_node.yaw)
                new_node.y ← sign * segment.last.y + i * sin(new_node.yaw)
                new_node.parent ← segment.last.parent
                new_node.f ← compute_fcost(new_node, goal)
                segment.append(new_node)
            end for
            if check_feasibility(new_node) then
                open_list.append(new_node)
            end if
        end for
    end for
end function
```

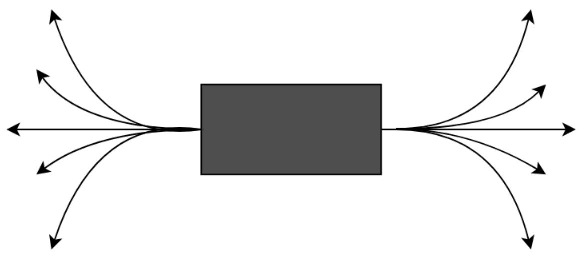

Figure 3.6: Hybrid A* node expansion.

more diversity of operatioal environments.

3.4 Controller

A path is nothing if it cannot be followed so, a controller capable of attending to this system's needs is required. This controller will receive as input the generated path by the planner and will try to track it. This is achieved in two steps, first, having the current positions in mind, it will choose the next subgoal in the path. Since the controller is a PP controller, the subgoal is chosen depending on the lookahead distance, as explained in 2.3.1. The next step is to compute the angular velocity needed for the robot to reach said subgoal. For this, two controllers were used:

- **Forwards Controller**: Used when the tractor is moving forwards.
- **Backwards Controller**: Used when the tractor is moving backwards.

The choice of controller depends on the orientation of the tractor and direction to the next subgoal. The need for two controllers arises from the fact that when moving backwards, the movement needs to be optimised for the trailer's position and orientation, while when moving forwards, the movement needs only to be optimised for the tractor's movement as the trailer positions are move predictable when moving forwards.

The forwards controller is defined by the following equations:

$$\omega = \tan^{-1}\left(\frac{2 * WB * sin(\epsilon_{\theta_0})}{l_f}\right) \tag{3.1}$$

where WB is the wheelbase of the tractor, l_f is the lookahead distance and ϵ_{θ_0} is the error in the angle between the tractor and the next goal.

$$\epsilon_{\theta_0} = \tan^{-1}\left(\frac{t_y - y_r}{t_x - x_r}\right) - \theta_0 \tag{3.2}$$

The t_x and t_y are target point coordinates and x_r and y_r are the current tractor center coordinates. As explained previously, this controller only cares about the tractor's configuration, which makes having a feasible path all the more important.

The backwards controller, on the other hand, is defined by the following equations:

$$\omega = V\left(-k(\theta_1 - \alpha^*) - \frac{sin\theta_1}{RTR}\right) \tag{3.3}$$

where RTR is the distance between the hitch joint and the trailer's real axle, k is a constant, θ_1 is the angle between the tractor and the trailer, V is the linear velocity of the tractor and

$$\alpha^* = tan^{-1}\left(\frac{2 * RTR * sin(\epsilon_{\theta_1})}{l_f}\right) \tag{3.4}$$

with

$$\epsilon_{\theta_1} = \tan^{-1}\left(\frac{t_y - y_t}{t_x - x_t}\right) - \theta_t \tag{3.5}$$

where t_x and t_y are the target point coordinates and x_t and y_t are the current trailer real axle coordinates and θ_t is the orientation of the trailer. It is observed that these equations are similar to the forwards controller, however, they differ in the fact that the goal pose is defined by the trailer's position. Even though this controller is taking the trailer's constraints into account, due to the inate nature of the tractor-trailer system being non-holonomic, the path planner is still the one responsible for maintaining the feasibility of the movement.

3.5 Collision Avoidance

The collision avoidance aspect of the controller is of the utmost importance in real world scenarios when dealing with possible workers in the field. To achieve this safety requirement, the controller is equipped with a collision detection algorithm which will check for obstacles in the way of the tractor-trailer system and remain still until the obstacle is no longer in the way or another path update has been sent.

The algorithm works by calculating the angle between the tractor and the chosen subgoal in the path, then, it will check if there are any obstacles in the rectangle created along the line connecting the tractor and subgoal. If true, the controller will give signal that a collision is imminent and commands will be to stop and wait.

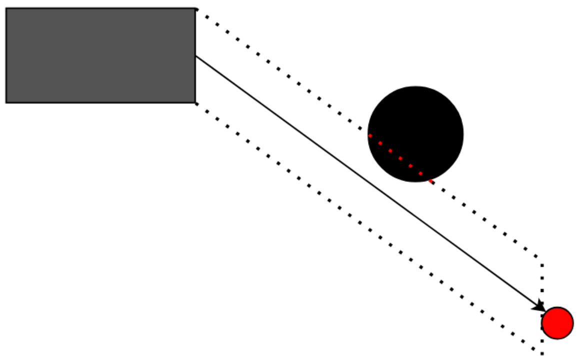

Figure 3.7: Representation of the collsion detection module. In red the goal pose, in grey the tractor, the black circle is the obstacle and then the doted rectangle the expanded rectangle used to check for collisions.

4

Implementation

In this chapter, the implementation of the different modules and components mentioned in the previous chapter will be described, as well as their process of creation and integration into the full system.

4.1 Software

Since this system is composed of several different components that need to run independantly, the software used will need to be robust enough to account for each component's requirements, and for this task there the Robot Operating System (ROS) framework is used. There are two versions of this framework, ROS and Robot Operating System 2 (ROS2), the latter being the most recent version, which is the one to be used in this project.

4.1.1 ROS2

The ROS2 framework is a set of software libraries and tools that help developers create robotic applications. It is provides a set of solutions to various problems, such as parallelism, communication, modularity, and hardware abstraction.

It works by creating a network of nodes (python or c++ programs) that can communicate with each other using a publish-subscribe model and a service-client model. The former is a broadcasting model where a node can publish messages to a specific topic, and any other program can subscribe to that same topic and receive those messages and usse them using callbacks. The latter is a request-response model where a specific node is the server and other nodes can call it to request a specific service, which helps with modularity as each node can have its own specific task and be developed independently. When it comes to the hardware abstraction, since the software is wildly used in the robotics community it already has a lot of packages and libraries that can be used to interpret data form sensors, control actuators, and even simulate the robot in a virtual environment.

4.1.2 NAV2

The Navigation 2 (NAV2) stack is a set of packages that provide a framework for autonomous navigation. It is built on top of ROS2 and provides a set of tools for coordinating the different components of the navigation system.

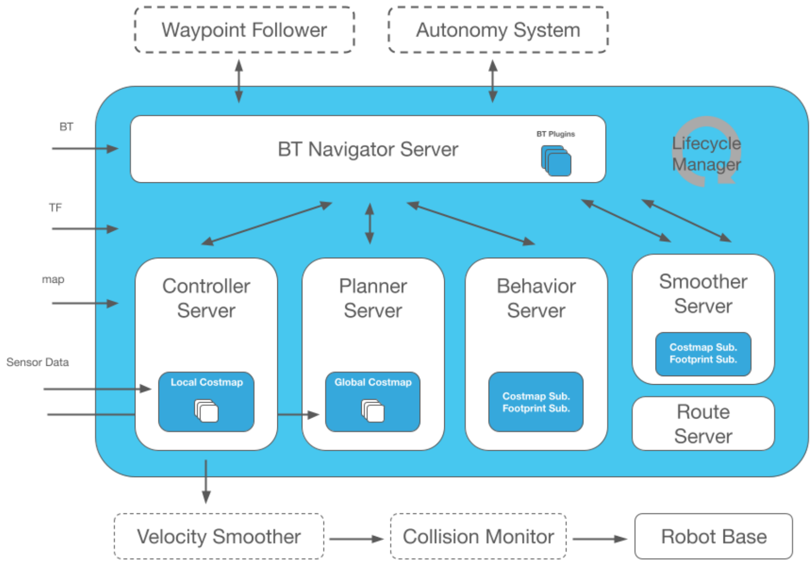

Figure 4.1: NAV2 stack architecture [48]

In figure 4.1 we can see the different components of the NAV2 stack required for robust autonomous behaviour. The main components are thr Controller server, which is responsible for, using a controller plugin, controlling the robot's movement, the Planner server, which is responsible for, using a custom plugin, planning the robot's path, the Behaviour server which, given a custom behaviour tree, is responsible for selecting the next action to be taken by the robot, the BT Navigator server, which coordinates all the other servers and the Lifecycle manager which controls the lifecycle of the different modules mentioned previously. The stack also includes a variety of already defined plugins to be used for localization, path planning and control, however, in this work, a custom planner, controller, behaviour tree and also state publisher are created to suit the specific needs of the robot.

4.2 Environment

For development purposes, a virtual environment is used to develop the navigation software and test it without the need for a physical robot. This environment can be created using Gazebo, a robotics simulator that is compatible with ROS2, and it allows the user to create a world in xml format, specifying the different physical properties of the environment as well as the structures and objects inside it.

Since the robot will be operating in an agricultural environment, the world created for the simulation would have to reflect that, so a simple world was created with five corridors with traffic bariers on each side (to simulate trees or crops) and some additional obstacles to make for a more complex environment.

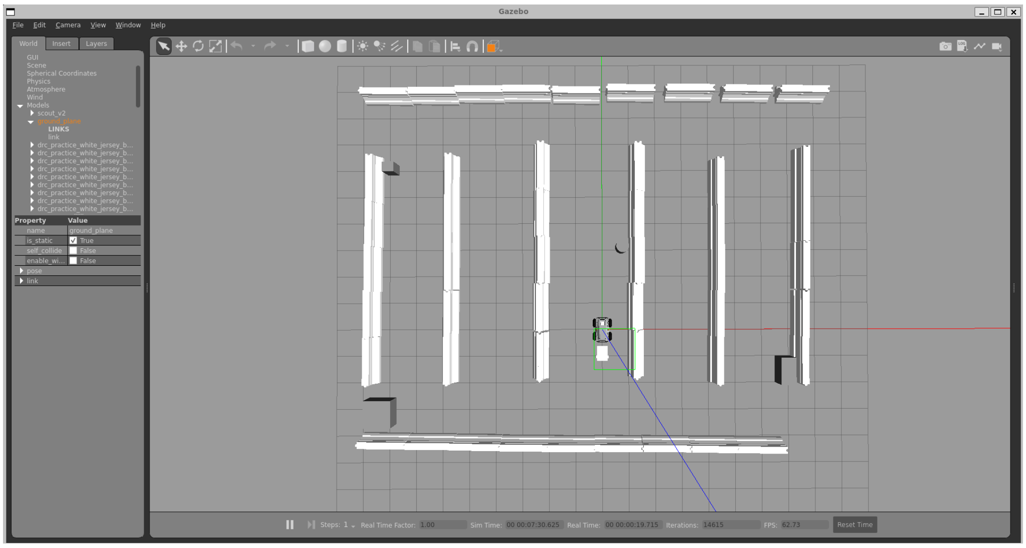

Figure 4.2: Gazebo world used for simulation.

To visualize the robot in this environment as well as the different data that is deemed relevant a software called Rviz2 is used. This software allows the user to select the various ROS2 topics that are being published by the running nodes and visualize them in a 3D environment, which is very useful for debugging and testing purposes.

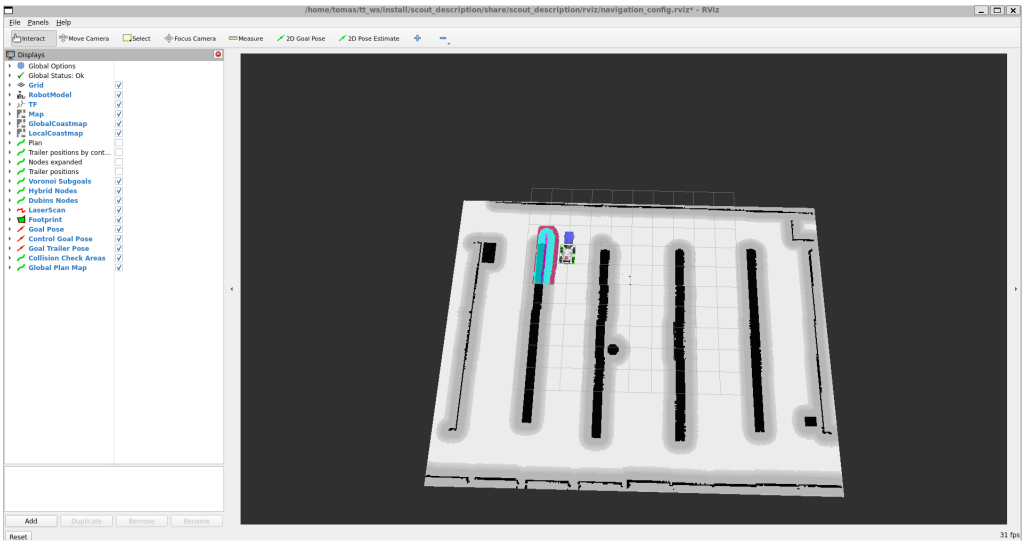

Figure 4.3: Rviz2 visualization of the robot in the simulated environment.

The figure 4.3 shows, at the left, the different data provided by the developed system and, above it the navigation tools like estimatin the robot's initial position and goal generation. These last two tools can be used manually in Rviz2 or be published by a custom node if the user requires it.

In a first iteration, the prototype for the robot was also used in a real environment, a private room with a few obstacles and after mapping it could be represented by the following figure 4.4.

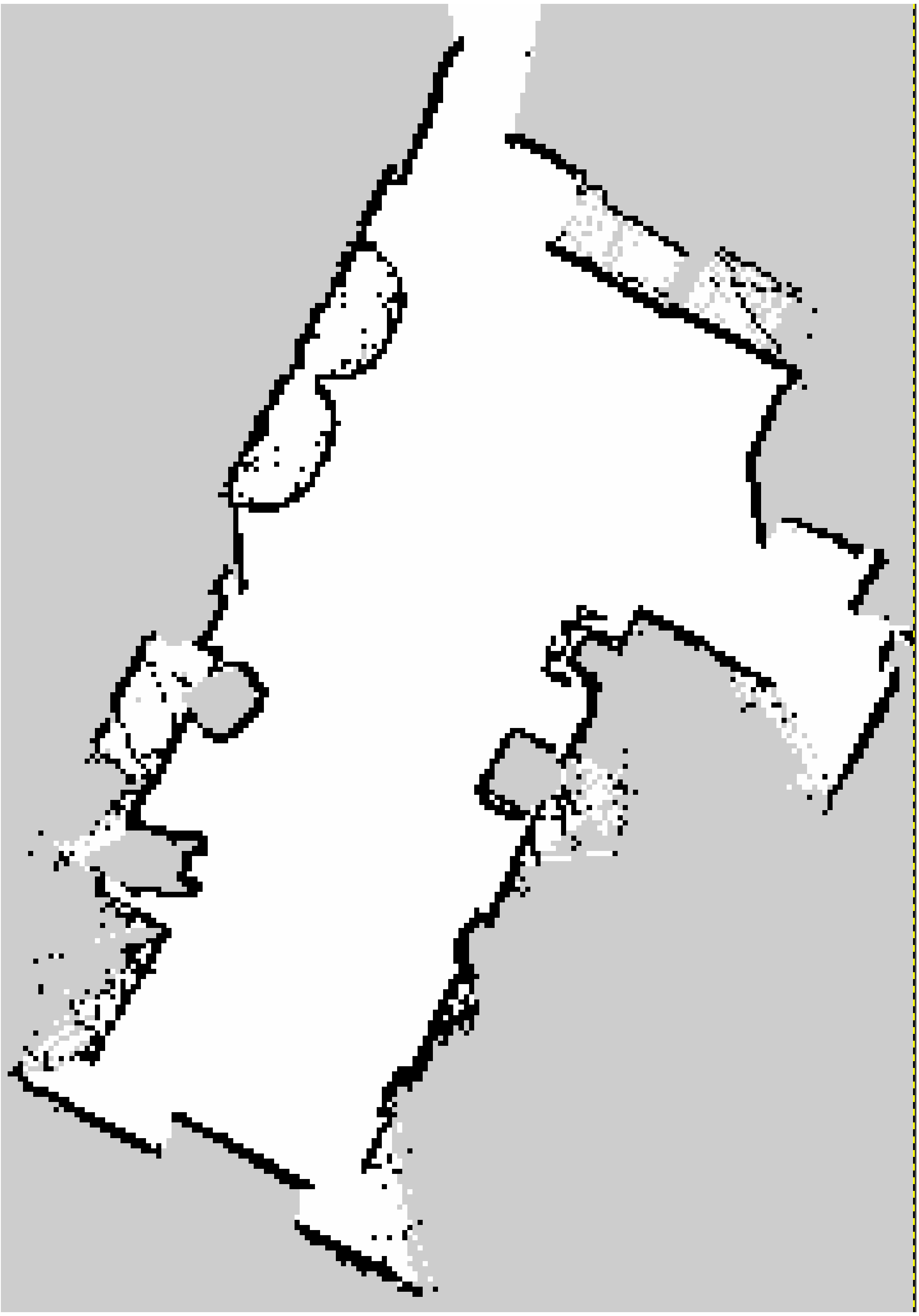

Figure 4.4: First environment used for testing the robot.

4.3 Planning algorithm

As explained in the previous chapter 3.3, the planner is devided into three main components, the Voronoi Graph, the Dubins Path and the Hybrid A* recovery algorithm.

4.3.1 Voronoi Graph

The main function of the Voronoi Graph is to create a set of points in space that can be searched for a simple path to the goal using an A* search. Due to the nature of calculating a Voronoi Graph, it can be very computationally expensive, so the algorithm is ran offline, when the environment is mapped, and the resulting graph is saved to a file for the online planner to use. To create this graph, a Ros2 python node was created that executes a script with the previous behaviour.

The algorithm first works by creating black and white image containing the edges of the obstacles and grouping them into a single column of points.

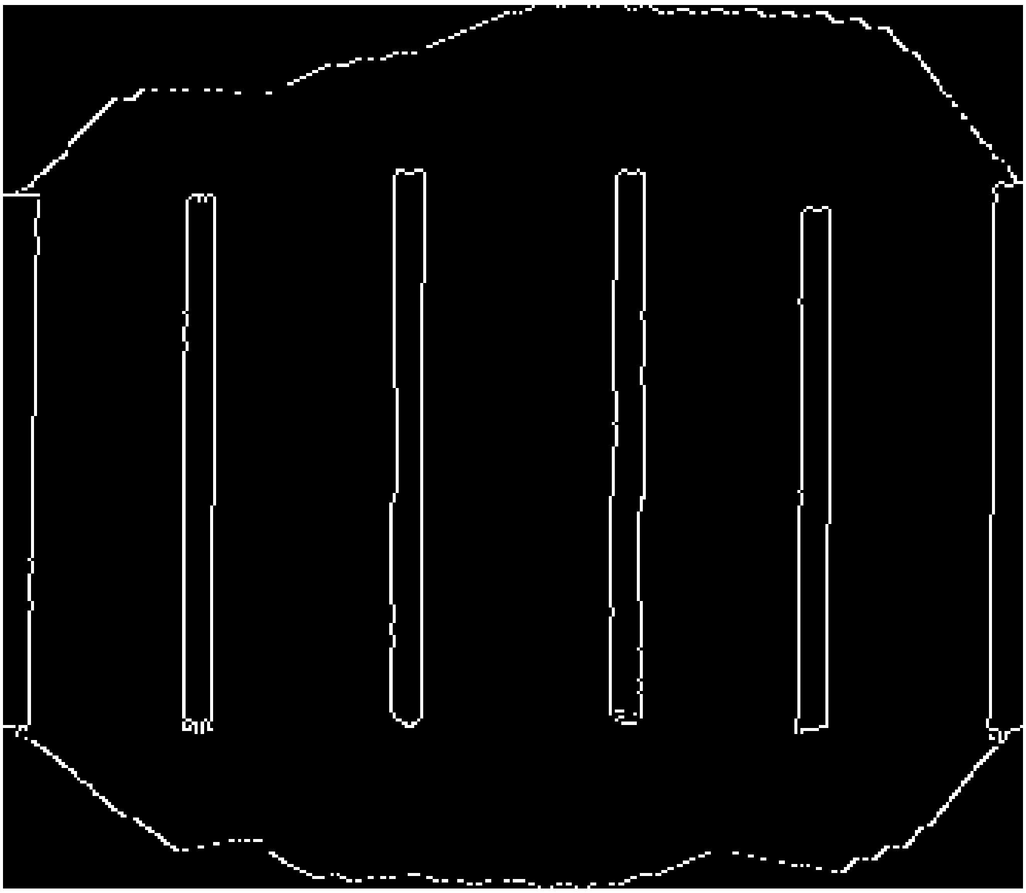

Figure 4.5: Image with the obstacle edges.

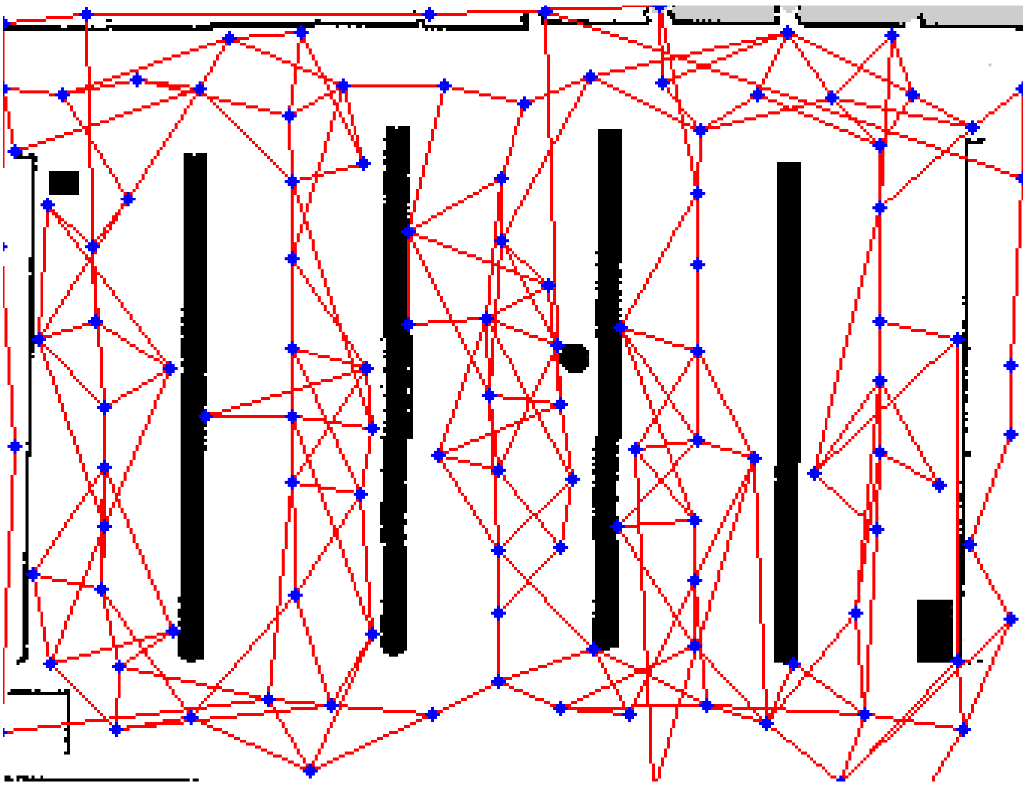

Figure 4.6: Voronoi Graph created for the environment in figure 4.2.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. url: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[2] A. Hafeez et al. "Implementation of drone technology for farm monitoring & pesticide spraying: A review". In: *Information Processing in Agriculture* 10.2 (2023), pp. 192–203. issn: 2214-3173. doi: https://doi.org/10.1016/j.inpa.2022.02.002 (cit. on p. 1).

[3] S. Qazi, B. A. Khawaja, and Q. U. Farooq. "IoT-Equipped and AI-Enabled Next Generation Smart Agriculture: A Critical Review, Current Challenges and Future Trends". In: *IEEE Access* 10 (2022), pp. 21219–21235. doi: 10.1109/ACCESS.2022.3152544 (cit. on p. 3).

[4] L. F. P. Oliveira, A. P. Moreira, and M. F. Silva. "Advances in Agriculture Robotics: A State-of-the-Art Review and Challenges Ahead". In: *Robotics* 10.2 (2021). issn: 2218-6581. doi: 10.3390/robotics10020052. url: https://www.mdpi.com/2218-6581/10/2/52 (cit. on p. 3).

[5] D. F. Yépez Ponce et al. "Mobile robotics in smart farming: current trends and applications". In: *Frontiers in Artificial Intelligence* 6 (2023-08). doi: 10.3389/frai.2023.1213330 (cit. on pp. 3, 4).

[6] V. Moysiadis et al. "Mobile Robotics in Agricultural Operations: A Narrative Review on Planning Aspects". In: *Applied Sciences* 10.10 (2020). issn: 2076-3417. doi: 10.3390/app10103453. url: https://www.mdpi.com/2076-3417/10/10/3453 (cit. on p. 4).

[7] K. G. Fue et al. "An Extensive Review of Mobile Agricultural Robotics for Field Operations: Focus on Cotton Harvesting". In: *AgriEngineering* 2.1 (2020), pp. 150–174. issn: 2624-7402. doi: 10.3390/agriengineering2010010. url: https://www.mdpi.com/2624-7402/2/1/10 (cit. on p. 4).

[8] L.-B. Chen, X.-R. Huang, and W.-H. Chen. "Design and Implementation of an Artificial Intelligence of Things-Based Autonomous Mobile Robot System for Pitaya Harvesting". In: *IEEE Sensors Journal* 23.12 (2023), pp. 13220–13235. DOI: 10.1109/JSEN.2023.3270844 (cit. on p. 4).

[9] E.-T. Baek and D.-Y. Im. "ROS-Based Unmanned Mobile Robot Platform for Agriculture". In: *Applied Sciences* 12.9 (2022). ISSN: 2076-3417. DOI: 10.3390/app12094335. URL: https://www.mdpi.com/2076-3417/12/9/4335 (cit. on p. 4).

[10] M. G. Tamizi, M. Yaghoubi, and H. Najjaran. "A review of recent trend in motion planning of industrial robots". In: *International Journal of Intelligent Robotics and Applications* 7.2 (2023), pp. 253–274. DOI: https://doi.org/10.1007/s10845-021-01867-z (cit. on p. 5).

[11] L. Liu et al. "Path planning techniques for mobile robots: Review and prospect". In: *Expert Systems with Applications* 227 (2023), p. 120254. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120254. URL: https://www.sciencedirect.com/science/article/pii/S095741742300756X (cit. on p. 5).

[12] B. Patle et al. "A review: On path planning strategies for navigation of mobile robot". In: *Defence Technology* 15.4 (2019), pp. 582–606. ISSN: 2214-9147. DOI: https://doi.org/10.1016/j.dt.2019.04.011. URL: https://www.sciencedirect.com/science/article/pii/S2214914718305130 (cit. on p. 6).

[13] S. Xie et al. "Distributed Motion Planning for Safe Autonomous Vehicle Overtaking via Artificial Potential Field". In: *IEEE Transactions on Intelligent Transportation Systems* 23.11 (2022), pp. 21531–21547. DOI: 10.1109/TITS.2022.3189741 (cit. on p. 9).

[14] C. Warren. "Global path planning using artificial potential fields". In: (1989), 316–321 vol.1. DOI: 10.1109/ROBOT.1989.100007 (cit. on p. 9).

[15] Y. Li et al. "Path planning of robot based on artificial potential field method". In: 6 (2022), pp. 91–94. DOI: 10.1109/ITOEC53115.2022.9734712 (cit. on p. 9).

[16] W. Xinyu et al. "Bidirectional Potential Guided RRT* for Motion Planning". In: *IEEE Access* 7 (2019), pp. 95046–95057. DOI: 10.1109/ACCESS.2019.2928846 (cit. on p. 9).

[17] A. A. Ravankar et al. "HPPRM: Hybrid Potential Based Probabilistic Roadmap Algorithm for Improved Dynamic Path Planning of Mobile Robots". In: *IEEE Access* 8 (2020), pp. 221743–221766. DOI: 10.1109/ACCESS.2020.3043333 (cit. on p. 10).

[18] J. Wang et al. "Efficient Robot Motion Planning Using Bidirectional-Unidirectional RRT Extend Function". In: *IEEE Transactions on Automation Science and Engineering* 19.3 (2022), pp. 1859–1868. DOI: 10.1109/TASE.2021.3130372 (cit. on p. 11).

[19] H.-y. Zhang, W.-m. Lin, and A.-x. Chen. "Path Planning for the Mobile Robot: A Review". In: *Symmetry* 10.10 (2018). ISSN: 2073-8994. DOI: 10.3390/sym10100450. URL: https://www.mdpi.com/2073-8994/10/10/450 (cit. on p. 11).

[20] L. Blasi et al. "Path Planning and Real-Time Collision Avoidance Based on the Essential Visibility Graph". In: *Applied Sciences* 10.16 (2020). ISSN: 2076-3417. DOI: 10.3390/app10165613. URL: https://www.mdpi.com/2076-3417/10/16/5613 (cit. on p. 11).

[21] L. E. Dubins. "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents". In: *American Journal of Mathematics* 79.3 (1957), pp. 497–516. ISSN: 00029327, 10806377. DOI: https://doi.org/10.2307/2372560. URL: http://www.jstor.org/stable/2372560 (visited on 2024-09-28) (cit. on p. 11).

[22] W. Lee, G.-H. Choi, and T.-w. Kim. "Visibility graph-based path-planning algorithm with quadtree representation". In: *Applied Ocean Research* 117 (2021), p. 102887. ISSN: 0141-1187. DOI: https://doi.org/10.1016/j.apor.2021.102887. URL: https://www.sciencedirect.com/science/article/pii/S0141118721003588 (cit. on p. 11).

[23] J. Wang and M. Q.-H. Meng. "Optimal Path Planning Using Generalized Voronoi Graph and Multiple Potential Functions". In: *IEEE Transactions on Industrial Electronics* 67.12 (2020), pp. 10621–10630. DOI: 10.1109/TIE.2019.2962425 (cit. on p. 12).

[24] S. Poudel, M. Y. Arafat, and S. Moh. "Bio-Inspired Optimization-Based Path Planning Algorithms in Unmanned Aerial Vehicles: A Survey". In: *Sensors* 23.6 (2023). ISSN: 1424-8220. DOI: 10.3390/s23063051. URL: https://www.mdpi.com/1424-8220/23/6/3051 (cit. on p. 13).

[25] M. Saska et al. "Robot Path Planning using Particle Swarm Optimization of Ferguson Splines". In: *2006 IEEE Conference on Emerging Technologies and Factory Automation*. 2006, pp. 833–839. DOI: 10.1109/ETFA.2006.355416 (cit. on p. 13).

[26] Q. Luo et al. "Research on path planning of mobile robot based on improved ant colony algorithm". In: *Neural Computing and Applications* 32 (2020), pp. 1555–1566. DOI: https://doi.org/10.1007/s00521-019-04172-2 (cit. on p. 14).

[27] X. Dai et al. "Mobile robot path planning based on ant colony algorithm with A* heuristic method". In: *Frontiers in neurorobotics* 13 (2019), p. 15. DOI: 10.3389/fnbot.2019.00015 (cit. on p. 14).

[28] X. Xie, Z. Tang, and J. Cai. "The multi-objective inspection path-planning in radioactive environment based on an improved ant colony optimization algorithm". In: *Progress in Nuclear Energy* 144 (2022), p. 104076. ISSN: 0149-1970. DOI: https://doi.org/10.1016/j.pnucene.2021.104076. URL: https://www.sciencedirect.com/science/article/pii/S0149197021004303 (cit. on p. 14).

[29] H. Guo et al. "Optimal search path planning for unmanned surface vehicle based on an improved genetic algorithm". In: *Computers & Electrical Engineering* 79 (2019), p. 106467. DOI: https://doi.org/10.1016/j.compeleceng.2019.106467 (cit. on p. 15).

[30] C. E. Okereke et al. "An Overview of Machine Learning Techniques in Local Path Planning for Autonomous Underwater Vehicles". In: *IEEE Access* 11 (2023), pp. 24894–24907. DOI: 10.1109/ACCESS.2023.3249966 (cit. on p. 15).

[31] S. Aggarwal and N. Kumar. "Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges". In: *Computer Communications* 149 (2020), pp. 270–299. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2019.10.014 (cit. on p. 15).

[32] S. Aradi. "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (2022), pp. 740–759. DOI: 10.1109/TITS.2020.3024655 (cit. on p. 16).

[33] S. Sombolestan, A. Rasooli, and S. Khodaygan. "Optimal path-planning for mobile robots to find a hidden target in an unknown environment based on machine learning". In: *Journal of ambient intelligence and humanized computing* 10 (2019), pp. 1841–1850. DOI: https://doi.org/10.1007/s12652-018-0777-4 (cit. on p. 16).

[34] J. Wang et al. "Neural RRT*: Learning-Based Optimal Path Planning". In: *IEEE Transactions on Automation Science and Engineering* 17.4 (2020), pp. 1748–1758. DOI: 10.1109/TASE.2020.2976560 (cit. on p. 16).

[35] S. Macenski et al. "Regulated pure pursuit for robot path tracking". In: *Autonomous Robots* 47.6 (2023), pp. 685–694. DOI: https://doi.org/10.1007/s10514-023-10097-6 (cit. on p. 16).

[36] E. Kayacan and G. Chowdhary. "Tracking error learning control for precise mobile robot path tracking in outdoor environment". In: *Journal of Intelligent & Robotic Systems* 95 (2019), pp. 975–986. DOI: https://doi.org/10.1007/s10846-018-0916-3 (cit. on p. 17).

[37] M. Kobayashi and N. Motoi. "Local Path Planning: Dynamic Window Approach With Virtual Manipulators Considering Dynamic Obstacles". In: *IEEE Access* 10 (2022), pp. 17018–17029. DOI: 10.1109/ACCESS.2022.3150036 (cit. on p. 17).

[38] I. Ullah et al. "Mobile robot localization: Current challenges and future prospective". In: *Computer Science Review* 53 (2024), p. 100651. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2024.100651. URL: https://www.sciencedirect.com/science/article/pii/S1574013724000352 (cit. on p. 18).

[39] K. S. Chong and L. Kleeman. "Accurate odometry and error modelling for a mobile robot". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. 1997, 2783–2788 vol.4. DOI: 10.1109/ROBOT.1997.606708 (cit. on p. 18).

[40] M.-A. Chung and C.-W. Lin. "An Improved Localization of Mobile Robotic System Based on AMCL Algorithm". In: *IEEE Sensors Journal* 22.1 (2022), pp. 900–908. DOI: 10.1109/JSEN.2021.3126605 (cit. on p. 18).

[41] Y. K. Tee and Y. C. Han. "Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review". In: *2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)*. 2021. DOI: 10.1109/GECOST52368.2021.9538731 (cit. on p. 18).

[42] A. Tourani et al. "Visual SLAM: What Are the Current Trends and What to Expect?" In: *Sensors* 22.23 (2022). ISSN: 1424-8220. DOI: 10.3390/s22239297 (cit. on p. 18).

[43] T. Thakkar and A. Sinha. "Motion Planning for Tractor-Trailer System". In: *2021 Seventh Indian Control Conference (ICC)*. 2021, pp. 93–98. DOI: 10.1109/ICC54714.2021.9703119 (cit. on pp. 19, 21).

[44] Z. Wang et al. "Motion Planning and Model Predictive Control for Automated Tractor-Trailer Hitching Maneuver". In: *2022 IEEE Conference on Control Technology and Applications (CCTA)*. 2022, pp. 676–682. DOI: 10.1109/CCTA49430.2022.9966181 (cit. on p. 19).

[45] N. Ito et al. "Configuration-aware Model Predictive Motion Planning in Narrow Environment for Autonomous Tractor-trailer Mobile Robot". In: *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*. 2021, pp. 1–7. DOI: 10.1109/IECON48115.2021.9589596 (cit. on p. 19).

[46] P. Svestka and J. Vleugels. "Exact motion planning for tractor-trailer robots". In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 3. 1995, 2445–2450 vol.3. DOI: 10.1109/ROBOT.1995.525626 (cit. on p. 21).

[47] *Agilex Scout dimensions*. https://static.generation-robots.com/media/datasheet-agilex-robotics-scout2.0-en.pdf. Accessed: 2025-07-29 (cit. on p. 24).

[48] *Nav2 documentation*. https://docs.nav2.org/. Accessed: 2025-08-07 (cit. on p. 33).