FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Protótipos de robots móveis com diferentes configurações de locomoção

**Gonçalo Rendeiro Brochado Garganta**

MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Supervisor: Paulo José Cerqueira Gomes da Costa

Second Supervisor: José Luís Magalhães Lima

July 31, 2023

# Resumo

Nas últimas décadas, a área da robótica evoluiu significativamente, tendo sido criadas novas e melhores soluções de robôs móveis para fins industriais, científicos, médicos entre muitos outros.

Entre estas soluções encontra-se o carro robô, que usa rodas como meio de locomoção. No entanto, existem várias opções contidas nesta solução, podendo o robô ter tipos diferentes de rodas e configurações das mesmas, sendo que cada uma oferece vantagens para uma variedade de objetivos.

Escolher uma configuração para as rodas para robôs móveis é extremamente importante para a mobilidade do robô, e o seu objetivo tem de ser considerado durante o estudo das opções.

Tendo como base protótipos existentes de veículos robóticos com uma configuração diferencial, é possível desenvolver outras configurações e estudar não só as diferenças, mas também os seus pontos fortes e fracos, os métodos de controlo que a configuração requere e as trajetórias que cada uma permite que o robô faça.

Esta análise tornará a escolha de configuração mais clara para cada cenário.

**Palavras Chave:** Robótica; Configurações de Rodas; Trajetórias

# Abstract

In the last few decades, the area of robotics has evolved immensely, creating new and improved robot mobility solutions for industrial, scientific, medical and several other purposes.

Among these solutions is the car-like robot, using wheels to move. However, there are many different options within this solution, different types of wheels and configurations on the robot that each offer key advantages for a variety of objectives.

Choosing a wheel configuration for robot vehicles is of extreme importance for the mobility of said robot, and its purpose must be taken into account while studying all the options.

Starting on an existing prototype with a differential configuration, other configurations were implemented. This will address the differences between configurations, their strong and weak points, the control methods for each configuration, and the trajectories each of them allow the robot to make.

This analysis will make the choice of configuration for each scenario clearer.

**Keywords:** Robotics; Wheel Configuration; Trajectories

# Acknowledgements

I would like to express my gratitude to my supervisor and second supervisor for the support provided during the course of this project.
Additionaly, I would like to extend my thanks to my classmates and friends, whose support during the project and the five years before it is immeasurable.
Lastly, I would like to acknowledge my family and my girlfriend who kept me motivated and focused during all the time studying at FEUP. Without their presence this would not have been possible.

*"I have no special talent. I am only passionately curious."*

Albert Einstein

# Contents

# List of Figures

# List of Tables

# Abbreviations and symbols

| | |
|---|---|
| AGV | Autonomous Guided Vehicle |
| APF | Artificial Potential Field |
| ICR | Instantaneous Center of Rotation |
| CAD | Computer Aided Design |
| CAM | Computer Aided Manufacturing |
| CAE | Computed Aided Engineering |
| PCB | Printed Circuit Board |
| ROS | Robot Operating System |
| PWM | Pulse Width Modulation |
| UART | Universal Asynchronous Receiver-Transmitter |

# Chapter 1

# Introduction

Robot vehicles are crucial nowadays for almost every industry, and there exists at least one in most factories. However, the wheel configuration on these robots presents a challenge since it may compromise the purpose of the robot if it is not adequate or less fitted.

This dissertation will cover different configurations for mobility and compare them all while studying their advantages and disadvantages.

This deep look into the robot wheel configurations will allow us to understand their working better and possibly discover new applications for some of these configurations.

The aim of this work is to study the different control methods necessary to implement for each configuration, as well as the trajectory generation that is allowed by different configurations. Other comparison topics will be the influence the floor and slippage have on the robot depending on the configuration, the mechanical complexity of the wheel and the configuration itself, and the maximum load the robot can carry, all of which can affect the choice of configuration to use.

In order to do this, we will need to study the different algorithms in use today for trajectory planning, as well as analyze whether the different wheel configurations will affect the algorithms employed for the better or worse.

The prototypes built during the dissertation and the knowledge gained will be used to aid future classes studying robotics and autonomous systems.

## 1.1 Motivation

The motivation behind this work stems from the need to better comprehend the various control methods and trajectory generation techniques required for different wheel configurations.

In addition to theoretical analysis, this thesis will involve the construction of prototype robots with different wheel configurations. These prototypes will serve as practical examples to facilitate the study of robotics and autonomous systems in future classes. Students will be able to gain hands-on experience and a deeper understanding of the complexities associated with different wheel configurations and their real-world implications.

## 1.2 Document Structure

To achieve these objectives, in Chapter 2, an exploration of the current state-of-the-art trajectory planning algorithms and search algorithms is necessary. This research will allow us to assess whether different wheel configurations have a significant impact on the performance of these algorithms, either enhancing or hindering their effectiveness.

Following this, Chapter 3 will present a comprehensive explanation of the different wheel configurations under consideration as well as discuss the software and hardware used for prototyping and experimentation.

Chapter 4 will describe the work conducted, including the design and construction of the prototype robots, the code developed for the tests realized, and provide a view of the fully built robots. It will also go over the hardware used during the project, explaining the reason for which they were chosen.

Chapter 5 will present the results and analysis of the experiments performed with the prototypes, along with a small discussion about them.

Finally, in Chapter 6, we will present a conclusion to the document as well as leave some suggestions for future work to be done that can add to the knowledge gained during this project

# Chapter 2

# State of the Art

In this chapter, we will discuss the current State of the Art of several components that will be used during this work as well as the most used algorithms for trajectory planning in the field of robotics.

## 2.1 Automated Guided Vehicles

Automated Guided Vehicles (AGVs) have emerged as cutting-edge technology, revolutionizing the field of mobile vehicles with their advanced capabilities and versatile wheel configurations. AGVs are autonomous or semi-autonomous vehicles designed to perform tasks traditionally carried out by human operators in industrial and logistics settings. These vehicles incorporate sophisticated technologies, including sensors, computer vision systems, and advanced control algorithms to navigate and interact with their environment. AGVs can be equipped with different wheel configurations, such as omnidirectional, differential, or swerve drives, allowing them to achieve enhanced maneuverability and adaptability to diverse operational requirements.

AGV's are widely used in the industry to substitute manned vehicles and conveyors and have been proved an efficient element in factory workspaces, helping to reduce logistic errors and operative costs [1].

State-of-the-art in AGV development encompasses features like real-time mapping and localization, obstacle detection and avoidance, collaborative behavior, and seamless integration with existing automated systems, making them indispensable tools for efficient material handling and logistics operations.

The continual advancements in AGV technology hold promise for increased productivity, improved safety, and enhanced operational flexibility in various industries.

## 2.2 Trajectory Planning

Trajectory planning algorithms play a pivotal role in the field of robotics, and their significance in the context of this thesis cannot be overstated.

Given the focus of this thesis on comparing different wheel configurations for robot mobility, it becomes essential to explore and understand some trajectory planning algorithms. Each wheel configuration may impose specific constraints and dynamics on the robot's movement, necessitating tailored trajectory planning approaches. By studying these algorithms, we can gain insights into how the choice of wheel configuration influences the selection and implementation of trajectory planning methods. This knowledge will enable us to make informed decisions regarding the most suitable wheel configuration for various applications, considering factors such as efficiency, maneuverability, and obstacle avoidance capabilities.

There are many trajectory planning algorithms that are used nowadays, which can be divided into different categories, four of which are:

- Bug algorithms;
- Roadmap;
- Cell decomposition;
- Artificial potential field.

Out of these categories, the algorithms can be sorted into global and local algorithms. Global algorithms, such as roadmaps and cell decomposition, are used when the dimensions and location of the obstacles are known. On the other hand, local algorithms like bug algorithms and the potential fields, are used when this information is not known [2].

### 2.2.1 Bug algorithms

Bug algorithms are the simplest method of trajectory planning, however, they are not the most efficient.

The first bug algorithm (Bug 1) consists of the following steps, which can be seen in Figure 2.1:

**1.** Calculating a direct line between the robot and the end point;

**2.** Traveling along that line until an obstacle is found;

**3.** Going around the whole obstacle and recording the closest point to the destination;

**4.** Go around the obstacle again until it reaches the recorded point;

**5.** Return to step 1.

This first iteration is not very efficient as it makes the robot move around the whole obstacle before continuing towards the destination, even if the desired path was crossed. It can easily become unfeasible when the obstacle is very large in size [3] [4].

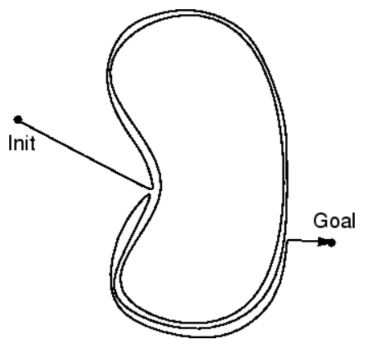


Figure 2.1: Bug 1 algorithm example [3]

The next iteration of the bug algorithms (Bug 2) improves upon the previous one, by changing step 3.

In this version, which is depicted in Figure 2.2, the robot only contours the obstacle until it finds the original line again and resumes from step 1 after that [3] [4].

However, this method still had a major problem. If the end point was in the obstacle then the robot would be trapped in a loop.

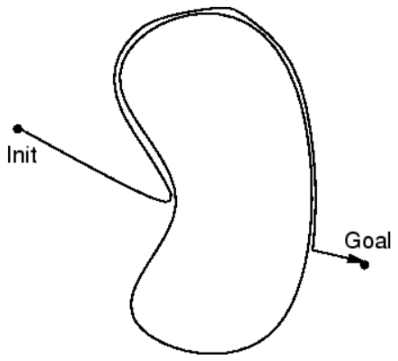


Figure 2.2: Bug 2 algorithm example [3]

To solve this, Bug 2+ was created, in which the robot only resumes following the line if it has not yet crossed the line at a closer point beforehand [3] [4].

An example of this can be seen in Figure 2.3.

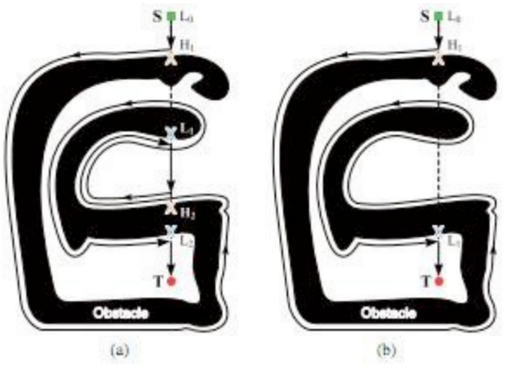


Figure 2.3: Bug 2+ algorithm example [5]

### 2.2.2 Roadmap

Roadmap algorithms use nodes and links between them to represent possible paths for the robot.

The main task in this approach is to create the roadmap itself, and we will look into two techniques to do it.

The first one is the Visibility Graph (VG), which can be seen in Figure 2.4.

In this method, the nodes are the vertices of the obstacles and the links are all connections between nodes that are visible to each other [6] [7].

This means the robot will be traveling from vertice to vertice until it reaches the final point, which increases the possibility of colliding with an obstacle and forces the vehicle to make small deviations from the path in order to avoid the vertices of the obstacles [5].

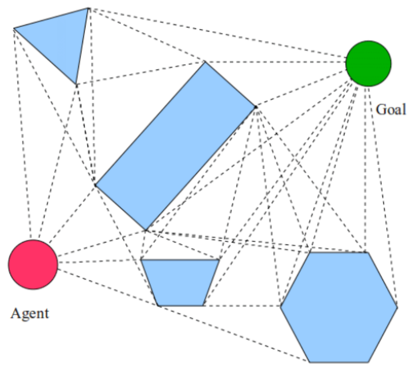


Figure 2.4: Visibility graph algorithm example [6]

The other method, which is shown in Figure 2.5, is the Voronoi Diagram (DV), and it consists of a set of points that are equidistant to two or more obstacles. This way, the map is divided into regions that each contain a single obstacle. The set of points creates a path for the robot to follow that will always be the furthest away possible from any obstacle [5].

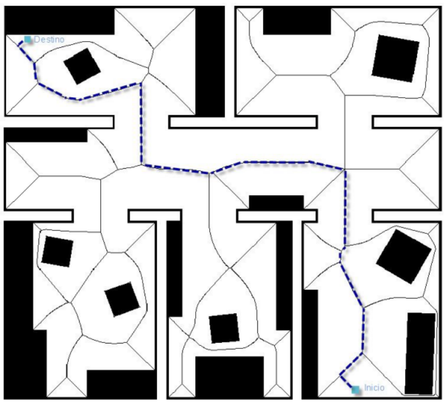


Figure 2.5: Voronoi diagram algorithm example [5]

### 2.2.3 Cell Decomposition

This algorithm is similar to the roadmap, but instead of using nodes and links, it splits the map into cells.

The decomposition can be divided into two methods:

**1.** Exact cells;

**2.** Approximated cells.

In the exact cell decomposition, the map accurately represents the real world, and cells represent spaces that are either occupied or free. The most common cells used are polygons and trapezoids. This method cannot be used in certain situations, such as when two vertices share the same coordinates, as this would not be represented in the decomposition [8].

On the other hand, using approximated cell decomposition means the map will not be a faithful representation of the real world, as the cells can represent spaces that are occupied, free, or partially occupied.

In this method, the cells are usually simple shapes such as squares, which makes the process faster. The decomposition can be made in two different ways [5]:

**1.** Fixed sized cells, as shown in Figure 2.6:

These must be small enough for a path to exist and do not use all of the available space, as the cells which are partially occupied are not used for the path planning [5].

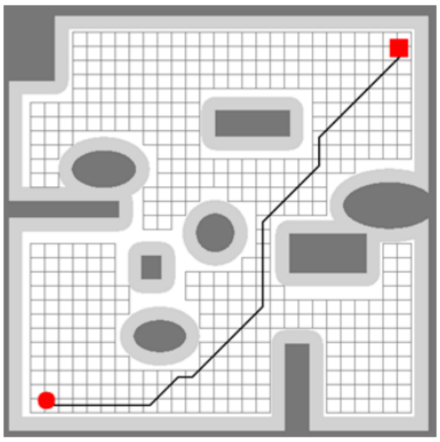


Figure 2.6: Fixed size cell decomposition example [5]

**2.** Quadtree, shown in Figure 2.7:

This decomposition is an iterative process that starts by dividing the map into four equal cells, any cell that is partially occupied is then divided into another four smaller cells until a minimum cell size is reached. This way, there is an optimization of the space available [5].

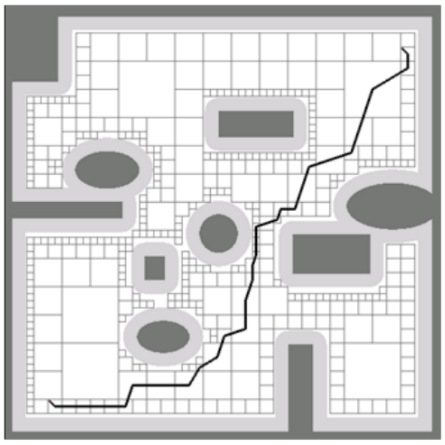


Figure 2.7: Quadtree cell decomposition example [5]

### 2.2.4 Artificial Potential Field

This is one of the most popular methods of path planning due to its' simplicity and safety. The Artificial Potential Field (APF) generates a field of forces that can be either repulsive or attractive, the resulting field will influence the path of the robot. The repulsive force is created by obstacles the vehicle must avoid, and the attractive force by the destination point which the robot must get to [9].

An example of this method is shown in Figure 2.8.

This technique has several advantages as it completely avoids obstacles, is very fast to compute, meaning it can be employed in real-time, and generates a smooth path with no sharp turns that would slow the robot down [10].

It also has, however, some disadvantages, such as the possibility of creating a point where both forces cancel each other, making it a local minimum where the vehicle can get stuck, and if the obstacle, the destination, and the robot are aligned with the obstacle being located between the robot and the goal and the repulsive force is greater or equal to the attractive then reaching the destination becomes impossible.

Both of these problems have been studied, and solutions to solve them have been created.

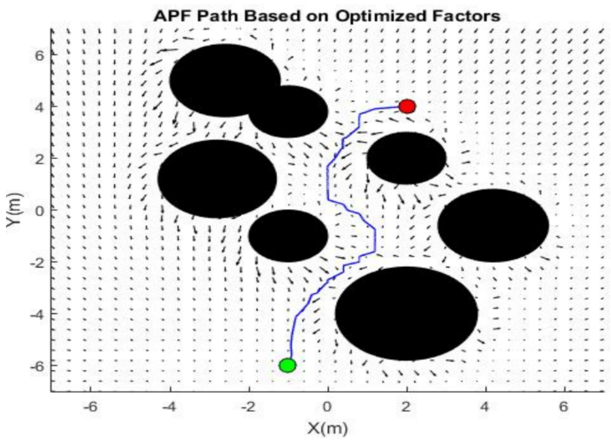


Figure 2.8: Potential field algorithm example [11]

## 2.3 Search Algorithms

Additionally, this thesis will also delve into the topic of search algorithms, which play a crucial role in robotics, particularly in tasks such as pathfinding and motion planning. Search algorithms enable the robot to navigate through complex environments by efficiently exploring the state space and identifying optimal paths or sequences of actions. Understanding different search algorithms and their suitability for specific scenarios is paramount to developing efficient and robust robot navigation systems.

In this section, we will explain two search algorithms, Dijkstra's and A*.

### 2.3.1 Dijkstra Algorithm

Dijkstra's algorithm finds the shortest path between two points in a graph. It starts at a specific starting point and explores the graph to determine the shortest distance to each other point in the graph or to a goal point if one is specified.

The algorithm keeps track of the current shortest distance from the starting point to any point in the graph and creates two sets of points, the already visited points whose optimal path has been found and the points whose optimal path is not yet known. Initially, the distance to the starting point is set to infinity for every point except the starting point, for which it is set to 0.

After this, it iteratively calculates the point closest to the starting point and considers it the current point. Afterward, it visits all the neighboring points of the current point and, for each one, calculates the distance to the starting point. The neighbor point with the shortest distance to the starting point is then considered the new current point, and the process is repeated until every point has been visited or the goal point has been reached.

This algorithm encounters a problem in situations where the graph used is too big to be stored in memory. This prevents the algorithm from storing all possible paths and finding the shortest one.

### 2.3.2 A* Algorithm

The A* algorithm is commonly used to find the shortest path between two points in a graph. In order to improve efficiency, it combines Dijkstra's algorithm with heuristic functions.

The algorithm follows the same principle as the Dijkstra, but, for each point, it estimates its distance to the goal point using a heuristic function. It chooses the point that minimizes the cost function of the combined weights of the current distance from the starting point to the goal point.

The algorithm iterates through the most promising points to lead to the shortest path, optimizing the search and, therefore, making it more efficient than the Dijktras algorithm.

## 2.4 Position Estimation

Accurate position estimation is essential for effective navigation and control of robot vehicles. Odometry is a commonly used method for estimating position based on measurements obtained from the robot's motion sensors, typically wheel encoders. It relies on tracking the changes in wheel rotations and converting them into estimates of the robot's displacement and orientation.

In the context of this thesis, due to the lack of sensors used in the motors, odometry plays a crucial role in determining the robot's position within its environment, facilitating the evaluation and comparison of different wheel configurations. By analyzing the accuracy and reliability of odometry estimates, valuable insights can be gained regarding the performance and limitations of each configuration.

It is important to note that while odometry provides a practical and cost-effective solution for position estimation, its accuracy tends to degrade over time due to accumulated errors. The errors can stem from various sources, including sensor noise, wheel slippage, and mechanical wear and tear. As a result, the longer the robot operates or, the more positions it tests, the higher the likelihood of significant deviations between the estimated position and the ground truth.

In odometry systems, there are two different types of errors, usually called errors of type A and type B.

Type A errors are rotation errors, which can be caused by an incorrect measure of the distance between the wheels or simply an incorrect measure taken from the motor encoders.

Type B errors, on the other hand, are liner movement errors, that are caused by differences in the wheel size, differences in the motor speed, or as with type A, incorrect measures taken from the motor encoders [12].

Examples of these errors can be seen in Figure 2.9.

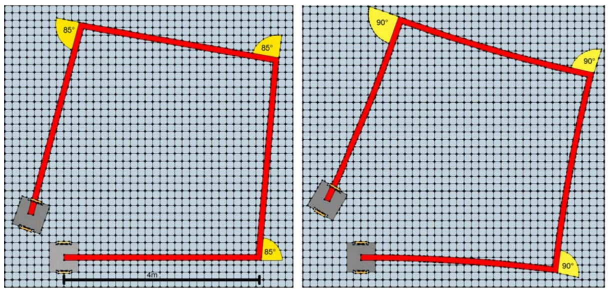


Figure 2.9: Example of odometry errors of type A (left) and type B (right) [12]

Due to these errors, the estimation of the position by the robot always has some uncertainty, which can be bigger or smaller, depending on the velocity of the robot as well as on the degree of the errors that occurred in the estimation phase. This uncertainty has to be taken into account when using path planning and obstacle avoidance algorithms [13].

In figure 2.10, some examples of this can be seen.

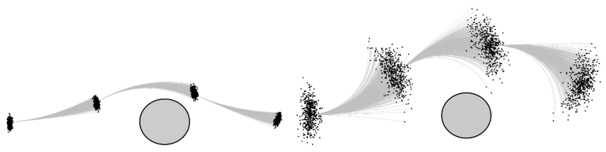


Figure 2.10: Examples of the robot position estimation in an obstacle avoidance scenario, with smaller (left) and bigger (right) uncertainties [13]

As can be seen, due to this uncertainty, the minimum distance that the robot must maintain to the obstacles may need to be increased over time, as the errors increase and the estimation is further from the real position.

## 2.5 Summary

In this chapter, we have undertaken a comprehensive exploration of the state-of-the-art in the field of robotics, focusing on AGVs, trajectory planning algorithms, and search algorithms.

Through an in-depth analysis of these topics, we have gained valuable insights into the cutting-edge developments and advancements that have shaped the current landscape of robotics and automation.

# Chapter 3

# Theoretical Background

## 3.1 Configurations

In this section, we will discuss the different wheel configurations that will be studied during this work.

### 3.1.1 Differential

The differential configuration on robots is the simplest, and one of the most commonly used by the most basic robots.

This configuration uses two non-steerable driving wheels on the front and a free-spinning wheel on the rear for stability, as can be seen in Figure 3.1.

This configuration is characterized by the distance between the wheels and the diameter of each wheel [14].

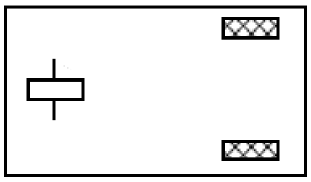


Figure 3.1: Wheel configuration on differential robot [15]

Considering the scenario where the two-wheel motors are activated at constant speeds. Let $dl$ and $dr$ represent the distances traveled by the left and right wheels, respectively. The following simple motions arise from this setup:
When $dl = dr > 0$ the robot moves straight and forward.

When $dl = dr < 0$ the robot moves straight and backward.

When $dr > 0$ and $dl = -dr$ the robot rotates counterclockwise in place.

When $dr < 0$ and $dl = -dr$ the robot rotates clockwise in place.

For the last two cases, it is important to determine the extent of the robot's rotation. Additionally, it is necessary to understand what occurs when $|dl|$ is not equal to $|dr|$. These questions can be answered by identifying the Instantaneous Center of Rotation (ICR) and applying a rotation matrix. The ICR lies along the common perpendicular line passing through the rear wheels and a specific point.

All four simple motions described above can be expressed using equation 1:

$$r_c = \frac{\gamma(dl + dr)}{(2(dr - dl))} \tag{1}$$

Where $r_c$ is the radius of the rotation.

When, for example, $dr = dl$, the formula yields $r_c$ equal to infinity, indicating that the robot moves straight along a "circle with infinite radius", which means the robot will move in a straight line [16].

### 3.1.2 Tricycle

The tricycle configuration is widely used in robotics, as it remains simple while improving some issues of the differential setup.

It is very similar to the Ackermann steering, the difference being that in an Ackermann configuration, there can be multiple driving wheels connected by an axis; Then, if the center of these wheels coincides with the central axis of the robot, the combination of the forces is in the same direction as on the tricycle, making their kinematic analysis similar.

This configuration, which can be seen in Figure 3.2, includes one driving and steering wheel on the front and two free spinning wheels on the rear for stability [14].

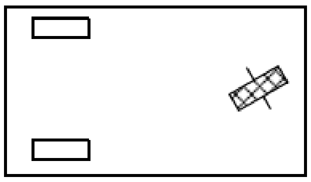


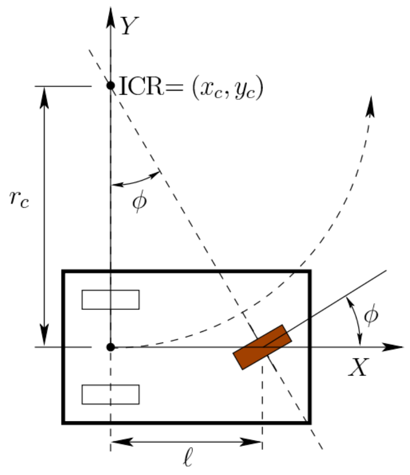
Figure 3.2: Wheel configuration on tricycle robot [15]

Let us now explore the determination of the robot's position in the tricycle configuration after it is driven. Assuming the robot starts with the front wheel steered to the left by an angle $\Phi > 0$, we can derive the coordinates of the ICR. In this case, the x-coordinate of the ICR, $xc$, is 0, while the y-coordinate, $yc$, is $\frac{\gamma}{tan\Phi}$.

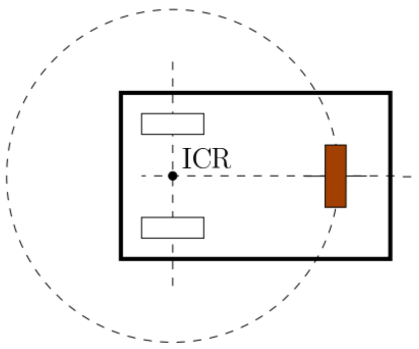
To calculate $yc$, we consider a right triangle formed by the line connecting the point between the back wheels, called the special point, and the front wheel center, along with the two lines perpendicular to the wheels. The base of the triangle is $\gamma$, and the height is $yc$. With the upper interior angle being $\Phi$, trigonometry tells us that:

$$tan\Phi = \frac{\gamma}{yc} \tag{2}$$

The height of the triangle also represents the radius, $rc$, of the circle traced by the robot's special point as it rolls. This circle is centered at the ICR *(xc, yc)*. It is noteworthy that as $\Phi$ approaches zero, $rc$ increases to infinity. In the limiting case where $\Phi = 0$, the robot moves straight, and the ICR does not exist or is located at "northern infinity" [16], as can be seen in Figures 3.3a and 3.3b.



(a) Representation of the ICR of a tricycle robot [16]



(b) Representation of the ICR of a tricycle when the wheel is at a 90º degree angle [16]

Figure 3.3: Representation of the ICR

### 3.1.3 Double Tricycle

This robot utilizes a double steering and driving wheel configuration that is very similar to the tricycle; the force applied by the wheels is in the same direction whether there are two wheels or just one.

This configuration is not commonly used due to being more complex than the tricycle despite it allowing for omnidirectional movement. However, this makes it a unique and interesting study case for this analysis.

This configuration uses two driving, independently steered wheels on the front and one free spinning wheel on the rear for stability and can be seen in Figure 3.4.

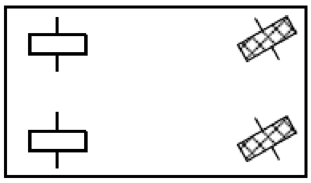


Figure 3.4: Wheel configuration on Ackermann robot

The double tricycle vehicle, like other similar robots, is equipped with multiple wheels, but it always possesses a single ICR. As all the zero motion lines from the wheels intersect at a single point, the ICR is uniquely determined to be located at this intersection. Therefore, the ICR is located precisely at this convergence point [15].

### 3.1.4 Omnidirectional

Omnidirectional movement can be achieved by various types of wheels, in this case, they will be mecanum wheels, and the configuration used can be seen in Figure 3.5.

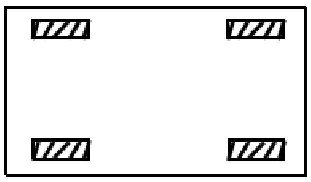


Figure 3.5: Wheel configuration on omnidirectional robot [15]

Mecanum wheels, also called Swedish, are a type of omnidirectional wheels, built using a set of rollers placed around the wheel at a 45º angle [17]. An example of these wheels is shown in Figure 3.6.

By building the wheel this way it is possible to change the direction of the force applied on the floor during the movement, making it perpendicular to the axis of rotation of the roller instead of simply pointing forwards [18].

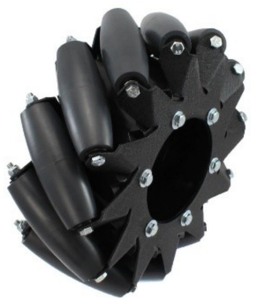


Figure 3.6: Example of mecanum wheel [19]

Using these wheels in a certain arrangement, the robot can move in every direction without steering the wheels, as can be seen in Figure 3.7.

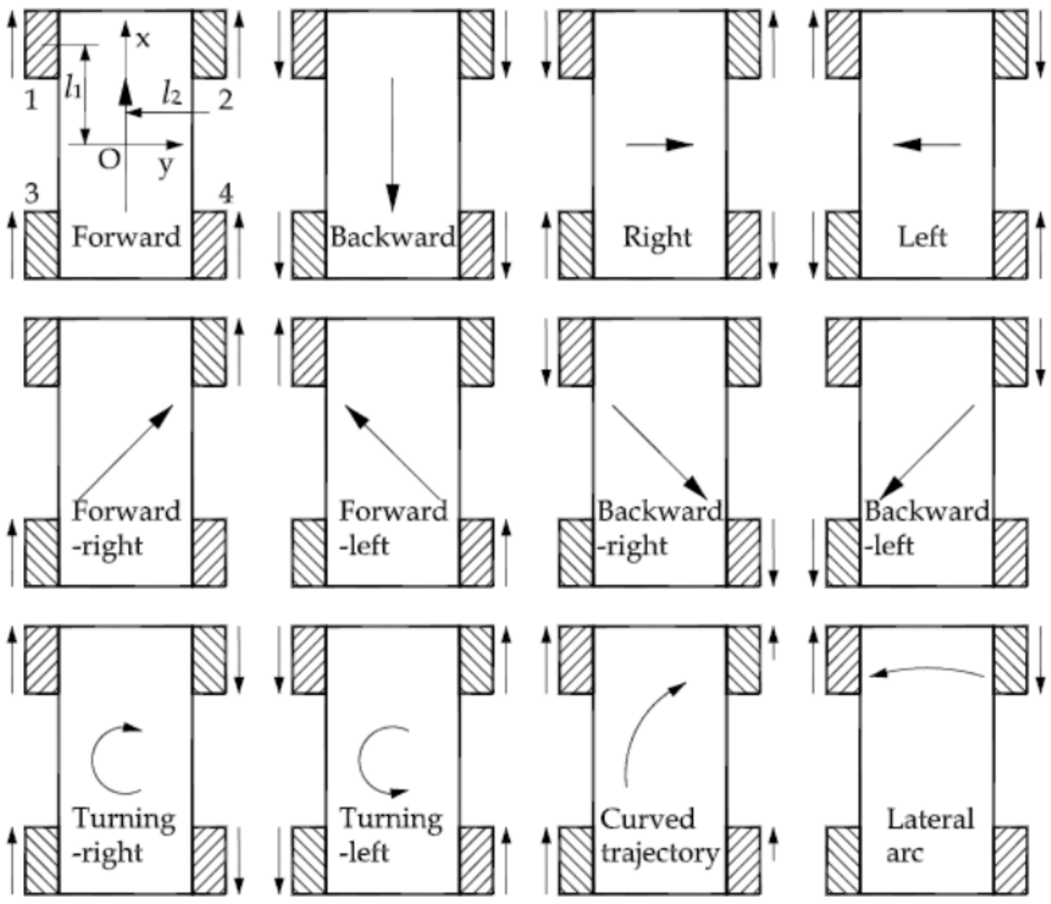


Figure 3.7: Example of movements achieved by using Mecanum wheels [20]

By driving certain wheels and not others, or driving some wheels forward and others backward, it is possible to achieve movement and rotation in any direction using the minimum space possible.

## 3.2 Summary

In conclusion, this chapter has provided a comprehensive exploration of four distinct robot configurations, namely the Differential, Tricycle, Double Tricycle, and Omnidirectional.

Through a detailed examination of each configuration, we have gained valuable insights into their unique designs, functionalities, and key differentiating features.

# Chapter 4

# Implementation and Design

In this chapter, we will discuss the work done and the methods used during this project, such as the design of the parts for the robots, the code development and algorithm implementation, and the tests performed, as well as the hardware and software used.

## 4.1 Hardware and Software

### 4.1.1 Design Tool

Fusion 360 is a Computer-Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Aided Engineering (CAE), and Printed Circuit Board (PCB) software that allows the user to design specific hardware parts of the robot with precision in 3D and export these files for 3D printing.

This software will be used to create the frames of the robots for future printings [21].

### 4.1.2 Microcontroller

Communicating with a system that cannot be physically reached is a critical feature that most robots nowadays must have. The most common way of achieving this is by using Wi-Fi communication, and in order to do this, the system must use a microcontroller board that supports Wi-Fi communication.

There are a variety of boards that can perform this, such as the STM32F4-Discovery and STM32 NUCLEO from STMicroelectronics, the MSP 430 series of microcontrollers from Texas Industries, and the Raspberry PI 3 and 4, among others.

However, in an effort to minimize the cost of the board and still maintain the key features needed, the chosen board will be a Raspberry PI pico W, which can be seen in Figure 4.1.

This board uses the RP2040 microchip and was designed to be a low-cost but flexible platform for the chip [22].

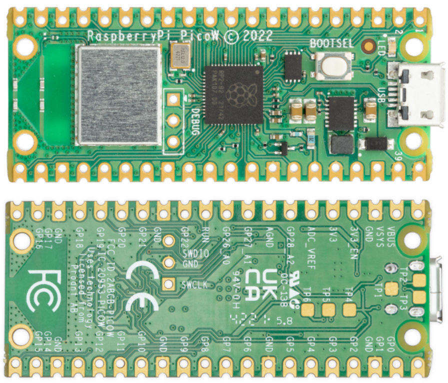


Figure 4.1: Raspberry Pi Pico W [22]

### 4.1.3 DC motor

In the field of robotics, the utilization of DC motors is very common, as these provide a reliable and efficient means of converting electrical energy into rotational motion.

The most simple DC motors, despite being efficient, do not generate much power, which in certain cases can mean the robot will not be able to move. To fix this, motors can be built with a gearbox, which increases the torque produced, meaning the motor has more power.

Another upgrade that DC motors can have is the usage of encoders, which give real-time feedback on the movement of the motor shaft, and in turn, the wheel.

Given the requirements for this project, a DC motor was chosen, which has a 120:1 gear reduction box and a quadrature encoder that provides a resolution of 960 within one round, this motor can be seen in Figure 4.2, [23].

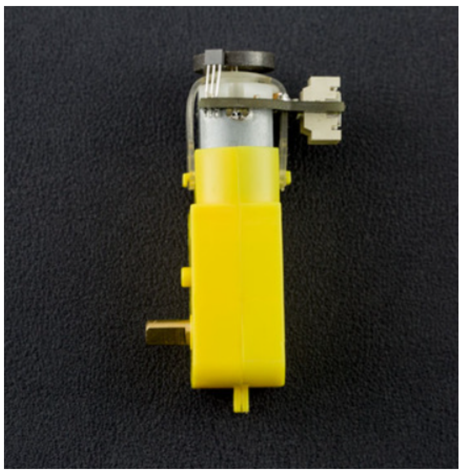


Figure 4.2: DC motor [23]

### 4.1.4 Servo

Servos play a vital role in robotics, providing precise and controlled motion to various components. These actuators allow for precise control of angular position and velocity, as well as acceleration.
They consist of a rotary actuator, a position feedback device, such as an encoder or potentiometer, and a closed-loop system, which ensures the servo maintains an accurate position.

For this project, having reliable and accurate values of the angular position of the servo is of utmost importance; therefore, using a servo that measures its absolute position is required.

Given the conditions of the project and in an effort to minimize the cost, the ones chosen were the serial bus intelligent servos LX-16A by LewanSoul, which can be seen in Figure 4.3.

These servos have many advantages, such as a high-precision potentiometer, the ability to connect several of them in series while using a single control board, two separate functioning modes, one with a range of 240º degrees and another that is able to rotate 360º continuously, with controllable speed and direction, and a compact design.

To facilitate the communication between the controller and the servos, a BusLinker board from LewanSoul will be implemented and will use UART protocol to communicate with the Pico board [24].

Figure 4.3: LX-16A servo [24]

### 4.1.5 Driver

Although microcontrollers can generate a small voltage output, it is not enough to consistently drive motors at high speeds.

To do this, the controller board must be used with an external motor driver to ensure enough power reaches the motors without burning any components on the microcontroller.

In this project, as it was already decided, we will use a Raspberry Pi Pico W board, so, taking that into account, a shield board that is specifically made to be used in conjunction with a Pico board, was chosen to be used. This board is the /PICO4DRIVE, shown in Figure 4.4

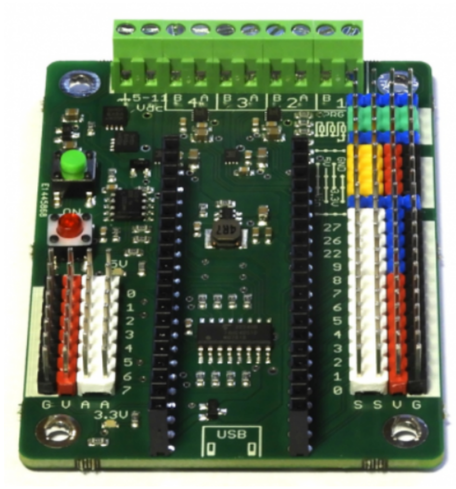


Figure 4.4: PICO4DRIVE driver board [25]

Moreover, the shield offers the capability to drive up to four motors simultaneously and also provides essential protection against overvoltage and excessive current flow [25].

This integration ensures the efficient and reliable operation of the motors, allowing for precise control and synchronization of wheel movements.

## 4.2 Part Design

In order to make a valid comparison between the several models of automated vehicles, it is important to guarantee that they are built similarly and use the same motors and servos and that their wheels are of similar size and material.

To achieve this, both robots were fully designed in Fusion360 and 3D printed to be later assembled and tested.

In this section, we will briefly discuss the several iterations of the parts designed for the robots, including the chassis and the driving wheel support that allows for rotation.

### 4.2.1 Tricycle

The design process began with the tricycle chassis. The first version of this robot used two DC motors with an encoder, one to drive the wheel and another to rotate it.

The second motor would be placed beneath the chassis, and its shaft would be secured to the wheel steering piece.

This design can be seen in Figure 4.5

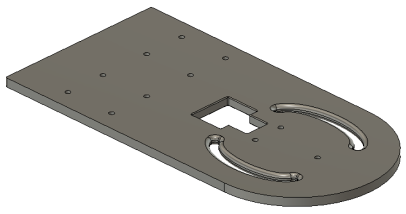


Figure 4.5: First version of the tricycle robot chassis

In order to make the front wheel turn along its z-axis, a special part had to be designed that would support the wheel and connect it to the chassis, while at the same time giving it freedom to rotate in any direction.

Some existing steered motorized wheels inspired the design for the wheel steering part, as can be seen in Figure 4.6.

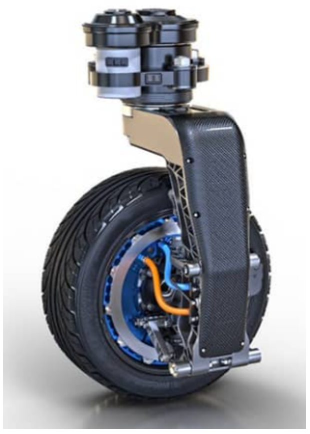


Figure 4.6: Tricycle wheel that served as inspiration for the design of the 3D model [26]

The first iteration of this piece was designed to have the motor in a vertical position that would protect it from any bumps or falls from the robot and can be seen in Figure 4.7.

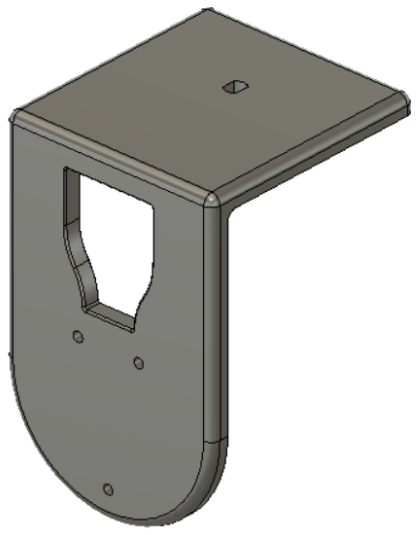


Figure 4.7: First version of the steering wheel support piece

This version was built, and through testing, several flaws were found.

By placing the driving motor vertically, the robot became too tall and, therefore, too unstable. This flaw was also increased by its rear wheels being too close together, causing it to fall over easily.

Also, using a dc motor with a partial encoder to handle the rotation of the wheel is not ideal, as it is not accurate and does not measure the absolute position of the wheel.

Due to this flaw, the second DC motor was replaced with a digital servo, which required a redesign of the robot.

The second version of the chassis tackled both of these flaws. Placing the servo on top of the chassis instead of beneath helped decrease the height of the robot, and making the back of the chassis wider allowed for more spacing between the support wheels, which increased stability as well.

In addition, holes were made in the chassis to secure the microprocessor boards and the batteries used.

This version can be seen in Figure 4.8.

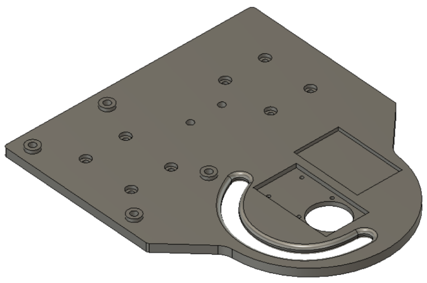


Figure 4.8: Final version of the tricycle robot chassis

The wheel steering piece was also redesigned so that it can be screwed to the servo and to allow the DC motor to be placed horizontally, decreasing the height of this piece and, consequentially, the whole robot significantly, as shown in Figure 4.9.

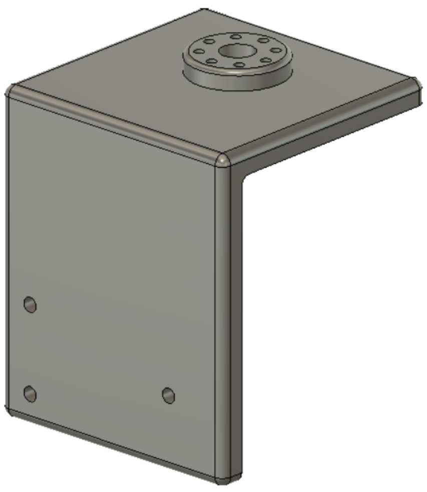


Figure 4.9: Final version of the steering wheel support piece

In addition to these parts, and due to the height of the robot, supports for the back wheels had to be designed and printed.

This part, shown in Figure 4.10, was made with a cross between the legs to add strength and will later be secured with screws onto the main chassis.

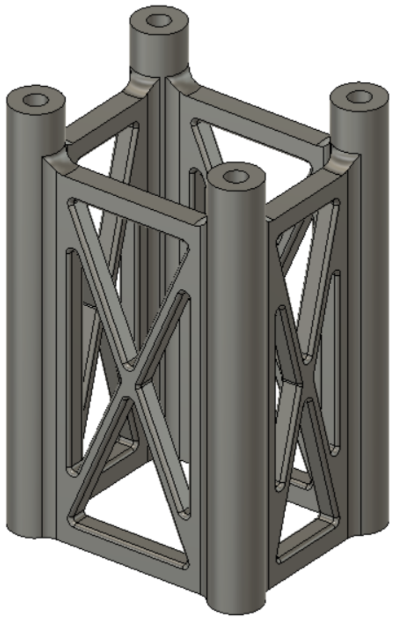


Figure 4.10: Design of the support for the back wheels of the robot

### 4.2.2 Double Tricycle

The design process for the double tricycle robot was simpler, as all the knowledge gained in the previous design was used in it.

The main chassis was made by mirroring the tricycle chassis and widening it to accommodate the two-wheel design, as is shown in Figure 4.11.

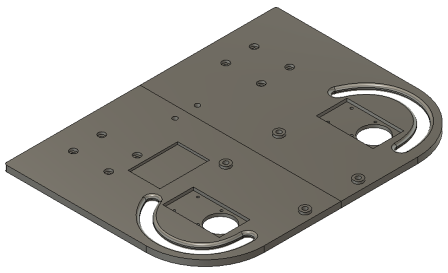


Figure 4.11: Design for the chassis of the double tricycle steering robot

However, by widening the frame, it became too big to be 3D printed in most printers, so to allow the part to be printed, it was divided into two parts that could be printed individually and glued together. To increase the load resistance of the chassis, holes were made in the face that would be glued for plastic pegs to be placed.

A demonstration of these pegs can be seen in Figure 4.12.

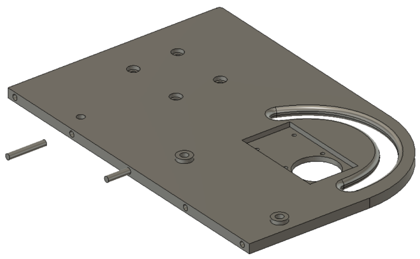


Figure 4.12: Demonstration of the pegs utilized to add robustness to the chassis

The design for the wheel steering part was not altered, but in order to have both DC motors facing the same way when assembled, one of the parts was mirrored when printed.

### 4.2.3 Complete Robot Showcase

In this section, we present visual representations of the fully built robots after the design and 3D printing processes.

Despite not being designed during this project, a third robot, which can be seen in Figure 4.13, was built with a differential configuration in order to use it as a comparison to the other robots.

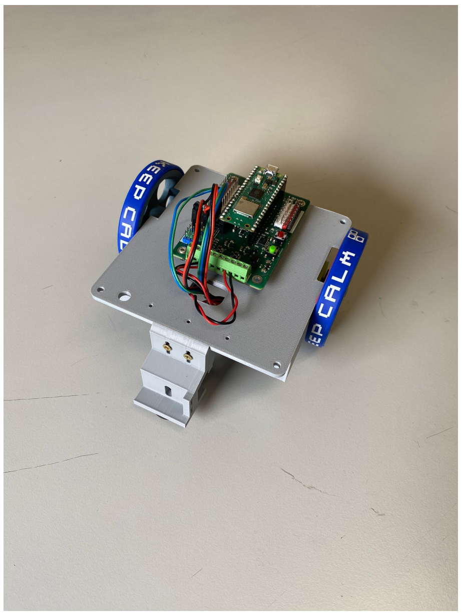


Figure 4.13: Robot with a differential configuration

In Figure 4.14 it is possible to see the finished version of the tricycle robot after 3D printing all the previously designed parts.

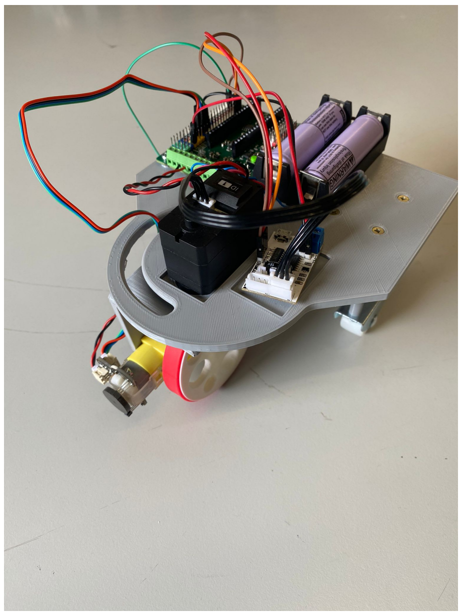


Figure 4.14: Robot with a tricycle configuration

Finally, in Figure 4.15, the fully built double tricycle robot can be seen.

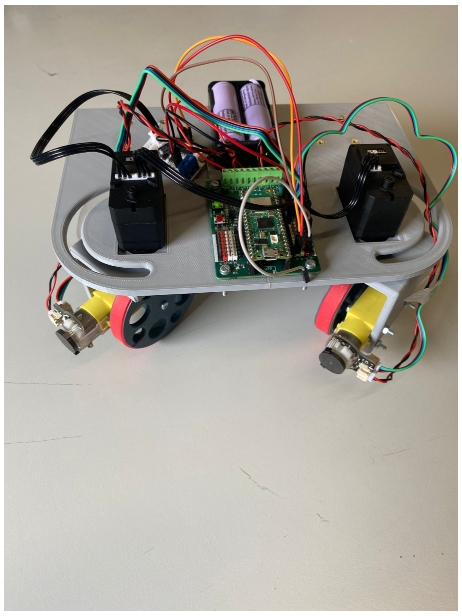


Figure 4.15: Robot with a double tricycle configuration

These visual representations of the complete robots provide a comprehensive view of the real robots, how they function, and how the designs previously discussed interact with each other.

## 4.3 Electronic components

In this section we will take a brief look at how the connections between the several different components were made, in Figure 4.16 the connections for the double tricycle robot can be seen.

Motor A, mounted on the left of the robot, was connected to the drive output 2 of the PICO4DRIVE board, and its encoder to the pins 27 and 26. Motor B, on the right of the robot, was connected to output 4 on the board, and its encoder to pins 22 and 9. Both of the encoders were also connected to ground and the 5V output from the board.

The BusLinker board was powered by the PICO4DRIVE board, and communication between them was made using UART protocol. Pin 0 of the driver board is a TX pin, which deals with transmiting data, and is therefore connected to the RX pin, which receives data, of the BusLinker. On the other hand, Pin 1 (RX) is connected to the TX pin of the BusLinker. The servos are also connected in series to the BusLinker with dedicated cables.

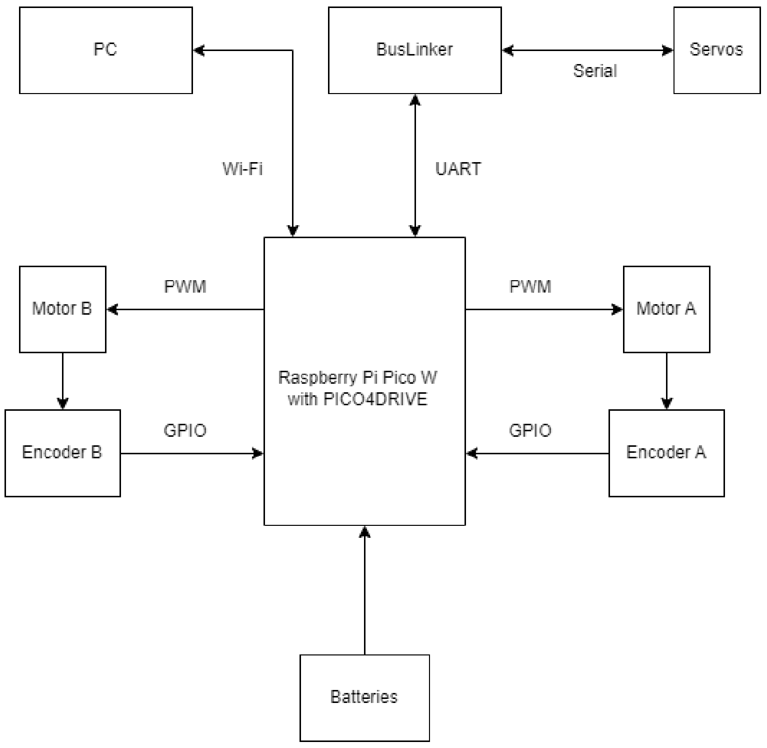


Figure 4.16: Diagram of the connections between the components of the double tricycle robot

This example can be used as a base for the connections of the differencial and the tricycle robots. The differencial robot is connected in the same way, with exception of the BusLinker board that is not used. The tricycle robot does not utilize motor B, therefore those connections do not need to be made.

## 4.4 Code and Algorithm

In this section, we will take a look at the code developed for this project, such as the path-finding algorithm, the interface used to communicate with the robot, and the motion equations implemented.

### 4.4.1 Internet connection

To be able to communicate with the robot at any time, the Raspberry Pi board hosts a webpage through which the user can send commands to the robot.

This webpage was made using HTML code and queries, making it easy to add more functions to it should it be needed.

An example of this webpage that was used during the testing phase of the project can be seen in Figure 4.17.

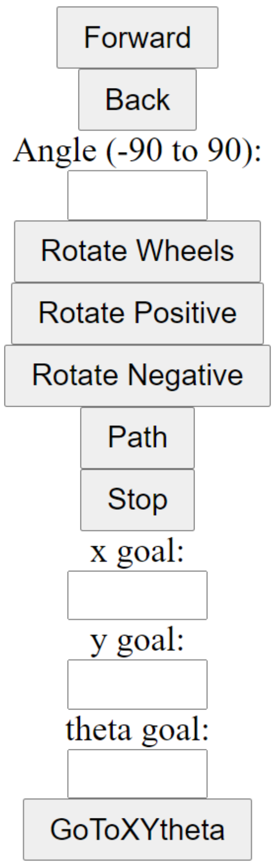


Figure 4.17: Example of the webpage used during testing to communicate with the robots

### 4.4.2 Speed test

In the previous tests, the robots calculated the distance between their position and a goal point, proceeding to move toward that point at a defined velocity.

However, it is important to be able to control the velocity directly and test the robots at several different velocities.

In order to control the velocity, firstly, we need to understand how the motors are controlled and how it relates to their mechanical output.

To control the motor, the microcontroller board simply outputs a PWM signal with a specific duty cycle.

The duty cycle of a motor control signal in a PWM system determines the average voltage applied to the motor and, consequently, affects the motor speed.

In a PWM system, the control signal consists of a square wave with a fixed frequency. The duty cycle represents the percentage of time the signal is "on" or active (high) compared to the total period of the signal. A duty cycle of 0% means the signal is always off, while a duty cycle of 100% means the signal is always on.

The average voltage applied to the motor is proportional to the duty cycle. For example, with a 50% duty cycle, the motor receives an average voltage that is approximately half of the maximum voltage.

The speed of a motor is often directly proportional to the average voltage applied. Increasing the duty cycle increases the average voltage, resulting in higher motor speed. Decreasing the duty cycle decreases the average voltage and lowers the motor speed.

In order to have a direct relationship between the duty cycle applied and the speed of the motor, the robot counted the time it took to travel 5 meters at each duty cycle, starting at around 80% (the maximum duty cycle that can be applied safely) down to around 15% (the minimum duty cycle that produced movement). The results can be seen in Figure 4.18.

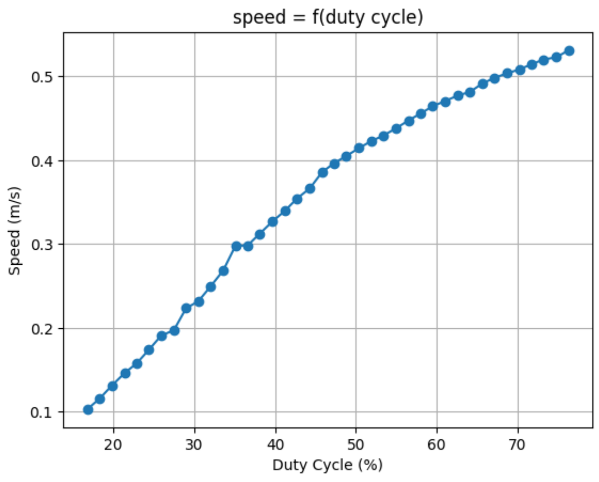


Figure 4.18: Graph depicting the relationship between duty cycle applied and motor speed

From this graph, it is possible to see that the maximum speed the motors can reach is around 0,5 meters per second, and the minimum is 0,1 m/s.

During the tests previously made, the duty cycle used was around 40%, corresponding to a speed of approximately 0,3 m/s.

### 4.4.3 Motion Equations

In order to be able to localize itself on the map without any sensors, the robot must make use of odometry to estimate its current position.

The encoders on the motor provide data on the rotation of the wheels. In this project, the motors used have encoders that generate 960 pulses for each rotation of the wheel.

Using this information, in addition to knowing the perimeter of the wheel, it is possible to calculate a constant that transforms pulses into meters.

Using this value, it is easy to calculate the distance traveled, which in turn can be used to estimate the coordinates of the robot.

The equations used were the following:

$$x(k+1) = x(k) + d(k) * cos(\alpha(k)) \tag{3}$$

$$y(k+1) = y(k) + d(k) * sin(\alpha(k)) \tag{4}$$

$$\alpha(k+1) = atan2(y(k+1) - y(k), x(k+1) - x(k)) \tag{5}$$

Where $k$ is the iteration, $x$ and $y$ are the coordinates and $d$ is the distance traveled.

In the case of the differential and the tricycle robots, $\alpha$ is the angle of the line the robot will follow, and therefore is the angle of the robot according to the global referential.

However, due to the error of the encoder and the lack of precision of the rotation, which becomes bigger the more the robot rotates, the angle used to calculate the desired rotation and the coordinates estimation was a preset angle that was calculated using the current point on the path the robot is on, instead of the estimated position.

On the other hand, given that the double tricycle steering robot has the ability to move diagonally, $\alpha$ is the angle at which the wheels are turned, which is the direction the robot will move in.

### 4.4.4 Path Following

In this project, to test the different trajectories made possible by each wheel configuration, the robots autonomously follow the path previously created by the A* algorithm.

#### 4.4.4.1 Differential

After the previous algorithm calculates the path, the differential robot uses these coordinates to calculate a line between the next point on the path and its current estimated position, calculated using equations 3 and 4.

After creating this line, the robot then calculates the angle $\gamma$ between it and the y-axis, using a different referential, that can be seen in Figure 4.19, where it considers the y-axis to be 0º and the x-axis -90º, using the equation:

$$\gamma = \alpha - 90 \tag{6}$$

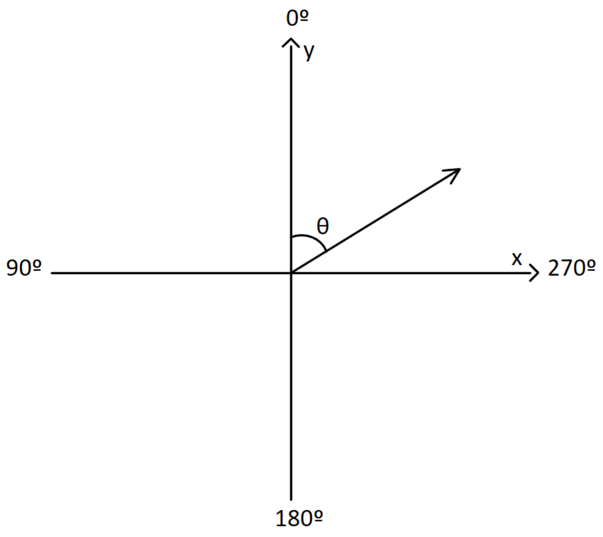


Figure 4.19: Angle referential used in the project

This is the angle to which the robot must rotate in order to align itself with the trajectory it will follow.

The robot then calculates the absolute rotation amount it must turn to match the desired orientation by using the following equation and determines the optimal direction to turn by checking if *rot* is bigger or smaller than 180º and -180º respectively, and removing or adding 360º to the rotation amount.

$$rot = (\gamma - \theta + 180)\%360 - 180 \tag{7}$$

The robot rotates around itself by rotating one wheel forward and another backward, and its current angle is calculated using the pulses received from the encoder in the following equations:

$$d_x = c_x * \frac{0.219}{960} \tag{8}$$

$$d_{avg} = \frac{(d_1 + d_2)}{2} \tag{9}$$

$$\Delta\theta = 360 * \frac{d_{avg}}{2 * \pi * r} \tag{10}$$

Where $d_x$ represents the distance traveled by each wheel, calculated using $c_x$, the number of pulses generated by each encoder. Afterward, $d_{avg}$ is the average of the distance traveled by both wheels, which is then used to calculate $\Delta\theta$, which is the total degrees rotated by the robot, and $r$ represents the radius of the rotation, that in this case is equivalent to the distance between the wheels of the robot. Once $\Delta\theta$ is within a certain threshold of *rot*, then the rotation stops, and the current orientation of the robot $\theta$ is updated using the following equation:

$$\theta(k+1) = \theta(k) + \Delta\theta(k) \tag{11}$$

Once the robot finishes the rotation, it calculates the distance from its current estimated position to the next point using the equation:

$$d_e = \sqrt{((x_f - x)^2 + (y_f - y)^2)} \tag{12}$$

Where $d_e$ stands for the distance error, $xf$ and $yf$ represent the coordinates of the goal point, and $x$ and $y$ represent the estimated position of the robot.

Subsequently, the motors are activated, and the robot begins moving toward the next point.

To accurately measure the distance traveled, the robot relies on the encoder readings from the motors. The robot continues moving until the measured distance approximates the desired distance.

In order to determine the traveled distance, the robot calculates the average distance that each wheel has moved using the following equation 9.

In order to minimize errors, as the robot moves, it calculates the deviation from its desired orientation using the angle calculated with equation 10, and whenever this deviation becomes too large, the robot accelerates one wheel, depending on which side the robot is deviating to, to correct it.

This process of line adjustment, angle calculation, wheel alignment, distance calculation, and controlled movement is repeated for each subsequent point along the path. The robot iteratively refines its trajectory and dynamically adjusts its motion to navigate toward the final destination.

The robot continues executing these steps until it reaches the final point on the path, at which point it concludes its navigation and awaits further instructions or commands.

#### 4.4.4.2 Tricycle

After receiving the path coordinates from the A* function, the tricycle robot initiates its trajectory calculation and motion execution. To ensure proper alignment with the next point on the path, the robot calculates a line between the current point and the next one and adjusts its orientation accordingly.

To rotate itself toward the desired direction, the robot determines the angle of the desired orientation by using equations 5 to 7. To rotate itself, the robot turns the front wheel to a 90º degree angle on its own referential and moves forward or backward, depending on what the shortest rotation is.

Once the robot's orientation approaches the desired angle, the motor gradually slows down and eventually stops. Subsequently, the wheel is adjusted to face forward, preparing the robot for forward motion. At this point, the robot proceeds to calculate the distance between its estimated position and the goal point using the equation 12.

The distance traveled by the robot is calculated using the equation:

$$d = c * \frac{0.219}{960} \tag{13}$$

Where $d$ is the distance traveled and $c$ represents the pulse count of the encoder. To ensure accurate navigation, the robot moves forward while continuously measuring the distance traveled until it approaches the desired distance. This process is repeated until the robot reaches the final point on the path. Upon reaching the destination, the robot stops and awaits further commands.

#### 4.4.4.3 Double Tricycle

Once the path has been determined by the A* algorithm, the double tricycle robot proceeds to calculate a line between its estimated position and the next point on the path. This line adjustment technique aims to minimize the final error by continuously refining the robot's direction for each point along the path.

After establishing the line, the robot calculates the angle to which it should move the wheels in order for them to be parallel to the line it must follow. To do so, it uses the following equation:

$$WheelAngle = \alpha - \theta - 90 \tag{14}$$

Due to the robot's ability to move omnidirectionally, it does not have the need to rotate around itself to move from point to point. However, it can also do so by using the same method as the differential robot.

Once the robot reaches this stage, it calculates the distance from its current estimated position to the next point using the equation 12 and activates the motors to move in the direction of the next point.

In order to measure the distance traveled, the robot utilizes the encoder pulse count for each wheel, with the equations 9 and 8. This distance is continuously updated until the robot approaches the desired point.

To minimize the error in the final position, the robot uses the equation 10 to maintain its orientation correct. In case the robot suffers some deviation, then it accelerates one motor more than the other in order to return to the correct path.

This iterative process is repeated until the final point is reached.

### 4.4.5 Go To XYtheta

Another test implemented to study the trajectory and the precision of the robots was a go to XYtheta function, in which the user specifies an x coordinate and a y coordinate for the robot to move to, and an orientation for its final position.

To do this the robot utilizes equations 5 to 12 to calculate the direction it must follow and the distance it will travel, as it did in the path following function.

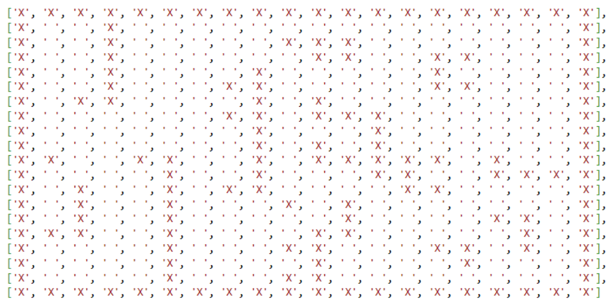
However, in this test, once it reaches the final position, it then rotates again in order to be facing the requested orientation.

The double tricycle robot, that did not rotate around itself previously, now uses equations 5 to 11 to rotate, around itself, similarly to the differential robot.

### 4.4.6 Path Finding Algorithm

As the robot is not equipped with sensors, it is impossible to utilize an online trajectory planning algorithm. Instead, a version of the map must be previously loaded to the robot in order for it to know the position of the obstacles. Using this map, the robot then finds the shortest path between its starting position and a defined goal point.

In this project, an exact cell decomposition was used employed to represent the map, transforming it into a matrix where a cell containing an "X" denotes the presence of a wall, while an empty string represents an unobstructed space, as can be seen on Figure 4.20.



```
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X', 'X', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X', 'X', ' ', ' ', 'X', 'X', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', 'X', ' ', ' ', ' ', 'X', 'X', ' ', ' ', ' ', ' ', ' ', 'X', 'X', ' ', ' ', ' ', 'X'],
['X', ' ', 'X', 'X', ' ', ' ', ' ', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', ' ', ' ', ' ', ' ', 'X', 'X', ' ', 'X', 'X', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', 'X', ' ', ' ', 'X', 'X', ' ', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', ' ', 'X', ' ', ' ', 'X'],
['X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', 'X', ' ', ' ', ' ', 'X', 'X', ' ', ' ', 'X', 'X', 'X', 'X'],
['X', ' ', 'X', ' ', ' ', 'X', ' ', 'X', 'X', ' ', ' ', ' ', ' ', 'X', 'X', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', 'X', ' ', ' ', 'X', ' ', ' ', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', ' ', 'X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', 'X', 'X', ' ', 'X'],
['X', 'X', 'X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', 'X', 'X', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X'],
['X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', 'X', ' ', ' ', ' ', 'X', 'X', ' ', 'X', ' ', 'X'],
['X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X'],
['X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
```

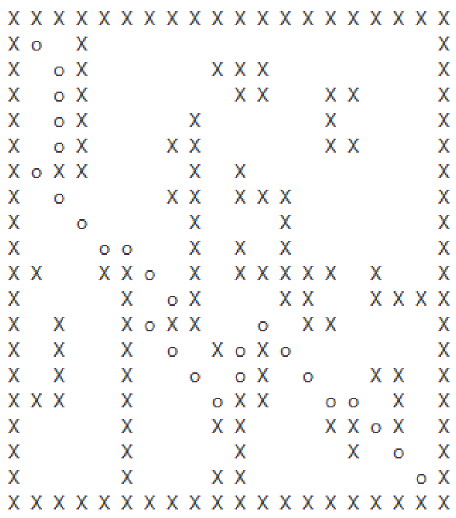
Figure 4.20: Example of the matrix used to draw the map

Each cell was assigned a value based on the presence (value of 1) or absence (value of 0) of a wall. The map representation was the foundation for implementing the A* algorithm, enabling efficient pathfinding between the starting and goal points.

The implementation of the algorithm allows it to use diagonal steps in the path found, which can cause problems if the space between two obstacles is not big enough for the robot to travel through. To prevent any problems caused by the diagonal moving, there is an option that can make the algorithm only allow orthogonal movement from the robot.

When the path is found, the algorithm returns an array of coordinates, representing each step the robot must make to reach the goal point.

A solved version of the previously shown map example can be seen in Figure 4.21.



```
X X X X X X X X X X X X X X X X X X X X
X o   X                               X
X   o X           X X X               X
X   o X             X X       X X     X
X   o X         X             X       X
X   o X       X X             X X     X
X o X X         X   X                 X
X   o         X X   X X X             X
X     o         X       X             X
X       o o     X   X   X             X
X X     X X o   X   X X X X X   X     X
X         X   o X       X X     X X X X
X   X     X o X X     o   X X         X
X   X     X   o   X o X o             X
X   X     X     o   o X   o     X X   X
X X X     X       o X X     o o   X   X
X         X       X X         X X o X X
X         X         X         X   o   X
X         X       X X               o X
X X X X X X X X X X X X X X X X X X X X
```

Figure 4.21: Example map solved by A* algorithm, where 'o' represents optimal path

The path generated is then used by the robot to navigate through the map around the obstacles.

## 4.5 Summary

In conclusion, this chapter has provided a comprehensive overview of the implementation and design process of the robots for this project.

We began by discussing the 3D modeling tool used, Fusion 360, and then the hardware components, which included DC motors, LX-16A servos, Raspberry Pi Pico W, and the PICO4DRIVE shield, each serving crucial roles in the robots' functionality.

The chapter proceeded to showcase the designed pieces and featured images of the fully assembled robots, providing a visual representation of the work done in this phase.

Furthermore, the section covering the electronic components of the robot provided valuable insights into the wiring and integration of the various hardware components.

Lastly, this chapter covered the implementation of several tests for the robot to perform, explaining how they work and what their purpose is.

# Chapter 5

# Results

This chapter presents the comprehensive analysis and discussion of the results obtained from the experimental evaluation of the tricycle, Ackermann steering, and differential robots.

The primary objective of this research was to compare the trajectory capabilities and assess the suitability of each robot configuration in different environments and tasks. By examining the obtained data and conducting a thorough analysis, valuable insights can be gained regarding each robot's performance, maneuverability, and practical applicability.

The discussion of results aims to provide a clear understanding of the strengths, limitations, and implications of using these robots, thereby facilitating informed decision-making for specific robotic applications.

## 5.1 Trajectory Analisys

The analysis of trajectories is a fundamental component of this research, as it allows for a detailed examination of the motion patterns and maneuvering capabilities of the different robots.

By comparing and contrasting the trajectories generated by each robot configuration, valuable insights can be gained regarding their navigational abilities and adaptability to different environments. This analysis serves to identify and evaluate the advantages, disadvantages, and practical implications associated with each robot's trajectory behavior. Furthermore, it provides a basis for understanding how the unique design and motion characteristics of each robot influence its performance and suitability for specific tasks.

Through a comprehensive examination of the trajectories, this section sheds light on the fundamental aspects that contribute to effective robotic locomotion and assists in drawing meaningful conclusions from the experimental findings.

### 5.1.1 Differential

The differential robot exhibits versatile trajectory capabilities, allowing it to navigate through a variety of paths and movements. It can rotate around itself, enabling precise turns and the ability to change its facing direction without requiring excessive space. This rotational ability enables the robot to execute curved trajectories, making it suitable for navigating around obstacles or following curved paths. Additionally, the differential robot can move in a straight line, both forward and backward, enabling efficient point-to-point movement.

One notable advantage of the differential robot is its small and compact size. This compact form factor allows the robot to navigate in tight spaces and confined environments, making it suitable for applications where space is limited. Furthermore, the differential robot is relatively easy to control and program due to its simple design and straightforward motion dynamics. This ease of control and programming facilitates the implementation of various navigation algorithms and facilitates rapid prototyping and development processes.

However, the differential robot also has some limitations. Due to its differential drive mechanism, the robot may experience difficulties in maintaining accurate trajectory control, particularly at high speeds or when encountering uneven surfaces. Moreover, the differential robot's steering mechanism lacks the ability to perform complex maneuvers or omnidirectional movements, limiting its agility in certain scenarios.

### 5.1.2 Tricycle

The tricycle robot demonstrates a diverse range of trajectory capabilities, enabling it to navigate through various paths and execute distinct movements. It possesses the ability to rotate around itself or a specific point, enabling precise turns and maneuverability in constrained spaces. Furthermore, the tricycle robot can execute curved trajectories and move in a straight line, both forward and backward, although moving backward may not be the optimal direction due to its design characteristics.

One advantage of the tricycle robot is its tighter turning radius. The configuration of a single driving and steering wheel on the front allows for sharper and more precise turns compared to other robotic platforms. This capability proves advantageous when navigating through narrow pathways or when executing intricate maneuvers.

However, it is important to consider the limitations associated with the tricycle robot. One notable disadvantage is its increased complexity compared to the differential robot. The presence of a steering wheel and the requirement for synchronized steering and driving mechanisms introduce greater complexity in the design and control of the tricycle robot.

Furthermore, the tricycle robot typically exhibits a larger size compared to other robotic configurations, which may limit its suitability in environments with space constraints. Additionally, the tricycle robot's control and stability can be more challenging due to the intricate coordination required between the driving and steering mechanisms. This increased complexity and potential for instability necessitate careful control strategies and thorough system calibration.

### 5.1.3 Double Tricycle

The double tricycle robot exhibits an extensive range of trajectory possibilities, showcasing its versatility in navigating through diverse paths and executing various movements. Similar to the differential robot, it can rotate around itself or a designated point, enabling precise turns and efficient reorientation. Moreover, the double tricycle robot can effortlessly perform curved trajectories, as well as move in a straight line in any direction, showcasing its omnidirectional movement capabilities. This omnidirectional feature allows the robot to reach any desired point without the need for rotational adjustments.

One significant advantage of the double tricycle robot is its omnidirectional movement, which surpasses the capabilities of both the tricycle and differential robots. With the ability to move in any direction without the need for rotation, the double tricycle robot offers enhanced navigational flexibility and efficiency. Furthermore, it can achieve the same trajectories as the other two robots while also enabling additional maneuvering options.

The double tricycle robot's ease of control is another notable advantage. Its control system can be implemented with relative simplicity, facilitating programming and operation. Additionally, the robot demonstrates high stability during its movements, thanks to its well-balanced design and coordinated driving and steering mechanisms.

However, it is essential to consider the associated disadvantages of the double tricycle robot. One such drawback is the increased mechanical complexity compared to the differential robot. The presence of multiple wheels and the intricate coordination required between the driving and steering mechanisms contribute to a more complex mechanical system, demanding meticulous design, calibration, and maintenance.

Additionally, the double tricycle robot typically exhibits a larger size compared to the differential robot, which may pose limitations in constrained spaces or certain operational environments. Consideration must be given to the available workspace and any potential constraints when deploying the double tricycle robot.

## 5.2 Go To XYTheta

This test allows the user to verify the precision of the robot, by giving it precise coordinates to move to, and a final orientation to turn.

Furthermore, the "Go to XYTheta" test provides crucial feedback on the robot's ability to operate reliably in real-world scenarios. It helps determine the robot's capacity to navigate to specific locations accurately, which is vital for applications such as autonomous exploration, mapping, surveillance, and robotic assistance. Additionally, the test enables comparative analysis between different robot configurations, algorithms, or control strategies, aiding in the selection and optimization of suitable robotic platforms for specific tasks or environments.

By comparing the desired point with the estimated position of the robot, as well as with the real position of the prototype, it is possible to determine whether the guiding system based on odometry is working as intended, and if the data received by the robot through its sensors are precise.

The results of this test can be seen in the Tables 5.1, 5.2 and 5.3.

Table 5.1: Differential robot go to XYTheta results

| $x_n$ | $y_n$ | $\theta_n$ | x | y | $\theta$ | time ms | $x_{error}$ | $y_{error}$ | $\theta_{error}$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 90 | 0,9994 | 0,9994 | 89,95 | 5314 | 0,0006 | 0,0006 | 0,0505 |
| 0 | 0 | 45 | 0,0006 | 0,0006 | 45,09 | 5128 | -0,0006 | -0,0006 | -0,0936 |
| 1 | 1 | 0 | 0,9993 | 0,9993 | -0,0594 | 5156 | 0,0007 | 0,0007 | 0,0594 |
| 0 | 1 | 0 | 0,0009 | 1 | 0 | 4100 | -0,0009 | 0 | 0 |
| 1 | 0 | 180 | 0,9994 | 0,0006 | -180 | 5511 | 0,0006 | -0,0006 | 360 |
| 0 | 0 | 0 | 0,001 | 0 | -359,9 | 4445 | -0,001 | 0,0000 | 359,9 |
| 0 | 1 | 90 | 0 | 0,9992 | -270,1 | 3422 | 0 | 0,0008 | 360,1 |
| 0 | 1 | 0 | 0 | 0,9992 | -359,9 | 744 | 0 | 0,0008 | 359,9 |

Table 5.2: Tricycle robot go to XYTheta results

| $x_n$ | $y_n$ | $\theta_n$ | x | y | $\theta$ | time ms | $x_{error}$ | $y_{error}$ | $\theta_{error}$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 90 | 0,9993 | 0,9993 | 89,61 | 14162 | 0,0007 | 0,0007 | 0,3946 |
| 0 | 0 | 45 | 0,0006 | 0,0006 | 45,48 | 13818 | -0,0006 | -0,0006 | -0,4827 |
| 1 | 1 | 0 | 0,9993 | 0,9993 | -0,4533 | 13784 | 0,0007 | 0,0007 | 0,4533 |
| 0 | 1 | 0 | 0,0008 | 1 | 0,4533 | 12291 | -0,0008 | 0 | -0,4533 |
| 1 | 0 | 180 | 0,9994 | 0,0006 | -179,5 | 13988 | 0,0006 | -0,0006 | 359,5 |
| 0 | 0 | 0 | 0,0009 | 0 | -359,6 | 12268 | -0,0009 | 0 | 359,6 |
| 0 | 1 | 90 | 0 | 0,9992 | -270,5 | 11430 | 0 | 0,0008 | 360,5 |
| 0 | 1 | 0 | 0 | 0,9992 | -359,6 | 3923 | 0 | 0,0008 | 359,6 |

Table 5.3: Double tricycle robot go to XYTheta results

| $x_n$ | $y_n$ | $\theta_n$ | x | y | $\theta$ | time ms | $x_{error}$ | $y_{error}$ | $\theta_{error}$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 90 | 0,9994 | 0,9994 | 89,95 | 8859 | 0,0006 | 0,0006 | 0,0505 |
| 0 | 0 | 45 | 0,0006 | 0,0006 | 45,03 | 8663 | -0,0006 | -0,0006 | -0,0341 |
| 1 | 1 | 0 | 0,9993 | 0,9993 | 0 | 8588 | 0,0007 | 0,0007 | 0 |
| 0 | 1 | 0 | 0,0009 | 1 | 0 | 6855 | -0,0009 | 0 | 0 |
| 1 | 0 | 180 | 0,9994 | 0,0006 | -180 | 8637 | 0,0006 | -0,0006 | 360 |
| 0 | 0 | 0 | 0,001 | 0 | -360 | 7570 | -0,001 | 0 | 360 |
| 0 | 1 | 90 | 0 | 0,9991 | -270,1 | 7374 | 0 | 0,0009 | 360,1 |
| 0 | 1 | 0 | 0 | 0,9991 | -359,9 | 606 | 0 | 0,0009 | 359,9 |

In order to get these results, this test was realized ten times for each robot, and the error presented is the maximum error over all the tests, with the corresponding position estimation. All the values were rounded to four significant algorisms.

Upon evaluating the results, it is evident that the robot is consistently reaching the goal point and the desired orientation within the defined threshold. The error that can be seen on the theta error column is derived from the difference in the value of the angles, despite them being in the same position.

However, the physical prototype of the robot deviates slightly from the desired goal, particularly in terms of the x and y coordinates and the overall distance traveled. Although these deviations are relatively small, over time, they will accumulate, causing a severe deviation from the desired position.

These deviations can be attributed to hardware errors, such as inaccuracies in the encoder pulse measuring and the motor speed. These discrepancies in the physical execution of the robot's movements contribute to the observed variations in position accuracy.

Nevertheless, the rotation error is the most significant issue encountered during this test. Initially, the robot exhibits a small error in its rotational movement. However, as the test progresses and the robot performs more position transitions, the rotation error accumulates, which leads to even more significant deviations in the x and y coordinates, amplifying the position error. It is crucial to address this rotation error as it has a significant impact on the overall accuracy and reliability of the robot's navigation system.

This error is more prevalent in the differential and the tricycle robot, as these rely on rotation to move to a desired goal, as opposed to the double tricycle that, thanks to its omnidirectional movement, can move directly from its current position to the next without the need to change its orientation.

The observed deviations in the x and y coordinates and the accumulation of rotation error underscore the need for further improvements in the robot's hardware or the incorporation of additional sensors. Addressing these issues will enhance the overall accuracy and reliability of the robot's navigation system, enabling it to more effectively reach its desired positions and improve its performance in real-world scenarios.

These results also provide a time value for how long each robot took to reach the desired position. As can be seen, the fastest robot is the differential, despite the rotation around itself that is required for the robot to face the direction it will follow. This is due to the fact that it does not have to rotate the wheels to any position, which, due to hardware limitations, takes 1,5 seconds for each rotation. On the last test, since it was only asked to rotate 90º, the time it took was similar to the differential.

The second fastest is the double tricycle robot, since it does not have to rotate around itself to move, it only requires the final rotation to reach the desired position. However, due to the need to rotate the wheels, which as mentioned before, takes 1,5 seconds, it is slower than the differential.

The slowest robot is the tricycle, as it suffers from the disadvantages of both the previous robots, it must turn the front wheel, and must rotate around itself, therefore being significantly slower than both the other robots.

From the data collected during this test, some graphs were plotted, showcasing the coordinates and orientation of the robot over time.

Due to the sampling time of 50 milliseconds, the values used for the plots are not always accurate, however it is possible to understand the motion of the robot from them.

The path used for these plots can be seen in Figure 5.1. The robot began at the coordinates (0,0) facing the positive y-axis. It then moved to (1,1) and rotated to 90 degrees. After that, it proceeded to (0,1) and rotated to 0 degrees. Next, it traveled to (1,0) and rotated 180 degrees. Finally, it arrived at (1,1) and rotated back to 0 degrees.

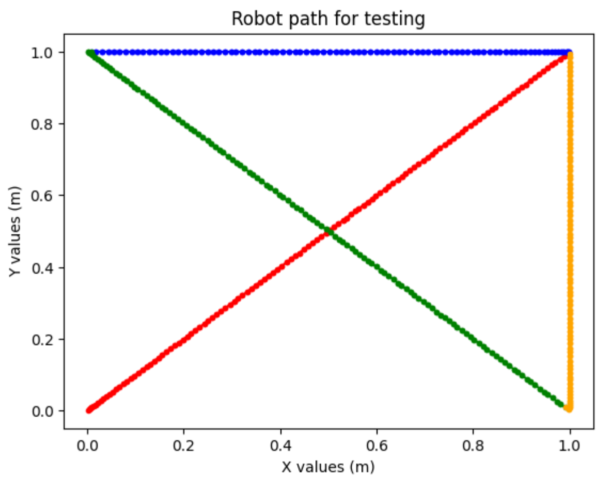
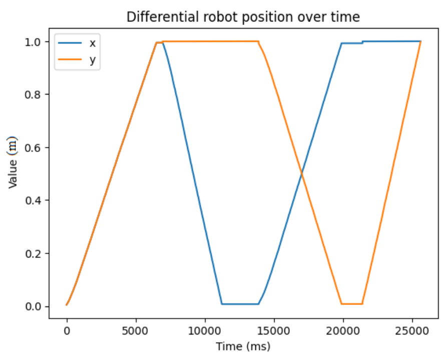
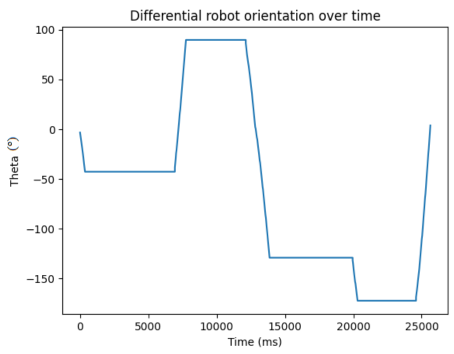


Figure 5.1: Path used for testing and generating plots, path order is red, blue, green, orange

The resulting graphs can be seen in Figures 5.2a to 5.4b:

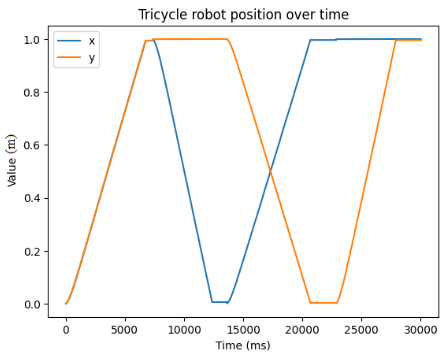


(a) Position of the differential robot over time

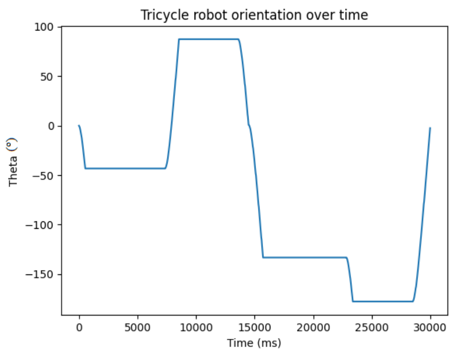


(b) Orientation of the differential robot over time

Figure 5.2: Graphs of position and orientation for the differential robot

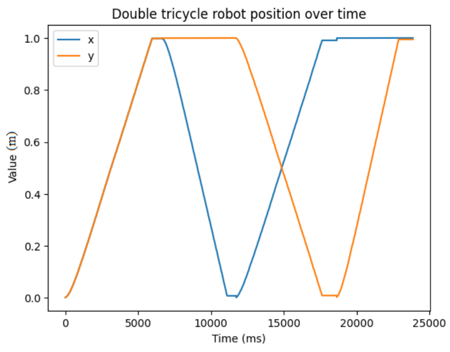


(a) Position of the tricycle robot over time

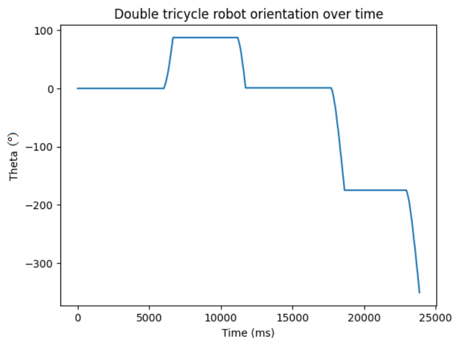


(b) Orientation of the tricycle robot over time

Figure 5.3: Graphs of position and orientation for the tricycle robot



(a) Position of the double tricycle robot over time



(b) Orientation of the double tricycle robot over time

Figure 5.4: Graphs of position and orientation for the double tricycle robot

Analysing these graphs it is possible to see how the robot moved.

In the position graphs, after each movement, the robot remains in the same position for some time, while it rotates to the final orientation. When looking to the orienation plots, it is possible to see that when the robot is moving, theta is constant. In the graphs, the first and third horizontal sections are bigger than the other two, since these are correspondent to the diagonals the robot traveled.

Both the differential and the tricycle robots have the necessity to rotate to face the direction they will travel, which implies extra rotations when compared to the double tricycle. These extra rotations can be seen, for example, in the first variation of theta to -45º. This rotation was made to travel to the point (1,1).

## 5.3 Results Discussion

In summary, the differential robot offers a range of trajectory possibilities, including rotation around itself, execution of curved trajectories, and linear motion in both forward and backward directions. Its advantages lie in its small and compact size, ease of control and programming, and simplicity. However, it is important to consider the limitations associated with trajectory control and maneuverability.

The tricycle robot offers few improvements to the differential robot while at the same time being more complex and less stable, therefore being a worse option for most scenarios. One advantage of the tricycle, however, is in the slippage category. Since it only has one wheel, if any slippage occurs, the direction of movement of the robot is known, as it is the direction the wheel is facing. In the other configurations, slippage is more unpredictable.

The double tricycle robot, however, offers an extensive array of trajectory possibilities, including all the ones possible for the differential and the tricycle and omnidirectional movement.

This omnidirectional mobility is its biggest advantage when compared to the other configurations, which makes it a good choice for most situations, despite its bigger size and complexity.

A comparison between these three configurations can be seen in Table 5.4, in which more "+" signs in a configuration means it has more of that category.

Table 5.4: Results Discussion

| **Categories** | **Differential** | **Tricycle** | **Double Tricycle** |
|---|---|---|---|
| **Movement Precision** | + | ++ | +++ |
| **Movement Time** | + | +++ | ++ |
| **Rotation Precision** | ++ | + | +++ |
| **Rotation Time** | + | +++ | ++ |
| **Trajectories** | ++ | ++ | +++ |
| **Control Complexity** | + | +++ | ++ |
| **Mechanical Complexity** | + | ++ | +++ |
| **Size** | + | ++ | +++ |
| **Stability** | +++ | ++ | +++ |
| **Slippage** | +++ | ++ | +++ |

These considerations contribute to a comprehensive evaluation of the robot's performance and suitability for specific applications.

# Chapter 6

# Conclusion

In this thesis, we set out to compare the trajectories and control capabilities of three different robot configurations: the differential, tricycle, and double tricycle. The primary purpose of this research was to gain insights into their navigational abilities and understand how their design and motion characteristics influence their performance in various scenarios.

Through the analysis of trajectories, we have observed distinct features and advantages associated with each robot configuration. The differential robot demonstrates versatility in its trajectory capabilities, allowing for precise turns, efficient point-to-point movements, and curved trajectories. Its small size and ease of control make it suitable for limited-space applications. However, maintaining accurate trajectory control and performing complex maneuvers may pose challenges for the differential robot.

The tricycle robot exhibits diverse trajectory capabilities, including performing tight turns and navigating through narrow pathways. Its single driving and steering wheel configuration enables precise movements. However, the tricycle robot's increased complexity and potential for instability require careful control strategies and system calibration.

The double tricycle robot surpasses the capabilities of the other configurations with its omnidirectional movement. It can move in any direction without rotation, offering enhanced navigational flexibility and efficiency. The double tricycle robot's ease of control and stability make it a practical choice for various applications. However, its increased mechanical complexity and larger size must be considered when deploying it in specific environments.

Overall, this research has provided valuable insights into the trajectory behavior of each robot configuration and has shed light on their strengths and limitations. The knowledge gained from this comparison can aid in selecting the most suitable robot for specific tasks and environments. This thesis also aims to contribute to the field of robotics education. By building and testing these robots, we have created valuable resources for future robotics classes. These robots can serve as practical tools to help students understand the concepts of robot control, trajectory planning, and navigation. They provide a hands-on learning experience bridging the theoretical knowledge gap and practical implementation.

In conclusion, comparing trajectories and control capabilities among the differential, tricycle, and double tricycle robots has provided valuable insights into their performance characteristics. These findings can guide the selection and utilization of robots in various applications, considering factors such as maneuverability, space constraints, and ease of control. Furthermore, the robots developed in this research can serve as educational tools to inspire and educate future generations of robotics enthusiasts and researchers.

## 6.1 Future Work

In this section, we will present some additional work that can be done in the future to improve upon what was already done.

### 6.1.1 Omnidirectional robot with Mecanum wheels

In order to make a valid comparison between robot models, their parameters, such as wheel diameter, must be equal. However, it proved difficult to find mecanum wheels that matched the dimensions of the wheels used for the other robots, and due to their complexity, 3D printing them to the correct size was not a viable option.

In the future, a prototype robot with mecanum wheels of the correct dimension should be built and compared to the previously studied models.

### 6.1.2 Code improvements

For further testing of trajectories and motions allowed by each model, different tasks can be programmed for the robot, for example, a Follow Line or Follow Square function that allows the user to test these trajectories specifically.

The self-localization and distance traveled by the robot, which are based on odometry, can also be improved by tuning the values used to transform the data from the encoders to meters.

### 6.1.3 Interface improvements

One of the main issues that can be resolved is improving the interface between the microprocessor and the user.

The webpage served by the Raspberry Pi Pico W should allow the user to insert the map that the robot will travel through instead of it only being accessible in the code written, meaning the robot must be connected to a computer in order to change the map.

This would facilitate testing many situations without requiring the robot to be turned off and plugged into a computer.

### 6.1.4 Sensors and online trajectory planning algorithms

The addition of sensors such as LiDARs, for example, is an important step to take in the future, as it would allow the robot to scan its environment and calculate its position much more accurately, as well as detect any objects that appear that were not detailed on the map.

This would also create the possibility of implementing other path-finding algorithms that would create the path as the robot travels through it, mapping the obstacles in the moment.

# References

[1] J Enrique Sierra-García and Matilde Santos. Mechatronic modelling of industrial agvs: a complex system architecture. *Complexity*, 2020:1–21, 2020.

[2] José Luis Guzmán, Manuel Berenguel, Francisco Rodríguez, and Sebastián Dormido. An interactive tool for mobile robot motion planning. *Robotics and Autonomous Systems*, 56(5):396–409, 2008.

[3] Steven M. LaValle. Bug algorithms. URL: http://msl.cs.uiuc.edu/~lavalle/cs497_2001/book/uncertain/node3.html.

[4] James Ng and Thomas Bräunl. Performance comparison of bug navigation algorithms. *Journal of Intelligent and Robotic Systems*, 50:73–84, 2007.

[5] Pedro Luís Cerqueira Gomes da Costa et al. Planeamento cooperativo de tarefas e trajectórias em múltiplos robôs. 2011.

[6] Hanlin Niu, Yu Lu, Al Savvaris, and Antonios Tsourdos. An energy-efficient path planning algorithm for unmanned surface vehicles. *Ocean Engineering*, 161:308–321, 2018.

[7] Dong-Hoon Yang and Suk-Kyo Hong. A roadmap construction algorithm for mobile robot path planning using skeleton maps. *Advanced Robotics*, 21(1-2):51–63, 2007.

[8] Omnia AA Salama, Mohamed EH Eltaib, Hany Ahmed Mohamed, and Omar Salah. Rcd: radial cell decomposition algorithm for mobile robot path planning. *IEEE Access*, 9:149982–149992, 2021.

[9] Charles W Warren. Multiple robot path coordination using artificial potential fields. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 500–505. IEEE, 1990.

[10] Zainah Md Zain, Hamzah Ahmad, Dwi Pebrianti, Mahfuzah Mustafa, Nor Rul Hasma Abdullah, Rosdiyana Samad, and Maziyah Mat Noh. *Proceedings of the 11th National Technical Seminar on Unmanned System Technology 2019: NUSYS'19*, volume 666. Springer Nature, 2020.

[11] Firas A Raheem, Mustafa M Badr, et al. Development of modified path planning algorithm using artificial potential field (apf) based on pso for factors optimization. *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, 37(1):316–328, 2017.

[12] Darci Luiz Tomasi and Eduardo Todt. Rotational odometry calibration for differential robot platforms. In *2017 Latin American robotics symposium (LARS) and 2017 Brazilian symposium on robotics (SBR)*, pages 1–6. IEEE, 2017.

[13] Jiyong Jin and Woojin Chung. Obstacle avoidance of two-wheel differential robots considering the uncertainty of robot motion on the basis of encoder odometry information. *Sensors*, 19(2):289, 2019.

[14] Ricardo B Sousa, Marcelo R Petry, Paulo G Costa, and António Paulo Moreira. Optiodom: a generic approach for odometry calibration of wheeled mobile robots. *Journal of Intelligent & Robotic Systems*, 105(2):1–22, 2022.

[15] Illah R Nourbakhsh and Roland Siegwart. Introduction to autonomous mobile robots, 2004.

[16] Steven M. LaValle. *Mobile Robotics An Information Space Approach*. 2013.

[17] Ksenia Shabalina, Artur Sagitov, and Evgeni Magid. Comparative analysis of mobile robot wheels design. In *2018 11th International Conference on Developments in eSystems Engineering (DeSE)*, pages 175–179, 2018. doi:10.1109/DeSE.2018.00041.

[18] Luigi Tagliavini, Giovanni Colucci, Andrea Botta, Paride Cavallone, Lorenzo Baglieri, and Giuseppe Quaglia. Wheeled mobile robots: State of the art overview and kinematic comparison among three omnidirectional locomotion strategies. *Journal of Intelligent & Robotic Systems*, 106(3):1–18, 2022.

[19] AndyMark. Mecanum wheel left. URL: https://www.andymark.com/products/10-in-mecanum-wheel-left-6-hole-bore.

[20] Ioan Doroftei, Victor Grosu, and Veaceslav Spinu. *Omnidirectional mobile robot-design and implementation*. INTECH Open Access Publisher London, UK, 2007.

[21] Autodesk. O que é o fusion 360? URL: https://www.autodesk.pt/products/fusion-360/overview?term=1-YEAR&tab=subscription.

[22] Raspberry Pi. Raspberry pi pico w datasheet, 2022.

[23] DFRobot. Tt motor with encoder (6v 160rpm 120:1). URL: https://www.dfrobot.com/product-1457.html.

[24] LewanSoul. Lx-16a bus servo user manual.

[25] BotnRoll. Pico4drive - placa de desenvolvimento para pi pico. URL: https://www.botnroll.com/pt/raspberry-pi/4722-pico4drive-placa-de-desenvolvimento-para-pi-pico.html.

[26] Inceptive Mind. All-in-one in-wheel motor tech to drive with a limitless-360 degree steering. URL: https://www.inceptivemind.com/protean360-one-wheel-motor-technology-drive-limitless-360-degree-steering/8158/.