

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Path Planning for Different Types of Robots in Docking and Obstacle Avoidance

Miguel Pinheiro Tavares

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Pedro Gomes da Costa

October 23, 2023

Resumo

Os robôs autônomos desempenham um papel crucial no auxílio ao ser humano em várias tarefas. No entanto, para garantir a execução segura de trajetórias autônomas, dois dos aspectos importantes a ter em consideração são: o carregamento da bateria e o desvio de obstáculos. Além disso, o percurso realizado por um robô pode variar significativamente com base no seu tipo específico e dimensões, mesmo quando começa e termina nos mesmos pontos da sua trajetória. Portanto, levar em conta essas variações é crucial para otimizar o planejamento de trajetórias e navegação do robô.

Um aspecto crítico desta pesquisa envolve o *base footprint* do robô, que tem uma importância significativa em várias tipologias de robôs, influenciando a sua navegação e manobrabilidade em ambientes diversos. A representação precisa do *base footprint* é essencial para o planejamento preciso de trajetórias, acoplagem e desvio de obstáculos para robôs omnidirecionais, diferenciais ou triciclos.

Esta tese de mestrado concentra-se no aprimoramento dos algoritmos de acoplagem e desvio de obstáculos de robôs móveis para navegação eficiente e segura em ambientes complexos. O estudo realizado apresenta algoritmos em modo offline e online, implementados em C++ e no *Robot Operating System (ROS)*, para enfrentar os desafios de acoplagem e desvio de obstáculos. No modo offline, o algoritmo reposiciona os vértices para garantir uma acoplagem segura, evitando colisões com paredes circundantes. Por outro lado, o algoritmo em modo online permite que o robô ajuste dinamicamente a sua trajetória para encontrar caminhos alternativos em tempo real, evitando colisões com obstáculos encontrados.

Os resultados demonstram a eficácia do algoritmo em modo offline, permitindo uma acoplagem eficiente ao manter uma distância ótima das paredes circundantes. Além disso, o algoritmo em modo online permite com sucesso a navegação dinâmica do robô, ajustando rapidamente a sua trajetória em resposta a obstáculos encontrados, garantindo assim o desvio de obstáculos e navegação ininterrupta.

Abstract

Autonomous robots play a vital role in assisting humans by performing various tasks. However, ensuring the safe execution of autonomous paths requires careful consideration of two key aspects: battery recharging and obstacle avoidance. Additionally, the path taken by a robot can vary significantly based on its specific type and dimensions, even when starting and ending at the same points in its trajectory. Therefore, accounting for these variations is crucial in optimising the robot's path planning and navigation.

A critical aspect of this research involves the base footprint of the robot, which holds significant importance across various robot typologies, influencing their navigation and manoeuvrability in diverse environments. The accurate representation of the base footprint is essential for precise path planning, docking, and obstacle avoidance for omnidirectional, differential, or tricycle robots.

This master's thesis focuses on enhancing mobile robot docking and obstacle avoidance algorithms for efficient and safe navigation in complex environments. The research introduces both offline and online mode algorithms, implemented using C++ and the Robot Operating System (ROS), to address the challenges of optimal docking and obstacle avoidance. In the offline mode, the algorithm readjusts the position of vertices to ensure safe docking while avoiding collisions with surrounding walls. On the other hand, the online mode algorithm allows the robot to dynamically adjust its trajectory to find alternative paths in real-time, avoiding collisions with encountered obstacles.

The results demonstrate the effectiveness of the offline mode algorithm, enabling efficient docking while maintaining an optimal distance from surrounding walls. Additionally, the online mode algorithm successfully empowers the robot to navigate dynamically, swiftly adjusting its path in response to encountered obstacles, thus ensuring obstacle avoidance and uninterrupted navigation.

Acknowledgements

I would like to express my heartfelt gratitude to my esteemed supervisor, Pedro Gomes da Costa for his unwavering guidance, insightful feedback, and continuous support throughout this research. His expertise and encouragement have been invaluable in shaping the direction and quality of this work.

Closely tied to the success of this research are the invaluable contributions from Paulo Rebelo for his invaluable assistance and support throughout this research. His constructive criticism and dedication were fundamental in enhancing the overall quality of this thesis.

Additionally, I am also deeply thankful to Héber Sobreira for his valuable contributions. His provision of access to crucial resources and essential data greatly contributed to the success and depth of this study.

Taking a moment to express my heartfelt gratitude, I extend my sincere appreciation to my friends and colleagues who have been a constant source of motivation and encouragement throughout this journey. Their support, camaraderie, and insightful discussions have enriched my academic experience.

I am deeply indebted to my family, especially my parents, for their unwavering love, understanding, and encouragement. Their continuous support and belief in my abilities have been the driving force behind my pursuit of academic excellence.

Last, but not least, I am grateful to all those who have contributed to this work in various ways, and their involvement has been invaluable in making this research possible.

Miguel Tavares

Contents

1	Introduction	1
1.1	Context	1
1.2	Problem Statement and Visual Representation	2
1.3	Goals and Objectives	3
1.4	Document Structure	4
2	Background and Fundamental Aspects	5
2.1	Robotic Field Overview	5
2.1.1	Robot Operating System - ROS	5
2.1.2	Robot Intercommunication Standards	6
2.2	Robot Classifications	7
2.2.1	Mobile Robot Designs	7
2.2.2	Base footprint and Base link	7
2.3	Path Planning	9
2.3.1	Classical methods	11
2.3.2	Heuristic methods	15
2.4	Docking systems	17
2.4.1	Docking station localisation methods	17
2.4.2	Docking station recharging methods	28
2.5	Obstacle avoidance systems	32
2.5.1	Obstacle avoidance methods	32
3	Literature Review	37
3.1	Obstacle avoidance: Path planning with graphs	37
3.2	Overview of the Project	38
4	System Architecture and Nodes Representation	41
5	Methodology and Algorithms Developed	43
5.1	Optimising Critical Point's Placement (Docking)	43
5.1.1	Preparatory Steps and Considerations	43
5.1.2	Algorithm performed	48
5.2	Algorithm for obstacle avoidance	50
5.2.1	Preparatory Steps and Consideration	50
5.2.2	Algorithm performed	51

6	Experimental Results and Analysis	55
6.1	Readjustment of critical point's (docking) position	55
6.1.1	Test 1: Testing for different robot's areas - Circle vs. Rectangle	56
6.1.2	Test 2: Extensive testing for all available possible positions: different sizes	61
6.1.3	Test 3: Corner and closed space with different orientations and sizes . . .	66
6.2	Obstacle avoidance	69
6.2.1	Setup	69
6.2.2	Results	70
6.2.3	Analysis	72
7	Conclusion and Future Work	75
	References	79
A	Results	83
A.1	Tables	83

List of Figures

1.1	Docking station appears to be well placed, but in this position, the robot cannot charge	2
1.2	Docking station properly placed	3
2.1	ROS communication structure [1]	6
2.2	Examples of classical methods for path planning	11
2.3	Potential field method representation	12
2.4	Visibility Graph (<i>left</i>) and Voronoi Graph (<i>right</i>)	12
2.5	Cell decomposition method representation	13
2.6	Vector field histogram method representation.	13
2.7	Paths of Bug 1 and Bug 2 algorithms (a) No obstacle (b) Bug 1 algorithm (c) Bug 2 algorithm [2]	14
2.8	Rapidly exploring random tree method representation	14
2.9	The diagram of the final docking process when the robot is detected by one of the IR sensors on the docking station. (a) The robot is detected by one of the IR sensors. (b) The robot is detected by both sensors. (c) The robot turns 90 degrees to face the docking station. (d) The robot connects itself to the docking station. [3]	18
2.10	IR distance sensors [4]	19
2.11	Robot’s views from: left, centre and right [4]	19
2.12	The global navigational space of the prototype robot [5]	20
2.13	Robot Moving in a Square path [6]	20
2.14	The markers used in this work: (a) ARTag, (b) AprilTag, (c) ArUco and (d) STag [7]	21
2.15	Overview of the experimental setup with the two axes gimbal on the right side of each image, where (a) the marker position is altered in the longitudinal direction and (b) the marker position is altered in the lateral direction. [7]	22
2.16	The AprilTag marker used (a) under normal lighting conditions and (b) with a shadow being cast as seen from the Logitech camera. [7]	22
2.17	The STag multi-tag bundles (a) with two equal sized markers and (b) with three differently sized markers [8]	23
2.18	The two marker bundle in the non-planar configuration. The marker is rotated about the vertical axis [8]	23
2.19	In the searching zone, the robot locates itself through ORBSLAM; A, B, C represent different results of searching the approaching point; In the approaching zone, the AprilTag localisation is adapted to calculate the relative position of charging device. [9]	25
2.20	ArUco Marker Detection [10]	25
2.21	Fitting results of the robot to the surrounding environment information [11]	26
2.22	Reflective tape [12]	27

2.23	Laser intensity value graph [12]	27
2.24	Human blocking the route between the robot and the docking station [12]	28
2.25	CAD model of the proposed surveillance robot and the docking station. [3]	29
2.26	Dock connector [4]	29
2.27	Docking station [4]	30
2.28	The Power Management System monitoring circuit [5]	30
2.29	IR transmitter and docking station [6]	30
2.30	Top view of automatic docking robot [6]	31
2.31	(a) The docking and charging station. (b) The robot docking mechanism robot we developed [9]	31
2.32	Artificial potential field model [13]	32
2.33	Modified artificial potential field model [13]	33
2.34	Collision-free path planning and tracking error [14]	33
2.35	Comparison of objective function computation between FDEDWA and the existing DWA [15]	34
3.1	Vertex representation in RViz	39
3.2	Edge representation in RViz	39
3.3	Vertex containing bigger robot	39
3.4	Vertex containing smaller robot	39
4.1	Representation of developed nodes and topics	41
5.1	Position of the base footprint for each robot	45
5.2	Robot's Area: Circular Shape	46
5.3	Robot's Area: Rectangular Shape	46
5.4	Vertex's orientations	47
5.5	Robot positioned according to vertex's orientations	48
5.6	Visual representation of the algorithm	50
5.7	First obstacle detection algorithm	51
5.8	First algorithm designed for obstacle avoidance from the top	51
5.9	First algorithm designed for obstacle avoidance from the bottom	52
5.10	Test Environment Walls	52
5.11	Final obstacle detection algorithm	53
5.12	Final algorithm designed for obstacle avoidance from the top	53
5.13	Final algorithm designed for obstacle avoidance from the bottom	54
6.1	Original trajectory in RViz	55
6.2	Testing Environment	56
6.3	New configuration for Vertex 26	67
6.4	New configuration for Vertex 101	67
6.5	Alternative Path	69

List of Tables

2.1	Different robot designs	8
2.2	Main differences between global and local path planning	10
2.3	Comparison of Various Navigation Algorithms	16
2.4	Experimental comparison summary: Main advantages and disadvantages of the four evaluated packages [8]	24
2.5	Experimental results for the first set of experiments [12]	27
2.6	Experimental results for the second set of experiments [12]	28
6.1	Circle vs Rectangle - Left Side	57
6.2	Circle vs Rectangle - Right Side	58
6.3	Circle vs Rectangle - Middle Section	59
6.4	Circle vs Rectangle - Console Output	60
6.5	Different sizes: Left side	62
6.6	Different sizes: Right side	63
6.7	Different sizes: Middle section	64
6.8	Different sizes: Console Output	65
6.9	Different orientations for vertex 26	68
6.10	Different orientations for vertex 101	68
6.11	Vertices different orientations - Console Output	68
6.12	Different tested cases - Robot moving forward	71
6.13	Different tested cases - Robot moving backwards	71
A.1	Different sizes: Middle section (extended version)	83
A.2	Different sizes: Left side (extended version)	84
A.3	Different sizes: Right side (extended version)	85
A.4	Different sizes: Console Output (extended version)	86

Abreviaturas e Símbolos

AGV	Autonomous Guided Vehicles
AIV	Autonomous Intelligent Vehicles
AMR	Autonomous Mobile Robots
CPU	Central Processing Unit
DW4DO	Dynamic Window for Dynamic Obstacles
DWA	Dynamic Window Approach
FDEDWA	Finite Distribution Estimation-based Dynamic Window Approach
FMF	Finite Memory Filtering
IR	Infrared
IRR	Infrared Receiver
IRT	Infrared Transmitter
LIDAR	Light Detection and Ranging
LMPPC	Local Model Predictive Contouring Control
LS	Limit switch
MPC	Model Prediction Controller
MQTT	Message Queuing Telemetry Transport
RMF	Robotics Middleware Framework
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
SLAM	Simultaneous Localisation And Mapping
Std Dev	Standard Deviation
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UWB	Ultrawideband

Chapter 1

Introduction

1.1 Context

Nowadays, autonomous guided vehicles (AGV), autonomous mobile robots (AMR), and, more recently, autonomous intelligent vehicles (AIV) increasingly represent an important role in performing tasks with less added value for humans. Considered robust, flexible, and accurate platforms, they enable seamless integration into different industries, environments, and scenarios. However, when these are complex, i.e. when there is a need for the robot to navigate in environments and/or spaces with the presence of humans, mobile industrial machinery, for example, forklifts, and where there are areas for cargo management, the need to detect these possible obstacles increases. In addition, mobile robotics in complex and unstructured environments (where natural contours are not enough) is under constant exploration. Research concerning the docking of the different mobile platforms at different locations is constantly being developed (charging stations, workstations, trailers, and others). Presently, there are already different solutions, from mechanical to software-based, that allow mobile robots to perform their functions with the desired precision for each task.

The context of this research is driven by the increasing demand for mobile robotics applications in industries such as logistics, manufacturing, and healthcare. The adoption of autonomous guided vehicles (AGVs) and other mobile robotic systems is accelerating due to the advantages they offer in enhancing productivity, reducing operational costs, and ensuring a safer work environment. As these robots operate in real-world environments, the need for sophisticated algorithms that can adapt to dynamic and unpredictable situations becomes paramount. This thesis aims to contribute to the advancement of mobile robotics by developing innovative solutions that enable seamless and efficient navigation in different environments. Moreover, with the growing emphasis on Industry 4.0 and smart automation, the deployment of mobile robots for material handling, goods transportation, and warehouse management is expected to further increase in the coming years. The proposed algorithms in this thesis aim to bridge the gap between theoretical research and practical implementation, providing real-world solutions that can be integrated into existing robotic systems with ease. By addressing the challenges of precise docking and obstacle avoid-

ance, this work contributes to the broader vision of achieving more autonomous and intelligent mobile robotics applications across diverse industries.

1.2 Problem Statement and Visual Representation

Ensuring a robot's awareness of its physical space is of extreme importance to prevent real-life issues, particularly during crucial trajectory points like docking. Unfortunately, simulations often neglect to consider the actual dimensions of the robot, leading to potential problems. These issues may include the mispositioning of docking charging stations too close to obstacles or at angles that impede proper docking, especially if the robot is too large. Therefore, this project aims to rectify this limitation by adjusting the position of all critical points in the simulation while taking into account the robot's actual dimensions.

During simulation, a vertex may appear to be suitably positioned, seemingly avoiding any collision with the wall. However, in reality, the ability of the robot to dock successfully depends on its size and type. In some cases, the docking station might be placed too close to the wall, impeding the robot's docking process.

Figure 1.1 presents an illustrative example to provide a clearer understanding of this issue. It showcases how a seemingly well-positioned vertex in the simulation might lead to unexpected collisions or docking difficulties in real-world scenarios. This discrepancy highlights the importance of accurately accounting for the robot's dimensions and type during the vertex placement process, ensuring seamless docking operations even in complex environments with proximity to walls.

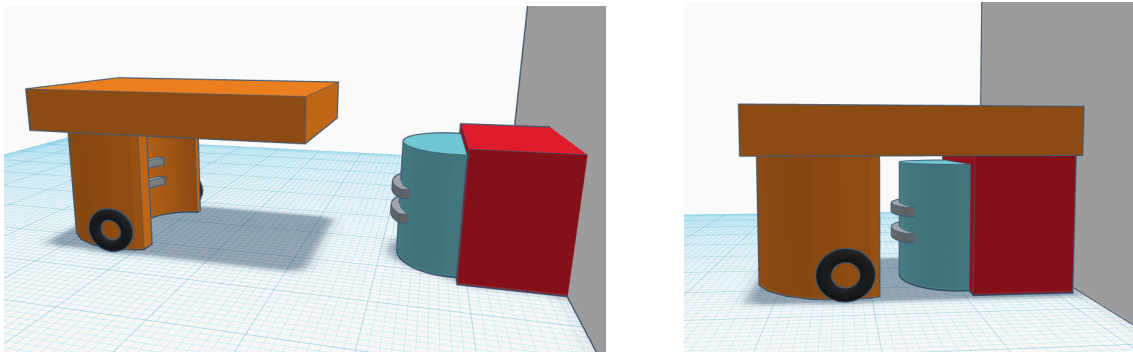


Figure 1.1: Docking station appears to be well placed, but in this position, the robot cannot charge

In this particular scenario, despite the docking station appearing to be appropriately positioned for the docking process, it becomes evident that an adjustment to the dock's position is required. To enable the robot to dock seamlessly, the station needs to be relocated further away from the wall, as shown in Figure 1.2. This relocation ensures that the robot has sufficient clearance to complete the docking operation successfully and avoids potential collisions or obstructions during the process. The precise adjustment of the docking station's position is essential to optimise the robot's navigation and docking performance, particularly in complex environments where obstacles and wall proximity can pose significant challenges.

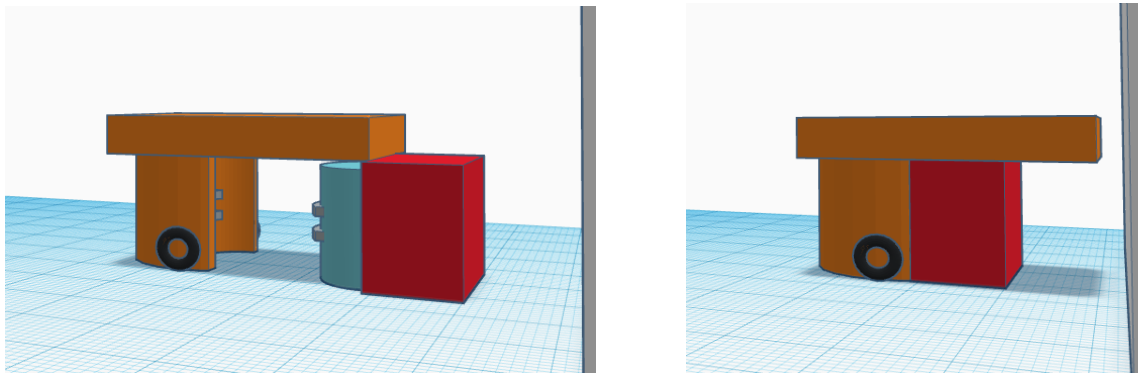


Figure 1.2: Docking station properly placed

Moreover, this thesis addresses the challenge of obstacle avoidance in real-time through an online mode algorithm. The goal is to enable the robot to find alternative paths and avoid collisions with obstacles along its trajectory. By analysing the environment in real-time, the robot can adjust its path, ensuring safe and efficient movement even in the presence of unexpected obstacles.

Additionally, both algorithms consider the robot's specific type (omnidirectional, differential, or tricycle) and adjust its behaviour accordingly, ensuring an effective docking process and obstacle avoidance. The implementation and evaluation of these algorithms will be performed through simulation, analysing their performance in various scenarios to demonstrate their effectiveness and practical applicability.

1.3 Goals and Objectives

The core focus of this master's thesis, as mentioned before, revolves around tackling two fundamental challenges in mobile robotics: docking and obstacle avoidance. The primary goal is to develop and evaluate novel algorithms that address these critical aspects of robot navigation.

The first goal involves creating an offline mode software algorithm using C++ and ROS to readjust the position of docking vertices. This algorithm aims to ensure collision-free docking of the robot with precision, allowing seamless integration into complex environments and spaces with the presence of obstacles and other machinery.

The second objective centres on designing an online mode obstacle avoidance algorithm, which allows the robot to detect obstacles and efficiently find alternative paths, avoiding collisions as it moves through its trajectory, ensuring safe and adaptive obstacle avoidance in various scenarios.

The study will consider different types of robots, including omnidirectional, differential, and tricycle, to validate the algorithms' efficiency and effectiveness across diverse mobile platforms.

The ultimate goal is to enhance the robot's autonomy and efficiency in navigation, facilitating its integration into diverse industrial and complex environments. Through the successful development and evaluation of these algorithms, this research aims to contribute valuable insights to the field of mobile robotics, facilitating their integration into various industries and applications.

1.4 Document Structure

This dissertation is comprised of eight chapters.

The first chapter corresponds to the **Introduction**, which includes a general framing of the issue and a study overview.

During Chapter 2, **Background and Fundamental Aspects**, robotic systems are explored, offering insights into various aspects such as an overview of the robotic field, along with robot classifications depending on their drive and steering, some of the most popular path planning methods, and a comprehensive review on docking systems. Additionally, diverse methods and approaches for obstacle avoidance are presented.

Throughout Chapter 3, **Literature Review**, a comprehensive investigation of relevant literature is conducted to lay a robust groundwork for the research. The primary emphasis lies in exploring algorithms previously developed by other researchers, providing valuable insights into the methodologies employed in those algorithms. Through this comprehensive review, promising opportunities for improving the current project are revealed.

In Chapter 4, **System Architecture and Nodes Representation**, the focus is on establishing the correlation between the newly developed nodes and topics, while also examining how they seamlessly integrate into the existing architecture.

Then, in Chapter 5, **Methodology and Algorithms Developed**, the dissertation outlines the specifics of the innovative offline mode software algorithm aimed at readjusting docking vertices, along with the online mode algorithm designed for obstacle avoidance. Extensive testing and rigorous analysis are conducted to thoroughly evaluate the performance of these algorithms in diverse scenarios, offering valuable insights into their efficiency and overall effectiveness.

Following, in Chapter 6, **Experimental Results and Analysis**, the dissertation presents the outcomes of various experiments conducted to assess the performance of the developed algorithms. A detailed analysis of the experimental results is undertaken, shedding light on the algorithms' strengths and potential areas for improvement.

Next, in Chapter 7, **Conclusion and Future Work**, the study provides a comprehensive summary of the research's outcomes and essential contributions in the domain of mobile robotics algorithms. A thorough conclusion is presented, encapsulating the achievements and valuable insights obtained during the research. Furthermore, the chapter outlines potential avenues for future investigations and advancements, offering valuable guidance for researchers and practitioners interested in furthering the field.

Lastly, the **References** and **Annexes** sections contain supplementary information on certain contents.

Chapter 2

Background and Fundamental Aspects

This chapter presents a comprehensive exploration of the foundational principles and key aspects that underpin the research in the field of mobile robotics. A broad overview of essential subjects is provided, offering valuable context to the work. Subsequently, the following chapter delves into a more specific and detailed analysis of these subjects.

Understanding the background knowledge and fundamental concepts in mobile robotics is crucial to grasp the significance and implications of the research. By laying a solid groundwork, the aim is to pave the way for the development and evaluation of innovative algorithms that address critical challenges in robot navigation, path planning, docking systems, and obstacle avoidance.

With a broader understanding of these essential aspects, the research will be better equipped to explore and contribute to the advancement of mobile robotics, particularly in dynamic and complex environments.

2.1 Robotic Field Overview

Nowadays, robots find widespread application in numerous fields, such as healthcare, warehouse automation, manufacturing, transportation, and surveillance, among others. Therefore, having a framework that facilitates its development and ensures continuous evolution in this field becomes crucial. In addition, in certain environments, the various robots used for different purposes are not fabricated by the same provider. Hence, ensuring the intercommunication between those robots is crucial for maintaining smooth workflow integration, as well as managing robot traffic in a warehouse, for example, [16]. In the next sections, one of the most popular frameworks is described, as well as three important robot intercommunication standards.

2.1.1 Robot Operating System - ROS

Robot Operating System (ROS) stands out as one of the most popular frameworks used in robot applications, and is frequently addressed in the research documents read, studied and referenced. Its versatility, accessibility, and widespread use in various methods and algorithms position it as an essential subject of study. ROS [1] is a free and open-source framework, backed by a robust web

community, dedicated to facilitating the development of robotic applications. The architecture of this network is a peer-to-peer topology, using the TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) [17]. ROS communication structure is presented in Figure 2.1.

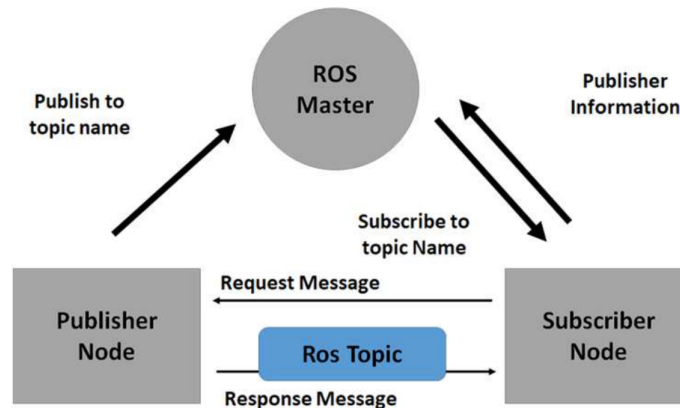


Figure 2.1: ROS communication structure [1]

The main ROS concepts revolve around nodes, messages, topics and services. A node represents a process that performs computation [1]. Nodes communicate with each other by publishing a message on a topic. A service, on the other hand, is a set of two defined messages: one for making requests, and one for providing responses. One of the notable advantages of this framework is its support for multiple programming languages, including C++, Python, LISP, and Octave, being the first two the most popular. Its flexibility in language choice allows developers to work with their preferred programming tools. It is designed with a micro-kernel architecture, providing various tools and commands to facilitate different tasks. This compromises efficiency in return of stability and complexity management.

2.1.2 Robot Intercommunication Standards

In this section, three robot intercommunication standards will be approached: the Robotics Robotics Middleware Framework (RMF), VDA 5050 and MassRobotics Interoperability Standard [16]. Firstly, RMF is a comprehensive standard that enables interoperability not only between robotic systems but also with facility resources such as elevators and doors. It is built upon a set of libraries and tools from ROS 2, which enhances its compatibility and effectiveness. Secondly, VDA 5050 is specifically designed for autonomous guided vehicles and it allows the exchange of information about status and orders with a master control system. It uses the Message Queuing Telemetry Transport (MQTT), which is lightweight publish/subscribe messaging transport protocol, to enable efficient communication. The MassRobotics Interoperability Standard, on the other hand, is lightweight and flexible, since it does not require a specific platform to work with, unlike RMF, which relies on ROS 2. Furthermore, it addresses different task allocation, since it was designed for AMR, and it requires less information about route configuration than VDA 5050, which was specifically designed for AGVs.

2.2 Robot Classifications

2.2.1 Mobile Robot Designs

To properly evaluate different behaviours in different fields, as for path planning in docking and obstacle avoidance, it is essential to analyse the outcomes for different robots. This approach enables a more extensive assessment of diverse robotic capabilities and functionalities. Each robot design has its unique set of strengths and weaknesses, as well as performance characteristics. By employing a variety of robots in various testing scenarios, the conducted evaluations can yield more robust and improved results in different configurations. When considering different types of robots, they can be categorised according to their drive mechanisms, manoeuvring capabilities, locomotion systems, applications, traction, and design, among others [18, 19]. According to the literature [18, 19], there is a wide range of mobile robot designs targeted to specific applications and environments. These designs can be divided into driving robots, omnidirectional robots, balancing robots, walking robots, autonomous planes, autonomous vessels and underwater vehicles, robot manipulators and simulation systems. Both driving robots and omnidirectional robots play pivotal roles in the field of robotics. Understanding these classifications is essential as they lay the foundation for creating efficient and specialised robots for diverse tasks.

In this section, a more detailed explanation of both driving robots and omnidirectional robots will be provided. Driving robots play a significant role in various fields, making them indispensable and impactful in modern society. In Table 2.1, several distinct designs will be presented, each equipped with specific drive systems that influence their manoeuvrability and performance.

2.2.2 Base footprint and Base link

The concepts of base footprint and base link hold outstanding significance in the realm of robotics, especially when dealing with diverse types of robots. In the context of ROS, the base link is a key frame of reference or coordinate frame used to represent the physical body of the robot. It typically denotes the geometric centre or reference point of the robot's chassis or the robot's mass centre. The base link plays a vital role in defining the robot's pose (position and orientation) in three-dimensional space. On the other hand, the base footprint refers to the two-dimensional projection of the robot's physical base on the ground plane. Conventionally, the footprint of a mobile robot refers to the area covered by the robot when it is stationary [21, 22]. In simpler terms, it represents the two-dimensional view of the base link from the top, projecting in the ground plane ($z=0$), with no consideration of the robot's actual physical body. Comparing the base footprints of various robot types is crucial as it enables to grasp how the physical design of each robot affects its navigation capabilities and overall performance. Subjecting various robot types to path-planning algorithms and obstacle avoidance strategies enables a thorough evaluation of their behaviour. This analysis allows discerning how distinct base footprints, tailored for different robots, can yield diverse results when implementing the same algorithm. The robot's footprint is defined as the maximum area within the camera's field of view that can be inspected [21]. Traditionally, it is

Single Wheel Drive
This unique configuration utilises a single wheel as the driving and steering mechanism, providing simplicity in its conception and ease of navigation. There are also two passive caster wheels in the back of the robot since at least three contact points are always required to ensure stability and balance during movement [18].
Differential Drive
This design employs two independently powered wheels, completing the minimum three ground contact points with one or two additional wheels [18].
Tracked Robots
This design can be viewed as a specialised variant of the wheeled robot, boasting improved terrain manoeuvrability and enhanced stability due to its tracked design. The presence of multiple points of contact with the surface provides higher traction and agility, rendering it well-suited for navigating challenging environments [18].
Synchro-Drive
This innovative design incorporates three wheels that are all being driven and steered together. These synchronised wheel movements enable unique locomotion capabilities by coordinating the rotation of multiple wheels. This configuration grants the robot almost holonomic abilities since it can drive in any desired direction. However, unlike those types of robots, it requires stopping and realigning its wheels when transitioning from forward to sideways motion [18].
Ackermann Steering
In this design, the two rear wheels are designed for driving, while the two front wheels are designed for steering. Compared to the differential drive, this approach allows for smooth straight driving, as there are no issues with veering off course. However, one limitation is that the vehicle cannot turn on the spot, meaning it cannot rotate independently in place. Additionally, during curves, the rear driving wheels may experience slippage, which can affect their handling and manoeuvrability [18].
Omnidirectional Mobile Robot (OMR)
The omnidirectional mobile robot (OMR) possesses a remarkable ability to execute holonomic motion, enabling it to move in any direction without the need to change its heading direction. This unique characteristic empowers the OMR to navigate effortlessly even in tight and intricate spaces. Unlike other wheeled robots that require changing their heading direction to follow curves, the OMR can achieve smooth, continuous motion along intricate paths without any turning manoeuvres. This exceptional capability allows the OMR to navigate with ease and precision, even along accentuated curves, making it highly effective for tasks that demand agile and precise movement in constrained environments [20, 18].

Table 2.1: Different robot designs

represented by a rectangle, although alternative representations include a circle [23] or a variable footprint [24]. For the purpose of this thesis and to maintain coherence with the provided source code, a rectangular footprint is considered for all types of robots. In the case of differential and omnidirectional robots, the footprint is located at the centre of the robot, while for tricycle robots, it is positioned in the middle of the back wheels.

2.3 Path Planning

Mobile robots were initially limited to manufacturing industries but are now widely used in agriculture, medicine, mining, rescue missions, and more [25, 2, 26]. Autonomous navigation is crucial for intelligent vehicles, involving position specification, path planning, and following [25, 27, 28]. While performing the task of navigation, robots must extract environmental information and take the necessary action required to plan feasible collision-free paths to reach their goals [2]. Therefore, it is equipped with many intelligent equipments, which is required to map the environment, localise its position, control the motion, detect obstacles, and avoid them by using navigational techniques [26]. This process involves four key components: perception, the robot uses its sensors to perceive and extract meaningful information about its environment, including past movements, directions and locations; localisation, the robot determines its current position in the working space; cognition and path planning, the robot decides how to steer and identify the target direction required to reach its goal; motion control, the robot regulates its movements to follow the desired trajectory [29, 27, 28]. In recent years, mobile robot technology has advanced rapidly, incorporating intelligent features due to developments in electronics and information technology. Path planning is a crucial aspect of mobile robot research, with the goal of efficiently avoiding obstacles during robot movement and swiftly calculating the shortest and safest route to the target point, saving time and conserving energy [29, 2, 30]. Path planning involves obtaining a collision-free route between start and goal points, using translation and rotation while avoiding obstacles in the working environment [31, 32, 2, 27, 28]. Mobile robots use sensors to gather information about the surrounding environment and their own state, enabling obstacle avoidance and movement towards the target point [31, 29, 30, 28]. In robot path planning, there are multiple feasible paths from the start to the target location. The best path is chosen based on criteria such as shortest distance, smoothness, or minimum energy consumption. Typically, the shortest distance with minimal time is the most preferred criterion [32]. Safe path planning, which involves detecting and avoiding obstacles, is a critical function in any navigational technique, whether fully or partially automated. It is considered one of the essential problems in robotics. Therefore, selecting the appropriate navigational technique is a crucial step in robot path planning [27, 26]. Path algorithms are essential for ensuring the safety and efficiency of robot and autonomous system movements. When selecting a method for path planning, several factors must be considered, including the intended optimisation type, such as path length, trajectory execution time, energy consumption, or other variables. Computational complexity is another critical aspect, as some

Global path planning	Local path planning
Work offline	Work online
Robot Map-based	Robot Sensor-based
Deliberative navigation	Reactive navigation
Workspace area is fully known and the terrain should be fixed	Workspace is not necessarily fully known or unknown
The algorithm produces a whole path from the initial point to the target point before the robot begins its movement	The algorithm produces a new pathway in echo to environmental moveable
Approximately slower response	Fast response

Table 2.2: Main differences between global and local path planning

theoretically demonstrated methods may be impractical to implement due to memory limitations or high execution times.

Generally speaking, path planning algorithms can be broadly categorised into two types: global or offline path planning and local or online path planning, based on the accessibility of environment information. In global path planning, the robot has access to a full representation of the environment, including information about all obstacles such as size, shape, and orientation, before it starts navigating. On the other hand, in local path planning, most of the environment information is unknown to the robot at the start, and the map is constructed or updated during navigation using on-board sensors [31, 29, 2]. Offline path planning initially plans the path offline but can switch to online mode when encountering new changes in the obstacle scenario [25]. Global planning methods show limited applications due to less robustness in terrain uncertainty and since they usually generate a low-resolution, high-level path based on known environmental map or its current and past perceptive information of the environment, whereas, local path planning methods show more flexibilities in partially known or unknown environments and provide an optimised paths [32]. The global path planning approach will generally use some form of optimisation technique in order to maximise the efficiency and performance of the search. The main constraint, however, is that this approach cannot be relied upon in a dynamic environment that contains multiple moving objects along its path [2]. On the other hand, local path planning algorithm does not need a priori information of the environment. It usually gives a high-resolution low-level path only over a fragment of global path based on information from on-board sensors. It works effectively in dynamic environments. The method is inefficient when the target is long distance away or the environment is cluttered. Normally, the combination of both methods is advised to enhance their advantages and eliminate some of their weaknesses [28]. The differences between these methods is summarised in table 2.2.

It can be further categorised as the classical approach and heuristic approach (Artificial Intelligence Technique).

2.3.1 Classical methods

In classical methods either a solution would be found or it would be proven that such a solution does not exist. The main disadvantage of such methods is their computationally intensiveness and their inability to cope with uncertainty. Such disadvantages make their usage brittle in real-world applications. This is due to the natural characteristics of such applications which is being unpredictable and uncertain. Some classical methods are represented in Figure 2.2, and a description of some of them is provided.

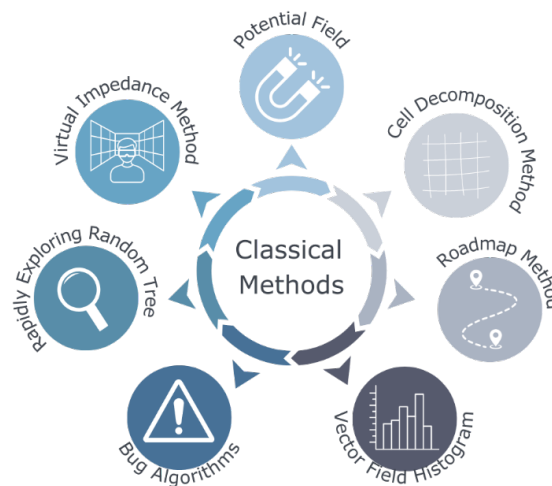


Figure 2.2: Examples of classical methods for path planning

2.3.1.1 Potential Field Method

The potential field method comes from the concept of potential field in physics, which regards the movement of objects as the result of two kinds of forces. It creates a virtual potential field, where the goal point creates a potential attractive force and obstacles have potential repulsive forces. The agent navigates along the gradient of the potential field towards the target point, under the action of the two forces. However, potential field methods may suffer from local minima problems. This method faces robot-blocking issues in two cases: obstacle between robot and destination with stronger repulsion than attraction, hindering progress or combined forces resulting in zero movement due to local minimum. Some ways address this problems include the use of potential fields that only have local minima at the destination and the implementation of techniques to escape from the local minima [31, 25, 29, 32, 2, 27, 30, 26].

2.3.1.2 Roadmap Method

The roadmap algorithm uses nodes and links between nodes that can have a physical meaning, i.e., the nodes may represent a location and the links correspond to the path between these locations [31, 25, 29, 32, 2, 27, 30]. It builds a collision-free network of paths from the robot's initial position

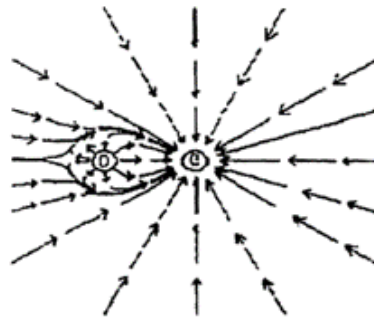
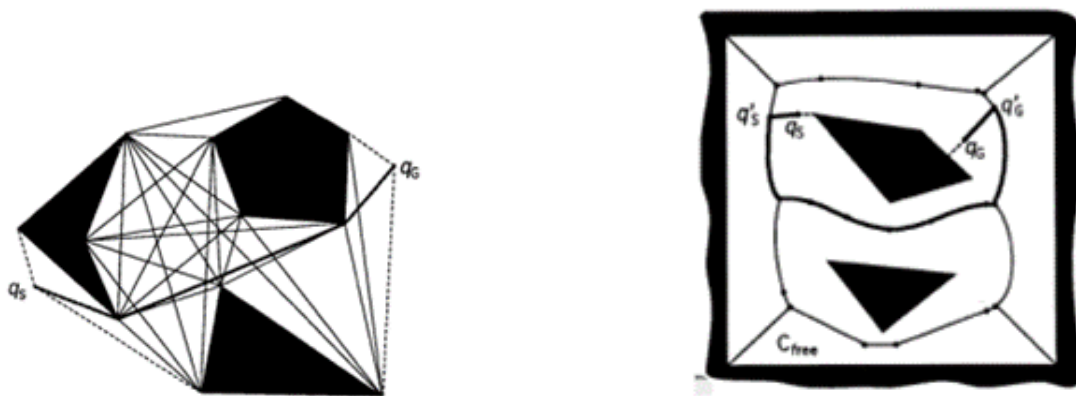


Figure 2.3: Potential field method representation

to the goal. This approach transforms the trajectory planning into a graph research problem, which can be solved using standard graph search techniques. The main challenge in this approach is the construction of the roadmap. Some techniques used to construct the roadmap are the visibility graph and the Voronoi Diagram [31, 25, 29, 32, 2, 27, 30]. The visibility graph connects visible vertices of polygonal obstacles present between start and target points, finding the shortest path and enabling efficient pathfinding in sparse environments using straight lines. It is commonly applied in 2D Cspaces with nodes as the vertices of the obstacles and links only between mutually visible nodes, i.e., there is a straight line joining the vertices and does not intersect any obstacle. On the other hand, the Voronoi Diagram is created using equidistant points between multiple obstacles, resulting in a safer path, though not always the shortest. It forms regions where each contains only one obstacle, and any point within a region is closer to its obstacle than to any other. Unlike the Visibility Graph, paths generated by the Voronoi Diagram tend to be farther away from obstacles [31, 25, 29, 32, 2, 27, 30].

Figure 2.4: Visibility Graph (*left*) and Voronoi Graph (*right*)

2.3.1.3 Cell decomposition

The cell decomposition approach divides the robot's workspace into non-overlapping regions or "cells." It identifies free and occupied areas. An availability chart is created by determining con-

tiguous cells. Relationships between adjacent cells are calculated to form a collision-free path from start to goal. The environment is divided into larger rectangles, each continuous. If a large rectangle contains obstacles or boundaries, it's subdivided into smaller rectangles. The process continues until a solution is reached at the boundaries [25, 29, 32, 2, 27, 30].

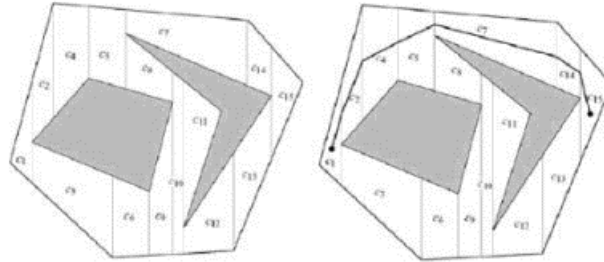


Figure 2.5: Cell decomposition method representation

2.3.1.4 Vector Field Histogram

The vector field histogram (VFH) is a real-time obstacle avoidance technique for robots. At every instant, a polar histogram is created to show obstacle density around the robot. The robot's steering direction is determined by selecting the direction with the least obstacle density and the shortest distance to the goal. As the environment changes, the polar histogram is regenerated regularly to adapt to new obstacle positions. It uses a 2D Cartesian histogram grid as a world model, continuously updated with data from on-board range sensors. The method involves a two-stage data reduction process to compute control commands. The first stage creates a 1D polar histogram from a fixed subset around the robot's location, with each sector representing obstacle density. The second stage selects the sector with the lowest obstacle density to align the robot's heading. This method is considered to be an improvement of the potential field method, allowing robots to detect and avoid unknown obstacles in real-time while pursuing their goals [25, 2, 27].

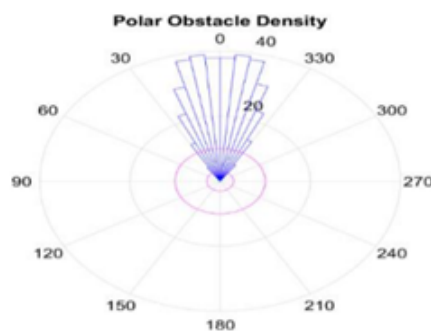


Figure 2.6: Vector field histogram method representation.

2.3.1.5 Bug Algorithms

Bug algorithms are used when the robot needs to reach a destination without prior knowledge of the global environment, and when maps are not built. The robot relies on sensors to explore the unknown environment. In Bug 1, when an obstacle is detected along the path to the goal, the robot follows the contour of the obstacle in a full cycle, while also trying to estimate the shortest distance to the goal. Bug 2 is more efficient, as it does not require a full cycle around the obstacle. Instead, it computes an initial path and stores its slope, switching to obstacle avoidance mode when needed, and returning to the initial path when the slopes match [2, 27].

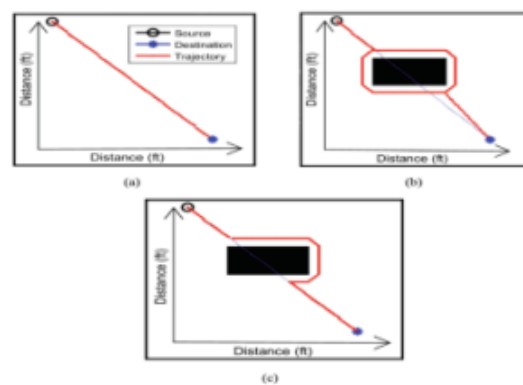


Figure 2.7: Paths of Bug 1 and Bug 2 algorithms (a) No obstacle (b) Bug 1 algorithm (c) Bug 2 algorithm [2]

2.3.1.6 Rapidly Exploring Random Tree

Rapidly-exploring random trees (RRTs) are introduced for efficient exploration of nonconvex high-dimensional spaces. They build a space-filling tree incrementally from random samples, biased to explore large unsearched areas. RRTs handle obstacles and differential constraints, making them widely used in robotic motion planning. They generate open-loop trajectories for nonlinear systems and approximate control policies for high-dimensional systems with state and action constraints. RRTs preferentially expand towards unsearched areas with controlled growth rate. Biased sampling guides RRTs towards planning goals with a higher probability of sampling the goal state [32, 27].

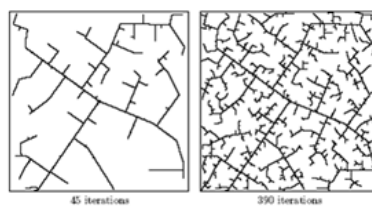


Figure 2.8: Rapidly exploring random tree method representation

2.3.1.7 Virtual Impedance Method

A virtual impedance technique improves motion smoothness in a versatile robot. It enhances robot performance in dynamic environments with both static and moving obstacles. Compared to previous methods, this technique is more efficient due to its simple framework. The algorithm incorporates impedance control into the collision avoidance process, converting the distance between the robot and obstacles into a virtual force for obstacle avoidance. The study evaluates the approach on a physically moved robot platform in an indoor setting. Efficiency and smoothness are crucial for advancing robot control. Combining teleoperated advancement control with autonomous collision avoidance proves effective in practical applications, especially in small local spaces [32]. The virtual impedance method enhances robot adaptability, enabling it to navigate intricate environments and respond swiftly to obstacles. This approach shows promising potential for improving human-robot cooperation in various real-world applications.

2.3.2 Heuristic methods

Classical approaches though found to be effective, take more time in the determination of feasible collision-free path. Also, classic approaches tend to get locked in local optimal solution which may be far inferior to the global optimal solution. These drawbacks make the classic approaches to be incompetent in complex environments. To solve the path planning problem quickly, evolutionary approaches are employed. The various methods within the heuristic or Artificial Intelligence category are described in Table 2.3.

Method	Description
Artificial Neural Network	Mapping from perceptual space to behaviour space, considering energy to guide the robot to a low-energy path without obstacles, but not necessarily shortest or optimal [29, 32, 2, 27, 30, 26].
Genetic Algorithm	Encodes all problem solutions into chromosomes forming an initial population. Basic operations include crossover, mutation, and selection. The individuals selected for each operation by the fitness values, which are calculated for each individual based on the goals [25, 29, 32, 2, 27, 30, 26].
Fuzzy Logic Technique	It uses human-supplied rules to convert to mathematical equivalents for improved system performance. It is used in combination with sensor-based, reinforced-based, and algorithm-based navigation techniques for optimal environment perception and managing dead-end situations [32, 2, 27, 30, 26].
Ant Colony Optimisation	Uses ant pheromones for path communication and selection. High pheromone paths become optimal due to positive feedback. Ants find shortest paths from nest to food, avoiding obstacles [25, 29, 32, 27, 30, 26].

Method	Description
Particle Swarm Optimisation	Inspired by bird cluster activity, this algorithm begins with a random solution and iteratively finds the optimal solutions. Fitness value evaluates solution quality, ultimately determining the global optimum. Easy implementation, high precision, and fast convergence [25, 29, 32, 2, 27, 30, 26].
Bacterial Foraging Optimisation Technique	Inspired by bacteria behaviour. It incorporates chemotaxis, swarming, reproduction, and elimination principles. Efficient nutrient search by attracting and warning other bacteria. Seeks highly nutrient regions on the map and disperses to explore for new nutrient regions [32, 2, 27, 30].
Bee Colony Optimisation Technique	Swarm-based approach inspired by honey bees. Consists of a population of inherent solutions (food sources). Population-based stochastic search in swarm algorithms. Food search cycle involves employed bees evaluating nectar quality [32, 2, 27, 30].
Firefly Algorithm Optimisation Technique	Inspired by fireflies' flashing behaviour, used as a metaheuristics algorithm. It imitates fireflies' random states and trial-and-error nature. Fireflies use bioluminescence to communicate, select mates, and protect themselves [32, 2, 27, 30].
Simulated Annealing	Stochastic optimisation algorithm based on the iterative strategy of Monte-Carlo. It begins with higher initial temperature, gradually decreasing it to find the global optimal solution through random exploration with probability jumps [29].
Dijkstra Algorithm	Shortest path in directed graph, starts from centre to end, using edge weights. The values of the edges of the graph are described by the weight function. Maintains vertex sets A and B, each time a vertex B moves to A, minimising edge weights to reach that vertex [29, 32, 2, 27].
A* Algorithm	Based on Dijkstra algorithm. Updates weighted values of child nodes from a specific node. Chooses the node with the smallest weighted value to update the current node. Uses an evaluation function for shortest path search. Higher efficiency due to considering the mobile robot's target point [29, 32, 2, 27].
D* Algorithm	For robot path exploration in dynamic environments. Represents the problem space as a series of states, and the states represent the direction of the robot's position [27].

Table 2.3: Comparison of Various Navigation Algorithms

2.4 Docking systems

To ensure the uninterrupted operation of the robot without the need for human intervention, it is necessary to recharge the robots periodically to uphold their autonomous functionality. This chapter presents details regarding various robot localisation methods used for docking and recharging energy, along with an analysis of their respective advantages, disadvantages, and experimental findings. Additionally, the mechanisms employed to power the robot and prevent errors in the recharging process are also discussed.

2.4.1 Docking station localisation methods

Selecting the appropriate sensors for recognising the recharging station during the docking process is one of the main challenges in the docking process. In certain scenarios, the path to the docking station may be predefined, but for fully autonomous robots, knowing their localisation and that of the docking station becomes essential. Moreover, the possibility of the docking station being movable (e.g., during cleaning or maintenance) introduces the risk of slight location changes that can lead to errors, necessitating careful consideration. The localisation of the docking station can fall into different categories: it may be precisely known, have partial information about the general area, or have no information at all, making the task more challenging and reducing the accuracy and efficiency of the chosen method accordingly. There are numerous approaches and methods for docking localisation including infrared (IR) sensor approaches [33], vision-based algorithms [34], laser-based methods [35], the use of ultrasound sensors [36], and radio-frequency [37, 38], among others. This chapter includes a comprehensive literature review on this subject. The literature presents approaches with single sensors, which may not be optimal, as they rely solely on one type of information. Alternatively, multi-fusion sensors and/or cameras are utilised, which significantly enhance the accuracy of the docking process. By leveraging the strengths of different sensors and compensating for their individual limitations, the docking process can capitalise on each sensor's advantages while mitigating their disadvantages through the integration of more suitable sensors tailored to specific tasks.

2.4.1.1 Infrared sensors approaches

In [3], the docking process is facilitated by employing a combination of two infrared sensors. Since the environment is known, the robot's path can be predicted according to its self-localisation, given that the initial point is known. As the robot approaches the vicinity of the docking station, it relies on Infrared Transmitters (IRT) installed within the docking station to provide guidance for the docking and recharging operations. To enhance the accuracy of the docking process, the robot adopts a cautious approach and initiates docking only when both infrared sensors detect its presence simultaneously (see Figure 2.9). By requiring dual sensor confirmation, the system minimises the chances of misalignment or errors during the docking procedure, resulting in a more reliable and precise recharging process for the robot.

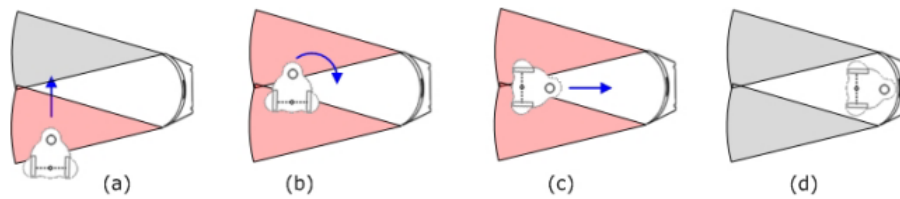


Figure 2.9: The diagram of the final docking process when the robot is detected by one of the IR sensors on the docking station. (a) The robot is detected by one of the IR sensors. (b) The robot is detected by both sensors. (c) The robot turns 90 degrees to face the docking station. (d) The robot connects itself to the docking station. [3]

The experiment aimed to evaluate the accuracy of the docking process using the described method under three different scenarios: docking, navigation, and a combination of both. Each case was thoroughly examined, and the input from various sources was considered, namely, the left IRT, the right IRT, both IRTs together, and without any IRT information. For each process, a total of 10 tests were conducted. The results of the experiment showed that the docking approach was highly successful. There were no failures observed in the first case, where input from both IRTs was used. However, there was one failure recorded in each of the remaining cases, where the left IRT or the right IRT was used as the sole input, and when no IRT information was available, respectively. However, in the case of the docking process, the accuracy was subject to variability based on the starting point of the robot. Some starting points exhibited two failures, while others experienced only one failure. Overall, out of a total of 60 docking attempts made during the experiment, the success rate was impressively high at 90%. These results indicate the effectiveness of the method in achieving reliable and accurate docking and recharging for the robot, particularly when both infrared sensors are utilised. However, it also highlights the importance of considering the starting point for further refining the method's performance in certain scenarios.

In [4], a combination of IR distance sensors and QR codes as landmarks is used for the docking process. The IR sensors are also used for obstacle avoidance while docking, helping the robot navigate safely around any potential obstacle present in the environment. Moreover, these IR sensors play a crucial role in preventing camera miscalibration errors, which lead to inaccuracies in the robot's perception of its surroundings. Additionally, the IR sensors are leveraged for distance and angle estimation, aiding the robot in precisely determining its position concerning the docking station. The two IR sensors are strategically configured within the robot at specific angles to optimise their sensing capabilities, as shown in Figure 2.10.

In the first step of the docking process, the robot turns around to find the QR code landmark. It takes a picture and records distance readings from the IR sensor. If the landmark is detected, the robot determines its location on the left or right side based on the relative sizes of the lateral edges of the QR code landmark, represented in Figure 2.11.

If the robot fails to find a QR code in the picture taken during its initial rotation, it proceeds to rotate an additional 30 degrees and repeats the process of taking another picture and scanning for the QR code landmark. The IR distance values are stored throughout this procedure. If the robot



Figure 2.10: IR distance sensors [4]

completes a full turn without detecting a QR code, it uses the IR sensors to check for any obstacles obstructing the path to the charging station. If an obstacle is detected, the robot manoeuvres around it and attempts to take a picture from a different perspective to find the QR code. However, if no QR code is present in the picture, and the IR sensors do not detect any obstacles, it indicates that the robot is located too far from the QR code (beyond 3 meters). In this case, the robot engages in a random walk pattern until it locates a QR code. The IR sensor readings are also used for assessing accuracy and determining whether it is possible to dock the robot at a particular angle, but this is only considered when the robot is in close proximity to the docking station (less than 0.75 meters). The experiment results have shown this approach to be robust and fully functional. However, it has been noted that the time spent in certain situations, such as encountering obstacles or angles where the QR code cannot be detected, was excessive and sub-optimal. To improve efficiency, alternatives to QR codes, such as other visual markers with real-time performance, could be explored and analysed.

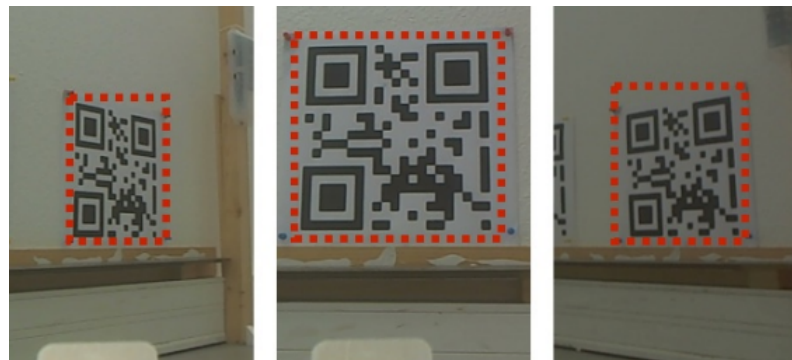


Figure 2.11: Robot's views from: left, centre and right [4]

In [5], infrared sensors are employed to localize the docking station. In this case, the docking station is equipped with three infrared transmitters (left IRT, central IRT, and right IRT), each emitting two signals (far signal and near signal), totaling six different infrared signals. The robot only considers the docking infrared transmitters (IRT) when it needs to charge. In this case, it searches for the docking station with his three infrared receivers (IRR). Each docking station's transmitter covers an area, as shown in Figure 2.12.

If the robot detects the signal from the docking station, it aligns the direction of the central

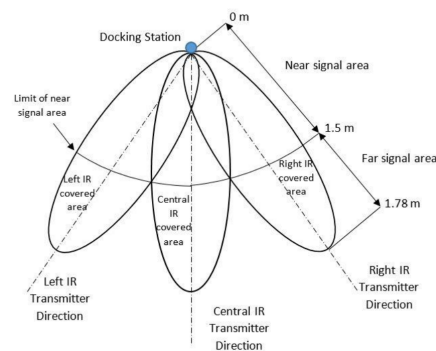


Figure 2.12: The global navigational space of the prototype robot [5]

IRR with the direction of the central IRT. Otherwise, it turns 45° randomly to the right or left and continues the search for the docking station. Test results showed that an optimal angle of 75° between the right IRR and the central IRR, and the left IRR and the central IRR resulted in the shortest arrival time (SAT).

In [39], improvements were made by adding a fourth IR sensor to aid in the docking process. This additional sensor cooperates with the limit switch (LS) to acknowledge when docking is complete. In [6], the robot follows a continuous squared path (30 by 30 inches) while equipped with an infrared receiver (IRR), as presented in Figure 2.13.

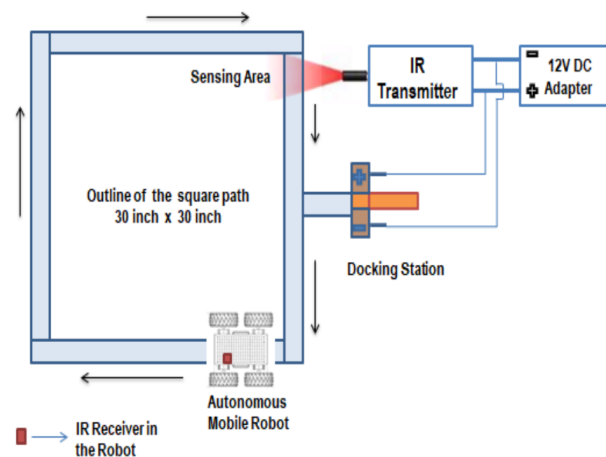


Figure 2.13: Robot Moving in a Square path [6]

The robot is equipped with an infrared receiver and the docking station with an infrared transmitter. When the robot's battery level reaches a threshold, it autonomously attempts to reach the docking station. Upon detecting the IRT in the docking zone, the robot starts moving towards the docking station. The docking algorithm presupposes that the robot's movement is near the docking station and follows a predefined path, leading to a 95% success rate.

2.4.1.2 Vision-based algorithms - Visual Markers

Nowadays, the cost of cameras has significantly reduced due to advancements in camera technology. Vision-based algorithms have become crucial in robotics, as they offer various benefits, such as pose estimation, accurate localisation data, structure from motion, and environmental information, all from a single visual marker.

As mentioned before, in [4], QR codes were used as landmarks, but their lack of real-time video decoding limited their utilization despite their performance and information storage capabilities, with a considerable amount of libraries. To address this limitation, various other visual markers are available.

The robotics field employs several fiducial visual markers, such as AprilTag, ArUco, ARTag, STag and ARToolKit). However, ArUco and AprilTag are the most popular markers in docking processes. In [40], experimental tests demonstrated that both markers performed well in real-time, even under different floor patterns, lighting conditions, and marker sizes. AprilTag, in particular, showed faster results and allowed for the reduction of the marker size, making it a preferred choice.

In [7], a comparison of four different open-source fiducial markers (AprilTag, ArUco, ARTag, and Stag) was conducted, evaluating their localization capabilities and computational efficiency 2.14.

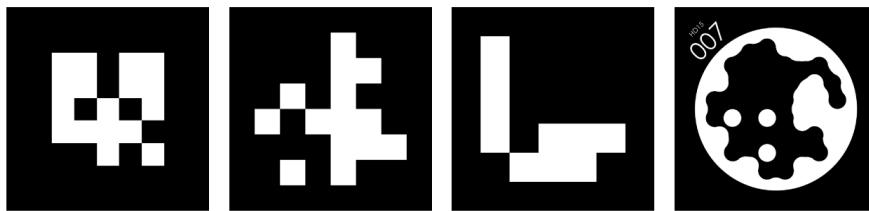


Figure 2.14: The markers used in this work: (a) ARTag, (b) AprilTag, (c) ArUco and (d) STag [7]

The study's objective was to compare four different open-source fiducial marker packages on three different small computers. The authors also developed and released a ROS package for the STag marker. The experiments were conducted to assess how the markers' stability, accuracy of pose measurements, and detection rate were affected by changes in the marker's distance and angle relative to the camera used (Figure 2.15) and varying lighting conditions (Figure 2.16).

In the results obtained, AprilTag, ArUco and Stag presented high detection rates in almost all the settings tested. The markers that presented better results in orientation and pose measurement are AprilTag and STag, respectively. ArUco marker came in second for both. Since STag is a fiducial marker package that focus on stability, its superior performance in this area was expected, and it indeed showed the lowest standard deviation in most settings. AprilTag and ARTag demonstrated comparable performances, with AprilTag displaying higher stability in orientation measurements.

In terms of CPU and memory usage, AprilTag required the most resources, while STag had the second highest CPU usage but low memory usage. ArUco and ARTag had the lowest computational resource usage. Considering the trade-off between measurement accuracy and detection

rate, ArUco is recommended for low-power computers, while for more powerful computers, all markers except ARTag represent excellent choices.

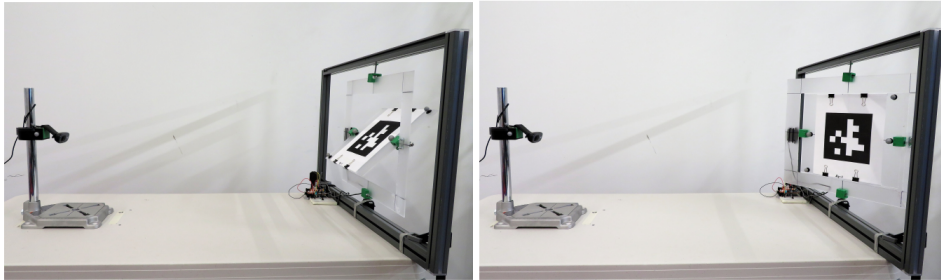


Figure 2.15: Overview of the experimental setup with the two axes gimbal on the right side of each image, where (a) the marker position is altered in the longitudinal direction and (b) the marker position is altered in the lateral direction. [7]

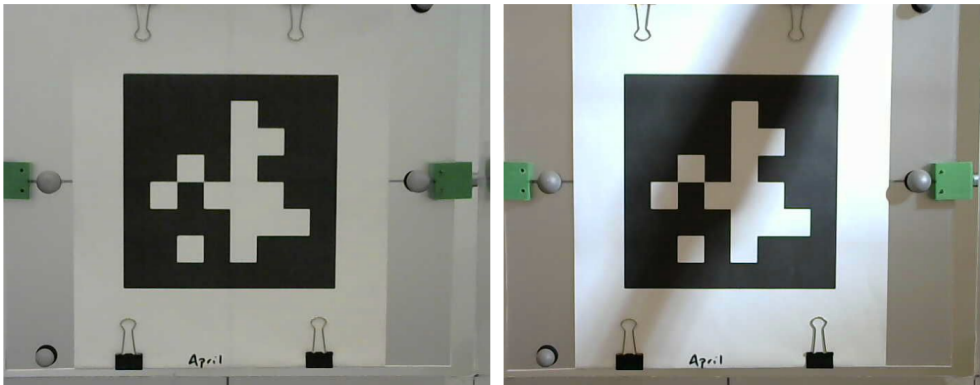


Figure 2.16: The AprilTag marker used (a) under normal lighting conditions and (b) with a shadow being cast as seen from the Logitech camera. [7]

In [8], the same authors of [7] released a more detailed paper one year later. This subsequent study evaluated how marker performance was affected by different configurations, such as single markers, planar and non-planar bundles (Figure 2.17), and multi-sized marker bundles (Figure 2.18). Additionally, the study investigated the impact of lighting conditions and simulated noise from shadows and motion blur on marker performance.

In the study, two different cameras with varying resolutions were used. Surprisingly, the lower resolution camera outperformed the higher resolution camera in most cases. The authors attributed this difference to the higher contrast and better white balance of the lower resolution camera. The single marker tests were similar to the previous paper. When comparing individual markers to bundles of multiple markers, all packages showed increased detection rates, especially ARTag.

However, the precision of ARTag was worse in the bundle configurations. The non-planar bundle demonstrated better orientation results compared to both single markers and planar bundles. In motion blur experiments, AprilTag's performance was reduced in both types of motion blur

(rotation and translation blur). The other experiments produced similar results to those mentioned in the previous paper.

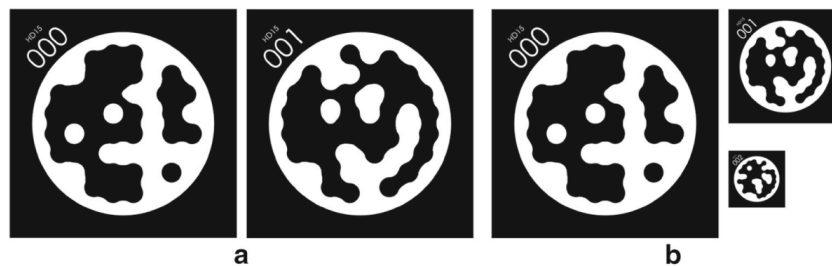


Figure 2.17: The STag multi-tag bundles (a) with two equal sized markers and (b) with three differently sized markers [8]

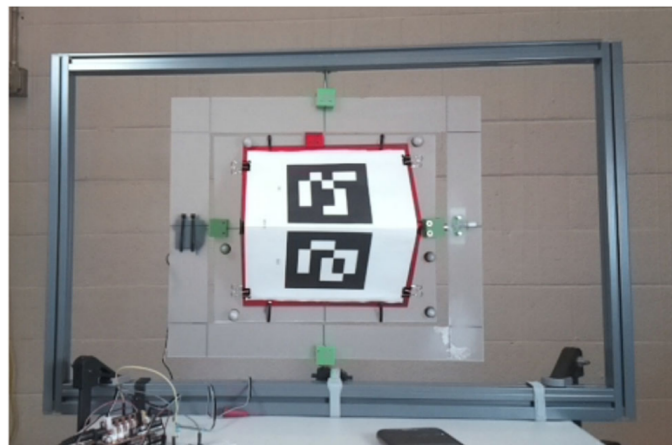


Figure 2.18: The two marker bundle in the non-planar configuration. The marker is rotated about the vertical axis [8]

Between the two papers, the STag package received improvements in computational efficiency. Although it still had the second highest CPU usage, these enhancements brought it closer to the least resource-consuming packages (ARTag and ArUco). In some cases, STag even outperformed the other packages in terms of efficiency. In Table 2.4, the pros and cons of each visual marker are presented, summarising the strengths and weaknesses of each package based on their performance and resource requirements.

In [9], the docking process uses AprilTag as the visual marker for determining the position between the docking station and the charging devices. When the robot's battery level is low, it employs the "enhanced ORB-SLAM" for self-localisation, which is more efficient and stable than the standard "ORB-SLAM." However, as the ORB-SLAM strategy is less accurate than AprilTag, once the robot gets close to the charging station (within a region approximately 5 meters ahead of it), it switches to using AprilTag.

The robot actively searches for the AprilTag and uses the relative position information from the tag to adjust its position and angle accordingly (see Figure 2.19). Despite challenging conditions

Marker	Pros	Cons
ARTag	Lowest computational cost overall	Low detection rate for single markers Extreme outliers & high Std Dev in marker bundles
AprilTag	Great orientation results and detection rate Good position results	Most computationally expensive Most sensitive to motion blur Worst results in the non-planar setup
ArUco	Good position and orientation results Great detection rate Low computational cost for single markers	Sensitive to smaller marker sizes and larger distances Computational cost scales with multiple markers
STag	Great position results and detection rate Good orientation results	Sensitive to smaller marker sizes and larger distances

Table 2.4: Experimental comparison summary: Main advantages and disadvantages of the four evaluated packages [8]

such as noisy images and obstacles in the path, the docking operation achieves a very high success rate of 97.33% with exceptional accuracy. This combination of localisation methods allows for efficient and reliable docking even under challenging circumstances.

In [10], the ArUco marker detection is used to localise the charging station. This method relies on recognising the corners of the ArUco marker. Once the marker's corners are detected, its size is known, and it is divided into a grid or matrix. Each cell in the grid is represented with binary code (1 for a black cell and 0 for a white cell), as shown in 2.20.

By knowing the geometric dimensions of the ArUco marker, its position and orientation relative to the camera's coordinate system, which is usually attached to the mobile platform structure, can be determined. This assumption is based on the camera always being in the same position relative to the robot's reference frame and the ArUco marker always having a consistent position relative to the charging station's reference frame. Using this information, the docking task becomes more straightforward and precise after the ArUco marker detection by the robot's camera. The algorithm calculates the relative position, on the world map, of the robot relative to the charging station, or vice versa.

This approach offers the advantage of avoiding false positive detections, which could be observed in LIDAR measurements for localising a charging station. Additionally, it provides a reasonably high localisation accuracy. However, the accuracy of this method depends on the resolution of the camera and the size of the ArUco marker.

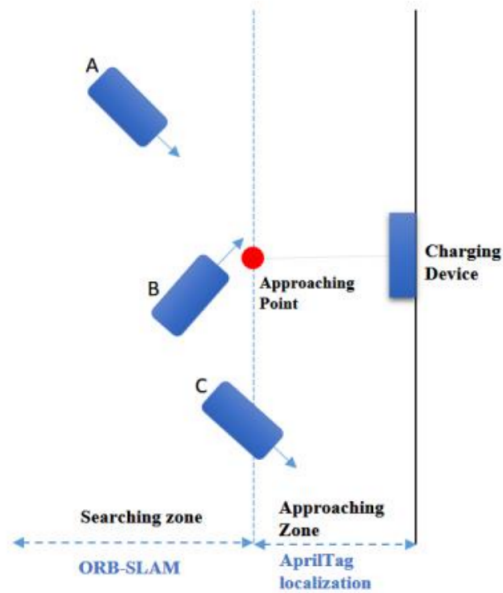


Figure 2.19: In the searching zone, the robot locates itself through ORBSLAM; A, B, C represent different results of searching the approaching point; In the approaching zone, the AprilTag localization is adapted to calculate the relative position of charging device. [9]

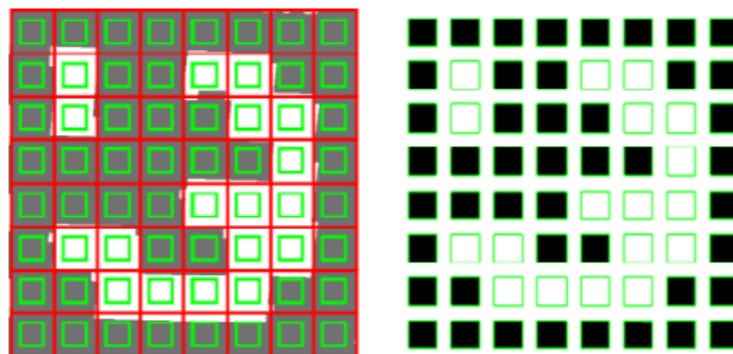


Figure 2.20: ArUco Marker Detection [10]

2.4.1.3 Laser-based methods

In [11], an automatic docking process using LIDAR sensors is presented. The controlled robot is a differential wheeled robot. When the robot needs to recharge its battery, it uses a map matching algorithm for localisation and navigation. In the first place, the laser data information is transformed to draw a map. Then, the docking station is identified based on its features among the other lines drawn. The docking is recognised based on the parallelism of the docking station with the wall, and the length and thickness of the line that hypothetically represents the docking station, Figure 2.21.

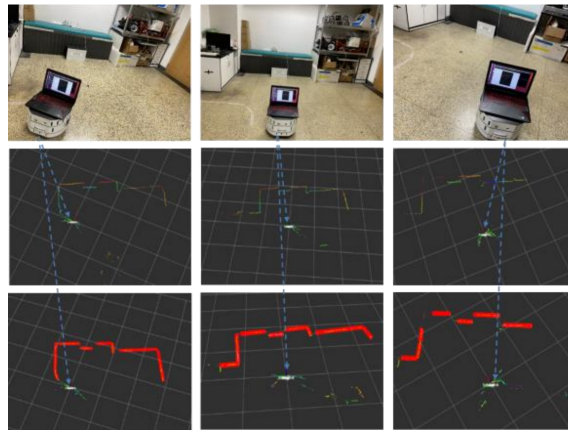


Figure 2.21: Fitting results of the robot to the surrounding environment information [11]

Afterwards, the middle point of the docking station is calculated. Then, the position of the robot is determined as well as the angle and distance deviation from the station through particle filter position. Finally, a multi-state stabilisation controller is presented to dock the robot efficiently and accurately. The error of the docking process in the x-axis is approximately 0, and the average error in the y-axis is less than 2 cm, while the average angle error is less than 3° . The experiments performed had different initial positions and angles. The coordinates obtained between the station and the robot were accurate, as well as the final results due to the stabilisation controller.

2.4.1.4 Laser intensity methods

Although laser intensity is implemented for many different mobile robot systems and applications, according to the author's research in [12] it has never been directly implemented for autonomous docking and charging in unstructured environments.

While the laser-based autonomous docking approaches detect the docking station by its contours, the laser intensity based autonomous docking [12] uses the intensity of the reflected laser.

Since most mobile robots already have laser range finders equipped for localisation and mapping, it is not necessary to have extra hardware as IR receivers and transmitters, and allows flexibility in the design of the platform, since the transmitters have to be on the same level as the receivers to do a reading. Also, for vision-computer based methods, lighting conditions might

affect the docking process as well as camera miss-calibrations. The other laser-based autonomous docking approaches, that detect the docking station by its contours, require a specific design shape, and have strong requirements for unstructured environments. The laser-intensity method not only is very easy to implement, but also it is cheap. It consists in a retro-reflective region, which is in the middle of two black-rubber regions, Figure 2.22. When the laser scans the area, the intensity of the reflected laser is much lower in the black-rubber bands than in the reflective band, as can be seen in Figure 2.23.

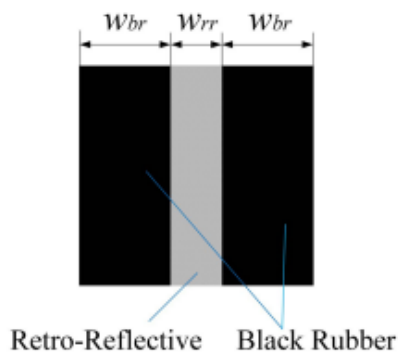


Figure 2.22: Reflective tape [12]

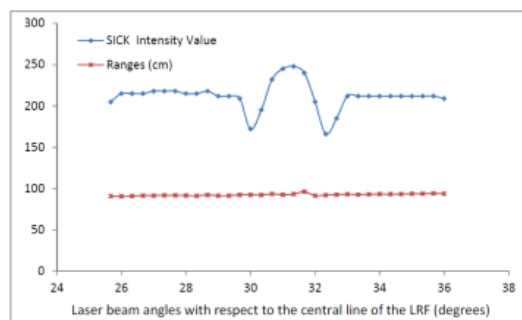


Figure 2.23: Laser intensity value graph [12]

Some experiments were made with the robot in an open area, with the docking station manually placed in 4 different places, and the results obtained were very good. With the box placed at 31.33° from the robot, there are 2 valleys in the blue line, representing the black-rubber material, with a peak in the middle, representing the retro-reflective material.

After these great results, two experiments were made to dock the robot to its station. One of them consisted on testing the laser-intensity based autonomous docking approach in 2 scenarios, see Table 2.5: in one of them the robot went through a corridor to the docking from the right side of the docking station, and in the other from the left. In the following table, we can see the results obtained, that represent 99.5% and 97.5% of success rate for docking and charging, respectively, while in the second scenario was 98.5% and 96.5%, considering that this experiment delivered good results in both scenarios.

Scenario	Scenario1	Scenario2
Total number of trials	200	200
Number of successful trials	195	193
Number of trials with failed charging	5	7
Number of trials with failed docking	1	3

Table 2.5: Experimental results for the first set of experiments [12]

In the second experiment, the same method was used, but the contour-based autonomous docking method was compared to the laser-intensity referenced approach. The scenarios were the same as presented in the previous experiment, but in the second one, the path between the robot and the docking station was blocked for some time by a person, see Figure 2.24.



Figure 2.24: Human blocking the route between the robot and the docking station [12]

The results obtained are presented in Table 2.6. Transforming the results into percentages, for the first scenario, the success rate for the laser-intensity approach is 99.5% and 97.5% for docking and charging, respectively, and the success rate for the contour technique is 97.5% and 95% for docking and charging, respectively.

In the second scenario, the success rate for the laser-intensity approach is 98.5% and 95% for docking and charging, respectively, and for the contour technique is 84.5% and 82.5% for docking and charging, respectively.

Therefore, in the unstructured environment with dynamic moving obstacles, the laser-intensity method presented better results than the contour-based technique.

Docking Methods	laser-intensity approach		contour technique	
	Scenario1	Scenario2	Scenario1	Scenario2
Total number of trials	200	200	200	200
Number of successful trials	195	193	190	165
Number of trials with failed charging	5	7	10	35
Number of trials with failed docking	1	3	5	31

Table 2.6: Experimental results for the second set of experiments [12]

Although these results were great, there were some fails. For most of the failed recharging trials in scenario 1, the autonomous docking was successful, but the charging station was tilted, resulting in a lack of contact with the charging bars. Other phenomenons compromised perfect results such as wheel slippage, backing motion errors, abandonment of the robot from the autonomous recharging due to timeout caused by the person blocking the retro-reflective material (second experiment, second scenario). In more complicated scenarios and for different environment conditions, such as the influence of dynamic light, different barrier materials and sunlight effects in outdoor environments, the behaviour of the laser-intensity based approach is unknown. However, in the tests performed in this environment, the results were great.

2.4.2 Docking station recharging methods

Another important aspect about the docking approach is how the recharging process functions. There are many different recharging contact methods help countering angle and distance errors in

the docking process, most of them being wireless. Different types of connections of robots with the respective docking station are described below.

In [3], the robot's shape for docking consists in a semi cylinder with two charging electrodes. There is an elastic incorporated to support the impact in the process. The docking station is arched and in the docking action, it is tolerable a position error of the robot up to 7 cm, and an angle error up to 60 degrees. The architecture of the robot and the respective docking station is shown in Figure 2.25.

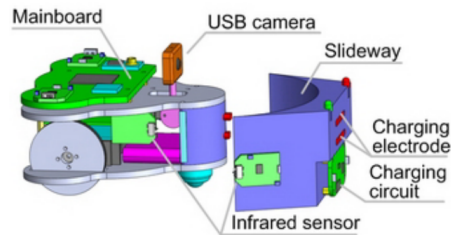


Figure 2.25: CAD model of the proposed surveillance robot and the docking station. [3]

In [4], the powering system dock consists in a metallic bar, with two contact surfaces 0,80m length, allowing the charging of two robots simultaneously, Figure 2.26 (tension provided: 18V; current provided: 8A).



Figure 2.26: Dock connector [4]

In the robot there is also a smaller two contact point metallic bar, that allows the robot to autonomously dock and recharge its battery, Figure 2.27.

In [5] a new Power Management System (PMS) is used for the battery recharging process. Not only detects the connection status between the robot and the docking station with a limit switch (LS) but also controls the continuous recharging process. The Power Management System's monitoring circuit is presented in Figure 2.28.

In [6] the recharging station consists in two metallic electric contacts, Figure 2.29, which in contact with the respective metallic electric contacts from the robot allows powering the robot, Figure 2.30. A 12V battery is used and its level is shown in an LCD display.



Figure 2.27: Docking station [4]

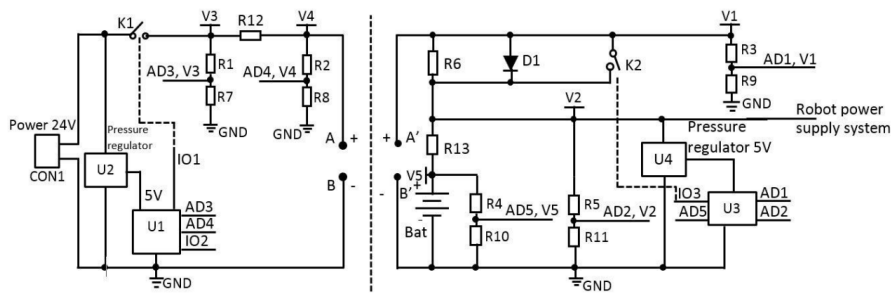


Figure 2.28: The Power Management System monitoring circuit [5]

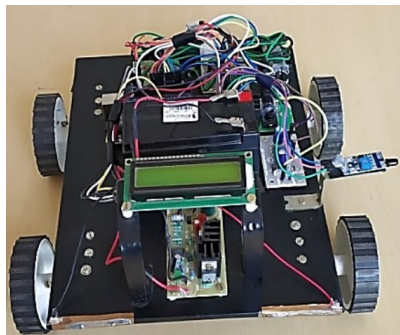


Figure 2.29: IR transmitter and docking station [6]

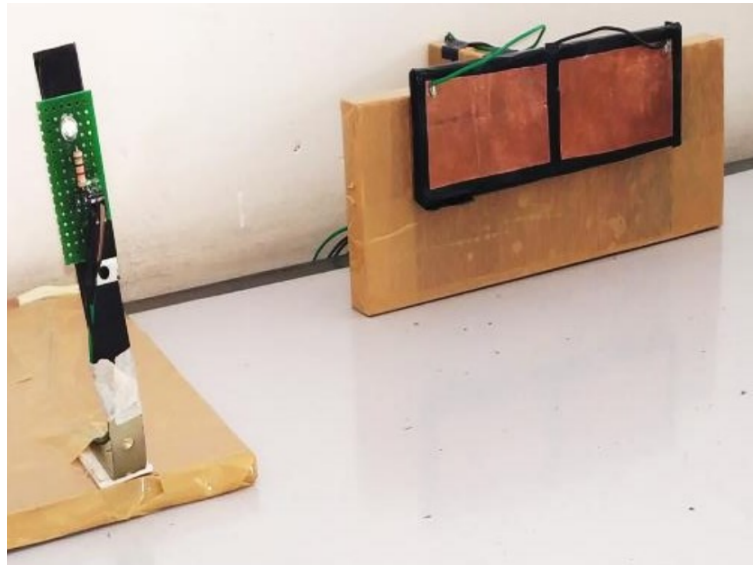


Figure 2.30: Top view of automatic docking robot [6]

In [9], the docking station has three flat metal plates, and the robot has three metallic cylinders, Figure 2.31. When the metals contact, the robot starts charging. This approach is more reliable than some others since it allows misalignment.



Figure 2.31: (a) The docking and charging station. (b) The robot docking mechanism robot we developed [9]

2.5 Obstacle avoidance systems

Obstacle avoidance is a critical aspect of mobile robot navigation, enabling them to move safely and efficiently in complex environments. Over the years, numerous obstacle avoidance algorithms have been developed, each employing various path planning approaches. These algorithms play a pivotal role in enabling robots to autonomously detect and circumvent obstacles in their surroundings. There are many obstacle avoidance algorithms using different path planning approaches, such as the Fuzzy Social Force Model [41], Vector Field Histogram algorithm [42], Improved Genetic Algorithm [43], Dynamic Recursive Ant Colony Algorithm [44], and Modified Particle Swarm Optimisation [45]. In this section, it will be discussed in detail two algorithms with variations of the artificial potential field and dynamic window approach.

2.5.1 Obstacle avoidance methods

For a robot to travel between two given points and under its trajectory, it is important to avoid obstacles that may appear along your route. There are many methods and algorithms for obstacle avoidance, but most of them get stuck at the local minimum, which can result in the stoppage of the robot when it encounters an object. The modified artificial potential field, proposed by [13], consists in a safely avoid collision with fixed obstacles in an optimal environment, allowing the robot to reach the final destination without facing the local minimum, unlike the conventional artificial potential field. The artificial potential field consists in a gradient function field, where the target (final destination) generates an attraction potential field as an attraction force (F_a), and the obstacle a repulsive force field, as a repulsive force (F_r) and the artificial potential is the sum of both forces (F_{all}), as it is possible to see in Figure 2.32.

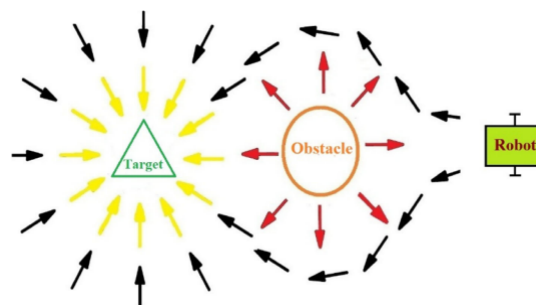


Figure 2.32: Artificial potential field model [13]

However, in some situations, the sum of the forces can be zero, resulting in the robot wandering around (local minimum) or it can even stop. To avoid that, the modified artificial potential field decomposes the repulsive force (F_r) into 2 forces (F_{r1} and F_{r2}), as seen in Figure 2.33.

A regulative factor (M) is added to the artificial potential field algorithm equations, and in this case, the robot does not get stuck in the local minimum. In the simulations, for the artificial potential field, the robot still got trapped in the local minimum, while in the modified artificial potential field it never got stuck, proving to be a better solution.

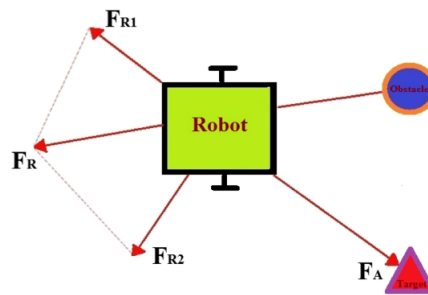


Figure 2.33: Modified artificial potential field model [13]

In [14], an hybrid obstacle avoidance method is used. It combines an informed-rapidly exploring random tree (RRT) with 3D target detection model to detect obstacles and model prediction controller (MPC) for obstacle perception, collision-free path planning and obstacle avoidance. The experiments performed are done in unstructured environments for wheeled mobile robots. Although LIDAR detects the location of obstacles accurately, being a great option to detect static obstacles, it can't determine the speed of a moving obstacle as the 3D target detection model. For the dynamic obstacles, a parametric ellipse represents the obstacle movement and its area. The orientation and speed of the dynamic obstacle allows to predict the future possible areas of the robot, delimited by the ellipse, and can choose a path with a lower object collision risk, as seen in Figure 2.34.

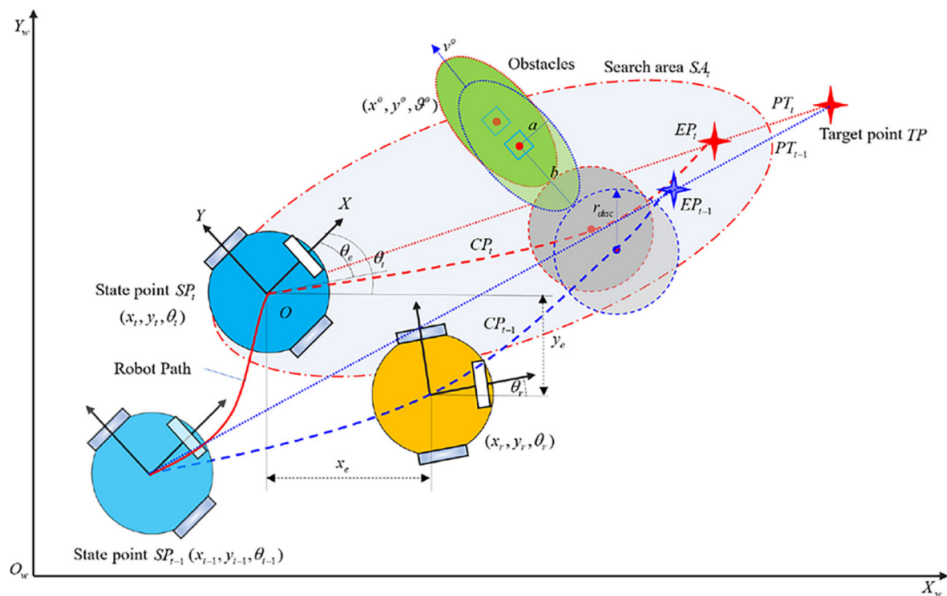


Figure 2.34: Collision-free path planning and tracking error [14]

For the collision-free path planning, the informed-RRT algorithm is used. This algorithm predicts a path according to the robot states and the final point, avoiding collision. This approach can reduce the amount of calculation not compromising its behaviour in real-time.

The control method used is the MPC optimisation function. Depending on the measured values

in each state, it predicts the next motion of the robot, guaranteeing a safe distance between the robot and the obstacle to reduce the possibility of collision.

The experiments were performed with different velocities of the dynamic obstacles and different scaling factors (ellipse size). With bigger obstacle velocities and the occupied area of the obstacles, the tests failed. Comparatively to other algorithms (DWA, MPC and LMPCC), this approach experienced better results in performing obstacle avoidance and a higher success rate. The possibility of collision in the dynamic obstacles was predicted in advance comparatively to the other algorithms, being easier to avoid obstacles.

In [15], the dynamic window approach is improved, with the name of finite distribution estimation-based dynamic window approach (FDEDWA). This algorithm estimates the overall distribution of obstacles through the finite memory filtering (FMF), predicting the future state and distribution of obstacles and allowing it to avoid them. It's performed by the estimation of its position, velocity and distribution. FMF has a good performance on estimation and response speed, that's why it was chosen. It has a fast and effective detection of the obstacles. The sensors used for self-localisation of the robot were ultrawideband sensors (UWB), allowing it to move to the final destination.

The algorithm developed on this paper (FDEDWA) maximises the objective function with the kinematic constraint. For this purpose, the robot's speed, angle to the final point and distance between the predicted trajectory and the obstacle are used. It provides more robust and better control for the robot for static and dynamic obstacles, comparing to the traditional DWA algorithm, Figure 2.35.

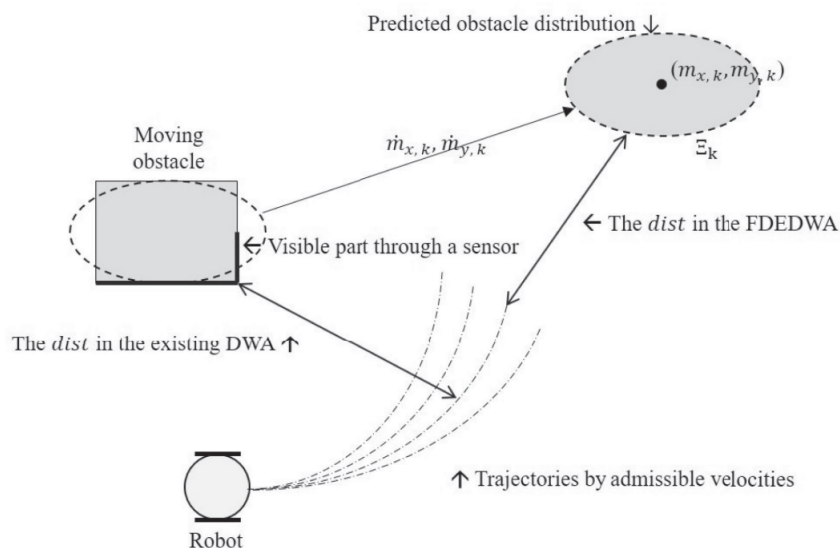


Figure 2.35: Comparison of objective function computation between FDEDWA and the existing DWA [15]

Comparing to the dynamic window for dynamic obstacles (DW4DO), which is also an improvement to the DWA algorithm for dynamic obstacles, the FDEDWA presents better results to

unexpected dynamic obstacles that appear suddenly, due to the fact that FDEDWA has a low computational time comparatively to the DW4DO because DW4DO uses occupancy grid estimation for localisation and it influences in the time calculation. In the experiments performed, the FDEDWA presented the best results between the three algorithms. In the other two algorithms, there were collisions due to their relatively slow convergence speed. While DWA collides in both static and dynamic obstacles, DW4DO has a better performance, but colliding with suddenly appeared moving obstacles. In every case, FDEDWA had great performances and avoided collision.

Chapter 3

Literature Review

3.1 Obstacle avoidance: Path planning with graphs

In this section, some papers that explore path planning using graph algorithms will be addressed, briefly touching upon their obstacle avoidance approaches.

In [46], an algorithm is introduced with the capability of generating smooth paths for non-holonomic mobile robots of any shape, while considering orientation restrictions. The primary objective of this algorithm is to navigate the robot in close proximity to obstacles, enabling efficient and obstacle-aware movement. The key innovation lies in the extension of the A* algorithm within a cell decomposition framework, considering both the robot's position and orientation during the path planning process. For this purpose, the orientation space is divided into 16 layers, each representing a unique range of orientations.

This statement describes a method aimed at enhancing collision checking and reducing the complexity of the state space for a robot's path planning algorithm. The approach involves using the robot's oriented footprint, which represents the area occupied by the robot considering its position and orientation during motion. By inflating the obstacles only in the orientation of the robot's motion, the algorithm ensures that the actual footprint of the robot can be used without significantly increasing the computational burden of collision checking.

As a result of this approach, the path planner becomes capable of calculating paths that are closer to obstacles, even in narrow spaces like those close to walls and inside corners. Simulation tests have been conducted to verify the effectiveness of this planner, and it has demonstrated the ability to create feasible paths in such challenging and constrained environments. By utilising the oriented footprint and optimising collision checking, the planner achieves a higher level of efficiency and adaptability, making it more suitable for navigating in complex and tight spaces.

In [47] a path planning system designed for a robot to navigate through an oil palm plantation is introduced. The system consists of three main components: a distance estimation algorithm, an obstacle detection method, and a path planning algorithm using the D* lite algorithm. The distance estimation algorithm accurately identifies the location of detected oil palm trees using

the Kinect camera and image processing techniques. It achieved an estimation accuracy of 80% within a 4-6 meter range from the tree.

The obstacle detection method utilises five ultrasonic sensors placed strategically on the robot to detect obstacles within a 4-meter range. Detected obstacles are inflated to account for the robot's size, ensuring a clear path for navigation. The system also includes a confirming method to filter out random erroneous readings.

The path planning algorithm, D* lite, was chosen for its exceptional performance in partially known or unknown environments. It rapidly re-plans paths by incrementally learning from search information. The algorithm accounts for the robot's kinematics by re-planning paths after each motion, taking into consideration the current position and orientation of the robot. This allows for efficient real-time planning, even in dynamic environments.

The integrated path planning system functions smoothly as a whole. The robot autonomously navigates towards the target tree while avoiding obstacles detected by the ultrasonic sensors. The path planning algorithm dynamically adjusts the path as needed when new obstacles are encountered or when the robot deviates from the planned path.

Real-time testing showed high accuracy in tree detection during robot motion, accurate estimation of distances to trees, and effective obstacle avoidance during navigation. The system performed efficiently in a simulated plantation environment, demonstrating its potential for automating tasks in oil palm plantations.

3.2 Overview of the Project

This section provides an overview of the project, covering its context, the technologies, and features of the provided code, along with the implementation of the algorithms in that specific environment. Additionally, a similar obstacle avoidance algorithm to the one used in this project is introduced, addressing the limitations identified.

The project integrates the algorithms into the navigation stack developed at the *Centre for Robotics in Industry and Intelligent Systems (CRISS)*. For robot navigation and path planning, a TEA* algorithm (Time Enhanced A-Star) is employed. The testing and simulations represent a FEUP laboratory environment, with visualisation facilitated by RViz.

A graph-based path-planning algorithm is adopted, with nodes indicating critical point positions and edges representing transitions between these vertices (Bèzier curves). The provided *CRISS* code presents the vertices and edges as shown in Figures 3.1 and 3.2.

In RViz, a vertex may appear to be colliding with a wall, but the size of the robot relative to the vertex is smaller, indicating that the vertex's positioning is appropriate. Conversely, when the robot is larger than the vertex, it may appear that the vertex is correctly positioned, but in reality, it is not. To illustrate this, Figures 3.3 and 3.4 display two scenarios: in the first figure, the robot is smaller than the vertex, and it appears that the vertex is colliding with the wall, but in reality, the robot, being smaller, avoids the collision. In the second figure, the robot is larger than the vertex,

and it may not seem to collide, but in reality, it does. Precise vertex positioning is essential, especially for vertices representing docking points, to ensure efficient charging operations.

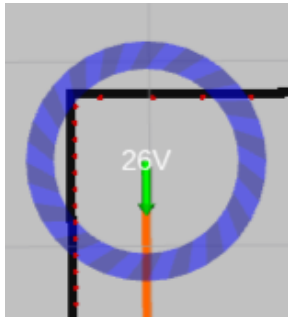


Figure 3.1: Vertex representation in RViz



Figure 3.2: Edge representation in RViz

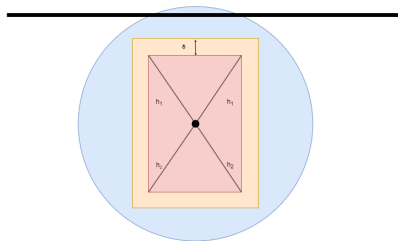


Figure 3.3: Vertex containing bigger robot

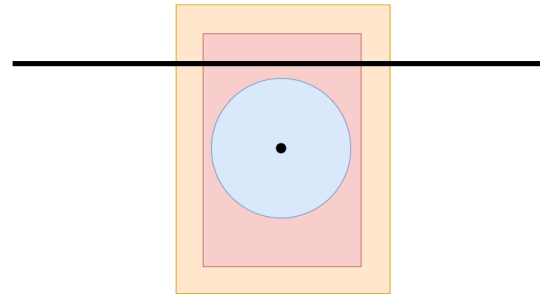


Figure 3.4: Vertex containing smaller robot

In terms of obstacle avoidance, the provided navigation stack is equipped to detect obstacles, but it merely stops the robot upon detection. Hence, an obstacle avoidance algorithm was required to navigate around the obstacles. The algorithm developed drew inspiration from the logic found in [31]. However, this particular approach took into account the robot's mass centre and height, making it suitable for navigating steep slope terrains. Given the study's focus on representing only 2-dimensional robots, this algorithm exhibits lower computational costs compared to the one presented in [31].

Despite the algorithm's robustness and versatility in various environmental scenarios, it lacks the ability to detect the full width of obstacles. It defines the encountered obstacle as a mere point and positions the avoidance vertex perpendicular to the detected point. A more effective approach would involve recognising the entire obstacle based on the complete set of laser readings.

To address this limitation, a solution was devised to enhance obstacle detection. The laser readings were divided into clusters, grouping closely spaced points within a specified Euclidean distance. Additionally, clusters in proximity to each other within a given distance were merged. This process allowed for the extraction of points that represent the boundaries of the cluster observed in the middle laser beam (initial and final points). By doing so, the obstacle avoidance process can be optimised to account for the specific shape of the obstacle, providing a more effective and adaptable strategy for navigation.

Chapter 4

System Architecture and Nodes Representation

In this chapter, the nodes developed for each algorithm and their interactions through published and subscribed topics are presented.

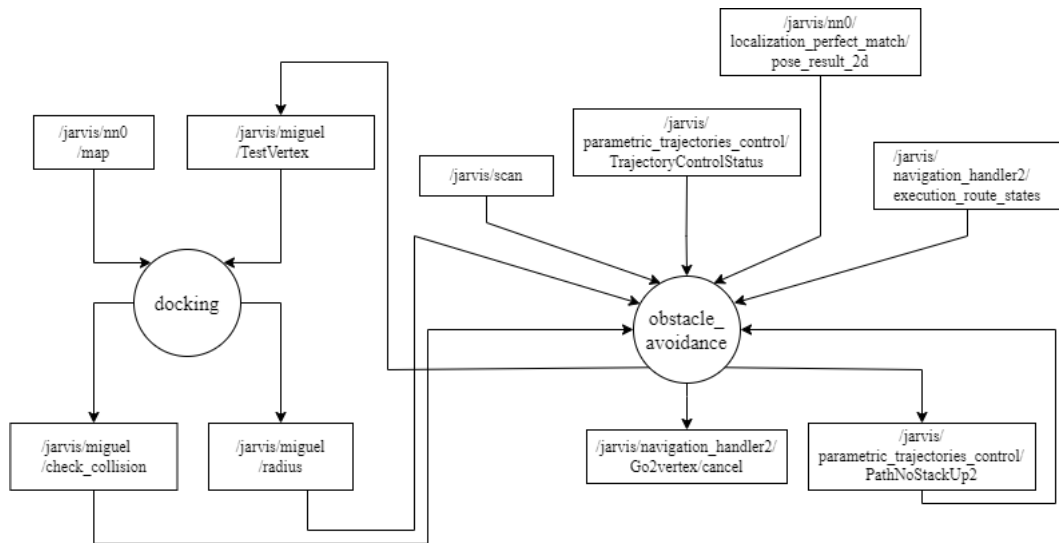


Figure 4.1: Representation of developed nodes and topics

The nodes "docking" and "obstacle_avoidance" are responsible for handling the respective algorithms. Additionally, the topics with "/miguel/" represents the created topics that share crucial information between nodes, while the remaining topics are those subscribed to or published by these nodes, already existing in the provided code. Below, each topic and its purpose are explained:

- **jarvis/nn0/map**: This topic receives data related to the grid's width, height, coordinates of origin, resolution, and the matrix containing occupancy mapping information (100 for occupied cells, 0 for free, and -1 for unknown or unexplored). Then, in "docking" node, this data is used to establish the position of each cell and the positions containing occupied cells (representing walls) are stored in a vector, which is crucial for the docking algorithm.

- **jarvis/miguel/TestVertex:** This topic receives information from a given vertex, allowing the obstacle_avoidance node to determine whether the vertex is colliding with any obstacles. This functionality can also be used by other nodes by sending messages of that type to this topic.
- **jarvis/miguel/check_collision:** This topic publishes a message "Collision" if the vertex received from the previous topic is colliding with a wall, and "NoCollision" otherwise.
- **jarvis/miguel/radius:** This topic publishes a message containing the area occupied by the robot in the simulation, which is used by the obstacle avoidance algorithm.
- **jarvis/scan:** This topic contains the laser readings of the robot, enabling the detection of imminent obstacles and providing information about the robot's surroundings.
- **jarvis/parametric_trajectories_control/TrajectoryControlStatus:** This topic receives the status of the controller, indicating whether it is currently moving or stopped. This information is crucial to determine when the robot has finished traversing the alternative path, which is indicated when the controller changes from "OS_WORKING" to "OS_IDLE."
- **jarvis/nn0/localization_perfect_match/pose_result_2d:** This topic provides the current pose of the robot, allowing for the extraction of its coordinates and orientation. This information is essential for placing vertices in the alternative path.
- **jarvis/navigation_handler2/execution_route_states:** This topic informs the current edge the robot is navigating, which is fundamental for determining the next vertex in the trajectory. Depending on the value (positive for moving forward, negative for moving backwards) and the percentage of the vertex covered, the robot's direction of movement can be determined. This information is used to trigger obstacle detection only when the robot is moving in that direction.
- **jarvis/navigation_handler2/Go2vertex/cancel:** This topic allows for the publication of a message to abort the controller, prompting the robot to stop and "forget" the main trajectory.
- **jarvis/nn0/localization_perfect_match/PathNoStackUp2:** This topic receives the Path Set sent from the controller, which is stored. When an obstacle is detected, the current edge the robot is on is identified, and the rest of the Path Set stored along with the alternative path are merged to a new Path Set. This combined Path Set is then published back to this topic, informing the robot of the new path to follow.

Chapter 5

Methodology and Algorithms Developed

The methodology employed in this study aims to address the critical aspects of docking repositioning and obstacle avoidance methods for robots. This chapter presents the algorithms developed for two ROS nodes, each tackling specific requirements: one for the docking issue and the other for obstacle avoidance. To handle the docking problem, the different types of robots considered were the ones available at *CRISS - Centre for Robotics in Industry and Intelligent Systems*, including the differential robot, the tricycle robot (single wheel drive robot), and the omnidirectional robot. As for obstacle avoidance, tests were conducted solely with the differential and tricycle robots. Unfortunately, tests with the omnidirectional robot could not be performed as it has not yet been implemented by *CRISS* with the required parameters in the simulation.

5.1 Optimising Critical Point's Placement (Docking)

In this section, the methodology and algorithm developed for optimising the placement of critical points, specifically for docking purposes, will be discussed. The preparatory steps and considerations that went into devising this algorithm will be highlighted, followed by a detailed explanation of the algorithm's execution.

5.1.1 Preparatory Steps and Considerations

Concerning the aforementioned docking issue, the primary objective is to readjust the position of the vertex, representing a docking station or a critical point, in response to the robot's dimensions. In the event of a collision with a wall, the vertex is shifted away from the wall's proximity, aiming to prevent the robot's charging disruptions or collision incidents, especially when security navigation is not enabled.

Firstly, to encompass various robot types with differing dimensions, the relative coordinates of the base footprint point are calculated using three variables (already defined and implemented in the provided source code):

1. The distance from the base footprint of the robot to the front.
2. The dimensions of the robot (width and length).

However, before proceeding with the calculations, certain premises need to be examined:

1. The robot's shape is assumed to be rectangular for all types of robots (omnidirectional, differential, and tricycle)
2. For each robot typology, the base footprint is calculated based on specific assumptions:

- Differential Robot:

The base footprint is considered to be in the middle of the robot.

$$Bf_{Differential} = (Bf_{Diff,x}, Bf_{Diff,y}), \text{ where}$$

$$Bf_{Diff,x} = \frac{RobotWidth}{2}$$

$$Bf_{Diff,y} = RobotLength - DistanceFromBaseToFront = \frac{RobotLength}{2}$$

- Omnidirectional Robot:

The base footprint is considered to be in the middle of the robot.

$$Bf_{Omnidirectional} = (Bf_{Omni,x}, Bf_{Omni,y}), \text{ where}$$

$$Bf_{Omni,x} = \frac{RobotWidth}{2}$$

$$Bf_{Omni,y} = RobotLength - DistanceFromBaseToFront = \frac{RobotLength}{2}$$

- Tricycle Robot:

The base footprint is considered to be in the middle of the back wheels.

$$Bf_{Tricycle} = (Bf_{Tri,x}, Bf_{Tri,y}), \text{ where}$$

$$Bf_{Tri,x} = \frac{RobotWidth}{2}$$

$$Bf_{Tri,y} = RobotLength - DistanceFromBaseToFront$$

These calculations serve as the foundation for determining the base footprint of each robot type, critical for the subsequent readjustment algorithm.

The first two typologies allowed for the calculation of the base footprint solely based on the robot's length, without considering the variable *DistanceFromBaseToFront*. However, by using an equation that incorporates both *DistanceFromBaseToFront* and *RobotLength* variables, we can standardise the calculation across all three typologies. Based on the premises mentioned for each robot, it is expected that both the differential and omnidirectional robots will have *DistanceFromBaseToFront* equal to $\frac{RobotLength}{2}$, thus explaining the aforementioned equations. The provided code assumes these premises, leading to the representation of the base footprint positions for each robot typology as shown in Figure 5.1.

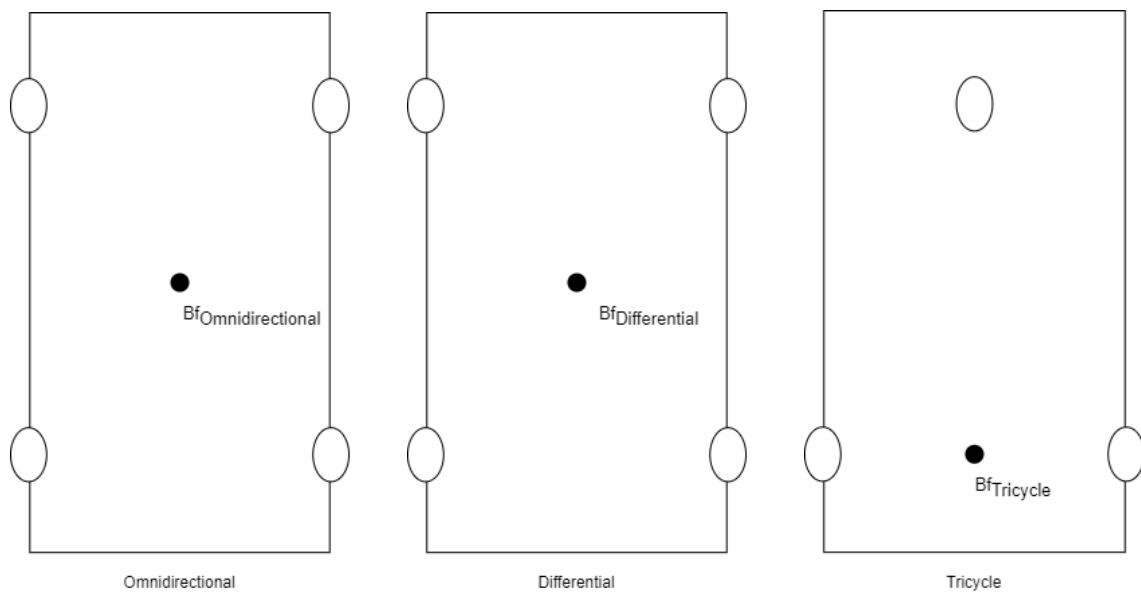


Figure 5.1: Position of the base footprint for each robot

After determining the base footprint of the robot through calculations, the next step was to define its occupied area. Ideally, considering the robot's rectangular shape, the occupied area should also be a rectangle, with a margin of error. However, due to the robot's orientation, which is closely tied to the positioning of critical points (vertices) or docking stations, it became challenging to obtain accurate corner points when the robot was oriented in specific directions or in collision with the wall. To address this, in the initial stages of the project, the robot's occupied area was approximated as a circle, where the radius represented the maximum distance from any given corner to the base footprint.

As shown in Figure 5.2, the occupied area of each robot was represented by its larger diagonal. For the tricycle robot, this approach did not yield an optimal representation, as the entire area enclosed by the diagonal was considered representative of the robot's occupied area.

While this implementation was simpler, it proved to be less accurate, as it treated the robot as if it could be oriented in any direction. To improve accuracy, the later stages of the code development entailed a transition to a more accurate rectangular representation, as illustrated in Figure 5.3.

With this implementation, the robot's area would precisely correspond to its base footprint, potentially producing improved results. However, it is crucial to consider specific aspects for each

robot type to ensure that the robot's orientation at each vertex aligns accordingly:

1. Omnidirectional:

The robot's orientation at each vertex aligns with the vertex's $theta_{holonomic}$.

2. Differential:

The robot's orientation at each vertex aligns with the vertex's $theta$.

3. Tricycle:

The robot's orientation at each vertex aligns with the vertex's $theta$.

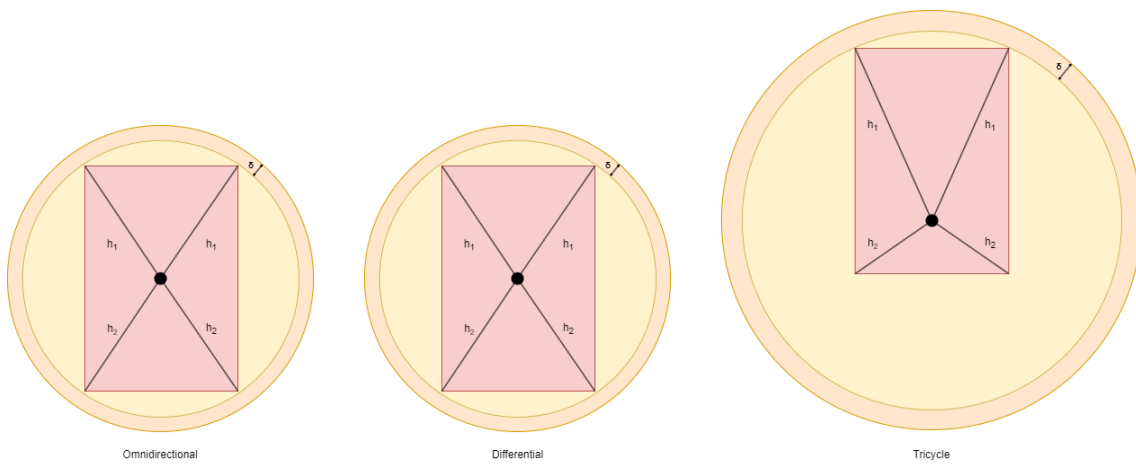


Figure 5.2: Robot's Area: Circular Shape

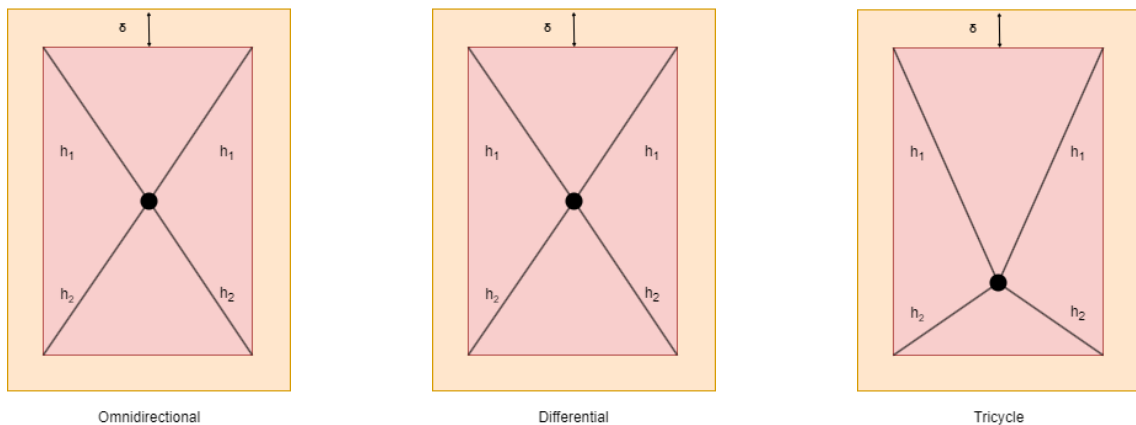


Figure 5.3: Robot's Area: Rectangular Shape

The information regarding the vertices (id , $position$, $orientation$) is extracted from the file named *trajectory_data.yaml*. This file includes the parameters for the previously mentioned orientations ($theta$ and $theta_{holonomic}$).

The content of *trajectory_data.yaml* is as follows:

```

Edges:
- {CurveType: spline, Destination_ID: 1, Id: 17, Origin_ID: 7,
  ParamB: 1.5, ParamF: 1.5,
  VelocityBackwards: 0.0, VelocityForward: 0.5}
- {CurveType: spline, Destination_ID: 2, Id: 12, Origin_ID: 1,
  ParamB: 1.5, ParamF: 1.5,
  VelocityBackwards: 0.5, VelocityForward: 0.5}
Vertices:
- {FrameId: map/nn0, Id: 1, Label: pass, Theta: 1.5707963267,
  ThetaHolomonic: 1.5699165535, X: 29.0, Y: 15.5}
- {FrameId: map/nn0, Id: 2, Label: pass, Theta: 3.1415926535,
  ThetaHolomonic: 0.0, X: 27.0, Y: 17.5}

```

The "Edges" section contains information about the edges, including curve types, destination IDs, and velocities. The "Vertices" section lists the vertices' details, such as frame IDs, labels, orientations (theta and theta_holomonic), and positions (X and Y).

Represented in Figure 5.4 is an example of a vertex along with its corresponding orientations.

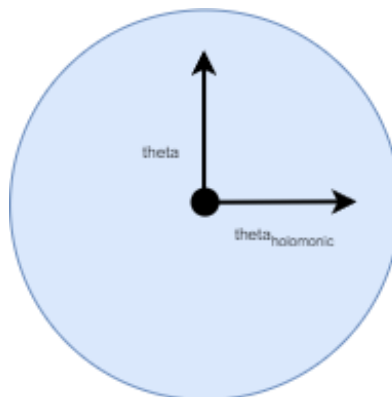


Figure 5.4: Vertex's orientations

Each vertex assumes a specific robot orientation as mentioned earlier. By hypothetically placing the robot at the centre of each vertex, aligned with its base footprint, the various robot types produce distinct outputs. For the preceding exemplified vertex, the resulting output would resemble that illustrated in Figure 5.5, with a robot larger than the vertex and centred within its base footprint.

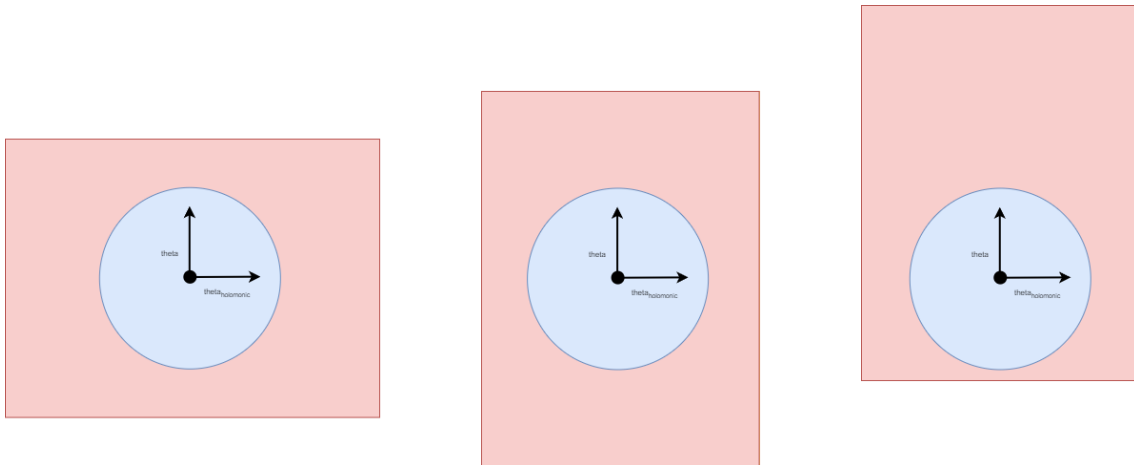


Figure 5.5: Robot positioned according to vertex's orientations

5.1.2 Algorithm performed

The algorithm's functioning relies on the extraction of wall coordinates, which demands an understanding of the process involved. The mapping of the environment is carried out using the occupancy grid mapping method. This approach entails representing the environment using a grid, where each cell corresponds to a specific area in the world. By extracting the grid's width, height, resolution and origin, the position of each cell can be determined. Additionally, each cell is assigned a value representing its state: 100 for occupied (wall), 0 for free, -1 for unknown or unexplored state. Consequently, the poses of all occupied cells are extracted into a vector, effectively containing a comprehensive set of points exemplifying the walls present in the world.

Initially, upon defining the robot's area, potential collisions are assessed between a hypothetical robot positioned at each vertex and the array of wall points.

If a collision is detected, the point closest to the robot's base footprint, denoted as C_1 , is determined. Subsequently, the closest wall point to C_1 , referred to as C_2 , is identified. A straight line is then drawn, representing the wall containing those two points. Utilising those points, the equation of the form $y = mx + b$ is obtained.

Then, a perpendicular straight line is established, passing through the base footprint of the robot and intersecting the previous line. This additional line determines the direction for the vertex to move away from the wall. The distance the vertex moves is calculated as the sum of the distance the robot was inside the wall and 20% of the robot's length.

The algorithm rechecks for collisions, and if one is detected, it iterates again to explore different scenarios. For instance, in the case of a corner, the robot must first distance itself from one wall and then from the other.

A final collision check is conducted in the algorithm. If it persists, it indicates that the robot cannot fit in that location, when the docking station is poorly placed in a narrow corridor, and the program provides feedback to the user through the console. The user is then advised to select an alternative location for the docking station. The pseudo-code below summarises the described

process.

Algorithm 1 Readjustment of Vertices' Positions

```

1: Read all wall points
2: Get the robot's dimensions
3: For all critical points
    Get a hypothetical robot in vertex and get corners with the robot's orientation
    For all wall points
        If collision
            Get closest wall point:  $C_1$ 
            Get closest wall point to  $C_1$ :  $C_2$ 
            Trace straight line through  $C_1$  and  $C_2$ :  $s_1$ 
            Trace perpendicular to  $s_1$  passing through base footprint:  $p_1$ 
            Check which corners are out of the map
            Get distance to wall from those corners:  $d_1$ 
            Move vertex in direction of  $p_1$  at distance  $d_1 + 20\%$  robot's length
        If still collision
            Get closest wall point:  $C_1$ 
            Get closest wall point to  $C_1$ :  $C_2$ 
            Trace straight line through  $C_1$  and  $C_2$ :  $s_1$ 
            Trace perpendicular to  $s_1$  passing through base footprint:  $p_1$ 
            Check which corners are out of the map
            Get distance to wall from those corners:  $d_1$ 
            Move vertex in direction of  $p_1$  at distance  $d_1 + 20\%$  robot's length
        If still collision
            Output a message that can't place the vertex in that narrow corridor
  
```

In addition to the pseudo-code demonstrating the sequential algorithm, Figure 5.6 provides a visual representation of the same algorithm using images, enhancing comprehension and facilitating a better understanding.

This ROS node includes an additional feature and implementation that is closely related to the following node but can be beneficial for other nodes as well.

Firstly, the node subscribes to a topic of type `<itrci_nav::vertex>` to receive information about a specific vertex. Upon receiving the vertex information from the other node, it checks whether the vertex collides with any wall. If a collision is detected, the node publishes a string message with the value "Collision" to a designated topic. Conversely, if no collision is found with any wall cells, it publishes the string message "NoCollision" to the same topic. This implementation proves invaluable for other nodes seeking to validate both offline docking vertices and, as implemented in the other developed node, online vertices, created in real-time. Further elaboration on this feature will be provided in subsequent sections of this document.

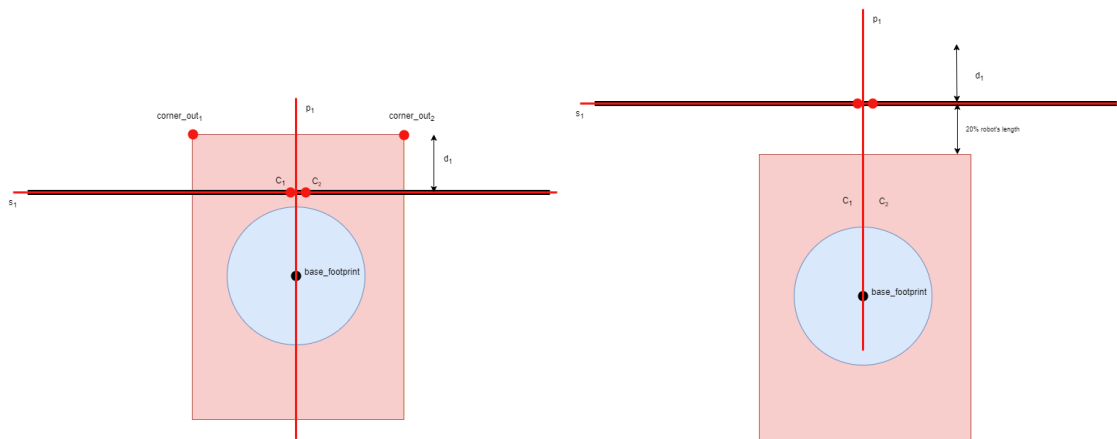


Figure 5.6: Visual representation of the algorithm

5.2 Algorithm for obstacle avoidance

This section delves into the methodology and algorithm designed to address obstacle avoidance challenges. The preparatory steps and essential considerations taken into account during the algorithm's development will be outlined. Additionally, a comprehensive explanation of how the obstacle avoidance algorithm performs in different scenarios will be provided.

5.2.1 Preparatory Steps and Consideration

Regarding the obstacle avoidance problem, the primary goal was to enable the robot to avoid encountering obstacles by planning an alternative route that takes the obstacle into account while considering the robot's base footprint.

To achieve this, several steps were involved. Firstly, the obstacle needed to be detected, and the robot's position and distance to the obstacle had to be determined. The obstacle detection and distance measurement were accomplished using laser sensors. When the robot detected the obstacle at a certain distance, a vertex was created in that position, and another vertex was generated to encircle the obstacle. This second vertex was placed at a distance equal to the robot's radius (obtained from a topic published from the previous node), to ensure that there were not any collisions in the course of the contouring. Additionally, two edges were formed: one connecting the first vertex (V_1) to the second vertex (V_2), and the other connecting (V_2) to the next vertex in the robot's original trajectory.

Although there is a topic displaying the robot's trajectory, it was deemed worthwhile to consider creating temporary vertices and edges during the obstacle avoidance process to ensure no duplicates with the same ID were generated. After collision avoidance was successfully completed, these temporary vertices and edges were promptly removed from the system.

5.2.2 Algorithm performed

In the initial phases of the project, the algorithm and thought process closely resembled the one presented in Chapter 3.1. However, in this specific scenario, the base footprint was taken into consideration rather than the mass centre. As the project progressed, an enhancement to this algorithm was implemented in its final stages. The first approach, closely resembling the final one, will be explained initially, followed by a detailed explanation of the implemented approach.

5.2.2.1 First developed algorithm

When an obstacle was detected at a distance d using either the front laser (while moving forward) or the back laser (while moving backwards), the robot would stop, as shown in Figure 5.7.

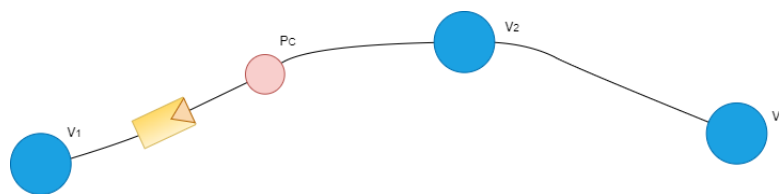


Figure 5.7: First obstacle detection algorithm

In this scenario, a vertex named V_{1_tmp} is created at the current pose of the robot. Subsequently, the position of the closest point of the obstacle to the robot is determined, referred to as the "point of collision" (P_C). A straight line, denoted as s_1 , is drawn between V_{1_tmp} and P_C . Then, a perpendicular straight line to s_1 , passing through P_C , is traced, and another vertex named V_{2_tmp} is generated on this line at a distance d from P_C . The purpose of V_{2_tmp} is to navigate around the obstacle. After circumventing the obstacle, the robot proceeds towards the next vertex on its original path. Figure 5.8 offers a visual representation of the previously mentioned concept.

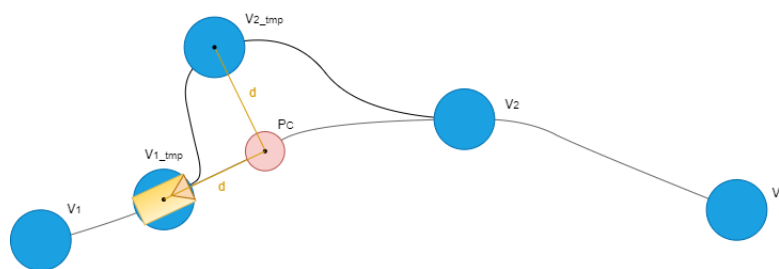


Figure 5.8: First algorithm designed for obstacle avoidance from the top

The alternative path (from V_{1_tmp} to V_{2_tmp} , and finally V_2) is forwarded to the controller, developed by *CRISS*. With this approach, it is possible to switch from the original route to the alternative path created. Additionally, the previous node developed for the docking process subscribes to a ROS topic published by this node that checks for collisions with walls. This step is essential to avoid potential collisions with the robot. If a collision is detected, a new attempt is made to move the robot at the same distance d , but in the opposite direction, as illustrated in Figure 5.9.

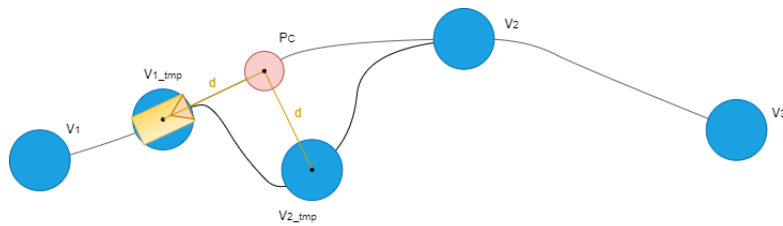


Figure 5.9: First algorithm designed for obstacle avoidance from the bottom

If the alternative path still resulted in a collision with the wall, the distance d was adjusted accordingly. In such cases, the distance $2*d$ was chosen initially, and if that distance still caused a collision, it was tested at the same distance in the opposite direction. If neither of these attempts proved successful, no new vertex was placed and the robot would wait for the obstacle removal.

5.2.2.2 Final developed algorithm

The previous algorithm was implemented during an early stage to streamline the problem. However, it is not an optimal solution. While it may work when the obstacle is represented by a single point, it becomes less effective when dealing with obstacles of length, such as walls shown in Figure 5.10, making it not optimal.

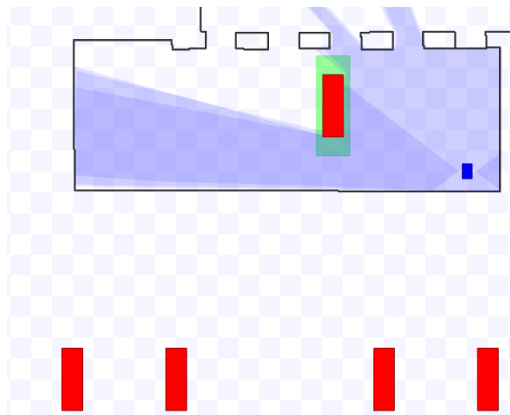


Figure 5.10: Test Environment Walls

Considering a scenario where a wall is perfectly centred with the middle laser beam and has a width of $1.2*distance$. In the previous algorithm, a vertex would be placed in collision with the obstacle since the first node solely detects walls mapped in the occupancy grid and not laser readings, leading to the robot's collision with the wall. Even if the first node were to detect walls using laser readings, the algorithm would fail for the first two vertex positions at distance d on both sides.

Ultimately, a vertex would be placed $2*d$ from the further obstacle point. This approach is evidently sub-optimal, and it would be better to recognise the entire obstacle as the whole set of laser readings.

To improve this, the set of laser readings was divided into clusters, grouping points that are close to each other within a specified Euclidean distance. Clusters that are in proximity to each other within a given distance were merged. Consequently, the points representing the boundaries of the cluster observed in the middle laser beam were extracted (initial and final points). This allows the obstacle avoidance process to be optimised for the specific shape of the obstacle.

In summary, when the robot's middle laser beam detects an obstacle at a distance d , the coordinates of point P_C corresponding to that beam are stored and the robot is stopped, as illustrated in Figure 5.11.

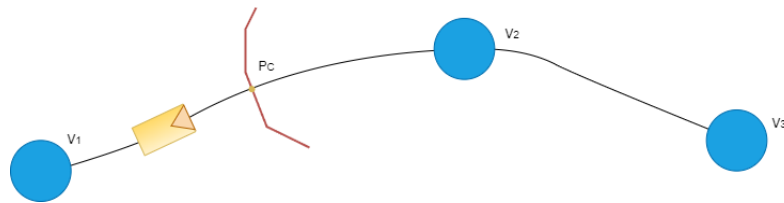


Figure 5.11: Final obstacle detection algorithm

Subsequently, the robot enters a waiting state for a period of 5 seconds, anticipating the removal of the obstacle. If the obstacle is successfully removed within this time frame, the robot proceeds along its original path without any further action. However, if the obstacle persists beyond the 5-second waiting period, the robot initiates its obstacle avoidance routine to navigate around the obstruction.

In that case, a vertex is created at the pose of the robot, P_1 , and a straight line is drawn from P_1 to the initial point of the cluster. Another vertex is placed on this straight line at a specified distance (radius received from the aforementioned node), aiming for the best and most optimal position possible, as shown in Figure 5.12.

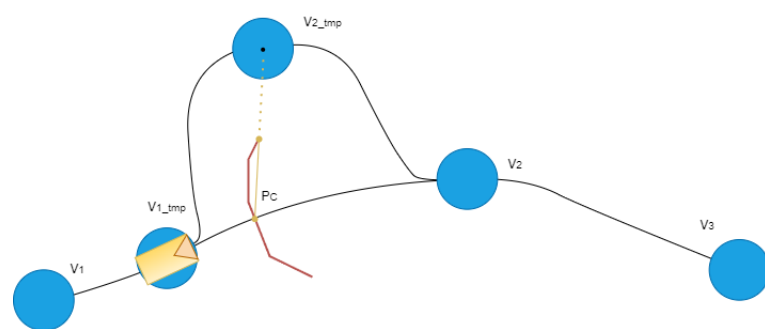


Figure 5.12: Final algorithm designed for obstacle avoidance from the top

Afterwards, the previously mentioned node designed to avoid placing vertices near walls is executed. If the vertex created is found to be in collision with the wall, the same algorithm is applied, but this time for the final point of the cluster (the other side of the obstacle), as shown in Figure 5.13. This process ensures that the robot optimally avoids obstacles and positions itself in the best possible location along the cluster's boundary.

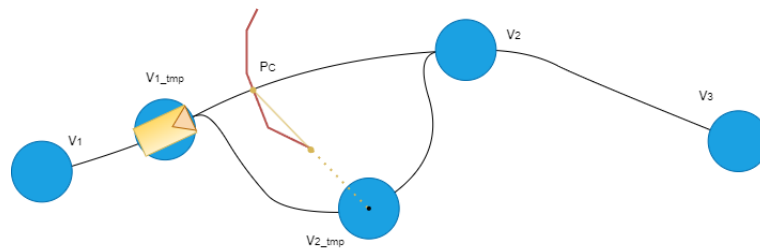


Figure 5.13: Final algorithm designed for obstacle avoidance from the bottom

If, even in that case, the vertex created for the alternative route is placed near a wall, then the robot will wait for the obstacle to be removed. Since avoidance of the obstacle from either side is not possible, the robot needs to ensure a safe distance from the wall before proceeding with its navigation.

In order to achieve these functionalities, the process begins by extracting the original path set from an existing topic. Upon detecting an obstacle, the algorithm subscribes to another topic providing information about the current edge. The algorithm then accesses the stored original path set, appends the two new edges from the alternative path, and subsequently adds the remaining positions necessary to reach the final destination. Finally, the updated path set is published to the controller, enabling the robot to navigate along the newly adjusted trajectory.

Chapter 6

Experimental Results and Analysis

6.1 Readjustment of critical point's (docking) position

First and foremost, various tests were conducted concerning the docking node that was created. In Figure 6.1, the original graph trajectory from a laboratory in FEUP is represented in RViz.

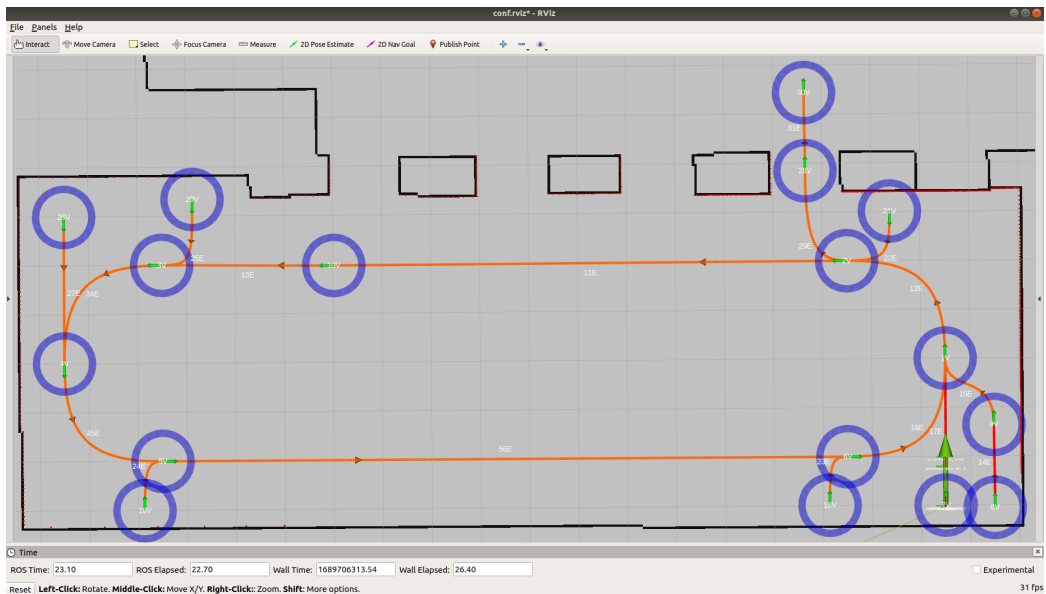


Figure 6.1: Original trajectory in RViz

To assess the behaviour of the created node with different types of robots, the positions of certain vertices were readjusted closer to walls on several occasions. Specifically, the positions of the vertices 20, 26, 4, 19, 18, 8, 9 and 21 were modified to be closer to walls, aiming to test the algorithm's performance under various scenarios available in the simulation. These scenarios included being near the top wall, top-left corner, left wall, bottom-left corner, bottom wall, bottom-right wall, right wall, and top-right corner, respectively.

Additionally, two new vertices with IDs 100 and 101 were added: one placed between two pillars and the other confined to a small compartment. The outcomes of these adjustments are

presented in Figure 6.2 and served as a foundation for the subsequent tests that were conducted. All of the following tests were performed for different types of robots, namely omnidirectional, differential, and tricycle.

For better presentation of the results, the outcomes from the left side (left wall, top-left and bottom-left corners, as well as the top wall) are presented in one table, while the results from the right side (right wall, top-right and bottom-right corners, as well as the bottom wall) are displayed in another table.

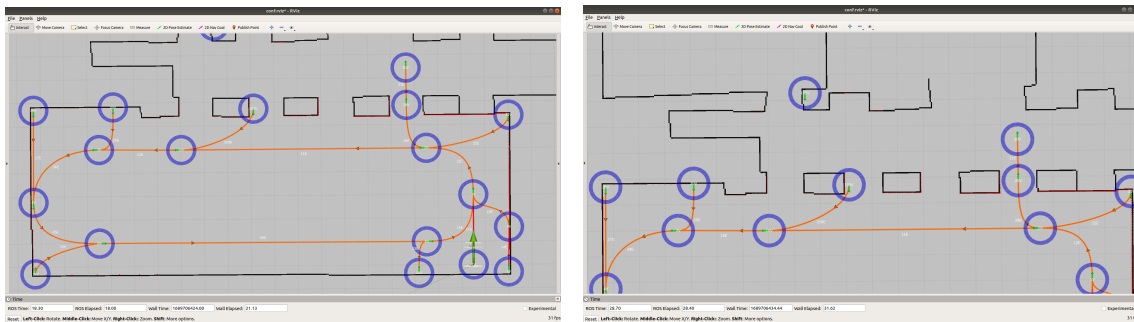


Figure 6.2: Testing Environment

Furthermore, a separate table is provided for the middle section, containing the isolated vertex placed between four walls and the vertex between two pillars. Additionally, there is a final table displaying the output of the program in the console, presenting parameters such as the robot's size, type of robot, base footprint coordinates, and details of vertices that collided with the wall. In case of a placement where the robot doesn't fit, an appropriate message indicating that the robot doesn't fit is included.

6.1.1 Test 1: Testing for different robot's areas - Circle vs. Rectangle

6.1.1.1 Setup

In order to compare the area occupied by the robot when initially considering a circle and later optimising it to a rectangle, an interesting test was conducted to assess whether the theoretical assumptions align with the experimental results. This comparison was carried out for all three types of robots, both for the robot's original size and for double the original size (1x and 2x).

The tests involved calculating the area occupied by the robot using both the circle and rectangle representations. By evaluating these results, valuable insights were gained regarding the accuracy and effectiveness of each representation. This comprehensive analysis was essential to validate the theoretical premises and their practical implications in real-world scenarios for different robot sizes and types.

6.1.1.2 Results

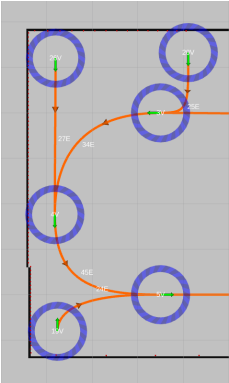
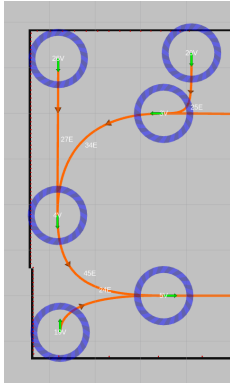
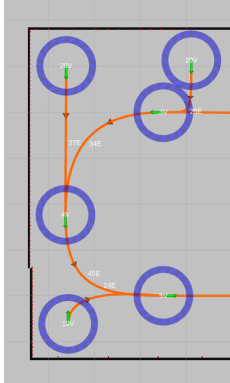
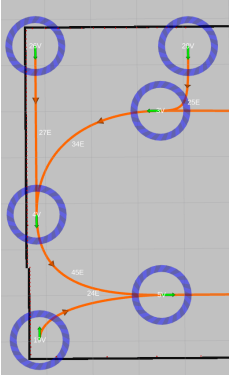
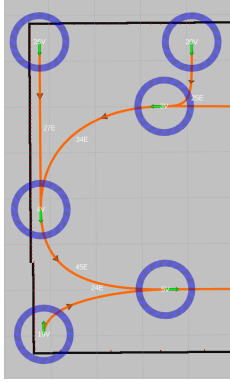
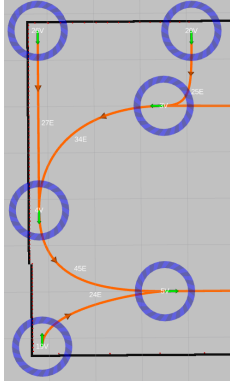
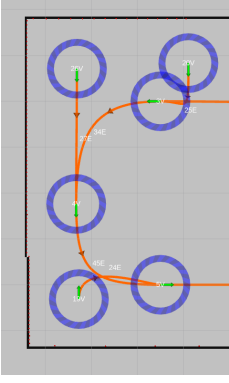
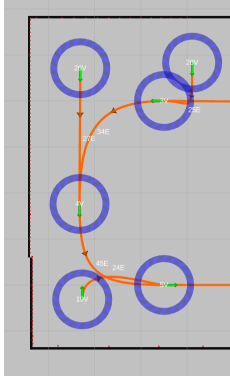
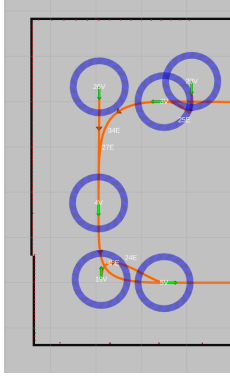
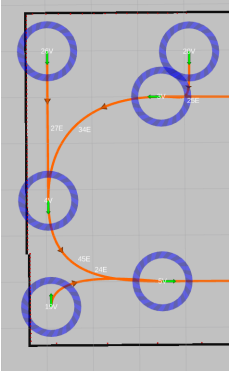
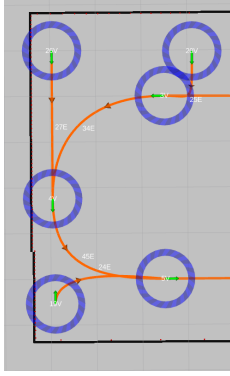
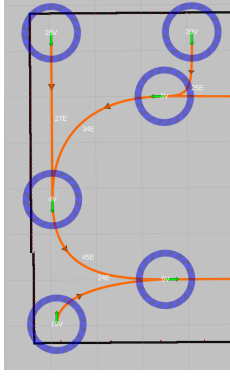
Left side				
Size	Area	Omnidirectional	Differential	Tricycle
1x	Circle			
	Rectangle			
2x	Circle			
	Rectangle			

Table 6.1: Circle vs Rectangle - Left Side

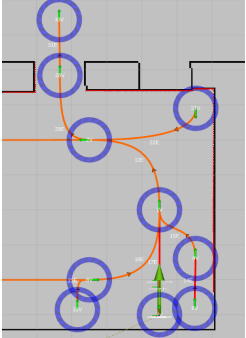
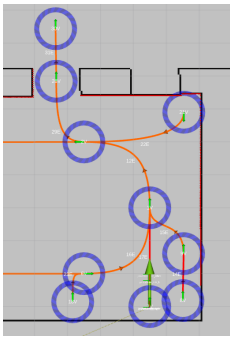
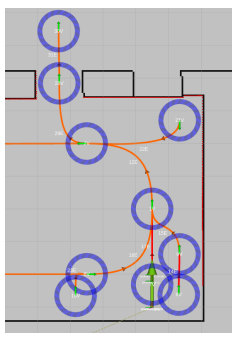
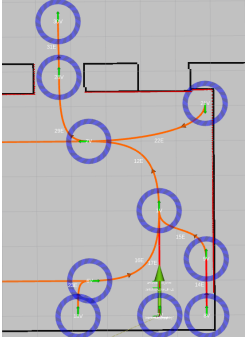
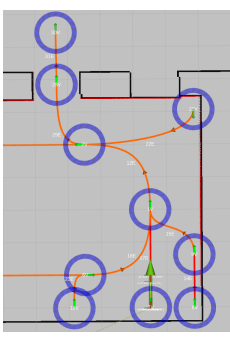
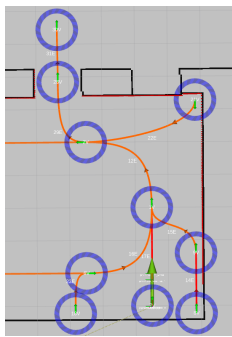
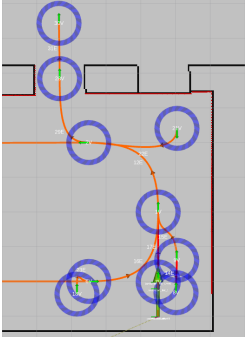
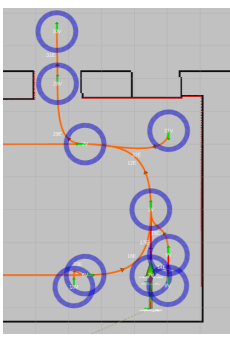
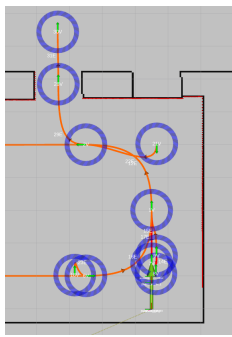
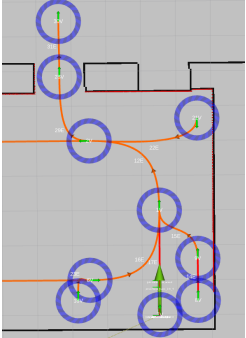
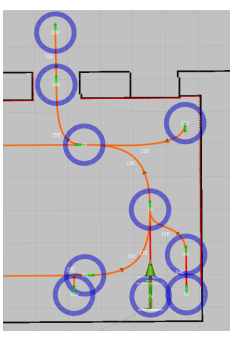
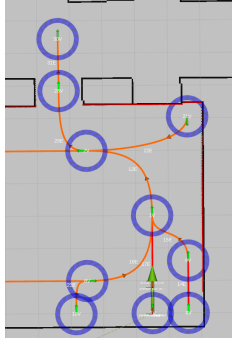
		Right side		
Size	Area	Omnidirectional	Differential	Tricycle
1x	Circle			
	Rectangle			
2x	Circle			
	Rectangle			

Table 6.2: Circle vs Rectangle - Right Side

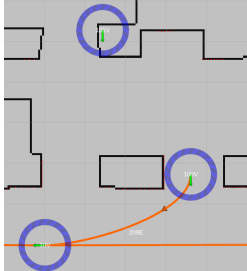
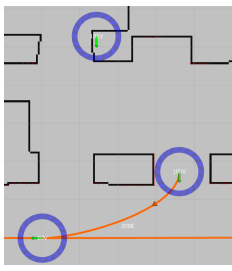
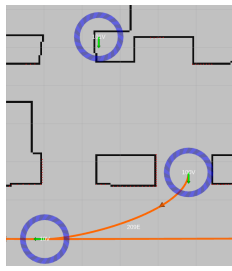
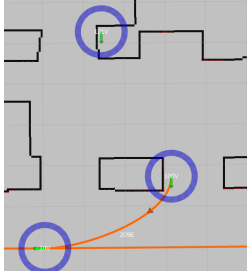
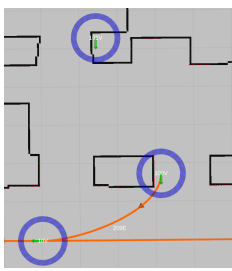
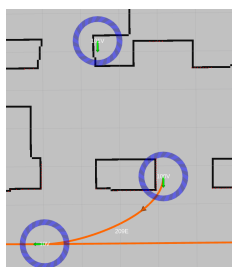
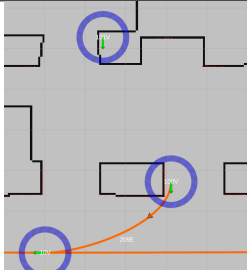
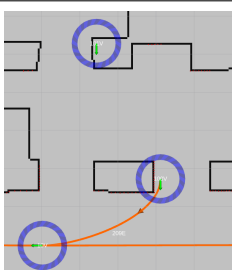
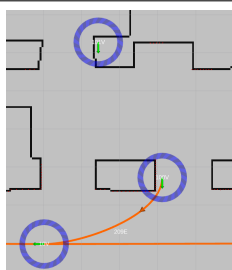
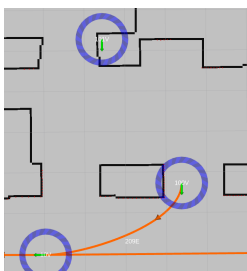
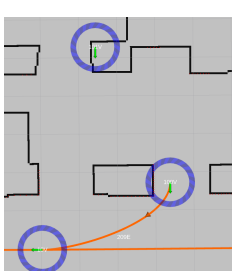
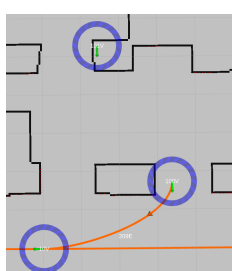
Middle Section				
Size	Area	Omnidirectional	Differential	Tricycle
1x	Circle			
	Rectangle			
2x	Circle			
	Rectangle			

Table 6.3: Circle vs Rectangle - Middle Section

Console Output				
Size	Area	Omnidirectional	Differential	Tricycle
1x	Circle	<pre>[INFO] [1689863625.92811768, 5.700000000]: COLLISION with vertex 9!! [INFO] [1689863625.938942368, 5.700000000]: COLLISION with vertex 9!! [INFO] [1689863625.937088414, 5.700000000]: COLLISION with vertex 21!! [INFO] [1689863625.935456692, 5.700000000]: COLLISION with vertex 20!! [INFO] [1689863625.941266158, 5.700000000]: COLLISION with vertex 20!! [INFO] [1689863625.942660100, 5.700000000]: COLLISION with vertex 4!! [INFO] [1689863625.947133038, 5.700000000]: COLLISION with vertex 19!! [INFO] [1689863625.95106613, 5.700000000]: COLLISION with vertex 18!! [INFO] [1689863625.952250689, 5.700000000]: COLLISION with vertex 100!! [INFO] [1689863625.95188965, 5.700000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689863827.838299670, 7.300000000]: COLLISION with vertex 9!! [INFO] [1689863827.840284912, 7.300000000]: COLLISION with vertex 9!! [INFO] [1689863827.846024092, 7.300000000]: COLLISION with vertex 21!! [INFO] [1689863827.852933571, 7.300000000]: COLLISION with vertex 20!! [INFO] [1689863827.849944300, 7.300000000]: COLLISION with vertex 20!! [INFO] [1689863827.862266630, 7.300000000]: COLLISION with vertex 18!! [INFO] [1689863827.864005099, 7.300000000]: COLLISION with vertex 18!! [INFO] [1689863827.862266630, 7.300000000]: COLLISION with vertex 100!! [INFO] [1689863827.863657954, 7.300000000]: COLLISION with vertex 10!! [INFO] [1689863827.865699152, 7.300000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689864453.427584555, 22.700000000]: COLLISION with vertex 9!! [INFO] [1689864453.429087744, 22.700000000]: COLLISION with vertex 8!! [INFO] [1689864453.431882513, 22.700000000]: COLLISION with vertex 7!! [INFO] [1689864453.435238206, 22.700000000]: COLLISION with vertex 21!! [INFO] [1689864453.437489105, 22.700000000]: COLLISION with vertex 20!! [INFO] [1689864453.439043775, 22.700000000]: COLLISION with vertex 20!! [INFO] [1689864453.443298530, 22.700000000]: COLLISION with vertex 4!! [INFO] [1689864453.446092424, 22.700000000]: COLLISION with vertex 19!! [INFO] [1689864453.449498186, 22.700000000]: COLLISION with vertex 18!! [INFO] [1689864453.451709027, 22.700000000]: COLLISION with vertex 10!! [INFO] [1689864453.453315363, 22.700000000]: COLLISION with vertex 10!! [INFO] [1689864453.454762487, 22.700000000]: Please chose a better location for vertex 10!</pre>
	Rectangle	<pre>[INFO] [168977526.107143264, 139.300000000]: COLLISION with vertex 9!! [INFO] [168977526.108551975, 139.300000000]: COLLISION with vertex 8!! [INFO] [168977526.112829308, 139.300000000]: COLLISION with vertex 21!! [INFO] [168977526.114529282, 139.300000000]: COLLISION with vertex 20!! [INFO] [168977526.115523091, 139.300000000]: COLLISION with vertex 4!! [INFO] [168977526.119073457, 139.300000000]: COLLISION with vertex 19!! [INFO] [168977526.121896924, 139.300000000]: COLLISION with vertex 18!! [INFO] [168977526.124476087, 139.300000000]: COLLISION with vertex 100!! [INFO] [168977526.125937533, 139.300000000]: COLLISION with vertex 10!! [INFO] [168977526.127019336, 139.300000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689783015.767068071, 6.000000000]: COLLISION with vertex 9!! [INFO] [1689783015.768775759, 6.000000000]: COLLISION with vertex 8!! [INFO] [1689783015.774288775, 6.000000000]: COLLISION with vertex 21!! [INFO] [1689783015.779297488, 6.000000000]: COLLISION with vertex 20!! [INFO] [1689783015.777709433, 6.000000000]: COLLISION with vertex 20!! [INFO] [1689783015.779717682, 6.000000000]: COLLISION with vertex 4!! [INFO] [1689783015.782199980, 6.000000000]: COLLISION with vertex 19!! [INFO] [1689783015.784262357, 6.000000000]: COLLISION with vertex 18!! [INFO] [1689783015.785793324, 6.000000000]: COLLISION with vertex 100!! [INFO] [1689783015.786893213, 6.000000000]: COLLISION with vertex 10!! [INFO] [1689783015.788488821, 6.000000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689784870.653620707, 5.500000000]: COLLISION with vertex 9!! [INFO] [1689784870.654892763, 5.500000000]: COLLISION with vertex 8!! [INFO] [1689784870.659275823, 5.500000000]: COLLISION with vertex 21!! [INFO] [1689784870.664094113, 5.500000000]: COLLISION with vertex 20!! [INFO] [1689784870.662221631, 5.500000000]: COLLISION with vertex 20!! [INFO] [1689784870.663995967, 5.500000000]: COLLISION with vertex 4!! [INFO] [1689784870.666094662, 5.500000000]: COLLISION with vertex 19!! [INFO] [1689784870.669278248, 5.500000000]: COLLISION with vertex 18!! [INFO] [1689784870.672322008, 5.500000000]: COLLISION with vertex 100!! [INFO] [1689784870.673023179, 5.500000000]: COLLISION with vertex 10!! [INFO] [1689784870.674498145, 5.500000000]: Please chose a better location for vertex 10!</pre>
2x	Circle	<pre>[INFO] [168986167.85230782, 12.400000000]: COLLISION with vertex 9!! [INFO] [168986167.854862729, 12.400000000]: COLLISION with vertex 8!! [INFO] [168986167.857961842, 12.400000000]: COLLISION with vertex 7!! [INFO] [168986167.862942105, 12.400000000]: COLLISION with vertex 21!! [INFO] [168986167.864885519, 12.400000000]: Please chose a better location for vertex 28. [INFO] [168986167.865899025, 12.400000000]: COLLISION with vertex 21!! [INFO] [168986167.868284609, 12.400000000]: COLLISION with vertex 20!! [INFO] [168986167.869807182, 12.400000000]: COLLISION with vertex 20!! [INFO] [168986167.87176787, 12.400000000]: COLLISION with vertex 4!! [INFO] [168986167.876164256, 12.400000000]: COLLISION with vertex 19!! [INFO] [168986167.880185365, 12.400000000]: COLLISION with vertex 18!! [INFO] [168986167.88349318, 12.400000000]: Please chose a better location for vertex 100. [INFO] [168986167.88322697, 12.400000000]: COLLISION with vertex 10!! [INFO] [168986167.885599463, 12.400000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689865905.076256929, 9.900000000]: COLLISION with vertex 9!! [INFO] [1689865905.078393340, 9.900000000]: COLLISION with vertex 8!! [INFO] [1689865905.081143931, 9.900000000]: COLLISION with vertex 7!! [INFO] [1689865905.084733747, 9.900000000]: COLLISION with vertex 21!! [INFO] [1689865905.086825984, 9.900000000]: Please chose a better location for vertex 28. [INFO] [1689865905.089737168, 9.900000000]: COLLISION with vertex 21!! [INFO] [1689865905.092555789, 9.900000000]: COLLISION with vertex 20!! [INFO] [1689865905.09277649, 9.900000000]: COLLISION with vertex 20!! [INFO] [1689865905.098210329, 9.900000000]: COLLISION with vertex 19!! [INFO] [1689865905.103988993, 9.900000000]: COLLISION with vertex 18!! [INFO] [1689865905.109007394, 9.900000000]: COLLISION with vertex 100!! [INFO] [1689865905.111008949, 9.900000000]: Please chose a better location for vertex 100. [INFO] [1689865905.111905074, 9.900000000]: COLLISION with vertex 10!! [INFO] [1689865905.113689707, 9.900000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689865212.32422173, 7.500000000]: COLLISION with vertex 9!! [INFO] [1689865212.326393493, 7.500000000]: COLLISION with vertex 8!! [INFO] [1689865212.330177079, 7.500000000]: COLLISION with vertex 7!! [INFO] [1689865212.334698540, 7.500000000]: COLLISION with vertex 21!! [INFO] [1689865212.3367735, 7.500000000]: Please chose a better location for vertex 28. [INFO] [1689865212.33760265, 7.500000000]: COLLISION with vertex 21!! [INFO] [1689865212.340708412, 7.500000000]: COLLISION with vertex 20!! [INFO] [1689865212.342931997, 7.500000000]: COLLISION with vertex 20!! [INFO] [1689865212.344446878, 7.500000000]: COLLISION with vertex 4!! [INFO] [1689865212.350479372, 7.500000000]: COLLISION with vertex 19!! [INFO] [1689865212.354411331, 7.500000000]: COLLISION with vertex 18!! [INFO] [1689865212.356898723, 7.500000000]: COLLISION with vertex 100!! [INFO] [1689865212.358961575, 7.500000000]: Please chose a better location for vertex 100. [INFO] [1689865212.35910054, 7.500000000]: COLLISION with vertex 10!! [INFO] [1689865212.36221123, 7.500000000]: Please chose a better location for vertex 10!</pre>
	Rectangle	<pre>[INFO] [1689782709.738177652, 6.000000000]: COLLISION with vertex 9!! [INFO] [1689782709.731463252, 6.000000000]: COLLISION with vertex 8!! [INFO] [1689782709.736073198, 6.000000000]: COLLISION with vertex 21!! [INFO] [1689782709.739707262, 6.000000000]: COLLISION with vertex 20!! [INFO] [1689782709.740934962, 6.000000000]: COLLISION with vertex 20!! [INFO] [1689782709.742511265, 6.000000000]: COLLISION with vertex 4!! [INFO] [1689782709.744924116, 6.000000000]: COLLISION with vertex 19!! [INFO] [1689782709.746590933, 6.000000000]: COLLISION with vertex 18!! [INFO] [1689782709.749226813, 6.000000000]: COLLISION with vertex 100!! [INFO] [1689782709.75293238, 6.000000000]: COLLISION with vertex 10!! [INFO] [1689782709.751618569, 6.000000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689783852.030794388, 5.100000000]: COLLISION with vertex 9!! [INFO] [1689783852.032106212, 5.100000000]: COLLISION with vertex 8!! [INFO] [1689783852.034507193, 5.100000000]: COLLISION with vertex 7!! [INFO] [1689783852.037375481, 5.100000000]: COLLISION with vertex 21!! [INFO] [1689783852.039077143, 5.100000000]: COLLISION with vertex 20!! [INFO] [1689783852.040170142, 5.100000000]: COLLISION with vertex 20!! [INFO] [1689783852.041836318, 5.100000000]: COLLISION with vertex 4!! [INFO] [1689783852.043373568, 5.100000000]: COLLISION with vertex 19!! [INFO] [1689783852.045715571, 5.100000000]: COLLISION with vertex 18!! [INFO] [1689783852.047073926, 5.100000000]: COLLISION with vertex 100!! [INFO] [1689783852.048078829, 5.100000000]: COLLISION with vertex 10!! [INFO] [1689783852.049177217, 5.100000000]: Please chose a better location for vertex 10!</pre>	<pre>[INFO] [1689785624.244563960, 5.300000000]: COLLISION with vertex 9!! [INFO] [1689785624.246203317, 5.300000000]: COLLISION with vertex 8!! [INFO] [1689785624.251811951, 5.300000000]: COLLISION with vertex 21!! [INFO] [1689785624.253760737, 5.300000000]: COLLISION with vertex 20!! [INFO] [1689785624.254938865, 5.300000000]: COLLISION with vertex 20!! [INFO] [1689785624.256954725, 5.300000000]: COLLISION with vertex 4!! [INFO] [1689785624.259500368, 5.300000000]: COLLISION with vertex 19!! [INFO] [1689785624.261420223, 5.300000000]: COLLISION with vertex 18!! [INFO] [1689785624.262960366, 5.300000000]: COLLISION with vertex 100!! [INFO] [1689785624.26411820, 5.300000000]: COLLISION with vertex 10!! [INFO] [1689785624.265319537, 5.300000000]: Please chose a better location for vertex 10!</pre>

Table 6.4: Circle vs Rectangle - Console Output

6.1.1.3 Analysis

As predicted, the node successfully adjusted the vertices that were positioned too close to the wall, regardless of whether the area was represented by a circle or a rectangle. Notably, it was observed that the circle representation tended to move the vertices farther away from the walls compared to the rectangle representation. It is worth emphasising that, as anticipated, the vertices moved more significantly away from certain walls depending on their orientation. For instance, considering vertex 26 (top-left corner) in Table 6.1, both in the original size (1x) and double size (2x) examples with a circle area, the vertex moved away equally from the top and left walls.

However, when represented as a rectangle, the robot exhibited a greater distance from the top wall compared to the left wall. This behaviour aligns with the expected outcome since the robot is represented by a rectangle oriented with $\theta = -\frac{\pi}{2}$ and $\theta_{holonomic} = -\frac{\pi}{2}$ at that vertex.

This observation is consistent across all vertices, where their orientations dictate the extent to which they move away from specific walls. The node is effectively accomplishing its intended task, demonstrating that it adjusts the vertices correctly based on their orientations.

6.1.2 Test 2: Extensive testing for all available possible positions: different sizes

6.1.2.1 Setup

After the initial test produced the expected outcome, where the circle representation moved too far away from the wall, subsequent tests were conducted solely considering the area of the robot as a representative rectangle.

Consequently, various scenarios were tested for different types of robots (differential, omnidirectional, and tricycle). For each type of robot, different robot sizes were compared (1x, 1.25x, 1.5x, 1.75x, 2x, and 3x to their original dimensions) to analyse the differences for varying robot sizes. The exaggerated size of 3x was included intentionally to better assess the results in a more extreme case. In Appendix A, detailed results are provided for all tested sizes, along with larger images.

For the purpose of this chapter's analysis and comparison, the images were cropped to facilitate better visualisation and comprehension, and only three different sizes (1x, 1.5x, and 2x) are shown.

6.1.2.2 Results

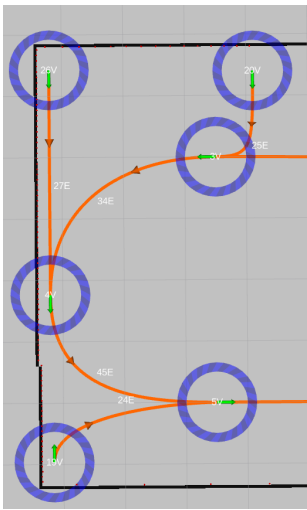
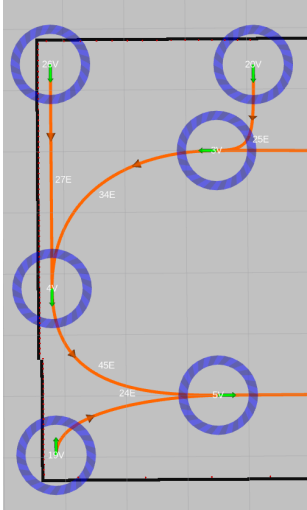
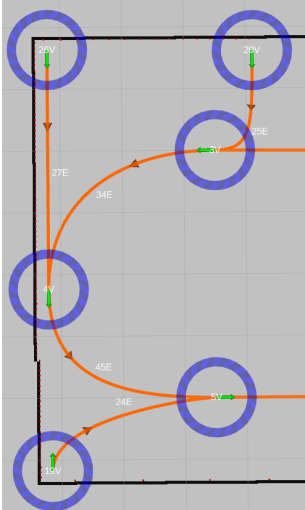
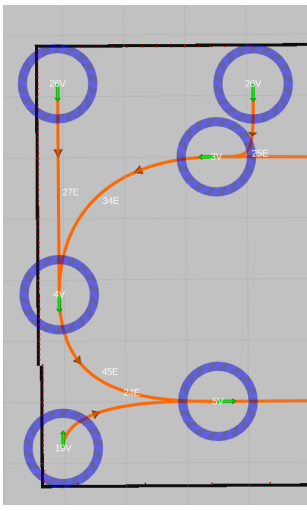
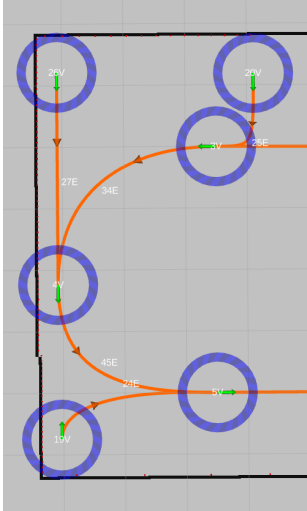
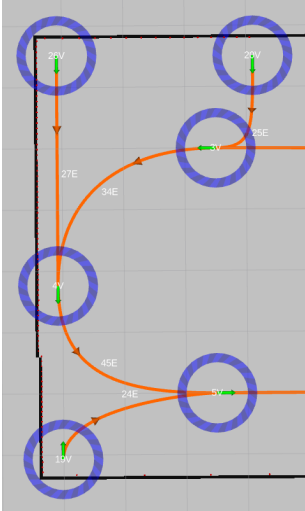
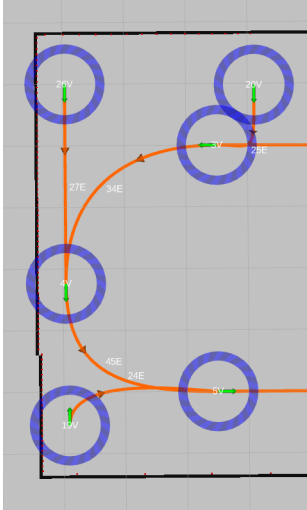
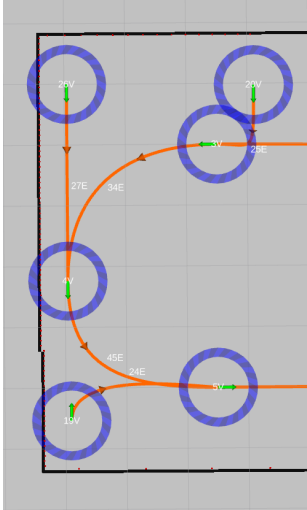
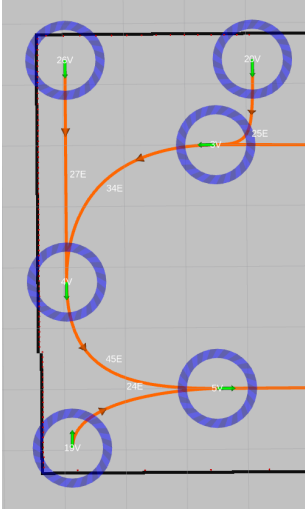
		Left side		
Size		Omnidirectional	Differential	Tricycle
1x				
	1.5x			
	2x			

Table 6.5: Different sizes: Left side

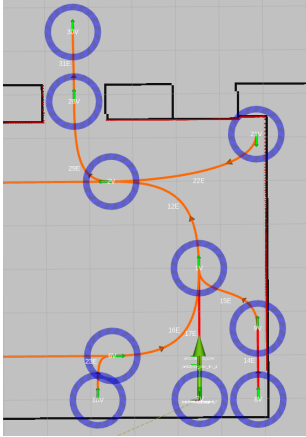
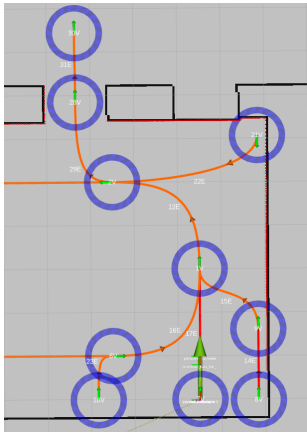
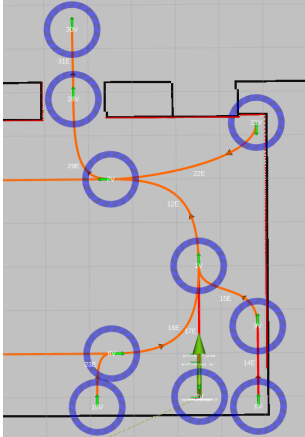
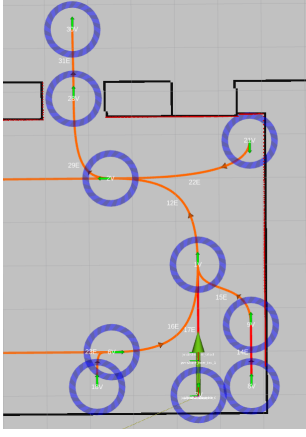
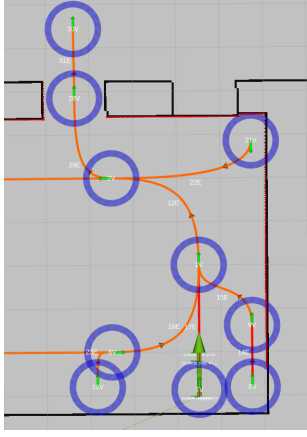
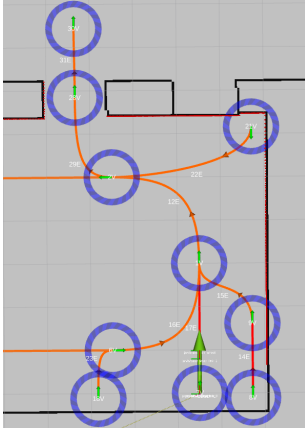
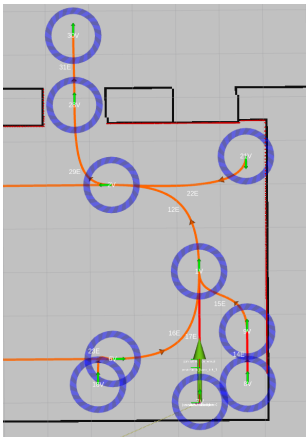
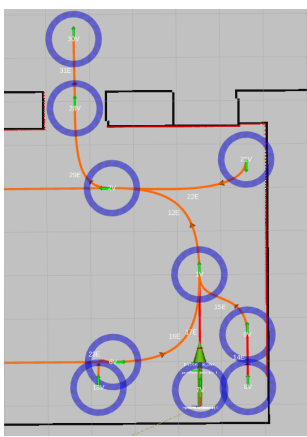
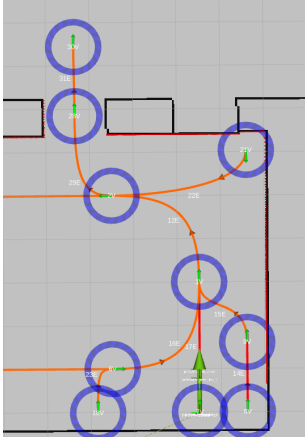
Right side			
Size	Omnidirectional	Differential	Tricycle
1x			
1.5x			
2x			

Table 6.6: Different sizes: Right side

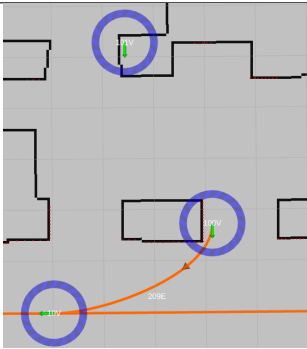
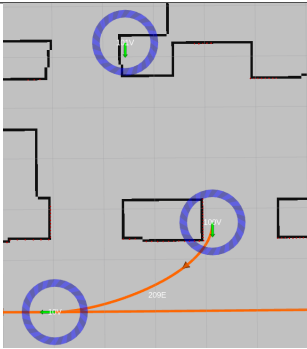
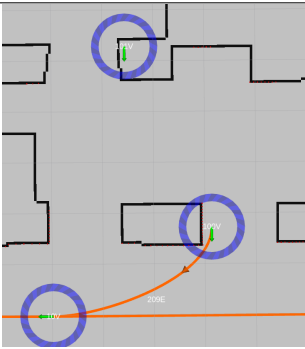
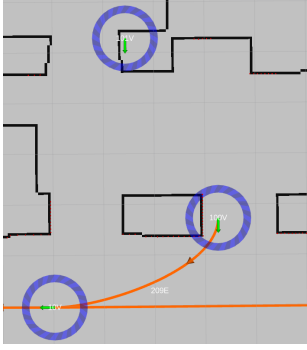
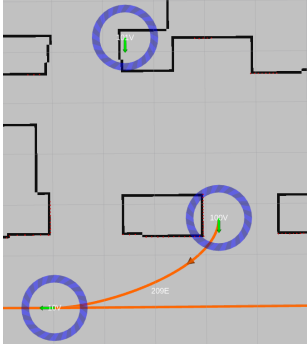
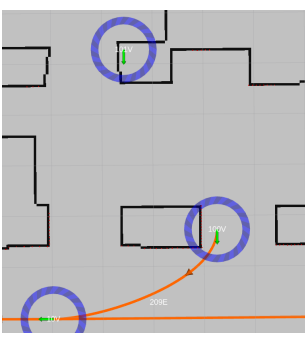
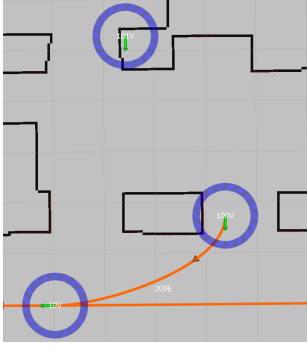
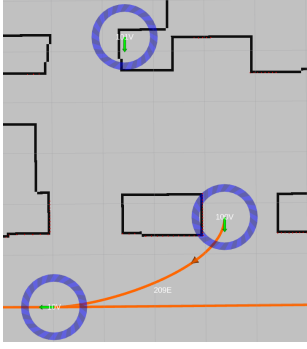
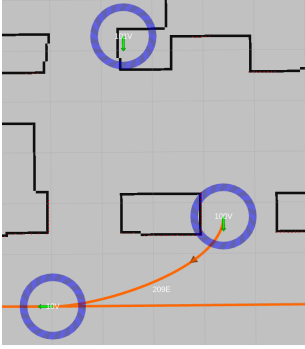
Middle section			
Size	Omnidirectional	Differential	Tricycle
1x			
			
			

Table 6.7: Different sizes: Middle section

Console Output			
Size	Omnidirectional	Differential	Tricycle
1x	<pre>[INFO] [168977526.107143264, 139.300000000]: COLLISION with vertex 9!! [INFO] [168977526.108551975, 139.300000000]: COLLISION with vertex 8!! [INFO] [168977526.112829308, 139.300000000]: COLLISION with vertex 21!! [INFO] [168977526.114520282, 139.300000000]: COLLISION with vertex 20!! [INFO] [168977526.115523091, 139.300000000]: COLLISION with vertex 26!! [INFO] [168977526.117088854, 139.300000000]: COLLISION with vertex 4!! [INFO] [168977526.119073457, 139.300000000]: COLLISION with vertex 19!! [INFO] [168977526.121896024, 139.300000000]: COLLISION with vertex 18!! [INFO] [168977526.124476087, 139.300000000]: COLLISION with vertex 100!! [INFO] [168977526.125937533, 139.300000000]: COLLISION with vertex 101!! [INFO] [168977526.127019336, 139.300000000]: Please chose a better location for vertex 101</pre>	<pre>[INFO] [1689783015.767068071, 6.000000000]: COLLISION with vertex 9!! [INFO] [1689783015.768717559, 6.000000000]: COLLISION with vertex 8!! [INFO] [1689783015.774288775, 6.000000000]: COLLISION with vertex 21!! [INFO] [1689783015.776507488, 6.000000000]: COLLISION with vertex 20!! [INFO] [1689783015.777709433, 6.000000000]: COLLISION with vertex 26!! [INFO] [1689783015.779717682, 6.000000000]: COLLISION with vertex 4!! [INFO] [1689783015.782199980, 6.000000000]: COLLISION with vertex 19!! [INFO] [1689783015.784262357, 6.000000000]: COLLISION with vertex 18!! [INFO] [1689783015.785793324, 6.000000000]: COLLISION with vertex 100!! [INFO] [1689783015.786893213, 6.000000000]: COLLISION with vertex 101!! [INFO] [1689783015.788108824, 6.000000000]: Please chose a better location for vertex 101</pre>	<pre>[INFO] [1689784870.653620767, 5.500000000]: COLLISION with vertex 9!! [INFO] [1689784870.654892763, 5.500000000]: COLLISION with vertex 8!! [INFO] [1689784870.659275823, 5.500000000]: COLLISION with vertex 21!! [INFO] [1689784870.661040413, 5.500000000]: COLLISION with vertex 20!! [INFO] [1689784870.662221631, 5.500000000]: COLLISION with vertex 26!! [INFO] [1689784870.663995967, 5.500000000]: COLLISION with vertex 4!! [INFO] [1689784870.666894662, 5.500000000]: COLLISION with vertex 19!! [INFO] [1689784870.669278248, 5.500000000]: COLLISION with vertex 18!! [INFO] [1689784870.671523088, 5.500000000]: COLLISION with vertex 100!! [INFO] [1689784870.673023179, 5.500000000]: COLLISION with vertex 101!! [INFO] [1689784870.674498145, 5.500000000]: Please chose a better location for vertex 101</pre>
1.5x	<pre>[INFO] [1689782316.318915725, 8.800000000]: COLLISION with vertex 9!! [INFO] [1689782316.320344677, 8.800000000]: COLLISION with vertex 8!! [INFO] [1689782316.325187974, 8.800000000]: COLLISION with vertex 21!! [INFO] [1689782316.327980354, 8.800000000]: COLLISION with vertex 20!! [INFO] [1689782316.329050744, 8.800000000]: COLLISION with vertex 26!! [INFO] [1689782316.330898923, 8.800000000]: COLLISION with vertex 4!! [INFO] [1689782316.333204032, 8.800000000]: COLLISION with vertex 19!! [INFO] [1689782316.335141131, 8.800000000]: COLLISION with vertex 18!! [INFO] [1689782316.336927692, 8.800000000]: COLLISION with vertex 100!! [INFO] [1689782316.33812804, 8.800000000]: COLLISION with vertex 101!! [INFO] [1689782316.339341138, 8.800000000]: Please chose a better location for vertex 101</pre>	<pre>[INFO] [1689783496.687262672, 6.200000000]: COLLISION with vertex 9!! [INFO] [1689783496.689292444, 6.200000000]: COLLISION with vertex 8!! [INFO] [1689783496.691002675, 6.200000000]: COLLISION with vertex 7!! [INFO] [1689783496.693656955, 6.200000000]: COLLISION with vertex 21!! [INFO] [1689783496.695405798, 6.200000000]: COLLISION with vertex 20!! [INFO] [1689783496.696506936, 6.200000000]: COLLISION with vertex 26!! [INFO] [1689783496.698335077, 6.200000000]: COLLISION with vertex 4!! [INFO] [1689783496.700644441, 6.200000000]: COLLISION with vertex 19!! [INFO] [1689783496.702438131, 6.200000000]: COLLISION with vertex 18!! [INFO] [1689783496.703964605, 6.200000000]: COLLISION with vertex 100!! [INFO] [1689783496.705780845, 6.200000000]: COLLISION with vertex 101!! [INFO] [1689783496.707260670, 6.200000000]: Please chose a better location for vertex 101</pre>	<pre>[INFO] [1689785282.192108696, 8.600000000]: COLLISION with vertex 9!! [INFO] [1689785282.193392850, 8.600000000]: COLLISION with vertex 8!! [INFO] [1689785282.197887807, 8.600000000]: COLLISION with vertex 21!! [INFO] [1689785282.200346199, 8.600000000]: COLLISION with vertex 20!! [INFO] [1689785282.202338742, 8.600000000]: COLLISION with vertex 26!! [INFO] [1689785282.205901988, 8.600000000]: COLLISION with vertex 4!! [INFO] [1689785282.209046896, 8.600000000]: COLLISION with vertex 19!! [INFO] [1689785282.211539005, 8.600000000]: COLLISION with vertex 18!! [INFO] [1689785282.213281080, 8.600000000]: COLLISION with vertex 100!! [INFO] [1689785282.214557965, 8.600000000]: COLLISION with vertex 101!! [INFO] [1689785282.215911437, 8.600000000]: Please chose a better location for vertex 101</pre>
2x	<pre>[INFO] [1689782709.730177652, 6.000000000]: COLLISION with vertex 9!! [INFO] [1689782709.731463252, 6.000000000]: COLLISION with vertex 8!! [INFO] [1689782709.736073198, 6.000000000]: COLLISION with vertex 21!! [INFO] [1689782709.739767262, 6.000000000]: COLLISION with vertex 20!! [INFO] [1689782709.740934962, 6.000000000]: COLLISION with vertex 26!! [INFO] [1689782709.742511265, 6.000000000]: COLLISION with vertex 4!! [INFO] [1689782709.744924116, 6.000000000]: COLLISION with vertex 19!! [INFO] [1689782709.746590933, 6.000000000]: COLLISION with vertex 18!! [INFO] [1689782709.749126813, 6.000000000]: COLLISION with vertex 100!! [INFO] [1689782709.750293238, 6.000000000]: COLLISION with vertex 101!! [INFO] [1689782709.751510586, 6.000000000]: Please chose a better location for vertex 101</pre>	<pre>[INFO] [1689783852.030794380, 5.100000000]: COLLISION with vertex 9!! [INFO] [1689783852.032106212, 5.100000000]: COLLISION with vertex 8!! [INFO] [1689783852.034507193, 5.100000000]: COLLISION with vertex 7!! [INFO] [1689783852.037357481, 5.100000000]: COLLISION with vertex 21!! [INFO] [1689783852.039077143, 5.100000000]: COLLISION with vertex 20!! [INFO] [1689783852.040178142, 5.100000000]: COLLISION with vertex 26!! [INFO] [1689783852.041836318, 5.100000000]: COLLISION with vertex 4!! [INFO] [1689783852.044037368, 5.100000000]: COLLISION with vertex 19!! [INFO] [1689783852.045715571, 5.100000000]: COLLISION with vertex 18!! [INFO] [1689783852.047873920, 5.100000000]: COLLISION with vertex 100!! [INFO] [1689783852.048078029, 5.100000000]: COLLISION with vertex 101!! [INFO] [1689783852.049177217, 5.100000000]: Please chose a better location for vertex 101</pre>	<pre>[INFO] [1689785624.244563960, 5.300000000]: COLLISION with vertex 9!! [INFO] [1689785624.246201317, 5.300000000]: COLLISION with vertex 8!! [INFO] [1689785624.251011951, 5.300000000]: COLLISION with vertex 21!! [INFO] [1689785624.253760737, 5.300000000]: COLLISION with vertex 20!! [INFO] [1689785624.254936865, 5.300000000]: COLLISION with vertex 26!! [INFO] [1689785624.256954725, 5.300000000]: COLLISION with vertex 4!! [INFO] [1689785624.259580368, 5.300000000]: COLLISION with vertex 19!! [INFO] [1689785624.261420223, 5.300000000]: COLLISION with vertex 18!! [INFO] [1689785624.262960366, 5.300000000]: COLLISION with vertex 100!! [INFO] [1689785624.264011820, 5.300000000]: COLLISION with vertex 101!! [INFO] [1689785624.265310537, 5.300000000]: Please chose a better location for vertex 101</pre>

Table 6.8: Different sizes: Console Output

6.1.2.3 Analysis

As mentioned in the previous test, let's consider vertex 26 as an example. The robot moved further away from the top wall and less from the left wall. This aligns with the expectations traced, as in this vertex, the robot is represented by a rectangle oriented with $theta = -\frac{\pi}{2}$ and $theta_holonomic = -\frac{\pi}{2}$. Hence, the node is effectively performing its intended task. This pattern applies to all vertices; their orientation determines how much they move away from certain walls.

Another crucial aspect to emphasise is the behaviour of the tricycle robot. Due to its base footprint being closer to the back of the robot, when the vertex is oriented in the opposite direction of the wall, it is expected that the vertex will not move as much from the wall compared to the other types of robots. Conversely, if the vertex is facing the wall, the tricycle robot should step back more than the others. Again, using vertex 26 as an example in the rectangle section, the tricycle robot indeed moves less from the wall than the others, confirming our expectations (Table 6.5).

The console outputs provide valuable information. The table displays which vertices have collided with the wall. If there is no feasible position for a vertex, a message advises selecting a better location. Comparing the results, the circle case collided with vertices where the robot didn't actually collide. In the bigger-sized robot test, the circle case collided with nearly all vertices, while the rectangle case did not, demonstrating its proper functioning.

Upon closer examination of the console outputs, we notice that all collision vertices were among the ten tested (four corners, four walls, and two additional vertices). However, there was a difference for the differential robot, where vertex 7 collided with the wall for 1.5x and 2x the original robot size, but not for the omnidirectional and tricycle robots. This is because in the *trajectory_control_data.yaml*, vertex 7 has $theta = \frac{\pi}{2}$ and $theta_holonomic = 0$. This explains the obtained results: for the omnidirectional robot, the orientation of the vertex matches the $theta_holonomic$, which is zero, so it is not colliding. Similarly, for the tricycle robot, since the base footprint is at the back, and the orientation is $\frac{\pi}{2}$, it also avoids collision. Overall, the node appears to be functioning as intended.

6.1.3 Test 3: Corner and closed space with different orientations and sizes

6.1.3.1 Setup

After analysing the previous tests, it became evident that no robot would fit inside the small box represented by vertex 101, as the algorithm always failed to place the vertex in an appropriate position. However, upon a closer examination of the box's configuration, it was noticed that since the box is wider than it is tall, a robot could potentially fit in that place if the orientation of the vertex was changed. Therefore, the parameters corresponding to the $theta$ and $theta_holonomic$ of

that vertex were modified to 0 and $-\frac{\pi}{2}$, respectively.

Similarly, concerning vertex 26, various adjustments were made to test different scenarios. The original setups had $\theta = -\frac{\pi}{2}$ and $\theta_{holonomic} = -\frac{\pi}{2}$. The parameters were changed to $\theta = \frac{\pi}{2}$ and $\theta_{holonomic} = \frac{\pi}{2}$. Figures 6.3 and 6.4 represent the new configurations of these vertices.

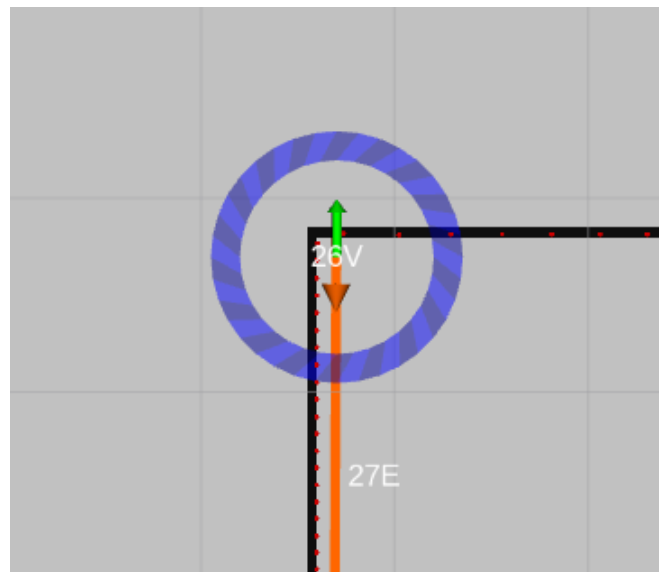


Figure 6.3: New configuration for Vertex 26

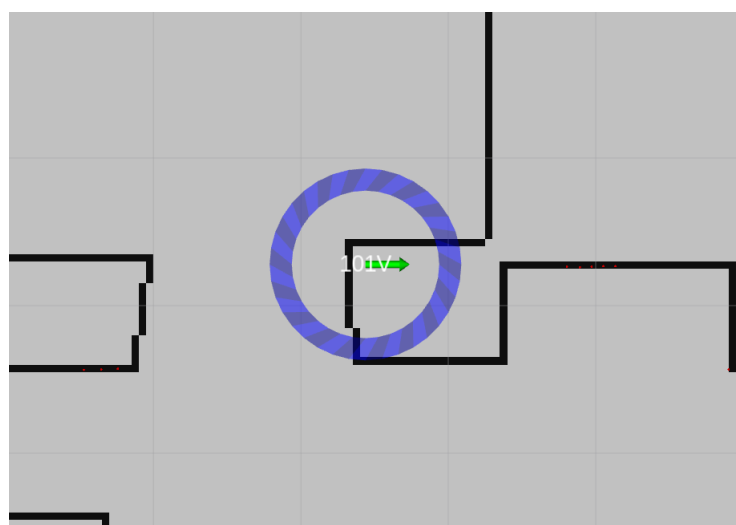


Figure 6.4: New configuration for Vertex 101

6.1.3.2 Results

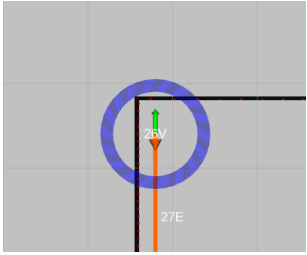
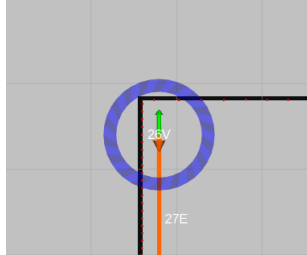
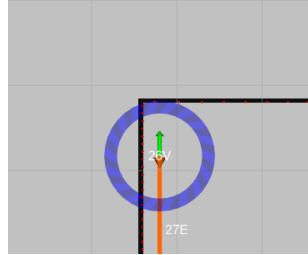
Vertex 26			
Size	Omnidirectional	Differential	Tricycle
1x			

Table 6.9: Different orientations for vertex 26

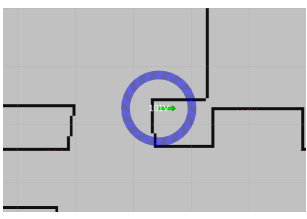

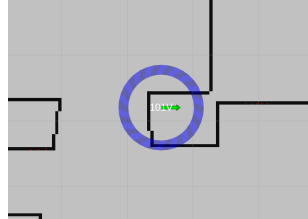
Vertex 101			
Size	Omnidirectional	Differential	Tricycle
1x			

Table 6.10: Different orientations for vertex 101

Vertex 26			
Size	Omnidirectional	Differential	Tricycle
1x	<pre>[INFO] [1689862556.571790509, 373.000000000]: COLLISION with vertex 9!! [INFO] [1689862556.573485247, 373.000000000]: COLLISION with vertex 8!! [INFO] [1689862556.578412222, 373.000000000]: COLLISION with vertex 21!! [INFO] [1689862556.580337456, 373.000000000]: COLLISION with vertex 20!! [INFO] [1689862556.581603341, 373.000000000]: COLLISION with vertex 26!! [INFO] [1689862556.583982853, 373.000000000]: COLLISION with vertex 4!! [INFO] [1689862556.586381009, 373.000000000]: COLLISION with vertex 19!! [INFO] [1689862556.588702397, 373.000000000]: COLLISION with vertex 18!! [INFO] [1689862556.590922949, 373.000000000]: COLLISION with vertex 100!! [INFO] [1689862556.592313697, 373.000000000]: COLLISION with vertex 101!! [INFO] [1689862556.593738185, 373.000000000]: Please chose a better location for vertex 101.</pre>	<pre>[INFO] [1689862861.067161176, 7.700000000]: COLLISION with vertex 9!! [INFO] [1689862861.068605942, 7.700000000]: COLLISION with vertex 8!! [INFO] [1689862861.073495401, 7.700000000]: COLLISION with vertex 21!! [INFO] [1689862861.075632198, 7.700000000]: COLLISION with vertex 20!! [INFO] [1689862861.077076149, 7.700000000]: COLLISION with vertex 26!! [INFO] [1689862861.079052784, 7.700000000]: COLLISION with vertex 4!! [INFO] [1689862861.081739854, 7.700000000]: COLLISION with vertex 19!! [INFO] [1689862861.084463909, 7.700000000]: COLLISION with vertex 18!! [INFO] [1689862861.086533164, 7.700000000]: COLLISION with vertex 100!! [INFO] [1689862861.088591775, 7.700000000]: COLLISION with vertex 101!!</pre>	<pre>[INFO] [1689863102.383003358, 6.100000000]: COLLISION with vertex 9!! [INFO] [1689863102.384728760, 6.100000000]: COLLISION with vertex 8!! [INFO] [1689863102.390371849, 6.100000000]: COLLISION with vertex 21!! [INFO] [1689863102.392773325, 6.100000000]: COLLISION with vertex 20!! [INFO] [1689863102.394870176, 6.100000000]: COLLISION with vertex 26!! [INFO] [1689863102.397518044, 6.100000000]: COLLISION with vertex 4!! [INFO] [1689863102.401666488, 6.100000000]: COLLISION with vertex 19!! [INFO] [1689863102.403892277, 6.100000000]: COLLISION with vertex 18!! [INFO] [1689863102.405744652, 6.100000000]: COLLISION with vertex 100!! [INFO] [1689863102.407349626, 6.100000000]: COLLISION with vertex 101!!</pre>

Table 6.11: Vertices different orientations - Console Output

6.1.3.3 Analysis

In the first case, for vertex 26, the results were as expected: for the omnidirectional and differential robots, with $theta = \frac{\pi}{2}$ and $theta_{holonomic} = \frac{\pi}{2}$, the outcome remained the same as in the previous testing. However, for the tricycle robot, with this orientation, the vertex had to move further away from the wall.

As for the second case, vertex 101, it was observed that with a different orientation ($theta = 0$ and $theta_{holonomic} = -\frac{\pi}{2}$) than the previously tested configuration ($theta = -\frac{\pi}{2}$ and $theta_{holonomic} = -\frac{\pi}{2}$), the vertex seemed to fit for both the tricycle and differential robots. In these cases, the tricycle robot (as expected) moved away from the wall less than the differential robot. However, since the $theta_{holonomic}$ remained the same as in the previous test, the vertex still collided with the wall and had no suitable position with that orientation. Nevertheless, for both the differential and tricycle robots (which use $theta$), the vertex fit into the narrow space without any collision, as there were no messages outputted in the console, and the changes in the vertex position were visible.

6.2 Obstacle avoidance

6.2.1 Setup

Throughout the testing process, additional vertices and edges used for the alternative path were created in the *trajectory_control_data.yaml* file. However, at the end of the alternative path, they were deleted from the file. To better observe their position, a test was performed where the application was stopped during the alternative path. When the application was restarted, the alternative vertices and edges were still visible in RViz, allowing for a closer examination of their positions, as shown in Figure 6.5.

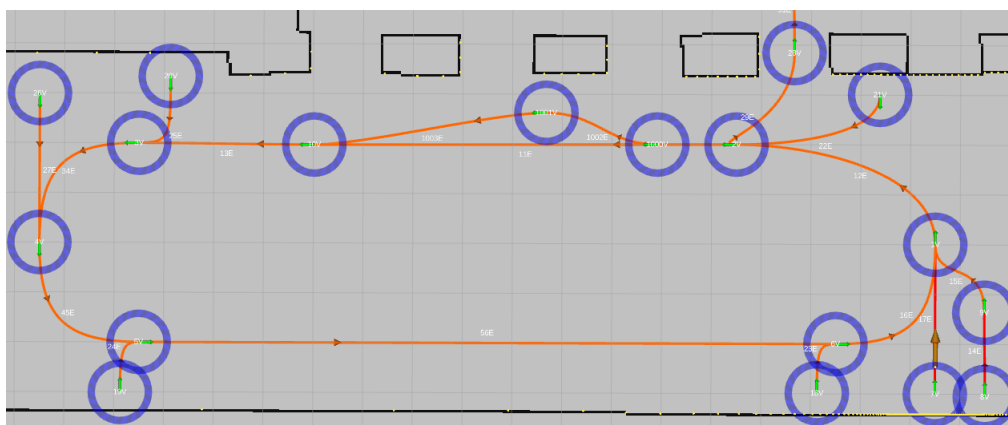


Figure 6.5: Alternative Path

During the testing phase of the obstacle avoidance algorithm, various scenarios were examined as follows:

1. With the robot going forward:

- (a) Removing the obstacle in less than 5 seconds
- (b) Encountering a vertical wall in a horizontal path with enough space to pass from the top
- (c) Encountering a vertical wall in a horizontal path without enough space to pass from the top, necessitating passage from the bottom side
- (d) Encountering an oblique wall in a horizontal path with enough space to pass from the top
- (e) Encountering an oblique wall in a horizontal path without enough space to pass from the top, requiring passage from the bottom side
- (f) Encountering an oblique wall in a curved path

2. With the robot moving backwards:

- (a) Encountering a vertical wall in a horizontal path with enough space to pass from the top
- (b) Encountering a vertical wall in a horizontal path without enough space to pass from the top, necessitating passage from the bottom side
- (c) Encountering an oblique wall in a horizontal path with enough space to pass from the top
- (d) Encountering an oblique wall in a horizontal path without enough space to pass from the top, requiring passage from the bottom side

6.2.2 Results

Throughout this subsection, the previously mentioned specific outcomes of each test scenario will be addressed, providing a comprehensive evaluation of the obstacle avoidance algorithm's performance under diverse conditions.

1. Robot moving forward		
Tests Performed	Differential	Tricycle
(a)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles). The robot's current position and heading are shown in a small inset window.	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles). The robot's current position and heading are shown in a small inset window.
(b)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).
(c)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).
(d)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).
(e)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).
(f)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).

Table 6.12: Different tested cases - Robot moving forward

2. Robot moving backwards		
Tests Performed	Differential	Tricycle
(a)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).
(b)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).
(c)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).
(d)	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).	A screenshot of a robot simulation showing a path (orange line) navigating through a maze of obstacles (blue circles).

Table 6.13: Different tested cases - Robot moving backwards

6.2.3 Analysis

Regarding the conducted tests, the overall output was satisfactory. However, there were certain factors that influenced the results and should be taken into consideration for further improvements:

- **Lower distance between clusters for merging:** In some cases, when the obstacle and the wall were too close to each other, the algorithm treated them as a single cluster, leading to sub-optimal results. To address this, the parameter determining the distance for merging clusters could be adjusted to a lower value. This would help in accurately distinguishing between separate obstacles and walls, resulting in more precise path planning.
- **Adjustment of control points of the Bèzier parametric curve considering the environment:** While the creation of vertices for the alternative path effectively avoided collisions, the temporary edges connecting these vertices were determined using a generic algorithm. To achieve better results in specific environmental contexts and for different robot types, a more adaptive algorithm for determining the control points of the Bèzier curve could be implemented. This way, the path would be optimised according to the environment and the robot's characteristics.
- **Consideration of a set of lasers to perform obstacle avoidance:** Currently, the algorithm allows for obstacle avoidance one obstacle at a time. In more complex scenarios with multiple obstacles, this approach may lead to inefficient path planning. To address this, a new algorithm could be developed to incorporate a set of lasers, including frontal and lateral ones. By adapting the distance detection parameters during the alternative path, the robot can better handle situations with multiple obstacles and perform obstacle avoidance more effectively.
- **Addressing proximity challenges:** A potential area of improvement in the obstacle avoidance algorithm lies in addressing challenges when obstacles are in close proximity to the vertices of the robot's path. In certain scenarios, an obstacle positioned just before a vertex could result in an excessively sharp curve, increasing the risk of collision. Additionally, when an obstacle is detected right after a vertex while the robot is still on the previous edge, it could lead to the next vertex being the one directly in front of the obstacle. To enhance the robot's adaptability in such situations, an alternative approach could be considered. Rather than proceeding directly to the next vertex after the current edge, if available, the robot could be redirected to the second next vertex in the path set. This adaptive strategy aims to provide more flexibility in navigation and mitigate challenges posed by obstacles near the vertices. In cases where no next vertex is available, further exploration of alternative strategies, such as implementing wider turns or advanced obstacle avoidance techniques, may be considered to ensure safe and efficient robot navigation. Future work in this area will involve refining and optimising these approaches through extensive simulations and real-world experimentation.

- **Optimising vertex placement:** In the current algorithm, the vertex is positioned along the line connecting the point where the front beam detects the obstacle and the extremity point of the cluster. However, in the case of an oblique wall with a low angle, this placement might lead to potential collisions with the obstacle from the side. An idea for enhancing the algorithm in the future is to position the vertex perpendicular to the trajectory, passing through the extremity point of the cluster. This adjustment would likely improve the robot's ability to avoid obstacles effectively.
- **Enhancing the robot's area inflation:** Currently, the robot's area is being inflated uniformly in all directions to account for its size and shape. However, in certain scenarios with complex obstacles, this approach may not be sufficient to ensure complete collision avoidance. To address this, a potential enhancement would be to inflate the robot's area selectively, focusing on areas close to obstacles or in the direction of motion. This targeted inflation strategy could further improve the robot's collision avoidance capabilities and allow it to navigate more effectively through intricate environments.

Addressing these factors would further enhance the obstacle avoidance algorithm's performance, making it more robust and adaptable to various environments and robot types. These improvements could be considered for future work to enhance the overall efficiency and reliability of the system.

Chapter 7

Conclusion and Future Work

In conclusion, this master's thesis successfully addressed the challenges of mobile robotics, specifically focusing on two critical aspects: docking and obstacle avoidance. The developed offline mode software algorithm proved effective in readjusting the position of docking vertices according to the robot's dimensions and base footprint. By optimising the placement of vertices, the algorithm demonstrated seamless integration into complex environments and spaces with the presence of walls, obstacles and machinery mapped as the environment. Moreover, the online mode algorithm for obstacle avoidance enabled the robot to dynamically find alternative paths and avoid collisions during its trajectory.

Although the simulation environment provided for testing only included vertical and horizontal walls, it is expected that the first developed algorithm would produce positive results with different types of walls, including oblique walls or walls with intricate shapes. The algorithm's adaptability and robustness are projected to facilitate the successful repositioning of vertices in various complex environments, encompassing a wide range of wall configurations and orientations.

However, a critical assumption in this approach was that vertices could not be placed outside the map. In the simulation environment used for testing, only the walls were considered as occupied cells in the grid mapping, while the outside of the map was not defined as occupied. As a consequence, if a vertex was mistakenly placed outside the map, the algorithm would not recognise it as such, since the type of cell would be the same as inside the map. As a result, the algorithm would move the vertex further away from the wall, inadvertently leading the robot to move out of the room "into the wall."

Ideally, the algorithm should have been designed to detect when a vertex was placed outside the room and adjust its position accordingly, optimising the distance from the walls. However, achieving this functionality would have required the provided simulation to include an additional algorithm to define the outside of the map as occupied. This adjustment would enable the main algorithm to be reconfigured appropriately, ensuring the robot's movements are optimised and confined within the room's boundaries.

The experimental results and analysis showcased the algorithm's performance in various scenarios, validating its efficiency and practical applicability. The research outcomes highlight the

significance of considering the robot's dimensions and specific type to achieve optimal results in docking tasks.

Although this research has made significant progress in mobile robotics algorithms, there remain several potential areas for future work and enhancements, particularly concerning the obstacle avoidance algorithm.

Firstly, considering a set of laser beams for obstacle avoidance instead of only the front and back beams (coordinated front and lateral laser beams, for example) could lead to more dynamic and sophisticated navigation, allowing the robot to respond promptly to new unexpected obstacles in its path. This approach could facilitate the exploration of recursive obstacle avoidance, enabling the development of strategies to manage multiple obstacles simultaneously, thus enhancing the robot's adaptability in intricate environments.

Another encountered limitation was when the obstacle was in close proximity to the vertices. In such cases, if the obstacle was positioned just before the vertex, the resulting curve became excessively sharp, significantly increasing the risk of colliding with the obstacle. While the differential robot performed better in navigating such sharp curves, the tricycle robot required a certain angle to turn, which could potentially introduce errors in the controller's performance. On the other hand, if the obstacle was too close but right after the vertex, the detection of the obstacle could happen while the robot was still on the previous edge, leading to the next vertex being the one directly in front of the obstacle.

To address both of these issues, an alternative approach could be considered. Instead of proceeding directly to the next vertex after the current edge, if available, the robot could be redirected to the second next vertex in the path set. This adaptive strategy aims to improve navigation flexibility and address challenges related to obstacles in close proximity to the vertices. However, in scenarios where no next vertex is available, various alternative strategies can be explored, such as implementing wider turns or employing advanced obstacle avoidance techniques, to ensure the robot's navigation remains both safe and efficient.

Currently, vertices are placed along the line formed by the point corresponding to the front beam detecting the obstacle and the point corresponding to the cluster's extremity. However, in the case of an oblique wall with a low angle, this placement might lead to potential lateral collisions with the obstacle. As a solution, a possible approach for future development involves situating the vertices along a perpendicular line to the trajectory, passing through the cluster's extremity point. This adjustment could strengthen the algorithm's ability to handle diverse obstacle configurations.

Another noteworthy aspect centres on the inflation of the robot's area for collision avoidance. At present, the robot's area is uniformly inflated in all directions to accommodate its size and shape. Although effective in many scenarios, this approach may fall short in dealing with complex obstacles. To address this limitation, a potential enhancement involves selective inflation of the robot's area, with a greater emphasis on inflating areas close to obstacles or in the direction of motion. This targeted inflation strategy holds promise in further fortifying the robot's collision avoidance capabilities, enabling it to navigate with heightened precision in intricate environments.

While the creation of alternative path vertices effectively addressed collision avoidance in simple configurations, the determination of temporary edge connections relied on the generic algorithm provided by the source code. To further enhance the algorithm's performance in different environmental scenarios and with diverse robot types, a more adaptive approach for calculating the control points of the Bèzier curve can be employed. By customising the path optimisation based on the specific environment, the obstacle encountered and robot characteristics, this adaptive algorithm promises to unlock the algorithm's full potential, resulting in more efficient and accurate obstacle avoidance capabilities.

This master's thesis has been divided into two major subjects: the first node, which deals with repositioning vertices representing critical points or docking stations, will be implemented in ongoing projects in *CRISS*. Meanwhile, the second subject, though more of a proof of concept with certain limitations, lays the groundwork for future work. The proposed algorithms make valuable contributions to the advancement of mobile robotics, opening doors for enhanced navigation and efficiency across a wide range of real-world scenarios. As the field of mobile robotics continues to progress, this research serves as a stepping stone for forthcoming innovations and progressions in this thrilling and rapidly evolving domain.

References

- [1] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.
- [2] Sean Campbell, Niall O’Mahony, Anderson Carvalho, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Path planning techniques for mobile robots a review. In *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)*, pages 12–16. IEEE, 2020.
- [3] Guangming Song, Hui Wang, Jun Zhang, and Tianhua Meng. Automatic docking system for recharging home surveillance robots. *IEEE Transactions on Consumer Electronics*, 57(2):428–435, 2011.
- [4] Roberto Quilez, Adriaan Zeeman, Nathalie Mitton, and Julien Vandaele. Docking autonomous robots in passive docks with infrared sensors and qr codes. In *International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCOM)*, 2015.
- [5] Mamadou Doumbia, Xu Cheng, and Vincent Havvarimana. An auto-recharging system design and implementation based on infrared signal for autonomous robots. In *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, pages 894–900. IEEE, 2019.
- [6] MV Sreenivas Rao and M Shivakumar. Ir based auto-recharging system for autonomous mobile robot. *Journal of Robotics and Control (JRC)*, 2(4):244–251, 2021.
- [7] Michail Kalaitzakis, Sabrina Carroll, Anand Ambrosi, Camden Whitehead, and Nikolaos Vitzilaios. Experimental comparison of fiducial markers for pose estimation. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 781–789. IEEE, 2020.
- [8] Michail Kalaitzakis, Brennan Cain, Sabrina Carroll, Anand Ambrosi, Camden Whitehead, and Nikolaos Vitzilaios. Fiducial markers for pose estimation. *Journal of Intelligent & Robotic Systems*, 101(4):1–26, 2021.
- [9] Fan Guangrui and Wang Geng. Vision-based autonomous docking and re-charging system for mobile robot in warehouse environment. In *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*, pages 79–83. IEEE, 2017.
- [10] Alexey M Romanov and Andrey A Tararin. An automatic docking system for wheeled mobile robots. In *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pages 1040–1045. IEEE, 2021.

- [11] Xiaofan Zhang, Xiuzhi Li, and Xiangyin Zhang. Automatic docking and charging of mobile robot based on laser measurement. In *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 5, pages 2229–2234. IEEE, 2021.
- [12] Yugang Liu. A laser intensity based autonomous docking approach for mobile robot recharging in unstructured environments. *IEEE Access*, 10:71165–71176, 2022.
- [13] Seyyed Mohammad Hosseini Rostami, Arun Kumar Sangaiah, Jin Wang, and Xiaozhu Liu. Obstacle avoidance of mobile robots using modified artificial potential field algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–19, 2019.
- [14] Huaidong Zhou, Pengbo Feng, and Wusheng Chou. A hybrid obstacle avoidance method for mobile robot navigation in unstructured environment. *Industrial Robot: the international journal of robotics research and application*, 2022.
- [15] Dhong Hun Lee, Sang Su Lee, Choon Ki Ahn, Peng Shi, and Cheng-Chew Lim. Finite distribution estimation-based dynamic window approach to reliable obstacle avoidance of mobile robot. *IEEE Transactions on Industrial Electronics*, 68(10):9998–10006, 2020.
- [16] Qing Rebecca Lia, Zachary Dydek, and Daniel Theobald. Why interoperability is critical to the warehouse of the future. In *ISR Europe 2022; 54th International Symposium on Robotics*, pages 1–7. VDE, 2022.
- [17] KDW Gunawardhana, DGDP Kularathana, WH Welagedara, HE Palihakkara, Pradeep KW Abeygunawardhana, and Sasini Wellalage. Indoor autonomous multi-robot communication system. In *2021 3rd International Conference on Advancements in Computing (ICAC)*, pages 199–204. IEEE, 2021.
- [18] Thomas Bräunl. *Embedded robotics*. Springer, 2003.
- [19] António Paulo Gomes Mendes Moreira. Mobile robotics - locomotion and traction. PDF document provided by Teacher in Course "Autonomous Systems", 2022.
- [20] Hamid Taheri and Chun Xia Zhao. Omnidirectional mobile robots, mechanisms and navigation approaches. *Mechanism and Machine Theory*, 153:103958, 2020.
- [21] Thejus Pathmakumar, Vinu Sivanantham, Saurav Ghante Anantha Padmanabha, Mohan Rajesh Elara, and Thein Than Tun. Towards an optimal footprint based area coverage strategy for a false-ceiling inspection robot. *Sensors*, 21(15):5168, 2021.
- [22] Thejus Pathmakumar, Madan Mohan Rayguru, Sriharsha Ghanta, Manivannan Kalimuthu, and Mohan Rajesh Elara. An optimal footprint based coverage planning for hydro blasting robots. *Sensors*, 21(4):1194, 2021.
- [23] Kossar Jeddisaravi, Reza Javanmard Alitappeh, Luciano C A. Pimenta, and Frederico G Guimarães. Multi-objective approach for robot motion planning in search tasks. *Applied Intelligence*, 45:305–321, 2016.
- [24] Alexander Wilhelm. Design of a mobile robotic platform with variable footprint. Master's thesis, University of Waterloo, 2007.
- [25] Purushothaman Raja and Sivagurunathan Pugazhenth. Optimal path planning of mobile robots: A review. *International journal of physical sciences*, 7(9):1314–1320, 2012.

- [26] BK Patle, Anish Pandey, DRK Parhi, AJDT Jagadeesh, et al. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606, 2019.
- [27] Mustafa S Abed, Omar F Lutfy, and Qusay F Al-Doori. A review on path planning algorithms for mobile robots. *Engineering and Technology Journal*, 39(5):804–820, 2021.
- [28] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28, 2016.
- [29] Han-ye Zhang, Wei-ming Lin, and Ai-xia Chen. Path planning for the mobile robot: A review. *Symmetry*, 10(10):450, 2018.
- [30] Yanwei Zhao, Yinong Zhang, and Shuying Wang. A review of mobile robot path planning based on deep reinforcement learning algorithm. In *Journal of Physics: Conference Series*, volume 2138, page 012011. IOP Publishing, 2021.
- [31] Luís C Santos, Filipe N Santos, António Valente, Héber Sobreira, José Sarmento, and Marcelo Petry. Collision avoidance considering iterative bézier based approach for steep slope terrains. *Ieee Access*, 10:25005–25015, 2022.
- [32] Mohd Nayab Zafar and JC Mohanta. Methodology for path planning and optimization of mobile robots: A review. *Procedia computer science*, 133:141–152, 2018.
- [33] P Mira Vaz, R Ferreira, V Grossmann, and MI Ribeiro. Docking of a mobile platform based on infrared sensors. In *ISIE'97 Proceeding of the IEEE International Symposium on Industrial Electronics*, volume 2, pages 735–740. IEEE, 1997.
- [34] Uri Kartoun, Helman Stern, Yael Edan, Craig Feied, Jonathan Handler, Mark Smith, and Michael Gillam. Vision-based autonomous robot self-docking and recharging. In *2006 World Automation Congress*, pages 1–8. IEEE, 2006.
- [35] Yuan Hu, Akash Vibhute, Shaohui Foong, and Gim Song Soh. Autonomous docking of miniature spherical robots with an external 2d laser rangefinder. In *2016 IEEE Region 10 Conference (TENCON)*, pages 3525–3529. IEEE, 2016.
- [36] Wei Wang, Zongliang Li, Wenpeng Yu, and Jianwei Zhang. An autonomous docking method based on ultrasonic sensors for self-reconfigurable mobile robot. In *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1744–1749. IEEE, 2009.
- [37] Y-S TAN, L-M REN, and H-B ZHANG. On laser diode and wireless communication based interactive docking for home service robots. *Lasers in Engineering (Old City Publishing)*, 33, 2016.
- [38] Kuo-Lan Su, Yi-Lin Liao, Shih-Ping Lin, and Sian-Fu Lin. An interactive auto-recharging system for mobile robots. *International Journal of Automation and Smart Technology*, 4(1):43–53, 2014.
- [39] Mamadou Doumbia, Xu Cheng, and Haixian Chen. A novel power management system for autonomous robots. In *Proceedings of the Future Technologies Conference*, pages 293–305. Springer, 2019.
- [40] Gaetano C La Delfa, Salvatore Monteleone, Vincenzo Catania, Juan F De Paz, and Javier Bajo. Performance analysis of visualmarkers for indoor navigation systems. *Frontiers of Information Technology & Electronic Engineering*, 17(8):730–740, 2016.

- [41] Akhmad Thalibar Rifqi, Bima Sena Bayu Dewantara, Dadet Pramadihanto, and Bayu Sandi Marta. Fuzzy social force model for healthcare robot navigation and obstacle avoidance. In *2021 International Electronics Symposium (IES)*, pages 445–450. IEEE, 2021.
- [42] Gürkan Şahin, Muhammet Balcılar, Erkan Uslu, Sırma Yavuz, and M Fatih Amasyalı. Obstacle avoidance with vector field histogram algorithm for search and rescue robots. In *2014 22nd Signal Processing and Communications Applications Conference (SIU)*, pages 766–769. IEEE, 2014.
- [43] Mao Lin, Ji Xiaoming, and Qin Fei. A robot obstacle avoidance method based on improved genetic algorithm. In *2018 11th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pages 327–331. IEEE, 2018.
- [44] Zhao Huadong, Lei Chaofan, and Jiang Nan. A path planning method of robot arm obstacle avoidance based on dynamic recursive ant colony algorithm. In *2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, pages 549–552. IEEE, 2019.
- [45] Yuxin Zhao and Wei Zu. Real-time obstacle avoidance method for mobile robots based on a modified particle swarm optimization. In *2009 International Joint Conference on Computational Sciences and Optimization*, volume 2, pages 269–272. IEEE, 2009.
- [46] Elisabete Fernandes, Pedro Costa, José Lima, and Germano Veiga. Towards an orientation enhanced astar algorithm for robotic navigation. In *2015 IEEE International Conference on Industrial Technology (ICIT)*, pages 3320–3325. IEEE, 2015.
- [47] Mohammed Ayoub Juman, Yee Wan Wong, Rajprasad Kumar Rajkumar, and Cong Yuan H’ng. An integrated path planning system for a robot designed for oil palm plantations. In *TENCON 2017-2017 IEEE Region 10 Conference*, pages 1048–1053. IEEE, 2017.

Appendix A

Results

A.1 Tables

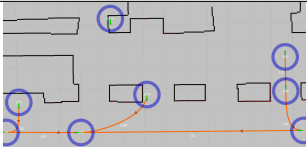
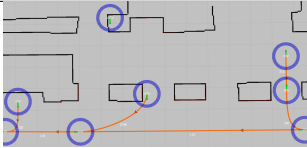
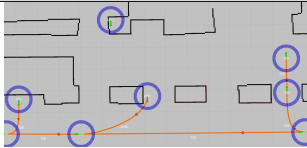
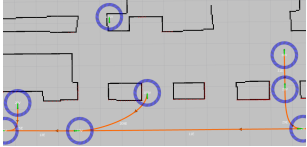
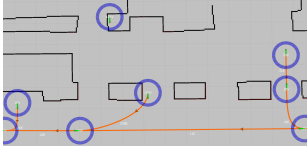
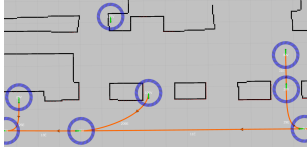
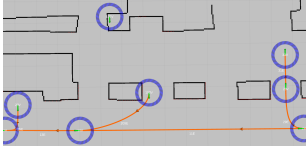
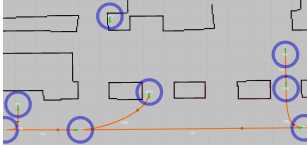
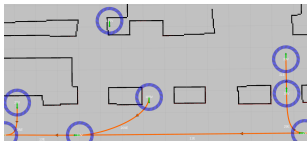
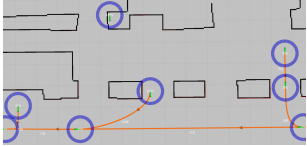
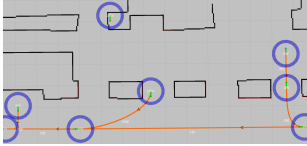
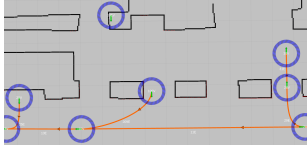
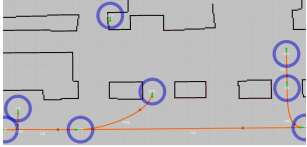
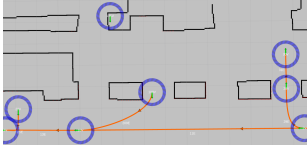
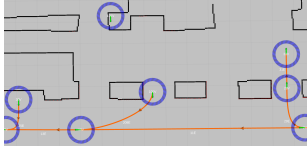
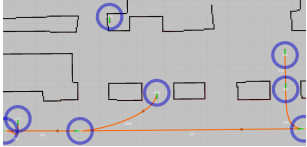
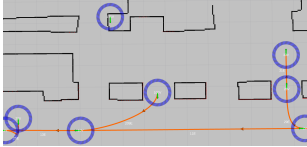
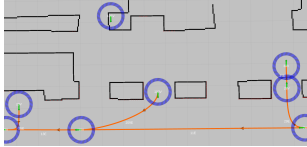
Middle section			
Size	Omnidirectional	Differential	Tricycle
1x			
1.25x			
1.5x			
1.75x			
2x			
3x			

Table A.1: Different sizes: Middle section (extended version)

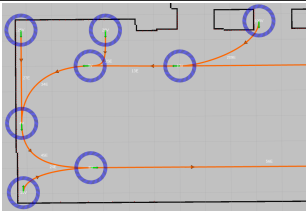
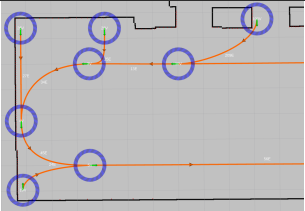
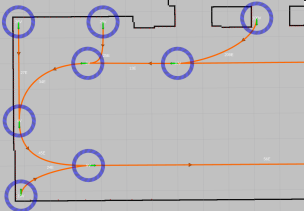
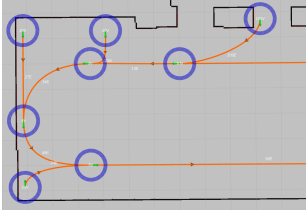
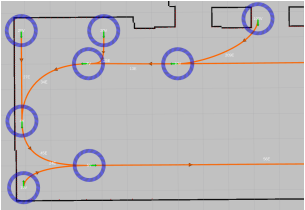
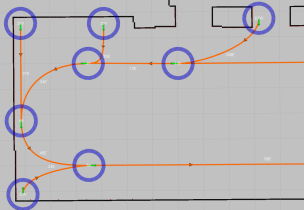
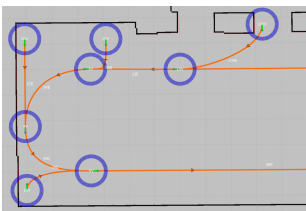
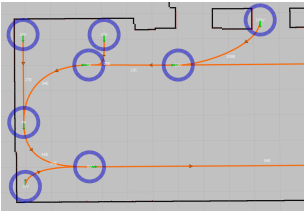
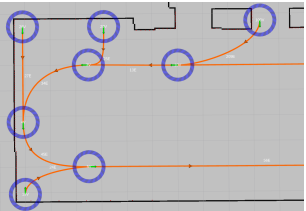
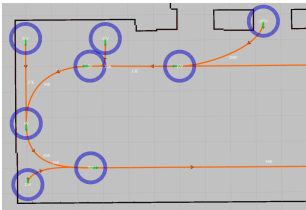
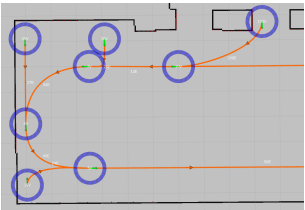
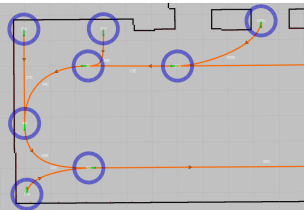
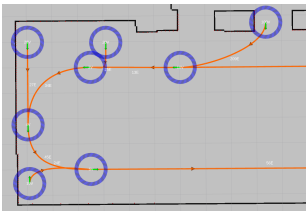
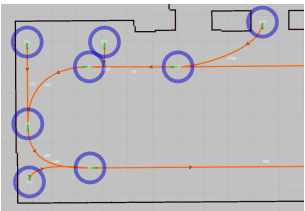
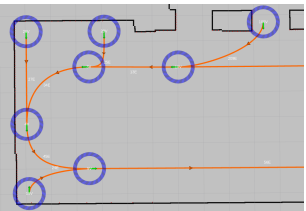
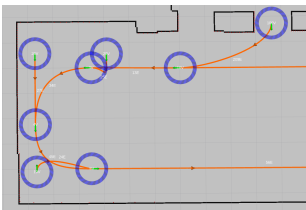
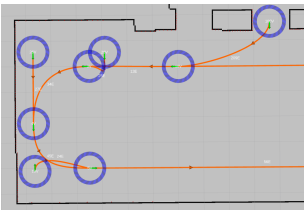
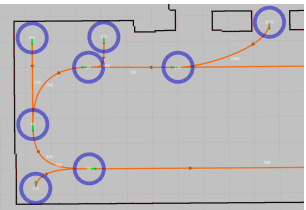
Left side			
Size	Omnidirectional	Differential	Tricycle
1x			
1.25x			
1.5x			
1.75x			
2x			
3x			

Table A.2: Different sizes: Left side (extended version)

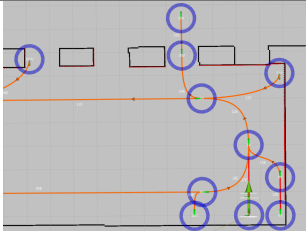
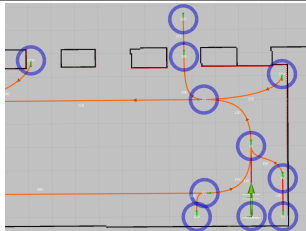
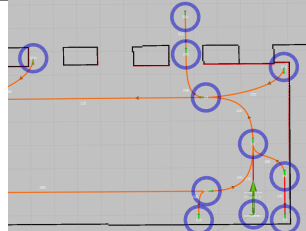
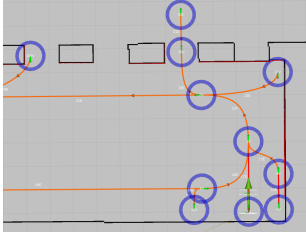
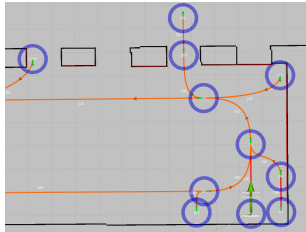
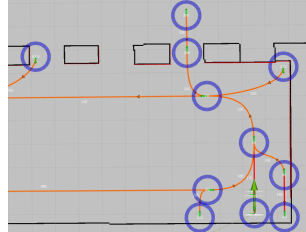
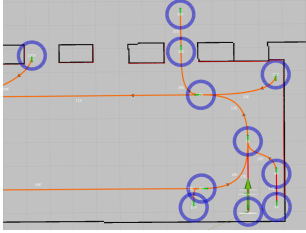
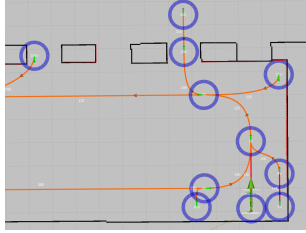
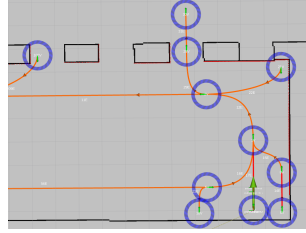
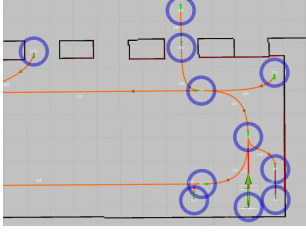
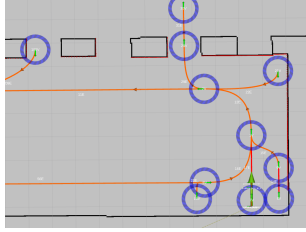
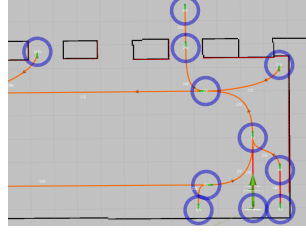
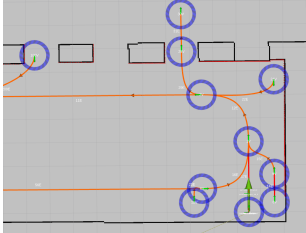
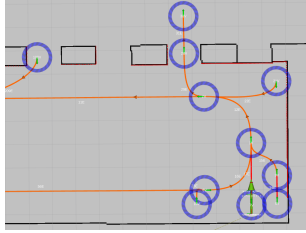
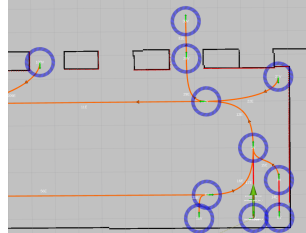
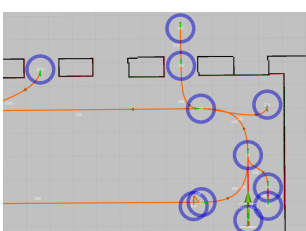
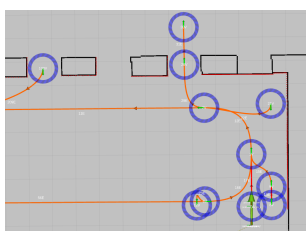
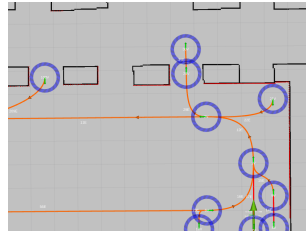
Right side			
Size	Omnidirectional	Differential	Tricycle
1x			
1.25x			
1.5x			
1.75x			
2x			
3x			

Table A.3: Different sizes: Right side (extended version)

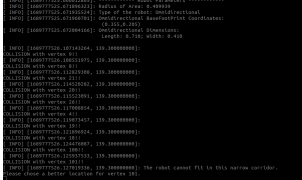
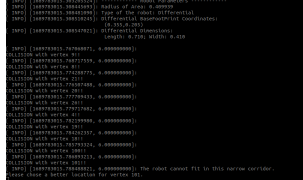
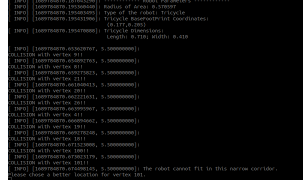
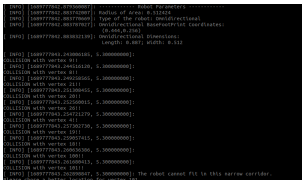
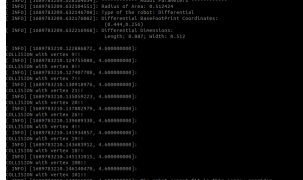
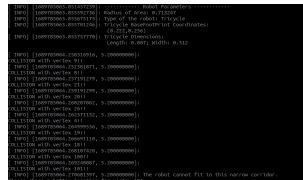
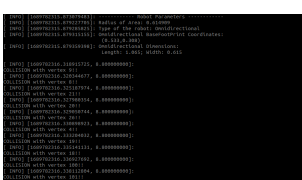
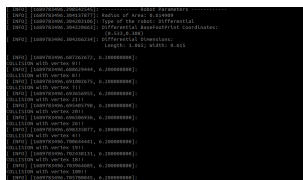
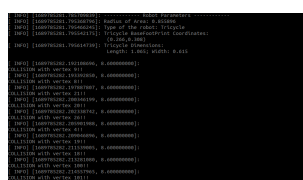
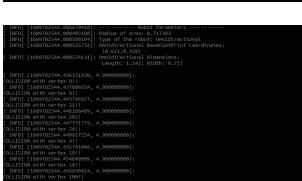
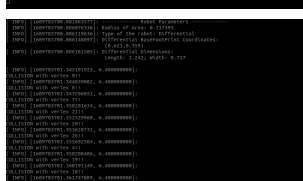
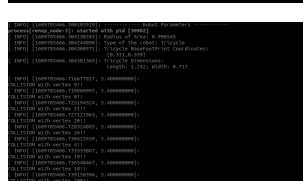
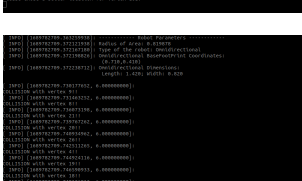
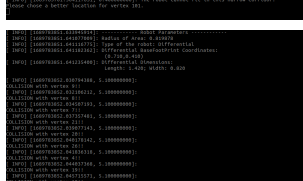
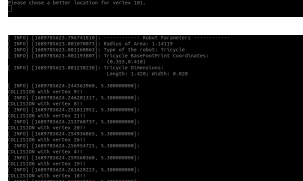
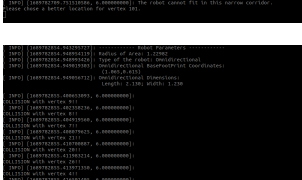
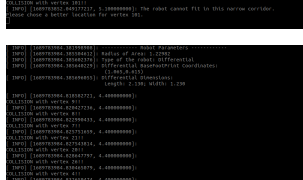
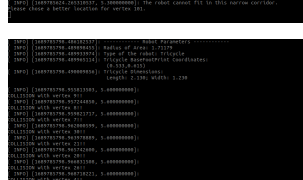
Console Output			
Size	Omnidirectional	Differential	Tricycle
1x			
1.25x			
1.5x			
1.75x			
2x			
3x			

Table A.4: Different sizes: Console Output (extended version)