UNIVERSIDAD PANAMERICANA

Facultad de Ingeniería

Con estudios incorporados a la
Secretaría de Educación Pública

A Comprehensive Autonomous Navigation Strategy for Mobile Robots using the Grey Wolf Optimizer with a Fuzzy Logic Controller and a Sense-to-avoid System based on Optical Flow

T E S I S

para obtener el título de
MAESTRÍA EN INGENIERÍA

P R E S E N T A
ESP. IRVINE JORDAN MONROY RUEDA DE LEÓN

DIRECTOR DE TESIS:
DR. ERNESTO MOYA ALBOR

CDMX 2024

Abstract

In the field of autonomous robotics, the development and deployment of mobile robots have significantly expanded due to technological advances. This has made the need for modern navigation strategies essential for enabling robots to operate safely in diverse environments. This thesis focuses on mobile robots and their navigation capabilities, emphasizing path and motion planning in both static and dynamic scenarios. The general objective is to develop a comprehensive autonomous navigation strategy for a wheeled mobile robot (WMR) able to navigate in static and dynamic environments by integrating global and local path planning with motion control. The methodology involves global path planning using the Grey Wolf Optimization (GWO) algorithm, local path planning using Optical Flow and ultrasonic sensors for obstacle detection, and motion control through Fuzzy Logic Controllers (FLCs). The results indicate that the GWO algorithm successfully identifies feasible paths in environments with up to 80% obstacle density. The FLC effectively follows these paths with minimal deviation, ensuring the robot navigates accurately and safely. The Sense-to-avoid System based on Optical Flow demonstrated high reactive capability, allowing the robot to evade both static and dynamic obstacles efficiently. This research contributes to the research of autonomous mobile robotics, offering a robust solution for modern navigation challenges.

Keywords: Robotics navigation, Global path planning, Local path planning, Path Following, GWO, Optical flow, Fuzzy logic controller

Resumen

En el campo de la robótica autónoma, el desarrollo y despliegue de robots móviles se ha expandido significativamente debido a los recientes avances tecnológicos. Esto ha hecho que la necesidad de estrategias de navegación modernas sea esencial para permitir que los robots operen de manera segura en diversos entornos. Esta tesis se centra en los robots móviles y sus capacidades de navegación, enfatizando la planificación de rutas y movimientos en escenarios estáticos y dinámicos. El objetivo general es desarrollar una estrategia integral de navegación autónoma para un robot móvil con ruedas (*WMR*, por sus siglas en inglés) capaz de navegar de en entornos estáticos y dinámicos mediante la integración de la planificación de rutas globales y locales con control de movimiento. La metodología implica la planificación de rutas globales utilizando el algoritmo de Optimización de Lobo Gris (*GWO*, por sus siglas en inglés), la planificación de rutas locales utilizando un sensor de visión para la estimación de flujo óptico y sensores ultrasónicos para la detección de obstáculos, y la dirección del movimiento a través de Controladores de Lógica Difusa (*FLC*, por sus siglas en inglés). Los resultados indican que el algoritmo *GWO* identifica con éxito rutas factibles en entornos con hasta un 80 % de densidad de obstáculos. El *FLC* sigue eficazmente estas rutas con una desviación mínima, lo que garantiza que el robot navegue con precisión y seguridad. El sistema de detección de obstáculos basado en flujo óptico demostró una alta capacidad reactiva, permitiendo al robot evadir obstáculos estáticos y dinámicos de manera eficiente. Esta investigación contribuye al campo de la robótica móvil autónoma, ofreciendo una solución sólida para los desafíos de navegación modernos.

Agradecimientos

Agradezco a la sublime Universidad Panamericana y a sus profesores por proporcionarme un entorno académico de excelencia durante más de diez años y por darme la oportunidad de crecer tanto profesional como personalmente.

A mi asesor, el Dr. Ernesto Moya Albor, por su invaluable orientación desde que entré a la carrera y por el apoyo y paciencia durante todo el proceso de mi investigación.

A la Dra. Laura Trujillo Liñán y al Lic. Ricardo Meneses Calzada, por la confianza depositada en mí desde la preparatoria y por las facilidades brindadas durante mis estudios universitarios, incluyendo la beca para completar esta maestría.

A mis padres, Leticia y Guillermo, por su amor, comprensión, apoyo y motivación constantes.

A mis hermanos, la Lic. Lilian Monroy y el Arq. Guillermo Monroy, por ser fuente de inspiración y ejemplo a seguir.

A la Ing. Estefanía Botello, colega y compañera incondicional.

A mis familiares, amigos y a todas las personas que siempre me brindaron una palabra de aliento y admiración.

A Dios.

Contents

List of Tables

List of Figures

Introduction

In the field of autonomous robotics (the realm where machines come alive), the development and deployment of mobile robots have expanded significantly in various sectors due to technological advances, making the need for modern navigation strategies essential to enable robots to operate safely in diverse environments. This thesis focuses precisely on mobile robots and their navigation capabilities, highlighting the importance of path and motion planning in both static and dynamic scenarios. This introductory chapter describes the relationship between the growing demand for mobile service robots and the need for robust autonomous navigation algorithms capable of detecting and avoiding obstacles in real time. Subsequently, the specific problem to be solved is identified and justified, which allows the defining of the hypothesis and general objective that delimit and guide the research work to resolve it.

1. Problem Statement

The development of autonomous robotics systems is a rapidly growing sector with a diversity of professional applications [1]. Technological advances have allowed the migration from engineering and science laboratories to industrial applications, transportation, manufacturing, and everyday life in general, making it common to even observe in homes how robots can help in almost any task efficiently, including serving meals, assisting humans, or cleaning rooms [2].

Particularly, the world is benefiting from the *Autonomous Mobile Robots (AMRs)*, automatic machines capable of moving around in a variety of environments, such as indoor spaces or outdoor terrain, and not fixed to one physical location [3]. Although autonomous mobility has been

studied since the middle of the last century, computing power, sensor accuracy, and powerful open-source software have led to mass production and deployment of these robots in various fields such as domestic use [4], automation industries [5], medical and healthcare sectors [6], schools [7], tourism and hospitality [8], construction sites [9], planetary exploration [10], patrolling [11], rescuer operations [12], surveillance [13], and many more industrial and non-industrial domains. The versatility of uses given to AMR is due to its ability to reduce the probability of errors, save resources, and reduce costs associated with human operation [14].

According to the International Federation of Robotics (IFR) in their last Executive Summary for Service Robots presented in 2023 [15], there are more than 975 service robot producers worldwide who sold 158,000 robots in 2022 (48% more than in 2021). AMR solutions are well-established in the transportation and logistics sector, with a 44% increase in units sold. While traditional applications sales remain the primary revenue channel, Robotics as a Service (RaaS) business models are gaining popularity, evidenced by a 67% growth in the RaaS fleet, including the rising demand for hospitality robots with a 125% increase in sales or the continuous efforts to use robots for agriculture, with an 18% increase in sales in 2022. Although sales of medical robots saw a 4% decline, the demand for professional cleaning robots increased by 8%, and 18% more robots for search, rescue, and security were sold.

From this growing demand in RaaS and the great variety of new applications arises the need to improve and generalize the way a robot moves. The process in which a robot reaches a given goal position from its current location is known as *Robotics navigation* [16]. The navigation process needs to analyze the initial and final configurations of a mobile robot to verify the possibility of continuous motion between them. If such a motion exists, the strategy must be capable of identifying it. However, autonomous navigation is not a simple task. According to [17], the primary challenge is not the navigation process itself but rather the reliable acquisition and extraction of environment information and the automatic correlation of that data with an action.

In specific, *path planning* is one of the most crucial tasks in the AMR navigation problem which requires the robot to find an optimal path based on desired performance outcomes such as the shortest time, the shortest route or the lowest energy consumption [18]. The path planning problem can be divided into two categories according to the type of environment information; namely *global path planning* and *local path planning*. Global path planning assumes complete knowledge of the environment and is best applied when the environment is static. The global

path is typically generated offline before the robot begins to navigate. On the other hand, local path planning assumes incomplete knowledge of the environment and is usually invoked online based on data from onboard sensors. It is thus more suited for navigation in unknown or dynamic environments. The most defining feature of an AMR is its ability to navigate without disruption by generating a new hazard-free local path when a previously unforeseen obstacle is detected and re-route to ensure the change in the planned global path [19]. Figure 1 illustrates the scenario where a local path is generated to avoid an obstacle that coincidentally crosses the global path as the AMR moves towards the goal.

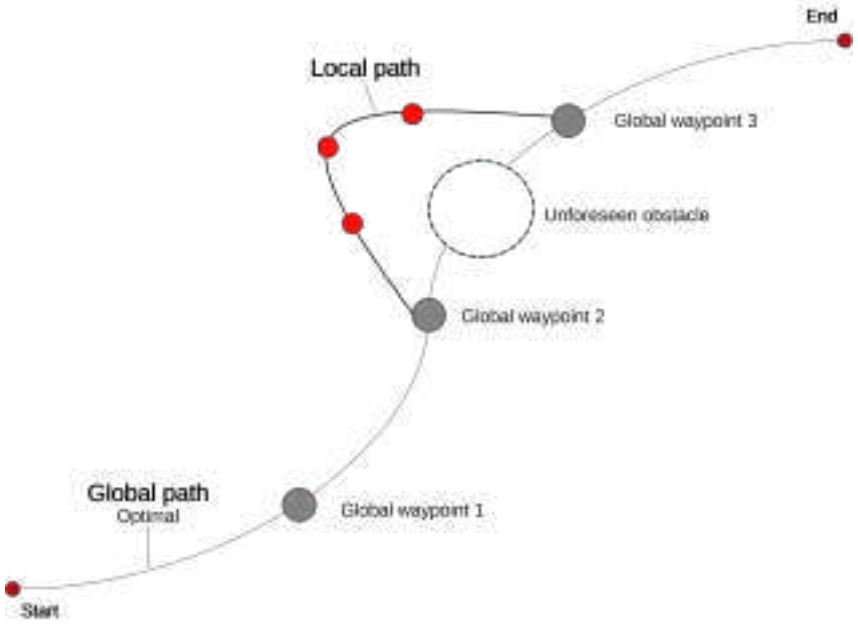


Figure 1: Global and Local Path Planning

As expected, complex environments and dynamic scenarios do not allow a prior identification of all potential obstacles. Consequently, it is imperative to enhance AMRs with advanced navigation strategies, which must be able to detect and avoid impending obstacles in real time and quickly recalculate safe and obstacle-free routes.

Justification

Navigation within dynamic environments is essential for representing real-world scenarios, but it has historically received less attention in the literature compared to navigation in static environments [20]. This disparity highlights a significant gap in the research that could be pivotal for advancing the real-world applicability of autonomous navigation systems. Furthermore, the majority of existing research predominantly features simulation analysis with a significantly smaller

proportion of studies examining real-time applications and without even considering *motion control planning*, which is a branch of robotic navigation that focuses on precisely controlling the movement of the robot's actuators to complete the planned path [21]. While simulation testing is more cost-effective and less time-consuming, it is imperative to direct research toward real-world implementations, considering at least the path following process that involves the real mechanical configuration of the robot. This approach is crucial as it more accurately reflects the current limitations in robot locomotion and mobility, which are likely to influence the overall performance of AMR in practical settings. This shift in research focus is essential for bridging the gap between theory and their application in dynamic and real-world environments.

In this work, a complete navigation strategy is developed that considers the implications of static and dynamic environments, through the complete execution of a control pipeline that includes global and local path planning, and motion planning. In this way, the global path planning deliberative control is used to optimize the route distance between a starting point and a static target before even moving to the destination position. Moreover, sensor-based reactive control of local path planning is used to avoid new obstacles or dynamic objects. Therefore, motion planning uses the knowledge of both methods to move toward the destination following the created route and avoiding collisions. The expected behavior of the robot is similar to that described in Figure 1.

Contribution of the Work

The research presented in this thesis does not aim to solve the paradigm of mobile robotic navigation but contributes by presenting a different and comprehensive solution.

A unique aspect of this navigation strategy is the integration of important techniques used in the literature, such as Optical Flow and Fuzzy Logic Controllers, whose combination with the Grey Wolf Optimizer Algorithm has not been previously explored in the literature. This combination allows for a complete pipeline of navigation strategy from planning to movement execution.

Additionally, the navigation strategy is presented and tested in a dynamic real-time simulation environment, considering the characteristics and mobility restrictions of a Wheeled Mobile Robot (WMR), thus closing the gap between theoretical implementation and its application in a real environment, an aspect often overlooked.

2. Hypothesis

The need to deal with complex dynamic environments has led to more efficient control algorithms for autonomous navigation systems. But even in the most sophisticated algorithm, the essence of the problem lies in its ability to:

Move an object from point A to point B efficiently, safely, and accurately.

In this context, "Move an object" refers to "Move a robot", but how does the robot move? Beyond all the development behind the robot movement [22], this question is mostly resolved by analyzing its mechanical configuration and locomotion system or, in other words, by knowing which robot is being controlled. Controlling a WMR is not the same as guiding a robot with legs or an Unmanned Aerial Vehicle (UAV).

On the other hand, "From point A to point B" refers to the context in which the robot will move: What is their environment?, What type of terrain will the robot move on? What types of obstacles exist? Is it a dynamic or static environment, or a combination?, How fast does it have to move? Is there an available path between A and B?, How to find that path? It is very important to answer these questions since the success of the navigation depends on them. It will not matter if the robot moves safely if it does not fulfill its mission efficiently.

Ultimately, once the path has been found, it must be ensured that the robot is capable of reaching its destination without the help of external intervention and without damaging itself or any external object (i.e. robot travels "safely"), but When a route is no longer safe?

It should be noted that these are only the considerations that arise from the basic problem. As described in the last section, one of the essential things is to test the performance of the robot in the real world, which implies that navigation can be as complicated as wanted: Should the robot reduce its energy consumption? Should the number of turns be limited? Is it necessary to save time? What is the computational limit? No matter how complex the problem to be solved is, what must be considered is that there are so many variants that at this moment it would be impossible to take into account each of the factors involved, so the solutions become so specific as implementations.

Based on these inquiries and the considerations found during the problem statement section, the following main hypothesis is established to correctly guide the research work of this thesis:

Hypothesis 1 (H1): By integrating a global path planning strategy with a local path planning strategy as inputs for the motion control of the robot, the AMR can complete efficiently, safely and accurately a route from a start position to a goal position in an environment with static and dynamic obstacles.

Since there are several considerations, it is necessary to define auxiliary hypotheses for this research. Figure 2 shows the interaction between the variables and the hypotheses in a graphical representation.

- **Hypothesis 2 (H2):** By developing a deliberative global path planning strategy, finding the efficient and feasible path between a start position and goal position in a known static environment is possible.
- **Hypothesis 3 (H3):** By developing a reactive sensor-based local path planning strategy, it is possible to safely avoid unknown static obstacles and dynamic obstacles as the AMR moves towards the goal.
- **Hypothesis 4 (H4):** By developing a motion planning strategy that considers locomotion limitations, the robot can accurately follow the path found under certain conditions.

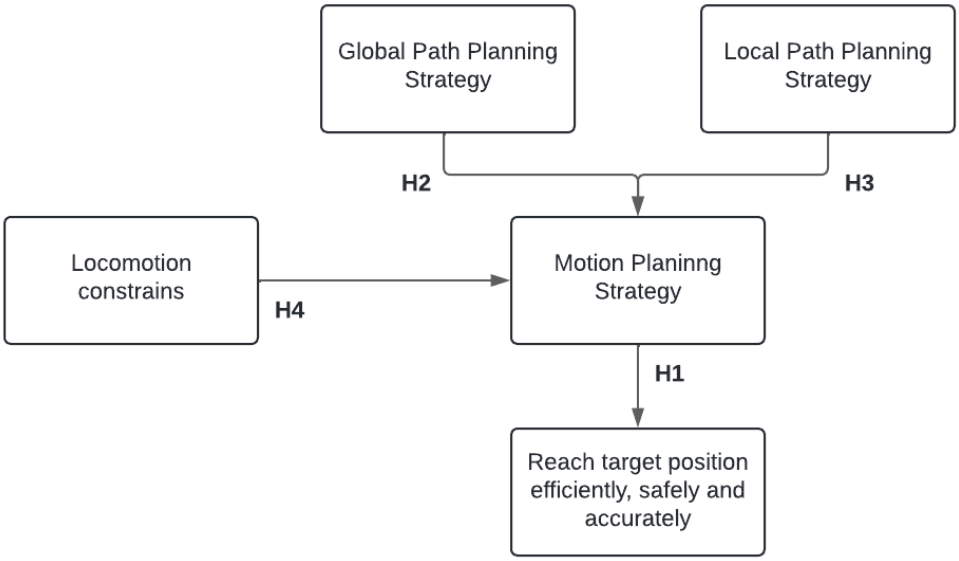


Figure 2: Graphical representation of the research hypothesis (H1) and the interaction with H2, H3, and H4

3. General Objective

Aligned with the need to test navigation algorithms in dynamic environments and considering the implications of the defined hypotheses, the general objective is to develop and implement a comprehensive autonomous navigation strategy so that a WMR is capable of moving in static and dynamic environments.

Three specific objectives are additionally considered in this research, which are described below.

1. Implement a global path planning strategy to find a feasible route between a starting position and a target point in an environment with static obstacles.
2. Implement a sensor-based reactive local path planning strategy to avoid unknown dynamic obstacles while the WMR moves along the global path.
3. Perform the motion planning strategy using the insights from both global and local path planning and test its functionality in a partially dynamic virtual environment.

Therefore, the functionality metrics of the proposed navigation strategy will be aimed at verifying that the robot followed the route automatically efficiently, safely, and accurately. In this context, "automatic" means finding an available route and following it from the beginning to the final position without human intervention. "Efficient" means that the route found is the shortest or at least a satisfactory approximation, while "safe" refers to guaranteeing collision-free movement. Finally, "accurate" indicates that a robot followed the found trajectory.

4. Thesis Organization

This document is organized into 4 chapters. In Chapter 1, the relevant information and theories that characterize the problem of autonomous navigation for AMRs are presented. It also describes how previous research has faced this problem and identifies its advantages, disadvantages, and areas of opportunity in new implementations.

Chapter 2 explains in detail the process followed to implement the solution based on the insights identified in Chapter 1. All methods, analysis procedures, and materials used are described in this chapter.

In Chapter 3, the methodology and the solution itself are validated. For this, the testing of the proposed solution is presented, as well as its analysis, discussion, and evaluation. The interpretation of the results is compared with the proposed hypotheses, to demonstrate that the proposal solves the problem and to explain the limitations found in the development of the solution.

Finally, Chapter 4 presents the conclusions reached after analyzing the results. Based on this, suggestions are proposed for future work related to this problem.

Chapter 1

Theoretical Framework & Literature Review

This chapter provides a comprehensive overview of the theoretical framework and literature related to the robotics navigation problem. It details the theoretical formulation of navigation, addressing key functions such as localization, mapping, and path and motion planning, with particular emphasis on the latter. The chapter reviews classical and heuristic path planning methodologies, highlighting their strengths and limitations. The integration of global and local path planning is examined to ensure efficient and safe navigation in dynamic settings. Finally, the need for effective motion planning and robust perception systems is emphasized, highlighting the continuous interaction between the robot and its environment for adaptive navigation.

1.1. Robotics Navigation Problem Overview

The world is definitely on the threshold of a robotics revolution. Numerous robotic systems are being developed, and their capabilities have been demonstrated to be efficient in a variety of circumstances, such as smart homes [23], airports [24], supermarkets [25], and industrial plants [26]. This is because robots are equipped with a type of intelligence to ensure optimal execution of tasks and complete their mission. This cognition generally represents the decision-making process that a system utilizes to achieve its highest-level goals. Nevertheless, integrating intelligence into robotic systems necessitates solving a wide variety of research issues. In the case of an AMR, the specific aspect of cognition capability is directly linked to providing *navigation competence* [27].

The classical mobile robot navigation problem consists of finding a path from a starting location to a goal position while safely avoiding any obstacles [28]. There is a common belief that this operation is just obstacle avoidance or collision detection, but robotics navigation involves a lot more than just that. It requires creating a collision-free path for a robot that may have complicated geometry and unique dynamics in a (mostly) unknown environment with moving obstacles of various shapes. As a result, the navigation planning strategy must address physical, geometric, and temporal restrictions.

A theoretical formulation of the navigation problem is presented to make this completely clear [29]. Assuming a mobile robot R in a given environment E, the objective is to generate a sequence of motion commands $A^* = \{a_0, a_1, a_2, \ldots\}$ to move the robot from its current start location s to a desired goal location g :

$$A^* = \arg\min_{A \in \mathcal{A}} J(R, E, s, g) \tag{1.1}$$

where $\mathcal{A}$ is the space of all possible sequences of motion commands and $J(\cdot)$ is a cost function that the navigation system aims to minimize. The action space depends on the type of robot R being controlled. For differential-drive robots, the actions can be linear and angular velocities; while for Ackermann-steer vehicles, the actions could be focused on steering control. On the other hand, the environment E is created using pre-designed maps of the surrounding area and/or using the robot's sensory perceptions (such as LiDAR, camera, wheel encoder, or inertial sensor), either in 2D or 3D for ground or aerial navigation. J can take many different forms depending on the navigation circumstance, including the robot's motion limitations, obstacle avoidance, shortest path or time, off-road stability, social conformity, and other domain-dependent measures.

The system in a robot that accomplishes these tasks and provides directions to a destination is called the *navigation system*, which translates a specified high-level task (e.g. "go to the kitchen" or "pick up a package at the entrance") into low-level descriptions of how the robot should move (e.g. "turn 90 degrees to the left" or "move forward 10 meters") [28]. To plan these low-level descriptions a robot needs a representation of the environment and localizes itself on this map. Furthermore, to increase the probability of reaching the target destination, must also evaluate potential dangers arising from the surrounding environment and modify its actions accordingly. Summarizing, to solve the robot navigation problem, it is necessary to determine the answers to the following three questions: "Where am I?", "Where am I going?", and "How do I get there?"

Respectively, these three questions are answered by the three fundamental navigation functions: *localization*, *mapping*, and *path and motion planning* [30]. This general and simplified version of the architecture of a navigation system is depicted in Figure 1.1. Since each of these functions represents a challenge, this research is focused solely on developing a path and motion planning process, assuming that when a robot navigation system is described, it has a localization and mapping module.

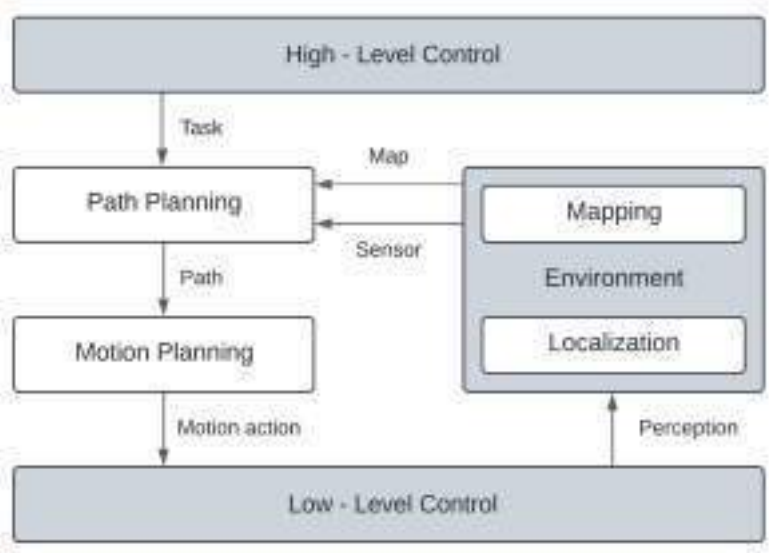


Figure 1.1: Simplified representation of a Mobile Robot Navigation System. The three functions the robot must accomplish are in between a high-level layer and a low-level layer.

1.1.1. Localization and Mapping

Localization function consists of determining the robot's location and orientation in an environment. A robot must use its observations to infer its possible global or relative location on the map. Many techniques have been implemented for localization, including the use of cameras [31], GPS in open spaces [32], ultrasonic sensors [33], and laser rangefinders [34]. The position can be expressed as an absolute coordinate (latitude, longitude, altitude), as an absolute reference (such as "center of a room"), or as a topological coordinate (such as "Room 23").

An AMR requires a map of its environment to identify where it has been traveling so far, and *Mapping* is the process of obtaining it. The robot uses this map to determine its location and routes. The map can be incrementally constructed as the robot learns about its new surroundings, or it can be manually entered into the robot's memory. In robotics navigation, mapping is a problem that is often ignored [35].

1.1.2. Path and Motion Planning

At first glance, path planning and motion planning can be intended to mean the same thing. Depending on the author and the article being consulted; these terms are considered synonymous, and even they are used interchangeably in robotics. This misconception may be due in part to the fact that both processes address aspects of the question, "How do I get there?" [36]. For this research work, path planning and motion planning are two different processes and both will be addressed to achieve a fully functioning robot.

Path planning, on the one hand, is the process that constructs a feasible route from a starting point to an ending point given a full, partial, or dynamic map. In this context, thinking about path planning should answer the following questions: "In which direction is the robot moving?", "Where is the next position?" and "How can the robot get to the destination with this obstacle in its way?".

Meanwhile, *motion planning* is the process by which a set of actions is defined to determine how to execute on the previously planned path. This task, also called motion control, is related to the path following process and answers questions like: "How to make the motors move?', "What type of sensors does the robot use?", and "What type of locomotion does the robot have?", among many others.

Although path planning and motion planning are decidedly different processes, they involve some coincidences. Therefore, the answer to the question: "How do I get there?" is divided into two components: path planning solves the spatial aspect of navigation, and motion planning addresses the mechanical components. Combined, both plans offer a complete procedure to move a robot from one point in space to another.

Moreover, these processes can affect one another when they interact. In robotics, path planning is typically used to handle navigational problems, with motion planning used afterward. In this way, path planning expands to motion planning because the trajectory influences the action commands for the robot's motors. This does not imply that once movement starts, path planning is finished and never done again. Path planning for a robot can be impacted by its current motion, and adjustments to the path may be necessary. The interaction between path planning and motion planning is visualized in Figure 1.2 [36].

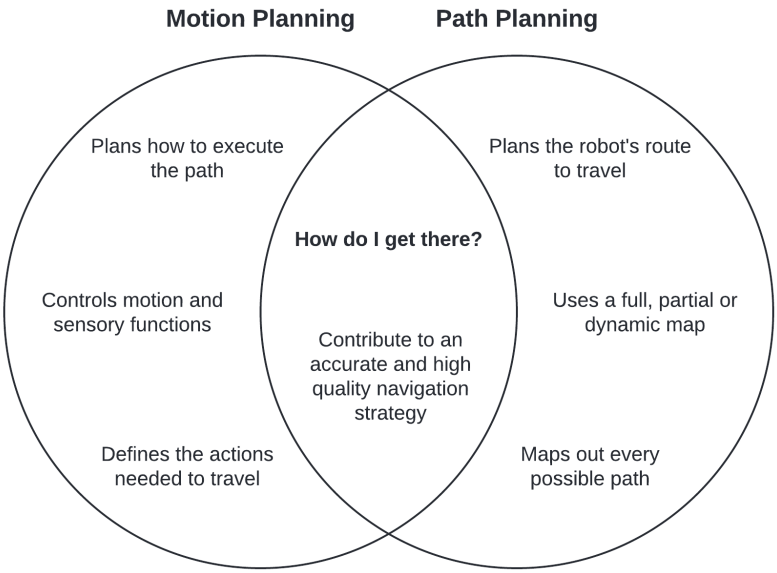


Figure 1.2: Path and Motion Planning interaction

The study, analysis, and application of both functions and their respective solutions are part of the objective of this thesis. So the rest of the chapter presents the relevant information, theories, and solutions that characterize these two aspects of navigation.

1.2. Path Planning

Path planning refers to the challenge of getting a robot to automatically decide how to move while avoiding collisions with objects. The formulation of this problem finds its origin in *The Piano Mover's Problem*, presented by Schwartz and Sharir in 1983 [37], which describes the challenges that movers face when navigating barriers with a piano without lifting it. Similarly, a robot must confront the basic problem of reasoning geometrically about its environment and planning to avoid collision.

The path planning problem emerges from the simplification of the basic navigation planning problem [28]. As explained before, path and motion planning are both part of solving the same task. In the basic path planning problem, the dynamic properties of the robot are ignored. Furthermore, movements are restricted to non-contact motions, so that the issues related to the mechanical interaction between two physical objects in contact can be ignored. These assumptions convert the path planning process into a purely geometric problem, while physical motion control considerations are taken into account during path execution, in the motion planning function. Even more, the geometric issues are further simplified by assuming that the robot is a single rigid object, so the movement of this object is only limited by the obstacles.

The basic path planning problem resulting from these simplifications is the following [38]:

> Let $\mathcal{R}$ be a single rigid object (the robot) that moves in a Euclidean space $\mathcal{W}$, called the *workspace*, represented as $\mathbb{R}^N$, with $N = 2$ or 3.
>
> Let $\mathcal{O}_i$ for $i = 1, \ldots, n$ be a number of rigid objects distributed in $\mathcal{W}$ called *obstacles*. The union of all obstacles is called the *obstacle region* and is denoted as $\mathcal{WO}$. The robot $\mathcal{R}$ must avoid touching the obstacle region.
>
> Assume that both the geometry of $\mathcal{R}$ and $\mathcal{O} \subset \mathcal{W}$ and the positions of $\mathcal{O}_i$'s in $\mathcal{W}$ are accurately known. Assume further that no kinematic constraints limit the motions of $\mathcal{R}$.
>
> *Given the initial (q_i) and goal (q_g) positions and orientations of $\mathcal{R}$ in $\mathcal{W}$, find a path c in the form of a continuous sequence of positions and orientations of $\mathcal{R}$ that do not collide or contact with $\mathcal{WO}$, that will allow $\mathcal{R}$ to move from its starting position and orientation q_i to its goal position and orientation q_g. Report failure if such a path does not exist.*

Informally, the basic path planning problem can be summarized into the following form: Given an initial robot configuration, compute how to gradually move the robot toward a desired goal placement, making sure it never crosses the obstacle region [39]. In addition to this definition, one may define a quality of measure on all possible paths and require that the path that optimizes this measure is found. Therefore, the necessary inputs of the path planning task are the initial position of the robot, the location of the intended target, and a geometric description of the obstacle zone and the robot. The expected results are a detailed explanation of how to advance the robot gradually from the initial position to the desired position, avoiding contact with the obstacle region.

1.2.1. Configuration Space

Although the path planning problem must be conducted in the workspace, the solution lives in another space. More precisely, this new domain describes the pose of all possible rigid body configurations that may be applied to the robot relative to a fixed coordinate system. A possible configuration implies that no point on the robot collides with an obstacle during the solution. This set of all possible configurations is called the *configuration space* or *$\mathcal{C}$-space*, concept that

was introduced more than 40 years ago [40], but it is still relevant because the robot configuration is represented as a single point, causing the problem of path planning of a robot arbitrarily to be transformed into the problem of plan the movement of a point. Moreover, this standardized framework enables the use of the same planning methods to handle a wide range of distinct motion problems in terms of geometry and kinematics. The use of the $\mathcal{C}$-space representation has so generalized solutions to the path planning problem that kine-dynamic considerations have been neglected. As mentioned previously in this research work, although the configuration space is used, one of the objectives in addition to performing path planning is to implement motion control planning to follow the path found and keep the development of navigation strategies as close to reality as possible.

Let's consider the configuration space $\mathcal{C}$ as the space of all configurations of the robot. A configuration, denoted as q, is simply a point in this abstract configuration space. The subset of the workspace $\mathcal{W}$ that is occupied by a configuration q of $\mathcal{R}$ is denoted as $\mathcal{R}(q)$. Cartesian coordinates are used to describe the position of a body (consider a value in the Euclidean space $\mathbb{R}$), while angular coordinates are used to represent the rotation of that body (consider a value in the Special Orthogonal Group SO). Therefore, the configuration space of a robot is then obtained in general as a Cartesian product of these two spaces [28]. For example:

- If the robot is a single point translating in $\mathcal{W} = \mathbb{R}^2$, $\mathcal{C}$ is a plane, and a configuration q can be represented using two parameters (x, y).
- If the robot is a 2-dimensional shape that can translate and rotate, still $\mathcal{W} = \mathbb{R}^2$. However, $\mathcal{C} = \mathbb{R}^2 \times SO(2)$, and a configuration q can be represented using three parameters (x, y, θ).
- If the robot is a 3-dimensional shape that can translate and rotate, $\mathcal{W} = \mathbb{R}^3$ and $\mathcal{C} = \mathbb{R}^3 \times SO(3)$, and a configuration q requires six parameters: (x, y, z) for translation, and, e.g., three Euler angles (ϕ, θ, ψ) for rotation.

After $\mathcal{C}$ has been specified, the objective is to identify a path c that takes the shape of a continuous series of configurations of $\mathcal{R}$, avoiding contact or collisions with $\mathcal{O}_i$, from an initial configuration q_i to a goal configuration q_g. The space of configurations for which a collision or contact occurs is defined by mapping the obstacles in the configuration space. A $\mathcal{W}$-obstacle in $\mathcal{C}$ is called a $\mathcal{C}$-obstacle and is defined as:

$$\mathcal{CO}_i = \{q \in \mathcal{C} \mid \mathcal{R}(q) \cap \mathcal{O}_i \neq \emptyset\} \tag{1.2}$$

The union of the total configuration space obstacles (n_o),

$$\mathcal{CO} = \bigcup_{i=1}^{n_o} \mathcal{CO}_i \tag{1.3}$$

is called the *configuration space obstacle region*. Its complement is

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{CO} \tag{1.4}$$

and is called the *free configuration space*. The basic path planning problem can now be defined as finding a path from q_i to q_g in $\mathcal{C}_{free}$. A path is defined as a continuous function c that maps a path parameter s (usually taken in the unit interval $[0, 1]$) to a curve in $\mathcal{C}$. So a path is defined as a continuous function

$$c : [0, 1] \to \mathcal{C} \quad \text{where} \quad c(0) = q_i, \quad c(1) = q_g \quad \text{and} \quad c(s) \in \mathcal{C} \ \forall s \in [0, 1] \tag{1.5}$$

Analogously, a *free path* is defined as a continuous function $c : [0, 1] \to \mathcal{C}_{free}$, as illustrated in Figure 1.3 [28]. That is, if $c(0)$ and $c(1)$ belong to the same connected component of $\mathcal{C}_{free}$. With $\mathcal{C}_{free}$ defined as the complement of $\mathcal{CO}$, configurations that belong to both spaces are excluded. The space that contains configurations that represent the robot touching an obstacle is called the contact space and is denoted as $\mathcal{C}_{contact}$. As this might be allowed or could even be desired, these configurations must be represented in $\mathcal{C}_{free}$ as well for some problems. This space is referred to as the closure of $\mathcal{C}_{free}$ or $cl(\mathcal{C}_{free}) = \mathcal{C}_{free} \cup \mathcal{C}_{contact}$. Indeed, it holds that:

$$\mathcal{C}_{free} \subseteq cl(\mathcal{C}_{free}) \subseteq \mathcal{C} \tag{1.6}$$

Correspondingly to the definition of a free path, a continuous function $c : [0, 1] \to cl(\mathcal{C}_{free})$ is defined as a *semi-free path*.

Let's consider the example of an AMR base with a circular geometry moving on a plane, so $\mathcal{W} = \mathbb{R}^2$ (obstacles can be modeled as convex polygonal regions). The configuration of this robot is described by two translations, x and y, and one rotation, θ. The robot's geometry is the same for every rotation as the geometry of the base is circular and thus the rotation is not necessary to describe a configuration. The construction of configuration space for three circular geometries is shown in Figure 1.4 [28]. The dimension of $\mathcal{C}$ is therefore equal to that of $\mathcal{W}$, as illustrated in Figure 1.4a. Although $\mathcal{WO}$ is not identical to $\mathcal{CO}$, to construct $\mathcal{CO}$, the robot must fit $\mathcal{WO}$. As the center of the circular base is chosen for a configuration q, in this case, it is

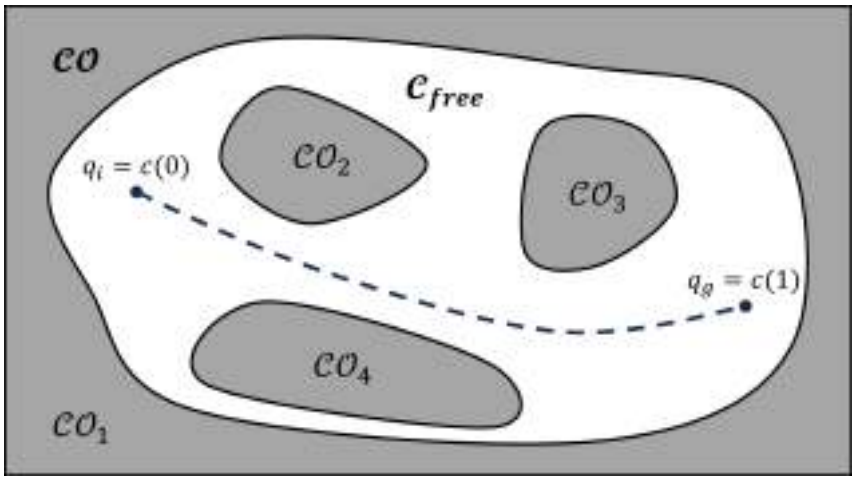
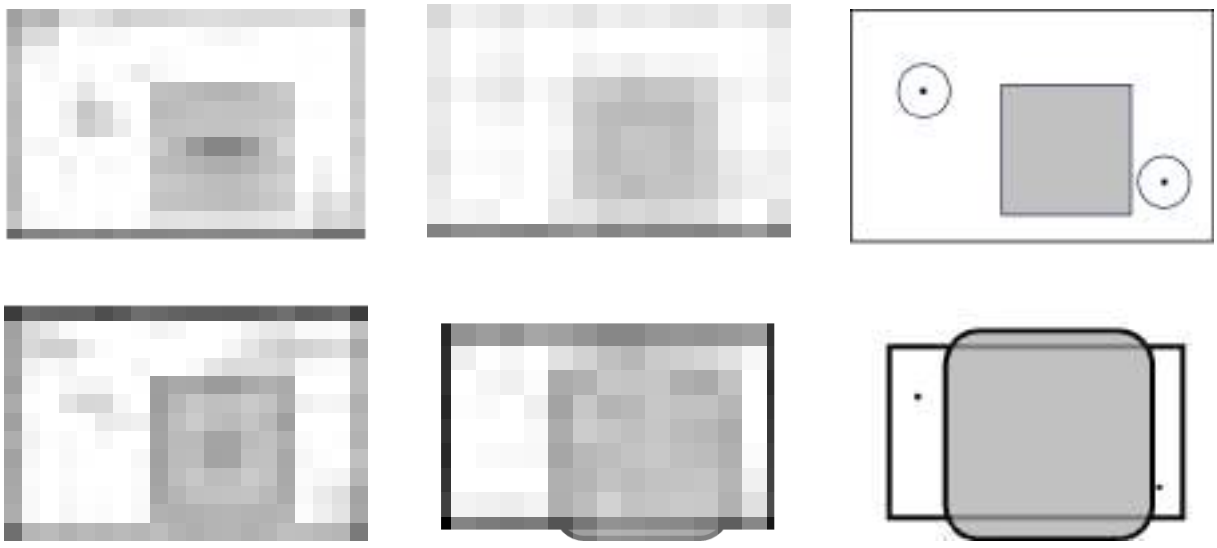


Figure 1.3: Robot Path Planning Diagram in Configuration Space. Path connecting q_i to q_g by a curve on the free configuration space $\mathcal{C}_{free}$

enough to inflate the obstacles with the radius of the circle (Figure 1.4b). At some point, a free path might not be available anymore, as can be seen in Figure 1.4c: q_1 cannot be connected to q_2.



(a) (b) (c)

Figure 1.4: The workspace $\mathcal{W}$ (top row) and configuration space $\mathcal{C}$ (bottom row) for a robot $\mathcal{R}$ at two points, represented as a point (a), a circle (b) and a larger circle (c). $\mathcal{W}$ is populated by obstacles that make up the obstacle region, $\mathcal{WO}$. This region is mapped into $\mathcal{C}$ as the configuration space obstacle region, $\mathcal{CO}$. In subfigure (a) the contact space ($\mathcal{C}_{contact}$) and free configuration space ($\mathcal{C}_{free}$) are also visualized, as well as the two robot configurations q_1 and q_2. In subfigure (b) and (c) these annotations are omitted.

To ensure that the routes developed are valid in both the workspace and in real life, it is crucial to consider the most appropriate representation of the robot as well as the obstacles present in the $\mathcal{C}$-*space*.

1.2.2. Planning in Dynamic Environments

At this point, the basic path planning problem has been presented as completely static with robot $\mathcal{R}$ as the only object moving in the environment. However, in current AMR applications, it is challenging for this to happen. This study will therefore go into depth on dynamic environments and the aspects related to these to be considered in planning.

In this context, a *dynamic environment* is one in which there are moving objects, such as different obstacles or multiple robots $\mathcal{R}_i$. In the presence of moving obstacles, the configuration space changes over time. Then, adding a time dimension to the configuration space can resolve the path planning problem in dynamic environments. This space is called *configuration-time space* [41] and it is denoted as $\mathcal{CT}$. $\mathcal{W}$-obstacles are mapped in static regions called $\mathcal{CT}$-obstacles. A cross-section through $\mathcal{CT}$ at time t represents the configuration space $\mathcal{C}$ of the robot at that moment. For a total time T and a $\mathcal{W} = \mathbb{R}^2$ with an $\mathcal{CT}$-obstacle in piecewise linear motion, three instances of time $t_i \in [0, T]$ are plotted in Figure 1.5 [28]. Path planning now involves finding a way around $\mathcal{CT}$-obstacles.

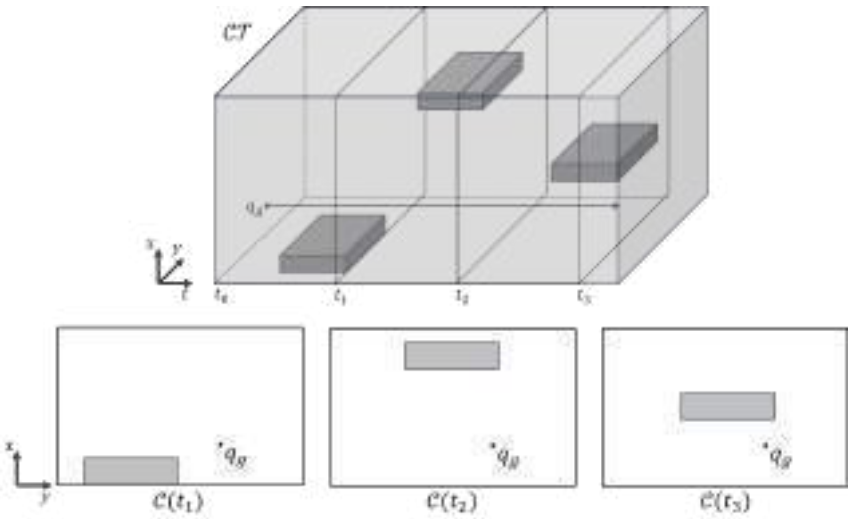


Figure 1.5: Configuration-time space, $\mathcal{CT} = \mathbb{R}^2 \times [0, T]$, for a workspace with a piecewise linear moving obstacle. On the bottom row, three instances or 'slices' of the space are depicted.

Nevertheless, not only the presence of moving objects should be considered, but in a dynamic environment, the exact knowledge of the workspace and the location and geometry of the obstacles cannot be assumed. Furthermore, the planned path is not usually executed exactly. Since assumptions are generally not realistic, uncertainty must be considered: in a priori knowledge of the workspace; in the information from the sensors that are acquired during the execution of the planning; and in the execution of the plan itself [28].

1.2.3. Global & Local Path Planning

The uncertainty caused by the existence of unknown and dynamic obstacles within the environment has made the task of choosing the most appropriate path extend beyond finding a feasible route between two configurations, presenting new significant challenges in both static and dynamic scenarios [14].

Guiding robots to a desired state by navigating through available free space can be done in many ways. In the workspace, there are numerous potential routes connecting two points and there are different methods for finding them. The final selected path typically optimizes a function that addresses a specific objective, such as the duration of travel (i.e. the shortest path with minimum possible time), which is particularly crucial in time-sensitive missions like search-and-rescue operations where timely assistance is vital for disaster victims in life-threatening situations [42]. Energy efficiency is another critical optimization function (i.e. a smooth path and minimum power consumption), for example in scenarios like planetary exploration where robotic rovers operate under severe energy constraints [43]. At the same time, the path generated by the planner must follow any imposed restrictions: locomotion system, adaptability to certain terrains, or traction configuration (i.e. collision-free motion). These considerations and the complexities added to the basic problem consequently reduce the number of feasible paths available to the path planner.

As the basic problem of path planning becomes more complicated, it is necessary to implement a more robust navigation system that, regardless of the complexity, satisfies two primary elements: efficiency and safety [30]. This means that any robot must find its optimal route efficiently while dealing with the circumstances of being in a dynamic environment, such as safely avoiding static and moving obstacles. In addition, a robot must also precisely follow its optimal, obstacle-free path, a process that is described by the motion planning function and which, as mentioned at the beginning of the chapter, is entirely related to path planning. These objectives are initially contradictory since it is not possible to find a fully optimized route if there is no prior knowledge of the presence and position of all obstacles.

In this way, the navigation problem (and more specifically the route planning problem) can be typically categorized into two classes: *global and local navigation* [44]. Each stage focuses on each objective (efficiency and safety) so that the overall planning function can be completed satisfactorily.

These two paradigms differ in terms of distance, scale, obstacle avoidance, and the inability to observe the goal state. Originally, a global planner was defined as a planner that can use the complete workspace information to return a solution, meanwhile, a local planner was referred to be a planner that uses a subset of that same workspace [28]. Although these definitions are still valid, the particularities of each type of planner involve much more than that. Global path planning assumes complete knowledge of the environment and is best applied when the environment is static. The global path is typically generated offline before the robot begins to navigate (that is why it is also known as *deliberative path planning*). On the other hand, local path planning (also known as *reactive path planning*) assumes incomplete knowledge of the environment and is usually invoked online based on data from onboard sensors. It is thus more suited for navigation in unknown or dynamic environments. Table 1.1 summarizes the differences between these two categories.

Table 1.1: Comparison of Local and Global Path Planning

Local path planning	**Global path planning**
Sensor-based	Map-based
Reactive navigation	Deliberative navigation
Fast response	Slower response
The workspace area is partially or completely unknown	The workspace area is known
Generate the path and move toward the target while avoiding obstacles or objects	Generate a feasible path before moving toward the goal position
Done online	Normally done offline

In the real world, both planning approaches are necessary to find the efficient, safe, and accurate route in any type of scenario, as they combine the knowledge of deliberative planning and the faster response of reactive planning. The main challenge of this approach is to coordinate global and local processes so that they are consistent and robust to produce valuable results in real time. The traditional mobile navigation strategy presented in Figure 1.6 shows the expected interaction between *the global and local planning subsystems* [29]. In the global planning subsystem, a global representation of the world is created based on a given map or recent perceptions, with a specific defined goal position (the configuration space). Next, a parameterized global planner takes the goal and world representation as inputs and generates a global efficient route, typically consisting of a series of local optimal goals. Then, in the local planning subsystem, the robot uses its sensor-based perception to create a local representation of the scene and to identify new objects (as unknown moving obstacles). Based on this local information, it selects the closest local goal and modifies it if necessary to safely avoid collisions. Finally, a parameterized local planner generates motion commands that are fed directly to the motor controllers, generating accurate and goal-oriented raw motor commands (considered in the motion planning control).

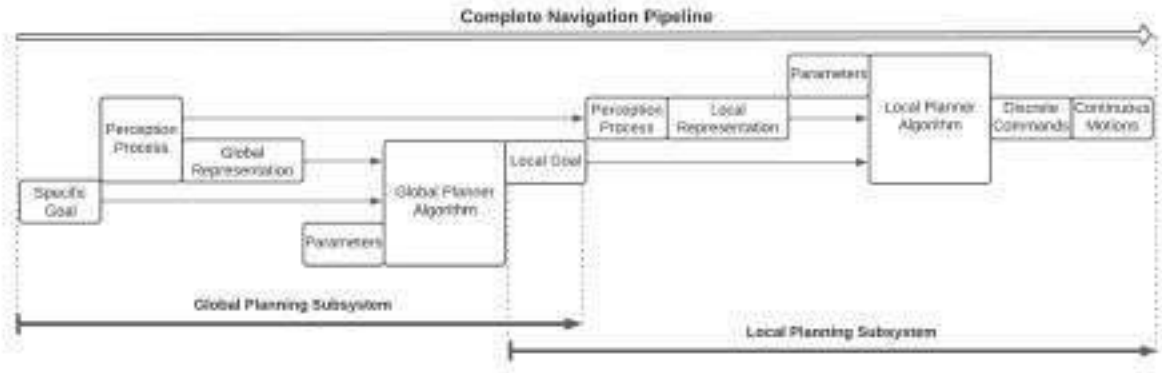


Figure 1.6: Complete Navigation Pipeline

The presented process gives the AMR its most defining characteristic: its ability to navigate without interruption, generating a new local hazard-free route when a previously unforeseen obstacle is detected and redirecting to ensure the change in the planned global path [19]. The AMRs are always deployed in environments full of uncertain factors. For example, serving robots in airports or stations must reasonably predict and react to pedestrians. Therefore, the AMR needs to have the local replanning ability to deal with unexpected situations while tracking the global trajectory. Figure 1.7 illustrates the scenario where a local path is generated to avoid an obstacle that coincidentally crosses the global path as the AMR moves towards the goal [18], as stated previously in Figure 1 during *Problem Statement* section. This is the expected performance in the navigation strategy that is developed as the objective of this research work.

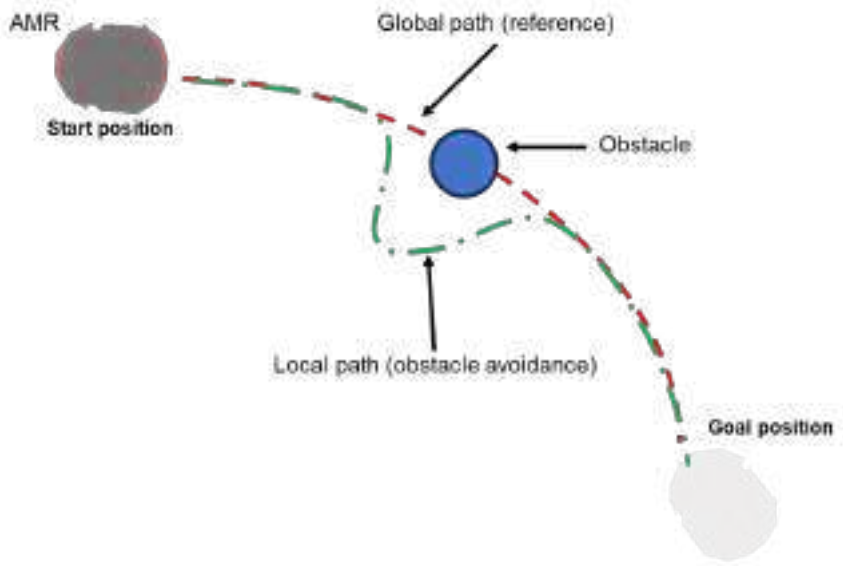


Figure 1.7: Representation of the execution of the Navigation Strategy through Global and Local Planning.

1.3. Path Planning Related Work

Since the 1950s, researchers have proposed many algorithms to find the optimal path of a mobile robot that solves the path planning problem (including both global and local scenarios). In general, the path planning strategies for navigation in mobile robots can be categorized into two distinct groups: *classical and heuristic approaches* [20], as depicted in Figure 1.8. These techniques, however, have their own merits and demerits. By hybridizing two or more, the drawbacks of these techniques can be eliminated/minimized.

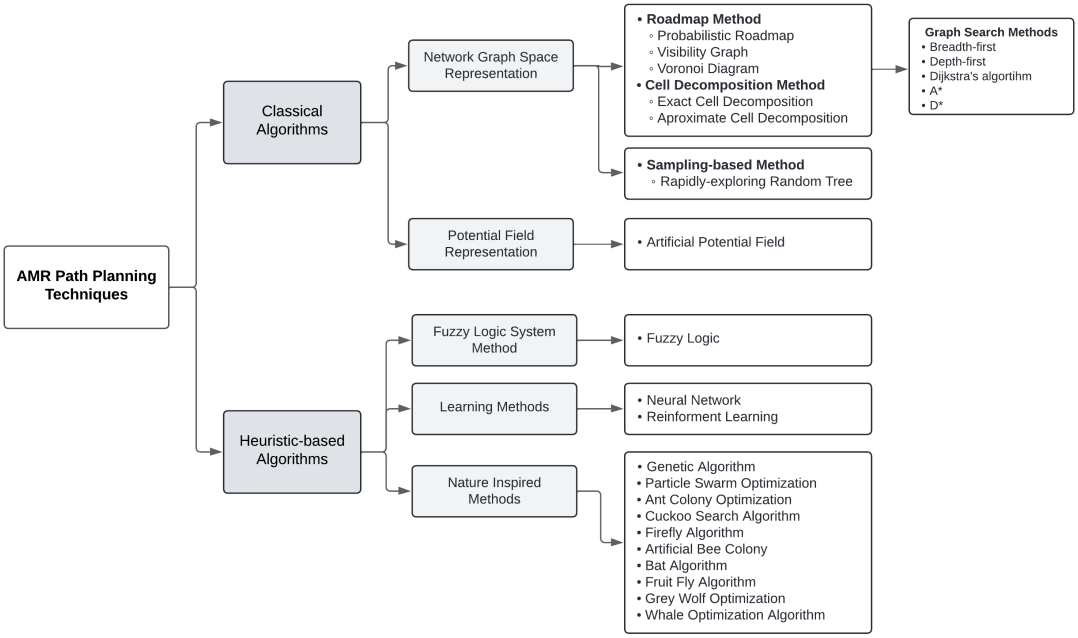


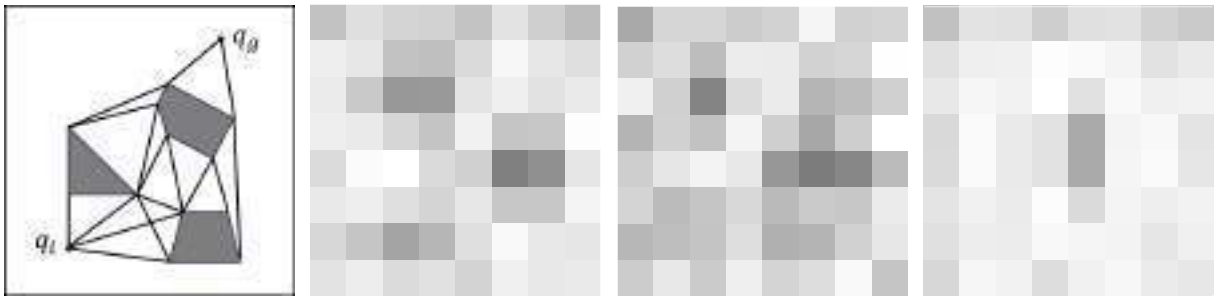
Figure 1.8: Overview of Classical and Heuristic Methods commonly used for AMR Navigation

Historically, classical approaches dominated the field of study in planning optimization, however, several significant limitations emerged, including susceptibility to becoming trapped in local minima, high computational demands, ineffectiveness across diverse environments, and a failure to manage high levels of uncertainty. In response to these challenges, researchers developed heuristic approaches to address and mitigate these drawbacks effectively. Even during the last decade, the number of articles on heuristic approaches to path planning grew at a faster rate than that on classical approaches, with the disparity between them increasing almost exponentially [18]. This trend not only suggests the superior effectiveness of heuristic methods in addressing diverse path-planning challenges but also indicates substantial opportunities for further enhancements. The potential for improvement within heuristic strategies continues to captivate the research community, prompting further investigation and development.

1.3.1. Classical Path Planning

Initially, classical procedures were widely used to solve robot navigation challenges because they were the only techniques that existed. By using classical approaches for performing a task, it is observed that either a result would be obtained, or it would be confirmed that a result does not exist. This strategy is not as desirable for real-time deployment due to its significant computing cost and inability to adapt to environmental variability. Therefore, these methods are quite easy and appropriate to execute in the static setting for robot path planning, which would be considered global planning.

The classic methods are fundamentally based on the way of representing the configuration space. As previously mentioned, the basic path-planning problem of an arbitrarily shaped robot is transformed by the configuration space into the problem of planning the movement of a single point. To be able to search for the correct sequence of configurations, the *connectivity* of the free configuration space ($\mathcal{C}_{free}$) must be captured [28]. This representation can be made in different forms as it is illustrated in Figure 1.9.



(a) Roadmap (b) Cell decomposition (c) Sampling-based (d) Potential Field

Figure 1.9: Classical Techniques for Path Planning that represents the connectivity of a configuration space with arbitrary obstacles

The *Roadmap* (Figure 1.9a), *Cell Decomposition* (Figure 1.9b) and *Sampling-base*d methods (Figure 1.9c) capture the connectivity of $\mathcal{C}_{free}$ into an abstracted graph. Within the roadmap and cell decomposition, the shortest route between the initial and final positions is usually determined via various graph search algorithms such as the breadth-first search, depth-first search, Dijkstra's algorithm (DA) or their variations A^* or D^*. On the other hand, unlike previous methods where a network graph is constructed, a *Potential Field* method is based on a different idea (Figure 1.9d), as it suggests that a robot moves under the influence of attractions (to points of interest such as the target position) and repulsions (from obstacles). Therefore, a potential field method does not need a graph search to return a path but instead guides the robot through the workspace in the continuous world.

Roadmap method

A roadmap is a network of curves that are in $\mathcal{C}_{free}$ and is defined as $\mathcal{R}$. It connects initial q_i and a goal q_g configurations to $\mathcal{R}$. Therefore, a roadmap satisfies two properties: accessibility (a free path can be computed from any $q \in \mathcal{C}_{free}$ to $\mathcal{R}$) and connectivity (if there is a free path between q_i and q_g, there is also a free path between $r(q_i)$ and $r(q_g)$, where $r(q) \in \mathcal{R}$) [18]. The Roadmap Approach (RA) can be constructed using different methods such as Visibility Graph (VG) [45], Voronoi Diagram (VD) [46], and Probabilistic Roadmap (PRM) [47].

A continuous path planning approach using VD has been applied in robotics and manufacturing [48]. This method efficiently represents the workspace and generates optimized paths that avoid obstacles. Tests in various scenarios showed that this approach improves route planning by reducing calculation time and optimizing route length compared to traditional methods.

Furthermore, an enhancement to the PRM algorithm for mobile robot path planning is presented in [49]. This modification includes a pseudo-random sampling strategy and optimized collision detection. As a result, tests in static environments demonstrated significant improvements in path length, execution time, and collision risk reduction.

Cell Decomposition Method

The Cell Decomposition (CD) method divides the configuration space $\mathcal{C}_{free}$ into smaller, non-overlapping portions called cells, ensuring that a path can always be found between any two robot configurations within a cell. By searching for a path from the cell containing the start configuration (q_i) to the cell containing the goal configuration (q_g), a sequence of adjacent cells, or channel, is formed [28]. Two well-known CD categories are Exact Cell Decomposition (ECD) [50], and Approximate Cell Decomposition (ACD) [51].

An innovative route planning method for curvilinear obstacles is presented in [52]. This technique combines precise cellular decomposition with an enlarged polygonal approximation using the Douglas–Peucker algorithm, enhancing route development efficiency and providing a realistic portrayal of curved barriers. Simulations in various scenarios showed that this method offers more effective and efficient pathways in the presence of curved obstructions, outperforming conventional techniques.

Similarly, [53] presents a Radial Cell Decomposition (RCD) algorithm for mobile robot path planning. This algorithm divides the workspace into radial cells, facilitating easier navigation and more efficient obstacle avoidance. Testing in simulated environments demonstrated that the RCD algorithm improves route planning efficiency and accuracy, providing optimal routes with shorter calculation times compared to traditional CD.

Sampling-based method

Representing the configuration space using a roadmap or exact cell decomposition requires an explicit representation of $\mathcal{C}_{free}$, which can be computationally intensive as the configuration space increases. Therefore, sampling-based methods reduce complexity by demonstrating the connectivity of $\mathcal{C}_{free}$ without explicitly representing $\mathcal{C}$. The most widely used sampling-based method is Rapidly-exploring Random Tree (RRT) [54].

Recent works aim to enhance the traditional RRT and its variants. For example, [55] presents an adaptive version of the RRT algorithm for path planning, improving route generation and search efficiency. Tested in static environments, this adaptive RRT outperforms the conventional version in computation time and route length, finding optimal solutions under various conditions.

Similarly, [56] introduces the F-RRT* algorithm, which incorporates an improved initial solution and optimizes the convergence rate for more efficient routes. Tested in static environments, the F-RRT* algorithm shows better route quality and significantly reduces computation time compared to traditional RRT* algorithms, effectively finding optimal solutions in different complex scenarios.

Potential field method

The Artificial Potential Field (APF) method is based on attractive and repulsive forces exerted on the AMR. The goal point applies an attraction force ($\vec{F}_{attr}$) while obstacles apply a repulsive force ($\vec{F}_{rep}$) [57].

An improved path planning method integrating decision trees with APF is presented in [58]. This hybrid method leverages APF's obstacle avoidance capabilities and decision trees' informed decision-making. Tested in both static and dynamic environments, the method provides safer and more efficient routes, overcoming traditional APF limitations by improving obstacle avoidance and reducing calculation time.

Additionally, [59] introduces the QAPF learning algorithm for path planning in both known and unknown environments. This algorithm combines Q-learning with APF to enhance learning speed and performance. Compared to classical APF, QAPF demonstrates superior results in path length, smoothness, and learning time for both offline and online path planning.

1.3.2. Heuristic Path Planning

Heuristic approaches have outperformed conventional methods and gained popularity due to their success in addressing issues involving multidimensionality, complex workspaces, and local minima [60]. Unlike classical algorithms, heuristic algorithms may not guarantee an optimal solution but aim for a sufficiently good solution in less time. This makes them more practical in real-world situations, as they can adapt to changing environments with uncertain and incomplete information. As shown in Figure 1.8, for this review, heuristic-based algorithms can be classified into three categories: *Fuzzy Logic System Methods*, *Learning Methods*, and *Nature-Inspired Methods*.

Fuzzy Logic System Methods utilize fuzzy logic to handle uncertainty and imprecision in decision-making, allowing for degrees of truth between 0 and 1, making them suitable for dynamic environments with uncertain conditions [61]. Furthermore, Learning Methods, including machine learning and deep learning techniques, enable systems to adapt and improve with experience, particularly useful for problems like path planning, where optimal solutions may not be obvious, by training predictive models with data [62]. Lastly, Nature-Inspired Methods, also known as metaheuristic algorithms, draw inspiration from natural processes such as genetics, swarm behavior, and evolution. These methods explore a broad search space to find the global optimum, using stochastic mechanisms to balance exploration and exploitation, thus avoiding local optima [63, 64].

Fuzzy Logic System Method

The Fuzzy Logic (FL) path planning method utilizes fuzzy rules to manage uncertainty and imprecision in navigation decision-making, first proposed by Zadeh in 1965 [65]. This approach evaluates parameters like distance, angle, and obstacles to generate efficient and adaptive routes in dynamic environments. In AMR navigation, FL can not only plan the path but also explicitly consider motion control planning since the robot's kinematic constraints are typically addressed during the design phase.

An article by [66] introduces a fuzzy rule-based path-planning algorithm for mobile robots in complex terrain. It combines FL and DA, dividing the planning process into three stages: The algorithm identifies edge points, uses DA for road map sorting, and uses a FL system to search road signs. Simulation results show it can solve complex environments, reduce planning time, and provide more flexible turning angles.

Researchers have also explored techniques to optimize FL parameters using various metaheuristic approaches to enhance robot navigation. In [67], an Ant Colony Optimization with Fuzzy Logic (ACO-FL) is proposed for local path planning in Unmanned Surface Vehicles (USVs), considering wind, current, wave, and dynamic obstacles. The strategy aims to optimize the USV route, considering energy efficiency, obstacle avoidance, and navigation safety. Similarly, [68] describes a hybrid method for UAVs path planning in multi-objective missions, combining fuzzy logic and genetic algorithms. This approach optimizes multiple objectives, considering environmental changes and constraints, such as route safety and flight duration reduction. Results showed significant enhancement in UAVs' ability to design effective and safe routes while adapting to diverse situations and mission requirements through the combination of evolutionary algorithms and fuzzy logic.

Neural Networks (NNs)

Since its inception in 1944, NNs have found wide application in engineering [69, 70]. An NN consists of distributed parallel processing units (neurons) organized in a graph structure. The neural network-based path planning method is trained with sensor information, such as distance from obstacles and heading angle, to predict and generate optimal paths in real time, adapting to changing and complex environments. The outputs represent the velocities of wheels and/or the steering angle applied for the subsequent motion [71, 72].

Recent research has shown a growing interest in integrating NN and FL methods to form a neuro-fuzzy control system to enhance robot navigation. In [73], authors developed a neuro-fuzzy controller based on artificial potential fields for autonomous mobile robot navigation. This approach combines potential fields for obstacle avoidance and neuro-fuzzy control for adaptive decision-making, tested in both static and dynamic environments, considering robot kinematics. Experimental results showed that the proposed controller improves robot autonomy in complex environments, ensuring precise and efficient obstacle avoidance.

Similarly, an adaptive neuro-fuzzy inference system (ANFIS) is employed in [74], combining fuzzy logic and neural networks to enhance the robot's decision-making capacity and adaptability in challenging scenarios. Experiments in static and dynamic contexts evaluated the methodology's effectiveness, demonstrating enhanced route planning accuracy and efficiency. This approach enables robots to navigate securely and successfully in various situations.

Reinforcement Learning (RL)

RL is a machine learning paradigm where agents learn to make decisions by interacting with their environment. Introduced by Watkins in 1989 [75], RL aims to maximize cumulative rewards over time through exploration and exploitation.

In [76], an Efficient Q-learning (EQL) algorithm is proposed for mobile robotics, overcoming slow convergence by improving reward functions and selection strategies. EQL initializes the Q-table with prior knowledge and employs an efficient selection strategy for accelerated learning and convergence. EQL demonstrates enhanced learning proficiency and training performance compared to existing Q-learning-based algorithms.

Similarly, [77] presents a dynamic obstacle avoidance method for manipulators using the Soft Actor-Critic (SAC) Deep Reinforcement Learning (DRL) algorithm. This method incorporates a reward function for obstacle avoidance and a targeted approach for real-time planning, addressing low sample utilization issues through prioritized experience replay. Simulation experiments confirm the effectiveness of this method in obstacle avoidance and planning tasks, achieving high success rates.

Genetic Algorithm (GA)

The Genetic Algorithm (GA), developed by Holland in 1992 [78], is an optimization method that mimics natural evolution principles like selection, crossover, and mutation to find optimal solutions. In path planning, this approach initiates an initial population of potential solutions and evolves them iteratively to enhance route quality, adapting to dynamic and complex environments.

In [79], an improved adaptive evolutionary technique with collision detection is proposed. This technique modifies route solutions based on collision detection, enhancing the robot's navigation safety and efficacy. Tested and simulated in static scenarios, the adaptive GA with collision detection outperforms conventional techniques, enabling precise and reliable navigation.

Moreover, [80] presents a novel knowledge-based genetic algorithm for mobile robot path planning in complex, unstructured environments. This system combines local search techniques with domain knowledge of robot route planning to create specialized operators. It offers a clear depiction of the robot's path and an effective means of assessing the path's quality by accurately identifying collisions. Demonstrating the importance of specialized genetic operators, the algorithm finds nearly optimal pathways in both static and dynamic complex environments.

Particle Swarm Optimization (PSO)

PSO is a nature-inspired metaheuristic algorithm developed by Eberhart and Kennedy in 1995 [81]. It mimics the social behavior of animals like fish schools and bird flocks to find optimal solutions. In path planning, particles represent potential paths and navigate through the solution space influenced by their best-known local and global positions. This iterative process enables the swarm to converge efficiently towards optimal routes.

Traditional particle swarm algorithms often encounter issues such as long paths, poor global search, and local development. Addressing these concerns, in [82], an Improved PSO scheme for mobile robot path planning, known as IPSO, is proposed. It incorporates uniform distribution, exponential attenuation inertia weight, cubic spline interpolation function, and enhanced control learning factor. IPSO achieves superior optimal results with fewer iteration steps compared to existing algorithms, reducing both path length and simulation time.

Moreover, [83] introduces an artificial potential field-based particle swarm algorithm (apfrPSO). This algorithm generates robot planning paths by adjusting the inertia weight parameter and ranking the position vector of particles. Comparative numerical experiments demonstrate that apfrPSO is highly competitive, offering improved environmental adaptation, path-planning accuracy, and efficiency in complex environments.

Ant Colony Optimization (ACO)

The ACO method draws inspiration from the foraging behavior of ants, where they deposit pheromones to mark efficient paths, aiding other ants in finding the shortest or most effective routes. Introduced by Dorigo and Gambardella in 1997 [84], this algorithm mimics this natural behavior.

In [85], authors proposed an Improved Adaptive Ant Colony Algorithm (IAACO) for path planning of indoor mobile robots, addressing limitations of conventional ACO methods. IAACO introduces angular guidance and obstacle exclusion factors, adaptive adjustment of heuristic information, and pheromone volatilization to enhance convergence and global search. Tested in a static environment, IAACO demonstrates high real-time performance, stability, and solution diversity in producing global optimal routes.

Additionally, in the article [86], an improved approach for robot path planning utilizing ACO in the context of the Internet of Things (IoT) is presented. This approach optimizes robot routes by leveraging connectivity and data from IoT devices, enabling more efficient and adaptive navigation strategies.

Cuckoo Search Algorithm (CSA)

The CSA draws inspiration from the breeding behavior of cuckoos and the nest selection mechanism of certain bird species. Yang and Deb introduced CSA in 2009 to address nonlinear optimization problems [87]. In path planning, CSA employs a combination of random search and intensive exploitation strategies on the best nests (solutions) to efficiently find optimal routes.

In [88], a modified CSA utilizing a tournament selection function for robot path planning is proposed. This modification addresses issues such as local minima by replacing random selection with a tournament selection function. By considering path length and time as parameters, the proposed method outperforms conventional approaches, producing better output in terms of path length and time.

Additionally, [89] presents a hybrid algorithm combining modified CSA, sine-cosine algorithm, and PSO for optimal path computation with collision avoidance. This algorithm integrates efficient global and local search strategies with a greedy approach, demonstrating effectiveness in achieving minimal time, shortest distance, collision avoidance, path smoothness, synchronized action, and reduced energy usage.

Firefly Algorithm (FA)

The FA, introduced by Yang in 2009 [90], finds inspiration from the behavior of fireflies, which utilize light intensity to attract others. Each firefly in the algorithm represents a potential solution, with its attractiveness determined by the quality of the solution (light intensity). Fireflies collectively converge towards more attractive solutions to discover efficient routes.

While FA is commonly used in path planning, it can sometimes get trapped in local optimal solutions. To address this limitation, [91] proposes a hybrid algorithm combining FA with GA. This approach aggregates local optimal fireflies into groups and subjects them to GA operations (selection, crossover, and mutation). Then, the accuracy and performance of FA are enhanced, providing more reliable solutions.

Additionally, in [92], an FA-based path planning algorithm for mobile robots is presented. This algorithm efficiently identifies obstacle-free trajectories, prioritizing path length optimization and safety assurance. Through rigorous testing on reference trajectories, it surpasses alternative algorithms, demonstrating superior trajectory planning capabilities and robust tracking performance.

Artificial Bee Colony (ABC)

The ABC algorithm, pioneered by Karaboga in 2005 [93], emulates the foraging behavior of bee colonies. In this method, scout bees explore new solutions (food sources) while employee bees refine existing solutions, exchanging information through a dance communication process. Through cooperation and adaptability, this approach enables the discovery of optimal routes.
In [94], a hybrid path planning approach integrating an offline global path optimization algorithm with online path planning schemes is proposed. Initially, common worker and obstacle positions are defined, followed by global optimization employing the ABC algorithm. The resulting shortest path is then selected using DA. This methodology ensures the generation of satisfactory paths with minimal computational effort.
Despite being a widely used optimization tool, the ABC algorithm faces limitations due to its one-dimensional search strategy. To address this, [95] introduces an RL-based variant, ABC-RL. By adjusting the number of dimensions in the bee phase search equation and implementing two enhanced search strategies, ABC-RL demonstrates superior performance compared to other ABC variants in terms of solution accuracy and robot path planning.

Bats Algorithm (BA)

Inspired by bats' swift ultrasonic signaling, the BA was developed by Xin-She Yang in 2010 to find global optimal solutions [96]. In BA, potential solutions, acting as bats, adjust their pulse frequency and echo emission rates to navigate the search space efficiently.
The BA, known for its slow convergence and low precision, has been enhanced for mobile robot path planning by integrating the dynamic window approach in [97]. Simulation results indicate that this modified bat algorithm surpasses both PSO and basic implementations of BA in terms of achieving optimal solutions.
Furthermore, [98] proposes a reformed variant known as the Reformative Bat Algorithm (RBA) for mobile robot path planning. RBA leverages the Doppler effect for frequency updates and incorporates adaptive compensation mechanisms to mitigate premature convergence. Additionally, the integration of Q-learning enhances RBA's local search capabilities, resulting in more robust planning of optimal paths within shorter time frames. Real-world navigation experiments highlight RBA's satisfactory real-time performance and effectiveness in path planning, positioning it as a valuable tool for addressing path-planning challenges.

Fruit Fly Optimization (FFO)

The FFO algorithm, initially introduced by Pan in 2012 [99], draws inspiration from the foraging behavior of fruit flies. This approach leverages the superior olfactory and visual senses of fruit flies to locate food sources, which represent optimal solutions. Through iterative updates to the positions of the flies, the algorithm collaboratively enhances the quality of solutions.

In an effort to enhance the global convergence speed and mitigate local optima, [100] introduces a multiple swarm fruit FFO algorithm (MSFFOA). This variant divides the fruit fly swarm into sub-swarms with multitasks, thereby expanding the search space and improving search capabilities. Simulation results demonstrate that the proposed MSFOA surpasses the original FFO algorithm in terms of convergence and accuracy.

Addressing the challenge of automatic recharging path planning for cleaning robots in complex industrial environments, [101] presents an improved version of the FFO algorithm (IFFOA). Initially, DA is employed to plan the global path and reduce the two-dimensional path planning to one dimension. Subsequently, IFFOA is applied to minimize the path length. Simulation experiments validate the effectiveness of these methods, showcasing their efficiency in terms of computation, precision, and convergence speed.

Grey Wolf Optimization (GWO)

The GWO algorithm, devised by Mirjalili et al. in 2014 [102], is inspired by the leadership structure and hunting mechanisms of grey wolves. This method categorizes solutions into alpha, beta, delta, and omega, with the first three guiding the search process. By iteratively updating the positions of the wolves, the algorithm converges toward optimal solutions.

In [103], the GWO algorithm is employed to enable rapid and global path planning in complex environments with static obstacles. A comparative study was conducted, and simulation results demonstrated that the GWO algorithm efficiently generates optimal path plans in terms of path distance and execution time.

Furthermore, [104] introduces a multi-objective path planning algorithm comprising three steps. Initially, a path is optimized using a novel GWO-PSO algorithm, minimizing path distance and smoothing it. Subsequently, infeasible points are transformed into feasible solutions. Finally, collision avoidance algorithms are applied to identify obstacles. Simulation tests across various environments validate the algorithm's feasibility, demonstrating its ability to overcome the shortcomings of conventional techniques.

Whale Optimization Algorithm (WOA)

The WOA algorithm, introduced by Mirjalili and Lewis in 2016 to address optimization problems, draws inspiration from the social behavior of humpback whales [105]. In this method, potential solutions (represented as whales) explore the search space by adjusting their position and speed to find optimal routes. This approach is based on the concept that whales follow patterns of migration and exploration to locate prey.

The WOA encounters challenges such as slow convergence speed, susceptibility to local optima, and lack of dynamic obstacle avoidance. To address these limitations, a novel whale optimization algorithm, NWOA [106], has been proposed. NWOA incorporates adaptive technology to enhance convergence speed and introduces virtual obstacles to evade local optima traps. Compared to WOA variants, NWOA demonstrates faster convergence speed and superior dynamic planning effects, resulting in reduced average fitness values and shorter planning times and path lengths.

Additionally, a hybrid firefly-whale optimization algorithm (FWOA) is presented in [107], intending to enhance solution accuracy in complex mobile robot working environments. FWOA efficiently identifies optimal paths while striking a balance between exploitation and exploration. Comparative results demonstrate its exceptional performance in terms of convergence speed and exploration capability when compared to ten other classical metaheuristic algorithms.

1.3.3. Analysis and Discussion

Table 1.2 presents a summary of the related works described above. This review includes 35 of the most relevant works on path planning from the past five years, considering the methods identified in Figure 1.8. The table presents the analysis of these works based on several criteria: (i) method, categorizing them as classical, fuzzy logic, learning, metaheuristic (nature-inspired), or hybrid methods; (ii) planning, distinguishing between global (if the method uses complete information about the environment) and local (if it uses partial information); (iii) time, indicating whether the planning was done online or offline; (iv) algorithms, specifying which ones were used; (v) environment, noting whether the method was tested in a completely static environment or a dynamic one (with moving objects); (vi) kinematics, considering whether the kinematic constraints of the robot for motion control and path following were included in the proposal; and (vii) evalaution, determining if the method was tested in a basic simulation (BS) with only a graphical representation of the environment and the robot as a point, in a realistic simulation (RS) within a software package that simulates the real robot, or in a real environment (RE).

Table 1.2: Analysis of the reviewed AMR Path Planning Techniques based on (i) method (classical, learning, metaheuristic, or hybrid); (ii) planning (global or local); (iii) time (online or offline); (iv) algorithms; (v) environment (static or dynamic); (vi) kinematics; (vii) evaluation type which is either basic simulation (BS), realistic simulation (RS), or real environment (RE)

Article	Year	Method	Planning	Time	Algorithms	Environment	Kinematics	Evaluation
[48]	2019	Classical	Global	Offline	VD	Static	No	BS
[49]	2023	Classical	Global	Offline	PRM	Static	No	BS
[52]	2019	Classical	Global	Offline	CD	Static	No	BS
[53]	2021	Classical	Global	Offline	CD (RCD)	Static	No	BS
[55]	2021	Classical	Global	Offline	RRT	Static	No	BS
[56]	2021	Classical	Global	Offline	RRT (F-RRT*)	Static	No	BS
[58]	2020	Classical	Global	Offline	APF	Static	No	BS
[59]	2022	Hybrid	Both	Both	APF, Q-Learning	Dynamic	No	BS
[66]	2022	Hybrid	Both	Both	FL, DA	Static	No	BS
[67]	2021	Hybrid	Both	Both	FL, ACO	Dynamic	No	BS
[68]	2022	Hybrid	Global	Offline	FL, GA	Static	No	BS
[73]	2021	Hybrid	Local	Online	NN, FL, APF	Dynamic	Yes	RE
[74]	2022	Hybrid	Local	Online	NN, FL, (ANFIS)	Static	Yes	RS
[76]	2020	Learning	Global	Offline	Q-learning (EQL)	Static	No	BS
[77]	2022	Learning	Global	Offline	SAC DRL	Dynamic	Yes	RS
[79]	2021	Metaheuristics	Global	Offline	GA	Static	No	BS
[80]	2022	Metaheuristics	Global	Both	GA	Dynamic	No	BS
[82]	2020	Metaheuristics	Global	Offline	PSO (IPSO)	Static	No	BS
[83]	2023	Hybrid	Global	Offline	PSO, APF, (apfrPSO)	Static	No	BS
[85]	2021	Metaheuristics	Global	Offline	ACO (IAACO)	Static	No	BS
[86]	2022	Metaheuristics	Global	Offline	ACO	Static	No	BS
[88]	2021	Metaheuristics	Global	Offline	CSA	Static	No	BS
[89]	2023	Hybrid	Local	Online	CSA, PSO	Dynamic	No	BS
[91]	2022	Hybrid	Global	Offline	FA, GA	Static	No	BS
[92]	2024	Metaheuristics	Global	Offline	FA	Static	Yes	BS
[94]	2021	Hybrid	Global	Both	ABC, DA	Dynamic	No	BS
[95]	2022	Hybrid	Global	Offline	ABC, RL	Static	No	BS
[97]	2021	Hybrid	Global	Both	BA, DA	Dynamic	Yes	BS
[98]	2022	Hybrid	Global	Offline	BA, Q-learning (RBA)	Static	Yes	RE
[100]	2020	Metaheuristics	Global	Offline	FFO (MSFFOA)	Static	No	BS
[101]	2021	Hybrid	Global	Offline	FFO, DA (IFFOA)	Static	No	BS
[103]	2020	Metaheuristics	Global	Offline	GWO	Static	No	BS
[104]	2021	Hybrid	Both	Both	GWO, PSO	Static	No	BS
[106]	2023	Metaheuristics	Global	Both	WOA (NWOA)	Dynamic	No	BS
[107]	2024	Hybrid	Global	Offline	WOA, FA (FWOA)	Static	No	BS

It's evident that there's a significant focus on improving existing algorithms to address their limitations, either through techniques that refine the algorithms themselves [53, 56, 82, 85, 100, 106] or by combining two methods to capitalize on their respective strengths [59, 66, 67, 68, 73, 74, 83, 94, 95, 97, 98, 101]. Furthermore, the fusion of nature-inspired algorithms has been explored to overcome individual limitations and enhance overall capabilities [89, 91, 104, 107]. In particular, classical methods are no longer as effective when addressing the path planning problem independently; however, they have found their role as valuable complements to heuristic algorithms [59, 66, 73, 83, 94, 97, 101].

It's also worth noting that implementations with FL are quite common, and in all cases, it has been combined with another method [66, 67, 68, 73, 74]. This is because FL is closely related to the actuators of robot movement but lacks a way to optimize itself. Therefore, the combination of FL with an algorithm that optimizes the route effectively solves both problems: path planning and motion control.

A large majority of the works presented utilize global planning in their methodology, meaning they have complete information about the environment, thus disregarding the local information that the robot could directly provide. In fact, only 20% of the consulted works incorporate local planning [59, 66, 67, 73, 74, 89, 104]. However, it can also be seen that the lack of local planning is replaced by online global planning, which was not traditionally done. The improvements in computation time brought about by heuristic algorithms offer the opportunity to re-plan over the entire environment at the same time as the robot moves in real-time and mobile obstacles appear [80, 94, 97].

Another aspect worth mentioning is the scenario in which the evaluation and validation of the method are performed. The vast majority still utilize static environments, which do not represent the conditions of a real implementation scenario. Only 9 out of 35 papers (25.7%) tested their proposal in a dynamic environment [59, 67, 73, 77, 80, 89, 94, 97, 106], of which only two did so in an environment with realistic conditions such as realistic simulation or a real environment [73, 77]. Two more proposals are tested in realistic conditions but in static scenarios [74, 98]. Moreover, the papers that considered the robot's kinematics and its motion control for path following were 6 out of 35 (17.1%) [73, 74, 77, 92, 97, 98]. Among the articles reviewed, only one considered a dynamic environment and implemented motion control in a realistic setting using a neuro-fuzzy approach [73]. However, despite offering a pragmatic solution, the resources and costs required to train, test, and validate this method across different scenarios are significantly higher compared to other approaches.

1.3.4. Path Planning Comprehensive Overview

Because the conducted review was not highly rigorous, the presented research could be biased. Therefore, the results and perspectives of four survey reviews presented in 2023 are also discussed. These surveys provide a comprehensive and exhaustive overview, describing the current state of the art in solutions to the problem of path planning.

In [18], a comprehensive and up-to-date literature review is presented on AMR route planning strategies that have garnered considerable attention over the past decade. This review compiles information from more than 200 articles, providing an extensive overview of the latest advancements and trends in the field. According to the review, at least 75% of the published work in the last 10 years has employed heuristic methods, suggesting their superiority over classical ones. However, only 22.5% of these works have addressed the problem of dynamic environments by combining heuristic and classical methods, and merely 20% have focused on real-time local planning (online). Additionally, the article highlights that kinematics (dynamic constraints) have received the least attention due to their relative complexity. Nonetheless, these considerations are crucial in navigation as they represent the practical motion limitations of the robot, influencing key performance metrics such as path length, smoothness, local collision avoidance ratio, and real-time execution.

In the article [60], the authors present an overview of path-planning strategies for mobile robots, analyzing nearly 150 articles. Their review reveals a predominant use of heuristic approaches over classical ones, with 95% of the works employing heuristic methods by 2022, compared to only 5% using classical approaches exclusively. This finding is consistent with previous research indicating the underutilization of classical algorithms. Specifically, 82% of the reviewed works focus on path planning in environments with fixed obstacles, while only 18% addresses scenarios involving moving obstacles. Interestingly, approximately 33% of the articles consider the kinematic constraints of robots for motion control and path following, which, although higher than previously observed, remains relatively low given its critical role in implementing a comprehensive navigation system.

Among the key conclusions highlighted in the article [108], a comprehensive review and prospective analysis of path planning techniques for mobile robots, it is noted that approximately 46% of the 150 reviewed articles employ local planning. However, this percentage is still significantly lower than the 80% utilizing global path planning. Additionally, among the nature-inspired algorithms discussed in the reviewed articles, the most commonly used are ACO (22.5%), GA (16%), and PSO (12.5%), which are also among the oldest. Conversely, the least utilized algorithms happen to be the most recent ones, such as FA (6%), GWO (4%), or WOA (4%). This observation aligns with findings presented in article [18], where similar trends in nature-inspired path-planning techniques for mobile robots were identified.

Finally, article [109] presents a survey of 46 articles, highlighting that more than 66% of the works are based on heuristic algorithms. The survey reviews the most and least utilized performance metrics across different path planning approaches in the literature: time, distance, energy, safety, and smooth path. Time (44.2%) and distance (42.3%) emerge as the most commonly considered metrics, whereas energy (2%), safety (5.8%), and smooth path (5.8%) are relatively neglected.

The general overview of the current state of the path planning problem, based on the works studied in this research (presented in Table1.1) and considering the results from the four systematic reviews, includes the following points:

- Heuristic methods have become increasingly popular over the past ten years, indicating their superiority over conventional approaches.
- Dynamic environments represent real-world situations. However, it hasn't received as much attention as planning in static environments.
- In the absence of dynamic environments, real-time applications are not the main focus of modern publications.
- Local planning has not been utilized as extensively, missing the opportunity to gather real-time information and perform reactive control.
- Instead of employing it in realistic mobile robot situations, the experiment verification method used in the majority of publications is a non-realistic computer simulation.
- There is a clear trend towards improving existing methods, particularly through fusing different algorithms.
- Including the AMR's dynamic and kinematic constraints in the cost function could complicate the path-planning approach. However, since it captures the robot's actual mobility constraints, which ultimately affect the AMR's overall performance in practice, it represents a valuable line of research.

These paradigms guide the perspective of this research. With the aim of developing a comprehensive navigation strategy capable of handling realistic dynamic scenarios, the integration of global path planning with local path planning is sought. While global path planning provides an initial plan, navigating mobile robots in diverse environments (such as military, agriculture,

and service sectors with rigorous real-time requirements) requires consideration of dynamic factors like robot state and obstacle characteristics. Therefore, continuous interaction between the mobile robot and its environment is crucial to derive optimal local routes. These local routes collectively form the global route, which is dynamically adjusted based on real-time information using feedback control.

However, exclusively focusing on path planning may ignore issues regarding real-world implementation. Therefore, a complete navigation system must consider the motion control planning for path following to enhance its robustness. This involves taking a planned geometric path and incorporating timing information to create a trajectory, optimizing the robot's dynamic capabilities and motion constraints to improve the navigation technique. Current methods often oversimplify path planning by assuming constant robot velocity, whereas real-world scenarios may involve variable speeds and acceleration. Furthermore, sensor perception presents challenges as it is frequently separated from planning. Effectively integrating sensors such as cameras and LiDARs can mitigate environmental uncertainties but necessitates efficient utilization of sensor feedback. Hence, integrating motion planning that addresses both locomotion and perception challenges is crucial for advancing navigation strategies.

1.4. Motion Planning

As introduced earlier in this chapter, autonomously navigating an AMR involves two primary tasks: path planning and motion planning. Path planning, extensively discussed in preceding sections, is a focal point for researchers studying autonomous robotic navigation. It entails finding an obstacle-free route that connects two points within a configuration space. In contrast, motion planning or motion control planning addresses the actual execution of the robot's movement, often overlooked but crucial for guiding the robot's actuators to its intended destination in a real implementation.

In the context of this work, motion planning specifically refers to *path following* task. This process is essential for AMRs, involving guiding a robot along a predefined path (identified through path planning) while adhering to a path-dependent speed profile [110]. This profile dictates the sequence of actions the robot must execute, considering its kinematic constraints, to reach its destination. Figure 1.7 illustrates the anticipated operation that ensures adaptive and efficient navigation in dynamic environments as follows:

1. The robot moves purposefully along a global route.
2. When encountering an unexpected obstacle, it adjusts its path locally to avoid the collision.
3. Once the obstacle has been overcome, the robot resumes its journey along the original global route.

This navigation strategy thus involves two operational modes: deliberate motion control along a global route and reactive motion control along a local route when encountering unknown dynamic obstacles. This framework considers two key aspects: how the robot will move (addressed by its kinematic system) and how it will detect new obstacles (handled by its perception system).

1.4.1. Kinematics System Overview

Kinematics is fundamental to understanding the mechanical behavior of mobile robots. It provides insights necessary for designing suitable robots for specific tasks and developing control software optimized for the robot's hardware configuration [27]. The process of understanding the motions of a robot begins with analyzing the types of locomotion mechanisms the robot employs.

The locomotion system of a mobile robot plays a crucial role in facilitating the robot's movement and navigation in its environment. In [111] is noted that there are several types of locomotion systems used in mobile robots, each with its advantages and disadvantages. The selection of a particular type of locomotion system is dependent on the specific application and requirements of the robot, which are determined by various factors such as the tasks that the robot will perform, the terrain and environmental conditions that it will encounter, the available power sources, and the cost and complexity of the design. The main classification of mobile robots is based on how and where they move and can include the following major categories: *land-based; air-based; water-based;* and *others*, as can be seen in the diagram shown in Figure 1.10.

Land-based robots are crucial for tasks that are difficult, dangerous, or impractical for humans, such as industrial automation, logistics, security, and surveillance [112]. They can be categorized by their locomotion systems: wheeled, tracked, and legged. Wheeled robots excel on smooth surfaces and offer high maneuverability but struggle with obstacles. Tracked robots handle uneven terrains and heavy loads well but require more power and maintenance. Legged robots can traverse challenging terrains but face issues with control and stability. Hybrid robots combine these locomotion systems for greater adaptability and versatility, making them suitable for a variety of environments and tasks.

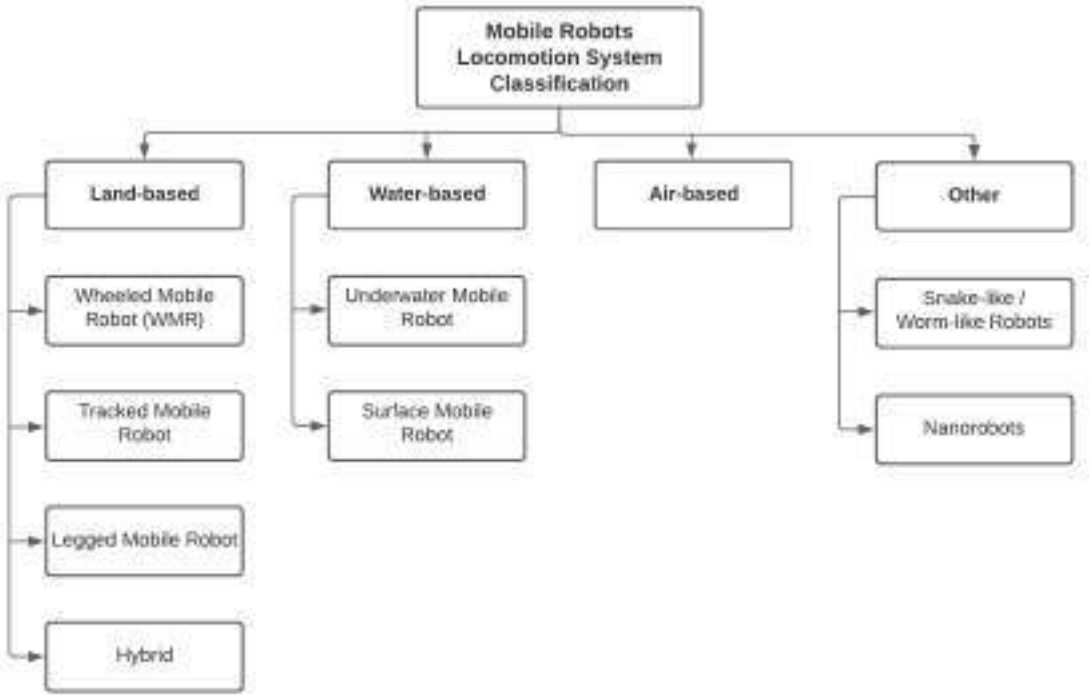


Figure 1.10: Classification for Mobile Robot Locomotion Systems

Water-based robots can operate either underwater or on the surface of the water, depending on their design and intended application. Underwater robots are designed to operate below the surface and can travel to great depths to perform tasks such as underwater exploration, mapping of the ocean floor, and marine biology research. On the other hand, surface robots are designed to operate on top of the water and are used for applications such as environmental monitoring, water quality sampling, and search and rescue operations [113].

Air-based robots, commonly referred to as drones, have proven to be valuable and versatile tools for businesses and organizations across various sectors [114]. With the capability of collecting high-quality data and images from the air, drones have transformed traditional operations in industries such as agriculture, construction, mining, and transportation. Notably, their ability to operate in dangerous or difficult-to-access areas, such as disaster zones and conflict areas, offers cost-effective data collection compared to conventional techniques [115].

The wheel has been by far the most popular locomotion mechanism in mobile robotics and man-made vehicles in general. It can achieve very good efficiencies and does so with a relatively simple mechanical implementation [27]. For this dissertation, a land-based WMR will be used as the primary platform for research and experimentation due to its straightforward physical implementation. WMRs requires minimal maintenance compared to other types of mobile robots

such as legged or tracked robots. They are also more affordable and have lower power consumption, making them a practical choice for research and experimentation. Furthermore, wheeled robots have a simple and intuitive control system, enabling researchers to focus on developing and implementing advanced algorithms for navigation, mapping, and other applications. By using a wheeled mobile robot, the research and experimentation process can be streamlined, allowing the focus to be on developing and testing advanced algorithms and control strategies for mobile robotics applications. The kinematic model of a WMR and the main considerations for the path following control system for this type of robot are described below.

Kinematics Modeling

Unlike ordinary holonomic mobile robots that can move freely in a plane, nonholonomic mobile robots belong to a category of mobile robots with limited mobility. Holonomic robots, equipped with omnidirectional wheels [116] or the ability to fly [117], can move instantly in any direction within their operational area without any constraints. In contrast, nonholonomic mobile robots achieve steering by independently regulating the speed of the wheels on either side of the vehicle [118]. When the wheel speeds on either side are not equal, the vehicle will turn (such as differential drive robots [119] or car-like robots [120]). For this research work, a basic differential steering robot is considered, which typically has two driving wheels and front and/or rear casters for increased stability.

The initial step is to determine the robot's position on the plane. A relationship is established between the plane's global reference frame and the robot's local reference frame, as illustrated in Figure 1.11. The axes X_I and Y_I define an arbitrary inertial basis on the plane, serving as the global reference frame from the origin $O : \{X_I, Y_I\}$. To specify the robot's position, a point P on the robot chassis is chosen as its position reference point. The basis defines two axes relative to P on the robot chassis $\{X_R, Y_R\}$, establishing the robot's local reference frame. The position of P in the global reference frame is specified by the coordinates x and y, and the angular difference between the global and local reference frames is given by the angle ψ. The pose of the robot can then be described as a vector containing these three elements (note the subscript I, indicating the global reference):

$$\xi_I = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix} \tag{1.7}$$

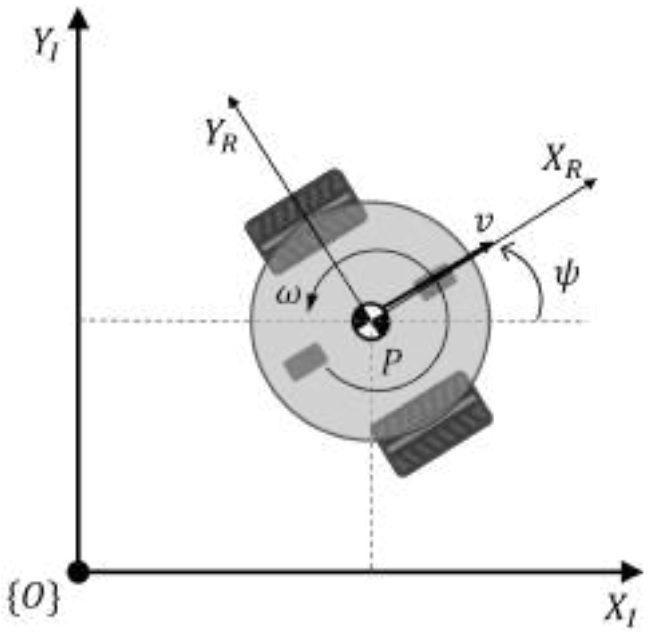


Figure 1.11: A two-wheeled independently driven nonholonomic mobile robot with the definition of the global coordinate frame $\{O\}$ and the body coordinate frame centered in P

The motion of the mobile robot is controlled by its linear velocity v and its rotational velocity ω, whose positive directions are defined in Figure 1.11. The kinematics of the mobile robot are then defined as follows [121]:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos\psi & 0 \\ \sin\psi & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u} = \begin{bmatrix} v\cos\psi \\ v\sin\psi \\ \omega \end{bmatrix} \tag{1.8}$$

where $\mathbf{u} = [v, \omega]^T \in \mathcal{U} \subset \mathbb{R}^2$ define input constraints. Path following of such a nonholonomic wheeled mobile robot involves the design of algorithms to generate reference commands for $\mathbf{u}$.

In the case of a differential wheeled robot (one wheel on each side), it moves by controlling the speeds of these wheels independently. The formula to calculate the linear velocity of the robot (v) in m/s is:

$$v = r \cdot \frac{\omega_{left} + \omega_{right}}{2} \tag{1.9}$$

where r is the radius of the robot's wheel in meters and ω_{left} and ω_{right} are the angular velocities of the left and right wheels in rad/s. The average is taken because, in a differential wheeled robot, the linear velocity of the robot is the average of the linear velocities of both wheels.

Path Following

Path following ($\mathcal{P}$) refers to the problem of making a vehicle converge to and follow a spatial trajectory while asymptotically maintaining a desired velocity profile along the path. The goal is to design a controller that ensures the mobile robot follows the parameterized reference path over an arc length [122]:

$$\mathcal{P} = \{\mathbf{p} \in \mathbb{R}^2 \mid \mathbf{p} = \mathbf{p}_r(\lambda), \forall \lambda \geq 0\} \tag{1.10}$$

For any given parameter λ in a predefined path $\mathbf{p}_r$, a local reference coordinate system $\{R\}$ centered at $\mathbf{p}_r(\lambda)$ can be defined (Figure 1.12).

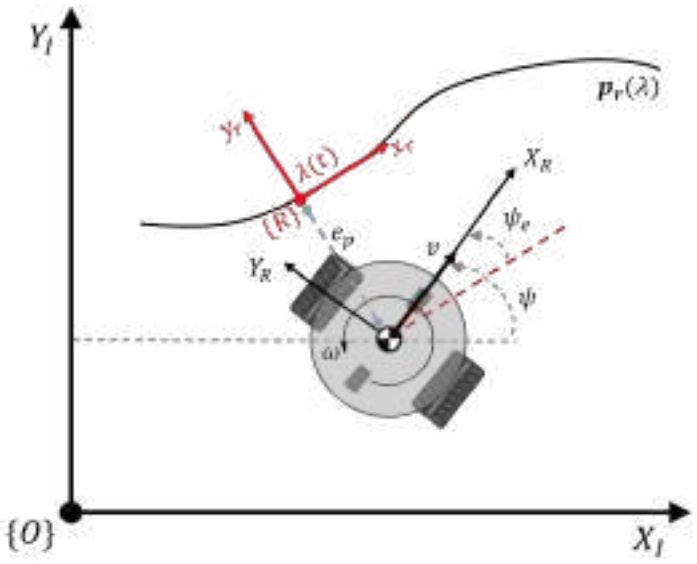


Figure 1.12: Local Reference Coordinate System in a predefined path

Considering the robot's posture at time t as $[x(t), y(t), \psi(t)]$, the cross-track error $e_p(t)$ can be defined as the perpendicular distance between the robot's position $(x(t), y(t))$ and the reference path $\mathbf{p}_r(\lambda(t))$:

$$e_p(t) = \left\| \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} - \begin{bmatrix} x_r(\lambda(t)) \\ y_r(\lambda(t)) \end{bmatrix} \right\| \tag{1.11}$$

where $(x_r(\lambda(t)), y_r(\lambda(t)))$ is the position on the reference path corresponding to $\lambda(t)$. To determine the point along the reference path for calculating the cross-track error, the point nearest to the robot is selected (this leads to an optimization problem of finding the parameter λ that minimizes the distance between the robot's position and the reference path).

Additionally, the heading or orientation error $\psi_e(t)$ between the robot and the reference path at time t is determined by:

$$\psi_e(t) = \psi(t) - \psi_r(\lambda(t)) \tag{1.12}$$

where $\psi(t)$ is the current orientation of the robot, and $\psi_r(\lambda(t))$ is the orientation of the reference path at the corresponding point $\lambda(t)$ on the path. This error indicates moving towards or away from the direction of the path.

The objective of the path-following controller is to minimize both the cross-track error $e_p(t)$ and the heading error $\psi_e(t)$ over time, ensuring that the robot converges to and accurately follows the desired path. Under ideal conditions, the path-following control system can be simplified by considering only the kinematic model, as shown in Figure 1.13 [110], computing steering commands and velocity profiles based solely on sensor feedback and the desired path.

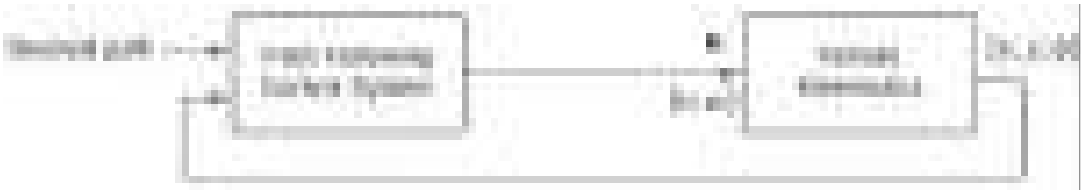


Figure 1.13: Path Following Control System: $\mathbf{u}$: linear v and rotational ω control velocities; (x, y) and ψ: the vehicle's position and orientation, respectively.

1.4.2. Perception System Overview

Robot perception is one of the core elements for achieving successful navigation. Perceiving correct information about the surrounding world efficiently enables robots to navigate safely in real-world environments [123]. Inaccurate or ineffective observation of the environment can present challenges for mobile robots, especially in tasks requiring precise object positioning [124].

As previously mentioned, the distinction between global and local path planning lies in the level of environmental information available to the mobile robot. Global planning involves a broader scope with a fully known environment. In contrast, local planning focuses on the immediate surrounding space within a short distance range to implement tasks such as collision prevention or obstacle avoidance, relying on real-time information provided by sensors. These sensors, which are part of the *perception system*, gather information about the surroundings and are essential for designing a robust motion control system.

In this context, the robot's perception system serves at least one of the following main purposes:

- To ensure safe navigation: Detect obstacles to avoid collisions.
- To model the environment: Build a map to plan the navigation path.
- To define the precise position: Determine the robot's exact position and orientation.

According to [125], the perception system of a mobile robot employs two types of sensors, namely Proprioceptive (PC) and Exteroceptive (EC) sensors, which are categorized based on the origin of the information they provide. PC sensors collect data that are internal to the robot's system, such as motor speed, wheel load, joint angles, position of the robot arm, and battery voltage. On the other hand, EC sensors obtain information from the surrounding environment, such as distance measurements, light intensity, and sound amplitude.

Sensors can also be classified as passive (P) or active (A) based on their mode of operation. Passive sensors measure the ambient energy, such as light, temperature, or sound, without emitting any energy of their own, like cameras, temperature sensors, or touch sensors. On the other hand, active sensors emit energy into the environment, such as light rays or sound waves, and then measure the resulting reaction (ultrasonic sensors or radars, for example). In Table 1.3, there is a categorization of the sensors that are highly relevant for robot applications [44, 125].

Table 1.3: Classification of Sensors used in Mobile Robotics Applications: (i) category, i.e. proprioceptive (PC) or exteroceptive (EC) and (ii) operation mode, i.e. active (A) or passive (P)

General Classification	**Sensor System**	**Category PC or EC**	**Operation Mode A or P**	**Description**
Tactile Sensors	Contact switches, bumpers	EC	P	Detection of physical contact or closeness; security switches
	Optical barriers	EC	A	
	Noncontact proximity sensors	EC	A	
Wheel/Motor Sensors	Encoders (optical, magnetic, etc.)	PC	A	Determine wheel/motor speed and position
	Potentiometers	PC	P	
Heading Sensors	Gyroscopes	PC	P	Measure the orientation of the robot with respect to a fixed reference frame
	Inclinometers	EC	A	
Ground-based Sensors	GPS	EC	A	Measure the localization of the robot with respect to a fixed reference frame
	Active optical or RF beacons	EC	A	
	Active ultrasonic beacons	EC	A	
	Reflective beacons	EC	A	
Active Ranging Sensors	Reflectivity sensors	EC	A	Detect objects using reflectivity, time-of-flight, and geometric triangulation
	Ultrasonic sensor	EC	A	
	Laser rangefinder	EC	A	
	Optical triangulation (1D)	EC	A	
	Structured light (2D)	EC	A	
Motion/Speed Sensors	Doppler radar	EC	A	Measure the velocity of the robot relative to fixed or moving objects
	Doppler sound	EC	A	
Vision-based Sensors	Camera	EC	P	Use visual ranging, whole-image analysis, segmentation, and object recognition to describe the environment
	Visual ranging packages			
	Object tracking packages			
Optical Sensors	Infrared	EC	A	Use the time required for the light to reach the target and return to estimate the distance to an object
	LiDAR	EC	A	

Common mobile robots use various sensors to learn about the environment and gather information [126]. The incoming data, originally provided by the robot's sensor as a reflected light beam or sound signal, requires the controller to perform certain computing processes that allow the robot to comprehend its surroundings from the signal's data stream. Predominantly, perception methods rely on two types of sensors: Light Detection and Ranging (LiDAR) sensors and vision sensors [108].

Vision sensors, such as cameras, are increasingly being used in various industries as a replacement for other types of sensors due to their ability to capture and analyze visual information [127]. Unlike active sensors (such as laser, ultrasonic, or infrared sensors) that can be affected by interference or noise, vision sensors can provide a clear visual representation of the environment or object being monitored, which can then be processed to extract valuable information. A vision-sensing-based movable robot with attached cameras can perform its tasks more accurately than those using other sensor-based systems [128]. Due to these advantages, the review will be done exclusively considering vision-based methods, which can be classified into three approaches: *Visual SLAM*, *Optical Flow*, and *End-to-end Learning*.

A. Visual SLAM Method

Simultaneous Localization and Mapping (SLAM) is a popular method in robot navigation, constructing a representation of the robot's environment while estimating its ego-motion, enabling robots to know their global position and location relative to surrounding objects [129]. SLAM methods can be categorized into visual SLAM (camera) and LiDAR SLAM. While the use of LiDAR offers robust 3D information with a wider field of view, cameras are smaller, lighter, and more cost-effective, making them popular for compact vehicles with payload restrictions. Although there are works that combine information from both sensors to improve the robustness and accuracy of the perception system [130, 131], in this work, we will discuss only the visual SLAM algorithm since it remains a more versatile and easy-to-implement approach for mobile robot navigation.

Generally, visual SLAM methods can be categorized into feature-based (indirect) methods or direct methods [123]. Feature-based methods extract and track feature points, requiring additional computations for feature information acquisition, while direct methods track information directly from all pixels (e.g., intensity gradient, color, etc.). MonoSLAM [132], Parallel Tracking and Mapping (PTAM) [133], and ORB-SLAM [134] are a few classical examples of feature-based

SLAM. In [135], an improved ORB-SLAM method for monocular vision robots in dynamic environments is proposed. It introduces reliability, a partial detection strategy, and a novel treatment mechanism for tracking failure issues, thereby improving localization and mapping accuracy. The article [136] proposes a novel visual SLAM method for 360-degree spherical video using the monocular PTAM approach, achieving high accuracy and stable performance in tracking and mapping undistorted patches. On the other hand, Dense Tracking And Mapping (DTAM) [137], Large-Scale Direct SLAM (LSD-SLAM) [138], and Semi-direct Visual Odometry (SVO) [139] are classical examples of direct SLAM methods. [140] proposes an improved binocular LSD-SLAM method with census transform to enhance localization accuracy and reduce data requirements. Additionally, [141] improves SVO mapping by initializing depth at feature locations based on depth prediction, resulting in reliable feature correspondence and fast convergence to true depth.

B. Optical Flow Method

There is the case where robots (due to their size or payload limitations) require more energy-efficient perception techniques than SLAM. In this regard, some researchers are inspired by animals or insects to create techniques that perform crucial and essential navigation tasks with the least amount of perceptual information possible, resulting in significant savings in memory and computational resources. One of the most recognized bio-inspired techniques is *optical flow*, first introduced by Horn and Schunck in 1981 [142]. Optical flow methods calculate the apparent spatial displacement between pixels of two consecutive images, typically associated with intensity variations in the image sequence [143].

Optical flow has been widely used in robot navigation, particularly on platforms with limited computational and payload capabilities. This visual technique is valuable for predicting robot motion relative to the environment [144], controlling or navigating robots [145], avoiding obstacles [146], and estimating distances to objects [147]. In [148] it is presented a vision-based autonomous robotics navigation system using a bio-inspired optical flow-based autonomous obstacle avoidance control approach [149] combined with a fuzzy logic controller for decision-making. The input membership functions are optimized using distributed evolutionary learning. The results demonstrate that optimizing input fuzzy membership functions significantly enhances navigation performance compared to empirical tuning. Moreover, in [143] the same bio-inspired optical flow approach is implemented but utilizing a Fuzzy Q-Learning (FQL) method for obstacle avoidance. The Hermite transform, a mathematical image model inspired by the human vision system, is used to highlight obstacles, while the FQL controller makes obstacle avoidance decisions. Preliminary results show successful navigation in unknown environments.

C. End-to-end Learning Method

In recent years, there has been an increase in research focused on artificial intelligence and computer vision, due to the significant benefits these disciplines offer for intelligent robotic systems [150]. Conventional vision-based navigation methods typically involve building a map of the environment and then using planning to reach the goal. Alternatively, Deep Convolutional Neural Network (DCNN) techniques do not require any map because neural networks directly take RGB images as input. They learn from raw pixel data to train an agent to produce desired control actions.

End-to-end learning methods can be categorized into two main types: supervised deep learning and deep reinforcement learning (DRL). In supervised deep learning, a DCNN learns a control strategy that emulates an expert's action choices for autonomous navigation of mobile robots using labeled data-driven information [123]. For instance, Chen et al. propose a data-driven deep learning model for mapping visual inputs to motion in unstructured indoor environments [151], employing transfer learning to enhance generalization, which demonstrates improved accuracy, efficiency, and feasibility. In contrast, DRL methods train all network parameters simultaneously to derive a navigation policy directly from visual images of the surrounding environment [152]. Ejaz et al. introduce a novel network architecture for mobile robots to autonomously navigate collision-free using DRL techniques [153]. Their approach enhances learning speed and exploration capabilities through input data normalization and parametric noise injection. DCNNs extract features from depth images, and their double deep Q-network-based model shows promising results when transferred directly to real robots, achieving superior performance across diverse environments.

Comparison of Vision-Based Techniques

Each vision-based technique for mobile robotics navigation offers distinct advantages and disadvantages. *Visual SLAM* is known for its ability to provide comprehensive mapping and localization simultaneously. This dual functionality is essential for precise navigation, as it allows robots to construct detailed environmental maps while accurately determining their position within the environment. Additionally, cameras used in Visual SLAM capture rich visual information, facilitating the identification and utilization of a wide range of features for mapping and localization. This method is also cost-effective, given that cameras are generally more affordable and compact compared to other sensors like LiDAR. Moreover, Visual SLAM's versatility enables its implementation across various platforms and environments, from indoor settings to outdoor terrains. However, Visual SLAM comes with challenges, including high computational

complexity, which necessitates significant processing power that may not be available on all robotic platforms. Furthermore, Visual SLAM is sensitive to lighting conditions, as changes in lighting can affect the quality of visual data and, consequently, the system's accuracy and reliability. Another limitation is its dependency on feature-rich environments, as detecting and tracking visual features can be difficult in featureless or highly dynamic settings.

Optical Flow methods, on the other hand, are advantageous for their low computational requirements and energy efficiency. These methods are particularly suitable for robots with limited processing capabilities, as they consume less power compared to more complex algorithms like SLAM. Inspired by biological systems, Optical Flow techniques allow robots to navigate using minimal perceptual information, similar to the way insects operate, which can result in significant savings in memory and computational resources. However, Optical Flow has its drawbacks. It primarily provides 2D motion information and lacks direct depth perception, which can limit its effectiveness in navigating complex 3D environments. Additionally, Optical Flow methods are sensitive to rapid or erratic movements, which can degrade the accuracy of motion estimations and negatively impact navigation performance. The information extracted through Optical Flow is often less detailed compared to SLAM, resulting in less precise navigation and obstacle avoidance.

End-to-end Learning represents a different approach, directly mapping visual input to control actions, thus eliminating the need for explicit feature extraction and mapping. This simplifies the navigation pipeline and allows the system to learn complex patterns and features directly from raw image data. End-to-end learning systems are highly adaptable and can be trained to handle a wide range of environments and scenarios, improving their robustness and versatility. However, these systems are data-intensive, requiring large amounts of labeled data for training, which can be time-consuming and costly to collect. They are also computationally demanding, necessitating powerful hardware for both training and inference, which may not be feasible for all robotic platforms. Furthermore, the black-box nature of these models often results in a lack of interpretability, making it challenging to understand and diagnose their decision-making processes, which can be a significant limitation in safety-critical applications.

Because each approach has advantages and disadvantages, it can be used in various situations and robot configurations based on the particular demands of the task under consideration. A summary and comparison between the three studied methods is presented in Table 1.4.

Table 1.4: Comparison of Vision-Based Techniques for Mobile Robotics Navigation

Method	Advantages	Disadvantages
Visual SLAM	- Comprehensive mapping and localization - Rich visual information - Cost-effective - Versatile for various environments	- High computational complexity - Sensitive to lighting conditions - Requires feature-rich environments
Optical Flow	- Low computational requirements - Energy efficient - Inspired by biological systems for minimal perceptual information	- Limited depth information - Sensitive to rapid or erratic movements - Simplistic representation compared to SLAM
End-to-End Learning	- Direct mapping from perception to action - Learns from raw data - High adaptability to various environments	- Data-intensive requiring large datasets - Computationally demanding - Black-box nature lacking interpretability

1.4.3. Motion Planning System Integration

Motion planning is focused on ensuring that the robot can follow a predefined path accurately and adapt to dynamic environments. By integrating robust kinematic models and advanced perception systems, robots can achieve efficient and safe navigation. This integrated approach is crucial for a *sense-to-avoid system*, which detects obstacles and provides real-time data inputs for the motion planning controller, as the one depicted in Figure 1.13. Many studies utilize proximity sensors to monitor object distances and redirect robots when thresholds are exceeded [154]. Occasionally, ranging measurements are combined with visual algorithms to navigate around obstacles effectively [155].

Integrating multiple sensor types ensures redundancy and robustness, allowing the robot to rely on different data sources if one sensor fails. This system not only improves obstacle avoidance but also contributes to a holistic understanding of the environment and a correct implementation of path-following control. This comprehensive approach enables the development of advanced navigation strategies that combine various perception methods, leading to more reliable and efficient robot operation. By leveraging the strengths of sensor fusion and kinematic considerations, robots can achieve a higher level of autonomy and adaptability in dynamic environments, ensuring both safety and efficiency in their navigation tasks.

1.5. Key Insights

In conclusion, this chapter has provided an extensive review of the theoretical framework and literature pertinent to the robotics navigation problem. It has explored the theoretical formulation of navigation, focusing on essential functions such as localization, mapping, and path and motion planning, with particular emphasis on the latter. Both classical and heuristic path planning methodologies were reviewed, discussing their respective strengths and limitations. An analysis of current related work has been conducted, identifying areas of opportunity and trends in the development of path-planning techniques. The integration of global and local path planning was studied to achieve efficient and safe navigation in dynamic environments. Furthermore, the importance of effective motion planning and robust perception systems was highlighted (often overlooked), emphasizing the continuous interaction between the robot and its environment necessary for adaptive navigation.

The next chapter will provide a detailed explanation of the process followed to propose a comprehensive autonomous navigation strategy for a mobile robot, the objective of this dissertation. The key factors identified in this chapter will be used to offer a practical perspective on the development and integration of this navigation strategy. The challenges encountered and the solutions designed to achieve a robust and efficient autonomous navigation system will be discussed.

Chapter 2

Proposal Description & Methodology

This chapter outlines the problem definition, simulation environment, methods, and techniques used for the proposal of the *Comprehensive Autonomous Navigation strategy for Mobile Robots*. Firstly, the problem is defined as navigating a mobile robot between a start position and a goal position in a given environment with both static and dynamic obstacles. Secondly, the simulation environment and the robot model used for testing are described. The chapter also describes the implementation of the global planning and the local planning subsystems for deliberative and reactive motion control, respectively.

2.1. Problem Definition

As previously discussed in the *Hypothesis* section of the *Introduction* chapter, it is imperative to ask pertinent questions to accurately delineate the boundaries of this project. Addressing all the challenges associated with autonomous navigation for AMRs is beyond the scope of this thesis. Therefore, the proposed solution, specifically the navigation strategy, must be precisely adapted to address the defined problem.

In Chapter 1 (Section 1.1), the basic theoretical formulation of the navigation problem was described in Eq. 1.1. In this definition, it is presented that the objective is to generate a sequence of motion commands to move a mobile robot in a given environment between a start position and a goal position. Based on this, it is necessary to describe the problem by determining: the robot to be controlled, and, most importantly, the characteristics of the environment that will be considered.

Firstly, the robot to be used is defined. As analyzed in Section 1.3 (subsections 1.3.3 and 1.3.4), one of the major areas of opportunity in the study and design of navigation strategies is their implementation in real conditions, where the existence of the robot is often omitted, but it is crucial as it more accurately reflects the current limitations in locomotion and mobility, which are likely to influence the overall performance in practical settings. This is why in Section 1.4 it was determined that the AMR to be used for implementing the strategy would be a land-based wheeled mobile robot. By using a WMR, the research and experimentation process can be streamlined, allowing the focus to be on developing and testing the control algorithms. In this regard, it is sufficient to say that the robot to be used has a differential-drive control, which can be managed by determining the speed of each wheel. If the speeds are different, the robot can then turn. Later in this section, the exact mobile robot to be used is presented.

On the other hand, the environment for which this proposal will be developed is defined. As mentioned before, it is not possible to cover all possible scenarios, but the aim is to create a robust navigation system that can generalize its functionality across various environments. The described environment ultimately reflects the implementation that could be provided to the robot. Based also on the review previously mentioned (Section 1.3, subsections 1.3.3 and 1.3.4) regarding current related works, applications in dynamic scenarios are rarely considered; however, these are the real conditions in which the robot will interact, and this research work places special emphasis on this aspect. Below are explained the conditions associated with the environment in which the robot will operate:

- The environment contains static obstacles (such as furniture or walls) that cannot change their position at any time. These obstacles are fully known to the robot before it begins to move.
- In this environment, dynamic obstacles such as new objects, people, or other robots appear. These dynamic elements move throughout the entire scenario; they may stop momentarily but must continue moving. Such obstacles are locally identified by the robot during its movement.

Finally, to completely limit the scope of the problem addressed in this thesis, the navigation strategy to be developed aims to find the shortest route between two positions. This means that through global planning, a feasible route optimized for distance must be found to allow the robot to move between a start position and a goal position.

2.1.1. Simulation Environment

A fundamental aspect of the design and testing of the proposed solution is the choice of the platform on which it will be developed. Once the testing conditions have been described, it is necessary to define the environment in which it will be executed. Unfortunately, due to resource, time, and availability constraints, the strategy cannot be executed in a real physical environment. However, given the focus on being as close to reality as possible, the proposal will have to be implemented in a realistic robotic simulation environment.

The use of computer-based simulations is good practice in the process of developing new robot designs and algorithms, before building and executing code on physical robotic systems. As robots and associated software become more dynamically capable and complex, simulation tools are increasingly being used to emulate the realistic motion of 3D robot models in a range of virtual environments [156]. One of the main advantages of using such a simulation environment is that it is more cost-effective and less time-consuming; it allows the exploration of a wider variety of scenarios without risking damage to the robot and without compromising real-time control and considerations of robot motion and dynamic environments.

CoppeliaSim, also known as V-REP, is a robotics simulation environment that offers a flexible and adaptable infrastructure for generating 3D simulations efficiently within a limited timeframe, making it a valuable tool for various applications [157, 158]. In this work, the educational version of CoppeliaSim software is used to implement, evaluate, and validate the proposed method for the entire navigation strategy. This software provides numerous benefits in terms of robotic system and controller development [143, 159, 160, 161]. For instance, it enables the simulation of diverse environmental conditions and physical interactions of robots and objects, including physics and collisions.

Moreover, CoppeliaSim is a cross-platform tool that supports multiple coding approaches and comprises a remote API that supports six programming languages. To conduct this research, CoppeliaSim is integrated with Python, which allows us to remotely control a virtual robot, leveraging its extensive library of modules. It is important to mention that, in addition to the conditions of the scenario described earlier, the simulation involves new conditions such as ignoring the robot's energy consumption or uneven ground and having easy access to all sensor information, including the exact position of the robot.

2.1.2. Pioneer 3-DX

On the other hand, it is also important to precisely identify the robot that will be used in the implementation. Pioneer research robots are the world's most popular intelligent mobile robots for education and research. Their versatility, reliability, and durability have made them the preferred platform for advanced intelligent robotics [162]. Therefore, as shown in Figure 2.1, the differential robot used as a testing robot is the Pioneer 3-DX model, which has proven effective for testing autonomous navigation strategies [163, 164, 165]. This robot is designed to move over indoor terrain with a maximum speed of 1.2 m/s. This robot is equipped with sixteen sonar sensors, which detect the surrounding unknown environment. Figure 2.2 illustrates the main dimensions of the robot.

Figure 2.1: Pioneer 3-DX Model in CoppeliaSim

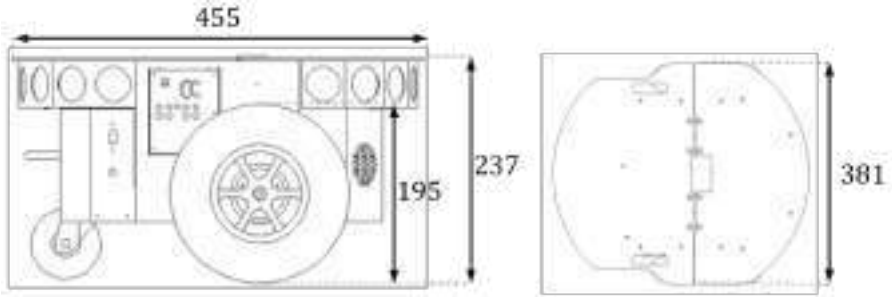


Figure 2.2: Pioneer 3-DX Dimensions in mm

With all these considerations, it can be determined that the specific problem to solve is to move a Pioneer 3-DX (a differential-driven WMR) within a complex indoor environment with both static and dynamic obstacles implemented in CoppeliaSim, such as those found in industrial settings, healthcare facilities, or domestic scenarios. In the following sections, the methods and subsystems proposed to solve this problem will be explained in detail.

2.2. Methods and Techniques

In Chapter 1, an extensive review of the concepts related to the problem of robotic navigation was conducted. Additionally, a review of the methods and techniques associated with solving the problem was performed. Some of these methods are utilized in the proposed navigation strategy, which will be described later. Each of these methods has been carefully selected for its ability to address different aspects of the problem of autonomous navigation in static and dynamic environments.

The following presents the theoretical foundations and key characteristics of the *Grey Wolf Optimization algorithm*, the *Optical Flow method*, and the *Fuzzy Logic Controller*, providing a solid basis for understanding their subsequent use in the implementation of the global and local planning subsystems.

2.2.1. Grey Wolf Optimization

The Grey Wolf Optimization GWO algorithm is a population-based meta-heuristics algorithm inspired by the hunting behavior of grey wolves, which was first presented by Seyedali Mirjalili et al. in 2014 [102]. This method mimics the rigid social hierarchy and hunting tactics of wolves to solve optimization problems. In GWO, the wolf population is classified into alpha, beta, delta, and omega, and three main behaviors are simulated: tracking, encircling, and attacking the prey. These behaviors are used to effectively explore and exploit the search space.

A. Inspiration of the algorithm

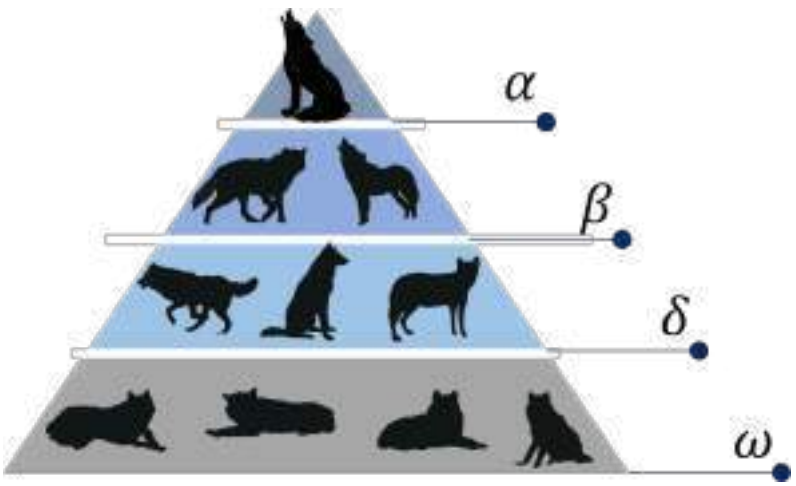


Figure 2.3: Social Hierarchy of Grey Wolf (dominance decreases from the top down)

All the individuals in a grey wolves group have a very strict social dominance hierarchy as illustrated in Figure 2.3, where dominance decreases from top to bottom. Grey wolf packs are led by an *alpha wolf* (α). The alpha wolves make decisions regarding hunting, sleeping arrangements, and waking schedules, which are communicated to the pack. Although alphas hold significant authority, they sometimes exhibit democratic behavior by following other pack members. The alpha role is based not solely on physical strength but also on the ability to manage the pack effectively, highlighting the importance of discipline and organization.

The beta (β) wolves occupy the second level of the hierarchy. Betas are subordinate wolves that assist the alpha in decision-making and pack activities. They are the most likely candidates to assume the alpha position if necessary. Betas respect the alpha but also have authority over lower-ranking members. They act as advisors to the alpha, reinforce commands, and ensure discipline within the pack.

The lowest-ranking member is the omega (ω), who serves as a scapegoat for the others and is the last to access food. Despite seeming insignificant, the omega plays a crucial role in maintaining the pack's social structure by absorbing aggression and frustration from other members, which helps preserve overall harmony. Omegas may also act as babysitters for the pack.

Wolves that are neither alpha, beta, nor omega are classified as subordinates, or deltas (δ). Subordinates must submit to alphas and betas but hold dominance over omegas. This group includes scouts, sentinels, elders, hunters, and caretakers. Scouts monitor the pack's territory, sentinels protect the pack, elders provide experience, hunters assist in securing food, and caretakers look after weak, ill, or injured members.

In addition to their hierarchical social structure, grey wolves exhibit coordinated group hunting behavior (Figure 2.4), which involves the following phases:

- Tracking, chasing, and approaching the prey (Figure 2.4a)
- Pursuing, encircling, and harassing the prey until it stops moving (Figure 2.4b)
- Attacking the prey (Figure 2.4c)

The social hierarchy and hunting behavior of grey wolves are mathematically modeled to design GWO.

(a) Tracking the prey

(b) Encircling the prey

(c) Attacking the prey

Figure 2.4: The Grey Wolf Hunting Strategy

B. Mathematical model and algorithm

When creating the GWO, the social hierarchy of wolves was used as a mathematical model. The fittest solution in the optimization process is designated as the alpha (α). The second and third best solutions are referred to as beta (β) and delta (δ), respectively. The rest of the candidate solutions are assumed to be omega (ω) wolves.

Encircling Prey As previously discussed, grey wolves surround their prey during the hunting process. To create a mathematical representation of this encircling behavior, the following equations are suggested:

$$\vec{D} = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)| \tag{2.1}$$

$$\vec{X}(t+1) = |\vec{X}_p(t) - \vec{A} \cdot \vec{D}| \tag{2.2}$$

where $\vec{D}$ is the vector distance between the prey and the wolf; t indicates the current iteration; $\vec{A}$ and $\vec{C}$ are coefficient vectors; $\vec{X}_p$ is the position vector of the prey; and $\vec{X}$ indicates the position vector of a grey wolf.

The vectors $\vec{A}$ and $\vec{C}$ are calculated as follows:

$$\vec{A} = 2 \cdot a \cdot \vec{r}_1 - a \tag{2.3}$$

$$\vec{C} = 2 \cdot \vec{r}_2 \tag{2.4}$$

where a is a scalar that decreases linearly from 2 to 0 over the iterations and $\vec{r_1}$, $\vec{r_2}$ are random vectors with coefficients between $[0, 1]$. a can be expressed as follows, where t represents the most recent iteration and t_{max} is the maximum number of iterations:

$$a = 2 - 2 \cdot \left(\frac{t}{t_{max}} \right) \tag{2.5}$$

In order to observe the impact of equations 2.1 and 2.2, Figure 2.5 displays a two-dimensional position vector and several potential neighbors. The figure reveals that a grey wolf located at position $\vec{X}(t) = (X, Y)$ can alter its position based on the prey's position $\vec{X}_p = (X^*, Y^*)$. By modifying the values of $\vec{A}$ and $\vec{C}$ vectors, diverse locations near the best agent can be identified from the current position. It is worth noting that the random vectors $\vec{r}_1$ and $\vec{r}_2$ enable wolves to reach any point in between the illustrated positions. Therefore, by utilizing equations 2.1 and 2.2, a grey wolf can change its position to any random location within the space surrounding the prey. This same principle can be applied to a search space with $n > 2$ dimensions, where the grey wolves move around the best solution found so far.

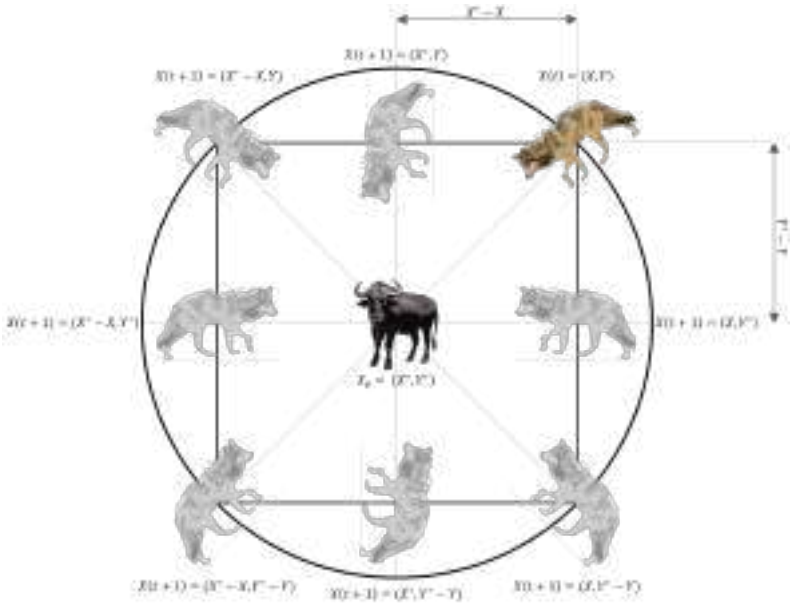


Figure 2.5: 2D Position Vectors in GWO and their potential future locations

Hunting Grey wolves have the capacity to locate their prey and hunt them down, usually led by the alpha (α). Sometimes the beta (β) and delta (δ) may also participate in the hunt. However, when dealing with an abstract search space, it is difficult to determine the location of the optimal prey. To simulate the hunting behavior of grey wolves mathematically, it is assumed that the alpha, beta, and delta have better knowledge of where the potential prey is located. As a result, the top three best solutions found are saved and require other search agents (including the omegas) to update their positions based on the best search agents positions, according to the formulas below:

$$\begin{aligned} \vec{D}_\alpha &= |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}| \\ \vec{D}_\beta &= |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}| \\ \vec{D}_\delta &= |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}| \end{aligned} \tag{2.6}$$

$$\begin{aligned} \vec{X}_1 &= \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha \\ \vec{X}_2 &= \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta \\ \vec{X}_3 &= \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta \end{aligned} \tag{2.7}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{2.8}$$

The diagram presented in Figure 2.6 illustrates how a search agent modifies its location based on alpha, beta, and delta within a two-dimensional search area. It can be seen that the final position will be a random point within a circular region that is outlined by the positions of alpha, beta, and delta in the search space. In simpler terms, alpha, beta, and delta are approximations of the prey's location, and the other wolves modify their locations indiscriminately around the prey.

In [166], a static weighting average strategy was presented, operating on the premise of assigning weights to the three leader wolves based on the hierarchical structure of a pyramid. Specifically, this approach involves allocating weights of 0.5, 0.3, and 0.2 to the wolves denoted as α, β, and δ, respectively. This ensures that the most prominent wolf in the hierarchy carries the most significant weight, followed by the second in command, and so on. Equation 2.9 illustrates this average weighting approach, unlike equation 2.8. Both approaches are considered during the design and testing of the proposal.

$$\vec{X}(t+1) = \frac{5\vec{X}_1 + 3\vec{X}_2 + 2\vec{X}_3}{10} \tag{2.9}$$

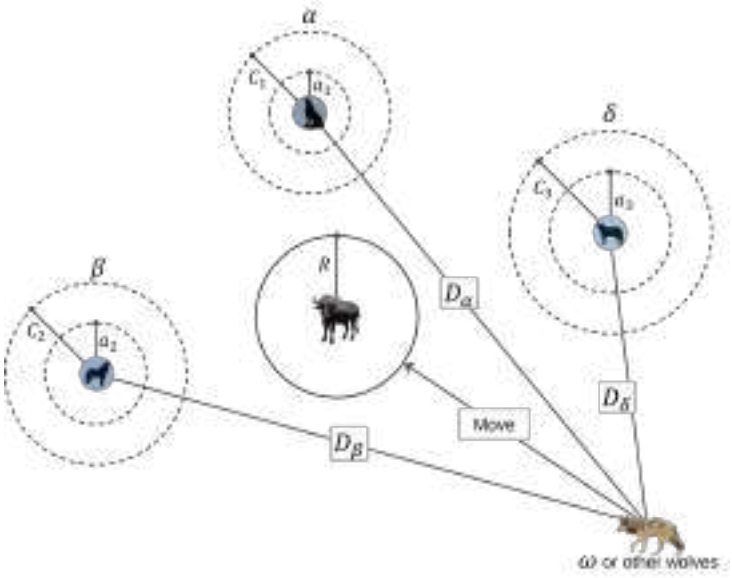


Figure 2.6: Position updating in the GWO

Attacking prey (exploitation) vs Searching for prey (exploration) As previously stated, grey wolves hunt by attacking their prey when it stops moving. To create a mathematical model for approaching the prey, the value of $\vec{a}$ is gradually decreased from 2 to 0 throughout the iterations. This results in a reduction in the range of fluctuation for $\vec{A}$ since it is a random value that falls within the interval $[-2a, 2a]$. When the values of $\vec{A}$ are in the range $[-1, 1]$, the agent can move to any location between its current position and that of the prey.

While the encircling mechanism provides some exploration, the GWO algorithm requires additional operators to enhance its exploration capabilities, as grey wolves disperse from each other during the hunt but converge when it's time to attack. To create a mathematical model for this dispersal, the parameter $\vec{A}$ is also used, with randomized values either greater than 1 or less than -1, forcing the searching agent to diverge from the target and encouraging exploration. Figure 2.7 illustrates the balance between attacking and exploring. $|A| < 1$ forces the wolves to attack the prey, while setting $|A| > 1$ causes the grey wolves to diverge from the prey, enabling them to search globally for more optimal solutions.

$\vec{C}$ element also supports the exploration in GWO. As shown in Equation 2.4, $\vec{C}$ consists of random values within the range of $[0, 2]$. This component randomly adjusts the weights assigned to prey, either amplifying ($C > 1$) or reducing ($C < 1$) their impact in Equation 2.1. This promotes exploration and prevents local optima by ensuring unpredictable behavior. Unlike $\vec{A}$, $\vec{C}$ is not linearly reduced, maintaining random values consistently to enhance exploration, especially during the final stages when the optimization process might stagnate at local optima.

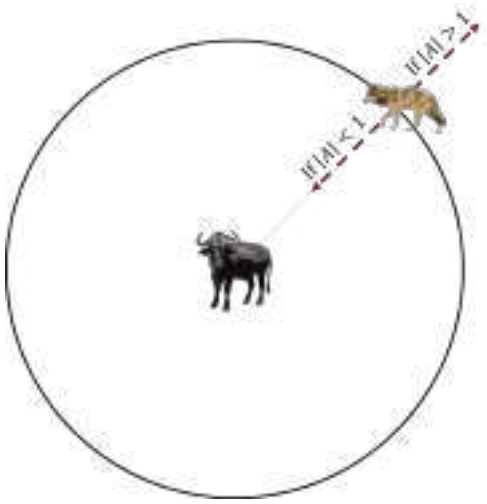


Figure 2.7: Attacking prey versus Searching for prey in GWO

C. GWO implementation

According to current trends, the global planner needed to be based on a metaheuristic algorithm. A natural question that arises from choosing GWO as the global path planner optimizer for this proposal is: What makes it special compared to other options? Among various metaheuristic algorithms studied in Section 1.3 for global path planning, GWO has demonstrated highly competitive results compared to other well-known approaches such as PSO or GA [102]. Additionally, it is a relatively new algorithm (10 years at the time of writing this thesis), thus there is a limited investigation by the research community with under-researched applications in the literature [167]. This implies a wide margin for exploration and study across different applications, to generate more knowledge related to measuring its performance in various possible implementations.

The potential of GWO in mobile robotics applications is promising, particularly in the context of path planning. According to [167], although less than 5% of the different applications of the GWO Algorithm presented between 2018 and 2023 are related to robotics and path planning optimization, recent research has demonstrated the successful use of the GWO algorithm in ensuring safe and optimal navigation of mobile robots [103, 168, 169, 170].

The GWO pseudocode (Algorithm 1) outlines the steps involved in initializing and updating the population of grey wolves, which are categorized as alpha, beta, and delta based on their fitness. The algorithm iterates to update the positions of the search agents, enhancing both exploration and exploitation, which is crucial for effective path planning.

Algorithm 1 Pseudocode of the GWO Algorithm

Initialize the grey wolf population with n search agents ($\vec{X}_i$, $i = 1, 2, 3, ..., n$) and j values each
Initialize $\vec{X}_\alpha, \vec{X}_\beta, \vec{X}_\delta$ to ∞ for minimization or $-\infty$ for maximization

while $t < t_{max}$ **do**
  for *each search agent* $\vec{X}_i$ **do**
    Calculate the objective function (*fitness*)
    Update $\vec{X}_\alpha, \vec{X}_\beta, \vec{X}_\delta$
  end for
  Estimate a value using Equation 2.5
  for *each search agent* $\vec{X}_i$ **do**
    for *each value* j *in* $\vec{X}_i$ **do**
      Estimate $\vec{A}$ vector using Equation 2.3
      Estimate $\vec{C}$ vector using Equation 2.4
      Update the j value of the search agent $\vec{X}_i$ by Equation 2.8
    end for
  end for
  $t = t + 1$
end while

Return $\vec{X}_\alpha$

The GWO algorithm ensures that the search agents, representing potential solutions, are dynamically adjusted to converge towards the optimal path. This iterative process, involving continuously updating positions and fitness evaluations, allows the algorithm to balance exploration and exploitation effectively, thus finding the optimal path for mobile robots. The implementation of this algorithm in the proposal is fully explained in Section 2.3, *Global Planning Subsystem* subsection.

2.2.2. Optical Flow

Optical Flow (OF) is a concept in computer vision that refers to the apparent motion of objects, surfaces, and edges in a visual scene, caused by the relative motion between an observer and the scene [171]. Recent breakthroughs in computer vision research have enabled machines to perceive their surrounding world through techniques such as object detection, which detects instances of objects belonging to a certain class, and semantic segmentation, which provides pixel-wise classification. However, for processing real-time video input, most implementations of these techniques only address relationships of objects within the same frame (x, y), disregarding time information (t), which is crucial in obstacle avoidance systems like the one proposed.

OF emerges as a powerful tool for real-time image analysis. The problem of optical flow can be illustrated as shown in Figure 2.8, where the image intensity (I) between consecutive frames can be expressed as a function of space (x, y) and time (t). If a first image $I(x, y, t)$ is taken and its pixels move by $(\delta x, \delta y)$ over δt time, a new image $I(x + \delta x, y + \delta y, t + \delta t)$ is obtained. This analysis aims to estimate the motion vector of the pixels, which is useful for real-time image analysis applications such as object tracking or motion detection.

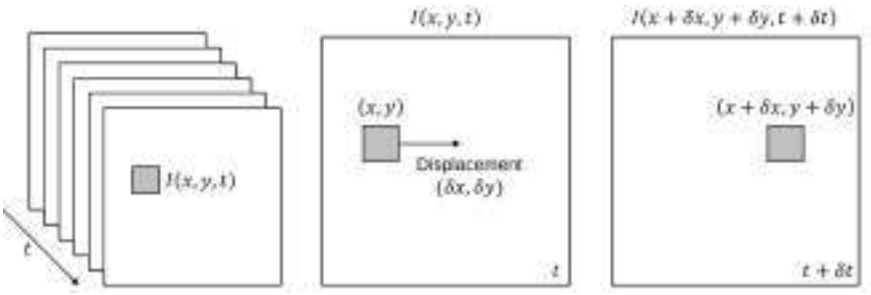


Figure 2.8: Optical Flow Problem

To solve the problem, it is first assumed that the pixel intensities of an object are constant between consecutive frames:

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{2.10}$$

Another assumption that has to be made is that the pixel displacement and difference in time t between the two frames are sufficiently small. Subsequently, the Taylor Series approximation of the right side of the Equation 2.10 is taken:

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + \text{higher-order terms}$$

By truncating the high-order terms and removing common terms:

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

Finally, dividing by δt, the optical flow equation is obtained:

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{2.11}$$

where $u = \frac{\delta x}{\delta t}$ and $v = \frac{\delta y}{\delta t}$ are the x and y components of the velocity or optical flow of $I(x, y, t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image along the horizontal axis, the vertical axis, and time, respectively.

Therefore, the problem reduces to solving $u = \frac{\delta x}{\delta t}$ and $v = \frac{\delta y}{\delta t}$ to determine movement over time, or in other words, finding the optical flow vector, denoted as ($\vec{F}$) and expressed as:

$$\vec{F} = (u, v) \tag{2.12}$$

However, it can be noted that the optical flow equation cannot be directly solved for u and v since there is only one equation for two unknown variables (also known as the aperture problem). To address this issue, optical flow methods introduce additional constraints and conditions.

It is important to highlight that for each pixel in the image and each moment (t), the optical flow vector ($\vec{F}$) can be calculated and decomposed into magnitude ($|\vec{F}|$) and direction ($\theta_{\vec{F}}$), which are determined using Equations 2.13 and 2.14, respectively.

$$|\vec{F}| = \sqrt{u^2 + v^2} \tag{2.13}$$

$$\theta_{\vec{F}} = \arctan\left(\frac{v}{u}\right) \tag{2.14}$$

The magnitude indicates the speed of the pixel's movement, while the direction indicates where the pixel is moving in the image plane. Depending on the implementation, it might suffice to use the information from either of these two components.

A. Optical Flow Methods

The concept of optical flow was first introduced by the American psychologist James J. Gibson in 1950, who used the term to describe the visual stimulus provided to animals moving through the world [172]. Its relation to computer vision did not emerge until the early 1980s, when computational tools for image analysis were introduced. Nowadays, methods to solve Eq. 2.11 can be distinguished into two types: *sparse optical flow* and *dense optical flow*.

Sparse Optical Flow Sparse optical flow selects a sparse set of feature pixels (e.g., interesting features such as edges and corners) to track their velocity vectors (motion). The extracted features are passed into the optical flow function from frame to frame to ensure that the same points are being tracked. There are various implementations of sparse optical flow, including the Lucas–Kanade method, which is a differential technique for optical flow estimation that assumes a constant flow within a small neighborhood of each pixel [173]. It solves the optical flow equation by applying the least squares criterion.

Specifically, this technique estimates the velocity vector by minimizing the sum of squared differences of the image intensity values in a local window:

$$\sum_i \left(\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} \right)_i^2 \tag{2.15}$$

This method is computationally efficient and works well for small motions and smooth image regions, but it may struggle with large displacements or regions with little texture.

Dense Optical Flow Dense optical flow attempts to compute the optical flow vector for every pixel in each frame. While such computation may be slower, it provides a more accurate and denser result suitable for applications such as learning structure from motion and video segmentation. Various implementations of dense optical flow exist. One of them, the Horn-Schunck method, is a global optical flow estimation technique that introduces a smoothness constraint to address the aperture problem [142]. It assumes that the flow varies smoothly over the entire image. The method is formulated as a global energy function to be minimized that combines the data term (measuring the optical flow equation error) and a smoothness term, as shown in Eq. 2.16:

$$E = \iint \left[\left(\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} \right)^2 + \alpha \left(\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 \right) \right] dx\, dy \tag{2.16}$$

where α is a regularization parameter. This method is capable of producing dense flow fields but can be sensitive to the choice of the regularization parameter and computationally intensive.

The Farneback method is another dense optical flow estimation technique that models the neighborhood of each pixel using quadratic polynomials as expressed in Eq. 2.17 [174]. It approximates the displacement field by finding the polynomial expansion of the signal in the first frame and then warping it to fit the second frame. This method involves two main steps: polynomial expansion and displacement estimation.

$$I(x, y) \approx \begin{bmatrix} x & y \end{bmatrix} A \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_x & b_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + c \tag{2.17}$$

where (x, y) represents the spatial coordinates of a pixel in the image, A is a symmetric 2×2 matrix, (b_x, b_y) represents a b coefficient vector, and c is a scalar. The Farneback method is known for its balance between accuracy and computational efficiency, making it suitable for real-time applications.

The implementations of the Lucas-Kanade, Horn-Schunck, and Farneback methods typically generate an image pyramid, a multi-scale structure that represents the image at various resolutions [174, 175]. This approach serves to improve the accuracy and robustness of flow estimation and to handle large displacements. Each level of the pyramid represents a smaller scale than the preceding one, with progressively lower resolution. This multi-scale approach enables the algorithm to handle larger displacements of points between frames effectively. The tracking process starts at the lowest resolution level and continues until convergence. The point locations detected at a level are propagated as key points for the succeeding level. In this way, the algorithm refines the tracking with each level. The result is a vector field (flow) that describes the pixel displacement from the previous frame to the current one, providing detailed motion information. Figure 2.9 illustrates an example of an image pyramid with three levels.

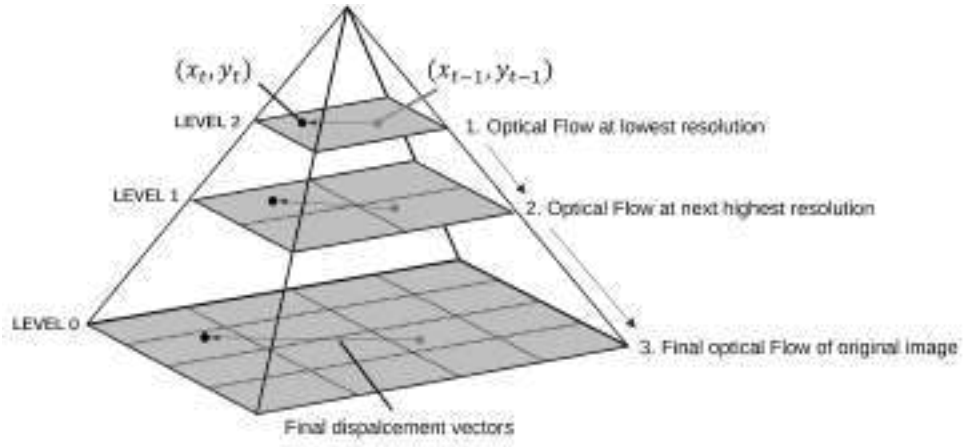


Figure 2.9: Image pyramid used in some techniques for Optical Flow estimation

Optical flow using Deep Learning In addition to the historical approaches to solving the problem of optical flow, which have treated it as an optimization problem, recent approaches applying deep learning have shown impressive results [176, 171, 177]. Generally, such approaches take two video frames as input to output the optical flow, which may be expressed as:

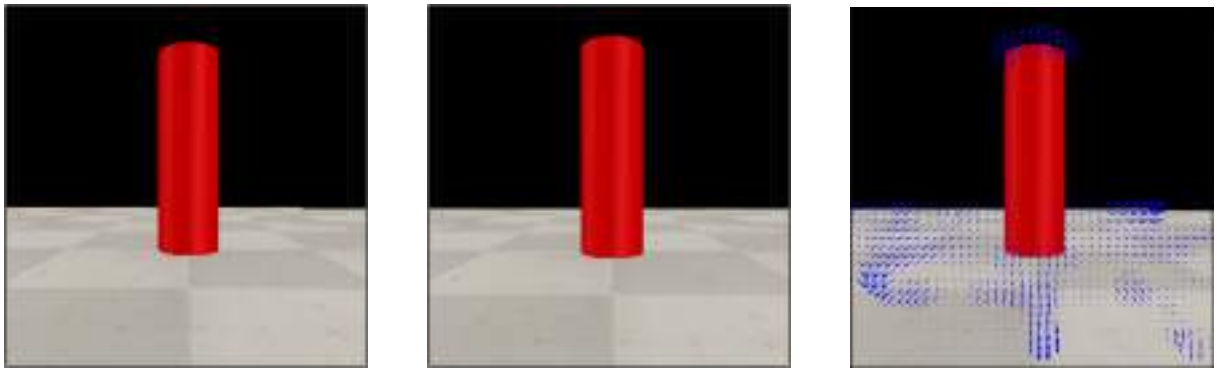
$$(u, v) = f(I_t, I_{t+1}) \tag{2.18}$$

where u is the motion in the x direction, v is the motion in the y direction, and f is a neural network that takes in two consecutive frames I_t (frame at time $= t$) and I_{t+1} (frame at time $= t + 1$) as input. However, computing optical flow with deep neural networks requires large amounts of training data, which is particularly hard to obtain because labeling video footage necessitates accurately determining the motion of every point in an image to sub-pixel accuracy.

B. Optical Flow Implementation

The purpose of this thesis is not to solve the problem of OF, but rather to use one of the methods described above to implement the obstacle avoidance system with optical flow estimation, as described in Section 2.3. The intuitive idea is that the more relative movement detected in a region of the image, the more likely it is to indicate the presence of obstacles.

Figure 2.10 shows an example of optical flow estimation. Figures 2.10a and 2.10b are two consecutive frames from a video obtained from CoppeliaSim. In these images, a red cylindrical obstacle can be seen in the center of the image, on a checkered floor and a black background. Concerning the camera, the obstacle moves closer from the first image to the second image. Figure 2.10c shows the optical flow estimation (using the Farneback method from the OpenCV library in Python). It can be observed that the vectors have greater presence and magnitude in areas where the intensity or color of the region in the image changes significantly, such as the upper edge of the obstacle or the lines of the squares on the floor.



(a) Frame 1 (b) Frame 2 (c) Optical flow estimation

Figure 2.10: Optical Flow Estimation Example

According to the review presented in Subsection 1.4.2 regarding perception systems, it was determined to use a visual detection system for the proposal. Among the methods studied, optical flow was chosen due to its relatively low computational requirements and its ease of real-time implementation, allowing robots to navigate using minimal perceptual information without significant memory usage. Additionally, its effectiveness has already been demonstrated for real-time obstacle avoidance systems in mobile robots [143, 146, 148, 149], where it has shown adaptability to complex environments, robustness, reliability, and compatibility with other systems (something necessary for the proposal developed in this thesis). The implementation of this method in the proposal is fully explained in Section 2.3, *Local Planning Subsystem* subsection.

2.2.3. Fuzzy Logic Controller

Fuzzy logic FL was first proposed by Lotfi Zadeh in a 1965 paper for the journal Information and Control, where he attempted to reflect the kind of data used in information processing and derived the elemental logical rules for this kind of set [65]. Since then, FL has been successfully applied in industrial processes [178], image processing [179], aerospace engineering [180], automotive traffic control [181], business decision-making [182], artificial intelligence [183], and other fields that rely on signals with ambiguous interpretation, including robotics navigation.

FL originates from the mathematical study of multi-valued logic. While ordinary logic, or Boolean logic, deals with statements of absolute truth (e.g., "Is it cold?" with a yes or no answer), fuzzy logic addresses sets with subjective or relative definitions, such as "very much," "little," or "very less." This approach aims to emulate the way humans analyze problems and make decisions, relying on imprecise values rather than absolute truth or falsehood. In the boolean system truth value, every statement must have an absolute value: true (1) or false (0). In fuzzy logic, truth values are replaced by degrees of "membership" from 0 to 1, where 1 is true and 0 is false.

A. Fuzzy Logic Architecture

The point of fuzzy logic is to map an input space to an output space. The primary mechanism for doing this is a list of if-then statements called *rules* or *rule base*. All rules are evaluated in parallel, and the order of the rules is unimportant. The rules are useful because they refer to variables and the adjectives that describe them. In general, fuzzy inference is a method that interprets the values in the input vector and, based on some set of rules, assigns values to the output vector. The fundamental architecture of the fuzzy controller comprises three main components: *fuzzification*, *inference engine*, and *defuzzification* [184], as can be seen in Figure 2.11.

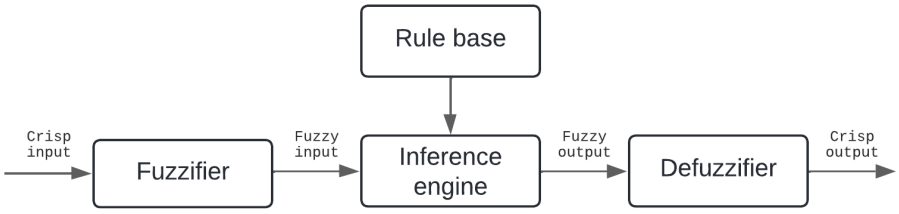


Figure 2.11: Fuzzy Logic System

The rule base contains the set of fuzzy rules provided by experts to govern the decision-making system. A fuzzy rule is expressed in the following format:

IF *antecedents* THEN *consequents*

Once the rule base is defined, the initial stage in implementing a fuzzy controller involves fuzzification, which converts real-valued inputs into fuzzy sets that measure the membership grades corresponding to fuzzy control terms. These crisp inputs are the exact inputs measured by sensors and passed into the control system for processing.

The subsequent phase is fuzzy inference, which combines the information derived from the fuzzification process with the rule base and performs the fuzzy reasoning procedure. In other words, fuzzy inference is the process of formulating the mapping from a given input to an output. Various methods of fuzzy inference exist, depending on the intended applications and the nature of the membership function.

The third and final stage of fuzzy logic is defuzzification. The objective of this part is to transform the fuzzy sets of the outputs, which are calculated by the inference engine, into a crisp value to be used as a control command for example.

Membership Function One fundamental concept of fuzzy logic is the membership function, which defines the degree of membership of an input value to a specific set or category. This function defines how each point in the input space is mapped to membership value between 0 and 1, where 0 indicates non-membership and 1 indicates full membership. Input space is often referred to as the universe of discourse or universal set (U), which contains all the possible elements of concern in each particular application. For example, a classical set might be expressed as:

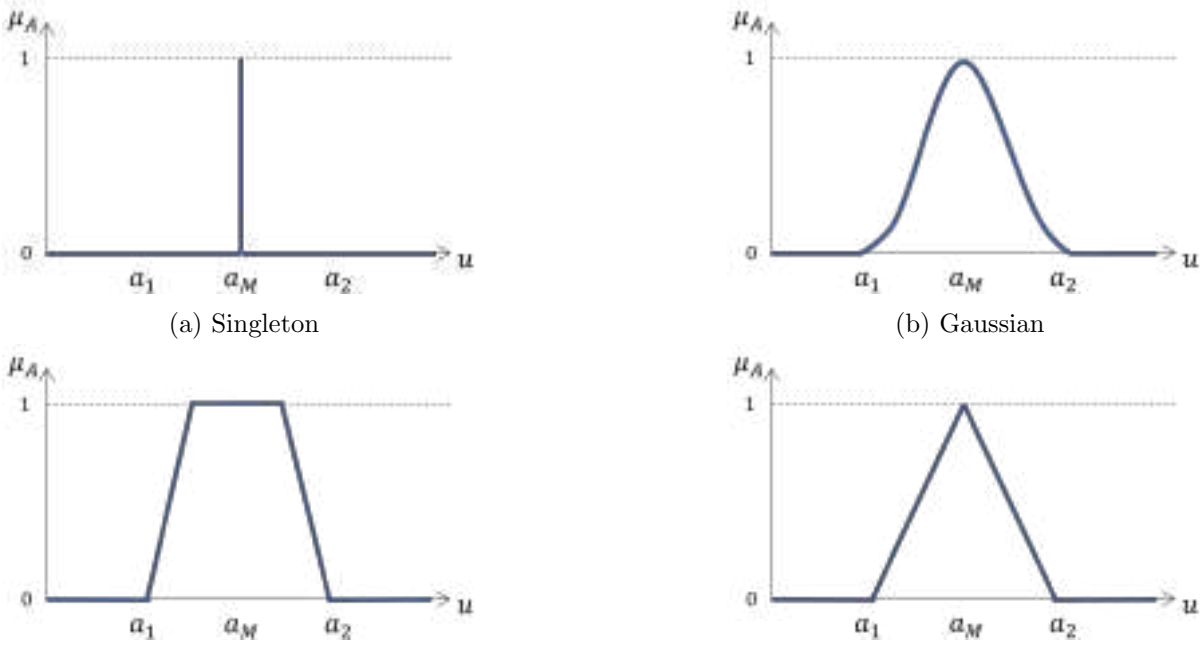
$$U = \{u | u > 6\}$$

A fuzzy set is an extension of a classical set. If U is the universe of discourse and its elements are denoted by u, then a fuzzy set A in U is defined as a set of ordered pairs:

$$A = \{u, \mu_A(u) | u \in U\}$$

where $\mu_A(u)$ is called the membership function of u in A.

As shown in Figure 2.12, the most common types of membership functions are *Singleton* (Figure 2.12a), *Gaussian* (Figure 2.12a), *Trapezoidal* (Figure 2.12c), and *Triangular* (Figure 2.12d). A singleton fuzzifier maps a crisp input to a fuzzy set where one element has a membership value of 1, representing the input as a single point. A Gaussian fuzzifier uses a bell-shaped curve defined by the mean and standard deviation to map inputs. Lastly, trapezoidal and triangular fuzzifiers use linear segments to form trapezoid or triangle shapes in mapping inputs to fuzzy sets.



(a) Singleton (b) Gaussian

(c) Trapezoidal (d) Triangular

Figure 2.12: Types of Membership Functions for Fuzzy Logic Rules

Types of Fuzzy Logic Controllers It is important to mention that there are two primary types of fuzzy logic controllers based on how inputs and outputs are represented in "if-then" rules, each with distinct characteristics and applications [185]. Firstly, the Mamdani fuzzy controller, introduced by Ebrahim Mamdani in 1974 [186], is the most traditional and widely used. It employs a rule-based system with fuzzy sets for both antecedents (input conditions) and consequents (output actions), producing a fuzzy set output that requires defuzzification to obtain a precise value. This controller is valued for handling complex systems with easily interpretable rules and is applied in areas such as mobile robotics navigation [187], manufacturing systems [188], and automotive sensing [189].

On the other hand, the Takagi-Sugeno (T-S) fuzzy controller, proposed by Takagi and Sugeno in 1985 [190], differs from the Mamdani controller in that its "if-then" rules have fuzzy antecedents

but typically uses singleton output membership functions that are either constant or a linear function of the input values, as consequents. This results in a weighted average output, providing precise control and easy integration with traditional techniques. The T-S controller is ideal for adaptive, dynamic systems requiring real-time adjustments and has been effectively utilized in tasks such as traffic light control [191], robotics [192], and aircraft systems [193].

Fuzzy inference is the process of formulating the mapping from a given input to an output using fuzzy logic [194]. Figure 2.13 shows the Fuzzy Inference Process for Mamdani and Takagi-Sugeno Systems [195].

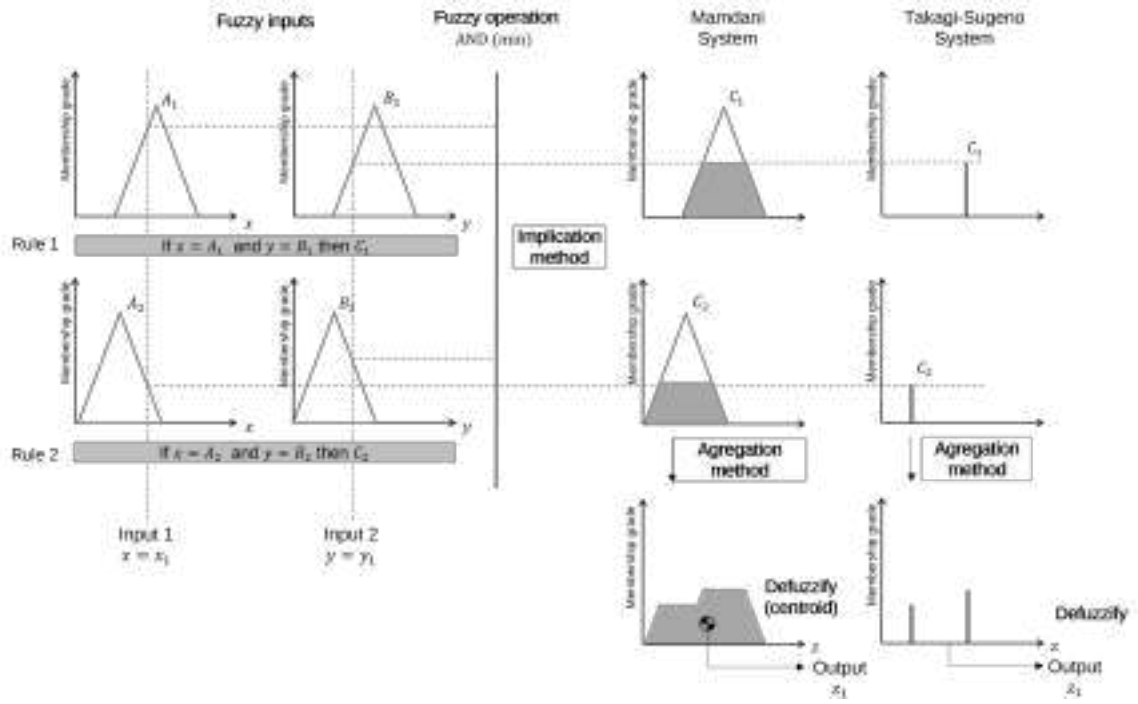


Figure 2.13: Fuzzy Inference Process for Mamdani and Takagi-Sugeno Systems

Initially, inputs are processed through *fuzzification*, where they are evaluated against relevant fuzzy sets using membership functions. Each rule is formulated based on these determined membership values. When a rule's antecedent consists of multiple parts, a *fuzzy operation* is applied, typically using the AND operator (interpreted as the minimum value function), to produce a single truth value representing the rule's antecedent. This truth value is subsequently applied to the output function. An *implication method* then determines the rule's consequent by producing a fuzzy set, which is truncated to the previously obtained value. This process is executed for each rule and is similar for both systems; however, Mamdani uses triangular membership functions as consequents, whereas Takagi-Sugeno uses singleton values.

Since decision-making involves evaluating all rules, it necessitates the combination of rule outputs. The *aggregation method* is employed to combine the fuzzy sets representing each rule's outputs into a single fuzzy set, which occurs once per output variable. Finally, in the *defuzzification* process, the aggregate output fuzzy set is converted into a single numerical value. The Mamdani system typically employs the centroid method to achieve a crisp output, whereas the Takagi-Sugeno system produces an output as the weighted average of all rule outputs.

B. Fuzzy Logic Implementation

Fuzzy logic is used in the design of solutions for local navigation, global navigation, path planning, steering control, and rate control of a mobile robot [196]. This algorithm was studied in Section 1.3, where it was highlighted that in the navigation of AMRs, fuzzy logic is not only useful for path planning but also considers motion control, particularly the path-following process (Section 1.4, *Path Following* subsection). This characteristic is precisely why fuzzy logic was chosen for the proposed solution (see Section 2.3): to address one of the major challenges in the development of navigation strategies, namely the lack of consideration for the real motion of the robot. Additionally, controllers based on artificial fuzzy logic algorithms have gained significant popularity in recent years. Fuzzy Logic Controllers (FLCs) are widely used to produce control outputs for complex and nonlinear systems due to their capability to handle heuristic and imprecise information [197]. The flexibility of FLCs to establish nonlinear control algorithms is a well-known advantage, especially in the motion control and tracking of mobile robots [184, 198].

Similar to the previously described methods of GWO and OF, the Fuzzy Logic Controller is intended to be integrated as another technique to solve the presented problem. The idea is to use raw data (or minimally processed data) from the robot's sensors, as well as the robot's relative position to the next target configuration. Through a fuzzy logic system, the output will be the speed of each wheel of the differential robot, as shown in the black box diagram in Figure 2.14. The implementation of this method in the proposal is fully explained in Section 2.3.2.

Figure 2.14: Black Box Diagram of Fuzzy Logic System Implementation

2.3. Navigation Strategy General Description

This section describes the proposed navigation strategy to solve the problem defined above. The aim of the strategy aligns with the primary goal of this research work (as described in *General Objective* section) and consists of developing and implementing a comprehensive autonomous navigation system for a WMR that can navigate in different dynamic environments automatically, efficiently, safely, and accurately. To achieve this, a global path planning strategy must be implemented to find the shortest feasible route between a starting position and a target point in an environment with static obstacles. Additionally, it is necessary to implement a sensor-based reactive local path planning strategy to safely avoid unknown dynamic obstacles while the WMR moves along the global path (sense-to-avoid system). Finally, unlike much of the related work, the motion planning strategy must be performed using insights from both global and local path planning and tested for functionality in a partially dynamic virtual environment (as described in the previous section).

The diagram representing the *comprehensive autonomous navigation strategy proposal* is shown in Figure 2.15. The proposal is based on the complete navigation pipeline described in Figure 1.6 (Section 1.2, *Global & Local Path Planning* subsection).

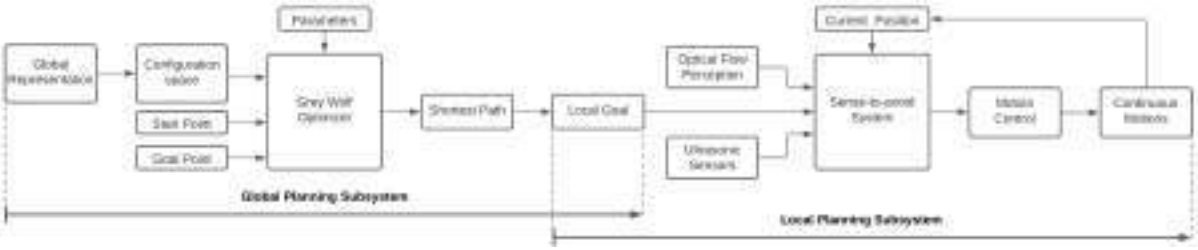


Figure 2.15: Comprehensive Autonomous Navigation Strategy Proposal

The description of the comprehensive autonomous navigation strategy is presented in the following sections.

2.3.1. Global Planning Subsystem

The *Global Planning Subsystem* is executed offline and is responsible for finding the approximate shortest path between two positions within the environment (if such a path exists). In this case, only static obstacles are considered, such as walls or furniture within the environment, which will not change their location during the robot's execution. It is important to highlight that this type of deliberative planning is intended to optimize the route the robot will follow; however, the time it takes to find such a route can be extended.

A. Global Representation and Configuration Space

The first stage in the strategy is to obtain the *Global Representation* of the workspace $\mathcal{W}$ (the environment where the robot will move). This representation involves obtaining a map or sketch with complete information about the work environment, including the positions of static obstacles. Therefore, the first step is to have the environment, which, in this case, is created in CoppeliaSim. The advantage of using a simulated environment is that multiple environments can be created to evaluate different features.

In Figure 2.16, a 5×5 meter environment enclosed by walls can be seen. The dimensions of the floor may vary, but it is expected to maintain a square configuration according to how the algorithm was developed. Subsequently, the global environment representation is obtained in Python from an orthogonal image of the space in CoppeliaSim through a vision sensor placed at the top of the scene (Figure 2.17a). The camera resolution was set to 256×256 as it was not necessary to have a high resolution to identify the obstacles as can be seen in Figure 2.17b. Once the image of the complete workspace is in Python, the connection with the simulator is no longer necessary, as from this point forward all processing for obtaining the path will be done offline.

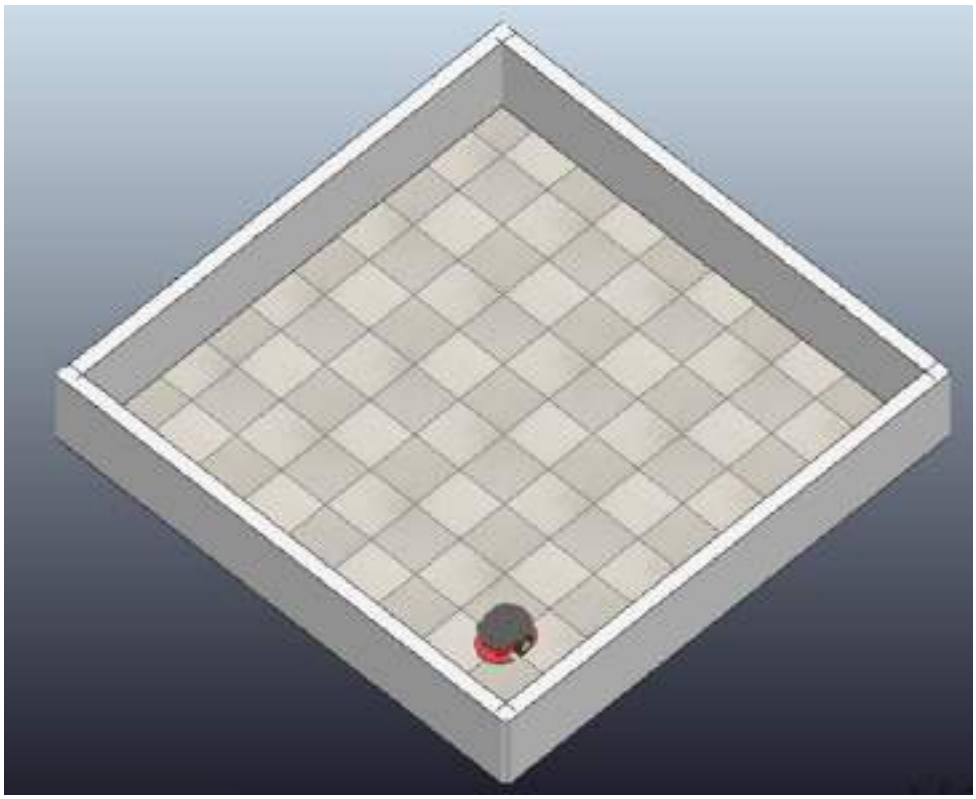
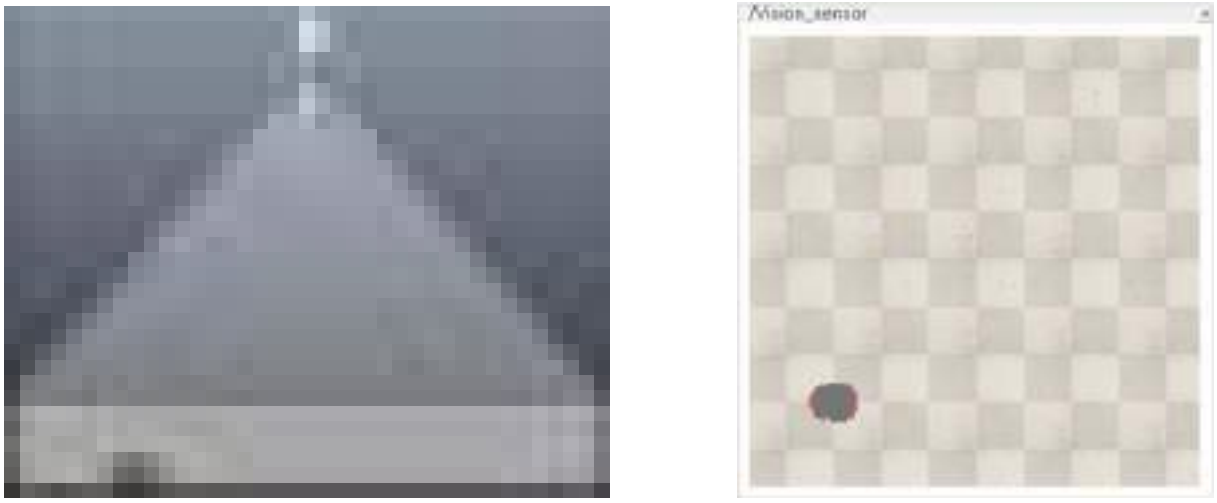


Figure 2.16: Workspace created in CoppeliaSim



(a) Front view of the environment without walls

(b) Vision sensor image (256 × 256)

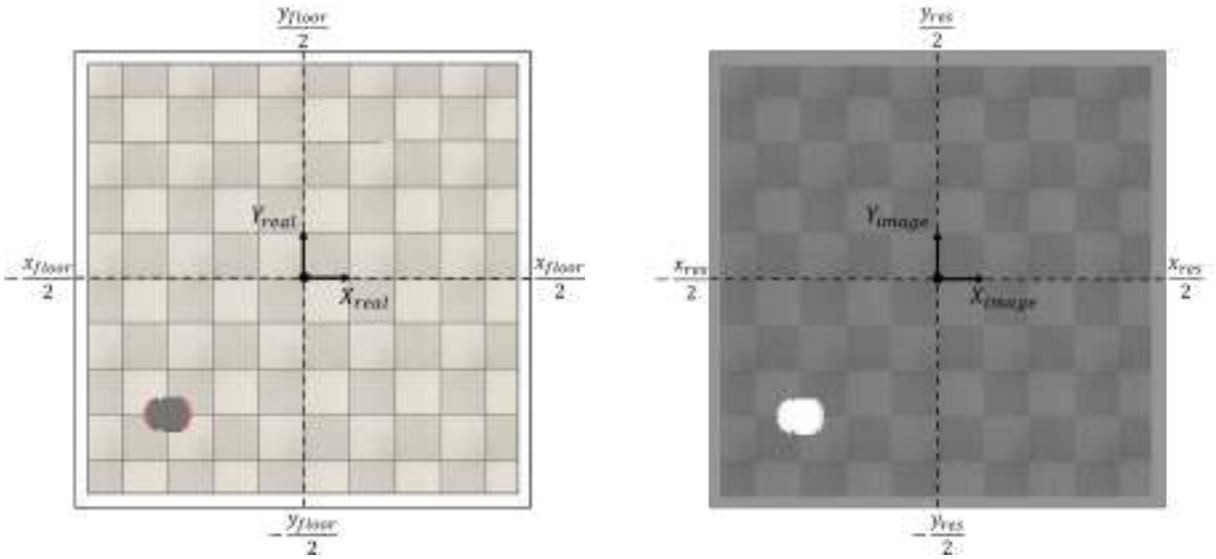
Figure 2.17: Vision Sensor placed in CoppeliaSim to obtain a Global Representation of the space

It is important to consider that global planning and path optimization will be done on the image. Therefore, it is necessary to determine the equivalence between the real coordinate system within CoppeliaSim as shown in Figure 2.18, which is continuous (Figure 2.18a), and the coordinate system of the image, which is discrete (Figure 2.18b). The equivalence will be determined by the following equations:

$$\begin{aligned} X_{image} &= X_{real} \cdot \frac{x_{res}}{x_{floor}} \\ Y_{image} &= Y_{real} \cdot \frac{y_{res}}{y_{floor}} \end{aligned} \tag{2.19}$$

where (X_{image}, Y_{image}) are the coordinates in the image within the range $[-\frac{x_{res}}{2}, \frac{x_{res}}{2}]$ and $[-\frac{y_{res}}{2}, \frac{y_{res}}{2}]$, while (X_{real}, Y_{real}) are the real coordinates in the environment within the range $[-\frac{x_{size}}{2}, \frac{x_{size}}{2}]$ and $[-\frac{y_{size}}{2}, \frac{y_{size}}{2}]$. Additionally, (x_{res}, y_{res}) is the resolution of the image and (x_{floor}, y_{floor}) are the real length and width dimensions of the floor in meters.

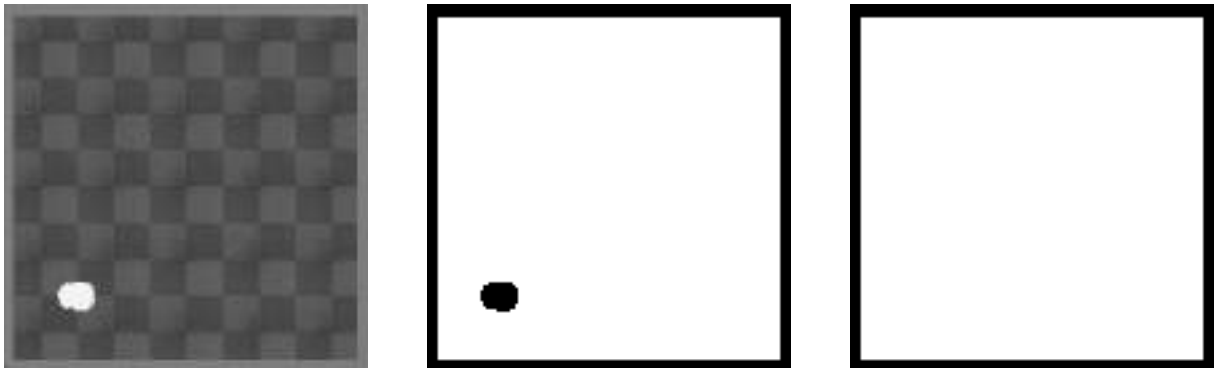
The global representation is then converted into a configuration space $\mathcal{C}$*-space* (as described in Section 1.2, *Configuration Space* subsection), transforming the problem of global path planning for the robot into the problem of planning the movement of a single point. In this process, the configuration space where a collision occurs ($\mathcal{C}$-obstacle or $\mathcal{CO}$) and the free configuration space ($\mathcal{C}_{free}$) are identified. To achieve this, the image obtained is converted to grayscale (Figure 2.19a), which is then binarized using a threshold-based function. Pixels with values above the threshold are set to 0 (black) to represent the obstacles in $\mathcal{CO}$, while those pixels below or equal to the threshold are set to 255 (white) representing the available space (Figures 2.19b and 2.19c).



(a) Real coordinate system

(b) Image coordinate system

Figure 2.18: Equivalence of the Coordinate Systems between Workspace and Map image



(a) Grayscale workspace image in Python

(b) Binarized image of the workspace with robot

(c) Binarized image of the workspace without robot

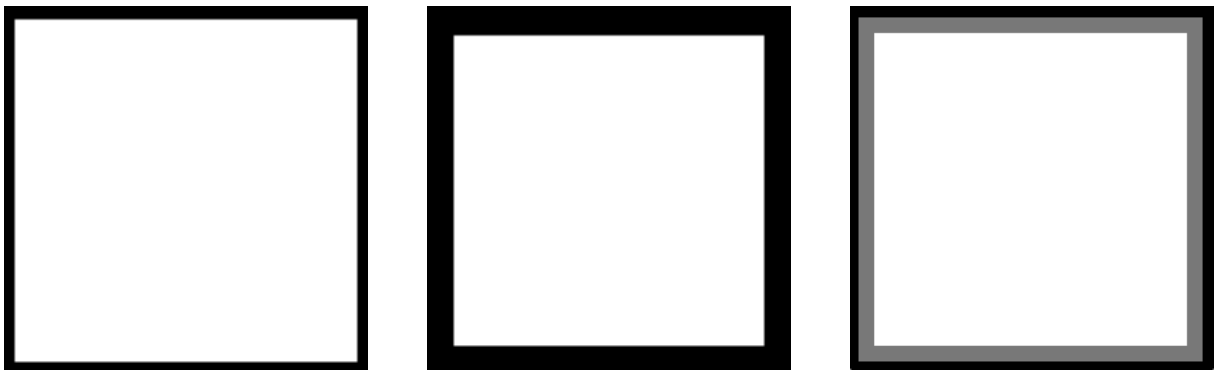
Figure 2.19: $\mathcal{C}$*-space* Creation Process. Black color represents $\mathcal{C}$-obstacle space and white color represents $\mathcal{C}_{free}$ space

A potential concern with this method is the ease with which $\mathcal{C}_{free}$ and $\mathcal{CO}$ can be separated. It should be noted that this advantage is indeed leveraged to obtain the $\mathcal{C}$*-space* both easily and quickly. As previously mentioned, the tasks of mapping in navigation are assumed to be entirely completed, and their implementation is not considered part of this strategy. Therefore, the environment in CoppeliaSim is specifically designed to facilitate this process, as the objective is not the mapping itself, but rather having the map. This ensures that the computational resources are primarily allocated to the planning and execution phases, rather than being diverted to preliminary mapping tasks. Additionally, this separation allows for a more efficient workflow, minimizing potential errors that could arise from concurrent mapping and navigation efforts.

However, simplifying the problem involves taking extra consideration since the $\mathcal{C}$-*space* treats the robot as a point, ignoring its dimensions. When moving through the workspace $\mathcal{W}$, it is evident that the robot's size needs to be taken into account. To address this, a process similar to the one shown in Figure 1.4 is applied, in which an extra safety margin is added to the obstacles so that the estimated route is not affected by the size of the robot and its proximity to any static object in the scene.

For example, in an image of 256×256 pixels representing a space of 5×5 meters, each pixel represents approximately 19.5 mm. If the maximum dimension of the robot is 455 mm (Figure 2.2), then the robot occupies a maximum space of 23×23 pixels. Considering that an obstacle is dilated on both sides, the necessary dilation in the obstacles would therefore be around 10-11 pixels or half the size of the robot. This ensures that the generated paths are feasible and do not lead to unintended collisions during real-world navigation.

This dilation process, illustrated in Figure 2.20, is implemented using Python's PIL library, specifically the `ImageFilter` module, to apply a minimum filter (`MinFilter`) with a kernel size of 23×23 pixels to the binarized image from Figure 2.20a. This filter replaces each pixel with the minimum value of its neighbors within the kernel area. Since the minimum value is 0, representing the color black, any pixel within the region where the filter is applied that has a neighboring black pixel will also be converted to black. In this example, the space occupied by the external wall is dilated by 11 pixels, which represents an extra safety margin of 21.45 centimeters. Figure 2.20b shows the result of the obstacle dilation, representing the $\mathcal{C}$-*space* in which the path planning will be developed, while Figure 2.20c visually shows the difference between the actual obstacles and the added dilation.

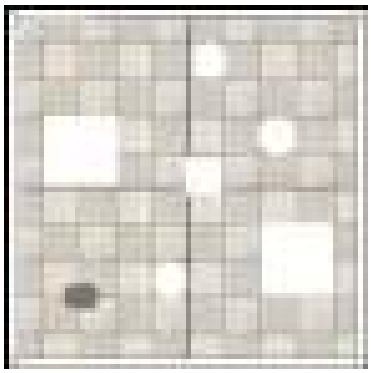


(a) Orginal $\mathcal{C}$-*space* (b) $\mathcal{C}$-*space* final result (c) $\mathcal{C}$-*space* with dilatation

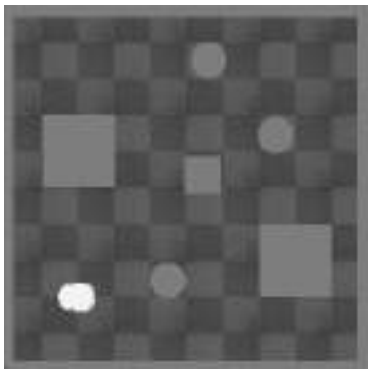
Figure 2.20: $\mathcal{C}$-*space* with Obstacles Dilatation

This process of global representation and creation of the $\mathcal{C}$-*space* is easily applicable to different environments with various obstacles, as illustrated in Figure 2.21. The generalization for global configuration space representation can be summarized as follows:

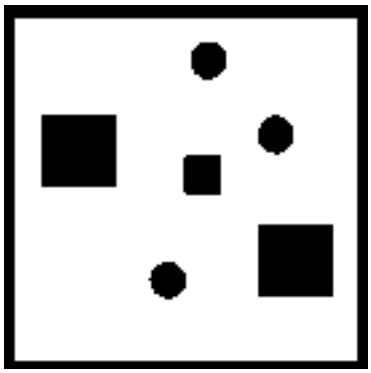
1. Create a new environment with static obstacles in CoppeliaSim (Figure 2.21a).
2. Acquire an image of the environment in Python through the vision sensor (Figure 2.21b).
3. Binarize the image to identify $\mathcal{C}_{free}$ in white and $\mathcal{CO}$ in black (Figure 2.21c).
4. Dilate obstacles (Figures 2.21d and 2.21e).



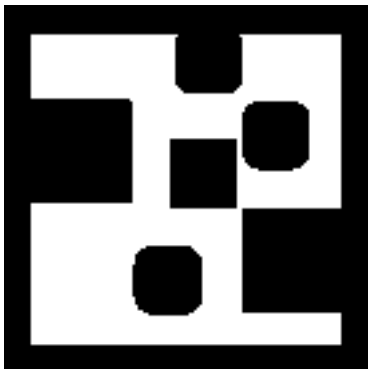
(a) Original environment



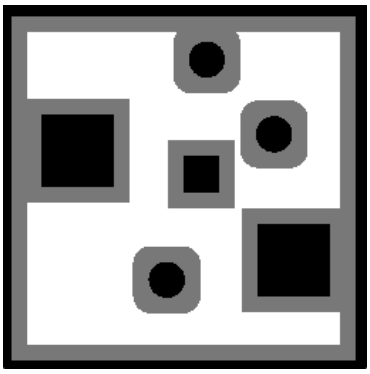
(b) Image acquired in Python



(c) Image binarized



(d) $\mathcal{C}$-*space* with dilated obstacles



(e) Visualization of obstacle dilation

Figure 2.21: Generalization for global configuration space representation

B. Grey Wolf Optimizer

Once the representation of the $\mathcal{C}$-*space* is obtained, global planning is conducted. For this, the meta-heuristic algorithm *Grey Wolf Optimization* is used. As mentioned in Section 1.2 (*Global & Local Path Planning* subsection), the problem of path planning can be approached and solved as an optimization problem since it essentially involves finding the best route between two points while minimizing or maximizing some criteria, such as distance, time, cost, or energy

consumption. In this case, as it is a simulation, the most coherent optimization criterion is the minimization of the total distance. Thus, the GWO algorithm receives as inputs the configuration space image, the initial position of the robot, the desired final position of the robot, the optimization function, and the algorithm's parameters such as the search agents size (number of wolves), the number of control points for each search agent, and the maximum number of iterations. The expected output is the positions of control points that describe the optimized path the robot should follow.

Given that GWO will be applied to find the shortest path between the initial and goal points, the fitness function (F) to be optimized is presented in Eq. 2.20, which describes the sum of the Euclidean distances between points in the trajectory, as detailed below:

$$F = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} + \sqrt{(X_2 - X_3)^2 + (Y_2 - Y_3)^2} + \ldots + \sqrt{(X_{n-1} - X_n)^2 + (Y_{n-1} - Y_n)^2} \tag{2.20}$$

where n represents the total number of waypoints in the route including start and goal points. (X_i, Y_i) represents the coordinate of a point on the trajectory and (X_{i+1}, Y_{i+1}) is the coordinate of the next point on the trajectory. The resulting path will then be composed of line segments created by connecting two consecutive points, as illustrated in Figure 2.22. The number of waypoints is an input parameter of the algorithm, and its determination is done empirically according to the size and complexity of the space.

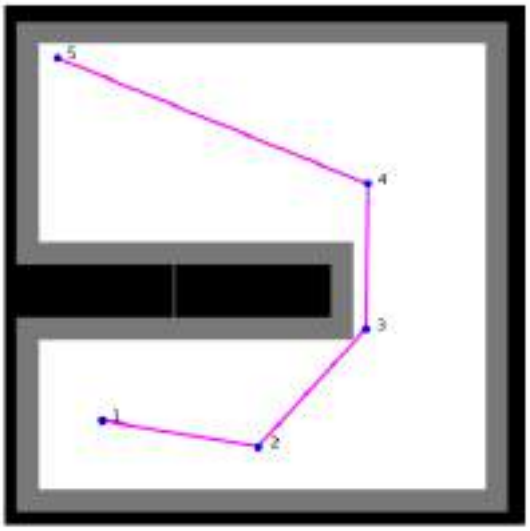


Figure 2.22: Building the Resulting Path Process where consecutive points are connected by a straight line to create the complete path

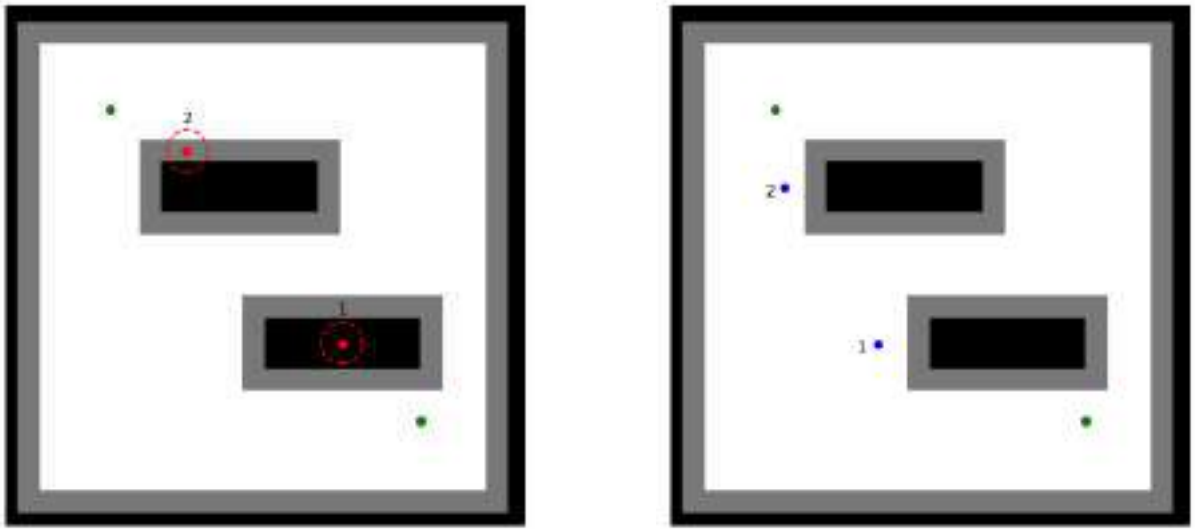
As presented in Section 2.2 (*Grey Wolf Optimization* subsection), a wolf is a possible solution to the function F and represents a search agent (SA). Therefore, each wolf will be composed of a set of coordinates in the following way:

$$\begin{aligned} SA = [X_1 \quad X_2 \quad \dots \quad X_{n-1} \quad X_n \\ Y_1 \quad Y_2 \quad \dots \quad Y_{n-1} \quad Y_n]. \end{aligned} \tag{2.21}$$

where (X_1, Y_1) denotes the coordinates of the initial point, and (X_n, Y_n) represents the coordinates of the goal point. These two points are fixed within the optimizer and will not change during the iterations (for obvious reasons).

Due to the stochastic nature of the GWO algorithm, the wolves or candidate solutions are initially distributed randomly within the search space. However, in the context of this path planning, such randomness is necessarily constrained by the places where a robot can be or move, as there are configurations restricted by the $\mathcal{C}$-obstacle. The necessary conditions for the waypoint positions included for each search agent are:

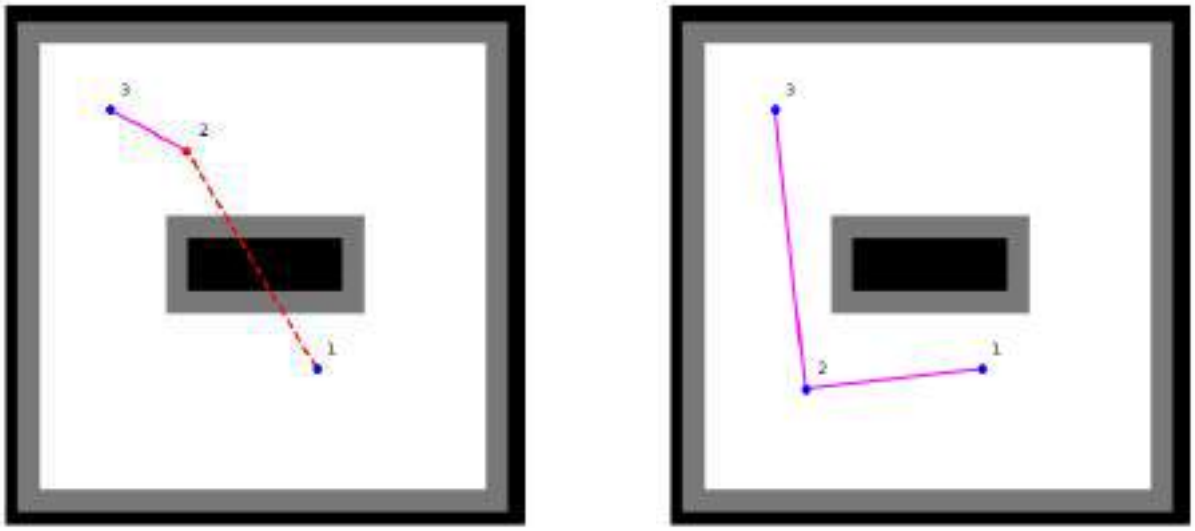
1. The waypoint falls within the range of the configuration space $[-\frac{x_{res}}{2}, \frac{x_{res}}{2}]$ and $[-\frac{y_{res}}{2}, \frac{y_{res}}{2}]$.
2. The waypoint does not fall on an obstacle (Figure 2.23).
3. The straight line that connects two consecutive waypoints in the trajectory does not cross any obstacles (Figure 2.24).



(a) Waypoints falling over obstacles

(b) New free points not falling over obstacles

Figure 2.23: Validation to avoid Waypoints over obstacles



(a) Waypoints falling over obstacles

(b) New free points not falling over obstacles

Figure 2.24: Validation to avoid path crossing over obstacles.

In the initialization process, m number of search agents and n required waypoints are created, as illustrated in the example of Figure 2.25, where five search agents with four waypoint positions each are created. During this creation process, the first two verifications are made, ensuring that all points fall within the range and none fall on an obstacle. The waypoint that meets these conditions is called a *free point*. It is important to mention that the created waypoints are consecutive, meaning they are labeled as $1, 2, 3, \ldots, n$, representing the order in which the robot will traverse them.

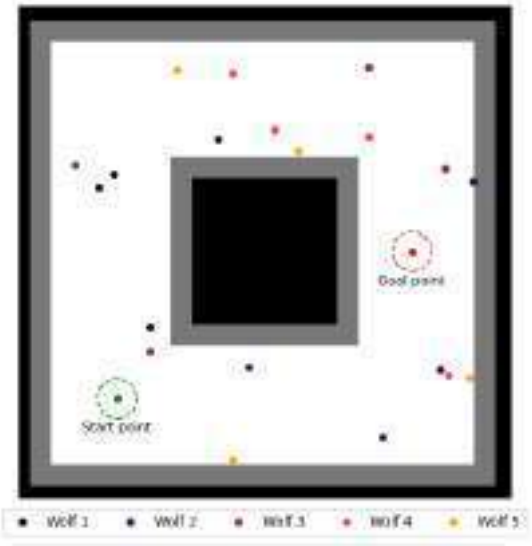


Figure 2.25: GWO Initialization Process. Each color represents a wolf or possible solution composed of four waypoints

Given that it is an iterative process where the positions and coordinates of the waypoints will constantly change according to the equations described by the GWO algorithm, in each iteration, it is verified that each search agent continues to meet the aforementioned conditions in this order:

1. All waypoints of the search agent must be within the image boundaries. If after estimating the new positions with Eq. 2.8, any coordinate falls outside this range, it is automatically adjusted to the lower limit or the upper limit, depending on which limit was exceeded.

2. All waypoints of the search agent must be within $\mathcal{C}_{free}$, meaning they do not fall on an obstacle. If a waypoint does not meet this condition, another free point is randomly generated.

3. There should be no obstacle in the straight path between two consecutive points. If there is an obstacle between the starting point (fixed) and the first waypoint, another free point is generated. If there is an obstacle between two consecutive waypoints, another free point is generated to replace the second waypoint. If there is an obstacle between the last waypoint and the goal point (fixed), another free point is generated.

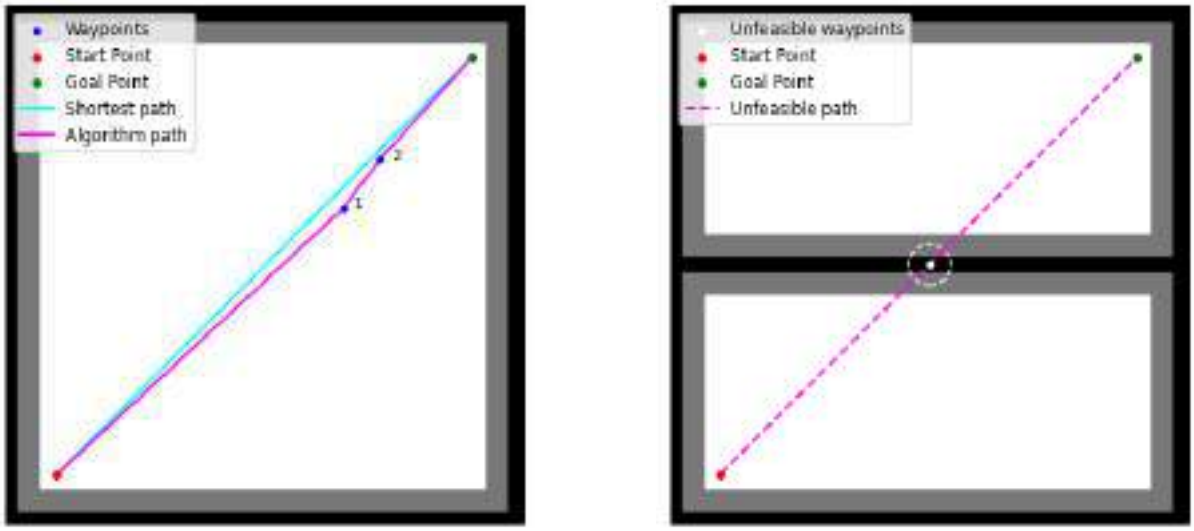
It should be noted that the three considered constraints directly reduce the performance of the GWO algorithm, as the optimization process is interrupted by creating a random free point each time a condition is not met, and the originally calculated positions are not used.

Although these verifications confirm that all the waypoints are free points, they do not guarantee that the created path is valid, as it does not ensure that the straight line between two consecutive waypoints is free of obstacles. This is done to reduce computational cost since the verification would need to change from an *if* statement to a *while* loop, which could suddenly increase processing time without the certainty that a point exists that truly meets the condition. The lack of assurance is compensated by the high exploration of the space provided by the algorithm and the randomness that intervenes during each iteration.

Once verification is done on the search agents, the value of the function F is estimated for each "wolf" (Eq. 2.20). Based on this evaluation, the three best solutions, α, β, and δ, are chosen (those with the lowest F value). If a wolf still has an invalid sequence of waypoints, its fitness calculation is directly adjusted to infinity to prevent the algorithm from considering it. Finally, the coordinate vector of each agent is updated with Eq. 2.8 or Eq. 2.9, creating new search agents. The best solution (α) will be then updated in each iteration until the maximum number of iterations is reached, to find the shortest path.

The algorithm should be capable of finding a free path (if it exists) in $\mathcal{C}_{free}$ that connects the robot's initial and final positions, or it should notify if no such path exists. If the final α score is infinite, it means that the algorithm did not find a feasible path. The expected output of this stage is the set of sequential positions that describe the local goals the robot must follow during its motion, i.e., the discretized route described by the waypoints on the path.

Figure 2.26 illustrates the expected result after executing the algorithm: a route composed of two waypoints between the initial point and the goal point after running 100 iterations with 12 wolves. Despite being a trivial case in a space without obstacles, it is used to verify that the algorithm is capable of finding a path with the shortest distance between two points (the straight line connecting them). In Figure 2.26a, the initial point is located at $(-2, -2)$ and the goal point at $(2, 2)$, so the shortest distance is 5.657 and the algorithm found a route of 5.691. In the case of Figure 2.26b, the path is not created because no route can connect both configurations.



(a) Example of path found

(b) Example of unfeasible path

Figure 2.26: Examples of Running the Global Path Planner. Subfigure (a) shows the comparison between the real shortest path and the one found by the algorithm; subfigure (b) shows the unfeasibility of creating a path

This concludes the global planning process. A flowchart showing the entire procedure conducted by the global planner is presented in Figure 2.27. It is important to highlight that this is performed entirely offline, meaning before the robot starts moving. The following section explains the local planning part where the robot begins to move.

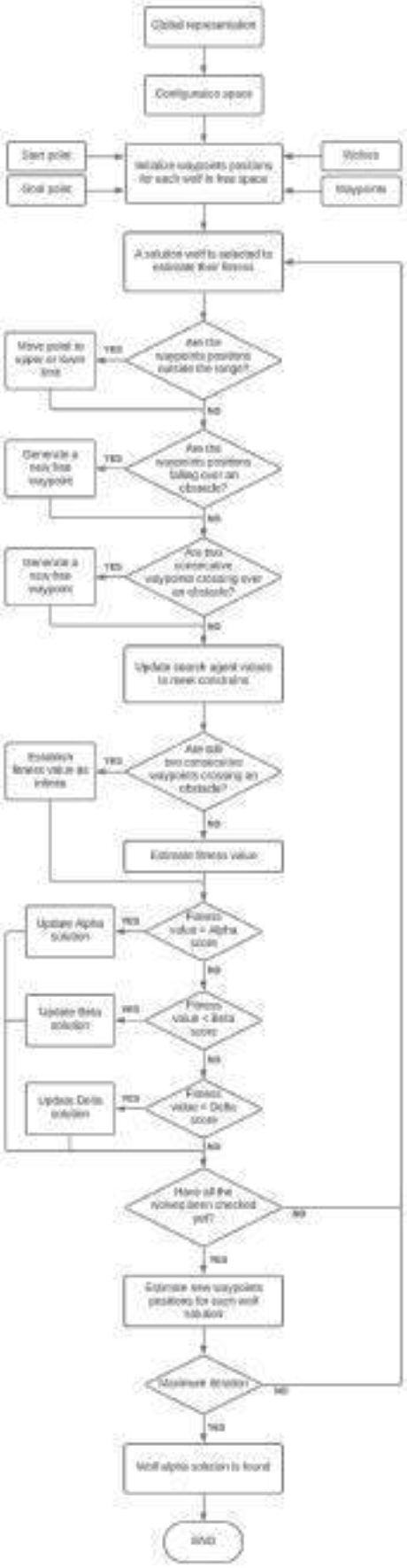


Figure 2.27: Global Planning Subsystem Flowchart

2.3.2. Local Planning Subsystem

When the local goals of the global path are determined, the *Local Planning Subsystem* presented in Figure 2.15 uses them to start moving the robot. In fact, this subsystem includes both the tasks of local perception and representation as well as motion control. Therefore, the strategy proposes using a reactive sense-to-avoid system that combines information from ultrasonic sensors with optical flow estimations (as studied in Section 1.4, *Perception System Overview* subsection) to safely avoid unknown dynamic obstacles while the WMR moves along the global path. The expected overall performance of this strategy has been previously described in Figure 1.7 (Section 1.2), which illustrates the scenario where a local path is generated on-line to avoid an obstacle that coincidentally crosses the global path as the AMR moves towards the goal.

Accordingly, the system will operate based on two fuzzy logic controllers designed for the robot's movement (as explained in Section 2.2.3): 1) A basic path-following controller known properly as the Fuzzy Logic Controller (FLC), based on the deliberative perspective of global planning, and 2) an Obstacle Avoidance Controller (OAC), based on the reactive perspective of local planning. Consequently, there are two modes of operation, which are illustrated in the control diagram in Figure 2.28.

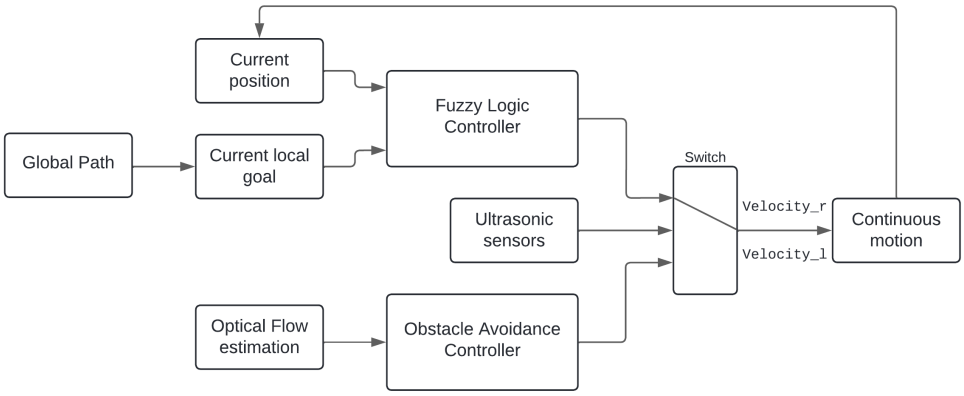


Figure 2.28: Sense-to-avoid System including a standard Fuzzy Logic Controller and an Obstacle Avoidance Controller

The FLC is the standard mode of operation that uses the current position of the robot and the next local goal from the global path as inputs, while the OAC relies on information from optical flow estimates. The OAC is activated when the ultrasonic sensors detect obstacles closer than the defined threshold, acting as a switch block. In both modes of operation, the output consists of motion control commands represented by the velocities of the left wheel (*Velocity_l*) and the right wheel (*Velocity_r*). The details and functions of each mode are explained below.

A. Fuzzy Logic Controller

The first mode uses the basic controller for path following (as described in Figure 1.13) based on a Mamdani fuzzy logic controller. The global path is composed of several waypoints that represent local goals through which the robot must move to reach the final goal point. Therefore, the purpose of the FLC is to estimate the appropriate velocity for the right and left wheels of the system to move the robot along the global path. The controller uses the distance (d) and the heading angle error (ψ) between the robot's current position and the current waypoint on the path as inputs (see Section 1.4, *Kinematics System Overview*). The specific values for the inputs are calculated using the following mathematical equations:

$$d = \sqrt{(X_{goal} - x_t)^2 + (Y_{goal} - y_t)^2} \tag{2.22}$$

$$\psi = \theta_t - \theta_{attack} \tag{2.23}$$

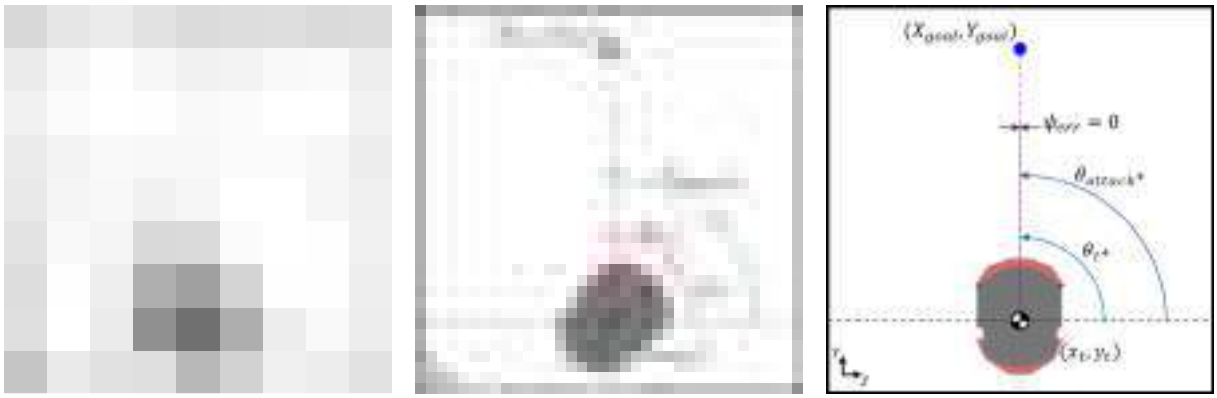
with

$$\theta_{attack} = \arctan\left(\frac{Y_{goal} - y_t}{X_{goal} - x_t}\right) \tag{2.24}$$

where (X_{goal}, Y_{goal}) are the coordinates of the current local goal and (x_t, y_t) are the coordinates of the WMR at the time of the estimation. Additionally, θ_{attack} is the angle of attack between the robot and the next goal position, while θ_t is the robot's angle relative to the global reference frame.

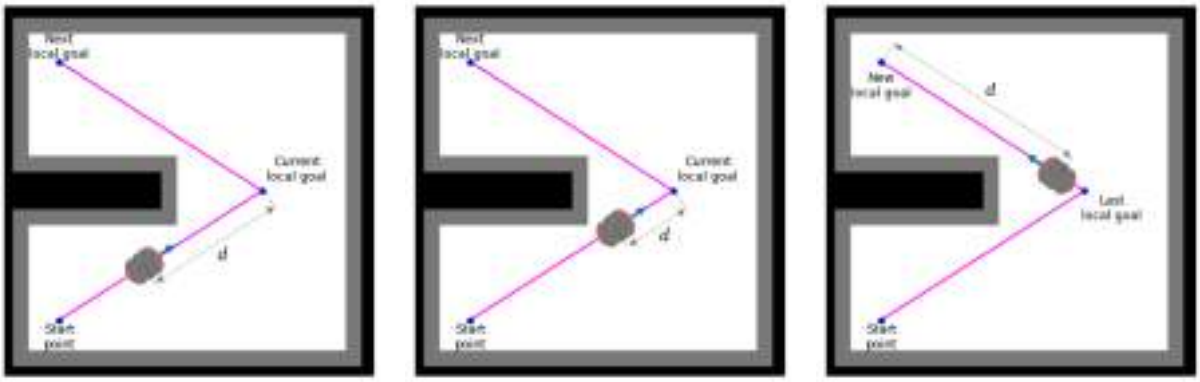
The objective is to make the heading angle error ψ as close to 0 as possible within the range of $[-180°, 180°]$, indicating that the robot is moving in the correct direction (Figure 2.29). As the robot approaches the local goal, the distance d will decrease. When this distance becomes sufficiently small (a value determined empirically), the local goal will be updated, and the robot will move towards the new target (Figure 2.30).

The fuzzy logic controller is defined using the Python library `scikit-fuzzy`, which requires parameterizing the inputs and outputs with membership functions. In this context, the distance d calculated using Eq. 2.22 can be classified according to how close the robot is to the local goal. Therefore, the distance values are defined as *zero* (Z), *near zero* (NZ), *near medium* (NM), *medium* (M), *near far* (NF), *far* (F) and *very far* (VF), as input membership functions. In total, *distance* input is categorized in seven different labels as can be seen in Table 2.1.



(a) Angle error $\psi > 0°$ (b) Angle error $\psi < 0°$ (c) Angle error $\psi = 0°$

Figure 2.29: Angle error as input to Fuzzy Logic Controller



(a) Distance d (b) Distance d decreases (c) New local goal

Figure 2.30: Distance as input to Fuzzy Logic Controller

Table 2.1: Membership Function Values for *distance*

Distance Category	**Type**	**Membership Function Range** $[m]$
Zero (Z)	Trapezoidal	[0,0, 0.05, 0.1]
Near Zero (NZ)	Triangular	[0.05, 0.1, 0.15]
Near Medium (M)	Triangular	[0.1, 0.5, 1]
Medium (M)	Triangular	[0.5, 1, 1.5]
Near Far (NF)	Triangular	[1, 2, 3]
Far (F)	Triangular	[2, 4, 6]
Very Far (VF)	Trapezoidal	[4, 6, 7, 7]

The closer the robot is to the next waypoint, the more precise its movement needs to be, as these control points not only indicate the route but also represent safety positions that help avoid static obstacles. Therefore, as can be seen in Figure 2.31, the membership functions for *distance* input classify values closer to 0 more precisely, whereas, for greater distances, such precision is not as necessary. Beyond the defined membership functions, it was empirically determined that when the distance is less than 0.05 meters, it is considered that the robot has reached its destination, and the local goal is updated (or the robot stops if it is the global goal point).

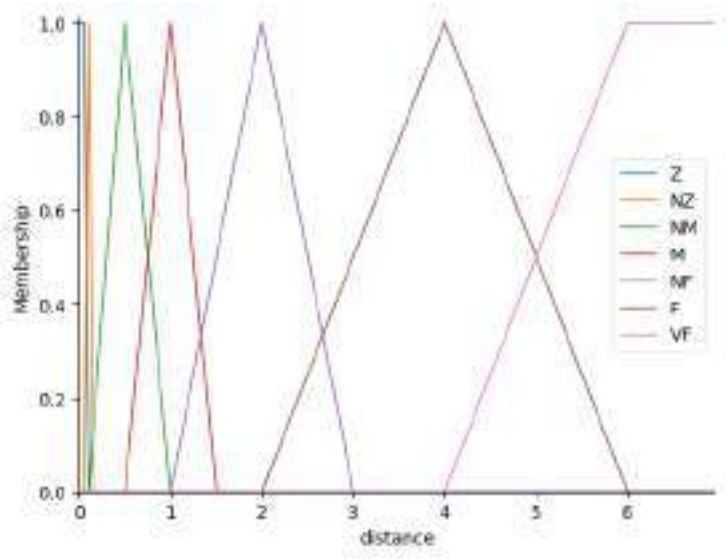


Figure 2.31: Membership Functions for *distance* categories: Zero (Z), Near Zero (NZ), Near Medium (NM), Medium (M), Near Far (NF), Far (F) and Very Far (VF)

On the other hand, the second antecedent or input angle error (*angle_err*) is calculated in degrees using Eq. 2.23. For this purpose, this angle can be classified as *negative* (N), *small negative* (SN), *negative near zero* (NNZ), *zero* (Z), *positive near zero* (PNZ), *small positive* (SP), and *positive* (P). Accordingly, the fuzzy membership functions are created empirically and presented in Table 2.2.

Table 2.2: Membership Function Values for *angle_err*

Angle Category	**Type**	**Membership Function Range** [°]
Negative (N)	Trapezoidal	[-180, -180, -135, -90]
Small Negative (SN)	Triangular	[-135, -90, -15]
Negative Near Zero (NNZ)	Triangular	[-45, -15, 0]
Zero (Z)	Triangular	[-15, 0, 15]
Positive Near Zero (PNZ)	Triangular	[0, 15, 45]
Small Positive (SP)	Triangular	[15, 90, 135]
Positive (P)	Trapezoidal	[90, 135, 180, 180]

In Figure 2.32, which illustrates the *angle_err* membership functions, it can be observed that there is a finer division in the classification of angles closer to 0. This is because greater precision is desired when the robot is nearly aligned with its correct orientation. In contrast, when the angle is very large, the goal is to rotate the robot more degrees without requiring as much accuracy.

Once the membership functions of the antecedents are identified, the functions to describe the outputs of the controller are defined: the angular velocity of the right (ω_R) and the left (ω_L) wheels. This definition should be linked to the desired linear velocity at which the robot should move.

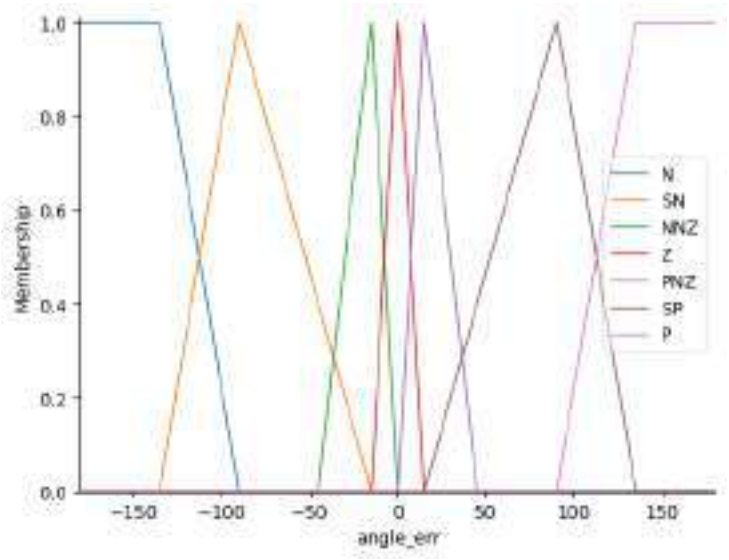


Figure 2.32: Membership Functions for *angle_err* categories: Negative (N), Small Negative (SN), Negative Near Zero (NNZ), Zero (Z), Positive Near Zero (NPZ), Small Positive (SP), and Positive (P)

The maximum safe speed for an AMR in an indoor space can depend on various factors, such as the type of robot, its responsiveness to braking or direction change commands, the accuracy of its sensors, and the environmental conditions (obstacles, presence of people, etc.). On one hand, since this is a simulation, there are no complications with the robot's maneuverability or the functioning and accuracy of the sensors. However, the robot will move in a congested space full of static and dynamic obstacles where it must have sufficient reaction capability. Therefore, as this is a strategy under development, more conservative and relatively low speeds are established.

Considering the above, the desired speed of the robot is set at 0.15 m/s. Using Eq. 1.9, the limit of the angular velocity is estimated to be 1.6 rad/s, considering a wheel diameter of 0.195 m (see Figure 2.2):

$$v = r \cdot \omega \Rightarrow \omega = \frac{v}{r}$$

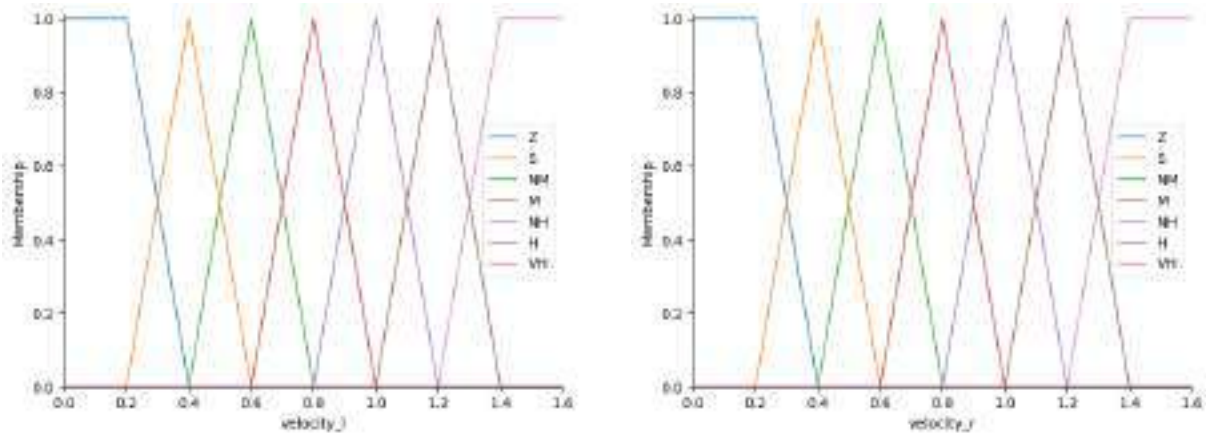
$$\omega_{max} = \frac{v_{max}}{r} = \frac{0.15}{0.195/2} = 1.54 \approx 1.6 \; rad/s$$

As a differential WMR, the robot turns based on the difference in wheel velocities: if $\omega_R > \omega_L$, the robot turns left, while if $\omega_R < \omega_L$, it turns right. It is important to note that negative velocities are not considered for the FLC, ensuring the robot always moves forward but limiting its ability to make sharp turns (or rotate on its axis). In this context, the angular velocities as outputs can be classified as *zero* (Z), *small* (S), *near medium* (NM), *medium* (M), *near high* (NH), *high* (H), and *very high* (VH), within a range of $[0, 1.6]$. Additionally, a total stop can be commanded in emergency situations, such as collisions or when the robot reaches its final goal.

Table 2.3 describes the membership function values for the angular velocities of ω_L (*velocity_l*) and ω_R (*velocity_r*) for the left wheel (Figure 2.33a) and the right wheel (Figure 2.33b), respectively. As can be seen in Figure 2.33, the same functions were defined for both wheels.

Table 2.3: Membership Function Values for ω_L and ω_R

Velocity Category	**Type**	**Membership Function Range** $[rad/s]$
Zero (Z)	Trapezoidal	[0, 0, 0.2, 0.4]
Small (S)	Triangular	[0.2, 0.4, 0.6]
Near Medium (NM)	Triangular	[0.4, 0.6, 0.8]
Medium (M)	Triangular	[0.6, 0.8, 1]
Near High (NH)	Triangular	[0.8, 1, 1.2]
High (H)	Triangular	[1, 1.2, 1.4]
Very High (VH)	Trapezoidal	[1.2, 1.4, 1.6, 1.6]



(a) Angular velocity ω_L

(b) Angular velocity ω_R

Figure 2.33: Membership Functions for *wheels velocity* categories: Zero (Z), Small (S), Near Medium (NM), Medium (M), Near High (NH), High (H), and Very High (VH)

With the definition of both the antecedents and the consequents, it is possible to define the fuzzy rules of the controller with the following structure:

IF (*distance* AND *angle_err*) THEN (*velocity_l* AND *velocity_r*)

The total number of rules to be defined is equal to the total number of all possible combinations between the labels of *distance* (7) and those of *angle_err* (7), resulting in 49 rules. These combinations can be seen in Table 2.4.

It would be impractical to describe all the rules, but their definition is explained. Generally, three states can be identified according to the robot's angle error ψ (see Figure 2.29): 1) $\psi < 0$; 2) $\psi > 0$; and 3) $\psi = 0$. Thus, it suffices to define the rules for case 1 (N, SN, and NNZ), as the rules for case 2 (P, SP, and PNZ) are equivalent, with the wheels' speeds inverted.

Table 2.4: Fuzzy Logic Controller Rules

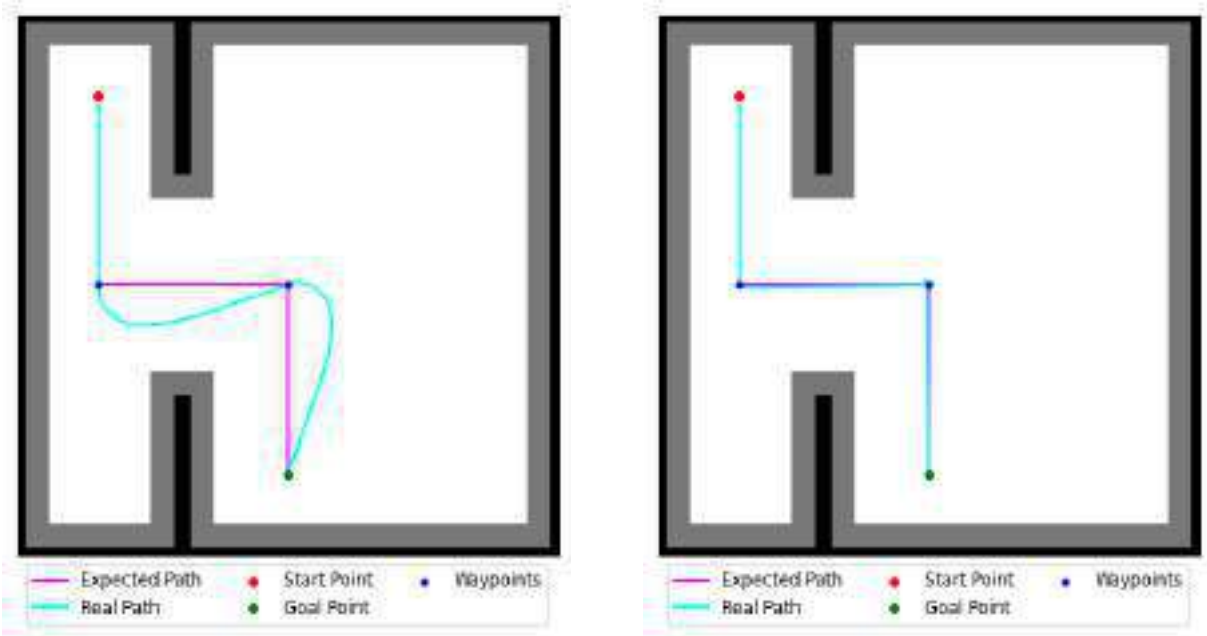
		Distance d						
		Z	NZ	NM	M	NF	F	VF
Angle error ψ								
N	left	Z	Z	Z	Z	Z	Z	Z
	right	VH	VH	VH	VH	VH	VH	VH
SN	left	Z	Z	S	S	S	S	S
	right	M	NH	H	H	H	H	H
NNZ	left	Z	S	S	NM	NM	M	M
	right	NM	NM	M	NH	NH	H	H
Z	left	Z	S	NM	M	NH	H	VH
	right	Z	S	NM	M	NH	H	VH
PNZ	left	NM	NM	M	NH	NH	H	H
	right	Z	S	S	NM	NM	M	M
SP	left	M	NH	H	H	H	H	H
	right	Z	Z	S	S	S	S	S
P	left	VH	VH	VH	VH	VH	VH	VH
	right	Z	Z	Z	Z	Z	Z	Z

For instance, if the angle error is large, either *negative* (N) or *positive* (P), the robot must orient itself as quickly as possible. In these cases, the difference between the wheel speeds should be the highest, allowing the robot to rotate without significant displacement. If the angle error ψ is negative (N), the angular speeds are set to $\omega_R = VH$ and $\omega_L = Z$ to turn left. Conversely, if the angle is positive (P), the speeds are $\omega_R = Z$ and $\omega_L = VH$ to turn right, regardless of the distance. As the angle becomes smaller, the difference in wheel speeds is maintained but less pronounced to avoid oscillation due to lack of precision. Thus, for *small negative* (SN) or *small positive* (SP) angles, the wheel speeds that were *zero* (Z) are slightly increased to *small* (S), and those that were *very high* (VH) are reduced to *high* (H), *near high* (NH), or even *medium* (M). A similar process occurs for *negative near zero* (NNZ) or *positive near zero* (PNZ) angle errors. Finally, if the angle error is *zero* (Z), the robot no longer needs to orient itself and can move straight towards its destination, so the velocities of both wheels are equal.

Similarly, the influence of *distance* is considered. If the robot is *far* (F) or *very far* (VF) from the next waypoint, it has more freedom to move, allowing its velocity to be as high as possible. As the distance to the waypoint decreases, movements need to be more precise and slower. For instance, when the distance is *medium* (M), the speed is also *medium* (M). When both the distance and the angle error are *zero* (Z), the speed is also *zero*.

As the rules were defined, the expected path-following process allows the robot to orient and move simultaneously, leveraging the benefits of fuzzy logic. However, as mentioned earlier, the robot cannot make sharp turns, which could be challenging in complex and confined workspaces. Given that this proposed path-following system aims to be accurate and safe, and the global path found may not allow for significant deviation from the original route, an additional operation is added to orient the robot before moving to the next position. Specifically, each time the robot reaches a local goal, it will orient itself on its axis by rotating one wheel at 0.3 rad/s and the other in the opposite direction until the angular error with respect to the next waypoint is zero.

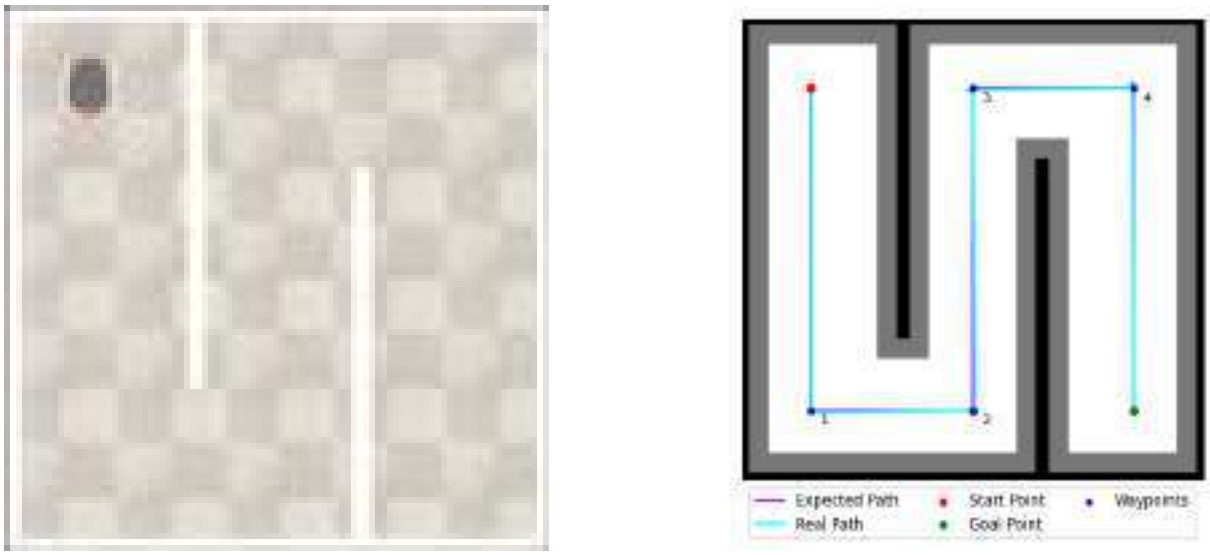
Figure 2.34 shows a comparison of the same route followed using the original FLC as it was defined (Figure 2.34a) and the FLC with orientation at each waypoint (Figure 2.34b). As observed, the original FLC deviates significantly from the route when orienting to the next waypoint since it is simultaneously moving forward. This deviation is corrected by adding automatic orientation before proceeding to the next waypoint, thereby reducing the angular error to zero. This does not replace the effect of the angular error in the controller, as the robot will only orient itself directly at each local goal and not throughout the entire trajectory. Therefore, it will still need to orient and move simultaneously when deviating from the original route due to unknown dynamic obstacles.



(a) Original FLC (b) Original FLC with automatic orientation

Figure 2.34: FLC proposed with Automatic Orientation

The operation of the FLC proposed is illustrated in Figure 2.35. A test environment was designed in a 5×5 meter space within CoppeliaSim (Figure 2.35a), and a path was generated for the robot to follow. This path was not created using the global planner presented in the previous section, as the current purpose is solely to demonstrate the functionality of path following. Based on the expected path compared to the route followed by the robot (Figure 2.35b), it can be seen that the path following is effective, with minimal deviation. The expected path covers a distance of 14 meters, while the robot actually traveled 14.236 meters, representing just a +1.02% deviation.



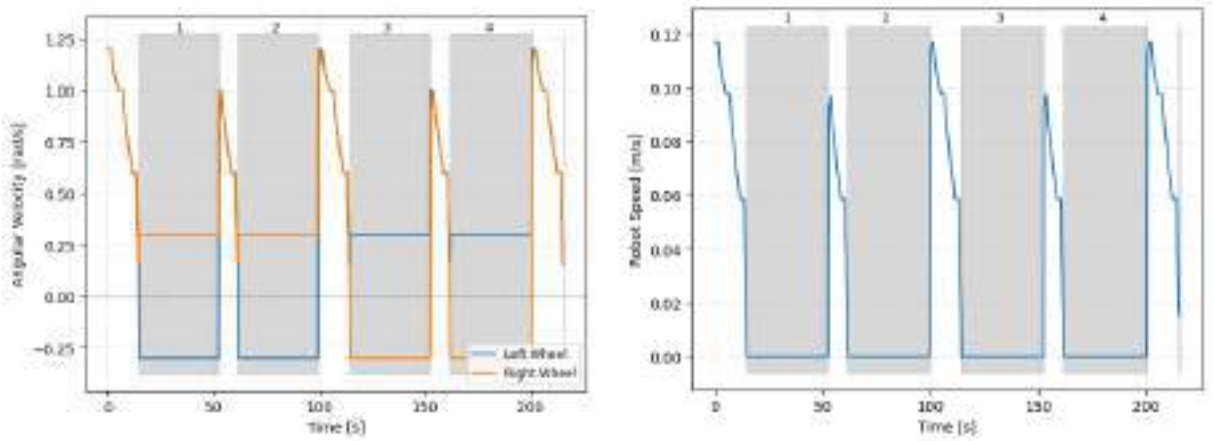
(a) Environment created in CoppeliaSim

(b) Expected path vs Real path

Figure 2.35: Environment for the Testing of the FLC proposed

Moreover, Figure 2.36 illustrates the angular velocities of the robot's wheels (Figure 2.36a) and the linear velocity of the robot during the path following (Figure 2.36b), where the two scenarios faced by the controller can be identified. The first scenario occurs when the robot orients itself towards the next local goal, which happens when the angular velocities of the wheels are opposite, and the linear velocity of the robot is 0 (indicated by the shaded areas, each number referring to the waypoints on the path in Figure 2.35b). The second scenario, where the robot moves towards the goal, is observed when the robot's linear velocity is greater than 0 and both angular velocities are equal. It can also be seen how initially the speed is high and decreases as the robot approaches the desired position in each segment.

Up to this point, the path following using the Fuzzy Logic Controller has been described to complete the global path in a completely static scenario. Next, the process that would be activated when unconsidered and dynamic obstacles interfere with the robot's path is described.



(a) Wheels' Angular Velocity

(b) Robot Speed

Figure 2.36: Robot Velocity during the navigation to test the Fuzzy Logic Controller. The shaded areas indicate the moments when the robot is orienting itself toward the next local goal

B. Obstacle Avoidance Controller

The second mode of operation of the Local Planning subsystem is another fuzzy logic controller called the Obstacle Avoidance Controller (OAC). This controller is activated in a switch-like manner when an object is detected at a distance closer than a defined threshold, using inputs from ultrasonic sensors. When this occurs, the OAC uses information provided by optical flow estimation to determine the direction to take to avoid colliding with new dynamic obstacles that appear in its trajectory. Once the obstacle is sufficiently distant, the robot switches back to the first mode of operation to return to the original path.

The first step is to determine the use of ultrasonic or sonar sensors as the method for obstacle detection. On one hand, this type of sensor has been widely used in robotic navigation tasks [199, 200, 201] due to its consistent results, relatively accurate measurements, and the high number of detections it can perform per second. The goal of the switch activator is to be reliable and highly reactive to changes in the environment, just like ultrasonic sensors. The Pioneer 3-DX model in CoppeliaSim includes 16 sonars for distance detection, as shown in Figure 2.37.

Despite the large number of available ultrasonic sensors, to avoid adding more complexity to real-time information processing, it was decided to use only 6 sensors in this proposal: two at the front (s_4 and s_5), two on the left (s_2 and s_3), and two on the right (s_6 and s_7). Although sensors s_1 and s_8 could enhance system redundancy, to avoid movements caused by obstacles at the sides of the robot, their use is not necessary. Similarly, sensors at the back are not considered, as the robot only moves forward, simplifying the controller's operation.

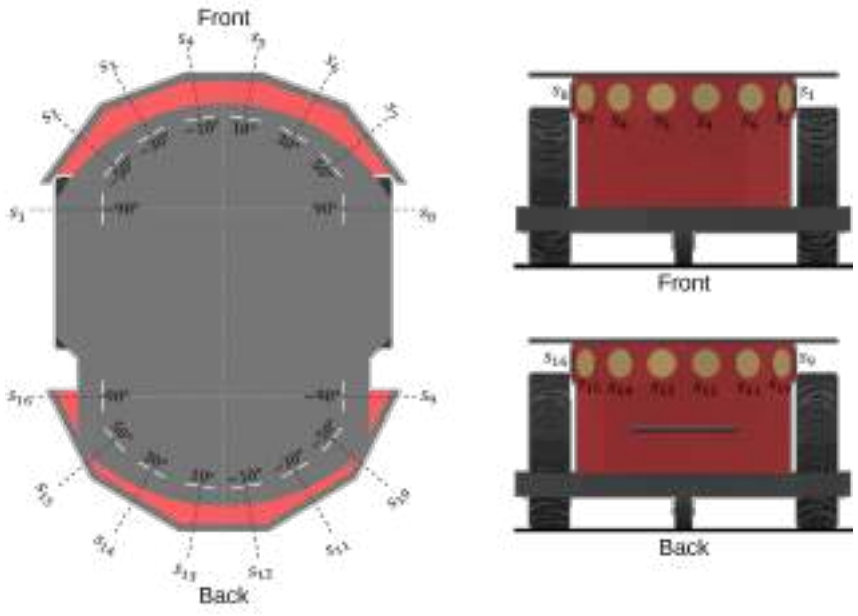
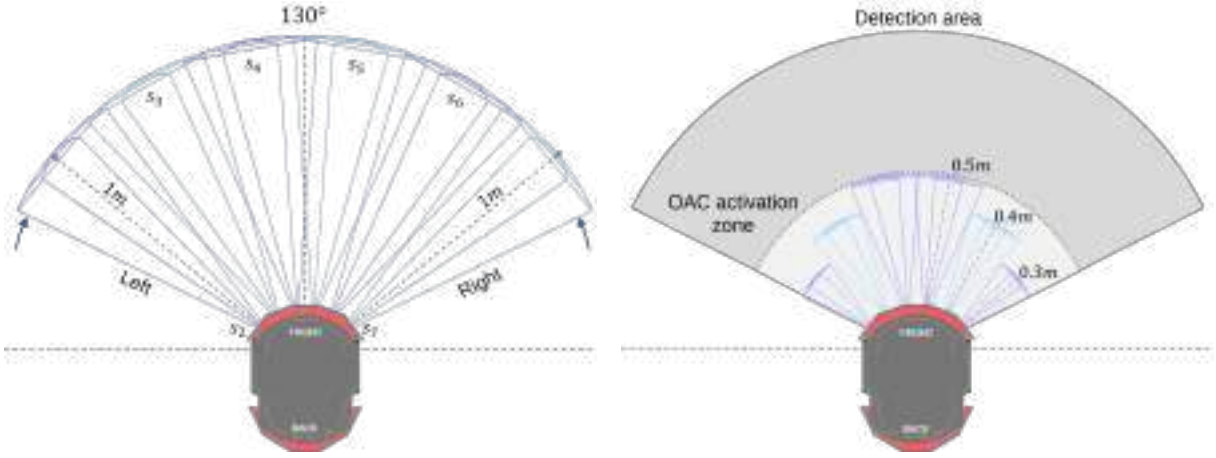


Figure 2.37: Sonar sensors arrangement of Pioneer 3-DX

Additionally, for this proposal, the detection range of the ultrasonic sensors was maintained at typical values, such as a detection angle of 30° with a range of 1 meter. With the chosen configuration, illustrated in Figure 2.38a, a total of 130° is covered in the detection range. The safety distance for activating the OAC, as presented in Figure 2.38b and to be evaluated in the next chapter, is proposed to be 0.5 meters for the front sensors (s_4 and s_5), 0.4 meters for sensors s_3 and s_6, and 0.3 meters for the outer sensors (s_2 and s_7). This proposal is based on the need for the robot to have more time (and thus more distance) to avoid an object coming from the front compared to obstacles on the sides. It is important to mention that activation occurs with the direct value detected by the sensors.



(a) Detection area using six ultrasonic sensors (b) Obstacle Avoidance Controller activation zone

Figure 2.38: Detection area and OAC activation zone using six ultrasonic sensors

Consequently, when any of the sensors detects an obstacle according to the defined threshold, the OAC will activate. This Obstacle Avoidance Controller is a Takagi-Sugeno fuzzy logic controller that uses optical flow estimates as inputs to determine the proximity of new static and dynamic obstacles and generate the necessary angular velocity signals for the wheels to avoid collisions.

Optical Flow (OF) is useful in this context because it not only indicates the direction of the apparent movement of pixels in the image, but its magnitude also reflects the presence of different objects, as optical flow values increase with more elements, edges, and intensity changes showing apparent movement within the image. Therefore, the goal is to implement OF to extract information from the captured images to determine the location of the nearest obstacle for avoidance purposes.

The images for calculating the OF are obtained from a single camera equipped on the robot that captures images of 256×256 pixels. The advantage of using a single camera is its low cost and easy mounting on the robot due to its reduced weight and size. In CoppeliaSim, this camera is integrated as a perspective-type vision sensor mounted on top of the robot (at 0.25 meters above the ground), capturing images with a 90° perspective angle. Figure 2.39 shows the image captured by the vision sensor.

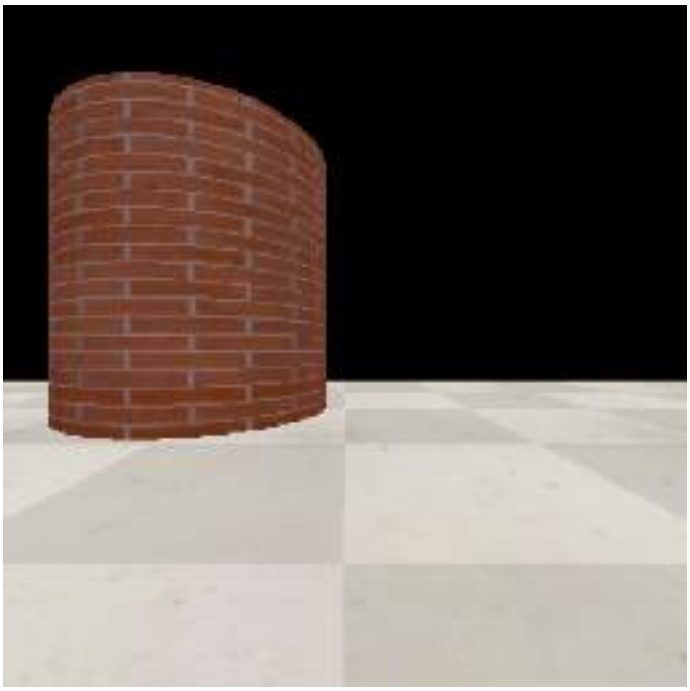


Figure 2.39: 256×256 image acquired from the Vision Sensor located in the robot (real size)

The process consists of dividing the obtained image into two parts (left and right), and calculating the mean magnitude flow values on the left (F_l) and right (F_r) sides using the following equations:

$$F_l(t) = \frac{1}{N} \sum_{F \in S_l} |\vec{F}(t)| \tag{2.25}$$

$$F_r(t) = \frac{1}{N} \sum_{F \in S_r} |\vec{F}(t)| \tag{2.26}$$

where N is the number of flow elements in the left or right half of the image, S_l is the left half of the image, S_r is the right half of the image, and $|\vec{F}(t)|$ is the magnitude of the optical flow vector in the t moment calculated using Eq. 2.13. In this context, if the average optical flow magnitude values on the left side are greater than those on the right side ($F_l(t) > F_r(t)$), it indicates a higher presence of nearby obstacles on that side, and therefore the robot should move to the right. Conversely, if $F_l(t) < F_r(t)$, the robot should move to the left.

The reasoning behind using magnitudes is to avoid the cancellation of negative and positive vectors. Additionally, it is not necessary to know the direction of the object, but rather to simply detect how close its presence is. On the other hand, the mean of these magnitudes is used because, although the OAC is activated only when an obstacle is near, the vision sensor continues to detect all elements in the scene, even those not close to the robot (range of 7 meters).

To estimate the optical flow within the image, the `OpenCV` library in Python is employed. This library facilitates the calculation of optical flow between two consecutive frames using either the sparse Lucas-Kanade method or the dense Farneback method (see *Optical Flow* subsection). In this study, the Farneback method was selected due to the relatively small size of the images, which permits the use of dense calculations without significant time loss, while preserving all information necessary for achieving accurate results suitable for control decisions.

The complete process developed in Pyhton involves capturing the environment image at each time step using the vision sensor on the robot. Then, optical flow vectors are computed from the image sequence (images at times $t-1$ and t) using the Farneback function from OpenCV (`cv.calcOpticalFlowFarneback`). The function takes as inputs the grayscale images of the previous frame (`prev_gray`) and the current frame (`gray`), along with several parameters, including those related to the pyramid image construction presented previously in Figure 2.9 (*Optical Flow* subsection).

The parameters to execute the function include pyramid scale factor (`pyr_scale`), number of pyramid levels (`levels`), window size (`winsize`), number of iterations (`iterations`), polynomial window size (`poly_n`), and Gaussian filter sigma (`poly_sigma`). The chosen parameters to determine the optical flow are a scale factor of 0.5, 3 pyramid levels, a window size of 15×15 pixels, 3 iterations, a polynomial window size of 5×5 pixels, and a sigma of 1.2, as per the default implementation presented in the OpenCV documentation [202].

Particularly, OpenCV's implementation computes the magnitude and direction of optical flow from a 2-channel array of flow vectors ($\frac{\delta x}{\delta t}, \frac{\delta y}{\delta t}$). It then visualizes the angle (direction) of flow by hue and the magnitude of flow by value in HSV color representation, with the saturation always set to a maximum of 255 for optimal visibility (Figure 2.40).

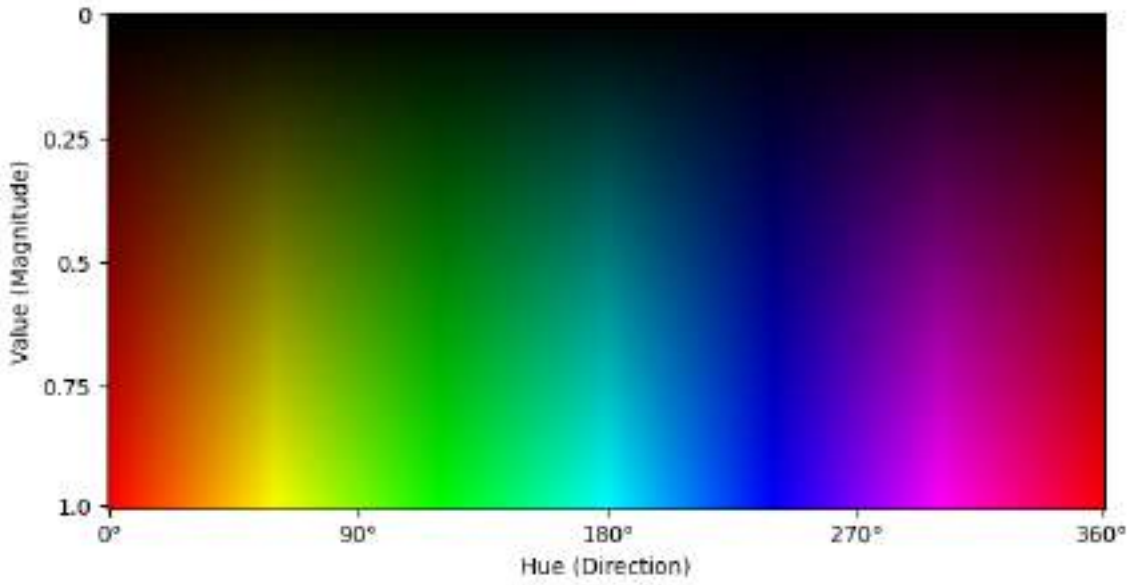
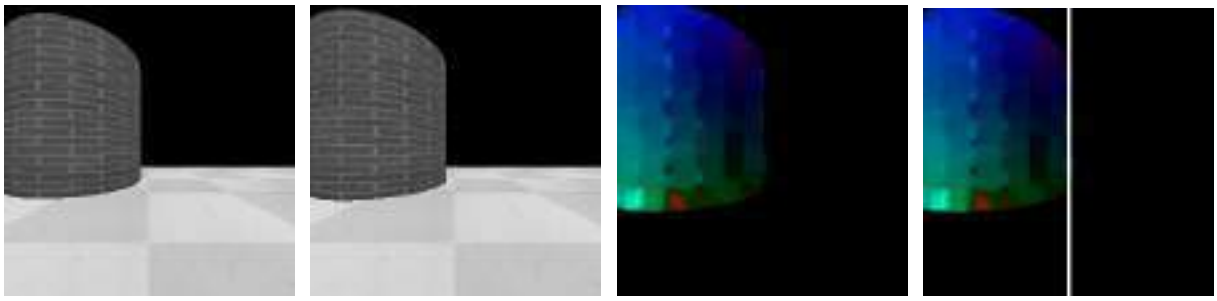


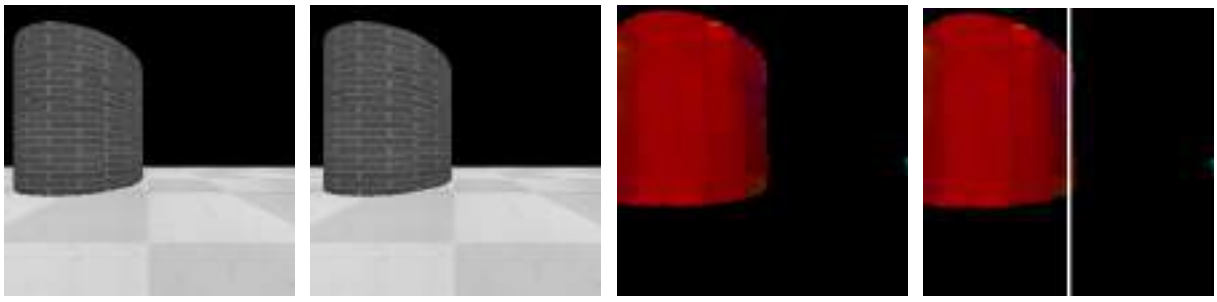
Figure 2.40: Color Palette in HSV Space

Considering this, Figures 2.41 and 2.42 show an example of the optical flow calculation implemented on the robot when the obstacle is approaching or moving away, respectively. Two consecutive grayscale frames taken by the vision sensor in CoppeliaSim are presented, and after the estimation, the optical flow can be visualized in an HSV representation. A brick texture was added to the obstacle in the scene because the presence of more edges and changes in color intensity allow for better determination of the optical flow. Additionally, Figures 2.41d and 2.42d illustrate the division into left and right sides that will be considered when calculating F_l and F_r (Equations 2.25 and 2.26, respectively), which will be the input for the OAC.



(a) Frame $t-1$ (b) Frame t (c) OF estimation (HSV) (d) Split OF image

Figure 2.41: HSV representation of Optical Flow estimation when an obstacle approaches the robot



(a) Frame $t-1$ (b) Frame t (c) OF estimation (HSV) (d) Split OF image

Figure 2.42: HSV representation of Optical Flow estimation when an obstacle moves away from the robot

Therefore, the average optical flow on the left side (F_l) and the right side (F_r) are the inputs to the Obstacle Avoidance Controller, which is also defined using the Python library `scikit-fuzzy`. The parameterization of the flow relates to the distance it represents: the higher the value of F_l or F_r, the closer the distance to an obstacle, since pixel apparent movement is greater. Based on this, the membership functions for the inputs are defined as *small* (S), *medium* (M), and *high* (H), according to optical flow values which correspond to far, medium, and near obstacle distances, respectively. As can be noted, fewer labels for inputs are identified compared to the previous controller, as the goal is to achieve a reactive (sometimes abrupt) movement that does not require high precision.

It is important to mention that the actual input values are the normalized values of F_l and F_r based on the maximum value of either of these two. This calculation is expressed by the following equations:

$$F_{l,\text{norm}} = \frac{F_l}{\max(F_l, F_r)} \tag{2.27}$$

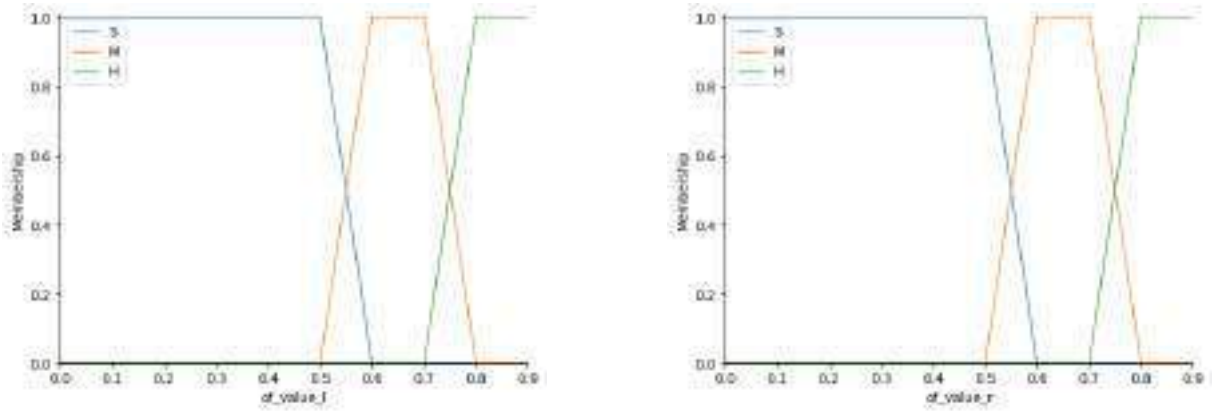
$$F_{r,\text{norm}} = \frac{F_r}{\max(F_l, F_r)} \tag{2.28}$$

This normalization is done to limit the input values within the range $[0, 1]$ as a way to prioritize the action given the presence of an obstacle. When normalized, one of the averages automatically becomes 1, indicating that the closest obstacle is on that side. Additionally, it is more precise to parameterize the membership function values within a limited range. For example, if the value is close to 0, the obstacle is far away; if it is close to 1 (or is 1), then the obstacle is near. Subsequently, Table 2.5 declares the membership functions for optical flow values (*of_values*).

Table 2.5: Membership Function Values for *of_value_l* and *of_value_r*

Distance Category	**Type**	**Membership Function Range**
Small (S)	Trapezoidal	[0,0, 0.5, 0.6]
Medium (M)	Trapezoidal	[0.5, 0.6, 0.7, 0.8]
High (H)	Trapezoidal	[0.7, 0.8, 0.9, 1]

In Figure 2.43, the membership functions are presented, which are the same for describing the optical flow values on both the left side (Figure 2.43a) and the right side (Figure 2.43b) of the image. It can be observed that for most of the range, the optical flow value is determined to be *small* (S) to relatively minimize the effect of distant objects detected by the vision sensor and prioritize those that are close (when optical flow is *high*).



(a) OF in left side $F_{l,\text{norm}}$ (b) OF in right side $F_{r,\text{norm}}$

Figure 2.43: Membership Functions for *of_values* categories: Small (S), Medium (M), and High (H), representing far, medium, and near obstacle distances, respectively

On the other hand, the consequents or outputs of the OAC are the avoidance angular velocities for the left wheel ($\omega_{avoid-L}$) and right wheel ($\omega_{avoid-R}$). In normal operation mode, the maximum angular velocity is set at 1.6 rad/s. However, in this case, given that it is a reactive movement in response to the presence of obstacles, it is considered that the movement should be slower, with the limit set at 1 rad/s.

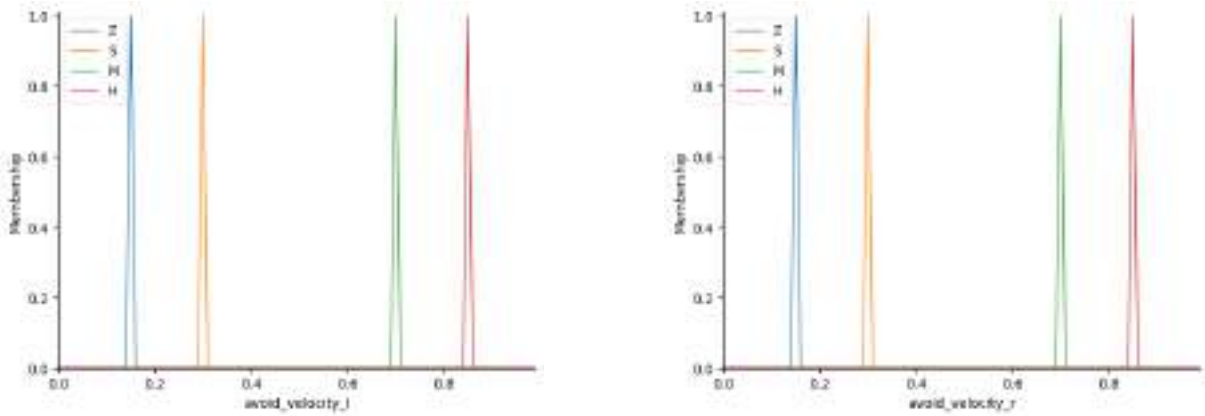
By using the Takagi-Sugeno approach for this fuzzy logic controller (see *Fuzzy Logic Controller* subsection), the velocities are determined as singleton membership functions due to their computational simplicity and ease of implementation, which is crucial for real-time decision-making. Additionally, this approach minimizes the impact of sensor noise by relying on unique values, allowing for a clear and direct determination of the robot's velocities, and making its behavior more consistent, as expected in this type of reactive mobile robotics applications.

In this context, the avoidance angular velocities as outputs can be classified as *zero* (Z), *small* (S), *medium* (M), and *high* (H), within a range of $[0, 1]$. Table 2.6 describes the singleton values for the velocities of both wheels. It can be observed once again that negative velocities are not considered, so the robot always moves forward and turns based on the difference in wheel velocities. The singleton values were defined in such a way that there is a significant difference between high and low velocities, enabling tighter turns (necessary in this mode of operation).

Table 2.6: Membership Function Values for $\omega_{avoid-L}$ and $\omega_{avoid-R}$

Velocity Category	**Type**	**Membership Function Range** $[rad/s]$
Zero (Z)	Singleton	[0.15]
Small (S)	Singleton	[0.3]
Medium (M)	Singleton	[0.7]
High (H)	Singleton	[0.85]

Figure 2.44 shows the membership functions of the avoidance angular velocities for the left wheel ($\omega_{avoid-L}$, *avoid_velocity_l*) and for the right wheel ($\omega_{avoid-R}$, *avoid_velocity_r*). Given that `scikit-fuzzy` does not explicitly support singleton functions, these were defined using a triangular function with the center at the desired value and an expansion of 0.05 on each side.



(a) Avoidance angular velocity $\omega_{avoid-L}$ (b) Avoidance angular velocity $\omega_{avoid-R}$

Figure 2.44: Membership Functions for *avoidance speed on wheels* categories: Zero (Z), Small (S), Medium (M), and High (H)

With the definition of both the antecedents and the consequents, it is possible to define the fuzzy rules of the controller with the following structure:

IF (*of_value_l* AND *of_value_r*) THEN (*avoid_velocity_l* AND *avoid_velocity_r*)

The total number of rules to be defined is equal to the total number of all possible combinations between the input labels (3 for each input), resulting in 9 rules (which is a considerably low number of rules). These combinations can be seen in Table 2.7.

Table 2.7: Obstacle Avoidance Controller Rules

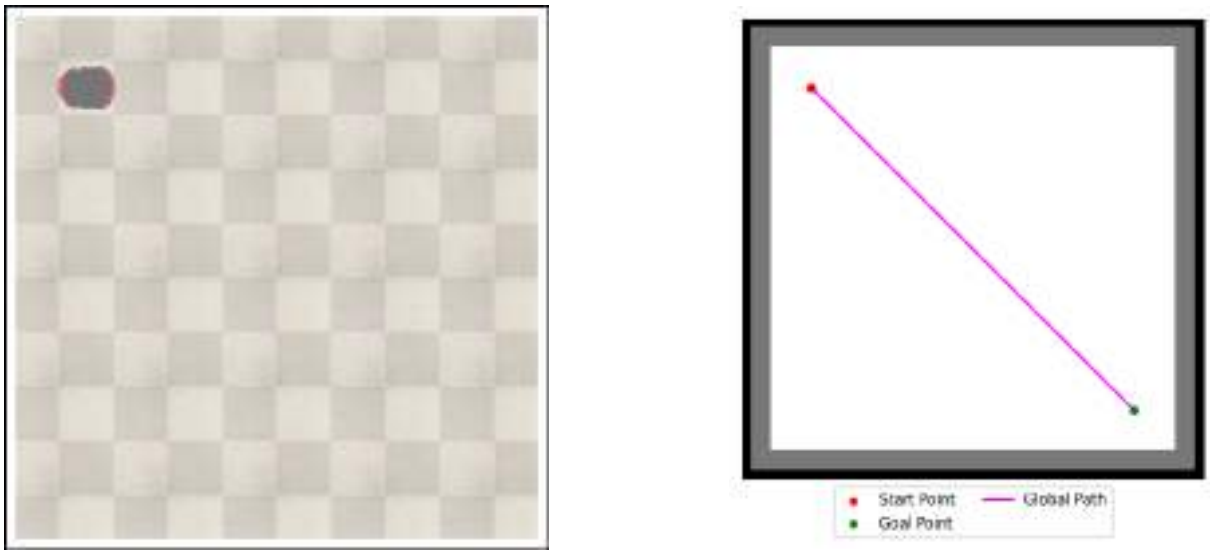
		$F_{l,\text{norm}}$		
		S	M	H
$F_{r,\text{norm}}$				
S	left	M	M	H
	right	S	S	Z
M	left	S	H	H
	right	M	Z	Z
H	left	Z	Z	Z
	right	H	H	H

The idea behind the rules is to simplify decision-making and the actions that the robot will execute. This mode of operation is solely intended to avoid collision with dynamic or unknown obstacles, so it is the task of the FLC from the first mode of operation to resume the correct path once the obstacle has been avoided. The rules presented for the OAC are focused on moving as far away as possible from the closest obstacles.

According to the fuzzy rules, if the estimated optical flow is *high* (H), it means that the obstacle is close and should be avoided as quickly and tightly as possible. That is why in all cases where $F_{l,\text{norm}} = H$ (the obstacle is close on the left side), the avoidance velocities are always *high* (H) for the left wheel and *zero* (Z) for the right wheel, causing the robot to move forward by turning as tightly as possible to the right. An analogous behavior occurs when $F_{r,\text{norm}} = H$ and the obstacle is close on the right side. In the case where both sides have a *high* (H) flow, it was decided by default to set $\omega_{avoid-L} = Z$ and $\omega_{avoid-R} = H$, so that the robot turns to the left if this occurs, remembering that for the estimation of optical flow, the robot must be in constant motion.

Although the four rules were defined for the combinations of optical flow *small* (S) and *medium* (M), that is, when no flow is *high* (H), the conditions for these rules are never met. This is because, after normalizing F_l and F_r (Equations 2.27 and 2.28), there will always be a value of 1, meaning that there will necessarily always be a flow categorized as *high* (H). This further reduces the processing complexity, which is desirable in a real-time reactive system, and leaves the OAC with only five rules. Further testing will determine whether this setup functions adequately.

An example of the OAC's operation is presented. A test environment was designed in a 5×5 meter space within CoppeliaSim (Figure 2.45a). The WMR was given the trivial task of crossing the area from one end to the other. As shown in Figure 2.45b, the planned path crosses straight through the center of the scene since no obstacles were identified. However, when the robot begins to move using the first mode of operation FLC, a static obstacle with brick texture is added in the center, obstructing the robot's global path (Figure 2.46a).

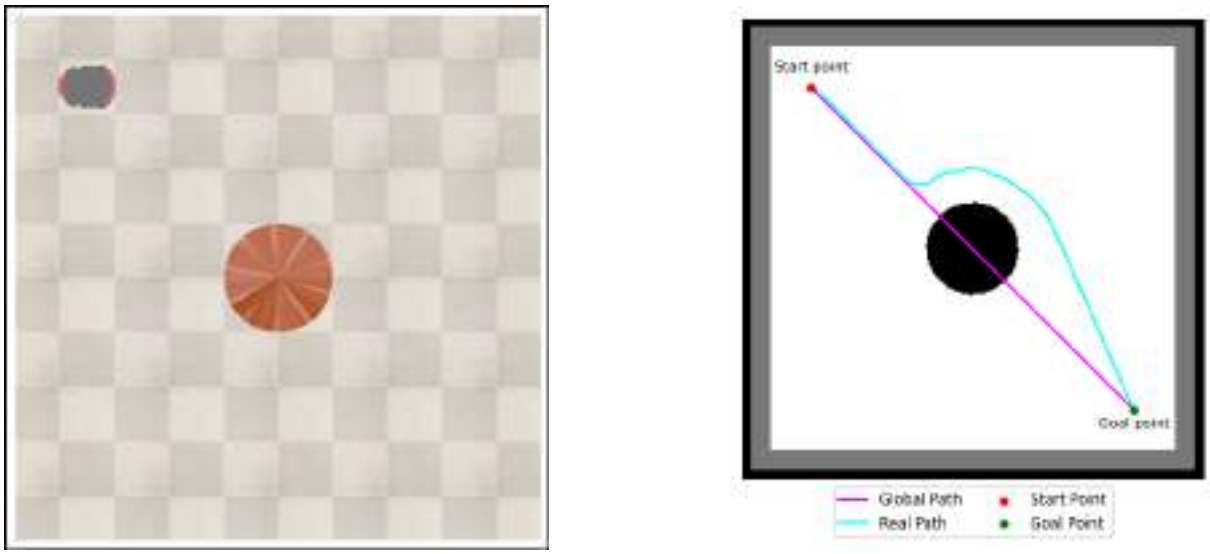


(a) Environment created in CoppeliaSim

(b) Expected global path

Figure 2.45: Environment for the Testing of the OAC proposed

In Figure 2.46b, it can be seen the result of the navigation in which the robot navigates around the previously unidentified static obstacle. The overall performance of this strategy aligns with the desired behavior previously described in Figure 1.7 (Section 1.2), where a local path is generated online to avoid an obstacle that intersects the global path as the AMR moves towards the goal. The next chapter will present the performance evaluation, including scenarios where the robot encounters unknown dynamic obstacles.

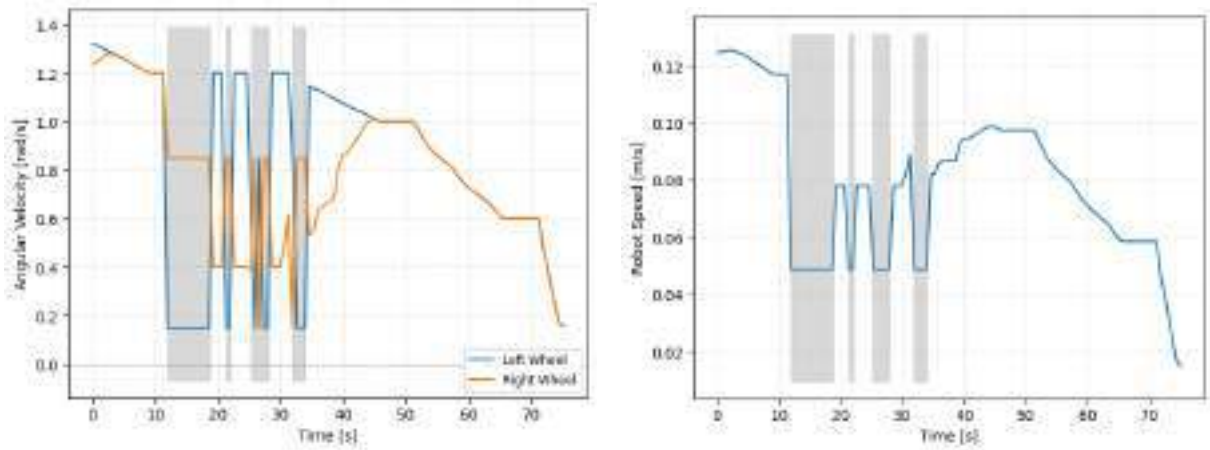


(a) Unkown obstacle added during simulation

(b) Real path traveled by the robot

Figure 2.46: Obstacle added in the middle of the robot's global path to test OAC

Moreover, Figure 2.47 illustrates the angular velocities of the robot's wheels (Figure 2.47a) and the linear velocity of the robot during the simulation (Figure 2.47b), where the modes of operation faced by the WMR can be identified. The first mode of operation (FLC) occurs when the obstacle is not detected by the sonars. The second mode of operation (OAC), indicated by the shaded areas, is activated whenever the robot detects an obstacle. As can be seen, the activation/deactivation of the modes of operation is alternated because each time the robot moves far enough away from the obstacle using OAC, the FLC redirects it towards the destination. A noteworthy point is that the movement of the wheels when the OAC is activated occurs at a constant linear velocity, indicating that the robot moves smoothly and continuously.



(a) Wheels' Angular Velocity

(b) Robot Speed

Figure 2.47: Robot Velocity during the navigation to test the Obstacle Avoidance Controller. The shaded areas indicate the moments when the OAC is activated.

This concludes the local planning and motion control process explanation. A flowchart showing the entire procedure conducted by the local planner is presented in Figure 2.48. It is important to highlight that this is performed entirely online in real time. Additionally, this also concludes the explanation of the proposed navigation strategy presented at the beginning of this section (*Navigation Strategy General Description*, Figure 2.15). Therefore, in the next chapter, the methodology and the solution itself will be evaluated and validated.

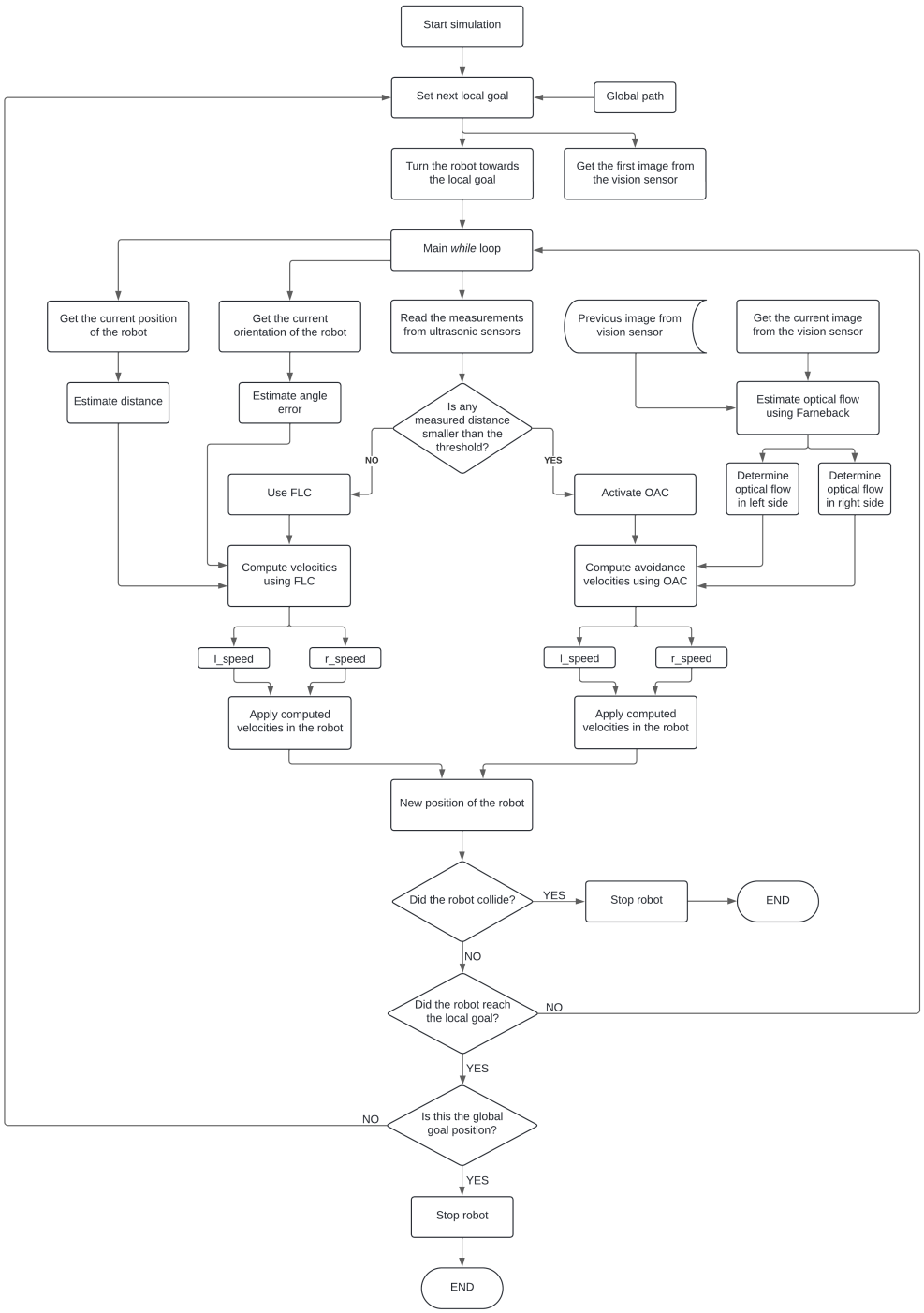


Figure 2.48: Local Planning Subsystem Flowchart

Chapter 3

Results & Analysis

In the previous chapter, all the components that integrate the *Comprehensive Autonomous Navigation Strategy for Mobile Robots* were explained in detail. This strategy is proposed and designed based on the general objective established in the *Introduction*, to make a WMR capable of moving in different static and dynamic environments automatically, efficiently, safely, and accurately.

Therefore, the performance measurement of the strategy is based on whether it can be considered automatic, efficient, safe, and accurate. Each designed module that integrates the strategy aims to satisfy these needs, as explained in the hypothesis (see Section 2). It is clarified that this research work does not intend to find the best navigation strategy nor solve the mobile robotics paradigm. Instead, it focuses on implementing a possible solution whose performance is evaluated in this chapter. In this regard, the deliberative-based global path planning algorithm to find the shortest feasible route relates to the efficiency and safety of the strategy. Subsequently, the reactive-based local path planning technique to avoid unknown obstacles aims to be automatic and safe. Finally, the motion control to follow the predetermined local or global path seeks to be automatic, safe, and accurate.

Considering the above, the navigation strategy is put to the test, evaluating and validating the design decisions of each component separately so that the overall functioning of the strategy can be thoroughly verified in both static and dynamic scenarios. All tests were executed on a Dell Precision 7760 with an Intel i7 11th Gen processor, 32 GB of RAM, and an NVIDIA RTX A3000 Laptop GPU. CoppeliaSim Edu version 4.5.1 (rev. 4) was used, and the code was implemented in Python version 3.10.9. The obtained results are presented below.

3.1. Global Path Planning Test

This section aims to evaluate the efficiency and safety of the global path planner described in the *Global Planning Subsystem* subsection. To achieve this, the robustness of the planner in finding routes in different environments will be tested.

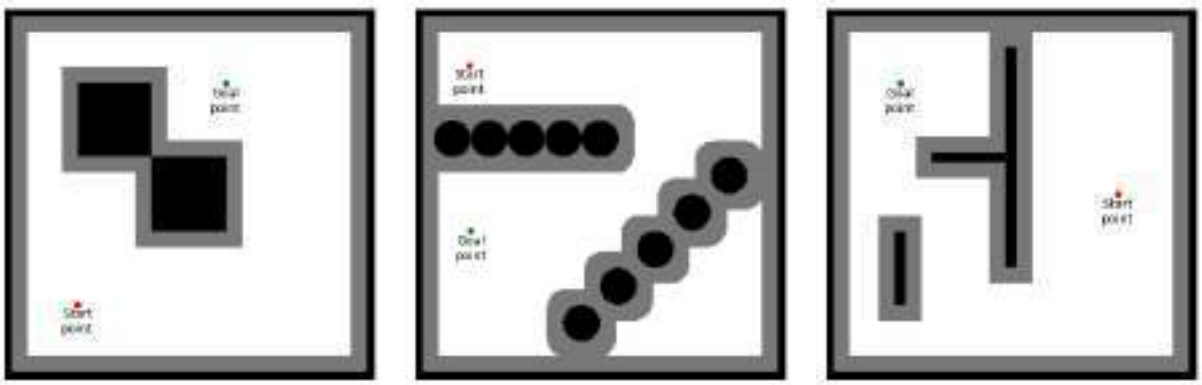
3.1.1. Experiment on GWO Position Update Equations

This experiment is used to compare the performance between estimating new positions in each iteration using the original equation (Eq. 2.8) or the average weighting approach presented in [166] (Eq. 2.9), as explained in Section 2.2 (*Grey Wolf Optimization* subsection). The authors of the second equation argue that in the position update, more weight should be given to the best solution α, followed by β and δ. The idea is to verify whether this modification truly impacts the context of path planning. As a result, the equation that finds the optimal route in the fewest iterations will be determined (if any). This equation will be used in the remaining tests.

$$\textit{Original approach: } \vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{2.8}$$

$$\textit{Weighting approach: } \vec{X}(t+1) = \frac{5\vec{X}_1 + 3\vec{X}_2 + 2\vec{X}_3}{10} \tag{2.9}$$

Three scenarios with different complexities were designed in CoppeliaSim (Figure 3.1), and the global paths were obtained using both equations and a maximum of 100 iterations for the GWO, 8 search agents (or wolves), and an empirical number of waypoints for each scenario. The experiment was repeated twice in each scenario for each of the equations and the results are presented in Table 3.1.



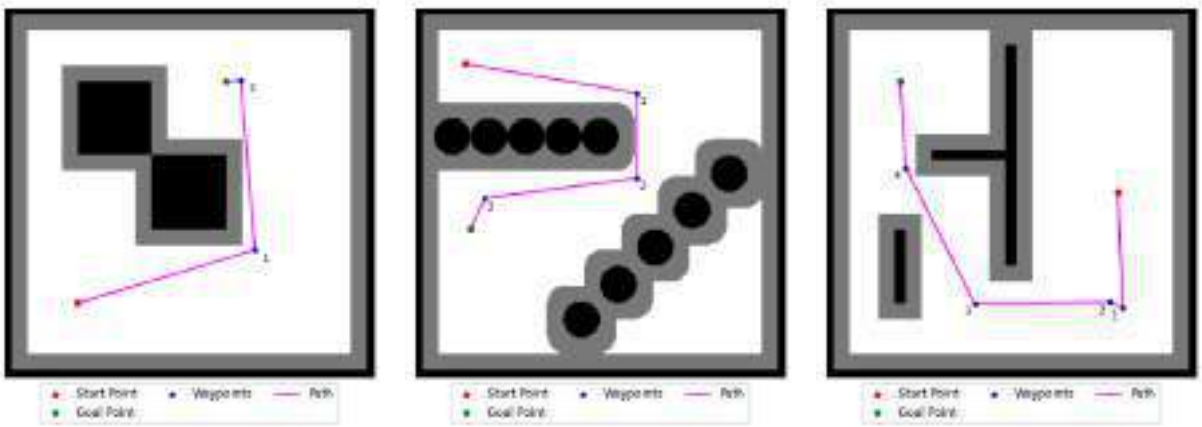
(a) Scenario A (b) Scenario B (c) Scenario C

Figure 3.1: Scenarios created to test equations for GWO position update

Table 3.1: Experiment on GWO Position Update Equations Results

Equation	Scenario		Final distance $[m]$	Convergence iteration	Waypoints
Original	A	Test 1	5.12	95	2
		Test 2	5.18	97	2
Weighting	A	Test 1	5.20	96	2
		Test 2	**5.04**	**20**	**2**
Original	B	Test 1	6.55	20	3
		Test 2	6.67	55	3
Weighting	B	Test 1	6.88	81	3
		Test 2	**6.06**	**11**	**3**
Original	C	**Test 1**	**6.82**	**80**	**4**
		Test 2	7.03	36	4
Weighting	C	Test 1	7.88	**21**	4
		Test 2	7.18	58	4

In each scenario, four potential paths were identified. Figure 3.2 illustrates the shortest paths for each scenario. The differences between using one equation over another are marginal, with no significant disparities in the estimation of the shortest distance or the number of iterations to convergence. In Scenario A, the original equation and its weighted variant produced similar results, but with the weighted equation achieving the shortest distance in the fewest iterations during its second test. Scenario B exhibited similar behavior. In Scenario C, the original equation found the shortest path but required the most iterations. Although prioritizing solution accuracy over iteration convergence is important, achieving similar results with fewer iterations is preferred.



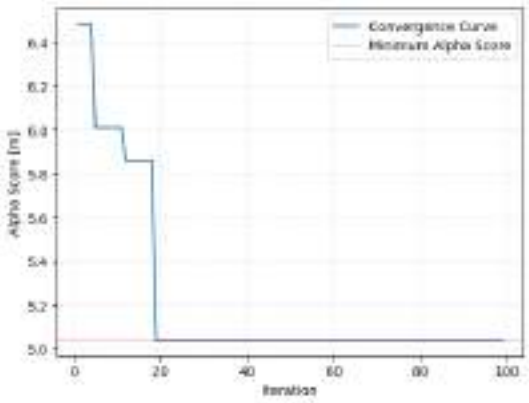
(a) Scenario A: Test 2 of the Weighting equation

(b) Scenario B: Test 2 of the Weighting equation

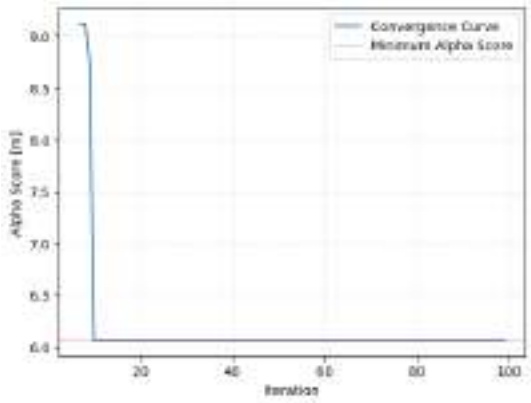
(c) Scenario C: Test 1 of the Original equation

Figure 3.2: Shortest paths found for each scenario in the test of equations for GWO position update

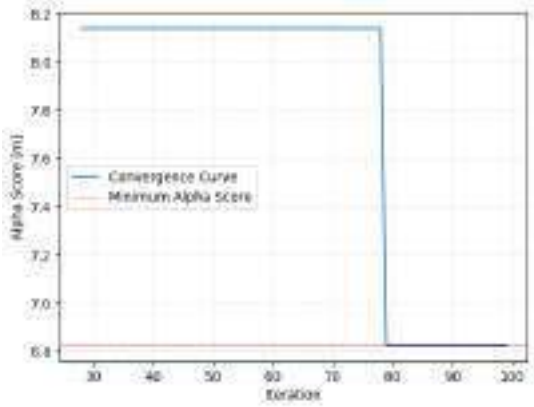
On the other hand, Figure 3.3 shows the convergence process of the α score (best solution) over the iterations for the best solutions found. It can be observed that in this context, the optimizing power of the GWO is lost due to the heavy constraints imposed by the obstacles in the configuration space. This is evident as, despite having 100 iterations, the best solution changes very few times. The lack of optimization capability is compensated by the high randomness of the algorithm.



(a) Scenario A: Test 2 of the Weighting equation



(b) Scenario A: Test 2 of the Weighting equation



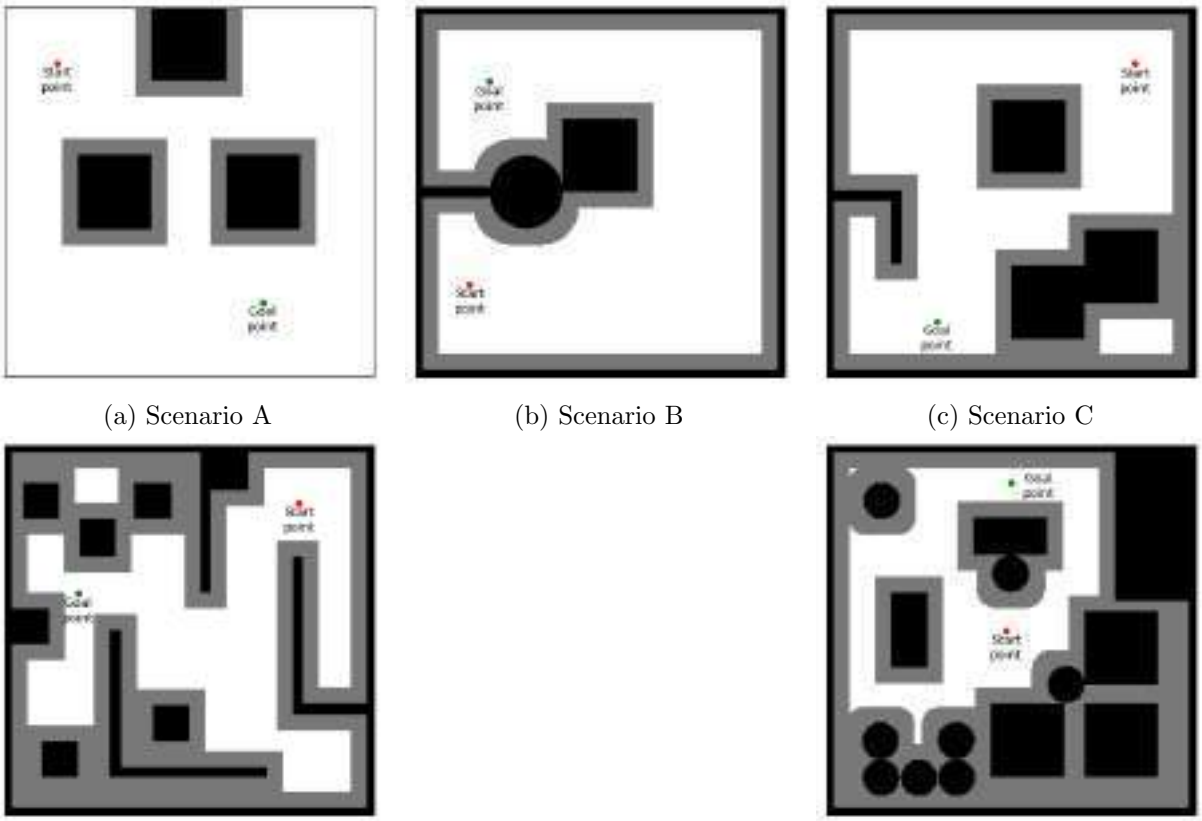
(c) Scenario C: Test 1 of the Original equation

Figure 3.3: Convergence curve of the alpha score for the shortest solutions in the test of equations for GWO position update

In conclusion of this experiment, even though it is not possible to distinguish a substantial difference in the performance of the algorithm, it was decided to use the *weighting approach* equation for the rest of the experiments. This decision was made because it was considered interesting to reward the best solution through the iterations. It should be noted that a difference could be found with more exhaustive experimentation, but since it is not a fundamental parameter in the functioning of the planner, a deeper study is not conducted.

3.1.2. Experiment on the Static Obstacles Density in the Scene

This experiment aims to measure the performance of the global planner in the presence of different proportions of static obstacles in the configuration space. To this end, five scenarios of 5×5 meters were designed in CoppeliaSim (Figure 3.4), each with varying obstacle densities: Scenario A with $\approx 20\%$, Scenario B with $\approx 35\%$, Scenario C with $\approx 50\%$, Scenario E with $\approx 65\%$, and Scenario D with $\approx 80\%$. The planner was executed offline to determine the route between two points in each scenario.



(a) Scenario A (b) Scenario B (c) Scenario C

(d) Scenario D (e) Scenario E

Figure 3.4: Scenarios created to test different obstacle densities in the scene

The obstacle density ($\mathcal{CO}_{\%}$) is estimated from the image representing the configuration space $\mathcal{C}$ obtained from CoppeliaSim. For this, the number of pixels (p_x) that belong to the $\mathcal{C}$-*obstacle* (including the dilation of the obstacles) is counted and divided by the total number of pixels in the image, as shown in the following equation:

$$\mathcal{CO}_{\%} = \frac{\sum_{p_x \in \mathcal{CO}}}{\sum_{p_x \in \mathcal{C}}} \times 100 \tag{3.1}$$

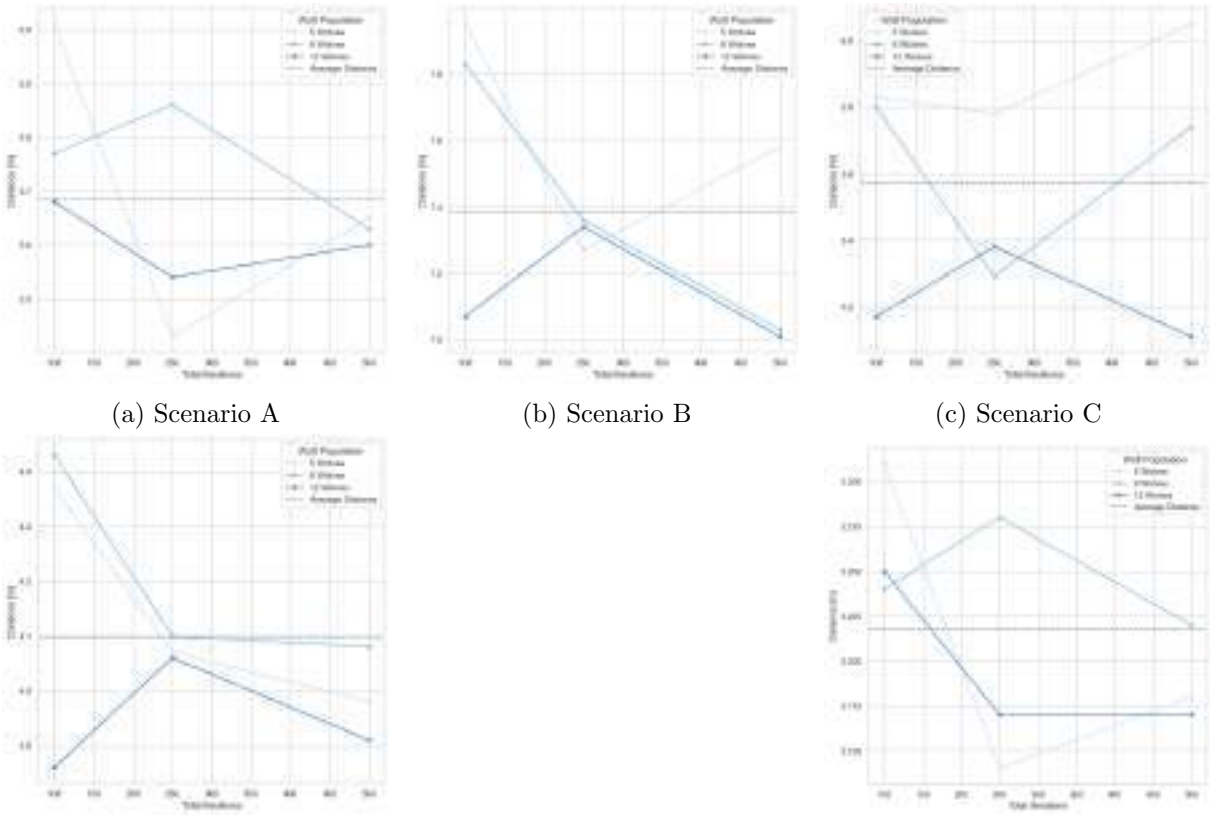
The configuration parameters for this experiment, in addition to the start point and goal point, include the number of iterations of the GWO, the number of possible solutions (or wolves), and the number of waypoints in the route. First, it was decided to test with 100, 250, and 500 iterations to show whether this quantity influences obtaining a valid route. If the route is not found after 500 iterations, it is concluded that the route is not feasible. On the other hand, the number of search agents or wolves will be set to 5, 8, and 12, chosen to balance execution time with exploration space. Finally, the number of waypoints is determined empirically for each scenario.

Therefore, a total of 45 tests were conducted by combining the five scenarios, the three options for the number of iterations, and the three options for the number of search agents. The results are presented in Table 3.2.

Table 3.2: Experiment on the Obstacles Density in the Scene Results

				Population size					
				5 wolves		8 wolves		12 wolves	
Scenario	Obstacle density	Waypoints	Total iterations	Distance $[m]$	Time $[s]$	Distance $[m]$	Time $[s]$	Distance $[m]$	Time $[s]$
A	23.49%	2	100	6.01	17.84	5.77	29.68	5.68	44.70
			250	5.43	47.91	5.86	75.48	5.54	111.02
			500	5.65	92.13	5.63	148.42	5.60	218.70
B	33.50%	3	100	7.95	21.75	7.83	33.61	7.07	48.70
			250	7.27	57.76	7.36	84.52	7.34	126.29
			500	7.58	109.03	7.03	159.24	7.01	237.67
C	50.02%	3	100	5.83	27.14	5.80	38.98	5.17	52.65
			250	5.78	78.55	5.29	105.94	5.38	137.75
			500	6.05	154.04	5.74	189.33	5.11	254.48
D	62.8%	3	100	4.37	29.63	4.43	41.61	3.86	55.54
			250	4.07	82.11	4.10	111.68	4.06	141.17
			500	3.98	163.67	4.08	206.04	3.91	263.64
E	78.36%	3	100	3.31	35.16	3.24	46.54	3.25	61.53
			250	3.14	97.73	3.28	127.14	3.17	149.00
			500	3.18	182.12	3.22	227.34	3.17	278.12

Figure 3.5 graphically shows the results. The intuitive idea is that the more iterations there are, the more likely it is to find the optimal path. Similarly, with a greater number of search agents or wolves, the exploratory capability of the algorithm increases and there is a higher likelihood of finding solutions.



(a) Scenario A (b) Scenario B (c) Scenario C

(d) Scenario D (e) Scenario E

Figure 3.5: Experiment on the Obstacles Density in the Scene Results

In Scenario A (Figure 3.5a), with an obstacle density of 23.49%, the solutions ranged from 5.43 to 6.01 meters. Notably, the shortest solution was found using 5 wolves with 250 iterations. However, 5 wolves showed the worst results with 100 and 500 iterations, where 12 wolves achieved the shortest distances. It is also important to note that for 500 iterations, the three results differed by only 14 centimeters.

Similarly, Scenario B (Figure 3.5b), with 33.50% obstacle presence, showed comparable trends. At 250 iterations, 5 wolves achieved the best result. Nevertheless, 12 wolves found the optimal solution at 100 and 500 iterations, with the best overall result in the latter case. Additionally, for 500 iterations, the result from 5 wolves deviated by half a meter from the other two distances (8 and 12 wolves).

In Scenario C, with approximately 50% obstacle density, the best overall solution was found with 12 wolves and 500 iterations (Figure 3.5c). The distance range between solutions was 0.85 meters, the largest among the scenarios. This is likely due to the two possible routes above and below the central obstacle (see Figure 3.4).

Conversely, in Scenario D, which has a 62.8% obstacle density, solutions with 12 wolves were consistently shorter than those with 8 wolves, which were in turn shorter than those with 5 wolves for all iterations (Figure 3.5d). Interestingly, for 500 iterations, all three solutions were below the average distance, while the best result was found with 12 wolves and 100 iterations.

Finally, Scenario E, with a 78.36% obstacle density, exhibited the smallest variation range of 17 centimeters (Figure 3.5e). This is because the path's inherent shortness and the high obstacle density limited optimization, making most solutions similar. Consequently, the best solution was achieved by 5 wolves with 250 iterations, showing minimal influence from iterations or the number of wolves in this scenario.

In general, it must be remembered that the paths are found by a stochastic and metaheuristic algorithm such as the GWO, meaning that if this experiment is repeated, different outcomes may likely be obtained. Despite this, certain conclusions can be drawn based on these results. Firstly, it seems that having more wolves or search agents is more determinant than the total number of iterations. In all scenarios, only the result of the combination of 12 wolves was always below the estimated average distance (except with 100 iterations in Scenario E, which is just 3 centimeters above the average). This means that larger population size has a greater effect on the result as it represents a greater exploration of solution space.

On the other hand, the effect of iterations depends on the position update strategy of the GWO algorithm. The more congested a scenario is or the higher the presence of obstacles, the fewer paths and the less opportunity for optimization. Therefore, the effect of iterations is lost in highly occupied scenarios, since there is practically only one path that can be found. In this sense, with so many obstacles, the new solutions proposed by the algorithm in each iteration are very likely to be unfeasible paths that cannot be considered during the optimization process.

In conclusion, it makes more sense to use the algorithm in a space without so many obstacles to take advantage of the GWO's optimization strategy. Otherwise, the solution only relies on the high randomness of the algorithm to find a route, meaning that with many proposed solutions, one is likely to be a feasible solution. For example, in a test with 12 wolves and 500 iterations, a total of 6000 possible solutions are proposed, and at least one of them will likely be a feasible solution. Additionally, whether it was 100, 250, or 500 iterations does not imply that the best solution was found in the last iteration, thus relying more on the stochastic process.

Beyond these considerations, it is notable that for all scenarios and all combinations, a route could be found. Figure 3.6 shows all the solutions found, with each row being a scenario. The first three solutions in each row correspond to 5 wolves for 100, 250, and 500 iterations respectively, the next three for 8 wolves, and the last three for 12 wolves successively.

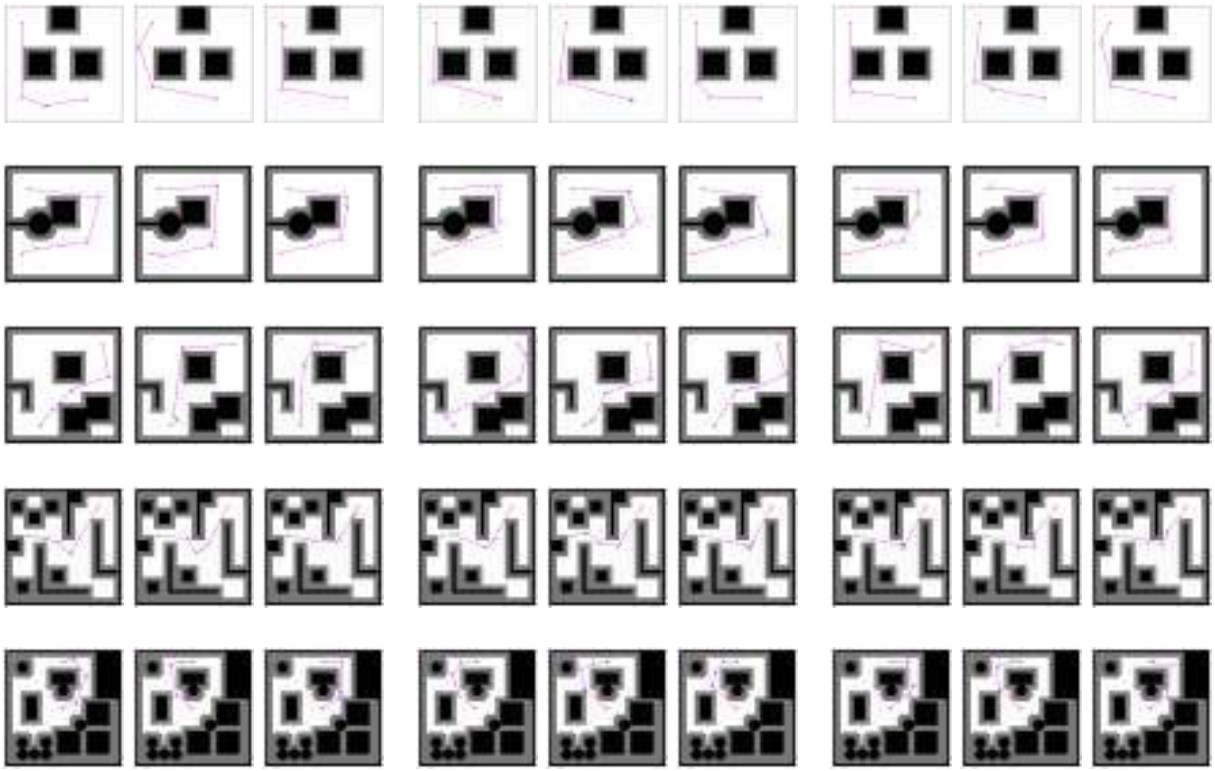
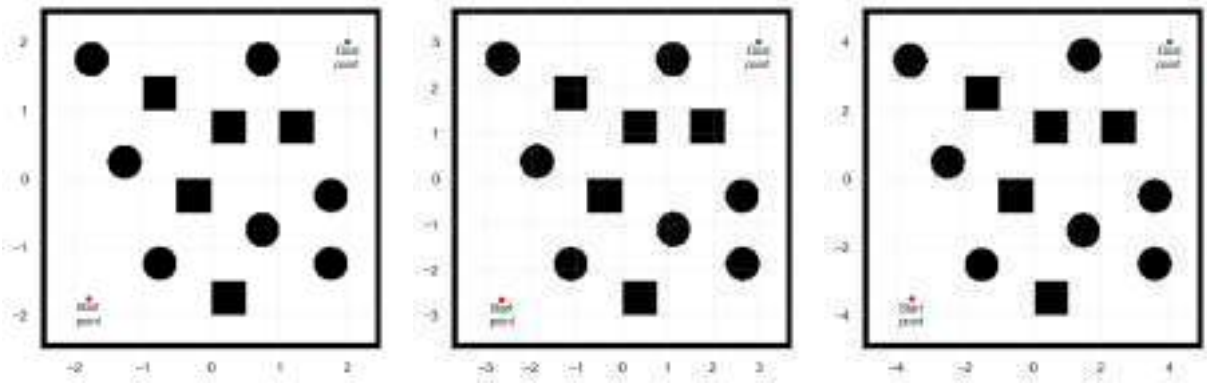


Figure 3.6: Global Paths found during the Experiment on Obstacles Density in the Scene

3.1.3. Experiment on Scene Size

Up to this point, the design and testing of the global planner have been conducted in 5×5 meter scenarios. This experiment aims to verify and compare the performance of the algorithm in larger scenarios. For this purpose, three scenarios were designed in CoppeliaSim (Figure 3.7): 1) 5×5 meters; 2) 7.5×7.5 meters; and 3) 10×10 meters. These environments are equivalent in the sense that the size, shape, and position of the obstacles are proportional to the dimensions of the scenario. In total, each scenario has approximately a 20% obstacle density.

It should be noted that only on this occasion were the obstacles not dilated to maximize the space that could be optimized. Additionally, the resolution of the image capturing the environment remains at 256×256.



(a) Scenario A: $5 \times 5\ m$ (b) Scenario B: $7.5 \times 7.5\ m$ (c) Scenario C: $10 \times 10\ m$

Figure 3.7: Scenarios created to test the impact of the Scene Size in the Global Planner

For each scenario, three tests were conducted using a constant combination of 12 wolves with 300 iterations, while the number of waypoints was determined empirically. The assigned task is to traverse each scenario from corner to corner, and the results are shown in Table 3.3.

Table 3.3: Experiment on Scene Size Results

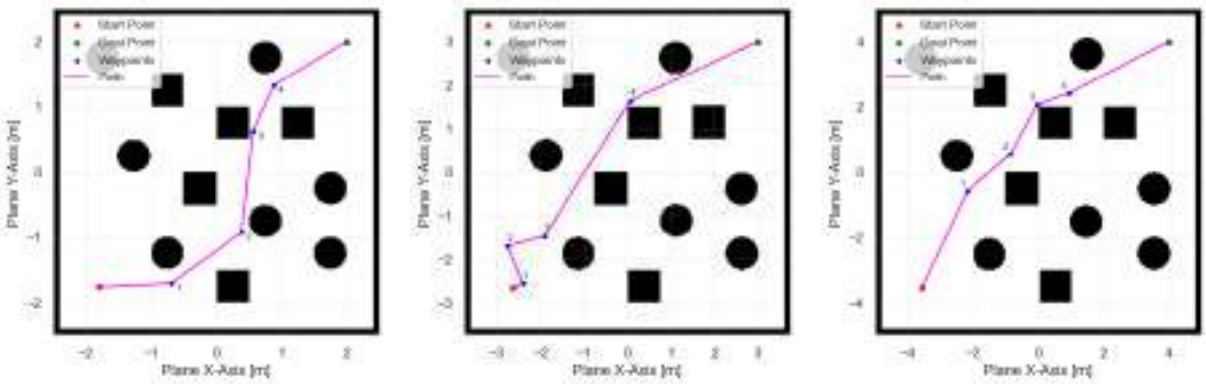
Scenario	Size $[m]$	Obstacle density	Waypoints		Converge iteration	Distance $[m]$
A	5×5	18.10%	4	Test 1	201	6.12
				Test 2	156	6.10
				Test 3	201	6.07
				Mean	186	6.09
B	7.5×7.5	18.18%	4	Test 1	212	9.25
				Test 2	57	9.48
				Test 3	153	8.97
				Mean	140.6	9.23
C	10×10	18.12%	4	Test 1	231	11.20
				Test 2	120	11.94
				Test 3	98	12.81
				Mean	149.6	12.98

As presented in the *Global Planning Subsystem* subsection, the optimization is performed on the image and not on the environment. By maintaining the image resolution at 256×256 and scaling the obstacles for each scene, the same optimization problem is being solved in the three scenarios, with the difference that each pixel represents a different distance: $1\ p_x \approx 1.95\ cm$ in the $5 \times 5\ m$ scenario, $1\ p_x \approx 2.93\ cm$ in the $7.5 \times 7.5\ m$ scenario, and $1\ p_x \approx 3.91\ cm$ in the $10 \times 10\ m$ scenario. Square environments are used to facilitate the conversion between real-world coordinates and image coordinates.

Considering this, the problem lies in how the environment mapping is obtained, as the way the algorithm is defined allows for finding global routes in different-sized scenarios using a 256×256 image, merely by adjusting the size that the image covers in the vision sensor within CoppeliaSim (see Figure 2.17). It should be noted that implementing the mapping process itself is not part of the objectives of this research work.

The results show this consistency throughout the tests, and, in general, similar global paths are obtained in a comparable number of iterations across the three scenarios. There are also outliers, such as in test 2 of scenario B, which found its convergence distance in only 57 iterations, although this was the longest distance in that scenario. Additionally, this means that from iterations 58 to 300, the route was not improved. Based on this, the trade-off between what is preferable must be considered: an acceptable path in a smaller number of iterations or the best possible path in a high number of iterations. This decision will depend on the environment, the needs, and the available time. In this case, available time is not a factor as all global planning tasks are performed online.

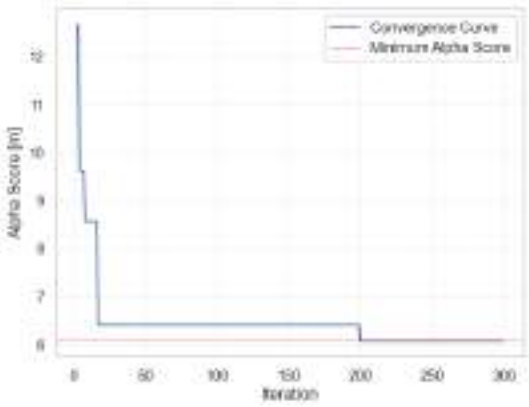
Figure 3.8 shows the best solution in each of the scenarios according to the shortest distance. As can be observed, the optimization task is the same, but the configuration space represents areas with different dimensions. This demonstrates the feasibility of using the algorithm in scenarios of different sizes.



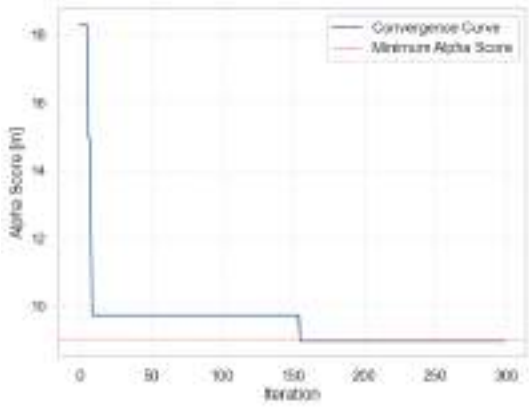
(a) Scenario A: 5×5 m Test 3 (b) Scenario B: 7.5×7.5 m Test 3 (c) Scenario C: 10×10 m Test 1

Figure 3.8: Shortest paths used to test the impact on Scene Size in the Global Planner

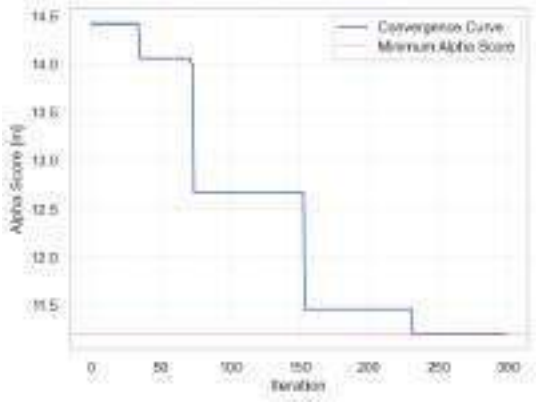
Furthermore, Figure 3.9 shows the convergence process of these three solutions, demonstrating that the full 300 iterations are not necessary.



(a) Scenario A: 5×5 m Test 3



(b) Scenario A: 7.5×7.5 m Test 3



(c) Scenario C: 10×10 m Test 1

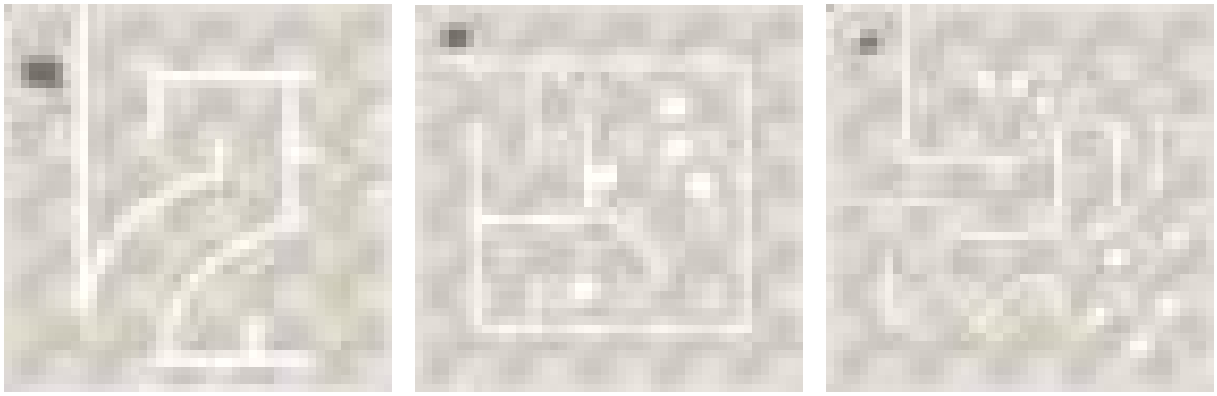
Figure 3.9: Convergence curve of the alpha score for the shortest solutions in the test of the impact on Scene Size in the Global Planner

3.2. Global Path Following Test

Once the correct functioning of the global planner has been tested, this section intends to verify the performance of the motion control for the global path following. Specifically, the effectiveness of the Fuzzy Logic Controller (see page 87) designed during the *Local Planning Subsystem* section for following the global paths is tested in various scenarios.

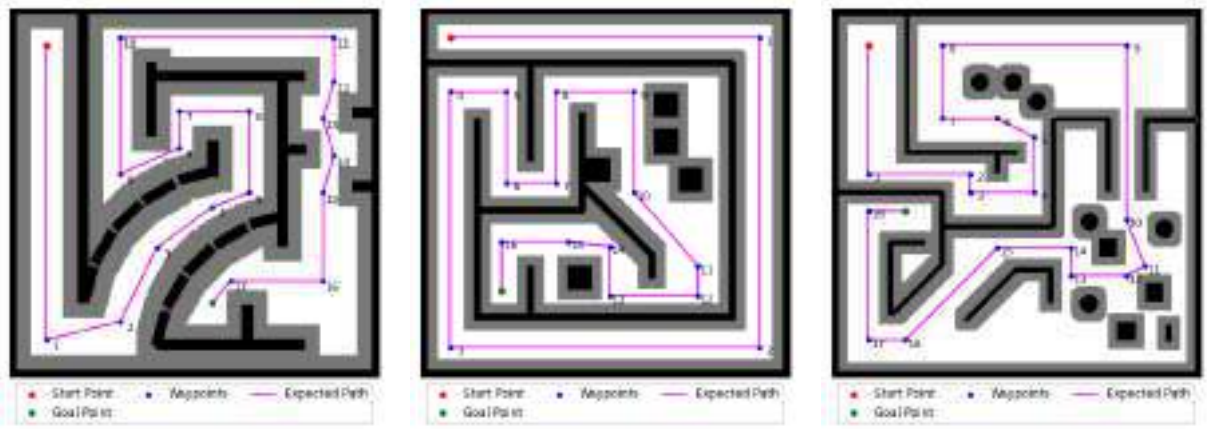
3.2.1. Experiment on the Effectiveness of Path Following

This is the only experiment within this section, but it effectively describes the performance of the path-following. It involves the creation of three completely static scenarios of 5×5, 7.5×7.5, and 10×10 meters that test the accuracy of the controller (Figure 3.10). The scenarios, routes, and start and goal positions are determined manually in a way that challenges the robot's movement (i.e., the path was not created using the global planner, as the current purpose is solely to demonstrate the functionality of the path following). The global paths created for each scenario are shown in Figure 3.11



(a) Scenario A: 5×5 m (b) Scenario B: 7.5×7.5 m (c) Scenario C: 10×10 m

Figure 3.10: Scenarios created to test the effectiveness of the Path Following Control



(a) Expected path of Scenario A (b) Expected path of Scenario B (c) Expected path of Scenario C

Figure 3.11: Global Paths created to test the effectiveness of the Path Following Control

For each scenario, the robot will follow the path using three different speeds. It is important to mention that in the controller's design, the angular velocities of the wheels were set in a range of 0 to 1.6 rad/s, which represents a speed limit of approximately 0.16 m/s (see Table 2.3), which is quite conservative considering that the robot can reach up to 1.2 m/s. Originally, it was designed this way to provide reactive capability in a 5×5 scenario, but in this case, where dynamic elements are not considered, it is unnecessary.

Therefore, in this experiment, the output velocities after defuzzification were multiplied by a gain to maintain the behavior but at a higher speed. The three speed limits were set as follows: 1) $1.6 \approx 0.16$ m/s (the original); 2) 4.8 $rad/s \approx 0.47$ m/s (3 times the original); and 3) 9.6 rad/s ≈ 0.94 m/s (6 times the original). The parameter to be evaluated is the difference between the expected distance of the path and the actual distance traveled by the robot (distance deviation).

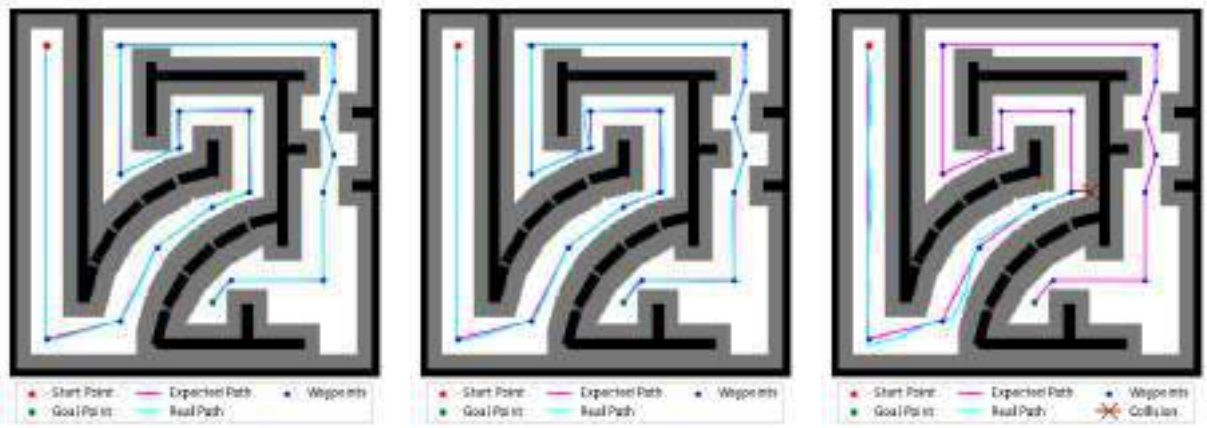
The results of the experiment can be seen in Table 3.4.

Table 3.4: Experiment on the Effectiveness of Path Following Results

Scenario	Size $[m]$	Speed limit $[m/s]$	Mean speed $[m/s]$	Expected distance $[m]$	Real distance $[m]$	Distance deviation	Time $[s]$	Path completed
A	5×5	0.16	0.06	20.60	21.39	+3.84%	444.89	✓
		0.47	0.19		21.80	+5.83%	280.29	✓
		0.94	0.43		7.54	N/A	72.06	✗
B	7.5×7.5	0.16	0.08	42.19	43.32	+2.67%	703.67	✓
		0.47	0.24		43.79	+3.79%	372.84	✓
		0.94	0.57		19.80	N/A	104.51	✗
C	10×10	0.16	0.07	39.56	40.75	+3.01%	710.38	✓
		0.47	0.22		41.14	+3.99%	346.03	✓
		0.94	0.44		13.24	N/A	114.23	✗

As shown in Figure 3.11, the environments and the paths created are intended to test the deliberative controller for global path following in a static environment. The scenarios were designed to be representative of various challenges the robot might face: long straight paths, turns of up to 90°, sharp curves, or narrow paths, to verify the accuracy and safety of the traversal.

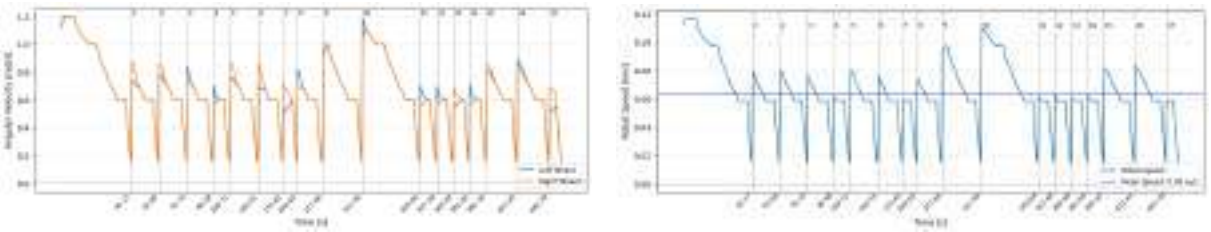
In Scenario A, within a 5×5 meter environment featuring an obstacle density of 61.4%, 17 waypoints were established between the start and goal points, resulting in a path length of 20.60 meters. Figure 3.12 illustrates the path-following results in Scenario A at three different speeds. The paths executed at speed limits of 0.16 m/s and 0.47 m/s completed the route without complications. However, the path at a speed limit of 0.94 m/s failed to correctly reach the 4^{th} waypoint, resulting in a collision. Notably, the performance at 0.47 m/s in terms of accuracy is comparable to that using 0.16 m/s, with a minimal distance difference of only 41 centimeters.



(a) Speed limit 0.16 $[m/s]$ (b) Speed limit 0.47 $[m/s]$ (c) Speed limit 0.94 $[m/s]$

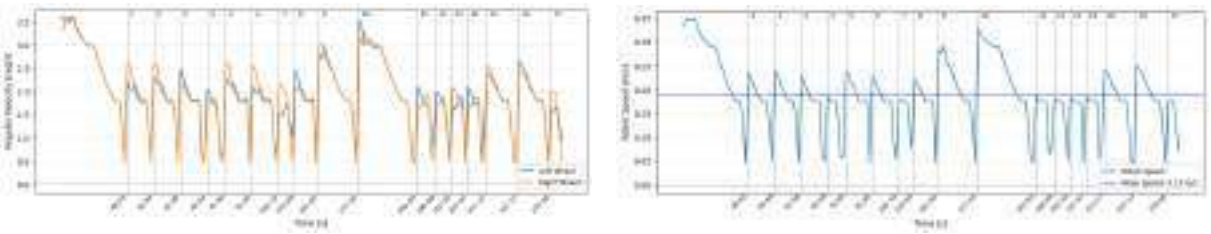
Figure 3.12: Path Following Control in Scenario A using three different speed limits

The average speeds in Scenario A were the lowest recorded among the three scenarios because the computation of speeds in the FLC is proportional to the distance. Therefore, being the smallest scenario, the distance between waypoints is not as long, resulting in a smaller average speed. The profiles of the angular velocities of the wheels and the linear robot speed can be seen in Figures 3.13, 3.14, and 3.15 for each of the three speed limits, respectively. These figures indicate the moment when the robot reaches each waypoint (numbered as in Figure 3.11a), but intervals where the robot automatically orients itself to the next local goal are not plotted. The profiles clearly show the FLC control, where a decrease in speed is observed as the robot approaches the waypoint. Another point to highlight is that in the case of the lowest speed limit, the oscillation and difference between the angular speeds of the wheels are smaller, which increases as the speed limit rises. All these oscillations cause the loss of precision, leading the robot to miss the corresponding waypoint and consequently collide.



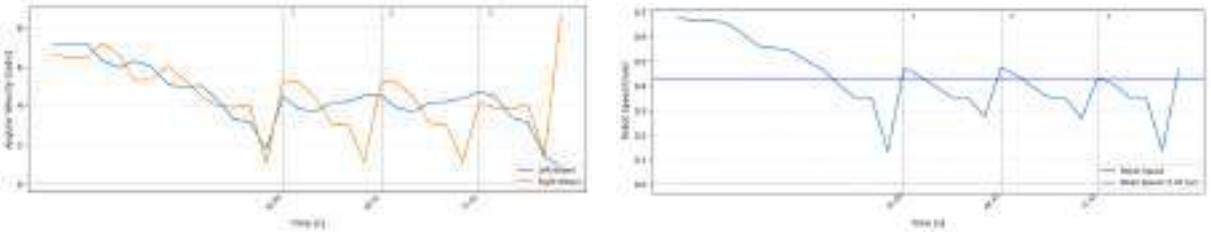
(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.13: Path Following Control in Scenario A with 0.16 $[m/s]$ as speed limit



(a) Wheels' Angular Velocity (b) Robot Speed

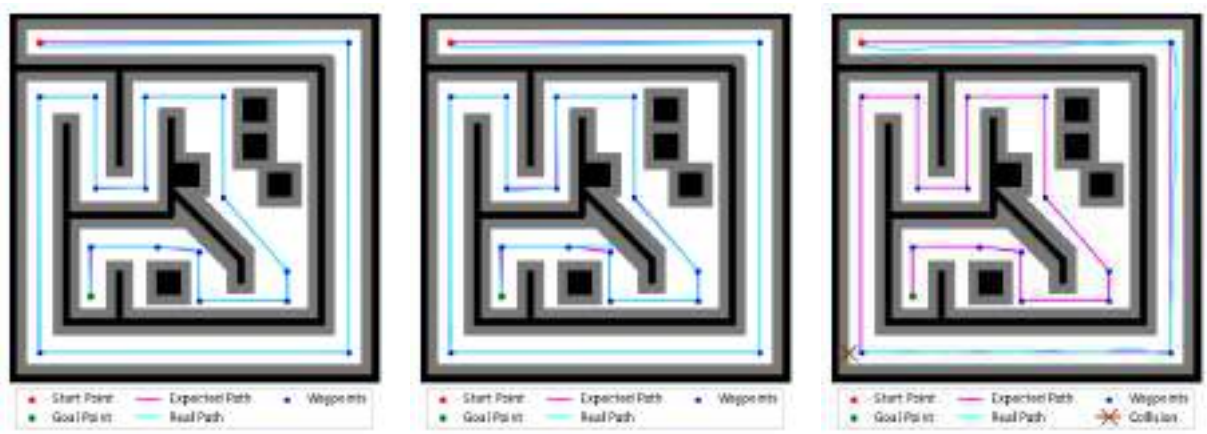
Figure 3.14: Path Following Control in Scenario A with 0.47 $[m/s]$ as speed limit



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.15: Path Following Control in Scenario A with 0.94 $[m/s]$ as speed limit

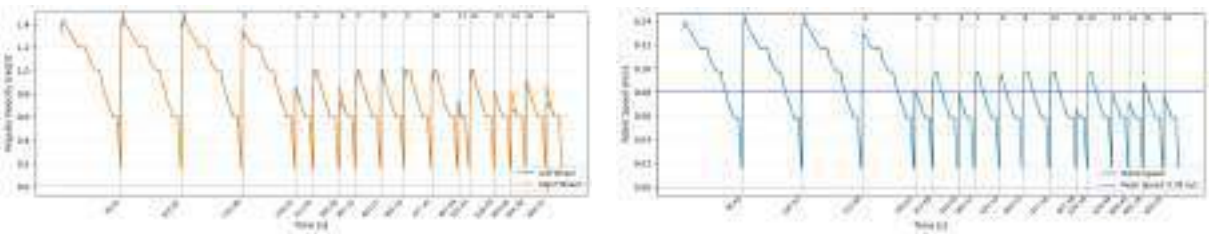
In Scenario B (7.5×7.5 meters) with an obstacle density of 54.03%, 16 waypoints were created to complete a path of 42.19 meters. Figure 3.16 illustrates the path-following results in Scenario B at three different speeds. Similar to the results in Scenario A, only the path following executed at speed limits of 0.16 m/s and 0.47 m/s completed the route. Again, the control performed at a speed limit of 0.94 m/s failed to correctly complete the route, colliding after failing to pursue the 3^{rd} waypoint. Notably, the performance using 0.47 m/s achieved accuracy comparable to that using 0.16 m/s (47 centimeters) but significantly reduced the time required.



(a) Speed limit 0.16 $[m/s]$ (b) Speed limit 0.47 $[m/s]$ (c) Speed limit 0.94 $[m/s]$

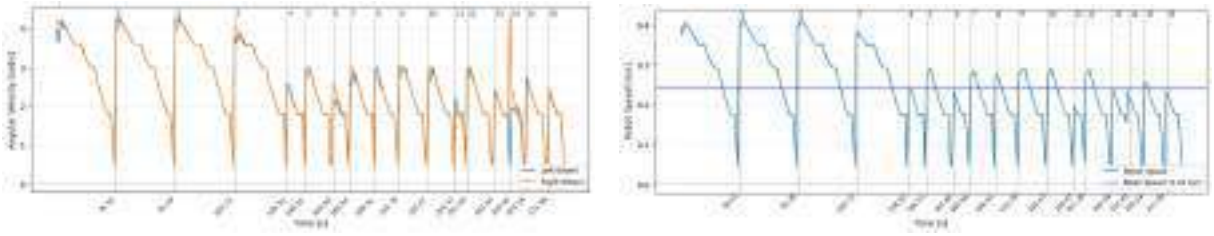
Figure 3.16: Path Following Control in Scenario B using three different speed limits

The speed profiles of the three tests in Scenario B are presented in Figures 3.17, 3.18, and 3.19. This was the scenario where the highest average speeds were achieved, as there were segments of nearly 7 meters in length (the highest distance considered in the design of the membership functions of the FLC). Similar to the previous scenario, the speed oscillations increase in the test at 0.94 m/s, causing the robot to fail to move in a straight line and consequently miss the waypoint. The tests with a speed limit of 0.47 m/s are practically proportional to the behavior observed in the tests using 0.16 m/s, implying equivalent functionality but at a faster pace.



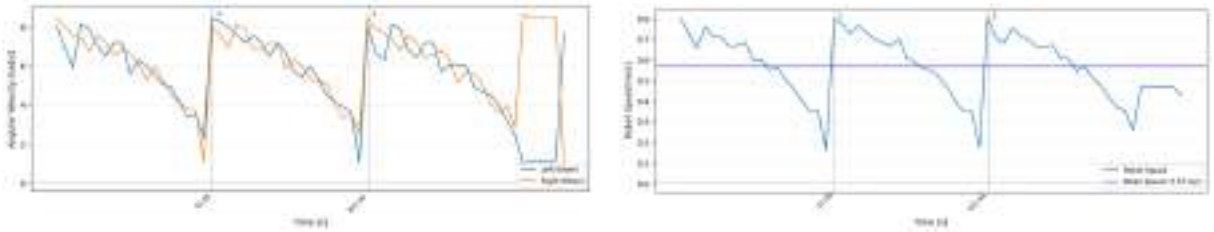
(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.17: Path Following Control in Scenario B with 0.16 $[m/s]$ as speed limit



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.18: Path Following Control in Scenario B with 0.47 $[m/s]$ as speed limit



(a) Wheels' Angular Velocity (b) Robot Speed

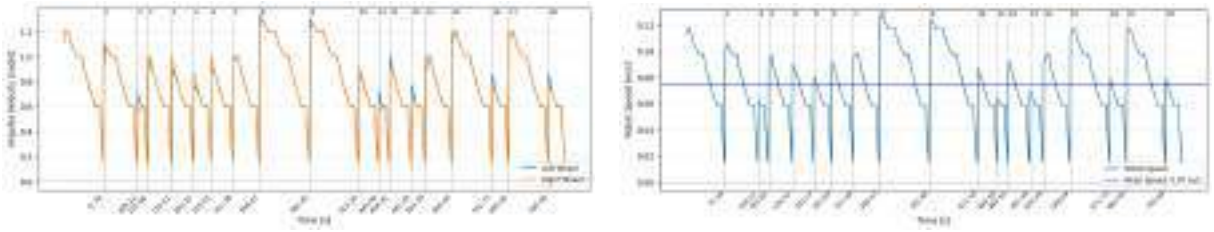
Figure 3.19: Path Following Control in Scenario B with 0.94 $[m/s]$ as speed limit

Finally, the paths executed in Scenario C of 10×10 meters (Figure 3.20), with an obstacle density of 45.57%, show consistency in the results considering the two previous scenarios: the path following executed at speed limits of 0.16 m/s and 0.47 m/s completed the route, whereas the control performed at a speed limit of 0.94 m/s failed to complete the task, colliding at the 5^{th} waypoint of the 18 created to form the 39.56 meter path.

(a) Speed limit 0.16 $[m/s]$ (b) Speed limit 0.47 $[m/s]$ (c) Speed limit 0.94 $[m/s]$

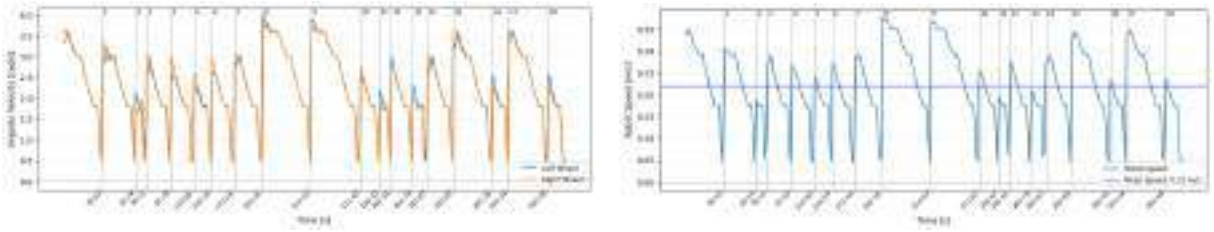
Figure 3.20: Path Following Control in Scenario C using three different speed limits

The speed profiles of the executions in Scenario C are shown in Figures 3.21, 3.22, and 3.21 for the speed limits of 0.16 m/s, 0.47 m/s, and 0.94 m/s, respectively. The first two tests show similar but scaled speeds, while once again, the oscillation present when using the highest speed causes the robot to collide.



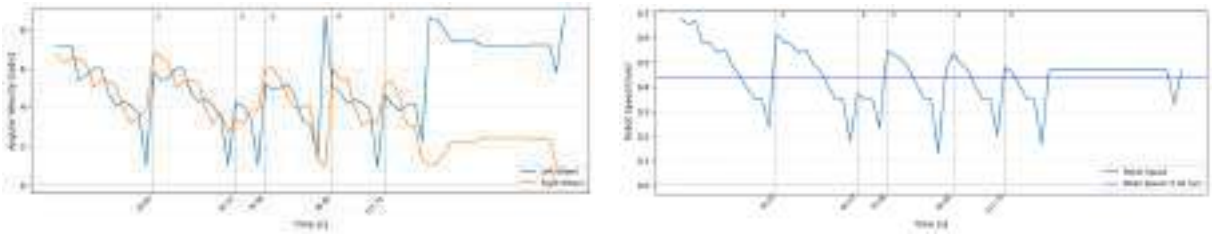
(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.21: Path Following Control in Scenario C with 0.16 $[m/s]$ as speed limit



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.22: Path Following Control in Scenario C with 0.47 $[m/s]$ as speed limit



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.23: Path Following Control in Scenario C with 0.94 $[m/s]$ as speed limit

As conclusions from this experiment, whose results are depicted in the previous Figures and summarized in Table 3.4, it can be confirmed that the design of the FLC controller for path following with a speed limit of 0.16 m/s is satisfactory in terms of accuracy and safety. This configuration allowed the robot to complete the routes in all three scenarios with a maximum deviation of +3.84% from the expected path (observed in Scenario A). However, the robot moves very slowly with this speed limit.

On the other hand, using a speed limit approximately three times the original (0.47 m/s) also shows consistent results across all three scenarios, with a maximum deviation of +5.83% and a difference of no more than 50 centimeters compared to the routes followed at 0.16 m/s, but in nearly half the time. The trade-off discussion would be to determine whether completing the route faster compensates for the small deviation. It seems initially possible to sacrifice some accuracy to complete the route faster without collisions, but this needs to be further discussed.

Meanwhile, the three tests conducted with a speed limit of 0.94 m/s resulted in collisions. This demonstrates that, as defined, the FLC controller cannot handle high speeds due to oscillations in angular velocities and a high tendency to deviate from the path. To move the robot faster, the controller would need to be completely redesigned.

Furthermore, it is important to mention that this result works well for the characteristics of these environments: static scenarios with global knowledge of the obstacles. The speed limit of 0.47 m/s may not be effective in dynamic scenarios where reactive capabilities are required.

3.3. Local Path Planning Test

Previously, the strategy had only been tested in static and global scenarios, using the deliberative control allowed by the total knowledge of the elements in the scene, whose configuration did not change. Therefore, this section intends to verify the performance of the reactive motion control using the sense-to-avoid system based on optical flow in the presence of static and dynamic obstacles unknown to the robot and detected locally by the perception system. Specifically, the effectiveness of the Obstacle Avoidance Controller (see page 95) designed during the *Local Planning Subsystem* section is evaluated. All experiments were executed in real-time mode.

3.3.1. Experiment on the Texture and Shape of Static Obstacles

This experiment aims to evaluate the performance of the OAC in the presence of various unknown static obstacles that appear in the robot's path. For this, a simple 5×5 meter scenario in CoppeliaSim is used. The robot will initially move using the FLC at a speed limit of 0.16 m/s, following a global path that takes it from one corner to another, passing through the center. However, an obstacle with certain characteristics will be placed in the middle of its path, which will trigger the OAC to avoid it (Figure 3.24).

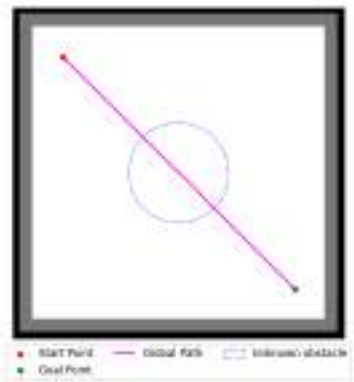
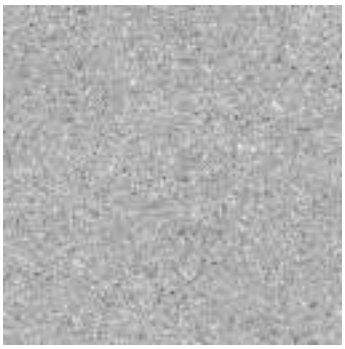


Figure 3.24: Simple Global Path that crosses the center of the scene

Since the idea is to demonstrate the effectiveness of collision avoidance given the characteristics of static obstacles, the path to follow will always be the same. Therefore, the variables in this experiment will be the texture and shape of the obstacles obstructing the path. Three representative textures were selected, illustrated in Figure 3.25: 1) Brick; 2) Wood; and 3) Cement. Additionally, four 3D geometric shapes were selected to represent dummy obstacles (Figure 3.26): 1) Sphere, 2) Cylinder, 3) Cone, and 4) Cuboid. For each obstacle, the maximum dimensions were set at $1 \times 1 \times 1$ meters.

(a) Brick texture

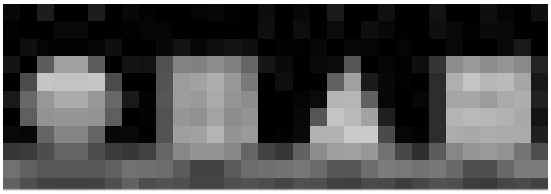
(b) Wood texture



(c) Cement texture

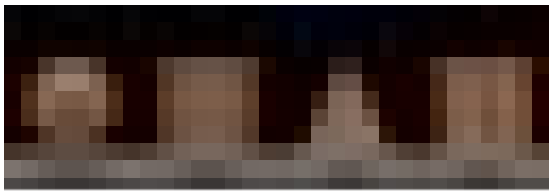
Figure 3.25: Textures used in dummy obstacles

(a) 3D Geometric shapes (Top view)

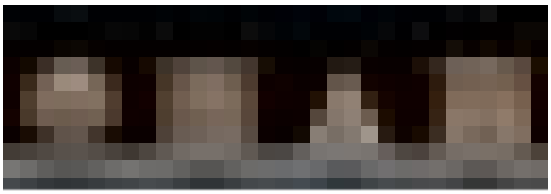


(b) 3D Geometric shapes (Front view)

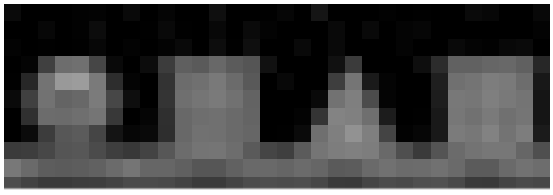
Figure 3.26: 3D Geometric shapes used to represent dummy obstacles



(a) 3D Geometric shapes with brick texture



(b) 3D Geometric shapes with wood texture



(c) 3D Geometric shapes with cement texture

Figure 3.27: 3D Geometric shapes with proposed textures

In total, the number of tests conducted is equal to the combination of the obstacle characteristics (4 shapes and 3 textures, as presented in Figure 3.27), resulting in 12 measurements, whose results are shown in Table 3.5.

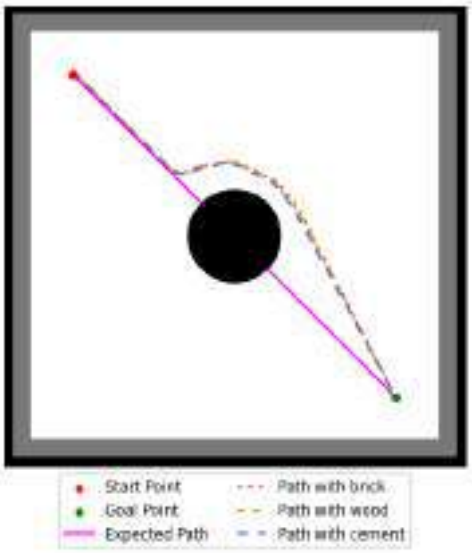
Table 3.5: Experiment on the Texture and Shape of Static Obstacle Results

Obstacle shape	Obstacle dimensions $[m]$	Obstacle texture	Real path distance $[m]$	Path completed
Sphere	$D = 1$	Brick	5.39	✓
		Wood	5.38	✓
		Cement	5.37	✓
Cylinder	$D = 1$ $H = 1$	Brick	5.52	✓
		Wood	5.51	✓
		Cement	5.50	✓
Cone	$D = 1$ $H = 1$	Brick	5.41	✓
		Wood	5.38	✓
		Cement	5.37	✓
Cube	$L = 1$	Brick	5.60	✓
		Wood	5.58	✓
		Cement	5.56	✓

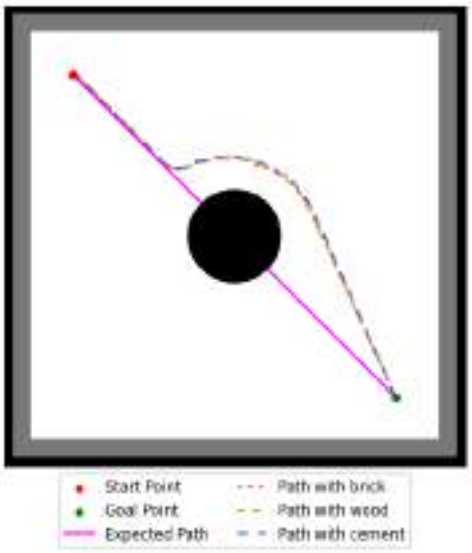
The first observation from the results is that in all cases, the route was completed. This means that given the conditions set for the experiment, the shape and texture of the obstacles do not represent a significant change in the performance of the local planning. However, many aspects can be analyzed.

Figure 3.28 graphically shows the paths followed by the robot when the obstacle was a sphere (Figure 3.28a), a cylinder (Figure 3.28b), a cone (Figure 3.28c), and a cuboid (Figure 3.28d). In each subfigure, the three paths (given the three added textures) are shown. As can be observed, for each shape, the paths are practically the same regardless of the texture, with only a small difference in the real path distance traveled, as confirmed by the data in Table 3.5.

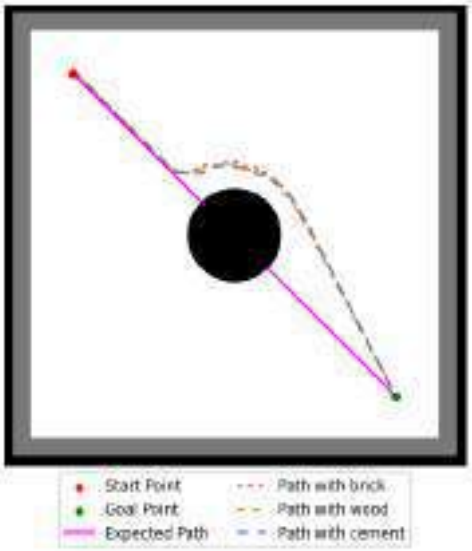
This difference is graphed in Figure 3.29. It can be seen that in all cases, when the obstacle had the brick texture, the distance traveled was greater, followed by the wood texture, and finally, the shortest distances were obtained when the obstacle texture was cement. Although this difference in the paths traveled is marginal, 4 cm in the worst-case scenario, it could be determinant when moving in more congested environments.



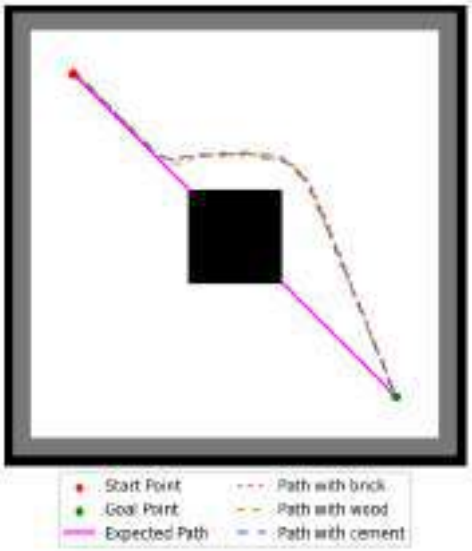
(a) Path with sphere as obstacle



(b) Path with cylinder as obstacle



(c) Path with cone as obstacle



(d) Path with cube as obstacle

Figure 3.28: Local Paths followed for each combination of texture and shape

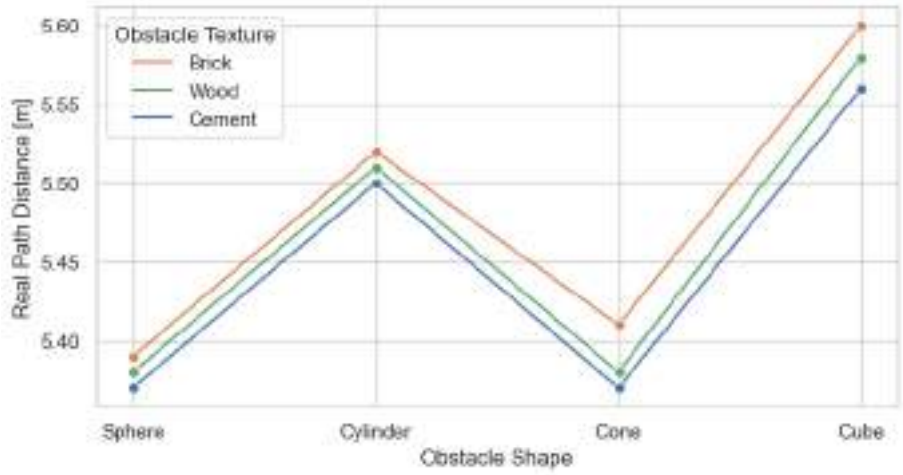
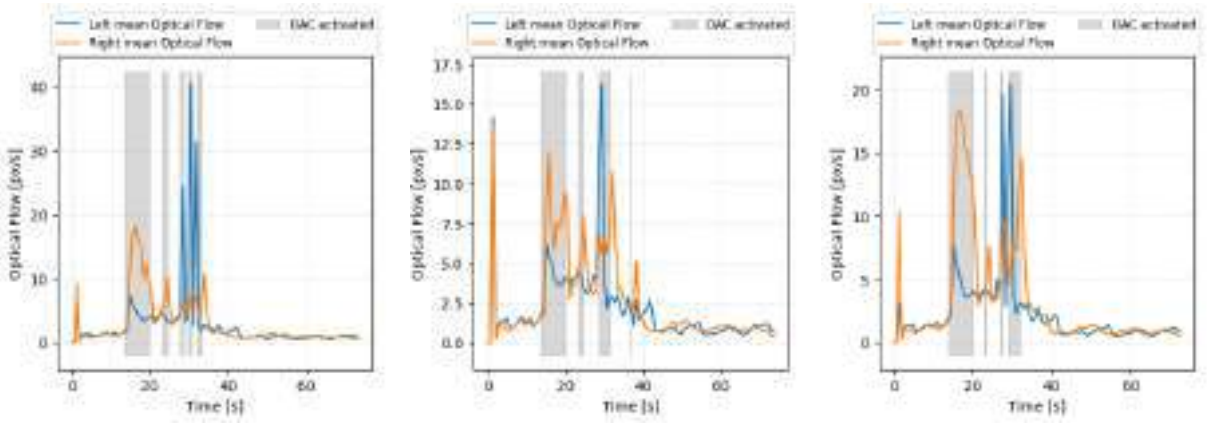


Figure 3.29: Comparison of distance traveled in the test of different textures and shapes

Another noteworthy point is that the sphere, cylinder, and cone can be considered equivalent in terms of the space they occupy, as all three obstruct a circular area of 1 meter in diameter. However, their 3D shape influences the final path. While the sphere and cone do not occupy the entire vertical space of the circle, the cylinder does, which modifies the estimation of the average optical flow. It can be observed that the results for the sphere and cone are similar, while the cylinder shows a proportionally greater distance traveled.

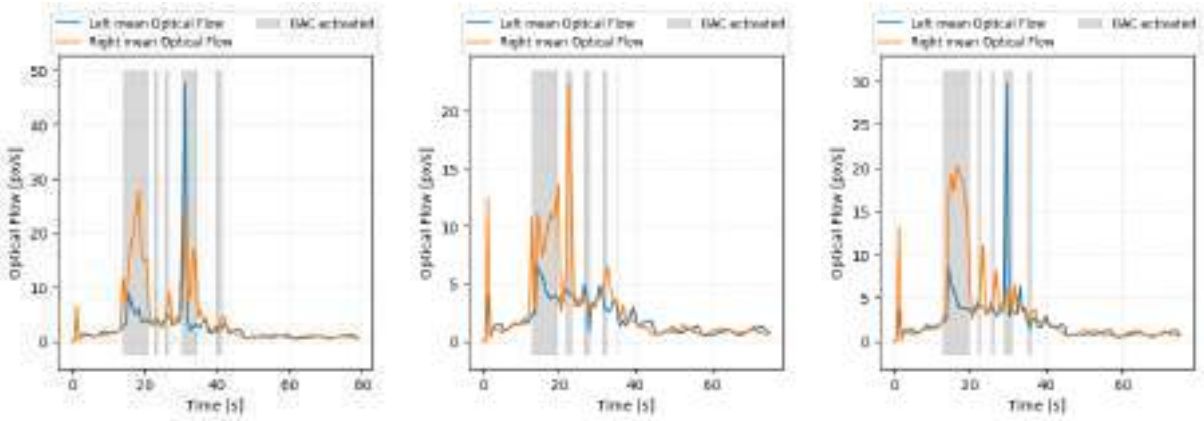
In Figures 3.30, 3.31, 3.32, and 3.33, the average estimated optical flow magnitude during paths with different obstacles (sphere, cylinder, cone, cuboid) is shown, highlighting periods of OAC activation. The mean optical flow increases when the robot detects an obstacle in all cases, with a significant peak around 20 seconds, likely due to the robot's abrupt turn to avoid the collision. No pattern is detected in the value of the optical flow magnitude concerning the texture used, unlike the shape and size of the obstacle within the field of view of the vision sensor. For obstacles like the cement cuboid and cylinder, which take up a larger space in the vision sensor image, a second higher peak on the left side is observed. This occurs because the robot needs to deviate slightly more and make a sharper right turn to reorient towards the goal position.

When the cuboid is used as an obstacle, 5-6 OAC activation zones are detected, compared to a maximum of 5 for the cylinder and 3-4 for the cone and sphere. This is due to the ultrasonic sensors and the shapes of the obstacles. The cuboid and cylinder occupy more vertical space, making them easier to detect and resulting in more frequent activations. The cuboid's extra corner also contributes to this. In contrast, the cone and sphere have shapes that make detection harder, leading to fewer activations.



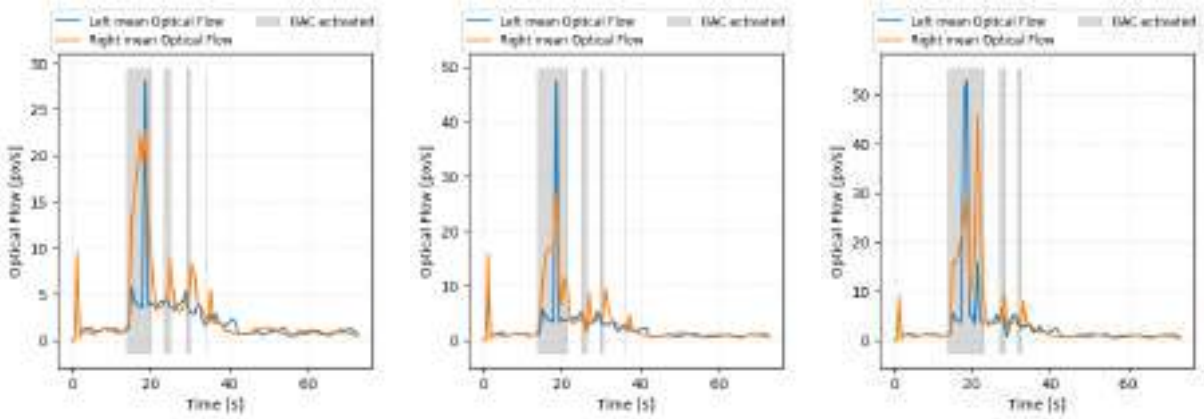
(a) Brick texture (b) Wood texture (c) Cement texture

Figure 3.30: Optical Flow Magnitude estimations with a sphere as obstacle



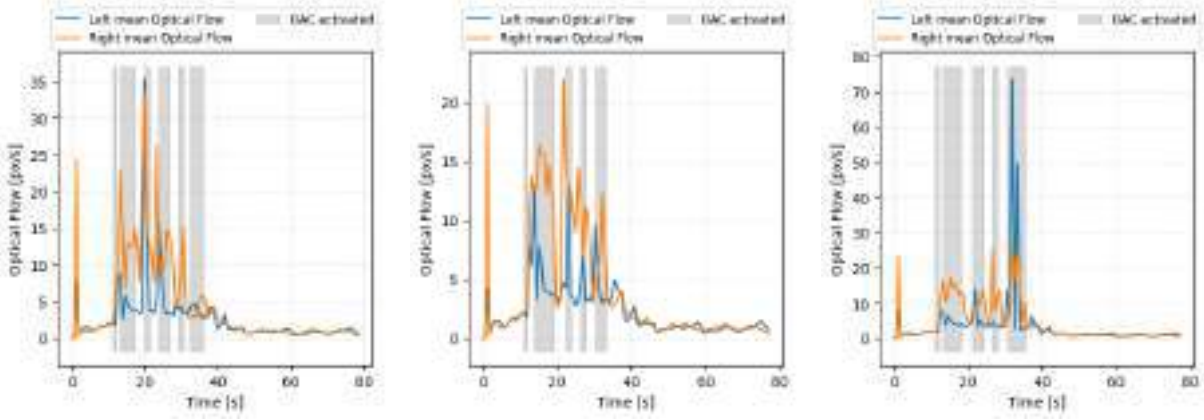
(a) Brick texture (b) Wood texture (c) Cement texture

Figure 3.31: Optical Flow Magnitude estimations with a cylinder as obstacle



(a) Brick texture (b) Wood texture (c) Cement texture

Figure 3.32: Optical Flow Magnitude estimations with a cone as obstacle



(a) Brick texture (b) Wood texture (c) Cement texture

Figure 3.33: Optical Flow Magnitude estimations with a cube as obstacle

3.3.2. Experiment on the Navigation Skills with Unknown Static Obstacles

Given that the effectiveness of the OAC has been proven against various types of obstacles, its performance is now tested in a scenario filled with unknown static obstacles. This means that the only way for the robot to detect and avoid them is through its local perception system. In these scenarios, the robot will start moving using the FLC control to navigate from one point to another and will activate the OAC control when necessary.

For this purpose, three 7.5×7.5 meter scenarios filled with unknown static obstacles were designed, where the robot must move and reach three different positions (P_1, P_2, and P_3) within the same scene (as shown in Figure 3.34).

(a) Scenario A

(b) Scenario B

(c) Scenario C

Figure 3.34: Scenarios to test the navigation skills with unknown static obstacles

The obstacles will have various shapes, sizes, and textures, similar to those in the previous experiment. When the robot moves using the FLC control, meaning it does not detect an obstacle, two-speed limits will be used to evaluate the reactive capability: the original 0.16 m/s and 0.47 m/s (three times the original). These speeds have already been tested for path-following control in globally known static environments (see page 118). In this case, if the robot collides, the simulation will not be stopped; however, it will be considered as an incomplete path.

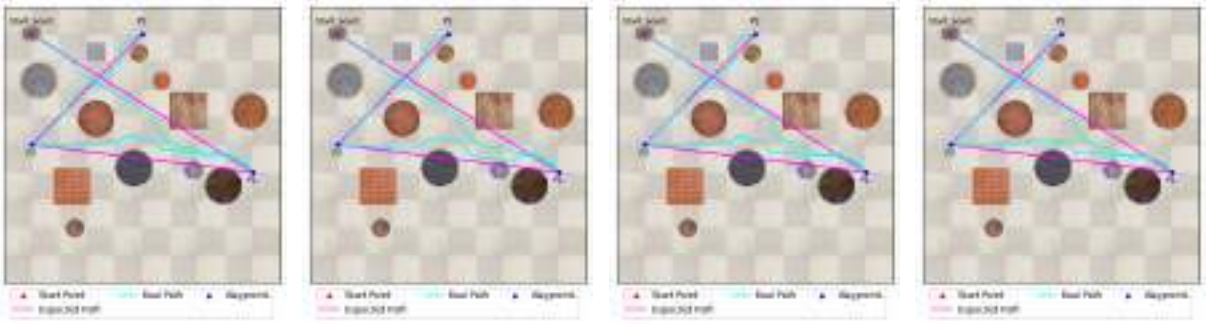
In total, 6 configurations will be tested, resulting from 3 scenarios and 2 speed limits. However, for each configuration, two trials will be conducted, resulting in a total of 12 measurements, which can be observed in Table 3.6.

Table 3.6: Experiment on the Navigation Skills with Unknown Static Obstacles Results

Scenario	Speed limit $[m/s]$	Attempt 1				Attempt 2			
		Mean speed $[m/s]$	Distance $[m]$	Time $[s]$	Path completed	Mean speed $[m/s]$	Distance $[m]$	Time $[s]$	Path completed
A	0.16	0.07	18.40	300.15	✓ ✓ ✓	0.07	18.53	292.38	✓ ✓ ✓
	0.47	0.16	18.59	168.21	✓ ✓ ✓	0.16	18.43	165.64	✓ ✓ ✓
B	0.16	0.07	17.60	278.45	✓ ✓ ✓	0.07	17.59	274.54	✓ ✓ ✓
	0.47	0.16	17.38	150.97	✗ ✗ ✓	0.16	17.58	150.75	✓ ✗ ✓
C	0.16	0.07	19.15	299.62	✓ ✓ ✓	0.07	19.11	295.01	✓ ✓ ✓
	0.47	0.16	19.80	161.48	✗ ✗ ✗	0.15	19.60	175.55	✗ ✗ ✗

It can be observed that the time taken to complete the paths is significantly greater for the FLC limit of 0.16 m/s compared to 0.47 m/s. This is natural, as the lower speed limit means the robot moves more slowly. However, for the 0.16 m/s limit, the robot completes all paths in all scenarios, whereas for the 0.47 m/s limit, there are more failures in completing the paths, especially in scenarios B and C. This indicates that although the navigation time is shorter, as the FLC and OAC controllers were designed, at higher speeds the probability of collisions and incomplete trajectories increases, indicating a trade-off between speed and obstacle avoidance ability. This difficulty is particularly evident in scenario C, where both attempts using 0.47 m/s limit fail to reach the three positions P_i.

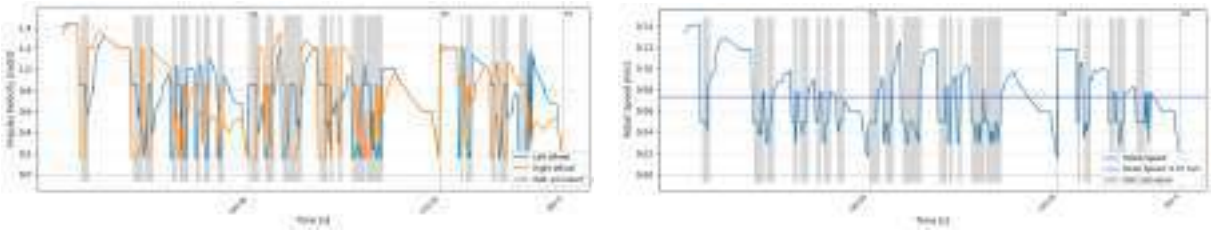
In particular, scenario A was designed for the WMR to traverse the scene from side to side through relatively clear paths but with the need to avoid some obstacles. This configuration allowed the robot to cross successfully on all occasions without colliding, as presented in Table 3.6 and observed in Figure 3.35.



(a) Attempt 1 - 0.16 m/s (b) Attempt 2 - 0.16 m/s (c) Attempt 1 - 0.47 m/s (d) Attempt 2 - 0.47 m/s

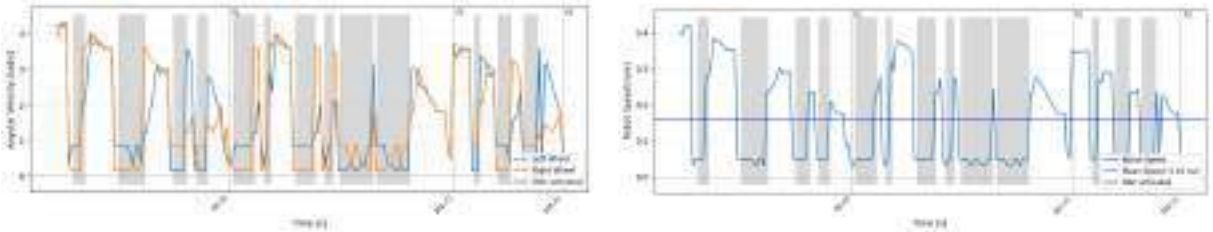
Figure 3.35: Local Navigation results in Scenario A with unknown static obstacles

Although all paths were completed in Scenario A, a potentially undesirable behavior can be observed when using 0.47 m/s as the speed limit. The angular wheel speeds range from 0.2 to 4 rad/s, caused by the avoidance speeds of the OAC not being scaled. This large acceleration and sudden braking can result in turbulent and unstable behavior, which is undesirable in delicate situations where the robot must be careful and precise. This wide range also translates to the robot's linear speed, which varies from 0.02 to 0.42 m/s (a 20-fold increase), whereas using the original FLC limit of 0.16 m/s, the robot moves within a range of 0.02 to 0.14 m/s (a maximum 7-fold increase). The velocities resulting from attempt 1 as an example for comparison can be seen in Figures 3.36 and 3.36 for the 0.16 m/s limit and 0.47 m/s limit, respectively.



(a) Wheels' Angular Velocity (b) Robot Speed

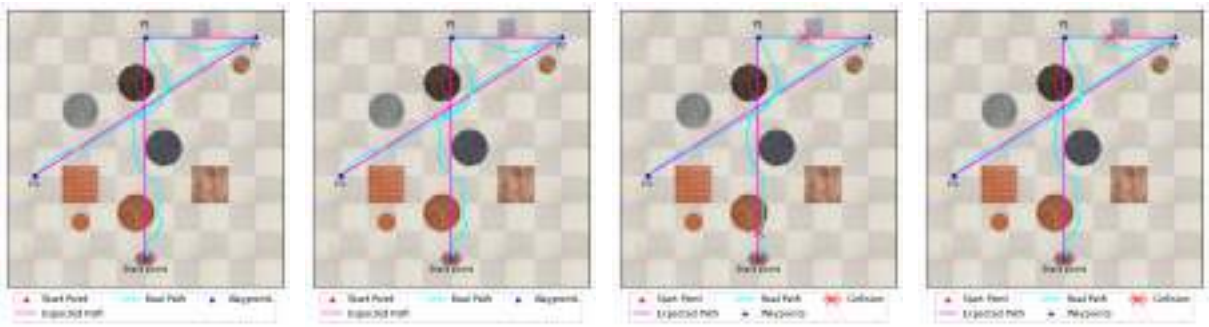
Figure 3.36: Velocities resulting from attempt 1 of local navigation in scenario A with a speed limit of 0.16 m/s



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.37: Velocities resulting from attempt 1 of local navigation in scenario A with a speed limit of 0.47 m/s

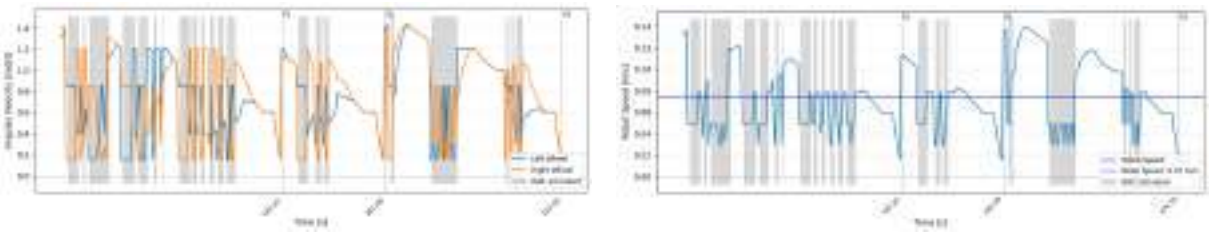
On the other hand, scenario B was designed to have more obstacles in the middle of the path, which would require the robot to weave from left to right to avoid them (Figure 3.38).



(a) Attempt 1 - 0.16 m/s (b) Attempt 2 - 0.16 m/s (c) Attempt 1 - 0.47 m/s (d) Attempt 2 - 0.47 m/s

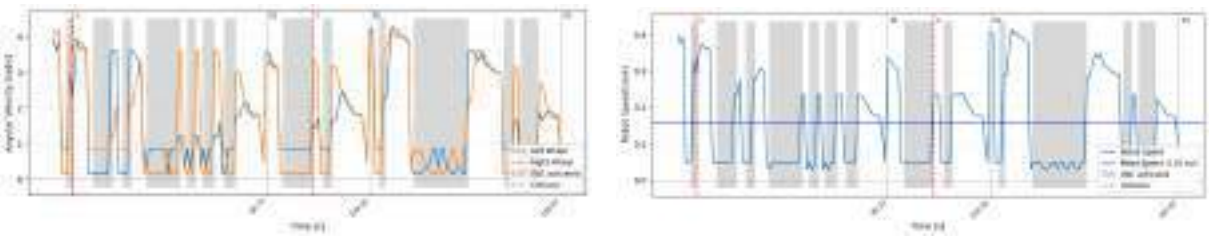
Figure 3.38: Local Navigation results in Scenario B with unknown static obstacles

For this scenario, the resulting speeds of the best case using the 0.16 m/s limit (attempt 2) are shown in Figure 3.39, and the worst case using the 0.47 m/s limit (attempt 1, 2 collisions) are shown in Figure 3.40. In this case, the higher speed and the presence of a cone as an obstacle caused the robot to fail to detect the obstacle in time and collide the first time. In fact, in the speed profiles for the 0.47 m/s limit, it can be observed in red that just when the OAC zone deactivates, the robot collides while heading to P_1. Conversely, in the case of the 0.16 m/s limit, the OAC zone remains active for a longer period, successfully avoiding the obstacle.



(a) Wheels' Angular Velocity (b) Robot Speed

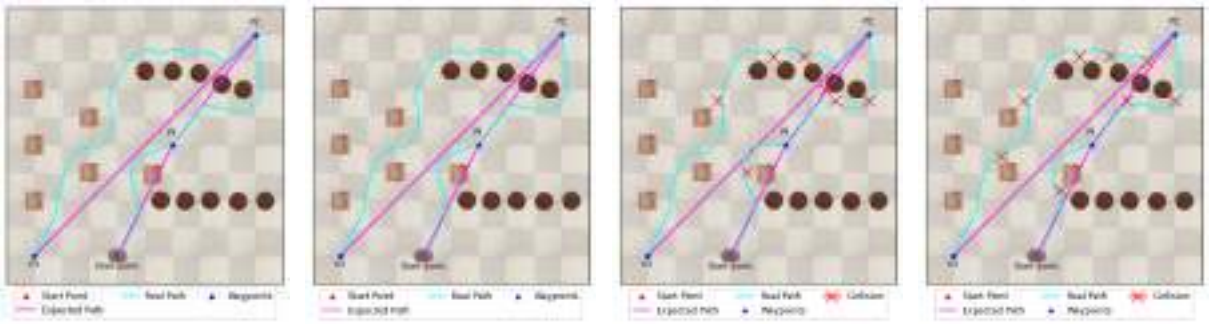
Figure 3.39: Velocities resulting from attempt 2 of local navigation in scenario B with a speed limit of 0.16 m/s



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.40: Velocities resulting from attempt 1 of local navigation in scenario B with a speed limit of 0.47 m/s

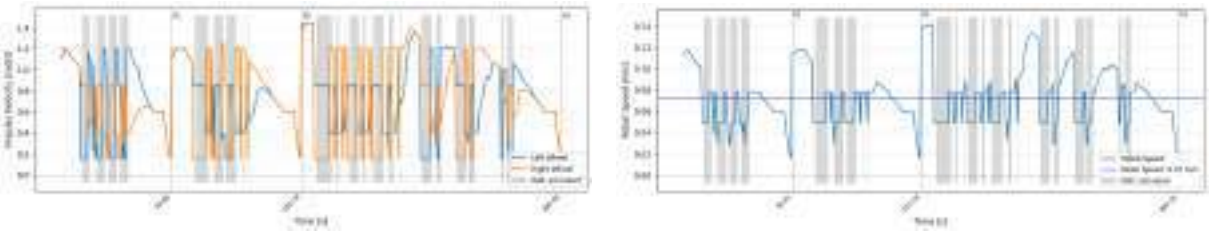
Finally, as seen in Figure 3.41, scenario C was designed to confront the robot with situations where it encounters multiple obstacles directly in front and has to navigate around them.



(a) Attempt 1 - 0.16 m/s (b) Attempt 2 - 0.16 m/s (c) Attempt 1 - 0.47 m/s (d) Attempt 2 - 0.47 m/s

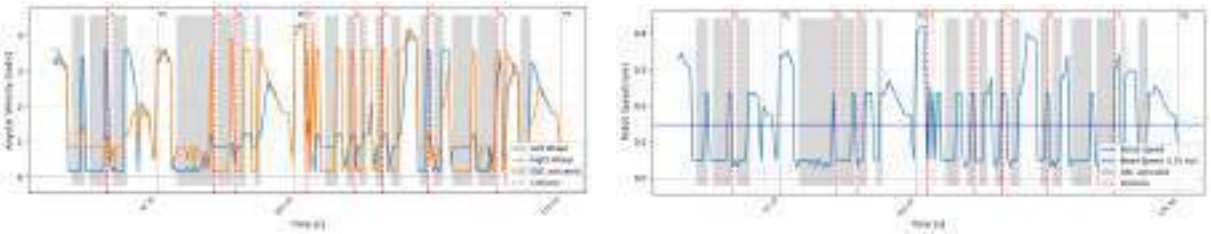
Figure 3.41: Local Navigation results in Scenario C with unknown static obstacles

The situations in Scenario C were the most challenging, requiring navigation around multiple obstacles. The 0.47 m/s speed limit in the FLC was problematic, as the robot collided in all attempts to reach P_1, P_2, or P_3. Most collisions occurred when the OAC deactivated; the high speed caused sharp turns back to the global route, leading to collisions. In contrast, the 0.16 m/s limit avoided sharp turns and large speed differences between FLC and OAC, allowing a gradual return to the global path. Figures 3.42 and 3.43 illustrate the stable behavior with the 0.16 m/s limit and the erratic performance at the higher speed.



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.42: Velocities resulting from attempt 2 of local navigation in scenario C with a speed limit of 0.16 m/s



(a) Wheels' Angular Velocity (b) Robot Speed

Figure 3.43: Velocities resulting from attempt 2 of local navigation in scenario C with a speed limit of 0.47 m/s

3.3.3. Experiment on Reactive Capacity against Dynamic Obstacles

One of the advantages of using the Sense-to-avoid System based on Optical Flow is that the control depends on the movement within the scene, which is captured by the vision sensor located on the robot. Therefore, the robot can react to unknown dynamic obstacles approaching it in addition to static obstacles. This experiment aims to demonstrate the robot's reactive capabilities.

For this experiment, the robot will be placed in a 5×5 meter scenario to reach three positions without any static obstacles but with unknown dynamic objects moving around it (Figure 3.44). Given the difficulty of fully parameterizing a dynamic simulation scenario, the obstacles will be moved manually using CoppeliaSim to test the robot in specific situations. The methodology involves activating the FLC control at a speed limit of 0.16 m/s to follow the predetermined path. At the same time, the operator moves the obstacles so that the OAC can be activated to avoid them. This experiment was conducted four times with different obstacles.

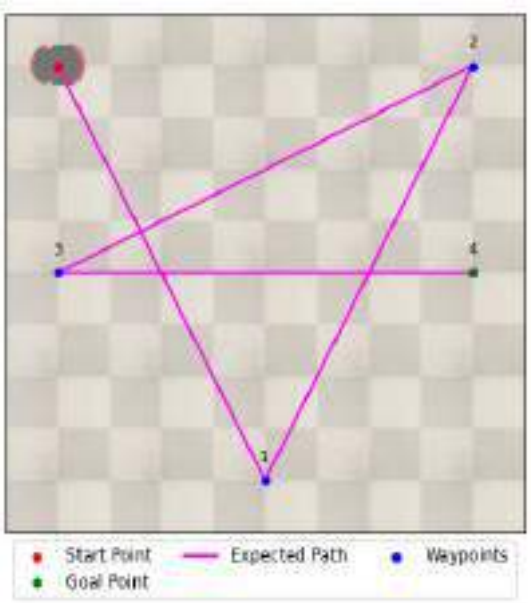


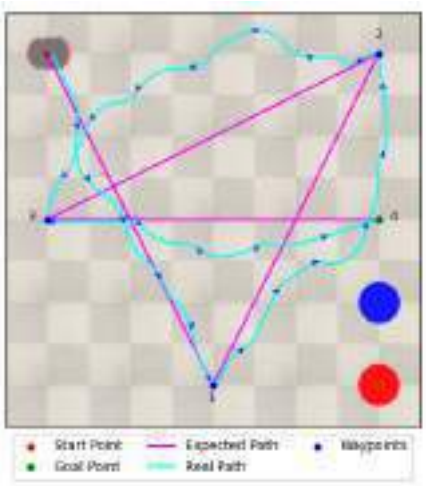
Figure 3.44: Reference path to test reactive capacity against dynamic obstacles

Table 3.7 shows the total distance traveled and the simulation time of the robot in each of the four tests. Due to the high variability and randomness of the experiment, it is impossible to determine an objective parameter for comparing and evaluating the results, not even by the number of collisions or the distance traveled, given the active intervention of the experiment designer. Nevertheless, the important information collected, such as speed profiles and optical flow estimation, is presented for analysis.

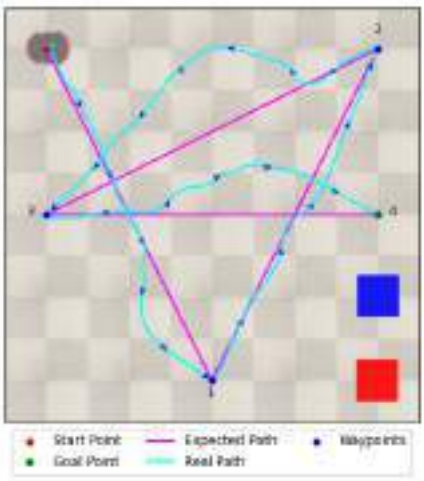
Table 3.7: Experiment on Reactive Capacity against Dynamic Obstacles Results

Test	Obstacle conditions	Distance traveled $[m]$	Total simualtion time $[s]$
1	Two cylinders $1m$ height and $0.5m$ diameter	19.41	354.33
2	Two cuboids of $1m$ height and $0.5m$ on each side	19.41	374.57
3	One cylinder $1m$ high and $1m$ diameter	19.68	354.37
4	One cube $1m$ on each side	18.41	337.63

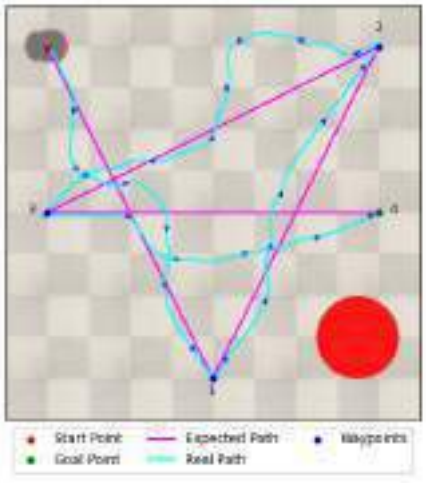
Figure 3.45 shows the actual path followed by the robot for each of the four tests. Each subfigure displays the obstacle or obstacles used as dynamic objects to actively divert the robot. This movement cannot be seen in a printed image, but what can be observed is how the robot deviates from the expected path.



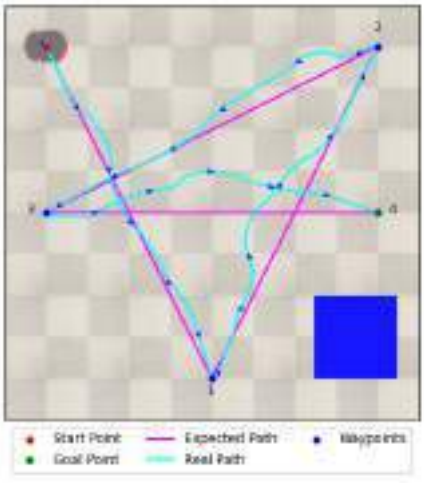
(a) Test 1



(b) Test 2



(c) Test 3



(d) Test 4

Figure 3.45: Resultant path in the presence of unknown dynamic obstacles

For Test 1 (Figure 3.45a), two cylinders (1 meter high, 0.5 meters in diameter) were used as dynamic obstacles. Figure 3.46 shows 50 frames of the robot's path from the Start point to position 1, illustrating its movement when approached by dynamic obstacles. This control was maintained throughout the entire route, as depicted in Figure 3.47, which displays the angular velocity, linear velocity, and optical flow estimation values for the entire 350-second, 19-meter simulation, including OAC activation zones. Generally, the optical flow estimates for the left and right sides remain similar, below 10 px/s (with an outlier peak nearly reaching 25 px/s), but this did not hinder decision-making when the OAC was active.

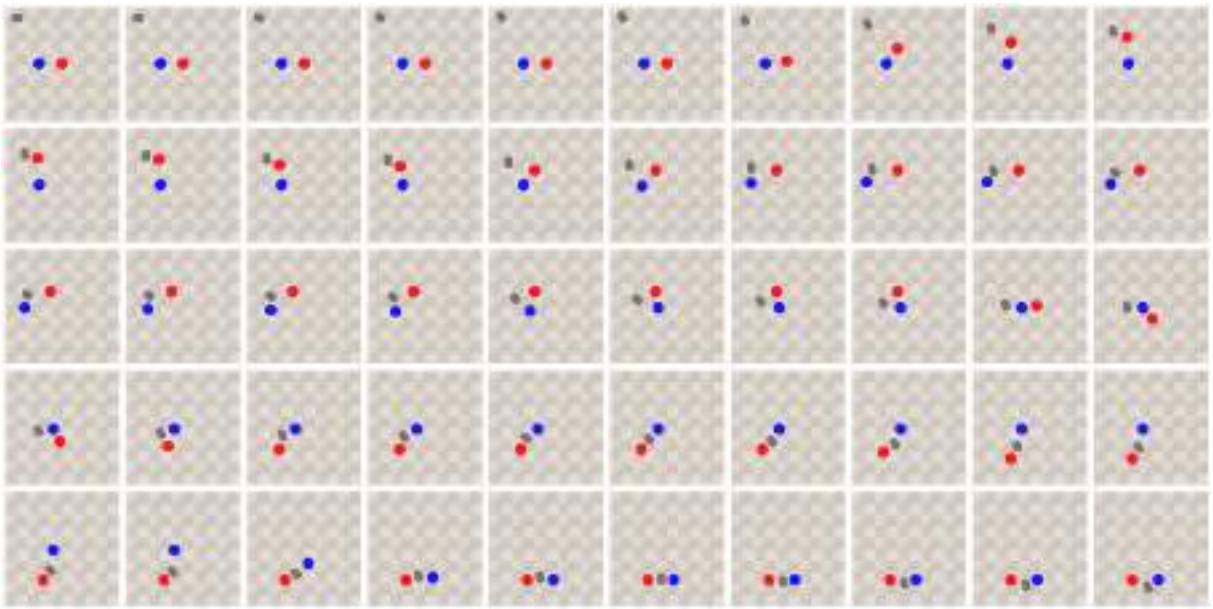
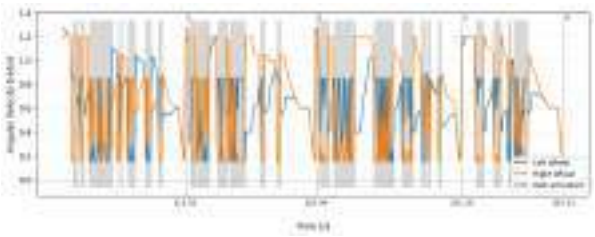
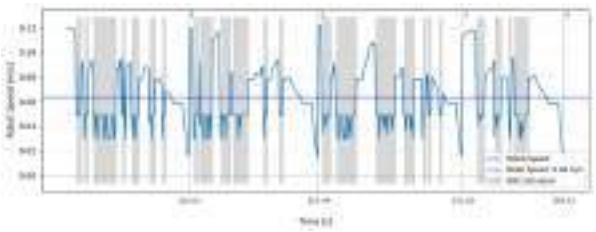


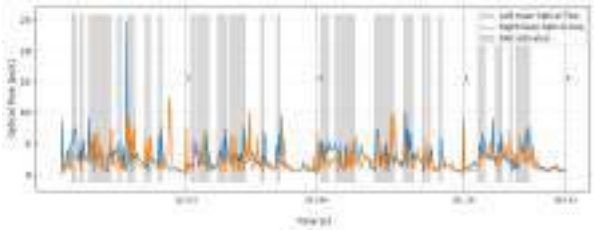
Figure 3.46: Frames of the navigation in Test 1 between start and position 1



(a) Wheels' Angular Velocity



(b) Robot Speed



(c) Optical Flow Magnitude

Figure 3.47: Velocities and Optical Flow estimations from Test 1

For Test 2 (Figure 3.45b), two rectangular prisms (1 meter high, 0.5 meters on each side) were utilized in place of the previously used cylinders, resulting in similar outcomes. The sequence of 50 frames depicting the robot's movement from position 1 to position 2 is presented in Figure 3.48. Concurrently, the results of velocity and optical flow estimations are illustrated in Figure 3.49. During this attempt, an increased level of indecision was observed in the robot's behavior when it was navigating around obstacles. This was attributed to the mean optical flow estimates for both sides being even more closely matched than those recorded in the preceding test. Despite this, the robot demonstrated adequate consistency in its obstacle-avoidance capabilities.

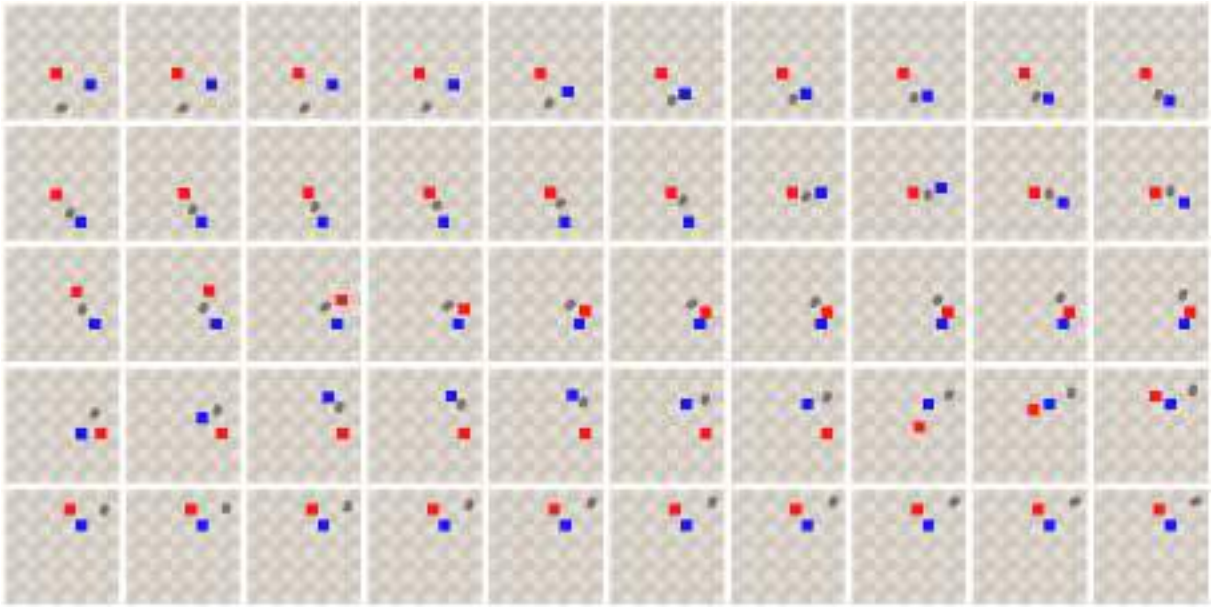
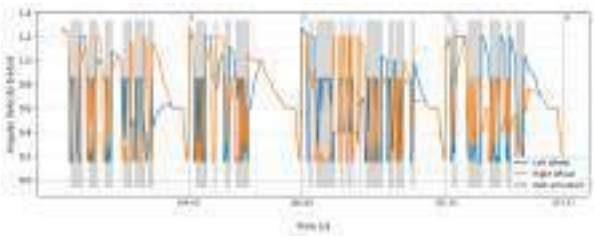
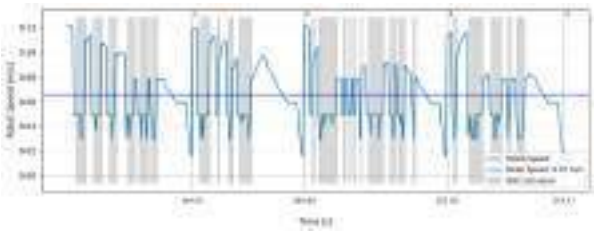


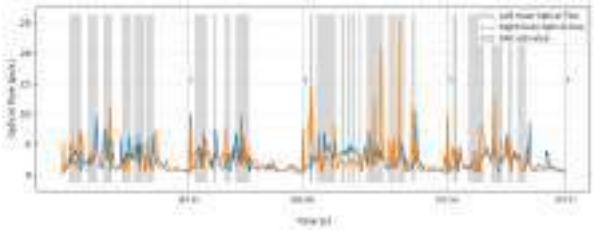
Figure 3.48: Frames of the navigation in Test 2 between positions 1 and 2



(a) Wheels' Angular Velocity



(b) Robot Speed



(c) Optical Flow Magnitude

Figure 3.49: Velocities and Optical Flow estimations from Test 2

For test 3, it was decided to use only one cylinder as an obstacle, but larger (1 meter high and 0.5 meters in diameter). Subjectively, this was the obstacle to which the OAC controller responded best. As seen in Figure 3.45c, the robot deviated more from the expected route than in any other test, covering up to 19.68 meters (the longest distance of the four tests). Figure 3.50 shows the 50 frames of the robot moving from position 2 to 3. As can be observed, the robot even moved horizontally nearly two meters when it had to move diagonally. Figure 3.51 finally shows the velocity profiles and optical flow estimation, illustrating the consistency in control during the entire test.

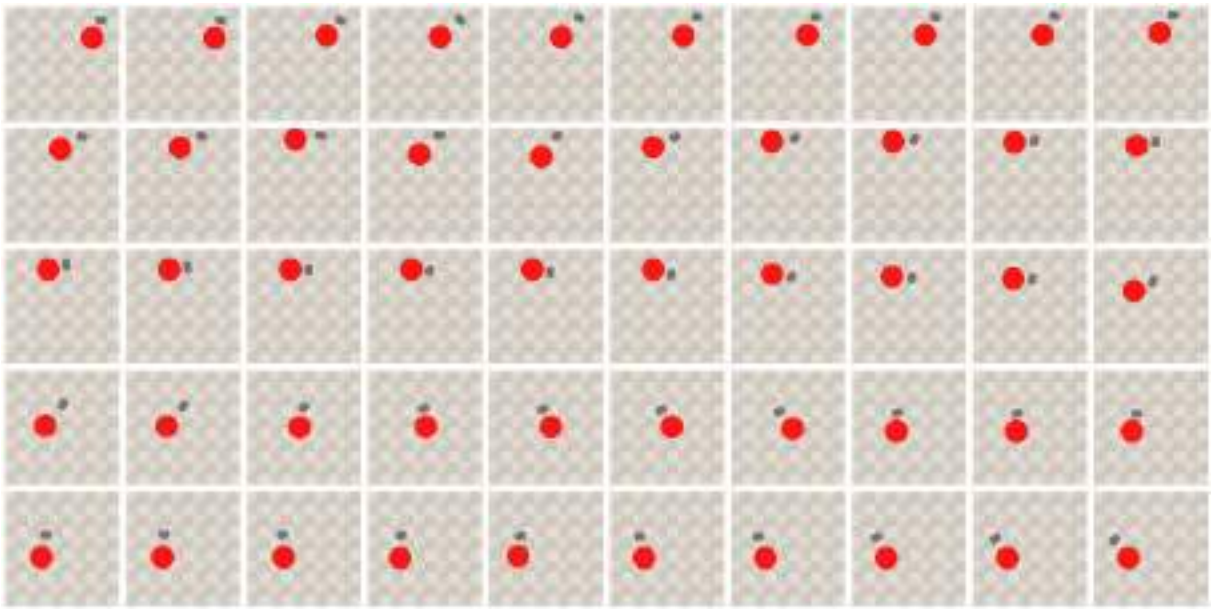
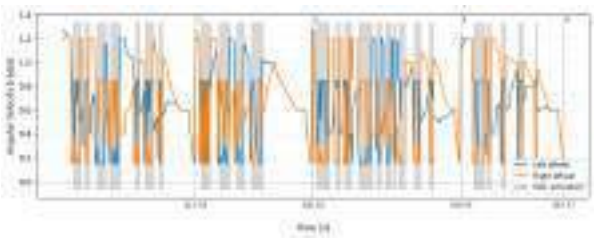
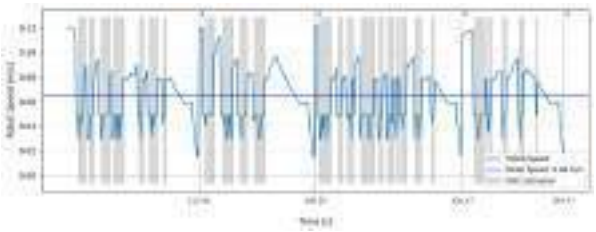


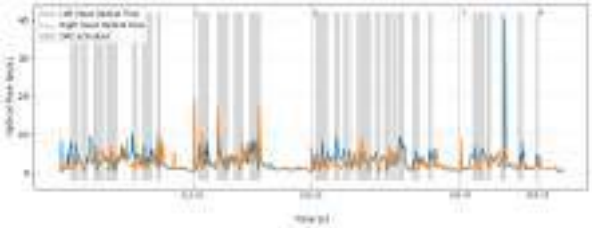
Figure 3.50: Frames of the navigation in Test 3 between positions 2 and 3



(a) Wheels' Angular Velocity



(b) Robot Speed



(c) Optical Flow Magnitude

Figure 3.51: Velocities and Optical Flow estimations from Test 3

In Test 4, a cube with 1 meter on each side was used as the obstacle. The OAC performed the poorest with this obstacle, often requiring active removal to avoid collision. As shown in Figure 3.45d, this test resulted in the least deviation from the expected path due to the robot's low reactive capability. Figure 3.52 illustrates 50 frames of the robot moving between position 3 and position 4, highlighting its inability to directly confront the obstacle. Additionally, Figure 3.53 shows the most consistent optical flow estimates, with minimal variation due to the robot's incorrect decision-making. This could be due to the vision sensor's perspective, either detecting the cube in both image regions or failing to see it when the ultrasonic sensors activated the OAC.

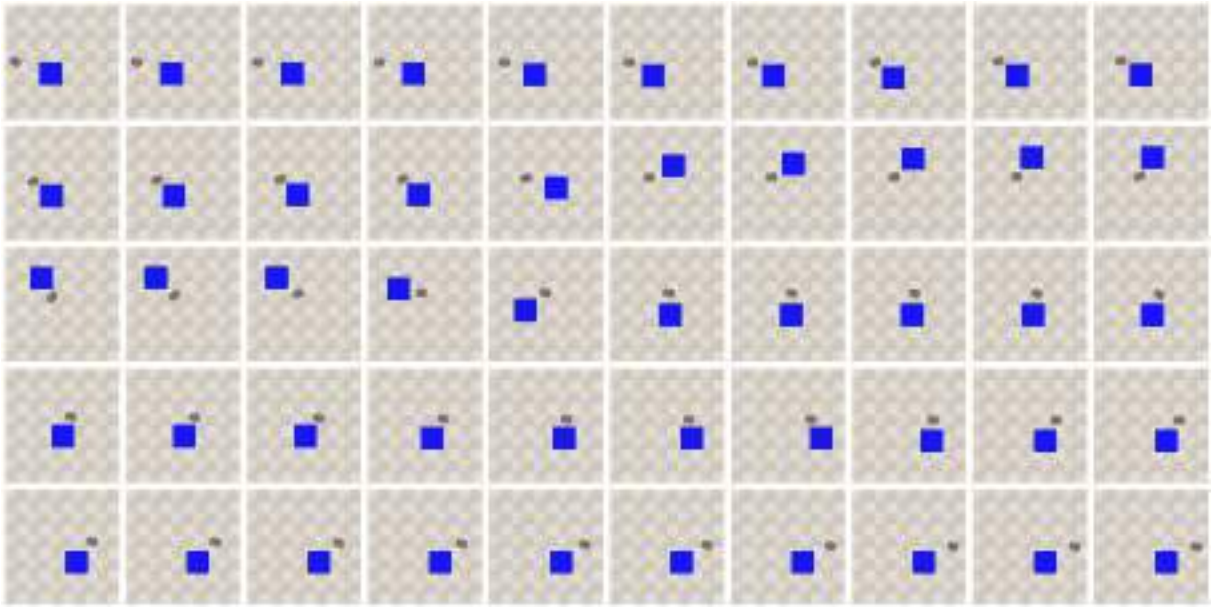
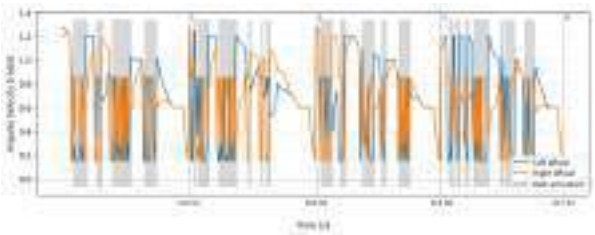
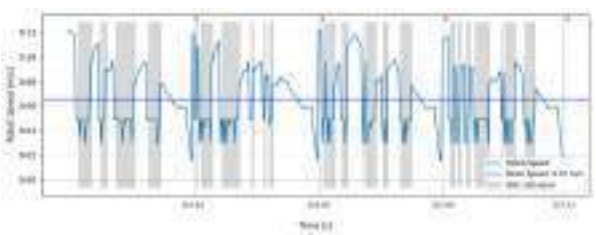


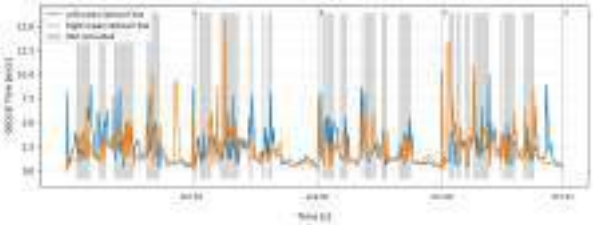
Figure 3.52: Frames of the navigation in Test 4 between positions 3 and 4



(a) Wheels' Angular Velocity



(b) Robot Speed



(c) Optical Flow Magnitude

Figure 3.53: Velocities and Optical Flow estimations from Test 4

3.4. Comprehensive Autonomous Navigation Strategy Test

The previous sections aimed to measure the performance of each component separately. In this section, the objective is to demonstrate the functioning of the complete navigation strategy, showing what the entire process would look like in a real implementation, including global path planning, local path planning, and motion control for path following.

For this purpose, the robot will be placed in two scenarios that meet specific characteristics to ensure the best possible performance of the strategy:

1. An indoor space where a global map can be obtained and where most of the obstacles are usually static and unlikely to move, such as walls or furniture.
2. In this space, there may be new unknown obstacles that can either move around the scenario or remain stationary for a while.
3. The new obstacles cannot directly head toward a collision with the robot.
4. The robot must have the repetitive task of moving to a specific position to act.

In this context, two scenarios were designed that can be well represented by the aforementioned conditions, such as a hospital room or a factory workstation. The results in each scenario are presented below.

3.4.1. Experiment on Scenario 1: Hospital Room

Studies indicate that approximately 40% of a nurse's time is spent on non-nursing tasks, such as delivering and retrieving meal trays or performing cleaning tasks [203]. Automating these tasks with AMRs frees up time for healthcare professionals to focus on critical patient care and reduces the strain of non-value-adding activities.

The implementation of AMRs streamlines routine tasks and reduces the physical demand on human workers, ensuring more consistent and efficient processes [204]. They are mainly used to transport meals between the kitchen and the ward, trash bins, linens, cleaning carts, sterile supplies, hospital medicines, laboratory samples, and other supplies.

Given the importance of a mobile robot in the healthcare sector, test scenario 1 for the comprehensive autonomous navigation strategy was determined to be a 10×10 meter space designed to simulate a hospital room as can be seen in Figure 3.54.

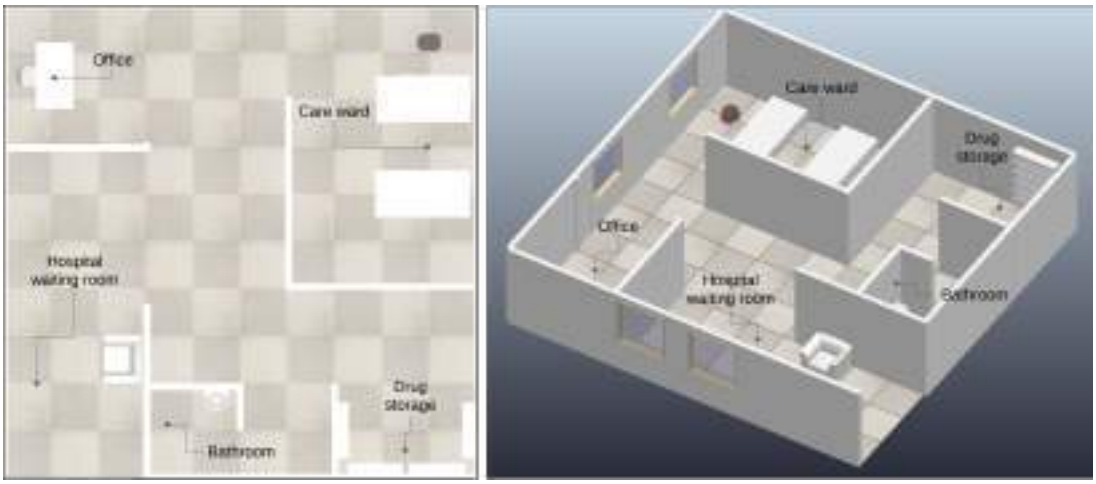
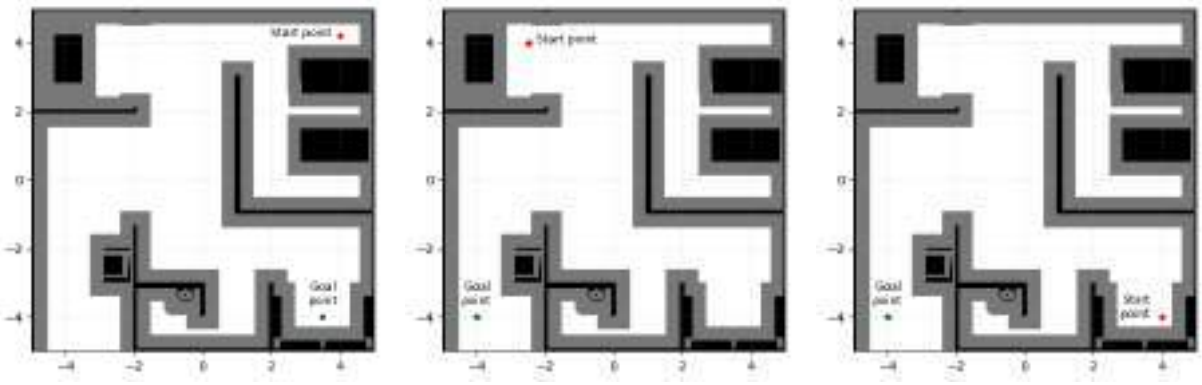


Figure 3.54: Scenario 1 designed to simulate a Hospital room

Within this scenario, three global routes will be created to represent repetitive tasks that the robot must routinely perform. These routes will be found using the global planner that utilizes the GWO in an offline mode. To find these routes, only static and immovable obstacles will be considered for designing the configuration space. Subsequently, for each identified route, the robot will follow them online and in real-time using the deliberative motion control for path following (FLC). However, there will be new unknown obstacles (both static and dynamic) that the robot must avoid using its local perception system and the OAC controller.

A. Offline mode: Global planning

After obtaining the configuration space, each of the three routes will be found using entre 100 y 500 iterations and 12 search agents or wolves as input parameters for the optimizer. The number of waypoints in the route will be determined empirically. The three starting and destination points within the scene can be seen in Figure 3.55.



(a) Route 1 configuration (b) Route 2 configuration (c) Route 3 configuration

Figure 3.55: Start and goal positions for the three test routes in Scenario 1

For each configuration, up to five attempts will be made to find the route using the described parameters (each attempt with 100, 200, 300, 400, and 500 iterations, respectively). The process will be terminated if a feasible path is not found after five attempts.

The process of obtaining the global path for routes 1, 2, and 3 can be seen in Tables 3.8, 3.9, and 3.10, respectively.

Table 3.8: Global Planning process for Route 1 in Scenario 1

Attempt	Iterations	Waypoints	Final distance $[m]$	Total time $[s]$
Attempt 1	100	3	16.05	52.02
Attempt 2	200	4	✗	145.24
Attempt 3	300	3	16.15	152.10
Attempt 4	**400**	**3**	15.52	202.38
Attempt 5	500	3	15.60	255.40

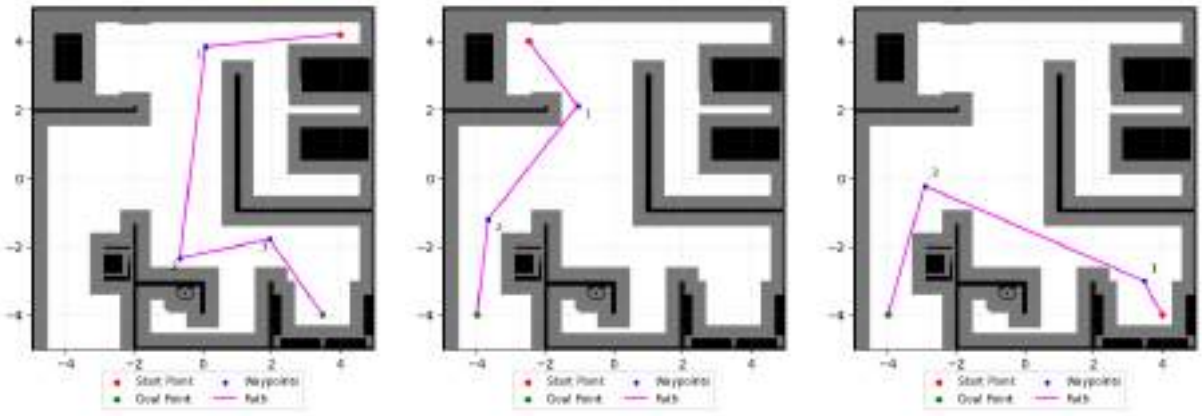
Table 3.9: Global Planning process for Route 2 in Scenario 1

Attempt	Iterations	Waypoints	Final distance $[m]$	Total time $[s]$
Attempt 1	100	3	10.47	80.85
Attempt 2	200	3	10.49	180.94
Attempt 3	300	2	9.45	182.91
Attempt 4	400	2	9.52	249.70
Attempt 5	**500**	**2**	9.42	317.80

Table 3.10: Global Planning process for Route 3 in Scenario 1

Attempt	Iterations	Waypoints	Final distance $[m]$	Total time $[s]$
Attempt 1	100	3	15.16	73.53
Attempt 2	**200**	**2**	11.91	95.69
Attempt 3	300	2	12.01	153.30
Attempt 4	400	2	12.15	196.15
Attempt 5	500	2	11.97	241.12

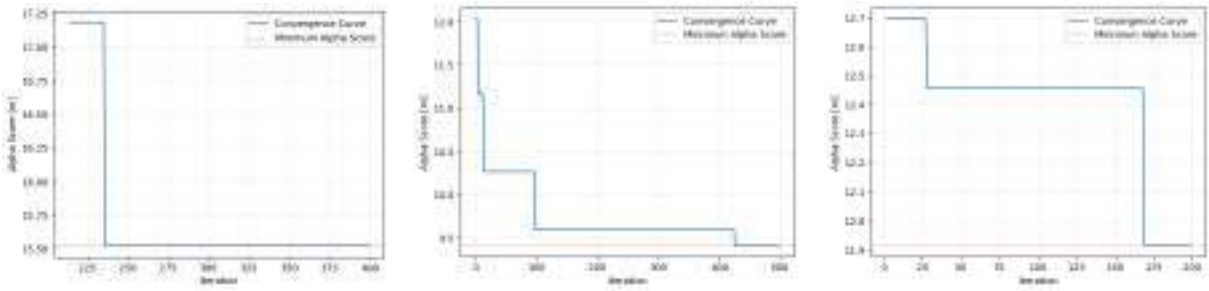
In this scenario, with a 50% occupancy of static obstacles including their safety margin, the global paths for the three routes were correctly found within the five attempts. Only one attempt failed to plan the path for route 1 (Table 3.8). Figure 3.56 shows the resulting paths for the three proposed configurations.



(a) Route 1 - 400 iterations (b) Route 2 - 500 iterations (c) Route 3 - 200 iterations

Figure 3.56: Global paths found using 12 search agents for the three test routes in Scenario 1

As can be seen, the number of iterations is not the determining factor in finding the route; rather, the high level of exploration allows for the result to be achieved. In Figure 3.57, which illustrates the convergence curve and optimization process for each route, it is apparent how the stochastic and random factor enables finding the best alpha (α) solution in the early iterations and maintaining it without further optimization.



(a) Route 1 - 400 iterations (b) Route 2 - 500 iterations (c) Route 3 - 200 iterations

Figure 3.57: Convergence curve of the best solution (α) for the three test routes in Scenario 1

B. Online mode: Local planning and motion control

Once the robot starts moving along the respective route, 3 cylindrical obstacles not identified in the global planning will be added, which the robot must avoid, as illustrated in Figure 3.58.

Figure 3.58: Scenario 1 with unknown obstacles added

These three new obstacles (two cylinders $0.5m$ height and $0.5m$ diameter, and one cylinder $1m$ high and $1m$ diameter) are not entirely static; they are manually moved to modify the robot's original path. Table 3.11 shows the collected data from the three followed routes, while Figure 3.59 illustrates the actual path executed by the robot.

Table 3.11: Experiment on Navigation in Scenario 1 Results

Route	Original distance $[m]$	Distance traveled $[m]$	Total simualtion time $[s]$
1	15.52	17.27	334.84
2	9.42	14.12	240.08
3	12.07	14.22	224.38

(a) Route 1 (b) Route 2 (c) Route 3

Figure 3.59: Resultant Paths in the presence of dynamic obstacles in Scenario 1

The first route in this scenario connects the care ward with the drug storage, a path that represents a repetitive task within the hospital room. This is the longest path, both expected and actual, of the three routes designed for this scenario, and it is composed of three waypoints in addition to the start and end positions. As can be seen in Figure 3.60, the obstacles obstructed the robot's path on several occasions. Additionally, the activation periods of the Obstacle Avoidance Controller can be observed in Figure 3.61 for angular and lineal velocities and Optical Flow estimations. It is important to remember that in all cases, intervals where the robot automatically orients itself to the next local goal are not plotted.

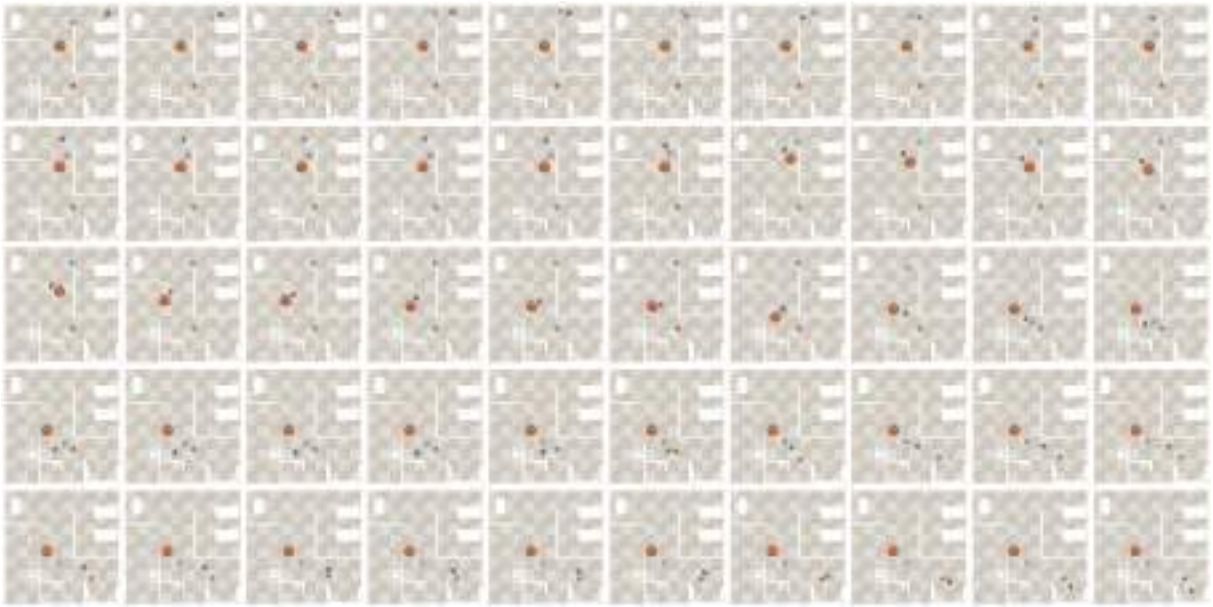
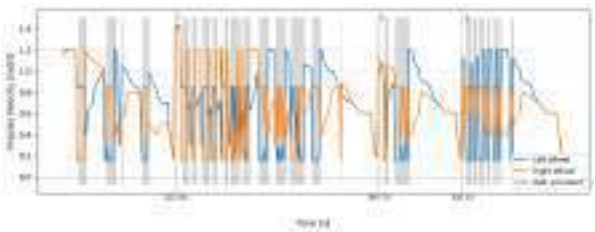
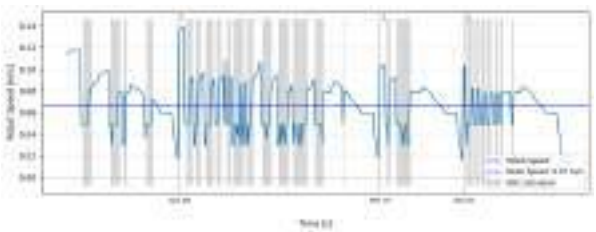


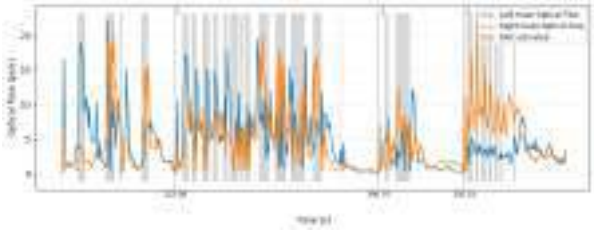
Figure 3.60: Frames of the path followed by the robot in Route 1 in Scenario 1



(a) Wheels' Angular Velocity



(b) Robot Speed

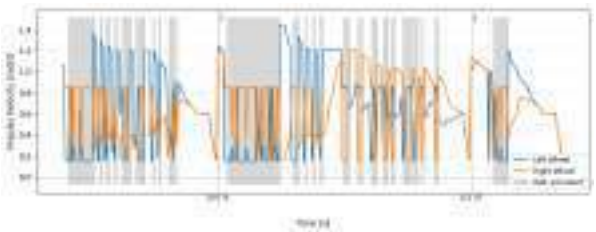


(c) Optical Flow Magnitude

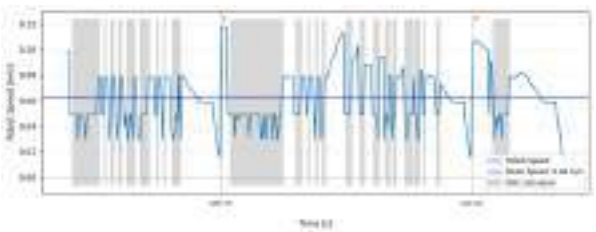
Figure 3.61: Velocities and Optical Flow estimations from Route 1 in Scenario 1

The second route represents the trip between the doctor's office and the waiting room, which also serves as a connection point with the rest of the hospital, exemplifying the task of delivering messages outside the care area. This is the shortest route of the three designed in this scenario, but the robot traveled proportionally more than on the planned path (almost 50% more). In Figure 3.62, it can be seen how the obstacles diverted the robot as much as possible from the next waypoint, but it always returned to the original path. Due to this high period during which the OAC was active, the robot's average linear speed was the lowest at 0.06 m/s, as shown in Figure 3.63.

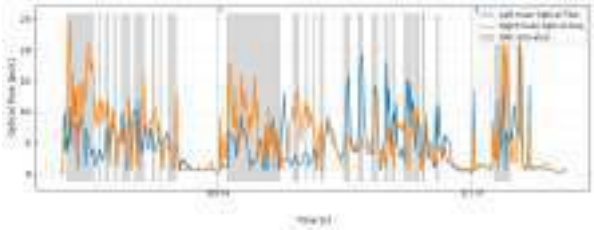
Figure 3.62: Frames of the path followed by the robot in Route 2 in Scenario 1



(a) Wheels' Angular Velocity



(b) Robot Speed



(c) Optical Flow Magnitude

Figure 3.63: Velocities and Optical Flow estimations from Route 2 in Scenario 1

The third and final route connects the drug storage with the waiting room, and therefore with the rest of the hospital, through two waypoints. This route is intended to exemplify the path the robot must follow to fetch more medication. Figure 3.64 shows the frames of the entire navigation, in which, as in the previous routes, obstacles were moved to modify the robot's path. Figure 3.65 displays the speed profiles and optical flow values obtained throughout the entire navigation in route 3. In none of the three routes did the robot collide, which was expected since the idea is that no obstacle is intentionally moved to collide with the robot.

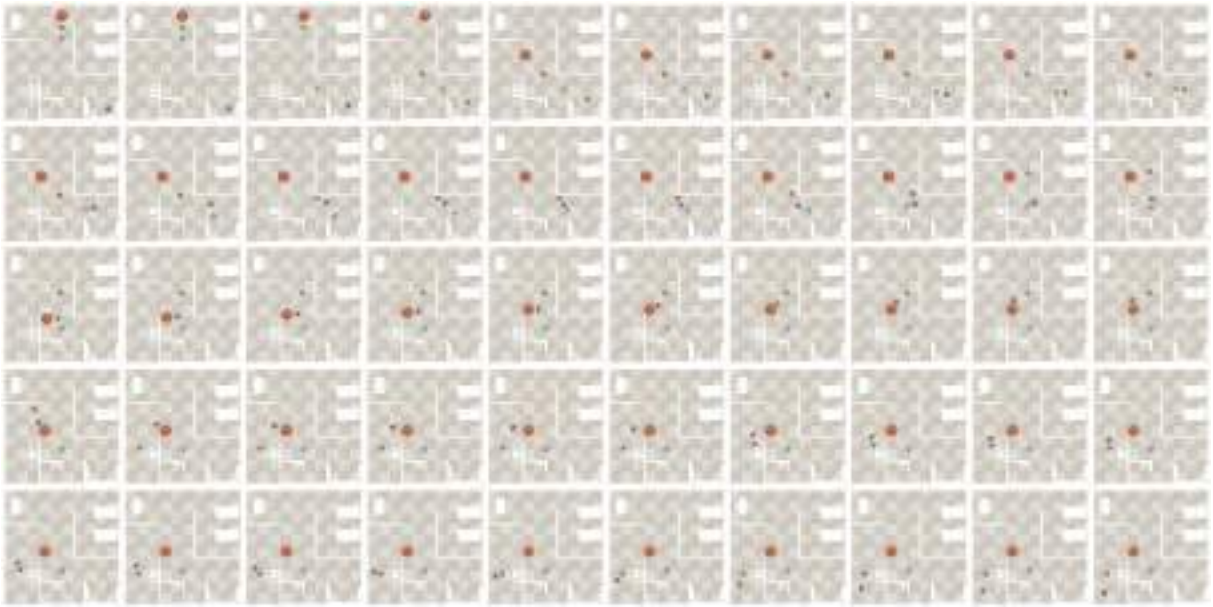
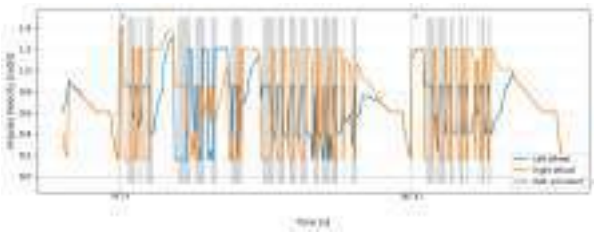
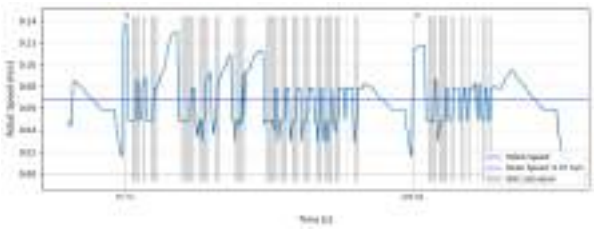


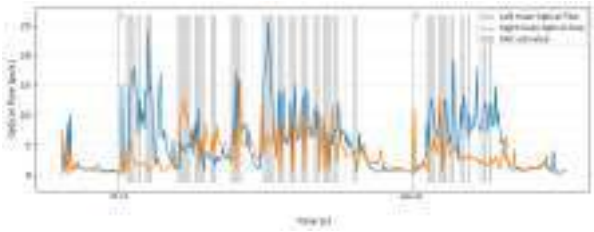
Figure 3.64: Frames of the path followed by the robot in Route 3 in Scenario 1



(a) Wheels' Angular Velocity



(b) Robot Speed



(c) Optical Flow Magnitude

Figure 3.65: Velocities and Optical Flow estimations from Route 3 in Scenario 1

3.4.2. Experiment on Scenario 2: Factory Workstation

In an increasingly competitive market, factories are turning to AMR to optimize operations and improve productivity. These robots assist human workers by excelling in repetitive, time-consuming, or physically demanding tasks. They handle material transport, inventory management, and other repetitive tasks, reducing employee workload and allowing them to focus on more complex functions [205].

Based on this, just as the hospital room was designed, a workstation in a factory with a 10×10 meter space was created as Testing Scenario 2 as can be observed in Figure 3.66.

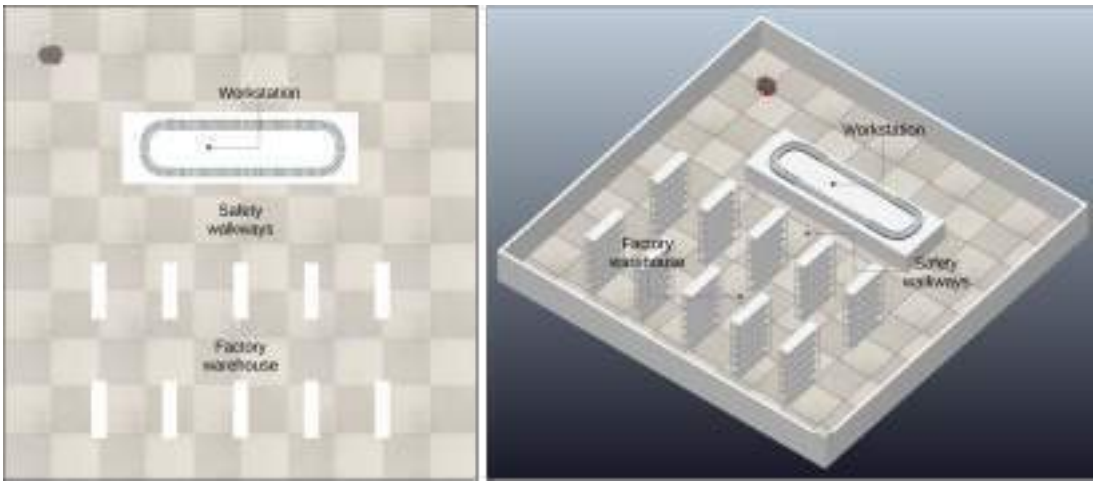
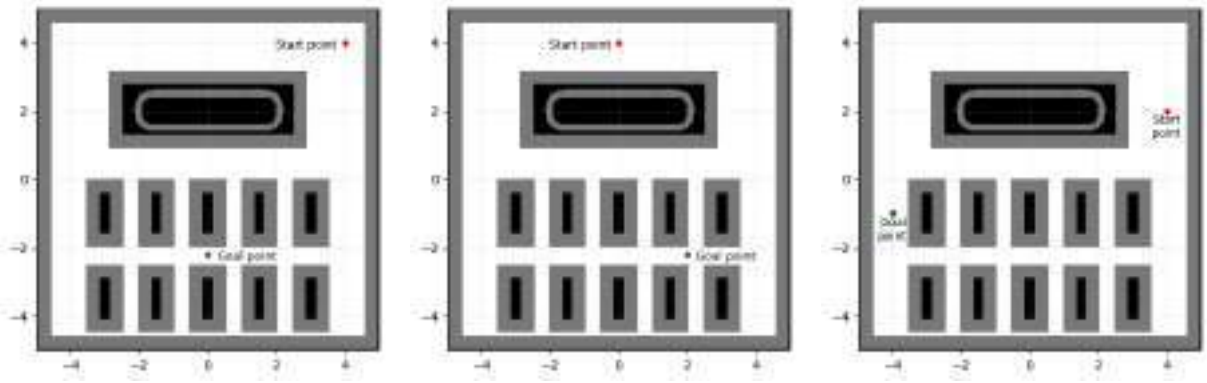


Figure 3.66: Scenario 2 designed to simulate a Factory workstation

As in the previous scenario, the objective is to find three representative and different paths within the scenario using the global planner. Subsequently, each path will be followed using Fuzzy Logic Control (FLC), and through the robot's local perception, unknown obstacles will be avoided using the Obstacle Avoidance Controller (OAC). Although this may seem like a simpler scenario, the arrangement of the shelves can present a challenge for the global planner, as more than half of the scene is occupied by narrow spaces.

A. Offline mode: Global planning

The path-finding procedure will be the same as described in Scenario 1. Using between 100 and 500 iterations and 12 wolves (search agents), there will be a maximum of 5 attempts to find each route. If no route is found after 5 attempts, then the path will be created manually. Figure 3.67 shows the three start and goal configurations designed for this scenario.



(a) Route 1 configuration (b) Route 2 configuration (c) Route 3 configuration

Figure 3.67: Start and goal positions for the three test routes in Scenario 2

The space occupied by the obstacles in this scenario, including the safety margin, is 51%. It is important to highlight once again that the number of waypoints is determined empirically for each route, which represents an advantage or disadvantage in the path design. The process of obtaining the global path for routes 1, 2, and 3 in Scenario 2 can be seen in Tables 3.12, 3.13, and 3.14, respectively.

Table 3.12: Global Planning process for Route 1 in Scenario 2

Path	Attempt	Iterations	Waypoints	Final distance $[m]$	Total time $[s]$
	Attempt 1	100	2	14.59	44.48
	Attempt 2	200	1	✗	98.77
	Attempt 3	300	1	✗	84.12
	Attempt 4	400	2	✗	167.62
	Attempt 5	**500**	**2**	9.27	207.42

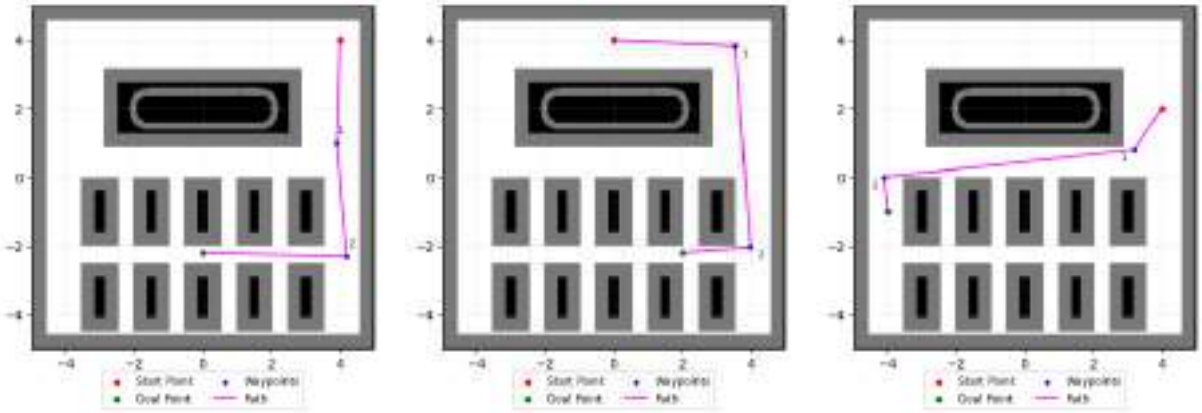
Table 3.13: Global Planning process for Route 2 in Scenario 2

Path	Attempt	Iterations	Waypoints	Final distance $[m]$	Total time $[s]$
	Attempt 1	100	2	13.18	34.60
	Attempt 2	200	2	11.73	70.04
	Attempt 3	300	2	11.57	106.46
	Attempt 4	400	2	✗	146.07
	Attempt 5	**500**	**2**	11.42	167.95

Table 3.14: Global Planning process for Route 3 in Scenario 2

Path	Attempt	Iterations	Waypoints	Final distance $[m]$	Total time $[s]$
	Attempt 1	100	2	9.94	37.28
	Attempt 2	200	2	12.45	78.80
	Attempt 3	300	2	9.81	113.41
	Attempt 4	**400**	**2**	9.78	139.61
	Attempt 5	500	2	9.79	188.92

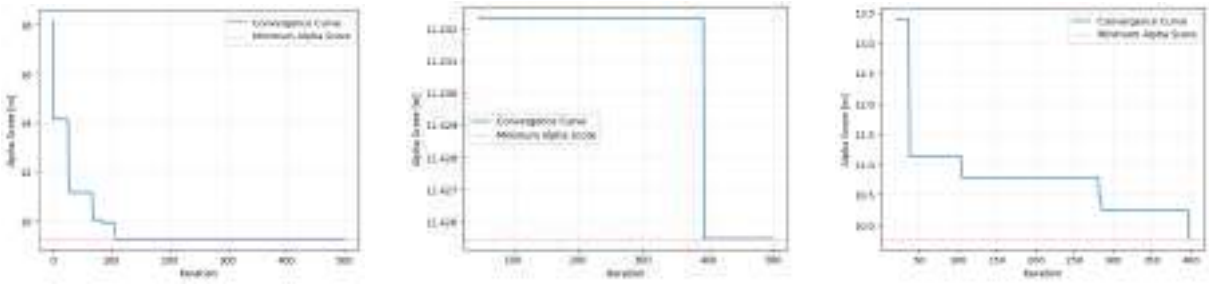
The global paths were correctly found for all three routes. However, it can be noted, particularly in routes 1 (Table 3.12) and 2 (Table 3.13), that these types of scenarios can be challenging for the planner due to incorrect estimation of waypoints or limited free space to create the routes. Figure 3.68 shows the resulting paths for the three proposed route configurations.



(a) Route 1 - 500 iterations (b) Route 2 - 500 iterations (c) Route 3 - 400 iterations

Figure 3.68: Global paths found using 12 search agents for the three test routes in Scenario 2

In this case, the higher number of iterations did affect finding the desired route, especially in the cases of routes 2 and 3, as can be observed in Figure 3.69.



(a) Route 1 - 500 iterations (b) Route 2 - 500 iterations (c) Route 3 - 400 iterations

Figure 3.69: Convergence curve of the best solution (α) for the three test routes in Scenario 2

B. Online mode: Local planning and motion control

Once the robot starts moving along the respective route, three cuboid obstacles not identified in the global planning will be added, which the robot must avoid, as illustrated in Figure 3.70.

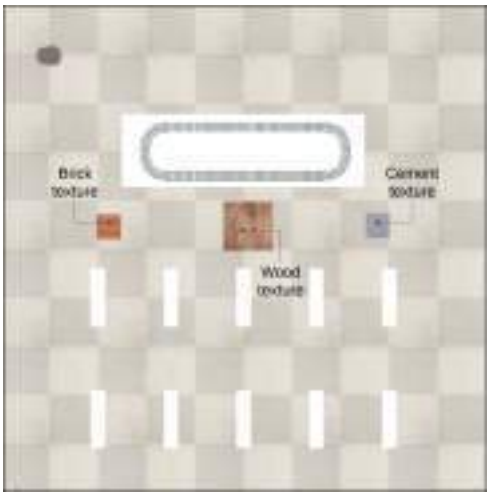


Figure 3.70: Scenario 2 with unknown obstacles added

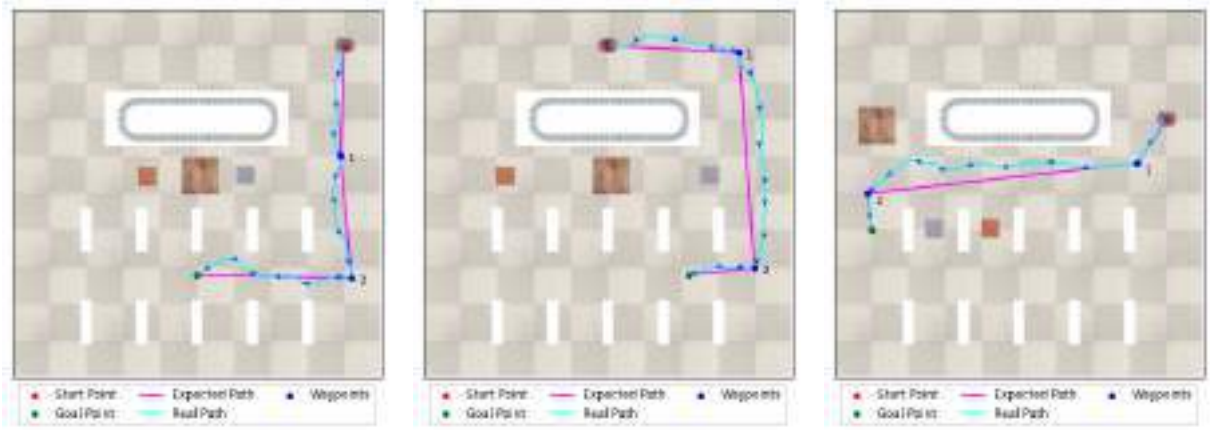
As in Scenario 1, these three new obstacles (two cubes $0.5m$ per side, and one cube $1m$ per side) are not entirely static; they will be manually moved to modify the robot's original path. Table 3.15 shows the summary data from the three followed routes.

Table 3.15: Experiment on Navigation in Scenario 2 Results

Route	Original distance $[m]$	Distance traveled $[m]$	Total simualtion time $[s]$
1	10.52	11.31	210.67
2	11.23	11.78	187.83
3	9.78	10.31	157.02

In this case, the routes are shorter, and the actual distance traveled is not more than 1 meter greater than the expected distance. As illustrated in Figure 3.71, the robot practically does not deviate from the original route in any of the three cases. This is because there was not much space in this scenario for the robot to encounter dynamic obstacles.

Normally, a robot within a factory is responsible for transporting objects, so the three presented routes aim to exemplify tasks in which the robot has to go from the workstation to fetch something from the factory warehouse, starting from different locations.



(a) Route 1 (b) Route 2 (c) Route 3

Figure 3.71: Resultant Paths in the presence of unknown dynamic obstacles in Scenario 2

The first route, despite not being the longest, is the one that took the most time to complete, partly due to constant obstacle avoidance, as seen in the frames representing the simulation (Figure 3.72). Despite moving through a narrow space for the entire route, it can be noted that it still had good reactive capability when necessary. On the other hand, Figure 3.73 shows the estimated speed and optical flow values throughout the entire route, where it can be observed that the average speed was relatively low, specifically around 0.06 m/s. Additionally, there were several zones along the path where the obstacle avoidance controller was actively engaged. This indicates that the system had to frequently adjust its trajectory to navigate around obstacles, which contributed to the overall reduced speed.

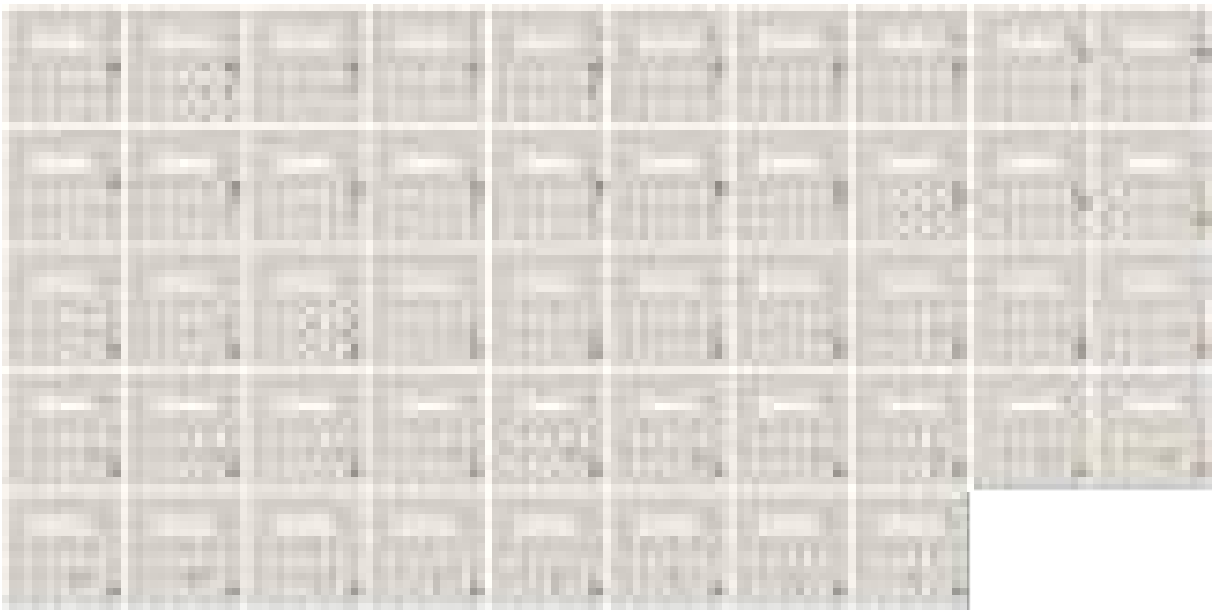
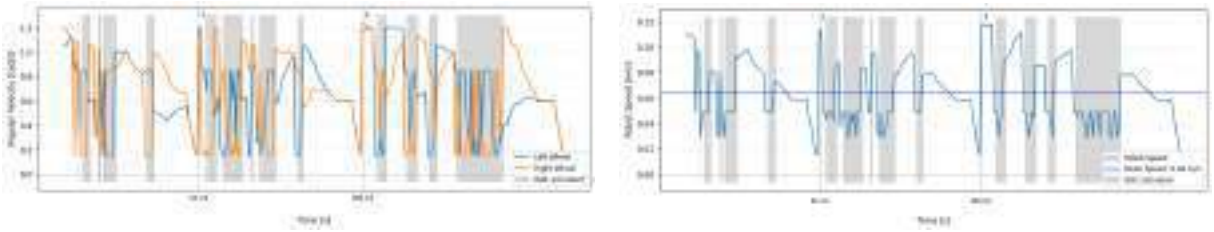
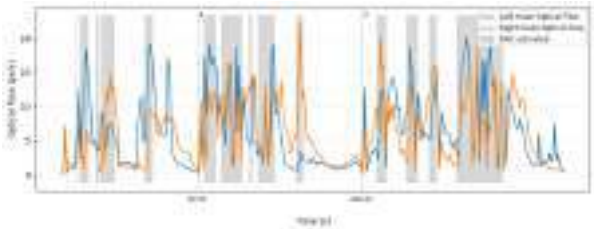


Figure 3.72: Frames of the path followed by the robot in Route 1 in Scenario 2



(a) Wheels' Angular Velocity

(b) Robot Speed



(c) Optical Flow Magnitude

Figure 3.73: Velocities and Optical Flow estimations from Route 1 in Scenario 2

Conversely, route 2 covered the longest distance but deviated the least from the planned route (only 4%). As mentioned, this is because it was not possible to divert the robot since there was no space for it to pass. If the dynamic objects were not actively and manually controlled during the simulation, the robot would likely be unable to avoid them. Figure 3.74 shows moments where the dynamic obstacle stands in front of the robot to block its path but must be immediately removed to allow it to pass. In the velocity profiles and optical flow estimations shown in Figure 3.75, it can be identified that there are indeed long periods where the robot does not activate the OAC.

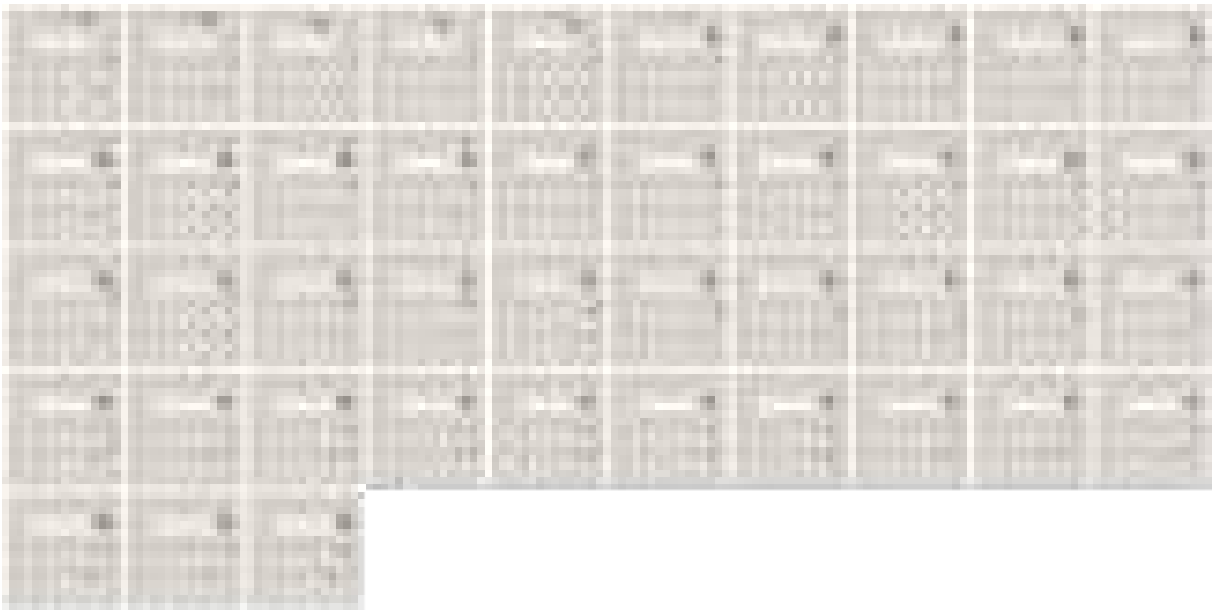
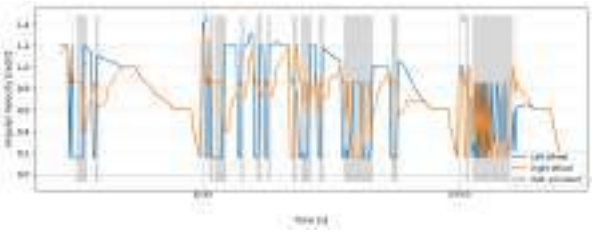
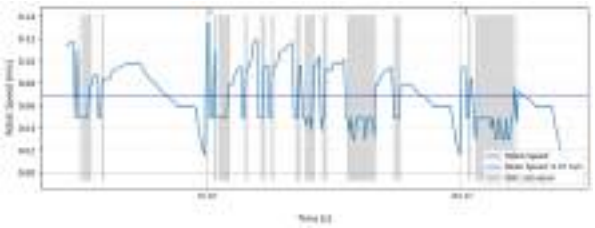


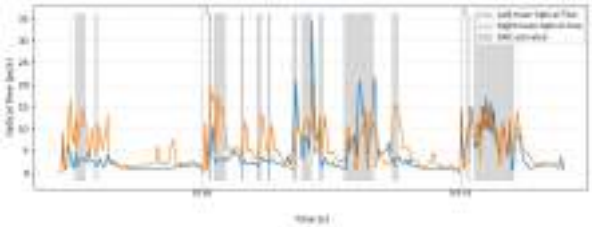
Figure 3.74: Frames of the path followed by the robot in Route 2 in Scenario 2



(a) Wheels' Angular Velocity



(b) Robot Speed



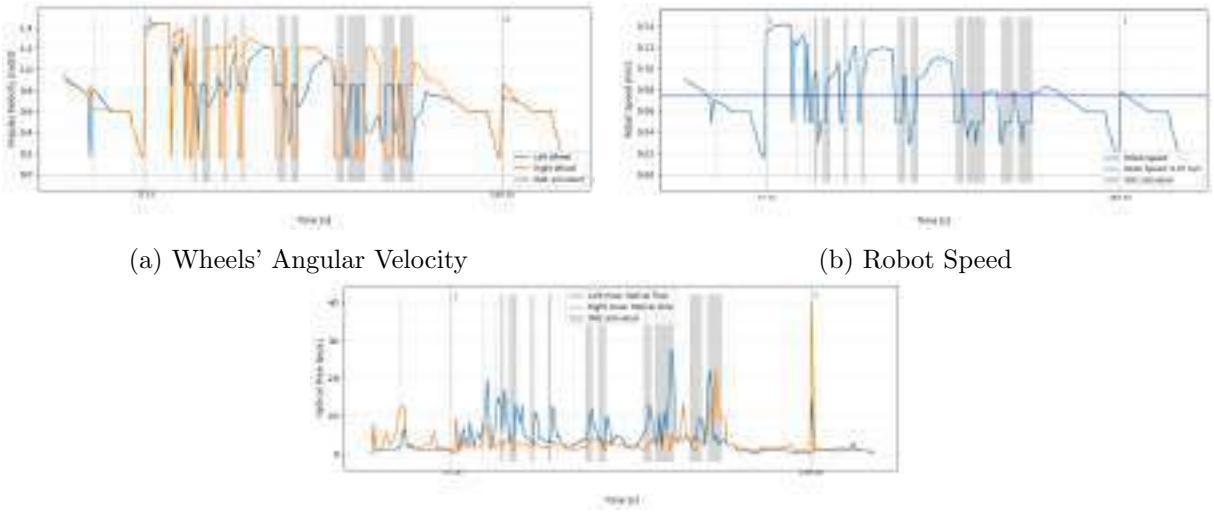
(c) Optical Flow Magnitude

Figure 3.75: Velocities and Optical Flow estimations from Route 2 in Scenario 2

Finally, route 3 was developed under the same conditions as the previous two routes: a limited space where the robot did not have enough room to evade an obstacle. Figure 3.76 illustrates the complete simulation and shows how the obstacles tried to keep it away from its original route, despite being limited by the workstation table. Nevertheless, the robot did not collide and was able to complete its route.

In Figure 3.77, it can be seen that there were very few periods of OAC activation, which even helped the robot achieve an average linear velocity of almost 0.08 m/s, which is relatively high.

Figure 3.76: Frames of the path followed by the robot in Route 3 in Scenario 2



(a) Wheels' Angular Velocity

(b) Robot Speed

(c) Optical Flow Magnitude

Figure 3.77: Velocities and Optical Flow estimations from Route 3 in Scenario 2

This not only concludes the tests in Scenario 2 but also completes the evaluation and validation tests to verify the entire navigation strategy. The general analysis of these tests is presented in the following section.

3.5. General Analysis

In this chapter, the performance of each element comprising the *Autonomous Navigation Strategy* under different conditions was reviewed. Each section provides a detailed analysis of the results, so this final section aims to offer a more general perspective, highlighting the scope and limitations associated with the proposed navigation strategy compared to the hypothesis that guided the design and implementation of this research work (see Section 2).

A. Global Path Planning Analysis

Firstly, the experimentation conducted for *Global Path Planning* (Section 3.1) aims to demonstrate the capability to find the global path efficiently and safely. In this regard, the planner showed the possibility of finding routes in spaces with up to 80% of obstacles in the scene and different scene sizes. Similar to the path planning strategies studied in Chapter 1, the path search was implemented in the configuration space rather than the workspace, facilitating the optimization of the route from a single point instead of considering the entire robot. However, given the comprehensive approach of a complete navigation strategy, the planning itself includes a safety margin for static obstacles to prevent the robot from colliding due to paths being too close to obstacles during movement.

Another important characteristic to highlight is the use of a relatively new bio-inspired algorithm that aligns with current related work trends. The metaheuristic and stochastic approach of the *Grey Wolf Optimizer* allows for high exploration within the available space, finding several paths due to the high commitment it provides in the exploitation of the optimal solution search. Although in most tests it cannot be determined that the shortest possible solution was indeed found, the paths identified tend to be close to an optimal solution.

Moreover, the use of waypoints to define the global route allows for the identification of secure positions that the robot must reach regardless of its new position, which represents the advantage of returning to the original path once a local obstacle forces the robot to deviate. This could also be considered a disadvantage due to the mandatory need to cross each waypoint, but it prioritizes the safety of the robot.

Just as the planner can demonstrate good performance, its design also presents certain limitations. For instance, in several scenarios, it is difficult to initialize a feasible solution due to the high presence of obstacles, as it must be ensured that the proposed positions do not fall on an obstacle and that consecutive waypoints do not cross over an obstacle. This reduces the possibility of planning complex routes with sharp changes in orientation, as low performance was identified in paths requiring more than 5 waypoints. This is because it represents a higher likelihood of some path segment crossing an obstacle and none of the proposed routes being valid.

This same inability to create feasible solutions limits the optimizing power of the position update equations of the GWO, as each time a route is found and updated, it is highly likely that the new solution will not be feasible, breaking the optimization thread of the route. It was identified that the high number of iterations has little effect, as the total number does not matter if there are no solutions to optimize. This necessitates increasing the number of wolves or search agents to leverage the randomness and high exploration of the algorithm to find a route, but this results in longer execution times.

Finally, time could be another limitation. Traditionally, global planning has been an offline task, but its online implementations are becoming increasingly common. This planner does not have the performance to do it in this manner, especially considering the ideal requirement of easily obtaining the global map, which could be a conflict in a real implementation.

B. Motion Control for Path Following Analysis

The experimentation for *Global Path Following Control* (Section 3.2) aims to demonstrate the capability to follow the path, particularly using the *Fuzzy Logic Controller*. One of the greatest advantages of this navigation strategy is precisely the inclusion of the movement of a real robot along the route, a capability often ignored in related works. In this strategy, the kinematics and size of a differential robot, the Pioneer 3-DX, are considered for the controller design.

Generally, the robot adequately follows the global path with minimal deviations and without collisions in completely static and globally known environments. Besides the FLC, the inclusion of automatic orientation to the next waypoint is a significant advantage, as it limits the robot to simply following the global route in static environments without deviating and risking collision.

Although it is an advantage in terms of safety and accuracy, the aforementioned automatic orientations towards the next waypoint are performed in a process external to the controller and take too much time. While this prevents collisions with sharp turns, the fuzzy logic controller could be improved to include these turns in their design.

Another limitation that becomes apparent during all executions is that the FLC was designed with a very slow speed limit (0.16 m/s), occasionally reaching averages of only 0.06 m/s. The controller was intentionally designed to move slowly to ensure correct path following and high reactive capability, but while low speeds are functional, they might not be desirable.

Finally, one of the main disadvantages is the reliance on position and orientation measurements directly from the CoppeliaSim simulation. The FLC controller actively uses both the orientation and position of the robot as well as the position of all waypoints, which could present a challenge in real implementation, requiring more complex instrumentation and equipment to obtain the same information with the same reliability as in the simulator.

C. Local Path Planning Analysis

The reactive-based *Local Path Planning* technique to avoid unknown obstacles aims to be automatic and safe, capabilities that were tested during the experimentation in Section 3.3. Particularly, the *Obstacle Avoidance Controller*, an Optical-Flow-based Sense-to-Avoid System, was tested. The first highlight is the high reactive capability in the presence of both static and dynamic obstacles.

Due to the integration of highly reliable distance sensors and the vision sensor in its perception system, the robot can extract most of the information from its local environment. Particularly, the integration of optical flow measurements with fuzzy logic is a success, as it allows the robot to be actively controlled using information from a single image, which in this case has a low resolution of 256×256, facilitating estimation without compromising information.

One of the great advantages of this sense-to-avoid system is the ability to automatically switch between the FLC and OAC controls as needed, allowing the robot to always return to the original route even if it deviates to avoid obstacles. Additionally, the obstacle avoidance control is composed of only 9 rules, which is relatively low given the high number of factors that could be considered. This allows for obstacle avoidance without much complexity.

However, one of the limitations comes precisely from the low number of rules, as sometimes a lack of robustness can be noticed in the face of the diverse array of possible scenarios and obstacles. As mentioned throughout this document, under these conditions, it would be impossible to design a system capable of handling every situation, making the solution as specific as the implementation. Even so, the lack of redundancy and collision safety systems is noticeable, limited by the way this controller was designed, as the robot is always moving and is forced to make a decision even when there is no truly viable option.

D. Complete Navigation Strategy Analysis

To demonstrate the process in which the proposed *Autonomous Navigation Strategy* could be implemented, two scenarios were designed in the experimentation of Section 3.4. It is worth noting that the robot faced controlled conditions that were not as demanding as the previous experiments.

In these scenarios, the robot was able to navigate without major complications in an environment containing known static obstacles and unknown static and dynamic obstacles in real-time. Under these conditions, the correct interaction and integration of deliberative and reactive motion controls were demonstrated, following a global route and deviating whenever necessary.

However, this strategy works only under particular circumstances, where it depends on the environment "knowing" that there is a robot within the scene. In other words, given how the controller was designed, the robot does not stop at any moment, particularly with the idea

of always having values in the optical flow estimation. The dynamic elements in the environment during these tests were moved manually, introducing a certain bias in the robot's reaction. Therefore, no object in the scenario has the intention of causing collisions or diverting the robot sufficiently to prevent it from returning to its route.

Finally, in this test, the hypothesis posed at the beginning of this research work can be validated:

By integrating a global path planning strategy with a local path planning strategy as inputs for the motion control of the robot, the AMR can complete efficiently, safely, and accurately a route from a start position to a goal position in an environment with static and dynamic obstacles.

Therefore, in the proposed navigation strategy, regardless of the conditions of the scenario in which it was tested, it was demonstrated that global path planning contributes to finding an efficient and safe reference route between a start position and a goal position in an environment with static obstacles. On the other hand, local path planning allows for safe collision avoidance in the presence of dynamic obstacles within the same environment. Finally, due to this planning, motion control can accurately follow the global route to the goal position, deviate if necessary, and return when possible, thus completing the navigation of the AMR.

Chapter 4

Conclusions & Future Work

For nearly 160 pages, implementing an autonomous navigation strategy for mobile robots has been presented, studied, and developed. Given the length of this dissertation, it is time to revisit the foundational goals that supported the research by retrospectively looking at the objectives presented.

Conclusions

The general objective was proposed to develop and implement a comprehensive autonomous navigation strategy so that a WMR can move in static and dynamic environments automatically, efficiently, safely, and accurately. The results presented throughout Chapter 3, particularly the tests in Section 3.4, demonstrate the correct alignment between the development and the objective set. A global path planning strategy was implemented through various methods to find the shortest feasible route between a starting position and a target point in an environment with static obstacles. Additionally, a sensor-based reactive local path planning strategy was developed to safely avoid unknown dynamic obstacles while the WMR moves along the global path. Finally, the motion planning strategy was performed using the insights from both global and local path planning, and its functionality was tested in a partially dynamic virtual environment.

As has been continuously mentioned, this strategy does not solve the problem of mobile robotics, nor does it intend to do so. These are merely techniques and methods that, reviewed in Chapter 1, were integrated to provide an alternative solution to the high demand for RaaS and AMRs. Given the many different implementations, this strategy, as proposed, can be effective for certain uses. The objective was to develop and implement the navigation strategy, which was achieved.

It is important to mention that the design process of the strategy described in Chapter 2 was not completely linear. Although only the final result is shown, it was an iterative process involving the configuration and integration of all elements. As highlighted earlier, the solution is as specific as the implementation and guided by the objective, each element was adjusted while associated requirements for global path planning, path following control, and local path planning were added, respectively, and in that order.

One of the most notable features of this strategy is that it successfully integrates two of the main components of path planning, often ignored: global and local planning. It does so by utilizing three of the methods with the most projection in industry and research: 1) a bio-inspired, stochastic, and metaheuristic optimization algorithm, namely the GWO; 2) a vision-based system using optical flow as the information acquisition method; and 3) fuzzy logic control capable of actively converting environmental measurements into instructions for the robot.

Additionally, the proposed navigation strategy considers the kinematic constraints of a differential robot, which are used in the design of motion control. This addresses one of the problems of the path planning solutions reviewed in the literature: the lack of real-time implementation. By using a highly reliable simulation such as CoppeliaSim, it is possible to measure the robot's performance while moving in dynamic scenarios, where the deliberative and reactive capabilities of the controller are tested.

Future Work

Being a traditional navigation strategy, it has a modular architecture, different from those using deep learning such as end-to-end navigation. Therefore, each component can be improved, modified, and replaced separately: using a different strategy for global planning, another control system for path following, a different mechanism to replace the sense-to-avoid system during local planning, and even the choice of another mobile robot. In this sense, it is proposed that for anyone considering using this research work as a starting point or reference, future work should be guided by three alternatives:

1. Improve the performance of the strategy based on the identified limitations.
2. Implement in a real physical environment and conduct objective evaluation.
3. Explore new alternatives for any of the components.

The first point refers to making modifications to the strategy using the proposed methods and interactions to provide more robust performance in terms of safety, efficiency, and accuracy. Based on the analyzed results, within the global path planning strategy, it is identified that the process of initializing possible solutions in the GWO should be improved, as a completely random method is not feasible in scenarios with a high presence of obstacles. It could even be considered to use network graph space representation, such as the Probabilistic Roadmap, exact cell decomposition, or rapidly-exploring random tree, and start optimization from these representations.

Another suggestion within global planning is to modify the definition of the fitness equation to include the presence of obstacles in some way, so that the update of positions implicitly considers this and constantly generates valid solutions, instead of having to generate new random positions that lose the sequence of optimization.

In the case of motion control for path following, it is proposed to modify the fuzzy logic rules to allow executing paths at a higher speed. Additionally, intending to reduce complexity, it is ideal that the orientation to the next waypoint of the route is part of the same controller. On the other hand, the development work should also focus on balancing the speeds of the FLC and OAC, so that during the robot's execution, the speed changes are not so high. Finally, either by adding more rules to the controller or modifying its definition, it should be sought that the robot does not move permanently and has pause periods during its movement, as it was shown that the reactive capability works, but sometimes the decision made may be hurried.

Regarding local planning, it is considered that future work should focus on strengthening the use of only the vision sensor, leveraging all the information that can be extracted from an image, and reducing reliance on ultrasonic sensors, which, although reliable, add complexity to the system. Particularly, the use of OF can be better utilized, as currently it only considers two areas of the image and calculates the average of the optical flow magnitude values, which results in a loss of information. Therefore, incorporating more areas and better parameterization could enhance the robustness of the optical-flow-based system.

In another sense, the proposed strategy prioritized functionality, meaning that the robot was effectively able to evade obstacles. However, it is necessary to consider the energy efficiency of this system in future work, especially when thinking about implementation in a real environment.

Precisely, the second proposed alternative to continue this work is the implementation in a real physical environment. As mentioned, using the simulator presupposes several parameters used in the strategy (such as the exact position and orientation of the robot at all times). Therefore, a real implementation would involve a process of adaptation and instrumentation so that it can function similarly to how it does in CoppeliaSim. In this sense, one of the main points during the evaluation of the strategy is that its performance is measured based on its capabilities and not in comparison with other strategies. Thus, the next step should also be the comparison with other proposals to objectively determine its capability, preferably in real environments.

Finally, the third alternative considered for future work is to seek other options for each of the modules that compose the strategy. Just as the methods used were chosen, it is possible to make another selection and integration of systems, such as using another optimizer instead of the GWO, using a completely different approach in global planning, using a different controller from fuzzy logic, using other sensors, or even implementing the strategy on another robot, taking advantage of the mentioned modular architecture with which it was designed.

Where machines come alive

This thesis focused precisely on mobile robots and their navigation capabilities, highlighting the importance of path and motion planning in both static and dynamic scenarios. Based on the need for modern navigation strategies due to the rapid expansion of the development and deployment of mobile robots in various sectors, a study was conducted on the techniques and methods used, as well as a literature review of current implementations, to select elements that, when integrated, could function adequately as a comprehensive navigation strategy. Subsequently, the chosen methods were implemented, and the necessary interfaces between components were created so that finally a differential robot could move efficiently, safely, and accurately.

This research work concludes and hopes to serve as a guide in the extensive and important field of autonomous mobile robotics, offering not just a path forward, but an inspiration for future innovations. This dissertation is a testament to human ingenuity and the relentless pursuit of progress, aiming to contribute, even if just a little, to an era where machines not only navigate our world but also transform it.

The realm where machines come alive and redefine the boundaries of possibility.

References

[1] Chong Sun et al. "Software development for autonomous and social robotics systems". In: *Intelligent Interactive Multimedia Systems and Services: Proceedings of 2018 Conference 11*. Springer. 2019, pp. 151–160.

[2] Anthony Brohan et al. "Rt-1: Robotics transformer for real-world control at scale". In: *arXiv preprint arXiv:2212.06817* (2022).

[3] Tlijani Hayet, Tlijani Hatem, and Knani Jilani. "Navigation Control of a Mobile Robot under Time Constraint using Genetic Algorithms, CSP Techniques, and Fuzzy Logic". In: *Nature-Inspired Computing: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2017, pp. 932–954.

[4] Nur Syazreen Ahmad, Ng Lai Boon, and Patrick Goh. "Multi-sensor obstacle detection system via model-based state-feedback control in smart cane design for the visually challenged". In: *IEEE Access* 6 (2018), pp. 64182–64192.

[5] Radim Hercik et al. "Implementation of autonomous mobile robot in smartfactory". In: *Applied Sciences* 12.17 (2022), p. 8912.

[6] Giuseppe Fragapane et al. "Autonomous mobile robots in hospital logistics". In: *IFIP International Conference on Advances in Production Management Systems*. Springer. 2020, pp. 672–679.

[7] Randi Williams et al. "Doodlebot: An Educational Robot for Creativity and AI Literacy". In: *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*. 2024, pp. 772–780.

[8] Stanislav Ivanov, Craig Webster, and Katerina Berezina. "Robotics in tourism and hospitality". In: *Handbook of e-Tourism* (2022), pp. 1873–1899.

[9] Nathan Melenbrink, Justin Werfel, and Achim Menges. "On-site autonomous construction robots: Towards unsupervised building". In: *Automation in construction* 119 (2020), p. 103312.

[10] Won S Kim et al. "Targeted driving using visual tracking on Mars: From research to flight". In: *Journal of Field Robotics* 26.3 (2009), pp. 243–263.

[11] Nicola Basilico. "Recent trends in robotic patrolling". In: *Current Robotics Reports* 3.2 (2022), pp. 65–76.

[12] Ibrahim Hasan and Tolgay Kara. "Design, Construction and Control of an Autonomous Mobile Rescue Robot with Visual Feedback". In: *Avrupa Bilim ve Teknoloji Dergisi* 37 (2022), pp. 65–71.

[13] Jae-Hong Shim and Young-Im Cho. "A mobile robot localization using external surveillance cameras at indoor". In: *Procedia Computer Science* 56 (2015), pp. 502–507.

[14] José Ricardo Sánchez-Ibáñez, Carlos J Pérez-del Pulgar, and Alfonso García-Cerezo. "Path planning for autonomous mobile robots: A review". In: *Sensors* 21.23 (2021), p. 7898.

[15] Christopher Müller et al. *World Robotics 2023 – Service Robots*. IFR Statistical Department, VDMA Services GmbH, 2023.

[16] R Abiyev, Dogan Ibrahim, and Besime Erin. "Navigation of mobile robots in the presence of obstacles". In: *Advances in engineering software* 41.10-11 (2010), pp. 1179–1186.

[17] D Regan, K Beverly, M Cynader, et al. "G. Westheimer and SP McKee,"Stereoscopic acuity for moving retinal images," J. Opt. Soc. Amer., vol. 68, pp. 450-455, Apr." In: *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION* 7.3 (1991).

[18] Anbalagan Loganathan and Nur Syazreen Ahmad. "A systematic review on recent advances in autonomous mobile robot navigation". In: *Engineering Science and Technology, an International Journal* 40 (2023), p. 101343.

[19] Guillaume Vailland, Valérie Gouranton, and Marie Babel. "Cubic bézier local path planner for non-holonomic feasible and comfortable path generation". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7894–7900.

[20] Anis Naema Atiyah Rafai, Noraziah Adzhar, Nor Izzati Jaini, et al. “A review on path planning and obstacle avoidance algorithms for autonomous mobile robots”. In: *Journal of Robotics* 2022 (2022).

[21] BK Patle et al. “A review: On path planning strategies for navigation of mobile robot”. In: *Defence Technology* 15.4 (2019), pp. 582–606.

[22] Luca Bruzzone, Shahab Edin Nodehi, and Pietro Fanghella. “Tracked locomotion systems for ground mobile robots: A review”. In: *Machines* 10.8 (2022), p. 648.

[23] Ameer Tamoor Khan, Shuai Li, and Xinwei Cao. “Control framework for cooperative robots in smart home using bio-inspired neural network”. In: *Measurement* 167 (2021), p. 108253.

[24] Gangqiang Chen. “Robotics applications at airports: Situation and tendencies”. In: *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*. IEEE. 2022, pp. 536–539.

[25] Benjamin Lewandowski et al. “Socially compliant human-robot interaction for autonomous scanning tasks in supermarket environments”. In: *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2020, pp. 363–370.

[26] Ruchi Goel and Pooja Gupta. “Robotics and industry 4.0”. In: *A Roadmap to Industry 4.0: Smart Production, Sharp Business and Sustainable Development* (2020), pp. 157–169.

[27] R Illah. *Autonomous Mobile Robots*. 2004.

[28] SAM Coenen et al. “Motion planning for mobile robots—A guide”. In: *Control Systems Technology* 79 (2012).

[29] Xuesu Xiao et al. “Motion planning and control for mobile robot navigation using machine learning: a survey”. In: *Autonomous Robots* 46.5 (2022), pp. 569–597.

[30] Anis Koubaa et al. “Introduction to mobile robot path planning”. In: *Robot path planning and cooperation: foundations, algorithms and experimentations* (2018), pp. 3–12.

[31] Mariana Rampinelli et al. “An intelligent space for mobile robot localization using a multi-camera system”. In: *Sensors* 14.8 (2014), pp. 15039–15064.

[32] Sofia Yousuf and Muhammad Bilal Kadri. “Information fusion of GPS, INS and odometer sensors for improving localization accuracy of mobile robots in indoor and outdoor applications”. In: *Robotica* 39.2 (2021), pp. 250–276.

[33] Tien Quang Tran, Andreas Becker, and Damian Grzechca. "Environment mapping using sensor fusion of 2D laser scanner and 3D ultrasonic sensor for a real mobile robot". In: *Sensors* 21.9 (2021), p. 3184.

[34] Dayi Zhang et al. "A framework of using customized LIDAR to localize robot for nuclear reactor inspections". In: *IEEE Sensors Journal* 22.6 (2021), pp. 5352–5359.

[35] David M Rosen et al. "Advances in inference and representation for simultaneous localization and mapping". In: *Annual Review of Control, Robotics, and Autonomous Systems* 4 (2021), pp. 215–242.

[36] Marcelina. "Motion planning vs path planning". In: *Medium* (2022). Accessed: 2024-02-17. URL: `https://medium.com/shade-robotics/motion-planning-vs-path-planning-e29ced760b90`.

[37] Jacob T Schwartz and Micha Sharir. "On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds". In: *Advances in applied Mathematics* 4.3 (1983), pp. 298–351.

[38] Jean-Claude Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012.

[39] Steven M LaValle. "Extensions of Basic Motion Planning". In: (2006).

[40] Tomas Lozano-Perez. *Spatial planning: A configuration space approach*. Springer, 1990.

[41] Michael Erdmann and Tomas Lozano-Perez. "On multiple moving objects". In: *Algorithmica* 2 (1987), pp. 477–521.

[42] Mohammad R Alenezi, Abdullah M Almeshal, et al. "Optimal Path Planning for a Remote Sensing Unmanned Ground Vehicle in a Hazardous Indoor Environment". In: *Intelligent Control and Automation* 9.04 (2018), p. 147.

[43] Genya Ishigami, Keiji Nagatani, and Kazuya Yoshida. "Path planning and evaluation for planetary rovers based on dynamic mobility index". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 601–606.

[44] Mary B Alatise and Gerhard P Hancke. "A review on challenges of autonomous mobile robot and sensor fusion methods". In: *IEEE Access* 8 (2020), pp. 39830–39846.

[45] Tomás Lozano-Pérez and Michael A Wesley. "An algorithm for planning collision-free paths among polyhedral obstacles". In: *Communications of the ACM* 22.10 (1979), pp. 560–570.

[46] Franz Aurenhammer. “Voronoi diagrams—a survey of a fundamental geometric data structure”. In: *ACM Computing Surveys (CSUR)* 23.3 (1991), pp. 345–405.

[47] Lydia E Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[48] Melih Özcan and Ulas Yaman. “A continuous path planning approach on Voronoi diagrams for robotics and manufacturing applications”. In: *Procedia Manufacturing* 38 (2019), pp. 1–8.

[49] Sunil Kumar and Afzal Sikander. “A modified probabilistic roadmap algorithm for efficient mobile robot path planning”. In: *Engineering Optimization* 55.9 (2023), pp. 1616–1634.

[50] Jean-Claude Latombe and Jean-Claude Latombe. “Exact cell decomposition”. In: *Robot motion planning* (1991), pp. 200–247.

[51] Jean-Claude Latombe and Jean-Claude Latombe. “Approximate cell decomposition”. In: *Robot Motion Planning* (1991), pp. 248–294.

[52] Jin-Woo Jung et al. “Expanded Douglas–Peucker polygonal approximation and opposite angle-based exact cell decomposition for path planning with curvilinear obstacles”. In: *Applied Sciences* 9.4 (2019), p. 638.

[53] Omnia AA Salama et al. “RCD: radial cell decomposition algorithm for mobile robot path planning”. In: *IEEE Access* 9 (2021), pp. 149982–149992.

[54] Steven LaValle. “Rapidly-exploring random trees: A new tool for path planning”. In: *Research Report 9811* (1998).

[55] Binghui Li and Badong Chen. “An adaptive rapidly-exploring random tree”. In: *IEEE/-CAA Journal of Automatica Sinica* 9.2 (2021), pp. 283–294.

[56] Bin Liao et al. “F-RRT*: An improved path planning algorithm with improved initial solution and convergence rate”. In: *Expert Systems with Applications* 184 (2021), p. 115457.

[57] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *The international journal of robotics research* 5.1 (1986), pp. 90–98.

[58] Xin Lin, Zhan-Qing Wang, and Xu-Yang Chen. “Path planning with improved artificial potential field method based on decision tree”. In: *2020 27th Saint Petersburg International Conference on integrated navigation systems (ICINS)*. IEEE. 2020, pp. 1–5.

[59] Ulises Orozco-Rosas et al. “Mobile robot path planning using a QAPF learning algorithm for known and unknown environments”. In: *IEEE Access* 10 (2022), pp. 84648–84663.

[60] Jaafar Ahmed Abdulsaheb and Dheyaa Jasim Kadhim. “Classical and heuristic approaches for mobile robot path planning: A survey”. In: *Robotics* 12.4 (2023), p. 93.

[61] Chatterjee Prasenjit et al. “Model for selecting a route for the transport of hazardous materials using a fuzzy logic system”. In: *Vojnotehnički glasnik* 69.2 (2021), pp. 355–390.

[62] Xiao-Huan Liu et al. “A new algorithm of the best path selection based on machine learning”. In: *IEEE Access* 7 (2019), pp. 126913–126928.

[63] Yiqi Xu et al. “Research progress of nature-inspired metaheuristic algorithms in mobile robot path planning”. In: *Electronics* 12.15 (2023), p. 3263.

[64] Saman M Almufti et al. “Overview of metaheuristic algorithms”. In: *Polaris Global Journal of Scholarly Research and Trends* 2.2 (2023), pp. 10–32.

[65] Lotfi Asker Zadeh. “Fuzzy sets”. In: *Information and control* 8.3 (1965), pp. 338–353.

[66] Jinghua Wang et al. “A fuzzy logic path planning algorithm based on geometric landmarks and kinetic constraints”. In: *Information Technology and Control* 51.3 (2022), pp. 499–514.

[67] Dimitrios V Lyridis. “An improved ant colony optimization algorithm for unmanned surface vehicle local path planning with multi-modality constraints”. In: *Ocean Engineering* 241 (2021), p. 109890.

[68] Charis Ntakolia et al. “A Genetic Algorithm enhanced with Fuzzy-Logic for multi-objective Unmanned Aircraft Vehicle path planning missions”. In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2022, pp. 114–123.

[69] Jia Hui Teo, Nur Syazreen Ahmad, and Patrick Goh. “Visual stimuli-based dynamic commands with intelligent control for reactive BCI applications”. In: *IEEE Sensors Journal* 22.2 (2021), pp. 1435–1448.

[70] Chin Kui Ku et al. “Jitter decomposition of high-speed data signals from jitter histograms with a pole–residue representation using multilayer perceptron neural networks”. In: *IEEE Transactions on Electromagnetic Compatibility* 62.5 (2019), pp. 2227–2237.

[71] Imane Arrouch et al. “Close proximity time-to-collision prediction for autonomous robot navigation: an exponential GPR approach”. In: *Alexandria Engineering Journal* 61.12 (2022), pp. 11171–11183.

[72] Asita Kumar Rath et al. “Application of artificial neural network for control and navigation of humanoid robot”. In: *Journal of Mechanical Engineering and Sciences* 12.2 (2018), p. 3529.

[73] Mahamat Loutfi Imrane et al. “Artificial potential field neuro-fuzzy controller for autonomous navigation of mobile robots”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 235.7 (2021), pp. 1179–1192.

[74] Mohammad Samadi Gharajeh and Hossein B Jond. “An intelligent approach for autonomous mobile robots path planning based on adaptive neuro-fuzzy inference system”. In: *Ain Shams Engineering Journal* 13.1 (2022), p. 101491.

[75] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: (1989).

[76] Abderraouf Maoudj and Abdelfetah Hentout. “Optimal path planning approach based on Q-learning algorithm for mobile robots”. In: *Applied Soft Computing* 97 (2020), p. 106796.

[77] Pengzhan Chen et al. “A deep reinforcement learning based method for real-time path planning and dynamic obstacle avoidance”. In: *Neurocomputing* 497 (2022), pp. 64–75.

[78] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[79] Kun Hao et al. “The application of an adaptive genetic algorithm based on collision detection in path planning of mobile robots”. In: *Computational Intelligence and Neuroscience* 2021.1 (2021), p. 5536574.

[80] Yanrong Hu and Simon X Yang. “A novel knowledge-based genetic algorithm for robot path planning in complex environments”. In: *arXiv preprint arXiv:2209.01482* (2022).

[81] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. ieee. 1995, pp. 1942–1948.

[82] Xun Li et al. “An improved method of particle swarm optimization for path planning of mobile robot”. In: *Journal of Control Science and Engineering* 2020.1 (2020), p. 3857894.

[83] Li Zheng et al. “Particle Swarm Algorithm Path-Planning Method for Mobile Robots Based on Artificial Potential Fields”. In: *Sensors* 23.13 (2023), p. 6082.

[84] Marco Dorigo and Luca Maria Gambardella. “Ant colonies for the travelling salesman problem”. In: *biosystems* 43.2 (1997), pp. 73–81.

[85] Changwei Miao et al. “Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm”. In: *Computers & Industrial Engineering* 156 (2021), p. 107230.

[86] Hongliu Huang, Guo Tan, and Linli Jiang. “Robot path planning using improved ant colony algorithm in the environment of internet of things”. In: *Journal of robotics* 2022.1 (2022), p. 1739884.

[87] Xin-She Yang and Suash Deb. “Cuckoo search via Lévy flights”. In: *2009 World congress on nature & biologically inspired computing (NaBIC)*. Ieee. 2009, pp. 210–214.

[88] Kaushlendra Sharma, Shikha Singh, and Rajesh Doriya. “Optimized cuckoo search algorithm using tournament selection function for robot path planning”. In: *International Journal of Advanced Robotic Systems* 18.3 (2021), p. 1729881421996136.

[89] Bandita Sahu, Pradipta Kumar Das, and Raghvendra Kumar. “A modified cuckoo search algorithm implemented with SCA and PSO for multi-robot cooperation and path planning”. In: *Cognitive Systems Research* 79 (2023), pp. 24–42.

[90] Xin-She Yang. “Firefly algorithms for multimodal optimization”. In: *International symposium on stochastic algorithms*. Springer. 2009, pp. 169–178.

[91] Ting-Wei Zhang et al. “A new hybrid algorithm for path planning of mobile robot”. In: *The Journal of Supercomputing* 78.3 (2022), pp. 4158–4181.

[92] Mourad Achouri and Youcef Zennir. “Path planning and tracking of wheeled mobile robot: using firefly algorithm and kinematic controller combined with sliding mode control”. In: *Journal of the Brazilian Society of Mechanical Sciences and Engineering* 46.4 (2024), p. 228.

[93] Dervis Karaboga et al. *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Erciyes university, 2005.

[94] Rafal Szczepanski and Tomasz Tarczewski. “Global path planning for mobile robot based on Artificial Bee Colony and Dijkstra’s algorithms”. In: *2021 IEEE 19th International Power Electronics and Motion Control Conference (PEMC)*. IEEE. 2021, pp. 724–730.

[95] Yibing Cui, Wei Hu, and Ahmed Rahmani. “A reinforcement learning based artificial bee colony algorithm with application in robot path planning”. In: *Expert Systems with Applications* 203 (2022), p. 117389.

[96] Xin-She Yang. “A new metaheuristic bat-inspired algorithm”. In: *Nature inspired cooperative strategies for optimization (NICSO 2010)*. Springer, 2010, pp. 65–74.

[97] Xin Yuan, Xinwei Yuan, and Xiaohu Wang. "Path planning for mobile robot based on improved bat algorithm". In: *Sensors* 21.13 (2021), p. 4389.

[98] Gongfeng Xin et al. "Mobile robot path planning with reformative bat algorithm". In: *Plos one* 17.11 (2022), e0276577.

[99] Wen-Tsao Pan. "A new fruit fly optimization algorithm: taking the financial distress model as an example". In: *Knowledge-Based Systems* 26 (2012), pp. 69–74.

[100] Kunming Shi, Xiangyin Zhang, and Shuang Xia. "Multiple swarm fruit fly optimization algorithm based path planning method for multi-UAVs". In: *Applied Sciences* 10.8 (2020), p. 2822.

[101] Bing Hao et al. "Automatic recharging path planning for cleaning robots". In: *mobile information systems* 2021.1 (2021), p. 5558096.

[102] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. "Grey wolf optimizer". In: *Advances in engineering software* 69 (2014), pp. 46–61.

[103] Tahseen Fadhel Abaas and Alaa Hassan Shabeeb. "Safe and optimum navigation of wheeled mobile robot using grey wolf optimization algorithm". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 928. 2. IOP Publishing. 2020, p. 022006.

[104] Faiza Gul et al. "Meta-heuristic approach for solving multi-objective path planning for autonomous guided robot using PSO–GWO optimization algorithm with evolutionary programming". In: *Journal of Ambient Intelligence and Humanized Computing* 12 (2021), pp. 7873–7890.

[105] Seyedali Mirjalili and Andrew Lewis. "The whale optimization algorithm". In: *Advances in engineering software* 95 (2016), pp. 51–67.

[106] Yaonan Dai et al. "A novel whale optimization algorithm of path planning strategy for mobile robots". In: *Applied Intelligence* 53.9 (2023), pp. 10843–10857.

[107] Tao Tian et al. "Hybrid Whale Optimization with a Firefly Algorithm for Function Optimization and Mobile Robot Path Planning". In: *Biomimetics* 9.1 (2024), p. 39.

[108] Lixing Liu et al. "Path planning techniques for mobile robots: Review and prospect". In: *Expert Systems with Applications* (2023), p. 120254.

[109] Zaharuddeen Haruna et al. "Path Planning Algorithms for Mobile Robots: A Survey". In: *Motion Planning for Dynamic Agents*. IntechOpen, 2023.

[110] Nguyen Hung et al. “A review of path following control strategies for autonomous robotic vehicles: Theory, simulations, and experiments”. In: *Journal of Field Robotics* 40.3 (2023), pp. 747–779.

[111] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. “A review of mobile robots: Concepts, methods, theoretical framework, and applications”. In: *International Journal of Advanced Robotic Systems* 16.2 (2019), p. 1729881419839596.

[112] Luca Bruzzone and Giuseppe Quaglia. “Locomotion systems for ground mobile robots in unstructured environments”. In: *Mechanical sciences* 3.2 (2012), pp. 49–62.

[113] Yaoyao Wang et al. “A new redundancy resolution for underwater vehicle–manipulator system considering payload”. In: *International Journal of Advanced Robotic Systems* 14.5 (2017), p. 1729881417733934.

[114] Sanjoy Kumar Debnath, Rosli Omar, and Nor Badariyah Abdul Latip. “A review on energy efficient path planning algorithms for unmanned air vehicles”. In: *Computational Science and Technology: 5th ICCST 2018, Kota Kinabalu, Malaysia, 29-30 August 2018* (2019), pp. 523–532.

[115] Paul J Springer. *Military robots and drones: a reference handbook*. ABC-CLIO, 2013.

[116] Riky Tri Yunardi et al. “Holonomic implementation of three wheels omnidirectional mobile robot using dc motors”. In: *Journal of Robotics and Control (JRC)* 2.2 (2021), pp. 65–71.

[117] Salua Hamaza and Mirko Kovac. “Omni-drone: on the design of a novel aerial manipulator with omni-directional workspace”. In: *2020 17th International Conference on Ubiquitous Robots (UR)*. IEEE. 2020, pp. 153–158.

[118] Rodríguez González, Francisco Rodríguez, and J Luis Guzmán. “Autonomous tracked robots in planar off-road conditions”. In: *Modeling, Localization, and Motion Control; Springer: Berlin, Germany* (2014).

[119] Alexandr Štefek et al. “Optimization of fuzzy logic controller used for a differential drive wheeled mobile robot”. In: *Applied Sciences* 11.13 (2021), p. 6023.

[120] Sotirios Spanogianopoulos, Konstantinos Sirlantzis, and Kenan Ahiska. “RRT-based Path Planning for Car-Like Vehicles with Nonholonomic Constraints”. In: *2022 30th Mediterranean Conference on Control and Automation (MED)*. IEEE. 2022, pp. 426–431.

[121] Yutaka Kanayama et al. “A stable tracking control method for an autonomous mobile robot”. In: *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE. 1990, pp. 384–389.

[122] Timm Faulwasser, Benjamin Kern, and Rolf Findeisen. "Model predictive path-following for constrained nonlinear systems". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 8642–8647.

[123] Chen Hua. "A Review of Visual Perception for Mobile Robot Navigation: Methods, Limitation, and Applications". In: *2022 2nd International Conference on Algorithms, High Performance Computing and Artificial Intelligence (AHPCAI)*. IEEE. 2022, pp. 729–737.

[124] David Kortenkamp. "Perception for mobile robot navigation: A survey of the state of the art". In: *Dual-Use Space Technology Transfer Conference and Exhibition, Volume 2*. 1994.

[125] MJ Gilmartin. "INTRODUCTION TO AUTONOMOUS MOBILE ROBOTS, by Roland Siegwart and Illah R. Nourbakhsh, MIT Press, 2004, xiii+ 321 pp., ISBN 0-262-19502-X.(Hardback,£ 27.95)". In: *Robotica* 23.2 (2005), pp. 271–272.

[126] Fabian de Ponte Müller. "Survey on ranging sensors and cooperative techniques for relative positioning of vehicles". In: *Sensors* 17.2 (2017), p. 271.

[127] Kale Aparna. "Bodhale, Umesh "Overview Of Sensors For Robotics"". In: *International Journal of Engineering Research and Technology (IJERT) Volume* 2 (2013).

[128] Adnan Mohsin Abdulazeez and Fayez Saeed Faizi. "Vision-based mobile robot controllers: a scientific review". In: *Turkish Journal of Computer and Mathematics Education (TURCOMAT)* 12.6 (2021), pp. 1563–1580.

[129] Davison. "Real-time simultaneous localisation and mapping with a single camera". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. IEEE. 2003, pp. 1403–1410.

[130] Young-Sik Shin, Yeong Sang Park, and Ayoung Kim. "DVL-SLAM: Sparse depth enhanced direct visual-LiDAR SLAM". In: *Autonomous Robots* 44.2 (2020), pp. 115–130.

[131] Chih-Chung Chou and Cheng-Fu Chou. "Efficient and accurate tightly-coupled visual-lidar slam". In: *IEEE Transactions on Intelligent Transportation Systems* 23.9 (2021), pp. 14509–14523.

[132] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.

[133] Georg Klein and David Murray. "Parallel tracking and mapping for small AR workspaces". In: *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE. 2007, pp. 225–234.

[134] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.

[135] Jianjun Ni et al. "An improved adaptive ORB-SLAM method for monocular vision robot under dynamic environments". In: *International Journal of Machine Learning and Cybernetics* 13.12 (2022), pp. 3821–3836.

[136] Ki-Sik Kim and Jong-Seung Park. "Spherical PTAM: a versatile SLAM for spherical video". In: *Multimedia Tools and Applications* 82.21 (2023), pp. 32151–32175.

[137] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. "DTAM: Dense tracking and mapping in real-time". In: *2011 international conference on computer vision*. IEEE. 2011, pp. 2320–2327.

[138] Jakob Engel, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-scale direct monocular SLAM". In: *European conference on computer vision*. Springer. 2014, pp. 834–849.

[139] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. "SVO: Fast semi-direct monocular visual odometry". In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 15–22.

[140] Jianwei Ren. "An improved binocular LSD_SLAM method for object localization". In: *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. IEEE. 2020, pp. 30–33.

[141] Shing Yan Loo et al. "CNN-SVO: Improving the mapping in semi-direct visual odometry using single-image depth prediction". In: *2019 International conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 5218–5223.

[142] Berthold KP Horn and Brian G Schunck. "Determining optical flow". In: *Artificial intelligence* 17.1-3 (1981), pp. 185–203.

[143] Ernesto Moya-Albor et al. "Optical Flow-Hermite and Fuzzy Q-Learning Based Robotic Navigation Approach". In: *2021 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*. IEEE. 2021, pp. 26–31.

[144] Haram Kim and H Jin Kim. "Real-time rotational motion estimation with contrast maximization over globally aligned events". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 6016–6023.

[145] Fuseini Mumuni, Alhassan Mumuni, and Christian Kwaku Amuzuvi. "Deep learning of monocular depth, optical flow and ego-motion with geometric guidance for UAV navigation in dynamic environments". In: *Machine Learning with Applications* 10 (2022), p. 100416.

[146] Celyn Walters and Simon Hadfield. "Evreflex: Dense time-to-impact prediction for event-based obstacle avoidance". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1304–1309.

[147] Hann Woei Ho, Guido CHE de Croon, and Qiping Chu. "Distance and velocity estimation using optical flow from a monocular camera". In: *International Journal of Micro Air Vehicles* 9.3 (2017), pp. 198–208.

[148] Ernesto Moya-Albor et al. "Vision-based autonomous navigation with evolutionary learning". In: *Advances in Computational Intelligence: 19th Mexican International Conference on Artificial Intelligence, MICAI 2020, Mexico City, Mexico, October 12–17, 2020, Proceedings, Part II 19*. Springer. 2020, pp. 459–471.

[149] Ernesto Moya-Albor et al. "Bio-inspired optical flow-based autonomous obstacle avoidance control". In: *2019 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*. IEEE. 2019, pp. 18–23.

[150] Niall O'Mahony et al. "Deep learning for visual navigation of unmanned ground vehicles: a review". In: *2018 29th Irish Signals and Systems Conference (ISSC)*. IEEE. 2018, pp. 1–6.

[151] Weili Chen, Chao Jia, and Shuai He. "End-to-End Vision-to-Motion Model with Auxiliary Segmentation Module for Indoor Navigation". In: *Proceedings of the 2019 3rd International Conference on Deep Learning Technologies*. 2019, pp. 75–80.

[152] Fanyu Zeng, Chen Wang, and Shuzhi Sam Ge. "A survey on visual navigation for artificial agents with deep reinforcement learning". In: *IEEE Access* 8 (2020), pp. 135426–135442.

[153] Muhammad Mudassir Ejaz, Tong Boon Tang, and Cheng-Kai Lu. "Vision-based autonomous navigation approach for a tracked robot using deep reinforcement learning". In: *IEEE Sensors Journal* 21.2 (2020), pp. 2230–2240.

[154] KN McGuire et al. "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment". In: *Science Robotics* 4.35 (2019), eaaw9710.

[155] Chao Tang et al. "A 3d range-only slam algorithm based on improved derivative ukf". In: *Electronics* 11.7 (2022), p. 1109.

[156] Andrew Farley, Jie Wang, and Joshua A. Marshall. "How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion". In: *Simulation Modelling Practice and Theory* 120 (2022), p. 102629. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2022.102629. URL: https://www.sciencedirect.com/science/article/pii/S1569190X22001046.

[157] Guelis Montenegro et al. "Modeling and control of a spherical robot in the CoppeliaSim simulator". In: *Sensors* 22.16 (2022), p. 6020.

[158] Saharsh Oswal and D Saravanakumar. "Line following robots on factory floors: Significance and Simulation study using CoppeliaSim". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 1012. 1. IOP Publishing. 2021, p. 012008.

[159] Alberto Marroquín et al. "Mobile robot navigation based on embedded computer vision". In: *Mathematics* 11.11 (2023), p. 2561.

[160] Wan Zafira Ezza Wan Zakaria et al. "Motion Planning of Differential Drive Mobile Robot Using Circular Arc." In: *Engineering Letters* 32.1 (2024).

[161] MA Pastrana et al. "Teaching Control Theory using Mobile Robot Obstacle Following/Avoidance with CoppeliaSim and MFO Algorithm." In: *2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE)*. IEEE. 2023, pp. 579–584.

[162] Adept MobileRobots. *Pioneer 3-DX User Guide*. Version A. 2011. URL: https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf.

[163] Jing Liu et al. "Autonomous Scouting Robot Based on Pioneer P3-DX". In: *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE. 2019, pp. 1392–1397.

[164] Esa Apriaskar et al. "Autonomous Mobile Robot based on BehaviourBased Robotic using V-REP Simulator–Pioneer P3-DX Robot". In: *Jurnal Rekayasa Elektrika* 16.1 (2020).

[165] Komal Parashar. "Automated Robotic Navigation with Obstacle Detection Using a Deep Learning Algorithm". In: *Advancing Sustainable Science and Technology for a Resilient Future*. CRC Press, 2024, pp. 339–342.

[166] Yilun Liu and Xiaoming Li. "A Hybrid Mobile Robot Path Planning Scheme Based on Modified Gray Wolf Optimization and Situation Assessment". In: *Journal of Robotics* 2022 (2022).

[167] Yunyun Liu et al. "Review of the grey wolf optimization algorithm: variants and applications". In: *Neural Computing and Applications* 36.6 (2024), pp. 2713–2735.

[168] L Doğan and U Yüzgeç. "Robot path planning using gray wolf optimizer". In: *Proceedings-International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18)*. 2018, p. 2018.

[169] Lili Liu et al. "Enhanced grey wolf optimization algorithm for mobile robot path planning". In: *Electronics* 12.19 (2023), p. 4026.

[170] Chittaranjan Paital et al. "Navigation of a wheeled mobile robotic agent using modified grey wolf optimization controller". In: *International Journal of Intelligent Unmanned Systems* 11.2 (2023), pp. 197–213.

[171] Eddy Ilg et al. "Flownet 2.0: Evolution of optical flow estimation with deep networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2462–2470.

[172] James J Gibson. "The perception of the visual world." In: (1950).

[173] Bruce D Lucas and Takeo Kanade. "An iterative image registration technique with an application to stereo vision". In: *IJCAI'81: 7th international joint conference on Artificial intelligence*. Vol. 2. 1981, pp. 674–679.

[174] Gunnar Farnebäck. "Two-frame motion estimation based on polynomial expansion". In: *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13*. Springer. 2003, pp. 363–370.

[175] "opticalFlowFarneback". In: *MathWorks* (2024). Accessed: 2023-08-12. URL: `https://www.mathworks.com/help/vision/ref/opticalflowfarneback.html`.

[176] Zhe Ren et al. "Unsupervised deep learning for optical flow estimation". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[177] Junhwa Hur and Stefan Roth. "Optical flow estimation in the deep learning age". In: *Modelling human motion: from human perception to robot design* (2020), pp. 119–140.

[178] Farli Rossi et al. "Implementation of fuzzy logic in PLC for three-story elevator control system". In: *2021 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE)*. IEEE. 2021, pp. 179–185.

[179] Yousef Ibrahim Daradkeh et al. "Development of effective methods for structural image recognition using the principles of data granulation and apparatus of fuzzy logic". In: *IEEE Access* 9 (2021), pp. 13417–13428.

[180] En-Hui Li et al. "A fuzzy coordination control of a water membrane evaporator cooling system for aerospace electronics". In: *Applied Thermal Engineering* 191 (2021), p. 116872.

[181] Abdelkader Chabchoub et al. “Intelligent traffic light controller using fuzzy logic and image processing”. In: *International Journal of Advanced Computer Science and Applications* 12.4 (2021).

[182] Milind Gupta et al. “Application of Fuzzy Logic Data Analysis Method for Business Development”. In: *International Conference on Business and Technology*. Springer. 2020, pp. 75–93.

[183] G Ramesh et al. “Estimation analysis of paralysis effects for human nervous system by using Neuro fuzzy logic controller”. In: *NeuroQuantology* 20.8 (2022), p. 3195.

[184] Hajer Omrane, Mohamed Slim Masmoudi, and Mohamed Masmoudi. “Fuzzy logic based control for autonomous mobile robot navigation”. In: *Computational intelligence and neuroscience* 2016 (2016).

[185] Halim Mudia. “Comparative Study of Mamdani-type and Sugeno-type Fuzzy Inference Systems for Coupled Water Tank”. In: *Indonesian Journal of Artificial Intelligence and Data Mining* 3.1 (2020), pp. 42–49.

[186] Ebrahim H Mamdani. “Application of fuzzy algorithms for control of simple dynamic plant”. In: *Proceedings of the institution of electrical engineers*. Vol. 121. 12. IET. 1974, pp. 1585–1588.

[187] Asep Najmurrokhman et al. “Mamdani based fuzzy logic controller for a wheeled mobile robot with obstacle avoidance capability”. In: *2019 International Conference on Mechatronics, Robotics and Systems Engineering (MoRSE)*. IEEE. 2019, pp. 49–53.

[188] Ehsan Pourjavad and Rene V Mayorga. “A comparative study and measuring performance of manufacturing systems with Mamdani fuzzy inference system”. In: *Journal of Intelligent Manufacturing* 30 (2019), pp. 1085–1097.

[189] Anazia E Kizito, Emmanuel Ojei, and MD Okpor. “A Fuzzy Logic-Based Automobile Fault Detection System Using Mamdani Algorithm”. In: *Valley International Journal Digital Library* (2024), pp. 1081–1093.

[190] Tomohiro Takagi and Michio Sugeno. “Fuzzy identification of systems and its applications to modeling and control”. In: *IEEE transactions on systems, man, and cybernetics* 1 (1985), pp. 116–132.

[191] Ria Yuliani Kartikasari, Graha Prakarsa, and Deden Pradeka. “Optimization of traffic light control using fuzzy logic sugeno method”. In: *International Journal of Global Operations Research* 1.2 (2020), pp. 51–61.

[192] Khaled Akka and Farid Khaber. "Optimal fuzzy tracking control with obstacles avoidance for a mobile robot based on Takagi-Sugeno fuzzy model". In: *Transactions of the Institute of Measurement and Control* 41.10 (2019), pp. 2772–2781.

[193] Weixuan Wang, Jingbo Peng, and Yu Zhang. "Modeling and control for an aero-engine based on the takagi-sugeno fuzzy model". In: *Aerospace* 10.6 (2023), p. 523.

[194] "Fuzzy Inference Process". In: *MathWorks* (2024). Accessed: 2024-09-16. URL: `https://www.mathworks.com/help/fuzzy/fuzzy-inference-process.html`.

[195] "Mamdani and Sugeno Fuzzy Inference Systems". In: *MathWorks* (2024). Accessed: 2024-09-16. URL: `https://www.mathworks.com/help/fuzzy/types-of-fuzzy-inference-systems.html`.

[196] A Prakash Moon, KK Jajulwar, et al. "Design of adaptive fuzzy tracking controller for Autonomous navigation system". In: *International Journal of Recent Trend in Engineering and Research* 2.2 (2016), pp. 268–275.

[197] Najah Yousfi Allagui, Farhan A Salem, and Awad M Aljuaid. "Artificial Fuzzy-PID Gain Scheduling Algorithm Design for Motion Control in Differential Drive Mobile Robotic Platforms". In: *Computational Intelligence and Neuroscience* 2021.1 (2021), p. 5542888.

[198] Najah Yousfi Allagui, Donia Benhalima Abid, and Nabil Derbel. "Autonomous navigation of mobile robot with combined fractional order PI and fuzzy logic controllers". In: *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*. IEEE. 2019, pp. 78–83.

[199] Lingling Li et al. "Mobile robot navigation control using recurrent fuzzy cerebellar model articulation controller based on improved dynamic artificial bee colony". In: *Advances in Mechanical Engineering* 8.11 (2016), p. 1687814016681234.

[200] Cheng-Hung Chen, Shiou-Yun Jeng, and Cheng-Jian Lin. "Using an adaptive fuzzy neural network based on a multi-strategy-based artificial bee colony for mobile robot control". In: *Mathematics* 8.8 (2020), p. 1223.

[201] Cheng-Hung Chen et al. "Using ultrasonic sensors and a knowledge-based neural fuzzy controller for mobile robot navigation control". In: *Electronics* 10.4 (2021), p. 466.

[202] "Optical Flow". In: *OpenCV* (). Accessed: 2023-06-24. URL: `https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html`.

[203] Po-Yin Yen et al. "Nurses' time allocation and multitasking of nursing activities: a time motion study". In: *AMIA annual symposium proceedings*. Vol. 2018. American Medical Informatics Association. 2018, p. 1137.

[204] Nord Module. *Autonomous mobile robots in healthcare: revolutionizing patient care and workflow efficiency*. Accessed on July 27th, 2024. 2020. URL: https://www.nord-modules.com/amrs-in-healthcare/.

[205] Hongwei Qin et al. "Review of autonomous path planning algorithms for mobile robots". In: *Drones* 7.3 (2023), p. 211.

Acronyms

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
ACD	Approximate Cell Decomposition
AMR	Autonomous Mobile Robot
APF	Artificial Potential Field
BA	Bats Algorithm
CD	Cell Decomposition
CSA	Cuckoo Search Algorithm
DA	Dijkstra's algorithm
DCNN	Deep Convolutional Neural Network
DRL	Deep Reinforcement Learning
DTAM	Dense Tracking And Mapping
EC	Exteroceptive
ECD	Exact Cell Decomposition
FA	Firefly Algorithm
FFO	Fruit Fly Optimization
FLC	Fuzzy Logic Controller
FL	Fuzzy Logic
FQL	Fuzzy Q-Learning
GA	Genetic Algorithm
GWO	Grey Wolf Optimization
IFR	International Federation of Robotics

IoT	Internet of Things
LiDAR	Light Detection and Ranging
LSD-SLAM	Large-Scale Direct SLAM
NN	Neural Network
OAC	Obstacle Avoidance Controller
OF	Optical Flow
PC	Proprioceptive
PRM	Probabilistic Roadmap
PTAM	Parallel Tracking and Mapping
PSO	Particle Swarm Optimization
RA	Roadmap Approach
RaaS	Robotics as a Service
RL	Reinforcement Learning
RRT	Rapidly-exploring Random Tree
SAC	Soft Actor-Critic
SLAM	Simultaneous Localization and Mapping
SVO	Semi-direct Visual Odometry
T-S	Takagi-Sugeno
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle
VG	Visibility Graph
VD	Voronoi Diagram
WMR	Wheeled Mobile Robot
WOA	Whale Optimization Algorithm