PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

ESCUELA DE POSGRADO

Autonomous Navigation of Differential Robot with Depth Camera and Laser Range Finder Using Deep Reinforcement Learning In Non-Stationary Indoor Environments

Tesis para optar el grado académico de Doctor en Ingeniería
que presenta:

Diego Martín Arce Cigüeñas

Asesor:

Cesar Armando Beltrán Castañón

Lima, 2024

Informe de Similitud

Yo, Cesar Armando Beltrán Castañón, docente de la Escuela de Posgrado de la Pontificia Universidad Católica del Perú, asesor de la tesis de investigación titulada “Autonomous Navigation of Differential Robot with Depth Camera and Laser Range Finder Using Deep Reinforcement Learning In Non-Stationary Indoor Environments”, del autor Diego Martin Arce Cigüeñas, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 51%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 09/10/2024. En este informe se incluyen las fuentes “mdpi-res.com” y “www.mdpi.com” con porcentajes de 22% y 18%, las cuales corresponden a los repositorios en donde se encuentra un artículo de la revista MDPI Sensors publicado por el autor a partir del trabajo de tesis. Excluyendo estas dos fuentes el índice de puntuación de similitud es de 12%.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:
10 de octubre del 2024

Apellidos y nombres del asesor / de la asesora: Beltrán Castañón, Cesar Armando	
DNI: 29561260	Firma 
ORCID: 0000-0002-0173-4140	

Acknowledgments

The authors of this thesis would like to thank CONCYTEC-PROCIENCIA and the Pontificia Universidad Católica del Perú for providing the means and resources for this research and development. This work was financially supported by CONCYTEC – PROCIENCIA within the framework of the call E074-"Tesis y pasantías en ciencia, tecnología e innovación" [Contract N° PE501081648-2022].

Abstract

Intelligent systems are an area of computer science that seeks to solve complex and multidisciplinary problems by using an automatic focus by capturing information from which it can perform actions and evaluate its result in order to learn from its experience and improve its performance and efficiency. These systems can be applied in robotics for autonomous navigation purposes, where the use of conventional automatic control techniques is very complex. The topics with the greatest potential are Reinforcement Learning and Deep Learning, which through their combination is possible to solve highly complex problems from the training of computational frameworks based on rewards for taking a correct action or penalties for incorrect actions.

The thesis project is focused on providing the autonomous navigation capacity to a differential robot making use of a Lidar depth camera and Deep Reinforcement Learning (DRL) techniques for use in indoor environments with non-stationary elements. For its development, a review of the DRL frameworks used for autonomous navigation will be carried out, from which the development of frameworks based on DRL techniques for autonomous robot navigation will be considered in indoor environments where elements are non-stationary. Subsequently, the evaluation of the frameworks in virtual platforms is proposed to determine its performance. Finally, a robotic platform with differential traction and a Lidar depth camera will be conditioned to be used in a validation stage through experiments in real environments.

Resumen

Los sistemas inteligentes son un área de las ciencias de la computación que busca resolver problemas complejos y multidisciplinares mediante el uso de un enfoque automático mediante la captura de información a partir de la cual puede realizar acciones y evaluar su resultado con el fin de aprender de su experiencia y mejorar su desempeño y eficiencia. Estos sistemas pueden ser aplicados en robótica con fines de navegación autónoma, donde el uso de técnicas convencionales de control automático es muy complejo. Los temas con mayor potencial son el Aprendizaje por Refuerzo y el Aprendizaje Profundo, que a través de su combinación es posible resolver problemas de alta complejidad a partir del entrenamiento de marcos computacionales basados en recompensas por realizar una acción correcta o penalizaciones por acciones incorrectas.

El proyecto de tesis está enfocado en dotar de la capacidad de navegación autónoma a un robot diferencial haciendo uso de una cámara de profundidad Lidar y técnicas de Aprendizaje Profundo por Refuerzo (DRL) para su uso en ambientes interiores con elementos no estacionarios. Para su desarrollo se realizará una revisión de los marcos DRL utilizados para navegación autónoma, a partir de los cuales se planteará el desarrollo de marcos basados en técnicas DRL para navegación autónoma de robots en ambientes interiores donde los elementos sean no estacionarios. Posteriormente, se propone la evaluación de los frameworks en plataformas virtuales para determinar su desempeño. Finalmente, se acondicionará una plataforma robótica con tracción diferencial y una cámara de profundidad Lidar para ser utilizada en una etapa de validación mediante experimentos en entornos reales.

Contents

List of Figures

List of Tables

Chapter I

Introduction

The first chapter describes the general aspects of this thesis, providing a comprehensive overview of the research context and its significance. Beginning with the background section, it situates the study within the broader field, highlighting the key motivations and underlying issues that requires this research. The justification elaborates on the importance of the study, explaining why this particular research is crucial at this point in time. The theoretical framework explores the foundational theories and concepts from computer science, robot navigation, and the intersection of these fields, offering a robust conceptual basis for the study. The state of the art examines current developments in DRL frameworks, navigation scenarios, sensors, and robotics platforms, identifying gaps and opportunities for innovation. The objectives and scope section clearly outlines the research aims and boundaries, while the hypothesis presents the primary research assumptions. The area of specialization defines the specific focus within the broader field. The methodology describes the research approach and techniques employed. Finally, the chapter discusses challenges and opportunities, and the expected contributions of the research, underscoring its potential impact and significance.

1.1 Background

Autonomous navigation of ground mobile robots is a topic highly studied internationally due to its potential application in multiple areas, such as logistics, mining, medicine, care and others [1]. During the last years, these studies have been focused on the use of Deep Learning by Reinforcement (DRL) to provide the autonomous navigation capacity to robots being validated with tests in simulated environments and in real environments [2–18].

Studies where DRL is used for robot navigation have been carried out both indoors [2, 4–9, 11–18] and outdoors [3, 10] where the algorithms for the development of computational frameworks require different considerations. Furthermore, the research carried out shows that it is possible to consider providing the autonomous navigation capacity to a robot by means of DRL from the use of information from known maps [2, 4, 6], partial information from maps [8] or without information on the environment [9, 11, 13–16] , for which different algorithms can be used. Another important aspect to take into account is the need to interact in environments with non-stationary elements. Therefore, some studies will be taken as a reference in which the interaction of the robot with people [3, 10] and other non-stationary elements are taken into account [6, 17] during robot navigation.

1.2 Justification

The use of mobile robots for different applications in indoor environments, such as warehouses, hospitals, offices, restaurants and others, is a trend that has been increasing in recent years. In most applications, the aim is to automate the processes within these environments through the integration of robots, for which it is required that they have autonomous navigation capacity in order to move without the need for an operator or some type of interaction with people.

DRL is a very convenient and proven alternative to provide autonomous navigation capabil-

ity to mobile robots. The use of these techniques allows reducing the complexity of programming control strategies that allow the robot to navigate indoors and instead use computational frameworks that can determine movement instructions.

The need to develop a research work focused on the use of DRL for navigation will allow a mobile robot to provide autonomy of movement in indoor environments, specifically in environments where non-stationary elements are present. The latter is relevant for this research because the thesis will be applied as part of a research project where it is required that a mobile robot can move inside a dynamic environment , where non-stationary elements can be found, such as movement of peopleor other items . Later, the autonomous navigation algorithm will be used in other robot platforms with similar characteristics, such as the differential traction system and the sensors used for navigation. This robotics platforms are aim to be used for autonomous navigation through shopping centers and logistic warehouses, which they have similar environment conditions.

1.3 Theoretical Framework

A brief review of the main theoretical aspects for this research are presented in this subsection considering the aspects of computer science and robot navigation.

1.3.1 Computer science

Artificial intelligence is a broad area of computer science, which includes the concepts of Machine Learning as shown in Figure 1.1 [18] . Within the field of machine learning, where it is sought that machines can "learn" to make predictions or make decisions by themselves, two concepts of interest for this research stand out: Reinforcement Learning and Deep Learning.

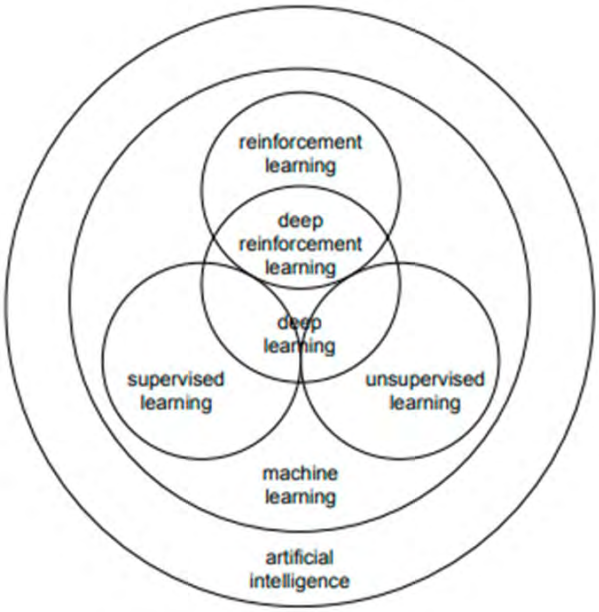

Figure 1.1: Artificial intelligence topics.

Source; [18]

The concept of Reinforcement Learning (RL) focuses on principles of "learning" using computational frameworks for decision making based on the execution of certain actions which are rewarded for a correct prediction or penalized for an incorrect one [19] . The main idea is that an artificial agent has the ability to learn through interaction with the environment, similar to what happens with a biological agent. In this way, the artificial agent, based on accumulated experience, will have the ability to optimize some objectives in the form of cumulative rewards [20]. The application of this concept can occur in any type of problem where there is sequential decision-making based on previous experiences.

The concept of Deep Learning (DL) is related to the combination of algorithms used to solve complex tasks with multi-layered neural networks through which compact low-dimensional characteristics can be automatically found for complex high-dimensional data, but with a high level of computational processing [21].

Therefore, when solving problems that involve multiple inputs (high number of dimensions), RL algorithms present complications due to the amount of calculation required to work with all dimensions [22]. For this reason, the computational capacity required to solve these algorithms is usually very high, especially when the number of analyzed inputs increases, making it difficult to define an efficient RL algorithm. On the other hand, DL provides the ability to train complex neural networks to learn the behavior of the system and the main characteristics of the input data. In this way, it is possible to solve high dimensional problems by integrating the RL and DL algorithms.

From the development of equipment with greater computational capacity, the use of Deep Learning algorithms was started in conjunction with the principles of Reinforcement Learning, beginning the development of research in Deep Reinforcement Learning (DRL). Through this new approach, it is possible to use deep neural networks to approximate the components used for reinforcement learning, among which are: agent behavior (policy), value function and the model. The DRL algorithms then comprises the components from a RL technique, such as the agent, environment, action and reward; and includes the components of a DL technique, such as policy and model (as shown in Figure 1.2).

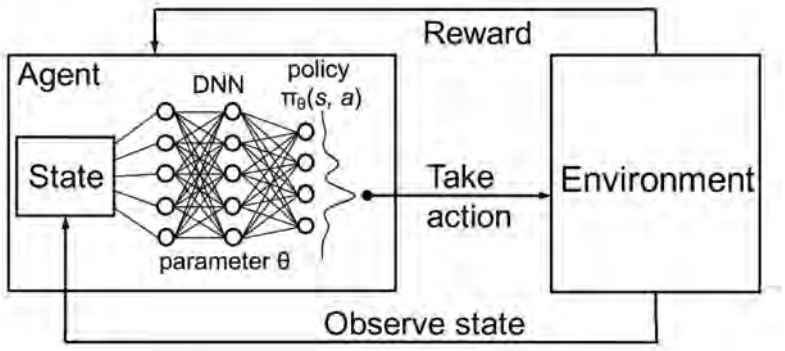

Figure 1.2: DRL components representation.

Source: Adapted from [23]

There are two different approaches to the application of DRL, which can be categorized as value-based or policy-based [22] :

- Value-based DRL: the value function is iteratively updated until an optimal value is achieved and in this way indirectly obtain the agent's policy from this optimal value.
- Policy-based DRL: makes use of the function approximation method to establish a policy network from which an optimized policy is obtained that allows to achieve an optimal reward value.

1.3.2 Robot Navigation

The navigation function within the robotics area is one of the most important because it allows the system to move through an environment safely in order to guarantee its arrival at the destination, taking into account environmental restrictions. In general, the navigation function of a mobile robot is achieved through point-to-point (P2P) movement and obstacle avoidance [22] :

- Point-to-point movement: For the development of this task, it is necessary to know the starting point and the destination point. This information is usually obtained through GPS devices, ultra-wideband location [24] or through a perspective image of the destination.
- Obstacle avoidance: The obstacles that appear can be static, dynamic or structural (continuous). This information is usually detected by distance or presence sensors, among which are lasers, ultrasounds, radars, cameras and others.

The navigation function within a robot is carried out from a traditional approach mainly by executing two tasks, localization and simultaneous mapping (known as SLAM). These tasks make it possible to generate a virtual map of the environment through which the robot is going

to move in order that using the information of this map it is possible to determine the position of the robot and to plan the path that it must follow [25] .

From a general perspective, it may not be very complex to implement a navigation function in the robot, but each aspect that this function involves turns out to be challenging and a research topic may be necessary for each of the aspects that make up if it is consider the traditional method. In Figure 1.3, the components that involve implementing robot navigation in a traditional way are presented.

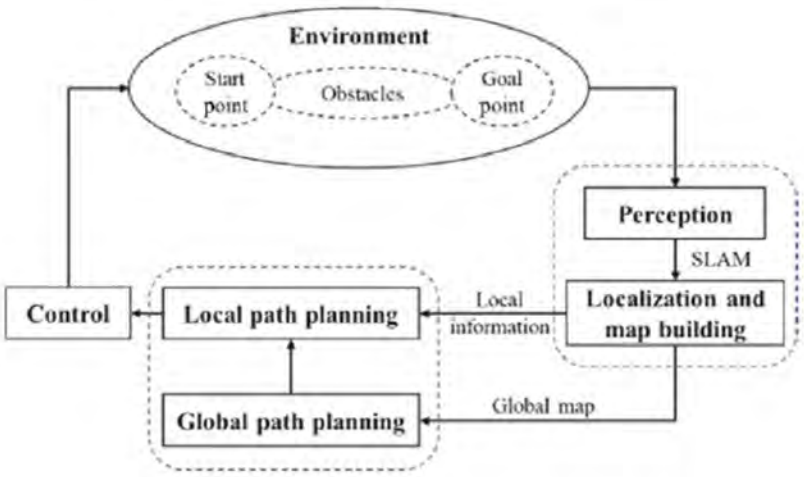

Figure 1.3: Traditional robot navigation.

Source: [22]

This diagram shows four main components:

- Environment: All obstacles (static, dynamic and structural) present in the area through which the robot is moving are considered. The starting and ending points that will be considered for planning are also taken into account.

- SLAM: The functions of identification of the surroundings or perception, location and generation of the virtual map are carried out. These activities will be performed simultaneously and repeatedly as the robot moves. SLAM is divided into visual and

laser SLAMs. The classic algorithms of visual SLAM are LSD-SLAM [26] and ORB-SLAM [27] , while the main algorithms of laser SLAM are GMapping [28] and Hector SLAM [29] .

- Planning: The environment map and local information (positions) are received as input to plan the trajectory and the path that the robot must follow. This activity is carried out globally and locally, where the first (global planning) allows determining the path to follow from the starting point to the end point, while the second (local planning) allows determining the trajectory that must be taken into account by the robot to perform the movement through the defined path. Global path uses algorithms such as A-star, ant colony optimization, and rapid exploration random tree [30] . Local path planning involves algorithms such as the Artificial Potential Field (APF) and dynamic window approach [31] .

- Control: It is in charge of executing the movement of the robot considering the type of traction system that it has and operates the actuators according to the information received from the planning module.

The implementation of the navigation function using the traditional method usually causes large computational errors or requires equipment with high computational capacity. In the absence of the correct equipment, the calculation errors accumulate during the execution of the mapping, positioning and planning tasks, causing an inaccurate movement. As previously described, this traditional method is mainly based on information from the virtual map, which is usually very sensitive to noise, which turns out to be very limiting, especially when trying to carry out applications in unknown or dynamic environments [22] .

1.3.3 Computer Science in Robot Navigation

The integration of the DRL algorithms in the autonomous navigation functions of the robot allows to determine the optimal policy capable of directing the movement of the robot from an initial position to a final position through continuous interaction with a static or dynamic environment. Different DRL algorithms, such as DQN, DDPG, PPO and their variants, have been applied to perform robot navigation [22] .

Some of the advantages of using DRL for navigation is that it does not require maps or precision sensors. On the other hand, as RL is based on trial and error principles, it is necessary to initially carry out virtual training, since during this stage collisions will inevitably be generated between the robot and obstacles. This process is carried out in simulation environments with a high level of similarity with real environments and allows defining the optimal DRL algorithms for implementation in a real robot.

The application of DRL for robot navigation modifies the traditional approach that was previously described, causing the new approach to be determined as the one presented in Figure 1.4. This diagram presents the navigation based on DRL and shows the interaction between the agent and the environment, where the agent replaces the localization process, map generation, local planning. When working with environments with simple structural obstacles (simple environments) this composition is effective and does not require more complexity, but if working with complex structural obstacles (complex environments) it is necessary to include other additional processes of traditional navigation. For these cases, an analysis of the environment is used to determine a global planning that allows having intermediate reference points that will be used by the agent to move the robot [32] .

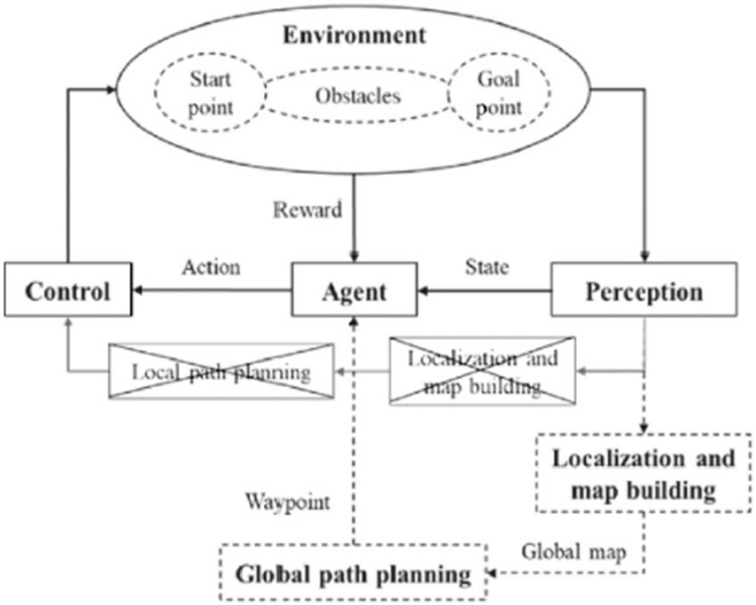

Figure 1.4: DRL-based robot navigation.

Source: [22]

The DRL-based navigation system considers as key elements the state space, the action stage and the reward function. These elements allows to determine the performance of the DRL algorithm in different application scenarios [22] .

- State space: This element mainly includes the information of the starting point, the finish line and the obstacles. Coordinates in space represent the start and destination point, while obstacles are represented by speed, position and size. In some cases, the data obtained directly from the sensors is used to represent obstacles, especially when working with lidar, ultrasonic and depth camera sensors.

- Action space: In this element, discrete movement actions (forward, backward, turn), continuous speed (linear and angular speed) and motor speed are considered. In most cases, a PID controller is used for discrete motion actions and continuous speed commands; whereas for the speed of the motor an end-to-end control is usually applied.

- Reward function: This element allows to train the RL agent by assigning positive or negative rewards when reaching the objective or colliding with obstacles, respectively. To improve training efficiency, they are considered as positive rewards when the robot reaches the goal or moves close to it; while they are considered as negative rewards for collision when you have a collision or pass very close to an obstacle and negative rewards for time passage when the robot takes more than the maximum time allowed.

1.4 State of Art

Deep reinforcement learning has been applied in different investigations related to mobile robotics for navigation issues [2–17] . Some studies have been oriented for navigation outdoors [3,9] and indoors [2,4–8,10–17] in known environments [2,4,6,7] and unknown [8,12–15] . All these investigations propose the use of different algorithms for the development of computational frameworks that will be used as the basis for the development of this work.

1.4.1 DRL Frameworks

The first DRL algorithm was developed in 2013 [31] , where they proposed a Deep Q Network (DQN) to learn how to play Atari video games from input images. Since then and making use of DRL algorithms, different methods have been developed to develop autonomous navigation frameworks making direct use of sensor information without the need for pre-processing. In this way, the DRL method determines an optimal policy that allows moving the robot to a target position through interaction with the environment. This has the advantage of not requiring maps for navigation and not depending on the precision of the sensor [22] .

The trend of applying DRL for autonomous robot navigation applications has been on the rise since 2016, due to good results reported in research [33] . Even in recent years, investi-

gations have been developed where DRL is applied for the cooperation of multiple agents [34] and for the visual navigation of artificial agents [35] . This section includes the five main frameworks applied for autonomous robot navigation as shown in Figure 1.5, which have been classified according to their approach method.

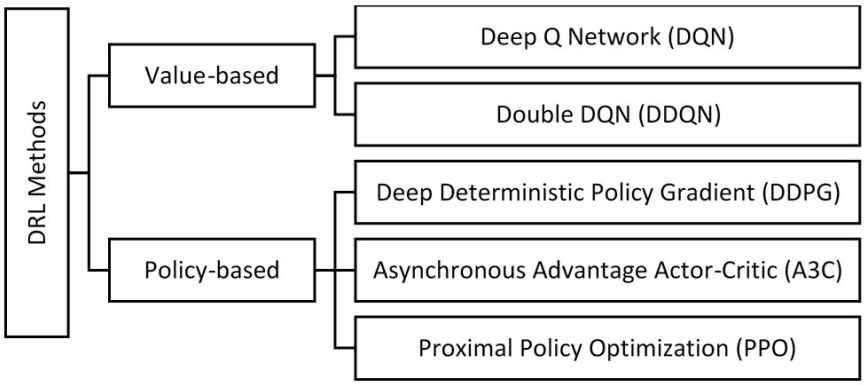

Figure 1.5: Main DRL frameworks for autonomous robot navigation.

Value- based DRL methods

A) Deep Q-network (DQN)

This method is based on the Q learning algorithm and integrates a convolutional neural network (deep neural network) that is used to represent the action value function [36] . The DQN network trains according to reward feedback. The main characteristics of the DQN are:

(1) The target network is designed to reduce the TD error in the time-varying algorithm separately. This prevents instability of the target Q network in the current Q network during training.

(2) An experience replay mechanism is used to select previously stored samples. This mechanism helps eliminate the correlation between training samples.

B) Double Deep Q-network (DDQN)

The use of DRLs became widespread since the development of DQNs, but one of its deficiencies is the overestimation of the value function [37] . Based on this, the DDQN was developed, which uses a dual network structure in the objective Q function. In this way, the first network Q defines the optimal action and the second network Q evaluates the selected optimal action. Hence, the risk of overestimation is reduced by separating the task of selecting actions and evaluating policies.

DQNN has obtained experimental results in games where results are twice as good as DQN without parameter adjustment, and up to three times better with parameter adjustments [22] .

Policy-based DRL methods

A) Deep Deterministic Policy Gradient (DDPG)

Value-based methods are used in scenarios with a high-dimensional observation space, but they have been designed to solve discrete low-dimensional spaces. Given this, the action spaces are usually discretized, but by doing this, the number of actions is increased by increasing the degrees of freedom [22] .

Given this, the DDPG method was proposed, which is designed to optimize policies for continuous action spaces using a deterministic policy function [38] . This method also uses a convolutional neural network and learns from the repetition of the experience and the DQN in order to stabilize the training and obtain a high efficiency in the use of the sample.

DDPG has demonstrated good experimental results in robotic arm control applications where more than 20 continuous control tasks have been solved.

B) Asynchronous Advantage Actor-Critic (A3C):

With the policy gradient method, data collection is often challenging and introduces large variations. For this reason, the critical actor method combines the value function with the policy gradient method, for the second to select the actions and the first to evaluate them, generating the A3C algorithm [39] . During the training it is sought to reduce the difficulty of data collection and the data variation generated by the policy gradient algorithm, so the actor and critic parameters are updated alternately.

Based on the AC structure, the A3C algorithm incorporates parallel agents, allowing the parameters of the main structure to be updated simultaneously in these parallel environments. Additionally, the A3C critical value function is updated based on the cumulative performance of multiple steps, allowing you to improve the speed of convergence and propagation during updates. An important feature is that A3C can run on a multi-core CPU, having a lower computational cost than DQN.

A3C has demonstrated successful experimental results in robot arm continuous control and maze navigation tasks. However, these experiments have also detected hyper parameter adjustment problems and low sampling efficiency.

C) Proximal Policy Optimization:

The PPO method was implemented to perform multiple minibatch update times, which, unlike traditional policy gradient methods, improves the efficiency of sample utilization [40] . This algorithm makes use of an alternate target to optimize the new policy using the old policy.

The PPO experimental results have been shown to perform better than the A3C and DDPG because it has a balance between sample complexity, time effectiveness, and simplicity.

1.4.2 Navigation Scenario

Over the last years, different applied DRL studies have been developed for autonomous navigation. Researchers use different classifications, such as "maples navigation" when a global map is not required and relative destination positions are obtained using GPS or Wi-Fi sensors; “map-based navigation” when a sensor-generated local map is required to define movement actions; and “visual navigation” when using a visual sensor type (RGB camera).

Regardless of the type of classification, multiple investigations have been developed in which different approaches are proposed, but which ultimately seek the same objective. The important thing to keep in mind when defining a DRL navigation policy is that if it is very complex, the training difficulty is greater, so it is usually work with specific scenes of less complexity, which are then generalized to scenes of greater complexity.

Based on the investigations found on DRL applied to autonomous navigation [14, 41–46, 46–61] , Table 1.1 presents a classification of these techniques according to their navigation scenario. These three scenarios will be the starting point for the development of this research since they cover the main problems to be solved for navigating a robot in indoor environments with non-stationary elements.

Table 1.1: Comparison of DRL-based navigation scenarios.

Source: Adapted from [22]

Navigation Scenario	Static Obstacle	Dynamic Obstacle	Structure Obstacle	Obstacle Scale	Obstacle Velocity	Randomness
Local obstacle avoidance	Y	Y	N	Low	Low	-
Indoor navigation	Y	N	Y	Low	-	-
Social navigation	Y	Y	N	High	High	High

Local obstacle avoidance

Local obstacle avoidance is the most common navigation application with DRL. Traditional navigation methods required a specialized sensor system combined with algorithms for processing and decision-making. The use of DRL avoids this level of complexity and instead the neural networks are in charge of processing the data obtained by the sensors to determine an action on the system.

Dugleana and Mogan [62] carried one of the first successful investigations out in 2016. They used a Pose-Net and Q leaning neural network, through which they managed to evade effectively static and dynamic obstacles. From this point, new DRL applications for navigation began to be developed, among which use of algorithms Deep Double Q Network (DDQN), Deep Deterministic Policy Gradient (DDPG), Fast Rent Deterministic Policy Gradient (Fast-RDPG), Asynchronous Advantage Actor-Critic (A3C), and other variations of the previous mentioned. As a summary, Table 1.2 presents a review of the investigations that use DRL navigation focused on local obstacle avoidance.

Indoor navigation

Indoor navigation covers cases in which the robot must move in an environment with several rooms or in the form of a 3D maze with several walls and corridors. The difference with the scenarios used in local obstacle avoidance is that in these cases the structures tend to be more complex. These algorithms originated from a maze navigation game, which was established at the DeepMind Lab [50] .

3D mazes are often used as test platforms for DRL algorithm training. Learning in these environments consists of training an agent by learning a policy that maximizes reward while navigating through the maze. For this, visual navigation is usually applied, where the objective is determined from an RGB image [17] , unlike local obstacle avoidance, which requires the

Table 1.2: Comparison of references for local obstacle avoidance.

Source: Adapted from [22]

Ref.	Algorithm	Perception type	Action space	Implement. type	Year
[41]	Uncertainty-aware RL	Monocular camera	2D Continuous	Sim + Real	2017
[42]	DDQN	Laser Range Finder	3D Discrete	Sim + Real	2017
[14]	ADDPG (Asynchronous DDPG)	Laser Range Finder	2D Continuous	Sim + Real	2017
[43]	SF-RL (Successor Feature RL)	Depth Camera	4D Discrete	Sim + Real	2017
[44]	DDQN	Laser Range Finder	8D Discrete	Sim + Real	2018
[45]	IL (Imitation Learning) + CPO	Laser Range Finder	2D Continuous	Sim + Real	2018
[46]	SAC (Soft AC)	Depth Camera	2D Continuous	Sim + Real	2019
[47]	DDPG	Laser Range Finder	2D Continuous	Simulation	2020
[48]	ICM A3C (Intrinstic Curiosity)	Laser Range Finder / Monocular camera	Discrete	Sim + Real	2020
[49]	DDQN	Monocular camera	3D Discrete	Sim + Real	2020

coordinates of the destination point.

Among the previous works, one of the first was developed in 2016 by Oh et al. [51] where a DRL algorithm was proposed to navigate in the virtual environment of the video game. This algorithm makes use of images to navigate through complex scenarios, but whose objective is fixed. Another of the initial investigations proposes the use of a global map to find the shortest path and from this plan navigation actions. In this case, retraining is required when modifying the objective. As a summary, Table 1.3 presents a review of other relevant investigations that use DRL navigation focused on indoos navigation.

Table 1.3: Comparison of references for indoor navigation.

Source: Adapted from [22]

Ref.	Algorithm	Perception type	Action space	Implement. type	Year
[50]	Nav A3C (A3C + LSTM)	Monocular camera	8D Discrete	Simulation	2017
[17]	AI2-THOR (Deep siamese AC network)	Monocular camera	4D Discrete	Sim + Real	2017
[53]	LSTM + DRL	Monocular camera	3D Discrete	Sim + Real	2018
[54]	A2CAT-VN	Monocular camera	8D Discrete	Simulation	2019
[55]	IMPALA	Monocular camera	3D Discrete	Sim + Real	2020
[56]	HISNav framework	Depth Camera	-	Sim + Real	2020
[57]	AppoNav (LSTM+PPO)	Monocular camera	8D Discrete	Simulation	2020
[47]	DDPG	Laser Range Finder	2D Continuous	Simulation	2020
[48]	ICM A3C (Intrinstic Curiosity)	Laser Range Finder / Monocular camera	Discrete	Sim + Real	2020
[49]	DDQN	Monocular camera	3D Discrete	Sim + Real	2020

Social Navigation

Social browsing refers to the navigation of a robot in environments with a high number of people that can also be represented as dynamic elements. In these cases, navigation through these scenarios is sought while dynamic obstacle avoidance is performed more frequently than in the first scenario described. Furthermore, the nature of people's movement is usually random, making it difficult to quantify or model, so training in simulated environments presents wide differences with real environments [22] .

In 2017, Chen et al. [57] proposed adapted the CADRL algorithm for social navigation using laser sensors, depth cameras, and RGB cameras installed in a differential traction robot. In order to model the mobility behavior of pedestrians, a bias was integrated into a known model. In this way, the RL training process obtained better performance at the time of the validation tests in

real environments. In other research developed by Sun et al. [58] proposed the application of a higher penalty according to the probability of collision between the agent and the dynamic elements.

In more recent research, Sathyamoorthy et al. [59] the problem of robot freezing is studied, which is generated in very dense environments. They proposed a hybrid approach based on DRL to solve this problem, using a variation of the A3C method. As a summary, Table 1.4 presents a review some relevant investigations that use DRL navigation for social navigation.

Table 1.4: Comparison of references for social navigation.

Source: Adapted from [22]

Ref.	Algorithm	Perception type	Action space	Implement. type	Year
[58]	SA-CADRL	Agent level data	2D Continuous	Sim + Real	2017
[59]	GA3C-CADRL	Agent level data	11D Discrete	Sim + Real	2018
[60]	A3C	Agent level data	Discrete	Sim + Real	2019
[46]	PPO + LSTM + collision prediction	Agent level data	2D Continuous	Simulation	2019
[61]	Frozone + DRL	Agent level data	2D Continuous	Sim + Real	2020

1.4.3 Sensors

As presented in the studies presented in Table 1.2 and Table 1.3, it has been observed that three types of sensors are used: monocular cameras, depth cameras and laser range finder. In most of these studies where DRL algorithms are applied, the use of the same sensors during validation as in the training stage is suggested, otherwise the system may fail [22] . One of the main drawbacks with these systems is the possibility that some of the sensors used for navigation are prone to failure. Given this, fault-tolerant navigation policies have been developed [46] , seeking to have robustness, practicality and scalability.

Another major drawback related to sensors is the limited field of view (FOV). For this, short and long-term memory (LSTM) agents have been used in conjunction with a local map critic (LMC) to store the information collected by the sensor with limited range and complement it with previous information [62]. In addition, this method improves the performance of the DRL agent in the real world by making and using dynamic randomization, in this way it seeks to give robustness to the system against possible changes in the characteristics of the sensor. In other studies, to eliminate the problem of limited FOV, they work with 2D point clouds of the agent incorporating the measurement angle as part of the observation [61] .

1.4.4 Robotics platform

There is a wide variety of robotic platforms with differential traction used for research and development of autonomous navigation applications, which vary in costs depending on the technologies and equipment they include. Among the most used are those that have the ability to modify their equipment and that have open source operating systems. The most widely used differential robots for different research on autonomous navigation are the Turtlebots [13–15, 47, 63] , which can be achieved in different configurations. In addition, there are other robots such as the Dingo-D, designed for research development applications, but for robust applications. Table 1.5 presents a summary of the main characteristics of these differential robots, which will be taken as a reference for the robot that will be developed and used in this research.

The robotic platform that will be used for the validation process have been design as shown in Figure 1.6. This platform in composed of a V-Slots aluminum chasis that can bear up to 50 Kg. The robot uses a differential traction system, which includes two DC motors with a gearbox, and an encoder aligned parallel to each other facing different directions. The main controller of the robot is a Jetson TX2, which will be used to process the DRL algorithms, process the data from the lidar sensor and depth camera, and control the direction of the robot.

Table 1.5: Comparison of differential robots for autonomous robots research and applications.

Source: Turtlebot and Clearpath

Characteristics	Turtlebot3 Burguer	Turtlebot3 Waffle	Clearpath Dingo D
Size	13 x 17 x 19 cm	28 x 30 x 14 cm	55 x 51 x 11 cm
Actuators	Servomotor	Servomotor	DC motor
Chasis material	Plastic	Plastic	Aluminum
Sensors	360° Lidar / IMU	360° Lidar / IMU / Camera	3D Lidar / IMU / Depth Camera
Controller	Raspberry Pi 3	Raspberry Pi 3	Jestson Developer Kit
Operative system	ROS	ROS	ROS
Price	∼500 USD	∼1,500 USD	∼11,500 USD

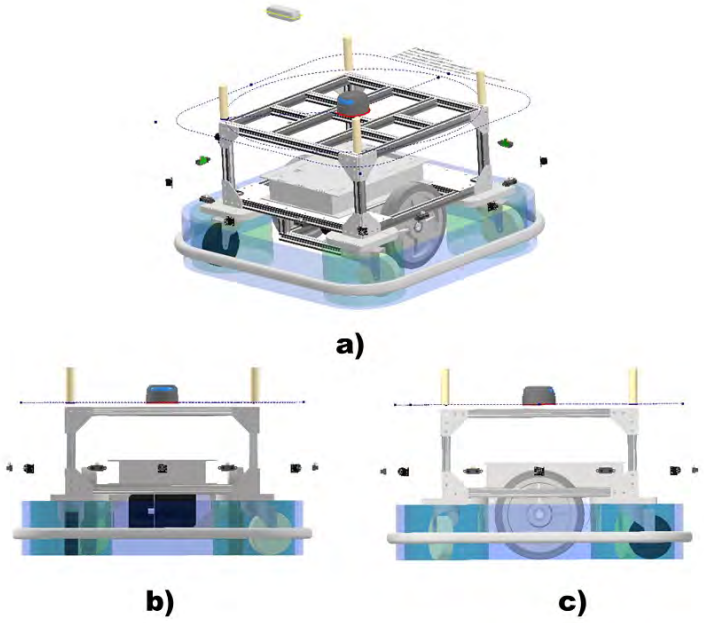

Figure 1.6: Robotic Platform designed for validation stage. a) Isometric view, b) Front view, and c) Lateral view.

1.5 Objectives and Scope

General objective

The general objective is to provide autonomous navigation capacity to a ground mobile robot in indoor environments, where non-stationary elements are present, through the study and application of Deep Reinforcement Learning techniques for a robotic platform with differential traction equipped with a Lidar depth camera and laser range finder. Moreover the evaluation and comparison between DRL-based algorithms and traditional algorithms for autonomous navigation will be performed.

Specific objectives

- Review of Traditional and Deep Reinforcement Learning based techniques used for autonomous robot navigation indoors
- Develop frameworks based on Traditional and Deep Learning by Reinforcement techniques for autonomous robot navigation indoors in indoor environments where non stationary elements are present.
- Conditioning a robotic platform with differential traction and Lidar depth camera for indoor navigation
- Evaluate the frameworks on a virtual environment using the rootic platforms to determine their performance with non-stationary elements.

Scope

Traditional and Deep Reinforcement Learning techniques used for autonomous robot navigation indoors will be studied and evaluated. To evaluate the frameworks based on the navigation

techniques, virtual platforms (ROS type) will be used. To validate the frameworks, a differential robotic platform with a Lidar depth camera will be used in indoor environments where non-stationary elements (people, mobile objects) are present, using a virtual environment similar to a real scenario.

1.6 Hypothesis

It is accepted that the use of Deep Reinforcement Learning techniques allows the generation of computational frameworks to provide the autonomous navigation capacity to indoor differential robots that have non-stationary elements using a Lidar depth camera.

1.7 Area of Specialization

This work focus in the area of computer science and the area of control and automation. In the first area, emphasis will be placed on intelligent systems, since the concepts of artificial intelligence and machine learning will be applied to provide autonomy capacity to real systems. In the second area, robotics topics will be developed combined with intelligent systems techniques, in order to provide autonomy to certain types of robots.

The research lines defined by the Pontificia Universidad Católica del Perú that are covered in this thesis work are listed below:

010-01-01 Computational Science - Intelligent Systems 020-01-02 Control and automation - Robotics and automatic control 020-01-05 Control and automation - Intelligent systems

1.8 Methodology

The methodology to be followed for the investigation will be of the quantitative type because parameters will be measured to determine the performance of the computational frameworks developed in the investigation. The following activities will be developed during the development of this thesis work:

I. Review of Reinforcement Deep Learning techniques used for autonomous robot navigation indoors

   i. Review of information regarding Deep Learning, Reinforcement Learning and Reinforcement Deep Learning

   ii. Review of Deep Learning techniques for autonomous navigation of land mobile robots

   iii. Review of Reinforcement Learning techniques for autonomous navigation of land mobile robots

   iv. Review of deep learning techniques for navigation in indoor environments with non-stationary elements

   v. Review of Reinforcement Learning techniques for navigation in indoor environments with non-stationary elements

II. Development of frameworks based on Deep Learning by Reinforcement techniques for autonomous navigation in indoor environments with non-stationary elements

   i. Research for computational frameworks for autonomous navigation of terrestrial robots

   ii. Research for computational frameworks for navigation in indoor environments with non-stationary elements

iii. Analysis of computational frameworks based on Deep Learning by Reinforcement techniques

iv. Development of computational frameworks for autonomous robot navigation indoors in indoor environments where non-stationary elements are present

III. Conditioning of a robotic platform with differential traction and integration of a Lidar depth camera for indoor navigation

i. Definition of elements and design of robotic platform with differential traction

ii. Definition of Lidar depth camera for environment identification

iii. Definition of processing unit to execute computational model

iv. Integration and testing in virtual environment of depth camera, robotic platform and processing unit

IV. Evaluation of the frameworks on virtual platforms to determine their performance in indoor environments with non-stationary elements

i. Review and definition of virtual platforms for simulation of mobile robot navigation

ii. Programming of computational frameworks for virtual platform tests

iii. Performance analysis of computational frameworks

1.9 Challenges and Opportunities

1.9.1 Challenges

Despite the several investigations carried out over the last few years, the application of DRL algorithms into autonomous navigation tasks faces some challenges:

I. Partial observation

When working with systems whose State Space is fully observable, RL algorithms have performed satisfactorily for motion control applications, such as control of manipulators. However, for mobile robots autonomous navigation, it can no longer be considered that there is a completely observable State Space, but partially observable, due to the uncertainty of the state of the other elements. This causes that the state cannot be distinguished by individual observations and that the agent's learning strategies are not optimal. A possible solution has been given by adding observations with multiple inputs [45, 54] , also by adding rewards and previous actions [14, 43, 50] . This strategy is called Expansion of network input. Another alternative to solve this problem has been through Addition of Memory Ability. For this, Recurrent Neural Networks (RNNs) have been integrated with DRL algorithms. Because RNNs have long-term dependency problems, several layers of LSTM have been used in research [50, 53, 59] .

II. Sparse reward

Because the agent's reward allocation is assign in each training epoch, the reward ends up being very sparse as it is only considered success on reaching a target or failure on colliding with an object. Low rewards do not usually occur with simple browsing environments, but when working with environments that are more complex and with dynamic elements, this probability increases. In this way, the low reward effect leads to prolonged training times or poor training. This problem has been addressed through three alternatives: reward shaping, auxiliary task and curriculum learning. Reward shaping consists of the manual design of dense rewards to facilitate the algorithm learning process. This alternative is commonly used in systems where they can be validated in real environments. This technique seeks to assign rewards related to the distance to the target [43] , the travel time [45, 46, 56] and the direction during each action [64] . Likewise, in order to determine the reward and the architecture automatically, the AutoRL

algorithm has been used [65] . The second alternative proposes the use of auxiliary tasks in such a way that additional pseudo rewards can be assigned. In this way, the original task is integrated and the auxiliaries are used to generate the representation of the model. Work has been carried out that includes auxiliary tasks to algorithms Nav A3C [50] , and A2C [54] . As a third alternative, there is Curriculum learning, which consists of progressively increasing complexity during the learning stage. In this way, training time is reduced starting with simple tasks until reaching more complex scenarios [66, 67] .

III. Poor generalization

DRL algorithms training are usually carried out in simulated environments, since training them in real environments would be very dangerous and time-consuming. For this reason, the training process is usually carried out in different simulation environments, but since the distribution is different in each environment, it is not easy to transfer the model trained in one environment to another. This problem also occurs during the application in a real environment since the simulation environments turn out to be less complex and dynamic than the real ones. One approach to solve this problem is with expansion of the Sample Space, for which noise is added in measurements, positions, obstacles and other elements of the stage. In addition, according to previous research experience, training in different environments is recommended, thus achieving transfer to real environments without the need for additional training [68, 69] . Another alternative is the State Space Reduction, a technique through which it is sought to use sensors with a low level of detail [14, 43] , such as laser sensors, or to reduce the amount of information from other sensors, such as depth cameras, to work with information in 2D [44] . In this way, the training time required by expanding the Sample Space is reduced, and the results when transferring the algorithm to real environments presents successful results.

1.9.2 Opportunities

The challenges previously described have been addressed using different algorithms and strategies in research in recent years. These investigations have made it possible to know the advantages that each algorithm provides for specific problems and based on this, solutions have been developed for specific applications. For this research,

the last challenge will be comprehensively covered by evaluating the performance of the DRL algorithms during a simulation stage and later during validations in real environments.

As part of the work, it is expected to carry out a comparative analysis of at least two DRL based algorithms and two traditional algorithms taking into account the problems described and determine the one that has the best performance for the specific application of autonomous navigation indoors with dynamic elements.

Since this thesis is included within a research project where it is required that a robot can navigate autonomously within an indoor environment with non-stationary elements, the solution will be addressed taking as reference works based on three different types of scenarios: i) Local obstacle avoidance, ii) Indoor navigation and iii) Social navigation.

1.10 Expected Contributions

This work is focused on the intelligent control for the autonomous navigation of differential mobile robots in indoor environments with non-stationary elements by learning from sensory information from the environment. Intelligent robot control is expected to reduce the complexity of programming conventional control algorithms for autonomous navigation applications. For this reason, it is expected to obtain three types of contributions from the research, development and application aspects:

I. Research: Review of the different DRL algorithms developed in recent years for au-

tonomous navigation applications, and specifically for the control of differential robots for indoor navigation with non-stationary elements. Within this research, the algorithms developed for the aspects of environment identification, location, trajectory generation and direction control will be taken into account. This collected information will be used to prepare a literature review article, which may be useful for future research focused on the use of DRL for autonomous navigation of interior robots.

II. Development and simulation: A computational model based on DRL algorithms with greater potential for this type of application will be developed. This algorithm will be validated through a virtual platform where its use and effectiveness will be simulated and it will be compared with other existing algorithms. For this stage, it is expected to contribute by defining a methodology that allows simulating the operation of computational frameworks developed for differential mobile robots and in turn comparing the effectiveness in different robotic platforms.

III. Application: The algorithm developed and validated will be applied in a robotic platform developed for tele-presence care in dynamic scenarios. Using the computational model, it is expected to be able to provide the robot with autonomous navigation capacity with the information obtained by a Lidar depth camera. If good results are obtained, it is expected to contribute to the development of an autonomous robot capable of moving in indoor environments with non-stationary elements, which can be replicated in other environments with similar characteristics, such as warehouses, malls, hospitals, workshops, laboratories, offices, etc.

Chapter II

Review of autonomous navigation algorithms for indoor environments

This chapter provides a detailed review of autonomous navigation algorithms for robots operating in indoor environments, presenting a comprehensive examination of both traditional and contemporary approaches. The discussion begins with traditional autonomous navigation, highlighting key methodologies such as Simultaneous Localization and Mapping (SLAM), global path planning, and local path planning, and concludes with a comparison of local planners to identify the strengths and limitations of each approach. The chapter then transitions to explore autonomous navigation algorithms using reinforcement learning (RL), beginning with a conceptual framework on reinforcement learning and deep reinforcement learning (DRL). It proceeds to classify various DRL algorithms and reviews DRL techniques for navigation in indoor environments, distinguishing between scenarios with known and unknown maps. The chapter also includes an in-depth review of local planners within the context of DRL, providing a critical analysis of the state-of-the-art in autonomous navigation and highlighting future research directions.

2.1 Traditional Autonomous Navigation

2.1.1 Simultaneous Localization and Mapping (SLAM)

Global positioning sensors (GPS) are supposed to represent a solution to address the localization problem. However, current GPS networks provide an accuracy of several meters and are not suitable for indoor or obstructed areas, which is unsatisfactory for robust autonomous navigation. Likewise, GPS provides information on the absolute position with respect to the earth's reference system, which does not completely involve the location task, since it is just as relevant to know the relative position with respect to objects in a certain area [70].

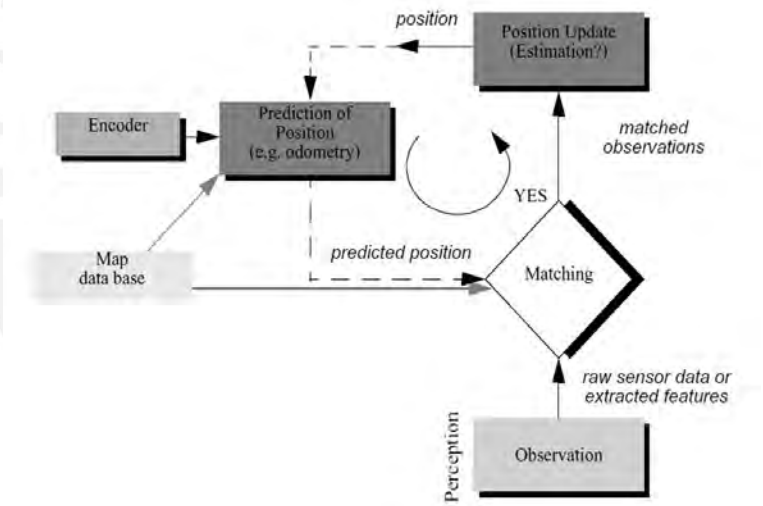

Figure 2.1: Location scheme in mobile robots.

Source: [70]

In this sense, the localization problem implies the need to acquire information about the environment, either through a map, to subsequently determine the precise position of the robot in said environment. In this context, the simultaneous localization and mapping (SLAM) technique uses information from sensors to capture information from the environment and combine it using algorithms, allowing you to create a map and locate the robot at the same time [71]. In

Figure 2.1, the main stages of localization are illustrated, in which the use of information from a map, encoders and perception sensors can be highlighted, which, through additional algorithms, are combined to predict the location. robot position.

In recent years, there have been significant advances in visual SLAM techniques. However, they still present impediments to reliable localization due to their difficulties in textureless environments. On the other hand, SLAM based on LiDAR provides a more robust localization since the 3D information is captured directly from a point cloud [72]. Therefore, below, the main algorithms based on the use of laser sensors that solve the task of simultaneous localization and mapping will be explained.

Gmapping

Gmapping improves the Rao–Blackwellized Particle Filter (RBPF), where each particle represents a possible position of the robot, allowing for the construction of the map and random distribution in the environment [73]. These particles propagate based on the robot's movement and, subsequently, update the map and particle weights according to the similarity between their distribution and sensor measurements. One advantage of Gmapping over the RBPF algorithm is the reduction in computational cost achieved by decreasing the number of particles in each iteration [74].

To accomplish this, Gmapping employs adaptive sampling, using Equation (2.1), where N represents the number of particles, w^i denotes the weight associated with particle i and $N_{\text{threshold}}$ is the threshold used for sampling:

$$N_{\text{threshold}} = \frac{1}{\sum_{i=1}^{N}(w^i)^2} \tag{2.1}$$

The value of $N_{\text{threshold}}$ measures the dispersion of the particle set. In other words, a larger value indicates a more concentrated particle set and lower estimation error [72]. Therefore,

when $N_{\text{threshold}} < \frac{N}{2}$, the sampling is performed, and particles with low weight are replaced by samples with higher weight.

Furthermore, it is worth mentioning that using only LiDAR information for location can generate inaccuracies in certain situations where there are few characteristics of the region, such as an open place. Figure 2.2 illustrates this, where there is greater uncertainty when using the laser only if it is compared to the algorithm also using odometry, since the particle distribution of the latter is more concentrated [74]. Consequently, alignment errors occur in the mapping when odometry is not used, as can be seen in Figure 2.3.

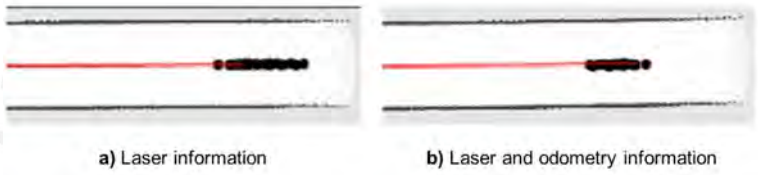

Figure 2.2: Comparison of effects on particle distribution.

Source: [74]

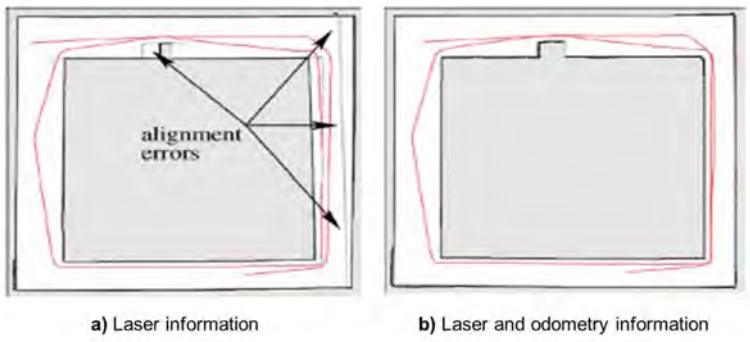

Figure 2.3: Comparison of mapping results.

Source: [74]

HectorSLAM

This technique allows for Simultaneous Localization and Mapping, not only in ground vehicles but also in aerial ones. Additionally, it presents low computational cost, making it suitable for low-resource processors. The algorithm consists of two main subsystems, one related to the 3D navigation system and the other focused on the 2D SLAM. The first subsystem uses inertial information from an IMU to provide an orientation estimation to the 2D-SLAM subsystem [75]. It is worth noting that, unlike other SLAM techniques, it requires synchronization between both subsystems through a real-time critical controller [72].

The 2D-SLAM subsystem uses scan matching as its fundamental component, which is a process for finding correspondences between the sensor-scanned information and the environment information provided by the map. For this purpose, it employs Equation (2.2), through which it finds the orientation and translation of the laser beam's endpoints, allowing alignment of one measurement with another and with the map created up to that point [75].

$$\varepsilon^* = \arg\min_{\varepsilon} \sum_{i=1}^{n} [1 - M(\varepsilon)]^2, \tag{2.2}$$

where ε represents the transformation, i.e., the orientation and translation, while ε^* corresponds to the transformation that generates the best alignment of the laser scan with the map and, therefore, minimizes the sum of the scan error in the formula [72]. On the other hand, $M(\varepsilon)$ is the transformation from a global coordinate system to one relative to the map.

The 3D navigation subsystem consists of a fusion of sensors that, through an Extended Kalman Filter, can estimate the 3D position and orientation of the robot. This information is projected on a 2D plane and, as seen in Figure 2.4, enters the SLAM subsystem for Scan Matching. Furthermore, in the opposite direction, the information estimated by the SLAM allows improving the estimation of the navigation subsystem [75]. In this context, the Extended Kalman Filter (EKF) is an extension of the Kalman filter for nonlinear systems that has as one

of its applications the improvement of the measured parameters through sensor fusion. The first step is the prediction phase and consists of an estimate of the state, while the second correction phase uses measurements to update the measurement of said state [76].

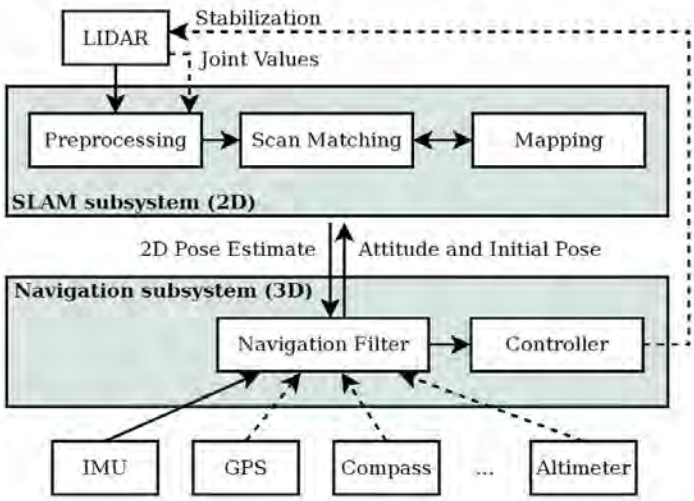

Figure 2.4: HectorSLAM algorithm architecture.

Source: [75]

Karto-SLAM

Karto-SLAM is a Simultaneous-Localization-and-Mapping algorithm based on graph optimization. Figure 2.5 illustrates the task that this method solves, in which it calculates all the states x taking into account the time range [72]. Thus, for the calculation of the latter, the control signals u and the previous x states are taken into consideration. In that sense, the main difference with respect to the particle filter is that the latter estimates only the state that is optimal at the current time x_t, while the former estimates a global configuration.

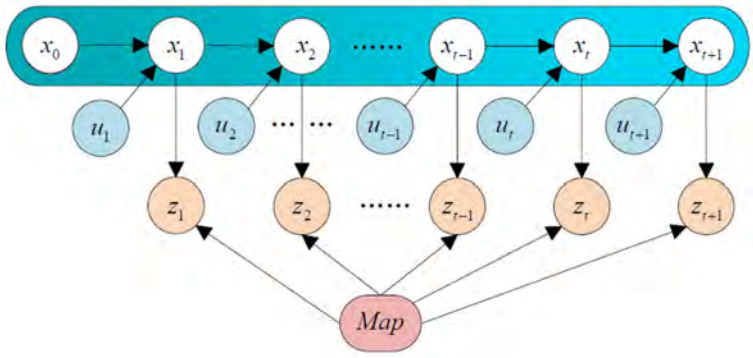

Figure 2.5: Graph optimization.

Source: [72]

In the context of Karto-SLAM, the graph optimization uses as nodes the robot positions and is connected by constraints that are obtained from sensor observations [77]. In that sense, the resulting graph is large and, therefore, the computational cost increases. In view of this, Karto-SLAM employs a method called Sparse Pose Adjustment (SPA), which allows fast convergence even in complex environments. Moreover, when a scan-matching system is added to the SPA, a real-time mapping system can be obtained, as the graph-based approach allows for the addition and removal of details to the map without any problem [77].

Comparison of SLAM algorithms

In Table 2.1 it can be seen the comparison of the main mapping and simultaneous localization algorithms. Where it should be noted that aspects such as localization performance were not included since these three algorithms were chosen considering those with the best results. Likewise, Karto-SLAM, being a graph-based method, is the only one that for the Scan Matching technique analyzes a scan-scan correspondence in addition to the scan-map correspondence, which is used by the other algorithms. On the other hand, it is observed that the Gmapping algorithm requires greater computational cost. While HectorSLAM is the only one that uses a

fusion of sensors between LiDAR and IMU and also allows position estimation 3D. Likewise, comparative studies have been carried out on the computational cost and accuracy of the map of the three algorithms [78], which are detailed in Table 2.1. Where it can be highlighted that the Gmapping algorithm is the one that requires the best computational load. While, Karto-SLAM presents greater precision when mapping.

Table 2.1: Comparison of SLAM algorithms

Method	**Gmapping**	**HectorSLAM**	**Karto-SLAM**
Algorithm	Particle Filter	Extended Kalman Filter	Graph Optimization
Sensors	Laser	Laser+IMU	Laser
Position Estimation	2D	3D	2D
Scan Matching Method	Scan-to-Map	Scan-to-Map	Scan-to-Map y Scan-to-Scan
Precision Error	15.00	17.53	4.27
Computacional Load (% CPU Usage)	52.01	15.31	29.23

2.1.2 Global Path Planning

Nowadays optimal path finding is a widely studied topic with varied applications. For example, app or web maps provide optimal routes to destinations that are entered by a user. To do this, this service requires first converting the map to a graph, as can be seen in Figure 2.6. From this, algorithms are used that allow finding optimal solutions to planning problems in specific environments. These take into consideration direct information from the environment through sensors and for optimization consider different aspects such as execution time, route distance, obstacle avoidance, among others. In this sense, the main algorithms used to solve this task will be detailed below.

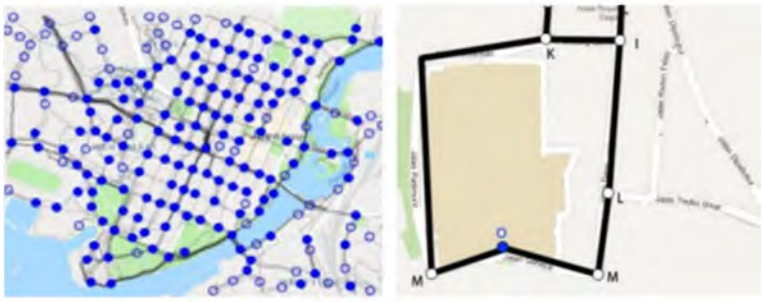

Figure 2.6: Web Maps.

Source: [79]

Artificial Potential Fields

The potential-fields method has as its main advantage its speed and effectiveness in solving the global-planning problem. This method suggests considering an initial approximation of the route to follow, in order to avoid stagnation at a local minimum, and then correcting it using the potential fields until an optimal solution is found [80]. Figure 2.7 illustrates the main idea on which this method is based, where the obstacles correspond to repulsive fields and the goal to attractive ones, therefore, the environment results from the vector sum of these fields.

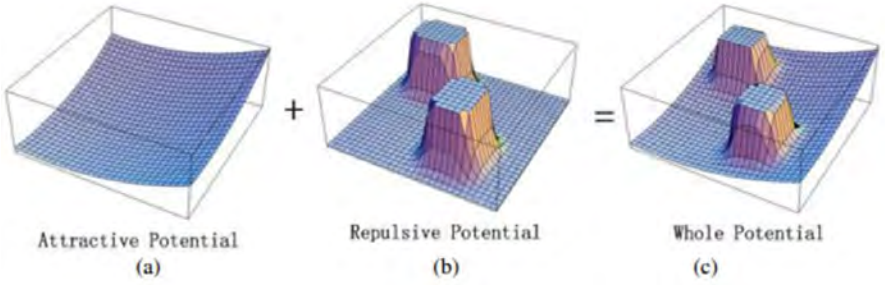

Figure 2.7: Attraction and repulsion fields.

Source: [81]

Using this method, the robot must move in the direction in which the potential field decreases, i.e., in the direction of the field gradient evaluated at the robot's position. The above

is formulated by Equation (2.3) [82], where an additional force F is introduced that drives the robot to follow the direction in which the potential field decreases:

$$\vec{F}(q) = -\nabla U(q) = -(\nabla U_{\text{atr}}(q) + \nabla U_{\text{rep}}(q)), \tag{2.3}$$

where U represents the resulting potential field and q the current position of the robot. From the above, path planning is achieved by defining a unit vector pointing to the gradient and a parameter δ defining the robot's feed size [82], as illustrated in Equation (2.4):

$$q_{i+1} = q_i + \delta_i \frac{\vec{F}(q)}{\|\vec{F}(q)\|}. \tag{2.4}$$

Dijkstra

This method is based on a greedy algorithm, due to its approach of making locally optimal decisions in each iteration to find the shortest global path in a weighted graph. Although greedy algorithms present an optimal solution in many cases, it is relevant to consider that they cannot always find the answer. Since, for example, some problems require considering long-term decisions.

The logic of this algorithm is detailed in Algorithm 1, where first, an infinite distance is assigned to all nodes except the source node, which is assigned a distance of 0. Then, the node is selected. visited with the shortest distance and is marked as visited. Next, the distances of adjacent nodes are updated, adding the distance of the selected node and the weight of the edge connecting them, only if this accumulated distance is less than the distance currently stored for the adjacent node. These selection and update steps are repeated until all nodes have been visited or until there are no nodes with finite distances left. Finally, the shortest path from the source node to any other node is obtained following the updated distances.

Algorithm 1 Dijkstra's algorithm [83]

```
Initialize distance to initial node to 0
Set other distances to other nodes to infinity
Initialize list of visited nodes S to be empty
Initialize queue with all nodes Q
while queue is not empty do
    Select the element of Q with the smallest distance
    Add element to list of visited nodes S
    for all neighboring elements of the element do
        if new smallest distance was found then
            Update minimum distance
        end
    end
end
```

A Star (A*)

This technique is based on the Dijkstra algorithm, which is detailed in the pseudocode Algorithm 1. First of all, an infinite distance is assigned to all nodes except the origin node, which is assigned a distance of 0. Then, the unvisited node with the shortest distance is selected and marked as visited. Furthermore, the distances of adjacent nodes are updated by summing the distance of the selected node and the weight of the connecting edge, only if this accumulated distance is less than the currently stored distance for the adjacent node. These selection and update steps are repeated until all nodes have been visited or until there are no nodes with finite distances left. Finally, the shortest path from the origin node to any other node is obtained by following the updated distances.

On the other hand, A* differs from Dijkstra's algorithm in the node-estimation function, as the latter finds the optimal route considering only the cumulative cost, whereas A* takes into account a heuristic estimation that allows for solving the problem more efficiently. For the cost estimation using the heuristic function, first the cost $c_{(\text{start,i})}$ between the initial node and a node i is found; subsequently, the nearest neighbor j is searched and the distance $d_{(\text{i,j})}$ between them

is estimated; and, finally, the distance $d_{\text{(j,goal)}}$ between node j and the node corresponding to the goal is estimated [79]. The total cost takes into account these three parameters, as can be seen in Equation (2.5):

$$C_i = c_{\text{start,i}} + \min_j(d_{i,j} + d_{j,\text{goal}}). \tag{2.5}$$

2.1.3 Local Path Planning

Global planning methods pose a problem when the map of the environment is unavailable or inaccurate, which is precisely the situation for most autonomous navigation problems [84]. Given this, local planners solve this problem, as they are capable of quickly adapting to variations in the environment in real time. Additionally, because their focus is limited to a portion of the environment, they require fewer computational resources, allowing them to update the world at an appropriate frequency as measured by sensors.

Dynamic-Window Approach (DWA)

This algorithm is an approach used in autonomous navigation, based on the velocity space of the robot and its optimization, to obtain an optimal local route. Unlike other methods, it takes into consideration the dynamics of the robot, which, within its approach, considers the physical limitations that the robot may have, in terms of its velocity and acceleration [85].

The DWA method starts from a velocity search for the robot in a two-dimensional space, as it considers only curvatures that are generated by pairs of linear and angular velocities (v, w). Figure 2.8 shows the velocity space V_s and the dynamic window V_d of a robot, where the gray area corresponds to the set of velocities that the robot cannot reach without stopping before a collision with an obstacle. The rest corresponds to admissible velocities, while the yellow zone, which is included within V_d, corresponds to V_r, the velocities that can be reached considering the physical limitations of the robot's velocity and acceleration.

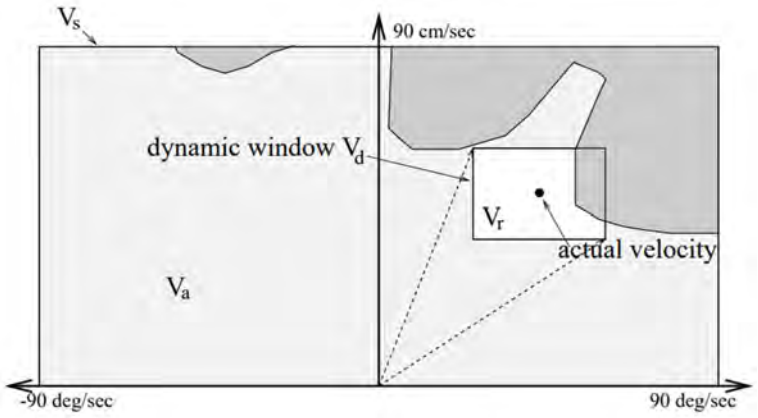

Figure 2.8: Dynamic Window Algorithm.

Source: [84]

After generating a feasible velocity space window for the robot, a cost function is generated to choose the velocity that the robot should follow, using Equation (2.6), shown below [84].

$$G(v, w) = \sigma(\alpha \cdot \text{direc}(v, w) + \beta \cdot \text{dist}(v, w) + \phi \cdot \text{vel}(v, w)). \tag{2.6}$$

From Of all the trajectories generated by V_r, the first component of the formula evaluates the alignment of the robot with the direction of the goal, as seen in Figure 2.9. After sampling the velocity space V_r, the curved trajectory of the robot when taking that velocity is simulated, to measure the angle towards the goal of this new estimated position. Then, the formula considers the distance to the nearest obstacle and, finally, the current velocity of the robot. From this cost function, the velocity that minimizes it and, therefore, optimizes the local planning is chosen.

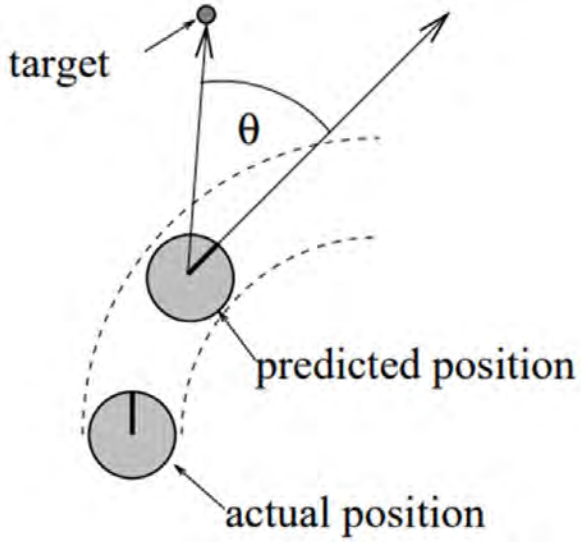

Figure 2.9: Direction towards the goal.

Source: [84]

Time Elastic Band (TEB)

The TEB algorithm arises from the possibility of a robot finding itself in an incomplete environment or with dynamic obstacles, where global gliders do not provide an adequate solution. In that sense, this approach takes into account the path generated by a global glider and is based on the idea of the elastic band, in which the initial path is deformed, considering it as an elastic band subjected to external forces that swings while keeping a certain distance from obstacles [86].

Unlike the elastic band, the TEB not only takes into account the intermediate states, such as orientation β_i and the position of the robot x_i, but adds the time intervals between states ΔT_i, as shown in Figure 2.10 . Within the objective functions of the TEB are conditions related to the dynamics of the robot and others regarding the efficiency or speed in fulfilling the trajectory [86]. In the following paragraphs, these aspects will be detailed.

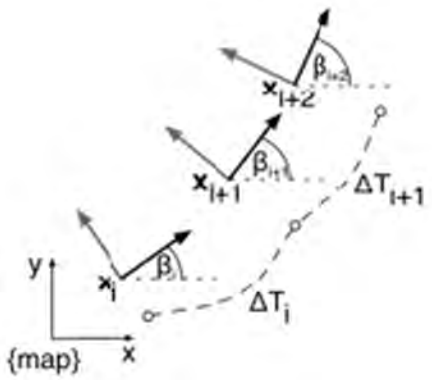

Figure 2.10: Sequences of states and times in TEB.

Source: [86]

First, the TEB seeks that the partial points of the trajectory generate an attractive force towards the rubber band, while the obstacles repel it [86]. This is achieved by obtaining the minimum distance to a waypoint or obstacle during the trajectory, as shown in Figure 2.11. Subsequently, objective functions that restrict the values of these distances according to the type of mark are considered, so that the distances to the obstacles are as large as possible and the tracking error of the route is minimal.

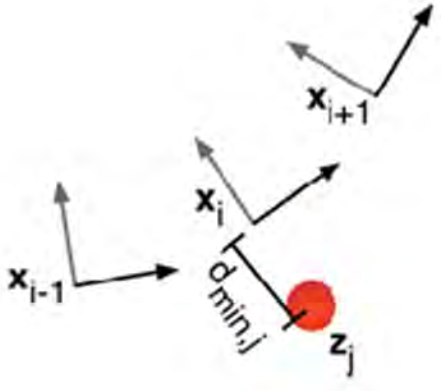

Figure 2.11: Minimun distante to waypoint or obstacle.

Source: [86]

Likewise, the TEB algorithm obtains the shortest route by generating internal forces that deform the elastic band [86]. In the case of the TEB, in addition to meeting that constraint, it is desired to obtain the fastest route, which is achieved by minimizing the summation of times between states ΔT_i. Finally, the dynamic constraints of velocity and acceleration are achieved in a similar way for the case of intermediate points and obstacles, as the minimum and maximum constraints of these are taken into account.

Model Predictive Control (MPC)

Model Predictive Control (MPC) is a control method that operates in discrete time. It employs a model of the system, optimizes a problem according to a given cost function and then outputs control signals [85]. In the particular case of the local planner, MPC is used to plan and control the movement of the robot in a local environment. It is also responsible for generating smooth and safe trajectories that allow the robot to navigate efficiently and to avoid obstacles.

$$\begin{bmatrix} x_{k+1|k} \\ x_{k+2|k} \\ \vdots \\ x_{k+H|k} \end{bmatrix} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^H \end{bmatrix} x_k + \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{H-1}B & A^{H-2}B & \cdots & B \end{bmatrix} \begin{bmatrix} u_{k|k} \\ u_{(k+1)|k} \\ \vdots \\ u_{(k+H-1)|k} \end{bmatrix} \tag{2.7}$$

MPC uses the mathematical model shown in Equation (2.7), which describes the dynamics of the robot and the ability to predict future states of the system in discrete time [85], where H represents the time period of the model prediction and u represents the control variable. It should be noted that the model shown is linear; however, it can be applied to nonlinear systems, previously performing a linearization around the state of the robot. From these predictions,

MPC seeks to find a sequence of control actions that optimize a defined cost function, which may be restricted to the physical limitations of the system, such as the speed and acceleration of the robot. The cost function may present different criteria, such as minimizing the distance traveled, arrival time or maximizing obstacle-avoidance capability.

2.1.4 Comparison of Local Planners

Table 2.2 shows a comparison of the most used local planning algorithms. For this, efficiency criteria were taken into account in the generation of the route, in terms of path length and time, computational or temporal complexity, and the strengths and weaknesses of each of them. Among the most notable features is that the DWA does not allow reverse movement. While the TEB and MPC do, while they are more suitable for navigation in dynamic environments [87]. Finally, these last two are very similar in terms of performance, with the difference that the MPC is suitable for robots of any configuration, even if they have complex dynamics.

Table 2.2: Comparison of local planners.

Algorithm	**Strenghts**	**Weaknesses**	**Route Efficiency**	**Time Efficiency**	**Computational Complexity**
Dynamic Window Approach [84]	Supports non-continuous cost functions	Not suitable for Ackerman type robots Does not allow reverse movement Problems with dynamic obstacles	Low	High	Low
Time Elastic Band [86]	Suitable for robots of any configuration with dynamics Allows dynamic obstacle avoidance Generated trajectories close to the optimal solution	High computational load	High	Medium	Medium
Model Predictive Control [88]	Allows dynamic obstacle avoidance Suitable for robots of any configuration with dynamics	High computational load	High	Medium	High

Table 2.3 shows a comparison of the revised State-of-the-Art of local-planning algorithms.

The research found in this regard is mainly focused on updates of these algorithms. It should be noted that most of these new enhancements are focused on improved performance in dynamic environments.

Table 2.3: Comparison of the State-of-the-Art of local planners.

Algorithm	Publication	Scenario		Upgrade	Strongness	Tests
		Static	Dynamic			
DWA	Y. Lin et al. [89]	X	X	Fusion of Best-First Search (BFS) and DWA.	Improved performance in dynamic environments.	Simulation
	L. Tianyu et al. [90]	X		Modification of evaluation function for angle variation.	Improved reaction to obstacles and smoother trajectory.	Simulation, Real
TEB	M. Li and C. Yang [91]	X	X	Parameter setting and optimization method for the Ackerman model.	Ackerman obstacle avoidance.	Simulation
	Q. Dai and X. Ma [92]	X	X	Fusion A* and TEB for collision detection.	Reduced navigation time, route efficiency and performance in dynamic environments.	Simulation
MPC	S. Ishihara and M. Kanai [93]	X		Adaptability for multiple robots and unmapped environments.	No prior information on the environment is required.	Simulation
	J. Li and M. Ran [94]	X	X	Improved speed and trajectory planning.	Route efficiency and performance in dynamic environments.	Simulation

2.2 Reinforcement Learning Algorithms

2.2.1 Conceptual framework on Reinforcement Learning and Deep Reinforcement Learning

In this subsection, will be explained some general concepts on what reinforcement learning techniques are based on. Moreover, its detailed how since due to the use of simulators for training they propose an inexpensive and robust alternative in autonomous navigation. Likewise,

they give agents the ability to learn to make decisions and actions in complex environments, such as uncontrolled or highly dynamic environments.

Reinforcement Learning (RL)

Reinforcement Learning (RL) is a branch of machine learning, in which an agent learns a certain task through trial and error. An RL system has three main components: policy, reward and value functions. The policy function allows mapping the states in a given environment to the actions to be taken. The reward function corresponds to the signal that evaluates the quality of those actions. The value function estimates the total cumulative reward expected for a given state [95]. In Figure 2.12, the above is illustrated, where an agent performs an action in a given environment, receives a reward and a new state. It should be noted that the agent's main objective is to maximize the accumulated reward through interaction with the environment.

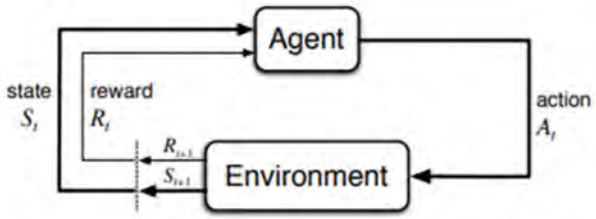

Figure 2.12: Reinforcement-Learning Scheme.

Source: [95]

The main goal of the agent is to maximize the accumulated reward through interaction with the environment. Thus, due to its efficiency in solving problems of all kinds, it has varied applications in video games, robotics, among others. For example, an RL algorithm can allow a robot to learn to manipulate objects and perform pick and place tasks. Through trial and error and interaction with the environment, the robot learns to execute precise movements to pick up and place objects efficiently.

Deep Reinforcement Learning (DRL)

On the other hand, Deep Learning (DL) emerges as a tool that, through the use of multilayer approximations of nonlinear functions [96], allows for solving different tasks of the machine-learning branch. In that sense, the graph in Figure 2.13 illustrates the different fields within this field of Artificial Intelligence. Where it can be highlighted that the development of deep learning directly influences all other fields. While deep reinforcement learning (DRL) is the intersection of the fields of DL and RL.

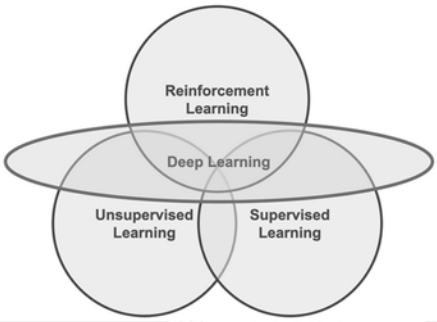

Figure 2.13: Areas of Machine Learning.

2.2.2 DRL Algorithms Classification

DRL algorithms can be classified into two main approaches: model-based and model-free (as shown in Figure 2.14). The difference between the two lies in the fact that model-based algorithms elaborate a model of the environment that allows the agent to predict how the environment will behave in response to its actions, whereas model-free agents learn from the interaction with the environment, without knowing its behavior. It should be noted that the following chapters will focus on the model-free algorithms, because, otherwise, the modeling of an uncontrolled environment is a problem.

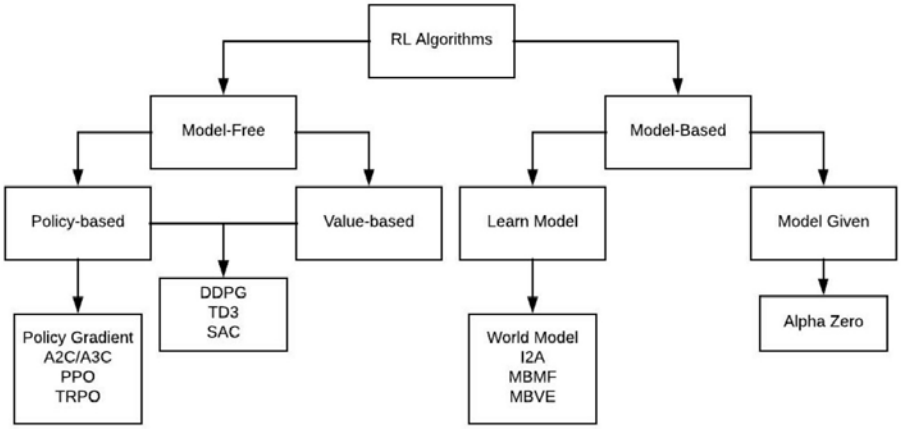

Figure 2.14: Taxonomy of RL algorithms.

Within the model-free approach are the algorithms based on the policy, value function and actor–critic. The value function is estimated to then derive the policy. In that sense, giving the function a state s and an action a, by means of the following formula the optimal policy is obtained [97]. That is, the action that maximizes the value function, on the basis of a given state, is taken as shown in Equation (2.8):

$$a = \arg\max_{a} Q(s, a), \tag{2.8}$$

while the policy-based algorithms directly optimize it without estimating the value function initially. For this, it is a matter of finding a series of parameters θ, which optimizes $\pi(s, a|\theta)$; this, being a more direct method, allows greater stability and reliability in the results [97]. Finally, the actor-critic algorithms combine the methods by value function and policy and obtain a neural network of the actor type that refers to the policy while the critic corresponds to the value function of the current policy $Q(s, a; \theta)$ [98].

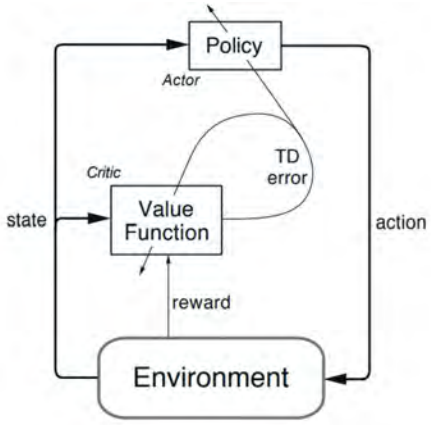

Figure 2.15: Architecture of actor-critic algorithm.

Source: [95]

Figure 2.15 illustrates the architecture of the actor- critic method, where what was explained above is extended. The critic evaluates the actor's performance and provides feedback. Whereas, the actor uses feedback to improve the training of the policy.

2.2.3 DRL techniques for navigation in indoor environments with known map

In this subsection, navigation systems based on reinforcement learning techniques that require prior information from the environment will be detailed.

Robot Navigation with Map-Based DRL

The proposed method develops an autonomous navigation technique that requires prior information from the environment for its implementation. The architecture of the algorithm is illustrated in Figure 2.16, where through the simultaneous localization and mapping defined by the SLAM algorithm, map information can be obtained and the state of the robot can be estimated. From this, a glider is defined, generating local goals from a global trajectory according to the

map. These target positions are sent to the DRL glider, which is responsible for local navigation and obstacle avoidance, by sending the actions that the robot must take in a certain state.

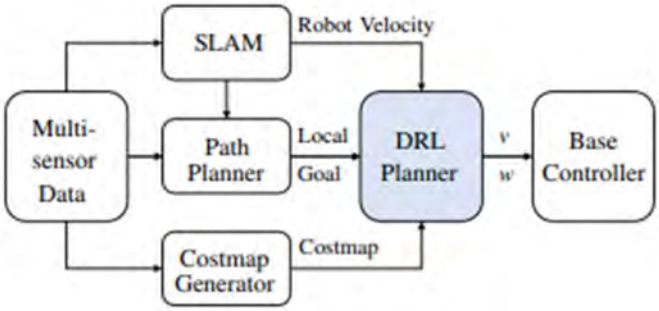

Figure 2.16: Architecture of map-based navigation system.

Source: [99]

A DQN algorithm was implemented for the DRL planner, which was trained using the curricular learning technique, in which the agent learns in environments that progressively increase its difficulty [99]. The architecture of the algorithm is illustrated in Figure 2.17, where it is highlighted that the inputs to the neural network are images as information about the goal and the state of the robot. On the one hand, one branch includes the angular and linear velocity and the position of the goal as input, which later together with the output of the other branch enter a CNN.

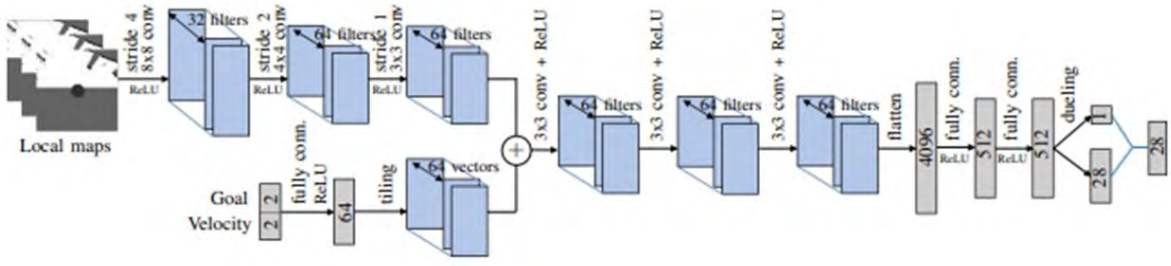

Figure 2.17: Architecture of DRL algorithm.

Source: [99]

On the other hand, the second branch presents as inputs three local images of the cost map

that are included in different channels of an image. To do this, they are based on cost map research, in which different map layers are created, where each of them tracks a specific type of obstacle or restriction [100].

Robot Navigation in Crowded Environments Using DRL

In recent years, mobile autonomous navigation in highly dynamic environments has been a problem, due to the difficulty of modeling the uncontrolled behavior of the movement of dynamic objects, such as people. Given this, this work proposes SOADRL, a navigation method that takes prior information from the environment and combines it with the robot's measurements. The architecture of the algorithm is illustrated in Figure 2.18, where the inclusion of mainly three types of input branches can be highlighted.

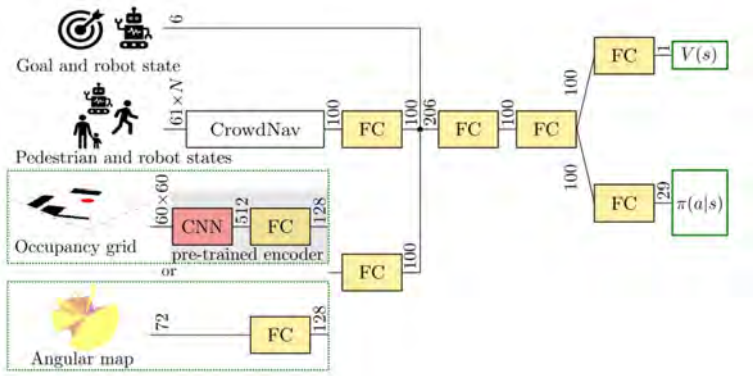

Figure 2.18: Architecture of navigation system.

Source: [101]

The first of them has information on static objects only, for this it takes into account the environment map and an angular map, which enter an encoder and an FCN layer, respectively. Then, the two output layers are combined to produce the input corresponding to the static obstacles. The types of maps mentioned above are shown in Figure 2.19, where angular maps stand out for being simple information to obtain by different types of sensors and the robust

information they provide about the obstacles around a robot [101]. On the other hand, the second branch has information on the state of the robot such as speed, robot radius, distance and orientation error to the goal.

(a) Angular map with full FOV. (b) Angular map with limited FOV. (c) Top-down view of the environment with robot.

Figure 2.19: Static obstacle branch entries.

Source: [101]

The dynamic agents branch uses the Crowdnav algorithm, which is shown in Figure 2.20, in which local maps containing information about the robot's interaction with passers-by are encoded. These maps are then combined using a self-attention algorithm in the pooling stage. Finally, the positions of the people and the robot are estimated [102]. It should be noted that the DRL algorithm used for policy optimization is A3C, due to the possibility of parallel training which accelerates the learning time [101].

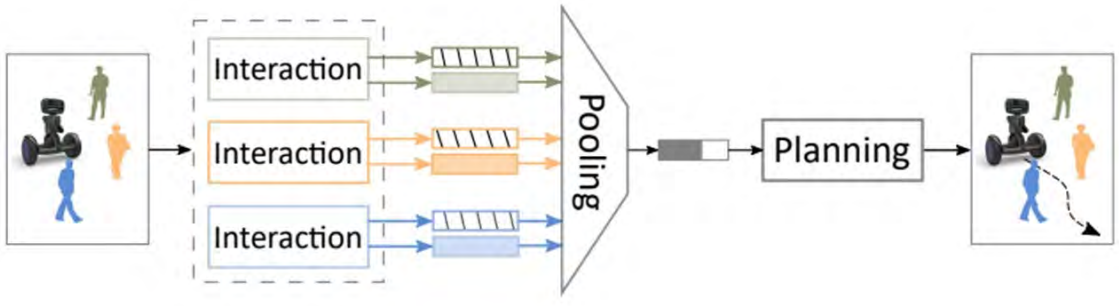

Figure 2.20: Architecture of Crowdnav algorithm.

Source: [102]

2.2.4 DRL techniques for navigation in indoor environments with unknown map

In this subsection, navigation systems based on reinforcement learning techniques that do not require prior information from the environment will be detailed.

Mobile robot navigation using DRL

This method proposes an autonomous navigation algorithm that does not require prior information about the environment, nor indications of the location of the goal. The block diagram of this system is illustrated in Figure 2.21. First, the MobileNetv2 detection model is used using an RGB and depth camera, through which the detection probability and distance to the goal are obtained. It should be noted that, initially, when the location of the goal is not certain, default values are considered for the goal parameters, such as distance or angle, until it is located [103].

For training, this proposal used the DDQN algorithm, achieving a 5.06% improvement in performance compared to DQN [103]. To do this, a state space was designed based on the laser sensor measurements, information on the location of the goal and the closest obstacle. On the other hand, they used a discrete state space, in which the linear velocity is constant and the angular velocity can have five possible values. Finally, the agent receives a positive reward when it approaches the goal and a negative reward when it collides with or moves away from the goal [103]. To do this, the following two equations 2.9 and 2.10 are used to calculate the reward based on the angle and distance towards the goal, R_θ and R_d respectively.

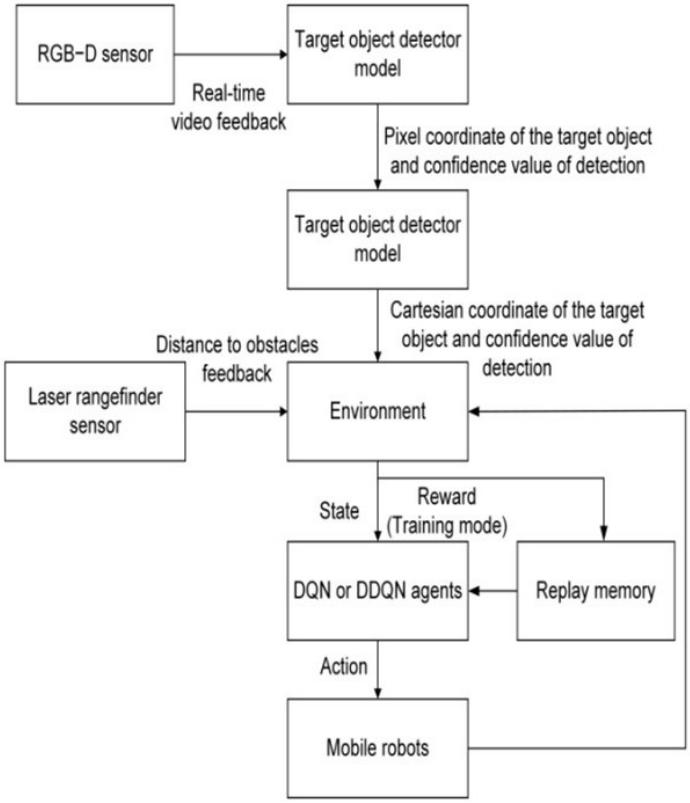

Figure 2.21: Navigation system flowchart.

Source: [103]

$$R_\theta = 5x1^{-\Lambda(\theta)} = \begin{cases} \geq 0, -\frac{1}{2}\pi < \frac{1}{2}\pi, \\ \\ < 0, otherwise \end{cases} \tag{2.9}$$

$$R_d = 2^{\frac{D_c}{D_g}} = \begin{cases} > 2, D_c < Dg, \\ \\ (1, 2], otherwise \end{cases} \tag{2.10}$$

Where θ represents the angle toward the goal, D_c the current distance toward the goal, and D_g the previous distance toward the goal [103].

Autonomous mobile robot navigation in uncertain dynamic environments using DRL

This method proposes an autonomous navigation algorithm for mobile robots in unknown environments based on Deep Reinforcement Learning techniques. For which they combine a DDPG with an LSTM for the actor-critic, as can be seen in Figure 2.22. The main reason for including an LSTM model is because laser sensors are susceptible to interference from the environment and that, the Action taking decision should not only take into account the current state, but also past information [104]. In that sense, the diagram shows that the input corresponds to the distance and direction towards the goal, orientation of the robot and the information from the laser sensor in the last 10 instants of time.

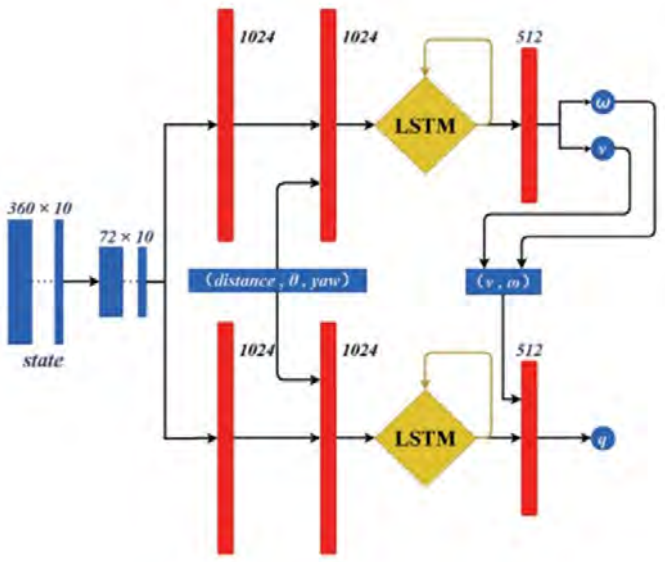

Figure 2.22: Detailed architecture of actor-critic algorithm.

Source: [104]

Likewise, it has a continuous action space since the output of the DRL model corresponds to the linear and angular speed of the robot. On the other hand, for the training of the model, a reward system was designed that is shown in the equation 2.11 [104], where R_a corresponds to the reward if the robot reaches the goal, R_c if it collides and if On the contrary, a formula is

used in which the robot's speed information and the difference between the current and previous distance to the goal are considered.

$$r = \begin{cases} R_a & if arrive \\ R_c & if collision \\ \lambda(C_goal - P_goal) + \lambda_2(v - |w|) & else \end{cases} \tag{2.11}$$

Finally, the architecture was compared and demonstrated improvement with respect to the DDPG and ADDPG algorithms in terms of security and route optimization [104].

2.2.5 Review of Local Planners

Table 2.4 shows a comparison of the autonomous navigation algorithms for mobile robots that employ DRL techniques reviewed in the literature. From which it should be noted that the application scenarios are mainly divided into two: Indoor navigation and Social navigation. And the difference is that the last one focuses on navigation in highly dynamic environments, such as a place with many passers-by. Likewise, according to the state of the art, for social navigation there is an additional perception to traditional sensors, which is based on a static or dynamic agent level. On the other hand, the action space can be varied depending on the dynamic limitations of the robot or algorithm to be used. Finally, it is highlighted that robust social navigation generally requires prior information about the environment, such as a map.

Table 2.4: Comparison of DRL-based navigation algorithms.

Article	Navigation type	Algorithm	Environment			Perception type			Action space	Tests
			Static	Dynamic	Social	RGB Camera	Depth Camera	Laser Sensor		
G. Chen et al [102]	Map-Based	DQN	X	X		X	X	X	Discrete (28)	Simulation, Real
L. Liu, et al [101]	Map-Based	A3C	X	X	X	X		X	Discrete (28)	Simulation, Real
Z. Lu & R. Huang [100]	Mapless	LDDPG	X	X	X			X	Continuous	Simulation
F. Leiva & Javier R. [47]	Mapless	DDPG	X	X				X	Continuous	Simulation, Real
R. Min-Fan & H. Sharfiden [103]	Mapless	DDQN	X	X		X	X	X	Discrete (5)	Simulation, Real
Y. Sasaki, et al [105]	Map-Based	A3C	X	X	X			X	Continuous	Simulation, Real

Chapter III

Development of frameworks for autonomous navigation

This chapter describes the methodology followed in order to execute this research. First, the simulation configuration describes in detail the robot characteristics and the simulation environment. Then, the ROS-navigation stack is detailed, focusing on the mapping, localization and path-planning steps. Finally, the implementation of the local-planners algorithms and the mapless-DRL-based algorithms are explained.

3.1 Traditional autonomous navigation algorithms

3.1.1 Simulation Configuration

In the past, the implementation of robots required to be evaluated once they were fully implemented. This entailed excessive development costs, in case of unforeseen events or failures. Currently, there are different 3D-simulation environments capable of faithfully representing physical and dynamic aspects of the real world. There are also tools that allow the speeding up

of the tasks that are most demanded in the field of robotics. Therefore, these two aspects related to the implementation of this project will be detailed below.

Robot Description

The description of the robot in the simulation was based on the tree diagram in Figure 3.1, where the connections between the main components of the robot and the transformations generated when creating the robot model using the Unified Robot Description Format (URDF) are defined. It can be highlighted that the robot used a differential drive, in which there were four pivoting wheels (as shown in the detailed design of the robotic platform in Figure 3.2). Likewise, the laser implemented in the simulation had a scanning range of 240°. On the other hand, it should be noted that the odometry of the real robot is calculated by means of a sensor fusion of an inertial sensor and encoders on the wheels. However, for simulation purposes, the simulator was in charge of the respective calculations.

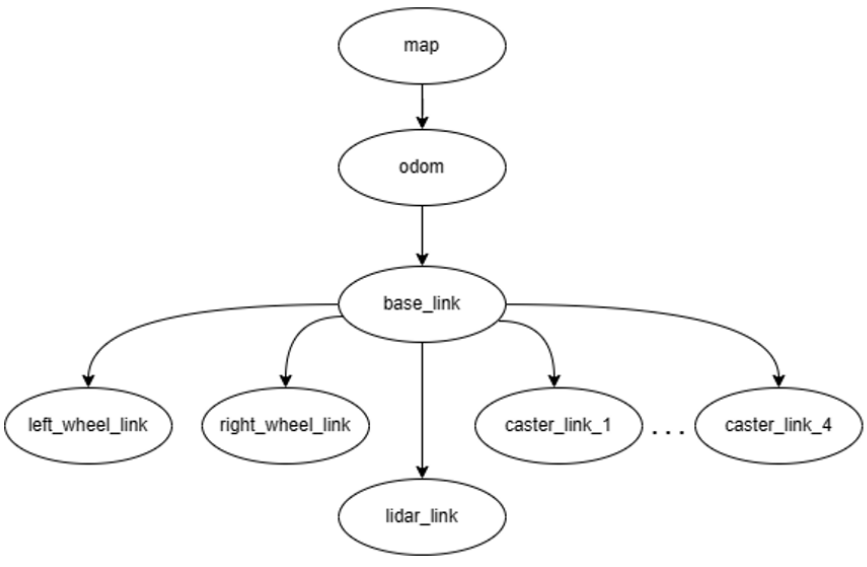

Figure 3.1: Robot URDF tree diagram.

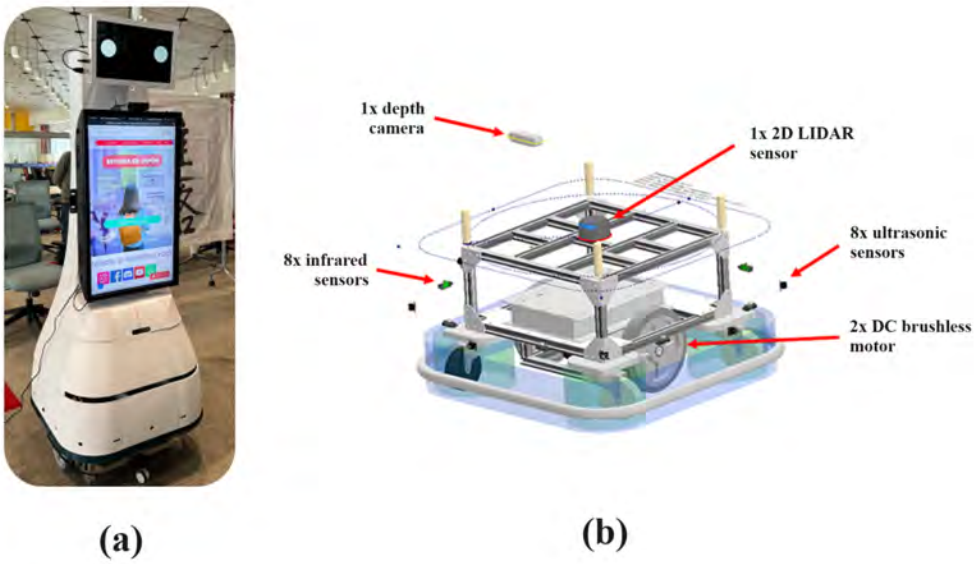

Figure 3.2: Robot used for the study: (**a**) Real robot image. (**b**) Internal components of robotic platform.

Differential Drive System Design

The robot is equipped with a movement system driven by traction and caster wheels. Its traction system employs a differential distribution mechanism (see Figure 3.3).

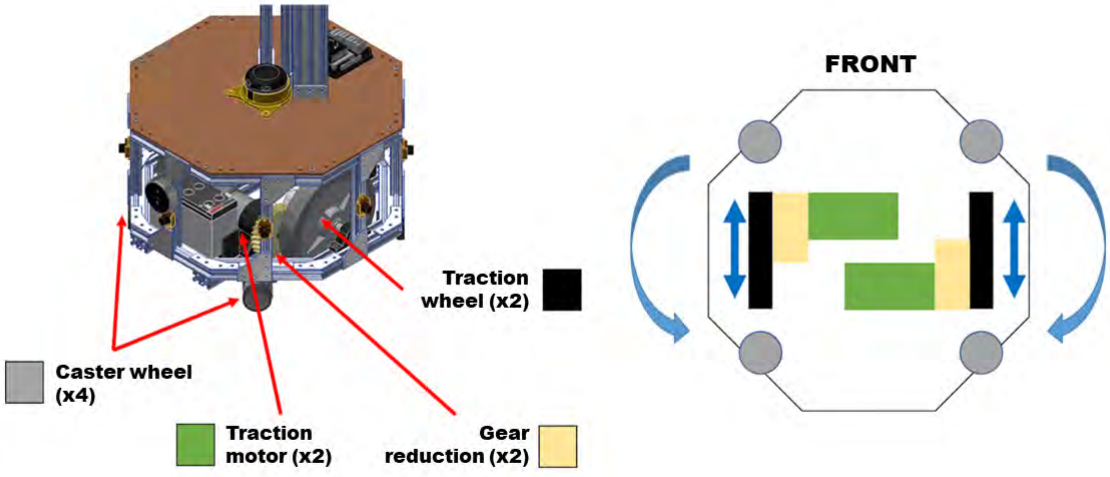

Figure 3.3: Differential drive system composition.

This differential setup enables two distinct types of movement: linear motion, allowing forward or backward movement, and rotational motion around its axis in either a clockwise or counterclockwise direction. The traction system functions seamlessly, comprising four key components: support pulleys, traction wheels, a gear system, and traction motors. Four (04) caster wheels bear the weight of the robot while facilitating multi-directional movement. Two (02) traction wheels transmit movement generated by the pair of traction motors. Additionally, two (02) gear systems positioned between the motors and drive wheels act as reduction boxes, enhancing torque by reducing the motor rotation speed, thereby enabling efficient movement of the robot.

Sensing System Design

The robot's navigation system, which allows it to move in environments with limited space and frequent moving obstacles, is composed of a sensing system of 360° cameras and sensors located in the upper front part to locate the users in front and in the lower part to move through dynamic environments (see Figure 3.4). The sensing system is composed of one (01) depth camera, one (01) LIDAR sensor, and an array of eight (08) ultrasonic sensors (as shown in Figure 3.2b).

The depth camera is located at the front of the robot to visually detect and identify obstacles in front and the areas through which the robot can move. This camera has an operating range of 0.25 to 9 m, a depth resolution of 1280 $\times$ 720, 30 FPS, and a depth field of 70 $\times$ 55. The Lidar sensor, located at the base of the robot (at a height of 40 cm above ground level), is used to identify static and dynamic obstacles that are in the path of movement and on the sides of the robot. This sensor has a working range of 0–360°, a scanning range of 0.2–25 m, and a scanning frequency of 5–15 Hz. The array of ultrasonic sensors is used to detect obstacles in the environment, on the front, sides, and rear area, with a measurement range of up to 10 m.

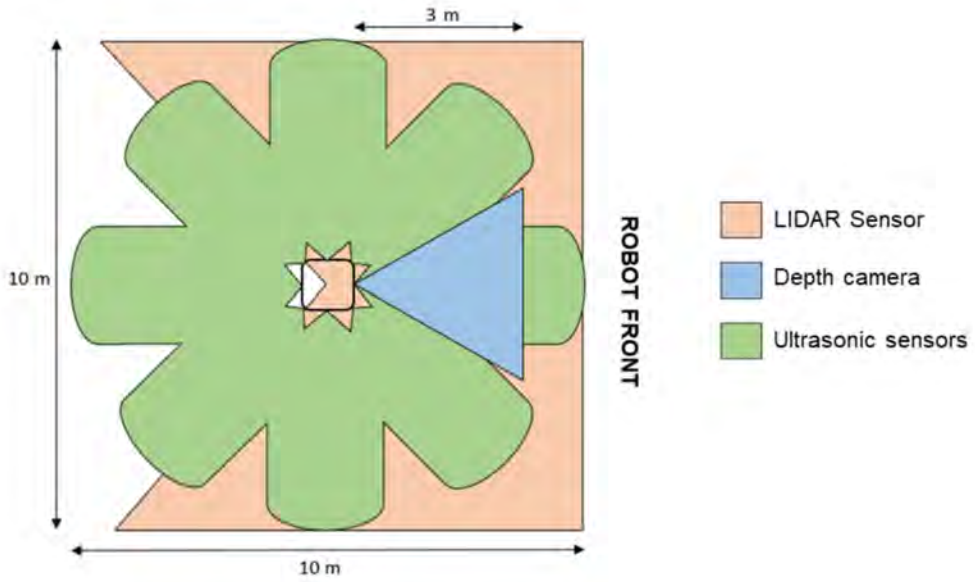

Figure 3.4: Top view of the robot that includes the field of vision of the components of the sensor system.

A strategy involving the use of multiple sensors utilizing different technologies for environmental identification was carried out in order to have adequate identification of static and dynamic obstacles [106]. Sensors, by using different technologies, provide different types of information (regarding the resolution, frequency, and amount of data) that complement each other to obtain an adequate mapping of the environment.

Control Systems

A dedicated navigation controller (Jetson Xavier, Nvidia Corporation) is used to execute the navigation algorithms, command the movement of the motors for the robot's movement, and process the information from the sensing system. The navigation controller is based on a two-level architecture: a local planner that aims to locally reach the positions established by the global planner. For this, the dynamics present in the navigation environment are taken into account, including fixed and moving obstacles measured by fusing the different sensors of the sys-

tem. The local planner has been developed using algorithms based on reinforcement learning, allowing for changing environments to be managed through the analysis of multiple scenarios (actions and observations), penalties, and rewards, imitating how humans can perform the task, imitating the same process of learning based on the collection of experiences.

Simulation Environment

For the implementation of classical-navigation algorithms, the Robot Operating System (ROS) framework and the Gazebo simulator were used. The latter provided information from the environment that was measured by the sensors and used by the ROS-navigation package, which consisted of mapping, localization and route planning, both global and local. As shown in Figure 3.5, this package sent speed commands to the robot that was simulated in Gazebo. Also, the information on grid maps, costs and the planned route were visualized in RVIZ.

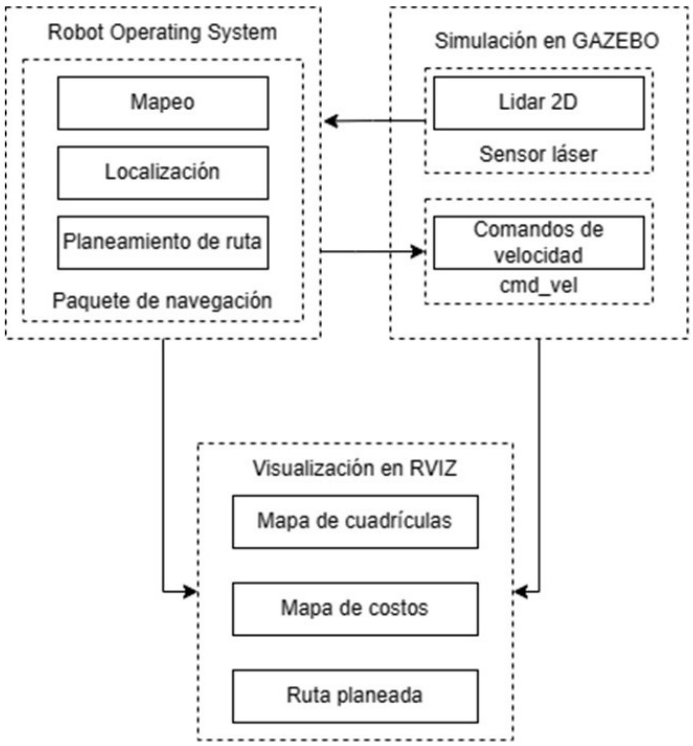

Figure 3.5: ROS-navigation stack for robot simulation.

3.1.2 ROS Navigation Stack

The ROS-navigation package facilitates the work of implementing each of the algorithms necessary for the autonomous navigation of mobile robots of a large number of configurations. Figure 3.6 illustrates the main components of the package and how they interact with each other. It highlights that there are specific nodes of the robotic platform, such as sensors or odometry source, which are detailed above. On the other hand, there are the route planning, mapping and localization nodes, which are detailed below.

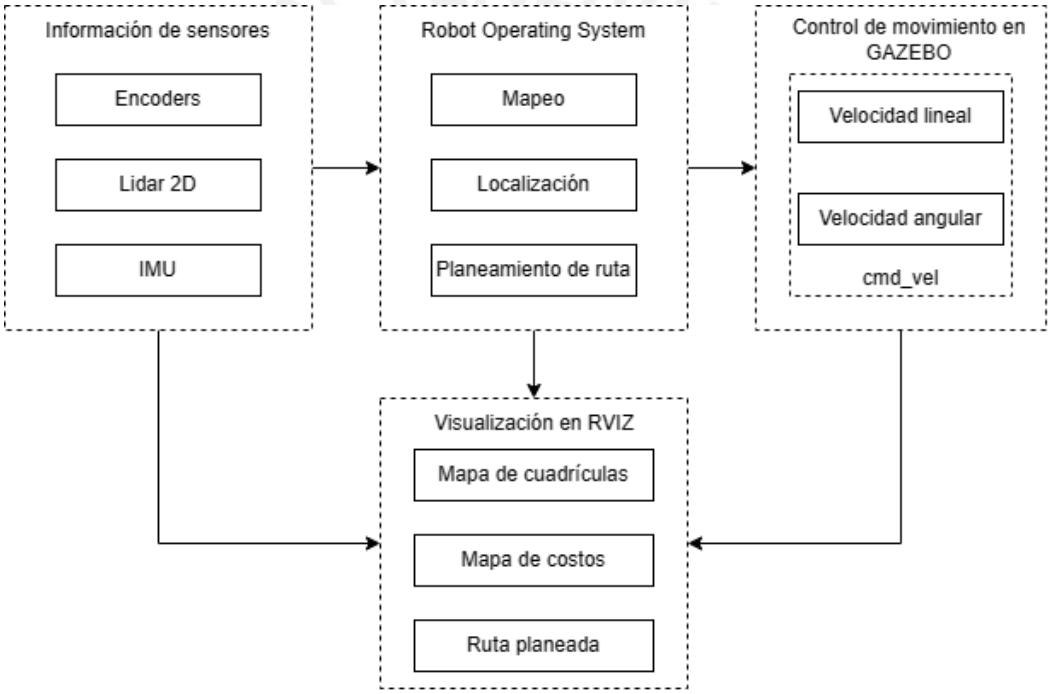

Figure 3.6: ROS Navigation Stack.

Source: [107].

Mapping

In order to obtain a map for navigation, a 3D environment was designed, in which the different navigation algorithms were evaluated. This environment was built in Gazebo 9.0 and was intended to simulate a dynamic social environment such as, for example, a shopping mall.

Therefore, it had corridors, entrances and rooms, as well as maximum dimensions of 27.5 $\times$ 34 m.

Figure 3.7a illustrates the recreated 3D environment, while Figure 3.7b shows the occupancy map. For the latter, the Gmapping package was used; it should be noted that only a 2D laser sensor was used for sensory perception.

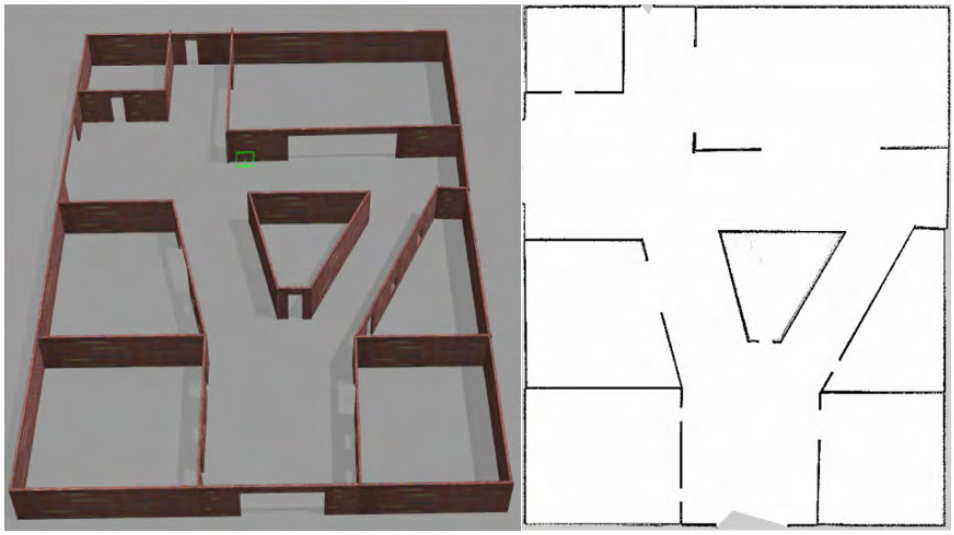

Figure 3.7: Mapping of the environment: (**a**) 3D environment. (**b**) Occupancy grid map.

Localization

The AMCL-ROS package was employed for the robot localization, i.e., for the robot position and orientation estimation. For this purpose, the Adaptive-Monte-Carlo-Localization-(AMCL) algorithm was employed, which subscribed to the 2D-laser-sensor topic, robot transformations and map. It is worth noting that the package improved its localization accuracy as more time elapsed.

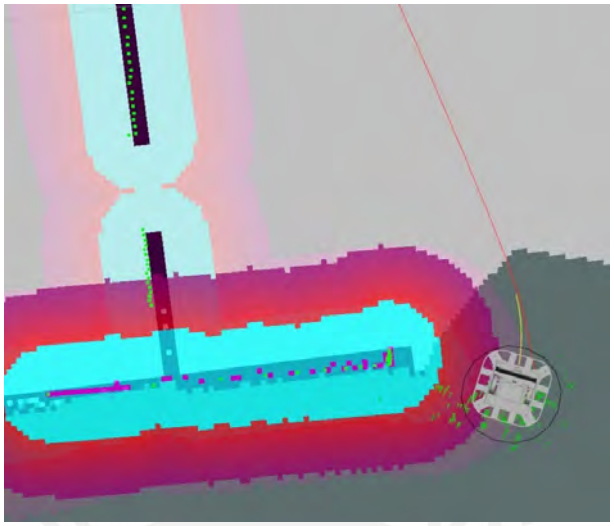

Figure 3.8: Robot localization algorithm during simulation.

The algorithm was based on a particle filter, to represent the hypotheses of the most probable positions of the robot. When the robot performed a measurement, the similarity of that information to the information provided by the map was evaluated and, in this way, after the robot's movement, the new state of the particles was predicted. Thus, the particle distribution was adaptively adjusted to the information from the environment and sensor measurements. Figure 3.8 shows the localization package in the operation of the robot, where the green arrows correspond to the particles generated from the filter.

Path Planning

The global route planning was generated by the Navfn algorithm, which used Dijkstra's algorithm to find the most optimal route. It should be noted that there was the possibility of it being modified to use the A* algorithm, but the results using Dijkstra were similar and did not affect navigation, because the local planner was responsible for generating the avoidance of obstacles not considered in the map. Finally, Figure 3.9 illustrates the route generated by the global and local planners, as a black line and a yellow line, respectively.

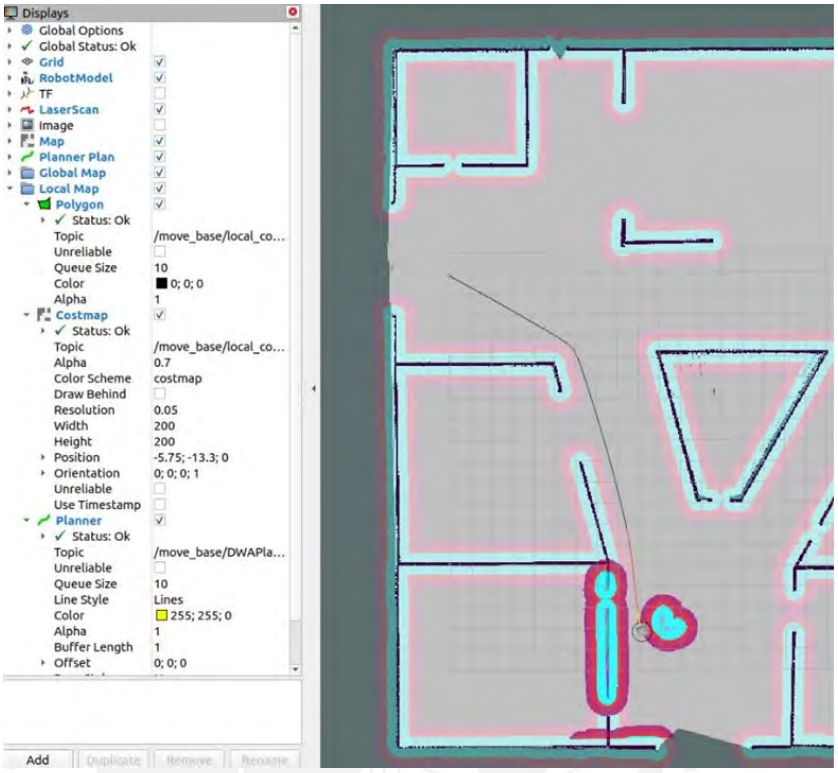

Figure 3.9: Robot localization algorithm during simulation.

It is worth mentioning that achieving optimal navigation performance involved fine-tuning parameters related to the inflation layer of the cost map. As illustrated in Figure 3.8, the cost associated with cells was heightened, as indicated by the expanded size of the sky-blue contour encircling the map. This adjustment resulted in the generation of a more secure route for the robot, steering it away from potential obstacles and enhancing overall safety.

3.1.3 Local-Planners Algorithms Implementation

Table 3.1 shows the main modified parameters of the local planners. Initially, it is possible to highlight the values that correspond to the physical limits of the robot, such as velocity and acceleration. On the other hand, the tolerance of distance and angle to the goal were 0.1 m and 0.2 rad, respectively.

Table 3.1: Common parameters for local planners.

Name	Notation	Value
Speed and acceleration limits	max_vel_x	0.4
	max_vel_theta	0.6
	acc_lim_x	2
	acc_lim_theta	0.1
Goal tolerance	xy_goal_tolerance	0.1
	yaw_tolerance	0.2
DWA parameters	sim_time	2.5
	vx_samples	20
	vth_samples	40
	path_distance_bias	32
	goal_distance_bias	24
	occdist_scale	0.02
TEB parameters	max_vel_x_backwards	0.4
	footprint_model_radius	0.335
	min_obstacle_dist	0.4
	include_dynamic_obstacles	True

Dynamic-Window Approach (DWA)

The first important parameter of the DWA algorithm is the sim_time, as it corresponds to the time that the glider will have to obtain an optimal solution. In particular, a value of 2.5 s was chosen, as it was an adequate time in which to solve situations that might require a high processing time and, at the same time, did not reach high times. On the other hand, as for the $vx\_samples$ and $vth\_samples$ parameters, values of 20 and 40, respectively, were taken. This was due to the fact that the higher the number of samples, the higher the computational cost and that, in general, the rotation task was more complex than the translation task in the x-axis.

Finally, other relevant parameters corresponded to the path-cost function implemented in ROS, shown in Equation(3.1):

$$\begin{aligned} \text{cost} = {} & \text{path\_distance\_bias} \cdot \text{distance\_to\_path} + \\ & \text{goal\_distance\_bias} \cdot \text{distance\_to\_local\_goal} + \\ & \text{occdist\_scale} \cdot \text{maximum\_obstacle\_cost\_on\_path}, \end{aligned} \tag{3.1}$$

where the parameter $path\_distance\_bias$ corresponded to how close the path was to the global route, $goal\_distance\_bias$ corresponded to the weight with which the robot would approach the local goal instead of the global route and $occdist\_scale$ was the weight with which the robot avoided obstacles [108]. These parameters were set to those shown in Table 3.1, because of their positive results for the simulated robotic platform.

Time Elastic Band (TEB)

Unlike the DWA, the TEB algorithm allows reverse motion, so one of the differentiating parameters with respect to the DWA in the dynamics of the robot was $max\_velocity\_x\_backwards$, and it was configured to have the same linear velocity as the frontal motion. Also, the 2D footprint model for the robot was redefined for this local glider, because unlike the footprint used for the cost map it was intended to be a simpler representation than the 2D projection of the CAD model of the robot. Figure 3.10 illustrates the comparison of the footprints for both situations.

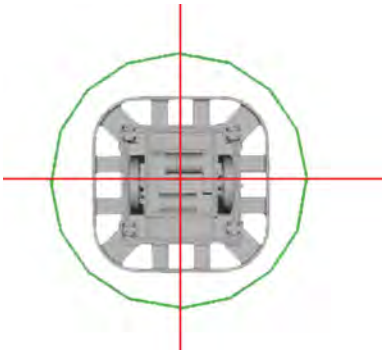

Figure 3.10: Robot base footprint.

The main reason behind this was that the cost map footprint was used to detect a possible collision and, therefore, required precision in its implementation, while TEB was used for optimization purposes; the complexity of the original figure was reduced to a circular one, to reduce computational time. With respect to obstacle avoidance, parameter values were defined by trial and error until adequate results were obtained in confined, wide or obstacle spaces. In this sense, the main modified parameters were $min\_obstacle\_dist$ and $include\_dynamic\_obstacles$, with values of 0.4 m and True, respectively.

3.2 Map-Based Deep-Reinforcement-Learning Algorithm

Navigation in dynamic environments is currently a problem because planners using reinforcement learning techniques require assumptions about the behavior of dynamic agents. In this sense, the CADRL algorithm [109] proposes a navigation system focused on obstacle avoidance, without assuming any type of rule for the behavior of the agents. Moreover, it has no limitation in the number of agents for the observations. It is worth mentioning that the algorithm employed in our application was open source [110] and had been adapted to suit the requirements of our robotic application.

3.2.1 Algorithm Architecture

The reason why CADRL allows a variable number of agents is the inclusion of an approach used in natural processing language known as LSTM. This way, in Figure 3.11 is possible to add a number of observations of the agents near the robot can be added. Meanwhile, observations concerning the robot are also counted. Finally, the network produces an output corresponding to the state value and to the policy.

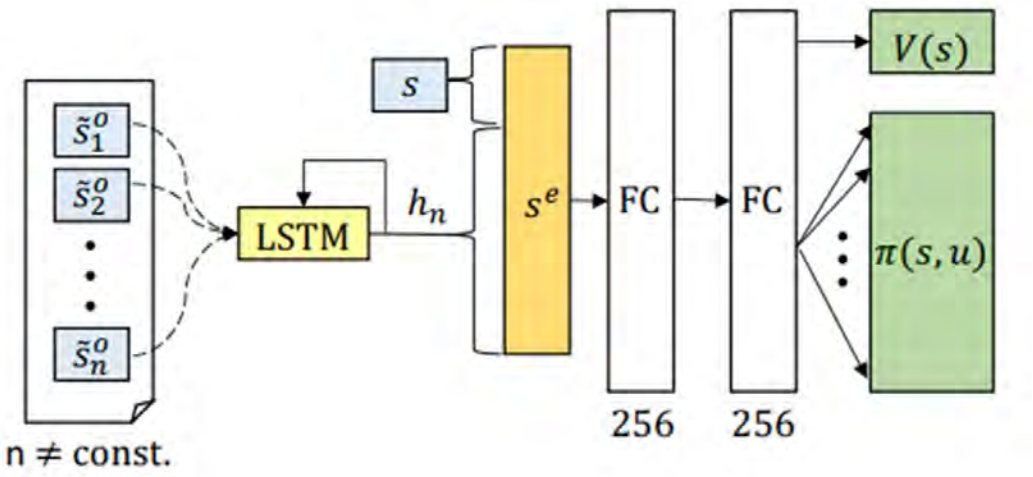

Figure 3.11: CADRL Algorithm architecture.

Within the observations are included the position, velocity, radius and distance to the goal for the robot, while for the dynamic agents, the orientation, goal position and distance to another agent are also included. On the other hand, the action space is discrete, with 11 possible values. Also, the reward function is defined in Equation (3.2) [109]:

$$R\_col = \begin{cases} 1 & \text{if reached the goal} \\ -0.25 & \text{if } d\_min < 0 \\ -0.1 + 0.05 * d\_min & \text{if } 0 < d\_min < 0.2 \\ 0 & \text{otherwise,} \end{cases} \tag{3.2}$$

where d_min is the distance to the nearest agent.

3.2.2 Robot parameter configuration

Due to the number of dependencies that the implementation of this algorithm in ROS uses, a free-to-use Docker container called arena-rosnav was used. On the other hand, it should be noted that for the execution of this algorithm, different parameters were configured referring to an intermediate glider that allows interaction between the global and local glider using DRL. Of which the most important are mentioned in Figure 3.12.

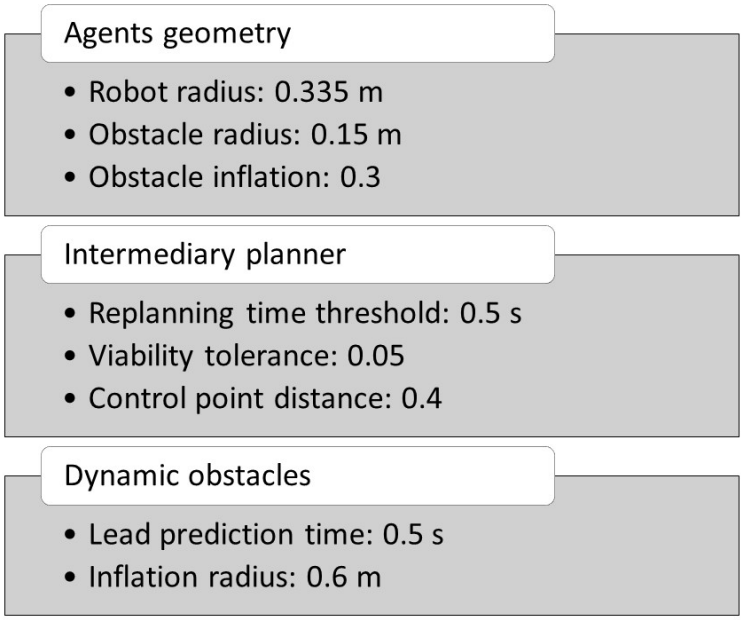

Figure 3.12: CADRL parameters configuration.

Likewise, to run the simulation in the container, the scenarios evaluated on the local gliders were included, as well as the entire robot configuration, which had to be adapted. Finally, Figure 3.13 shows the navigation of the robotic platform in RVIZ using the CADRL algorithm.

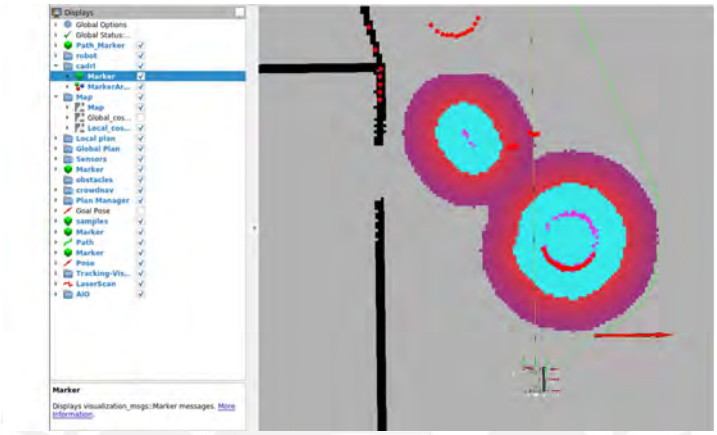

Figure 3.13: Autonomous Navigation with CADRL.

3.3 Mapless-Deep-Reinforcement-Learning Algorithms Implementation

Traditional approaches to autonomous navigation commonly rely on accurate maps of the environment. However, in situations where a detailed map is not available, an algorithm that allows adaptability in such environments is critical. Given this problem, different DRL models were tested and compared in the Arena2D simulator [111].

3.3.1 State Space

The state space consists of the following information:

- Distance and angle to the goal;
- 2D point cloud from the laser sensor;

- Distance and angle to the nearest obstacle.

It is worth noting that the 2D point cloud from the Lidar sensor was discretized into 240 samples, using the simulator's tools. In this way, as the robot initially had a field of view of 240°, there was one sample for every approximately 1°, as shown in Figure 3.14.

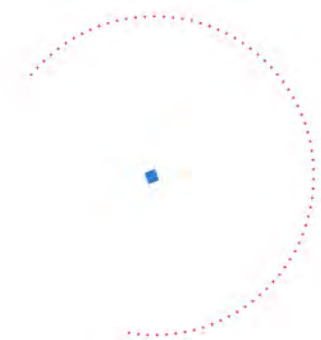

Figure 3.14: Robot laser sensor sampling.

The choice of this value instead of a complete 360 sampling is mainly due to computational cost effects. While as seen in Figure 12, this sampling value was taken instead of a lower one due to the reduction in the algorithm's convergence time.

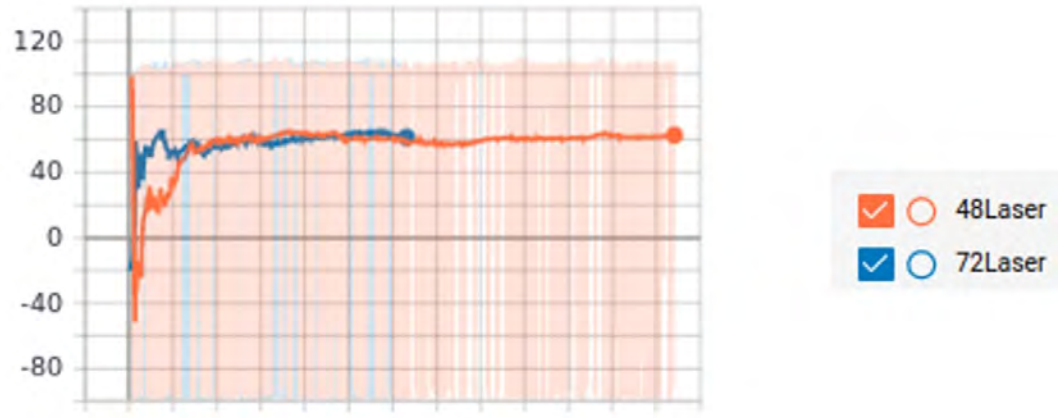

Figure 3.15: Algorithm training using laser sampling of 48 and 72.

3.3.2 Action Space

For the action space, linear velocities were taken in the range of $[-0.4, 0.4]$ m/s and angular velocities in the range of $[-0.6, 0.6]$ rad/s. Likewise, the algorithms were tested with both discrete and continuous action spaces, in order to analyze their influence on navigation performance. Table 3.2 presents the velocities for each possible robot action.

Table 3.2: Discrete and Continuous Motion Commands.

Discrete	**Continuous**
• Forward: $(0.4\,\text{m/s}, 0\,\text{rad/s})$	• Linear velocity: $[-0.4, 0.4]$ m/s
• Reverse: $(-0.4\,\text{m/s}, 0\,\text{rad/s})$	• Angular velocity: $[-0.6, 0.6]$ rad/s
• Left: $(0\,\text{m/s}, 0.6\,\text{rad/s})$	
• Left-Center: $(0.3\,\text{m/s}, 0.3\,\text{rad/s})$	
• Right: $(0\,\text{m/s}, -0.6\,\text{rad/s})$	
• Right-Center: $(0.3\,\text{m/s}, -0.3\,\text{rad/s})$	
• Stop: $(0\,\text{m/s}, 0\,\text{rad/s})$	

3.3.3 Reward Model

First, the agent received a positive sparse reward equal to 100 if it reached the goal. Meanwhile, a dense reward was given as it got closer to or farther from that point, as seen in Equation (3.3) based on [112], where a $k = 10$ was considered for the second reward:

$$\text{Reward\_closeness} = \begin{cases} 100 & \text{if reached the goal} \\ k \cdot (d_{t-1} - d_t) & \text{otherwise.} \end{cases} \tag{3.3}$$

The primary objective of the robot, in addition to reaching the goal, was obstacle avoidance

in both static and dynamic environments. In this regard, a reward function was designed to assign a negative reward when the robot was at a certain distance from an obstacle, as depicted in Figure 3.16.

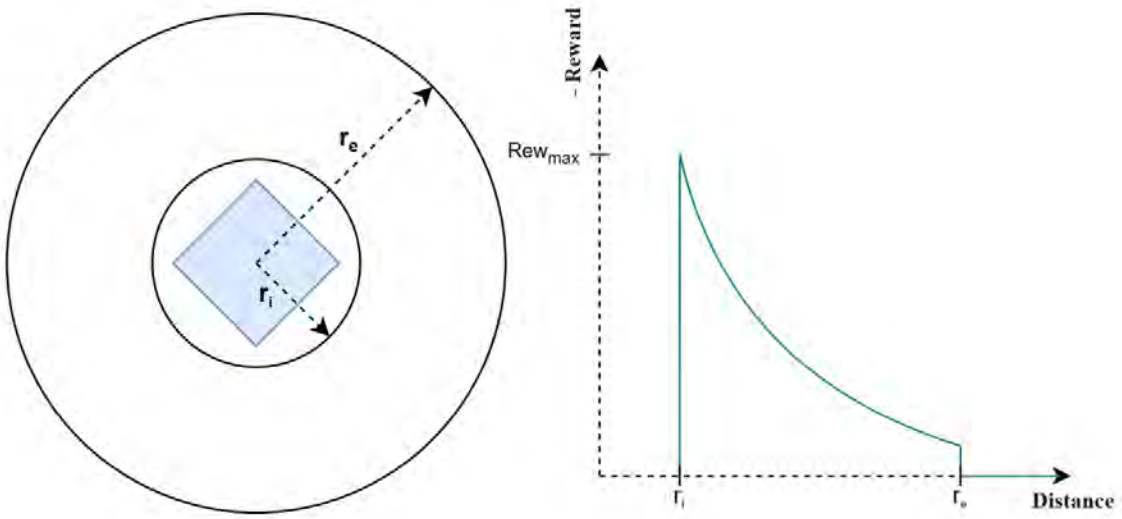

Figure 3.16: Characteristics of the reward model for obstacle avoidance.

The exponential function allowed the reward to decrease significantly as the distance to the robot approached r_i. If the distance exceeded a certain value r_e, which corresponded to the robot's safe zone, the reward became 0. Therefore, to achieve obstacle avoidance, the robot received a reward according to the set of Equation (3.4),

$$\text{Reward\_avoidance} = \begin{cases} -\text{Rew\_max} \cdot e^{-c \cdot (distance - r_i)} & r_e \geq \text{distance} \geq r_i \\ 0 & \text{distance} > r_e : \end{cases} \tag{3.4}$$

- Where the parameter Rew_max $= 10$ corresponded to the reward assigned to the robot when an obstacle was at a distance $r_i = 0.335$.
- The parameter c allowed for controlling the fall of the exponential function. In this case, it was set to 5.

- The parameter r_e was the most relevant for training, as it determined the start of the safe zone in which the robot no longer received a reward for the obstacle's location.

To choose the parameter r_e, tests were conducted in an environment with static obstacles, taking into account that the total reward was the sum of the avoidance and closeness rewards:

$$\text{Reward} = \text{Reward\_avoidance} + \text{Reward\_closeness}. \tag{3.5}$$

As shown in Table 3.3, as the value of r_e decreased, the mean collision and average collision increased. This was because the robot would tend to navigate closer to obstacles, increasing the probability of collision. Therefore, the robot would have a lower ability to evade obstacles and would reach the goal instead of being stuck in an attempt to avoid an obstacle. The optimal value according to the table was 0.8, because the mean collision converged to the mean with $r_e = 0.1$, and the success rate was higher than that of the latter. Additionally, the average reward was higher for this case.

Table 3.3: Evaluation metrics for different values of r_e.

r_e	**Mean Reward**	**Success Value (%)**	**Mean Collision**
0.6	197.14	98.10	0.085
0.8	200.58	97.50	0.063
1	196.47	96.40	0.062

3.3.4 DRL Algorithm

In alignment with Figure 2.14, widely recognized algorithms such as DQN, PPO, and SAC were selected for each category of model-free models (value-based, policy-based, actor–critic).

- Deep Q Network (DQN) [31]: This algorithm is based on a value function—that is, the assignment of a numerical value for each state. A deep neural network is in charge of

approximating this value, so that then the policy determines the action to be taken by means of the Equation (3.6), where the policy π chooses the action that maximizes the value function $Q(s, a)$:

$$\pi(s) = \arg\max_{a} Q(s, a). \tag{3.6}$$

DQN is an off-policy algorithm, because during training it employs a policy that is different from the optimal policy it is trying to estimate, because, first of all, in order to solve the problem of divergence generated by non-stationary targets, two instances of the neural-network weights are created. In this way, there is a network focused on saving an objective for multiple iterations and there are accumulated-experience data that allow us to treat the optimization problem as supervised learning.

- Soft Actor–Critic (SAC) [113]: This is an algorithm that belongs to the actor–critic category, whereby it is based on the estimation of a policy and a value function. Like DQN, SAC is also an off-policy algorithm, because it employs experiences from a behavior-focused policy that is different from the one used for optimization. SAC is well known for its ability to handle continuous action spaces, which is critical in environments where actions are not limited to discrete choices. In SAC, a deep neural network is used to model the Q function, which assigns a numerical value to each state-action.

  However, unlike DQN, SAC seeks to learn stochastic policies through the inclusion of an entropy term in the loss function. What this means is that the policy is not a deterministic function that maps states directly to actions, but rather produces probability distributions over actions. This component allows for randomness in actions and, thus, greater exploration in the environment. Consequently, it is important to emphasize that the agent's objective is not limited to maximizing only the total expected reward, but also entropy. Thus, diversity in the agent's behavior is promoted, in parallel to the optimization of the

objective function.

- Proximal Policy Optimization (PPO) [40]: An algorithm that focuses on improving policies in sequential decision-making environments. Unlike the off-policy approaches discussed above, PPO belongs to the on-policy category, which means that it learns directly from the policy it is executing in the current environment.

  PPO, instead of approximating a value function, as in DQN, focuses on direct policy optimization. The goal is to find a policy that maximizes the cumulative reward over time, taking into account efficient exploration and exploitation. PPO addresses the policy-optimization problem in a more stable manner by limiting the policy updates at each iteration. This is achieved by imposing a constraint on the magnitude of policy change, which avoids drastic updates that could lead to learning instability.

3.4 Training scheme

One of the main problems with using algorithms based on artificial intelligence is the time it takes to train them. In this context, because there must be a great similarity between the training scenario and the real one, it is often necessary to use 3D simulators with high computational cost. This, added to the fact that generally the number of episodes in reinforcement learning training is high, means that a long period is required to obtain functional models. Therefore, the following paragraphs explain the training routine used, which consists of a 2D and a 3D simulator, to obtain a preliminary trained model and fine-tune it, respectively.

3.4.1 2D Simulator

The Arena2D framework [111] is a 2D environment that allows the simulation of physical and dynamic parameters of a real environment. For the present work, the parameters of the robot

footprint shown in Figure 3.17 were modified, so that it resembles as closely as possible the robotic platform simulated in Gazebo. One of the most notable parameters that were modified is the laser viewing range. Due to the structure of the platform itself, the range is approximately 240°.

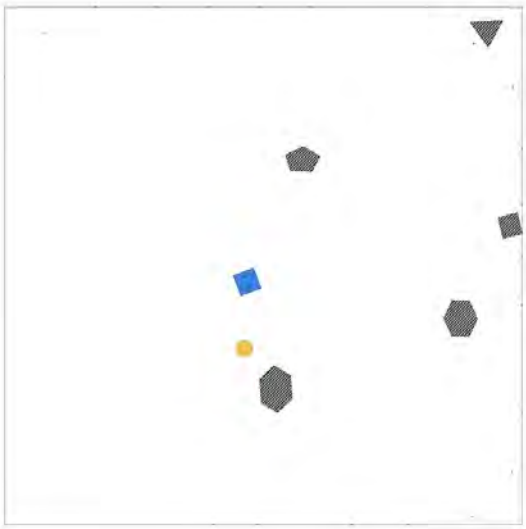

Figure 3.17: 2D simulator graphical interface.

3.4.2 Transfer learning in 3D Simulator

The $openAI\_ros$ package [114] consists of various components as can be seen in the architecture of Figure 3.18. Many of them are already implemented to directly code the reinforcement learning algorithm. However, because it has its own robotic platform, the training algorithm, robot environment and environmental task have been implemented taking into account similar parameters to the 2D simulator training. It should be noted that the GazeboEnvironment component provides the connection with the Gazebo simulator, so it was not necessary to modify it.

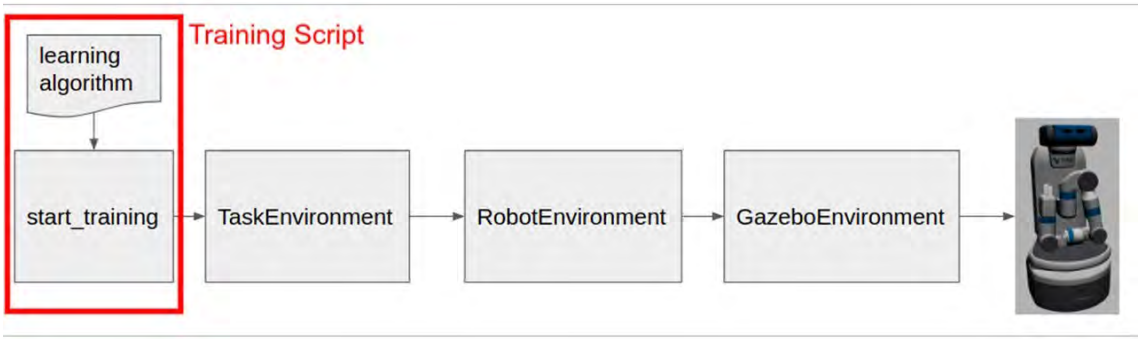

Figure 3.18: $OpenAI\_ros$ package architecture.

Source: [114]

Although arena2D provides an environment very similar to a real one, it is not exactly the same. Mainly due to the noise present in the perception sensors and the environment, in general. Likewise, physical and dynamic parameters of the robot are not taken into account in a 2D projection. In that sense, $openAI\_ros$ proposes a framework that would allow loading and training a preliminary model in Gazebo trained in arena2D, thus improving the model through transfer learning. However, due to the excessive computational cost involved, for the purposes of this project it will not be used, but rather, the 2D environment will be modified so that it is as identical to the real environment as possible.

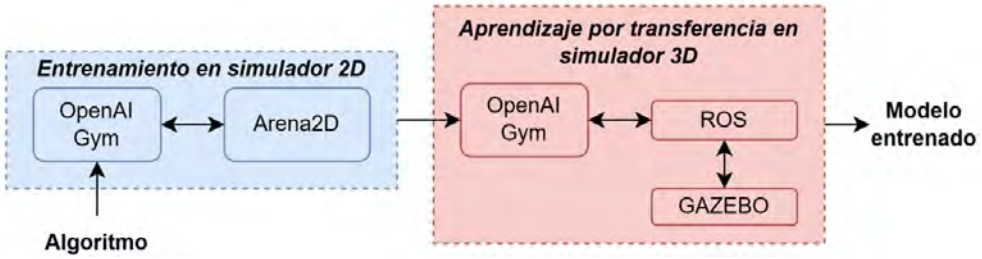

Figure 3.19: Training Architecture.

3.4.3 Training Results in static environments

In order to obtain a model with a good performance in the autonomous-navigation task in the Arena2D simulator [110], different DRL algorithms were tested. For this purpose, as a starting point, 10 static obstacles were randomly placed in a quadrangular environment of dimension 24 m, as shown in Figure 3.20.

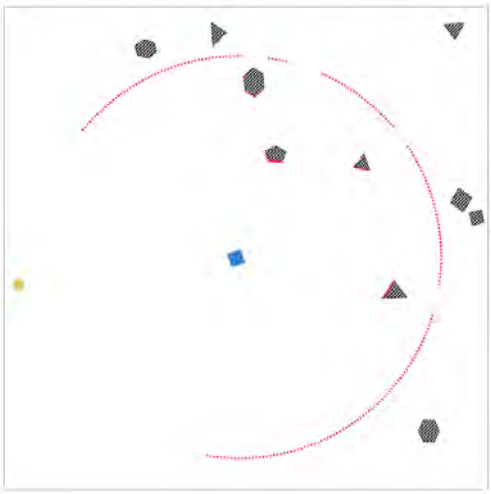

Figure 3.20: Training environment with stationary obstacles.

Additionally, in order to conduct a thorough comparison to traditional methods, it was fine-tuned the hyperparameters for each algorithm. First of all, the learning rates were customized, with DQN being set at 0.001—a notably higher value compared to the 0.0003 assigned to PPO and SAC. This decision stemmed from the significantly longer training time needed for DQN compared to the other algorithms, as can be seen in Table 3.4. The higher learning rate was selected to accelerate the training process while maintaining a balance that avoided negatively impacting performance. Secondly, the discount factor (γ) had a notable impact on performance. It was observed that a value of 0.95 yielded optimal results for both DQN and PPO, while SAC exhibited superior performance, with a discount factor of 0.99. Finally, another parameter that held significance for DQN but not for the other algorithms was the exploration rate. To govern

the exploration–exploitation trade-off, a value of 0.1 was assigned.

Table 3.4: Evaluation results for stationary environments.

Algorithm	Action Space	Mean Reward	Success Rate (%)	Mean Collision	Training Time
SAC	Continuous	200.58	97.50	0.063	1 h 4 min
PPO	Discrete	170.58	97.60	0.372	1 h 23 min
	Continuous	151.43	98.00	0.483	1 h 24 min
DQN	Discrete	170.29	84.00	1.360	1 d 15 h 58 min
	Continuous	161.79	80.90	0.089	1 d 11 h 47 min

In Table 3.4, various evaluation metrics for the algorithms in the previously described environment are presented. It is worth noting that only for the DQN algorithm did the choice of a continuous over a discrete action space result in significant improvements. In general, DQN had a longer training time, which may have been due to the approximation it employed, based on approximating a Q-function, which can require a large number of iterations to converge to the optimal policy. Furthermore, SAC was the algorithm that showed the best combined results. In fact, SAC (off-policy) was more efficient in its samples than the PPO algorithm (on-policy). Consequently, it obtained a much lower collision rate, a lower training time, a higher average reward and had a slightly lower success rate than PPO.

3.4.4 Training Results in dynamic environments

SAC algorithm obtained the best performance in stationary environments. For this reason it will be used for training in dynamic environments through transfer learning. The simulator offers the possibility of including different types of obstacles, of which those similar to passers-by were chosen due to the application of this project (see Figure 3.21). For training, it was considered that the obstacle has a speed range of 0.2 to 0.4 m/s. The results are presented in Table 3.5.

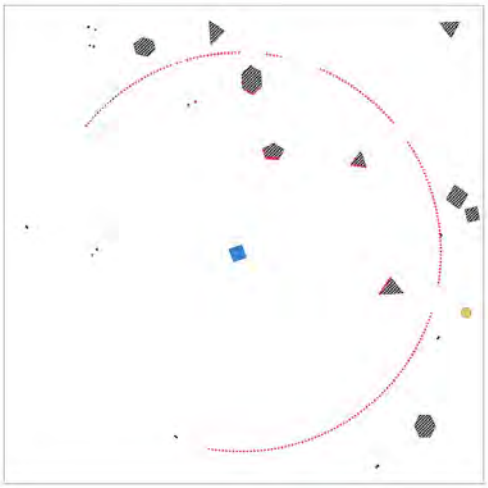

Figure 3.21: Training environment with dynamic obstacles.

Table 3.5: Evaluation results for dynamic environments.

Algorithm	Action Space	Mean Reward	Success Rate (%)	Mean Collision	Training Time
SAC	Continuous	206.306	99.10	0.574	2 h 50 min

3.4.5 Discussion

- DQN has a high training time
- On average SAC presents better performance. This algorithm (off-policy) is more efficient in its samples than the PPO (on-policy) algorithm:
  - Obtains a much lower collision rate
  - You get a higher average reward
  - Obtains a slightly lower success rate than the PPO
- For the present application the choice of a continuous or discrete action space is immaterial

Chapter IV

Evaluation of algorithms on virtual environments

In the following chapter, a comprehensive comparative analysis was conducted, to juxtapose conventional autonomous-navigation algorithms with Reinforcement-Learning-(RL) methodologies. In this context, it is selected DWA and TEB as the conventional local-planning methods of choice. This choice stemmed from the fact that DWA has been widely used as a benchmark in prior studies [115, 116], serving as a reference point for evaluating and understanding the improvements and challenges offered by novel approaches. Conversely, TEB, also featured in previous comparison studies [115, 116], has emerged as a significant enhancer of traditional-navigation algorithms, particularly in dynamic and non-stationary environments. Furthermore, TEB demonstrates superior computational efficiency compared to DWA [117], making it a suitable candidate for comparison to the most robust RL approaches available.

From the RL side, CADRL was chosen from among the other DRL approaches [58, 118], because it proposes a navigation system focused on obstacle avoidance without assuming any rules in the behavior of the agents. Furthermore, SAC was chosen after testing and performing better among different DRL algorithms, as can be seen in Table 3.4. SAC merited inclusion in

the comparison for its remarkable ability to function as a mapless algorithm while achieving comparable performance to map-based alternatives across several critical metrics.

4.1 Evaluation Methodology

4.1.1 Review and Definition of the Evaluation Environment

In order to evaluate the navigation algorithms, an environment and performance evaluation methodology was designed. To avoid the randomness of the results arising from taking into account different departure and arrival points for each iteration, routes between points A and B in both directions were considered, as shown in Figure 4.1. Ten attempts for each combination of parameters in the simulation were performed.

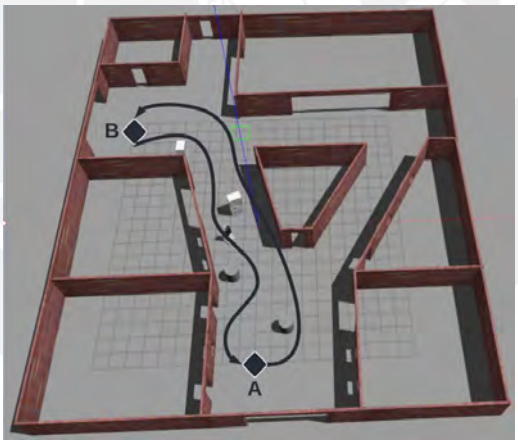

Figure 4.1: Test environment.

On the other hand, it should be noted that three types of environments were designed: empty, with static obstacles and with dynamic obstacles. For the empty environment, the world consisted only of passages and walls. In the environment with static obstacles, the number of obstacles was constant and equal to four. Finally, for the environment with dynamic obstacles, different parameters were varied, such as obstacle speed, type of movement, number of

obstacles, etc.

The dynamic environment consisted of actors emulating the movement of passers-by, based on the social movements used in [119]. In that sense, the actors had three types of possible movements: parallel to the robot's movement, perpendicular to the robot's movement and random, as illustrated in Figure 4.2, where eight actors, each with a specified behavior, are shown in the simulated environment.

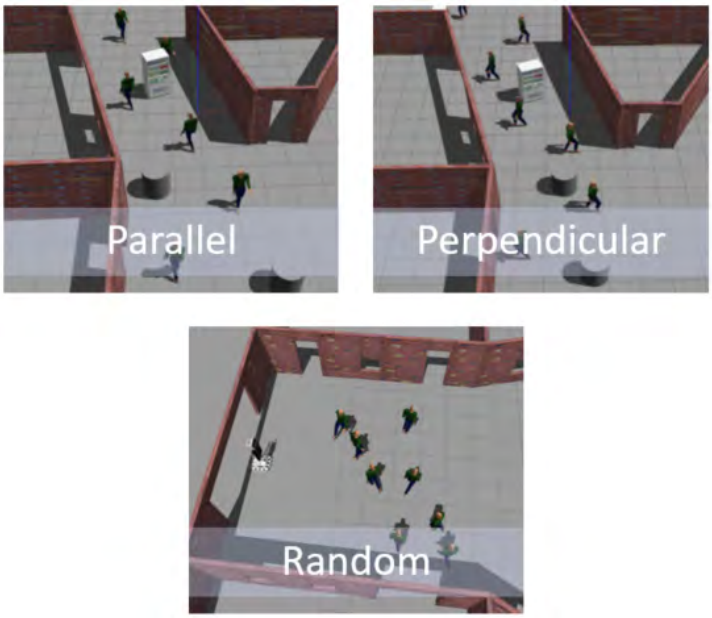

Figure 4.2: Movement types of dynamic obstacles.

In each iteration, three parameters were varied fundamentally: type of motion, velocity and number of dynamic actors. As shown in Figure 4.3, the velocities of the actors were considered to be lower than the linear velocity of the robot, from 0.2 to 0.4 m/s. Likewise, the number of dynamic actors was chosen considering the dimensions of the designed world, in which four or eight passers-by were placed. It should be noted that the 10 evaluation iterations were for each of the possible combinations of the parameters of the three categories, so that a total of 180 samples were taken for each algorithm to be evaluated.

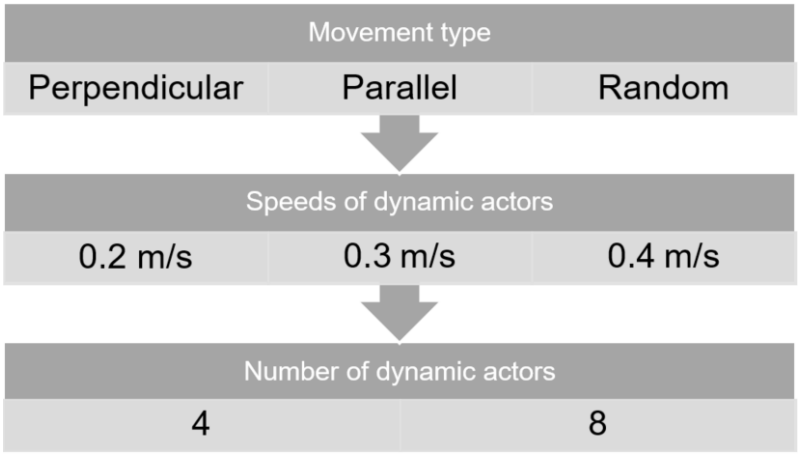

Figure 4.3: Variable parameters of the dynamic actors.

4.1.2 Definition of Evaluation Metrics

The evaluation metrics for the comparison of the different autonomous-navigation algorithms took into account trajectory and obstacle-avoidance efficiency. These metrics are shown below, with their respective units:

- Number of collisions;
- Safety score (%);
- Navigation time (s);
- Route length (m).

It is worth mentioning that, for navigation time, a route was considered unsuccessful if the robot became stuck in a recovery behavior for more than 15 seconds. Also, the metric used to evaluate the correct obstacle avoidance was the Safety score [117]. For this purpose, the total time spent by the robot on the route was divided into a given number of steps, the score was calculated by dividing the sum of the periods of time in which no pedestrian came within a

distance $2 \times r_{\text{robot}}$ to the center of the robot, divided by the total number of steps, as shown in Equation (4.1):

$$\text{Safety}_{\text{score}} = \frac{\sum_{i=1}^{k} T_i}{T}, \tag{4.1}$$

where r_{robot} represents an approximation of the robot radius, considering the platform base as circular, T_i represents the ith time period during which the robot was not surrounded by any obstacle and T represents the total navigation time.

4.2 Performance Analysis of Computational Frameworks

A performance analysis was performed by evaluating each of the selected algorithms considering no-dynamic and dynamic obstacles. For the analysis was used the procedure describe in the previous subsection.

4.2.1 Performance with Non-Dynamic Obstacles

The results for scenarios without dynamic actors were observed. For this purpose, the designed map was considered totally empty or with four stationary objects arranged along the robot's path. It should be noted that 10 runs were performed for each type of algorithm and type of environment. It should also be noted that all the algorithms obtained 100% accuracy, i.e., they reached the goal in all the executions. From Table 4.1, graphs were generated to facilitate the comparison of the algorithms, as shown in Figure 4.4. It is important to highlight that in Figures 4.4–4.7 the metric values have been normalized relative to the maximum value. This normalization allowed that a higher bar corresponded to superior performance in the respective metric. From this plot, the following conclusions were drawn:

- TEB provided a more optimal route on average, in terms of the navigation time.

- SAC had a less safe route on average, as it was the only algorithm that had a non-zero safety score for stationary environments.
- CADRL provided a more optimal route, in terms of distance traveled.
- All the algorithms provided collision-free routes.

Table 4.1: Results with non-dynamic obstacles.

	Scenario	**Empty Map**	**Static Obstacles**
Number of collisions	DWA	**0.0**	**0.0**
	TEB		
	CADRL		
	SAC		
Safety score (%)	DWA	**0.0**	**0.0**
	TEB		
	CADRL		
	SAC		1.57
Path length (m)	DWA	22.90	24.44
	TEB	23.24	24.85
	CADRL	**22.27**	**24.05**
	SAC	24.03	26.50
Navigation time (s)	DWA	64.70	69.51
	TEB	**60.07**	**64.95**
	CADRL	60.58	68.47
	SAC	63.05	71.08

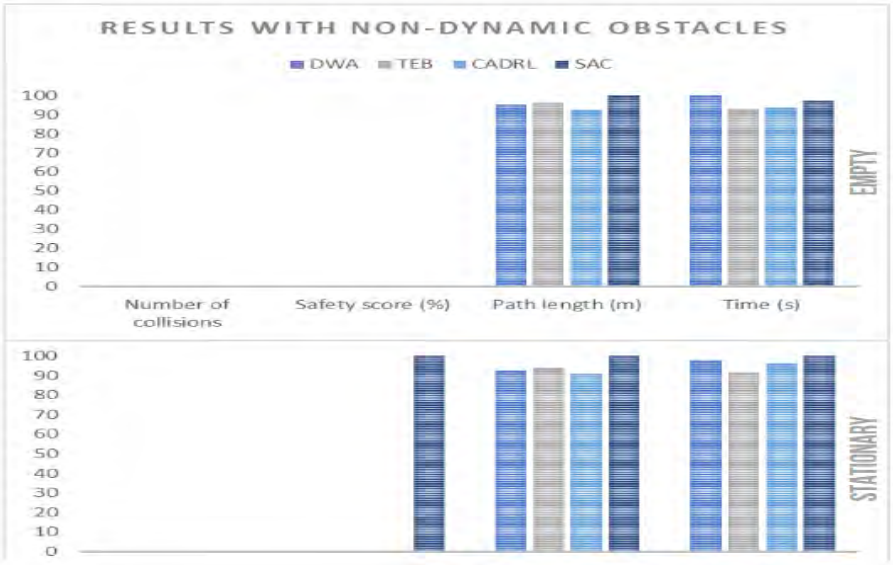

Figure 4.4: Plot for non-dynamic obstacles.

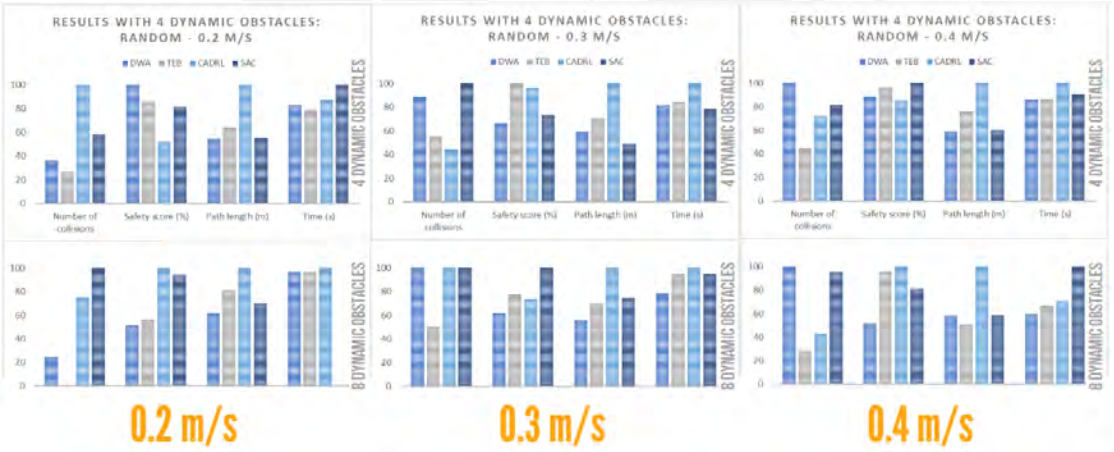

Figure 4.5: Plot for crossing social environment.

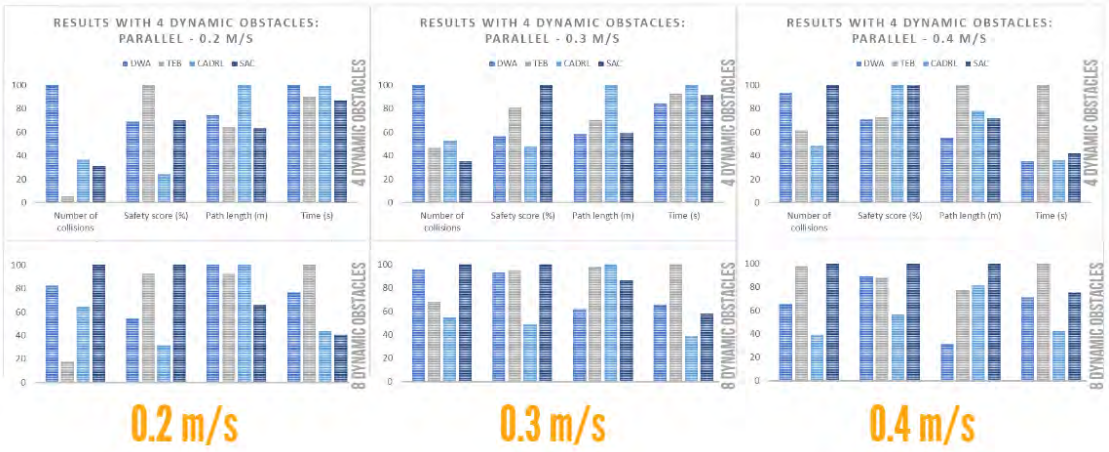

Figure 4.6: Plot for parallel social environment.

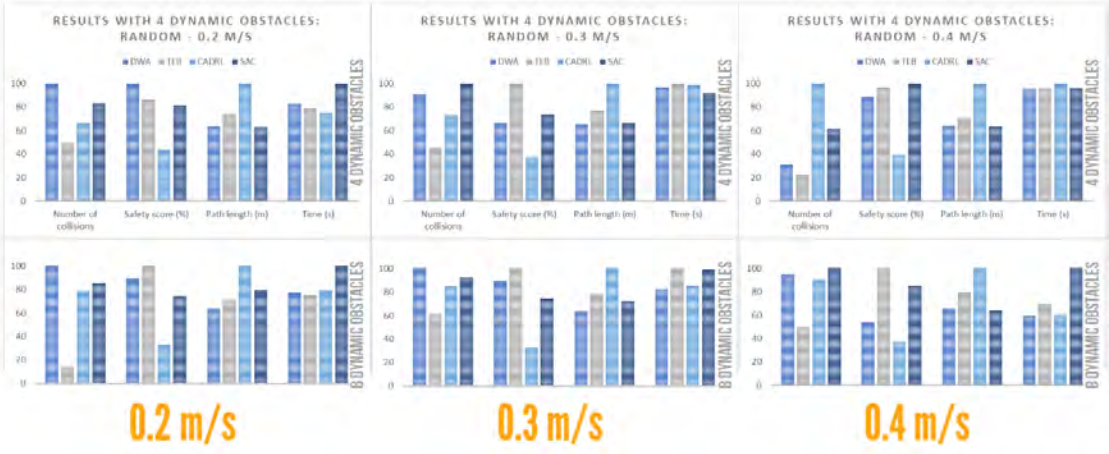

Figure 4.7: Plot for random social environment.

4.2.2 Performance with Dynamic Obstacles

Performance with Dynamic Obstacles: Crossing

The results for dynamic actors with a motion type crossing or perpendicular to the robot path are shown below. It should be noted that 10 runs were performed for each type of dynamic configuration and that the average of the metrics was calculated. Likewise, the TEB, CADRL and SAC algorithms obtained 100% accuracy, i.e., they managed to reach the goal. However,

the DWA obtained an accuracy of 90% when the dynamic objects had velocities of 0.4 m/s, both for four and eight agents. From Table 4.2, graphs were generated to facilitate the comparison of the algorithms, as shown in Figure 4.5. The following conclusions were drawn, based on obstacle concentration:

- Low obstacle concentration:
  - DWA provided the second-shortest route with the highest temporal efficiency. However, its collision rate was higher for obstacles with a velocity of 0.3 m/s.
  - TEB provided the route with the least number of collisions, away from obstacles (high safety score). It also had high temporal efficiency.
  - CADRL, on average, provided the longest route with low temporal efficiency. Additionally, it exhibited a high collision rate.
  - SAC, on average, had the lowest safety score and the highest number of collisions but with high temporal efficiency.
- High obstacle concentration:
  - DWA provided high temporal efficiency and, on average, the shortest route but at the cost of a significant increase in the collision rate.
  - TEB provided the route with the least number of collisions, away from obstacles (high safety score). However, it significantly reduced its temporal efficiency compared to low obstacle concentration.
  - CADRL improved its collision rate and provided the second route with the lowest percentage. However, it offered the longest path and a low safety score.
  - SAC provided a route with high navigation time efficiency and a short route length. However, it had the highest collision rate and a fairly low safety score.

Table 4.2: Dynamic Scenario 1—Perpendicular movement.

	Collisions Number	Safety Score (%)	Navigation Time (s)	Path Length (m)	Collisions Number	Safety Score (%)	Navigation Time (s)	Path Length (m)
$v_{\text{obs}} = 0.2$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	0.7	7.41	80.78	27.10	2.8	18.01	**85.90**	**27.09**
TEB	**0.3**	**5.87**	**77.70**	29.13	**0.3**	**5.87**	91.00	32.27
CADRL	0.9	25.30	90.16	36.85	1.2	24.03	90.40	42.90
SAC	1.4	23.44	78.41	**26.89**	1.5	25.44	90.14	28.82
$v_{\text{obs}} = 0.3$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	2.1	9.30	79.72	26.34	2.8	21.61	83.02	**25.90**
TEB	**0.5**	**7.33**	81.11	30.08	**1.4**	**15.33**	106.24	37.39
CADRL	1.4	22.60	87.27	42.98	1.8	29.65	90.25	43.05
SAC	2.0	22.97	**73.13**	**26.06**	3.9	35.00	**79.11**	27.30
$v_{\text{obs}} = 0.4$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	1.3	9.45	77.06	26.34	2.5	23.97	89.43	27.98
TEB	**0.9**	**6.94**	82.93	30.48	2.2	**14.59**	107.20	36.93
CADRL	2.0	24.71	90.69	43.02	**1.9**	25.08	92.20	43.25
SAC	1.7	13.68	**73.11**	**26.04**	3.5	33.13	**76.80**	**26.80**

Performance with Dynamic Obstacles: Parallel

The results for dynamic actors with a type of movement in parallel to or in the same direction of movement as the robot are shown below. It should be noted that 10 runs were performed for each type of dynamic configuration and that the average of the metrics was calculated. Likewise, the DWA and CADRL algorithms obtained 100% accuracy, i.e., they managed to reach the goal in all runs at the cost of an increase in the number of collisions. On the other hand, the TEB also obtained 100% accuracy with four dynamic agents; however, when this value increases to eight, it obtained an accuracy of 90% for speeds of 0.2 and 0.3 m/s and 80% for speeds of 0.4 m/s, while the SAC obtained an accuracy of 100%, except for speeds of 0.4 m/s, where it obtained 90%.

From Table 4.3, graphs were generated to facilitate the comparison of the algorithms, as shown in Figure 4.6. The following conclusions were drawn, based on obstacle concentration:

- Low obstacle concentration:

  - DWA provided, on average, the route with the highest number of collisions. However, it had high temporal efficiency, similar to the shortest navigation route.
  - TEB provided a route with the least number of collisions at low speeds. However, the number of collisions and navigation time increased, while the safety score decreased considerably at high speeds.
  - SAC had the highest number of collisions only at very high speeds; otherwise, it performed well. It also had the route with the best average safety score.
  - CADRL had the least number of collisions at high speeds but a low average for low speeds. However, its safety score was the lowest and it provided a long route.

- High obstacle concentration:
  - DWA provided the route with the shortest path length. However, its average collision rate increased significantly compared to a low-obstacle-concentration environment.
  - TEB provided a route with a low number of collisions, away from obstacles, except at high speeds. However, it reduced its temporal efficiency.
  - CADRL provided the route with the least number of collisions and better temporal efficiency. However, it had a low safety score.
  - SAC provided the route with the highest safety score. However, it had, on average, the route with the highest number of collisions.

Table 4.3: Dynamic Scenario 2—Parallel movement.

	Collisions Number	Safety Score (%)	Navigation Time (s)	Path Length (m)	Collisions Number	Safety Score (%)	Navigation Time (s)	Path Length (m)
$v_{obs} = 0.2$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	1.9	8.9	91.5	32.16	1.4	17.14	156.04	**26.50**
TEB	**0.1**	**6.15**	82.64	27.79	**0.3**	10.13	203.09	39.80
CADRL	0.7	25.00	90.70	43.17	1.1	29.67	89.19	43.13
SAC	0.6	8.77	**79.55**	**27.34**	1.7	**9.39**	**82.31**	28.33
$v_{obs} = 0.3$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	1.7	18.25	**74.36**	**25.38**	2.1	15.78	142.50	**26.73**
TEB	0.8	12.65	82.21	30.34	1.5	15.47	215.74	42.41
CADRL	0.9	21.71	88.26	43.06	**1.2**	30.11	**83.25**	43.10
SAC	**0.6**	**10.31**	81.43	25.70	2.2	**14.70**	125.33	37.30
$v_{obs} = 0.4$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	2.9	22.35	**83.51**	**25.97**	3.0	19.38	146.38	**26.36**
TEB	1.9	21.8	234.26	47.03	4.5	19.71	205.31	40.78
CADRL	**1.5**	15.81	85.29	36.74	**1.8**	30.64	**86.71**	43.03
SAC	3.1	**10.4**	98.24	33.84	4.6	**17.34**	154.99	52.65

Performance with Dynamic Obstacles: Random

The results for dynamic actors with a random type of movement are shown below. It should be noted that 10 runs were performed for each type of dynamic configuration and that the average of the metrics was calculated. Also, even with random motion, all the algorithms except DWA in an environment of eight dynamic obstacles obtained 100% accuracy. This was mainly due to the fact that, despite being a more unpredictable motion for the robot, it presented a greater range of motion when moving within a region and not a linear motion.

From Table 4.4, graphs were generated to facilitate the comparison of the algorithms, as shown in Figure 4.7. The following conclusions were drawn, based on obstacle concentration:

- Low obstacle concentration:
  - DWA provided the second-shortest route.
  - TEB provided the route with the least number of collisions, away from obstacles.

However, it had a long trajectory.

- CADRL provided the longest route and, on average, the lowest safety score.
- SAC provided a route with a high safety score. It also provided the shortest route.

• High obstacle concentration:

- DWA provided the shortest route. However, the number of collisions increased significantly compared to an environment with low obstacle concentration.
- TEB provided the route with the least number of collisions, away from obstacles.
- CADRL provided an efficient route, in terms of navigation time. However, it had a low safety score and the longest path length.
- SAC had the highest average collision rate and was the least efficient, in terms of navigation time.

Table 4.4: Dynamic Scenario 3—Random movement.

	Collisions Number	**Safety Score (%)**	**Navigation Time (s)**	**Path Length (m)**	**Collisions Number**	**Safety Score (%)**	**Navigation Time (s)**	**Path Length (m)**
$v_{obs} = 0.2$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	0.6	**6.85**	89.16	26.97	1.4	10.97	86.82	**27.14**
TEB	**0.3**	7.95	84.88	31.63	**0.2**	**9.78**	**84.46**	30.73
CADRL	0.4	15.66	**80.96**	42.74	1.1	29.67	89.19	43.13
SAC	0.5	8.41	107.15	**26.86**	1.2	13.11	112.33	34.17
$v_{obs} = 0.3$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	1	10.99	83.43	**26.62**	1.3	10.97	**80.19**	**27.16**
TEB	**0.5**	**7.29**	86.32	31.56	**0.8**	**9.78**	96.92	33.78
CADRL	0.8	19.38	85.22	40.84	1.1	30.10	83.24	43.10
SAC	1.1	9.92	**79.56**	27.23	1.2	13.11	96	31.06
$v_{obs} = 0.4$ m/s		4 Dynamic Obstacles				8 Dynamic Obstacles		
DWA	0.4	9.62	**80.54**	27.52	1.9	21.34	**85.62**	27.86
TEB	**0.3**	**8.84**	81.17	30.29	**1.0**	11.50	99.54	34.34
CADRL	1.3	21.60	84.68	42.90	1.8	30.64	86.71	43.03
SAC	0.8	8.53	81.04	**27.09**	2.0	13.49	143.53	**27.59**

4.3 Statistical Analysis

The data obtained from the previous experiments was comprised into dataset for an statistical analysis, which was performed by using RStudio.

4.3.1 Data Description

The dataset comprises several variables capturing the performance of autonomous navigation algorithms in different environments. The key variables include:

- **Environment**: Categorical variable indicating whether the environment is *Static* or *Dynamic*.
- **Obstacles**: Categorical variable with levels *Empty*, *Fixed*, *4 actors*, and *8 actors*.
- **Movement Type**: Categorical variable indicating the type of movement (*Random*, *Parallel*, *Crossing*).
- **Movement Speed**: Categorical variable with levels *0.2*, *0.3*, *0.4*.
- **Algorithm**: Categorical variable indicating the navigation algorithm used (*DWA*, *TEB*, *SAC*, *CADRL*).
- **Number of Collisions**: Numerical variable indicating the count of collisions.
- **EGO_SCORE**: Numerical variable representing the ego score.
- **Total Distance**: Numerical variable indicating the total distance traveled.
- **Total Time**: Numerical variable indicating the total time taken.

Descriptive statistics for the numerical variables are summarized in Table 4.5.

Table 4.5: Descriptive Statistics for Numerical Variables

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Number of Collisions	0.00	0.00	1.00	1.32	2.00	14.00
EGO_SCORE	0.00	6.38	13.31	15.10	22.72	65.72
Total Distance	0.00	26.12	28.58	32.35	42.57	221.58
Total Time	55.87	74.07	82.21	92.85	92.95	668.10

4.3.2 Statistical Tests

To evaluate the performance differences across various conditions and algorithms, several statistical tests were performed:

- A Welch two-sample t-test to compare **Total Time** between *Static* and *Dynamic* environments.
- ANOVA tests to compare **Total Time** and **Number of Collisions** across different *Algorithms*.
- Pearson correlation analysis to examine the relationships between numerical variables.
- Linear regression models to analyze the influence of *Environment*, *Obstacles*, *Movement Type*, *Movement Speed*, and *Algorithm* on **Total Time** and **Number of Collisions**.

4.3.3 Results and Discussion

Descriptive Statistics: The analysis of descriptive statistics (Table 4.5) reveals that the number of collisions ranges from 0 to 14, with a mean of 1.32. The total time varies widely, from 55.87 to 668.10 seconds, with an average of 92.85 seconds. These statistics indicate substantial variability in the performance metrics.

T-test for Total Time: The t-test results (Table 4.6) show a significant difference in total time between static and dynamic environments ($t = 13.598$, $p < 2.2 \times 10^{-16}$). The mean

total time in dynamic environments (95.99 seconds) is significantly higher than in static environments (65.25 seconds), suggesting that dynamic environments pose more challenges for the navigation algorithms.

Table 4.6: T-test for Total Time between Static and Dynamic Environments

	Mean (Dynamic)	Mean (Static)
Total Time	95.99	65.25

ANOVA for Total Time and Number of Collisions: The ANOVA results (Table 4.7 and Table 4.8) indicate significant differences in both total time ($F = 7.517$, $p < 0.001$) and the number of collisions ($F = 13.43$, $p < 0.001$) across different algorithms. These findings suggest that algorithm choice significantly impacts both performance metrics.

Table 4.7: ANOVA for Total Time by Algorithm

	F-value (p-value)
Total Time	7.517 ($p = 5.79 \times 10^{-5}$)

Table 4.8: ANOVA for Number of Collisions by Algorithm

	F-value (p-value)
Number of Collisions	13.43 ($p = 1.48 \times 10^{-8}$)

Correlation Analysis: The Pearson correlation matrix (Table 4.9) shows that the number of collisions has moderate positive correlations with ego score ($r = 0.45$) and total time ($r = 0.38$). Total distance and total time are strongly correlated ($r = 0.66$), indicating that longer distances result in longer times.

Table 4.9: Pearson Correlation Matrix

	Number of Collisions	EGO_SCORE	Total Distance	Total Time
Number of Collisions	1.00	0.45	0.25	0.38
EGO_SCORE	0.45	1.00	0.29	0.10
Total Distance	0.25	0.29	1.00	0.66
Total Time	0.38	0.10	0.66	1.00

Linear Regression Analysis: The regression models (Table 4.10 and Table 4.11) reveal the impact of various factors on total time and number of collisions. Significant predictors for total time include obstacles with 8 actors (increasing total time by 23.83 seconds) and movement type parallel (increasing total time by 24.22 seconds). For the number of collisions, significant predictors include static environment (decreasing collisions by 1.16), obstacles with 8 actors (increasing collisions by 0.74), and lower movement speed (decreasing collisions by 1.10 for speed 0.2).

Table 4.10: Linear Regression for Total Time

	Estimate	p-value
(Intercept)	71.33	$< 2e-16$
EnvironmentStatic	-10.47	0.1285
Obstacles8 actors	23.83	$< 2e-16$
ObstaclesEmpty	-6.25	0.4553
Movement_typeCrossing	-3.52	0.3083
Movement_typeParallel	24.22	4.69×10^{-12}
Movement_speed0.2	-2.28	0.5094
Movement_speed0.3	-3.39	0.3269
AlgorithmDWA	6.48	0.0858
AlgorithmSAC	5.05	0.1831
AlgorithmTEB	19.05	5.10×10^{-7}

Table 4.11: Linear Regression for Number of Collisions

	Estimate	p-value
(Intercept)	0.98	2.62×10^{-13}
EnvironmentStatic	-1.16	1.81×10^{-7}
Obstacles8 actors	0.74	7.63×10^{-16}
ObstaclesEmpty	0.004	0.9874
Movement_typeCrossing	0.73	5.52×10^{-11}
Movement_typeParallel	0.77	5.80×10^{-12}
Movement_speed0.2	-1.10	$< 2e-16$
Movement_speed0.3	-0.57	2.35×10^{-7}
AlgorithmDWA	0.46	0.000153
AlgorithmSAC	0.50	3.50×10^{-5}
AlgorithmTEB	-0.27	0.0243

Conclusions

This thesis presented an evaluation of autonomous navigation algorithms on a differential robot equipped with depth camera and laser range finder for non-stationary indoor environments. The methodology followed involved a detailed review of autonomous navigation algorithms, the implementation of these algorithms on a virtual framework, the design of a differential robot equipped with specialized sensors for navigation, and the evaluation of the algorithms using a virtual environment to determine their performance with non-stationary elements.

As part of the study, a review of indoor-autonomous-navigation algorithms for dynamic scenarios applied to mobile robots was performed. Based on a detailed review, it was selected two traditional-navigation algorithms (DWA and TEB) and two DRL-based algorithms (CADRL and SAC). On the other hand, the differential robot platform includes two wheels coupled to DC motors to generate enough traction to move the robot. These traction wheels are complemented with four castor wheels to distribute the weight of the robot. The robot platform is equipped with one depth camera in the front, one 360° Lidar sensor in the front and an array of infrared and sonar sensors for the surroundings of the robot base.

A virtual robotic platform and a virtual environment was used to evaluate their performance and identify their advantages and disadvantages. Based on the results and performance of each algorithm, its possible to conclude the following:

- DWA provides a route with a high percentage of collisions, mainly due to its inability to generate reverse motion. However, it has, on average, good temporal efficiency and path

length.

- TEB provides, on average, the route with the least amount of collisions and away from obstacles at low speeds and concentrations. However, at high speeds or concentrations, evasion and, consequently, temporal efficiency worsens.

- CADRL provides good obstacle avoidance in concentrated and fast environments, but at the cost of low temporal and routing efficiency.

- SAC provides a route with a high percentage of collisions, mainly due to the fact that it is a map-less algorithm. However, it has, on average, good temporal efficiency and path length.

Moreover, it is important to take into consideration the characteristics of the environment in which the robot must navigate. When navigating through dynamic environments the behavior of the actor can be different. For this reason, some conclusions can be described based on the following results:

- For the exposed robot configuration, a dynamic environment where the movement of the actors is parallel to or in the same direction as the movement of the robot is more complicated to navigate. This is due to the limited range of vision, due to the dimensions of the robot, mainly in reverse movements.

- The most relevant dynamic parameter for social environments is the number of dynamic actors, as its variation entails a considerable reduction in the performance of the navigation algorithms.

- On average, metrics such as the number of obstacles, navigation time, path length and safety score increase as the number of dynamic agents and their speed increases.

As a result of this study, all the evaluated algorithms allowed the robot to navigate autonomously through a dynamic scenario. Moreover, the following recommendations can be stated for the usage of algorithms for autonomous navigation of ground-mobile robots in dynamic scenarios:

- When the information of the map is not available for the robot, it must be used with a mapless-DRL-based algorithm, such as SAC.
- If the map information is available and it is required to reduce the number of collisions even if the movement is slower, then a traditional algorithm, such as TEB, can be used.
- If the map information is available and it is required that the movement must be fast, then a traditional algorithm, such as DWA, can be used.
- If the map information is available and it is required that the trajectory must be the shortest, then a DRL-based algorithm, such as CADRL, can be used.

Based on the previous conclusions, some cases of practical applications for indoor dynamic environments are presented, in order to determine the suggested algorithm. For any of the cases, if the information from the map is not available, the best option is to use a mapless-DRL-based algorithm:

- CASE 1: Robot for delivery of products in indoor environments (such as hospitals or offices): This type of application usually prioritizes the time it takes to deliver the products, for which it is recommended to use a traditional algorithm, such as DWA, or a DRL-based, such as CADRL, that uses shorter trajectories.
- CASE 2: Robot for marketing and interaction with users (in malls, universities, events, etc.): For any type of HRI applications, such as this one, it is recommended to use an algorithm that prioritizes the number of collisions, such as TEB, which is a traditional algorithm.

- CASE 3: Robot for security and inspection (in malls, offices, industrial plants or warehouses): This type of application usually prioritizes the shorter trajectories, in order to increase the robot´s autonomy. Hence, a DRL-based algorithm, such as CADRL, could be the best alternative.

This study was carried out in a virtual environment, in order to consider the different characteristics of the dynamic actors and their possible behaviors, which would not have been possible to do in a real environment. Through analysis in the virtual environment, it has been possible to evaluate multiple cases that could occur in a real environment. This will allow these results to demonstrate the functionality of the algorithms through tests in real environments, such as malls, hospitals, offices or industrial plants.

As part of the future work for this research, the author will implement the evaluated algorithms, using the robot in real use during the simulations and will evaluate its performance in navigating a mall and an office. Moreover, the evaluation will be performed by modifying the characteristic of each scenario, in order to evaluate the effect of generalization, especially with the DRL-based algorithms.

Bibliography

[1] B. Siciliano, O. Khatib, and T. Kröger, *Springer handbook of robotics*, vol. 200. Springer, 2008.

[2] J. Bruce, N. Sünderhauf, P. Mirowski, R. Hadsell, and M. Milford, "One-shot reinforcement learning for robot navigation with interactive replay," *arXiv preprint arXiv:1711.10137*, 2017.

[3] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 international conference on robotics and automation (ICRA)*, pp. 6015–6022, IEEE, 2019.

[4] G. Chen, L. Pan, P. Xu, Z. Wang, P. Wu, J. Ji, X. Chen, *et al.*, "Robot navigation with map-based deep reinforcement learning," in *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pp. 1–6, IEEE, 2020.

[5] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 5129–5136, IEEE, 2018.

[6] Y. Kato, K. Kamiyama, and K. Morioka, "Autonomous robot navigation system with learning based on deep q-network and topological maps," in *2017 IEEE/SICE International Symposium on System Integration (SII)*, pp. 1040–1046, IEEE, 2017.

[7] J. Kulhánek, E. Derner, T. De Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," in *2019 european conference on mobile robots (ECMR)*, pp. 1–8, IEEE, 2019.

[8] J. Lin, X. Yang, P. Zheng, and H. Cheng, "End-to-end decentralized multi-robot navigation in unknown complex environments via deep reinforcement learning," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 2493–2500, IEEE, 2019.

[9] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5671–5677, IEEE, 2020.

[10] E. Marchesini and A. Farinelli, "Discrete deep reinforcement learning for mapless navigation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10688–10694, IEEE, 2020.

[11] H. Quan, Y. Li, and Y. Zhang, "A novel mobile robot navigation method based on deep reinforcement learning," *International Journal of Advanced Robotic Systems*, vol. 17, no. 3, p. 1729881420921672, 2020.

[12] X. Ruan, D. Ren, X. Zhu, and J. Huang, "Mobile robot navigation based on deep reinforcement learning," in *2019 Chinese control and decision conference (CCDC)*, pp. 6174–6178, IEEE, 2019.

[13] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments," *arXiv preprint arXiv:2005.13857*, 2020.

[14] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 31–36, IEEE, 2017.

[15] P. Yue, J. Xin, H. Zhao, D. Liu, M. Shan, and J. Zhang, "Experimental research on deep reinforcement learning in autonomous navigation of mobile robot," in *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1612–1616, IEEE, 2019.

[16] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2371–2378, IEEE, 2017.

[17] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364, IEEE, 2017.

[18] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 6, pp. 2064–2076, 2019.

[19] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[20] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[21] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[22] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[23] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, pp. 50–56, 2016.

[24] Q. Shi, S. Zhao, X. Cui, M. Lu, and M. Jia, "Anchor self-localization algorithm based on uwb ranging and inertial measurements," *Tsinghua Science and Technology*, vol. 24, no. 6, pp. 728–737, 2019.

[25] W. Rone and P. Ben-Tzvi, "Mapping, localization and motion planning in mobile multi-robotic systems," *Robotica*, vol. 31, no. 1, pp. 1–23, 2013.

[26] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*, pp. 834–849, Springer, 2014.

[27] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[28] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[29] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE international symposium on safety, security, and rescue robotics*, pp. 155–160, IEEE, 2011.

[30] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[32] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil, *et al.*, "Vector-based navigation using grid-like representations in artificial agents," *Nature*, vol. 557, no. 7705, pp. 429–433, 2018.

[33] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 5113–5120, IEEE, 2018.

[34] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[35] F. Zeng, C. Wang, and S. S. Ge, "A survey on visual navigation for artificial agents with deep reinforcement learning," *IEEE Access*, vol. 8, pp. 135426–135442, 2020.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[37] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[38] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[39] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[41] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[42] Y. Kato and K. Morioka, "Autonomous robot navigation system without grid maps based on double deep q-network and rtk-gnss localization in outdoor environments," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, pp. 346–351, IEEE, 2019.

[43] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2371–2378, IEEE, September 2017.

[44] X. Lei, Z. Zhang, and P. Dong, "Dynamic path planning of unknown environment based on deep reinforcement learning," *Journal of Robotics*, vol. 2018, 2018.

[45] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4423–4430, 2018.

[46] J. Choi, K. Park, M. Kim, and S. Seok, "Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 5993–6000, IEEE, May 2019.

[47] F. Leiva and J. Ruiz-del Solar, "Robust rl-based map-less local planning: Using 2d point clouds as observations," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5787–5794, 2020.

[48] H. Shi, L. Shi, M. Xu, and K. S. Hwang, "End-to-end navigation strategy with deep reinforcement learning for mobile robots," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2393–2402, 2019.

[49] K. Yokoyama and K. Morioka, "Autonomous mobile robot with simple navigation system based on deep reinforcement learning and a monocular camera," *In*, vol. 2020 IEEE/SICE International Symposium on System Integration (SII) (). IEEE, pp. 525–530, January 2020.

[50] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, others, and R. Hadsell, "Learning to navigate in complex environments." arXiv preprint, 2016.

[51] J. Oh, V. Chockalingam, and H. Lee, "Control of memory, active perception, and action in minecraft," in *on Machine Learning* (I. Conference, ed.), pp. 2790–2799, PMLR, June 2016.

[52] G. Brunner, O. Richter, Y. Wang, and R. Wattenhofer, "Teaching a machine to read maps with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1)*, April 2018.

[53] S. H. Hsu, S. H. Chan, P. T. Wu, K. Xiao, and L. C. Fu, "Distributed deep reinforcement learning based indoor visual navigation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2532–2537, IEEE, October 2018.

[54] J. J. Kulhánek, E. Derner, T. de Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," in *2019 European Conference on Mobile Robots (ECMR)*, pp. 1–8, IEEE, September 2019.

[55] A. Devo, G. Mezzetti, G. Costante, M. L. Fravolini, and P. Valigi, "Towards generalization in target-driven visual navigation by using deep reinforcement learning," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1546–1561, 2020.

[56] A. Staroverov, D. A. Yudin, I. Belkin, V. Adeshkin, Y. K. Solomentsev, and A. I. Panov, "Real-time object navigation with deep neural networks and hierarchical reinforcement learning," *IEEE Access*, vol. 8, pp. 195608–195621, 2020.

[57] F. Zeng and C. Wang, "Visual navigation with asynchronous proximal policy optimization in artificial agents," *Journal of Robotics*, vol. 2020, 2020.

[58] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1343–1350, IEEE, September 2017.

[59] L. Sun, J. Zhai, and W. Qin, "Crowd navigation in an unknown and dynamic environment based on deep reinforcement learning," *IEEE Access*, vol. 7, pp. 109544–109554, 2019.

[60] A. J. Sathyamoorthy, U. Patel, T. Guan, and D. Manocha, "Frozone: Freezing-free, pedestrian-friendly navigation in human crowds," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4352–4359, 2020.

[61] F. Aznar, M. Pujol, and R. Rizo, "Obtaining fault tolerance avoidance behavior using deep reinforcement learning," *Neurocomputing*, vol. 345, pp. 77–91, 2019.

[62] M. Duguleana and G. Mogan, "Neural networks based reinforcement learning for mobile robots obstacle avoidance," *Expert Systems with Applications*, vol. 62, pp. 104–115, 2016.

[63] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[64] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. De La Puente, and P. Campoy, "Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1024–1031, IEEE, October 2018.

[65] H. T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[66] G. Chen, S. Yao, J. Ma, L. Pan, Y. A. Chen, P. Xu, others, and X. Chen, "Distributed non-communicating multi-robot collision avoidance via map-based deep reinforcement learning," *Sensors*, vol. 20, no. 17, p. 4836, 2020.

[67] W. Chen, S. Zhou, Z. Pan, H. Zheng, and Y. Liu, "Mapless collaborative navigation for a multi-robot system based on the deep reinforcement learning," *Applied Sciences*, vol. 9, no. 20, p. 4198, 2019.

[68] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364, IEEE, May 2017.

[69] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6252–6259, IEEE, May 2018.

[70] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Cambridge, Massachusetts: The MIT Press, 2004.

[71] S. Thrun and J. J. Leonard, "Simultaneous localization and mapping," in *Handbook of Robotics*, pp. 871–889, Springer, 2008.

[72] Q. Zou, Q. Sun, L. Chen, B. Nie, and Q. Li, "A comparative analysis of lidar slam-based indoor navigation for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, pp. 6907–6921, July 2022.

[73] K. P. Murphy, "Bayesian map learning in dynamic environments," *Advances in neural information processing systems*, vol. 12, 1999.

[74] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, pp. 34–46, February 2007.

[75] S. Kohlbrecher, O. V. Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR*, vol. 2011, pp. 155–160, 2011.

[76] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[77] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems*, pp. 22–29, IROS 2010 - Conference Proceedings, 2010.

[78] K. Trejos, L. Rincón, M. Bolaños, J. Fallas, and L. Marín, "2d slam algorithms characterization, calibration, and comparison considering pose error, map accuracy as well as cpu and memory usage," *Sensors*, vol. 22, September 2022.

[79] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robots," *Journal of Intelligent Manufacturing*, vol. 33, no. 2, pp. 387–424, 2022.

[80] C. Warren, "Global path planning using artificial potential fields," in *International Conference on Robotics and Automation*, 1989.

[81] E. Sabudin, R. Omar, and C. C. K. Melor, "Potential field methods and their inherent approaches for path planning," *Journal of Engineering and Applied Sciences*.

[82] J. Savage, M. Morales, and R. Osario, "Learning mobile robot's paths using potential field methods," *IFAC Management and Control of Production and Logistics*, 2000.

[83] L. Veeravagu and L. B. O. line]. Available in: url = http://www.cs.utexas.edu/ EWD/, "Dijkstra's algorithm."

[84] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[85] J. Ferrer, "Implementation and comparison in local planners for ackermann vehicles," *POLITECNICO*, vol. DI MILANO, 2017. Consultado: el 2 de junio de [En línea]. Disponible en, 2023.

[86] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK - 7th German Conference on Robotics*, (Germany), Munich, 2012.

[87] C. Rösmann, "Difference between dwa and teb local planners," *cit. on*, p. 21, 2022.

[88] C. Rosmann, A. Makarow, and T. Bertram, "Online motion planning based on nonlinear model predictive control with non-euclidean rotation groups," *European Control Conference (ECC)*, 2021.

[89] Y.-C. Lin, C.-C. Chou, and F.-L. Lian, "Indoor robot navigation based on dwa*: Velocity space approach with region analysis," *ICROS-SICE International Joint Conference*, 2009.

[90] T. Liu, R. Yan, G. Wei, and L. Sun, "Local path planning algorithm for blind-guiding robot based on improved dwa algorithm," *Chinese Control And Decision Conference (CCDC)*, 2019.

[91] M. Li and C. Yang, "Navigation simulation of autonomous mobile robot based on teb path planner," in *International Conference on Control and Intelligent Robotics*, 2021.

[92] W. Dai and X. Ma, "Improvement of collision detection using time elastic band algorithm," in *International Conference on Information Technology: IoT and Smart City*, 2021.

[93] S. Ishihara, M. Kanai, R. Narikawa, and T. Ohtsuka, "A proposal of path planning for robots in warehouses by model predictive control without using global paths," *International Federation of Automatic Control (IFAC)*, 2022.

[94] J. Li, M. Ran, H. Wang, and L. Xie, "Mpc-based unified trajectory planning and tracking control approach for automated guided vehicles," in *International Conference on Control and Automation (ICCA*, 2019.

[95] R. Sutton and A. Barto, *Reinforcement Learning: An introduction, Second*. Cambridge, Massachusetts: The MIT Press, 2018.

[96] M. Morales, *Grokking Deep Reinforcement Learning*. United States of America: Manning Publications Co, 2020.

[97] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in *IEEE International Conference on Robotic Computing*, 2019.

[98] V. François-Lavet, P. Henderson, M. Bellemare, J. Pineau, and R. Islam, "An introduction to deep reinforcement learning," in *Foundations and Trends® in Machine Learning, vol. 11, -Delft: NOW*, 2018.

[99] G. Chen *et al.*, "Robot navigation with map-based deep reinforcement learning," *ICNSC*, 2020.

[100] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," *IROS*, 2014.

[101] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *International Conference on Intelligent Robots and Systems*, 2020.

[102] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *International Conference on Robotics and Automation (ICRA*, 2019.

[103] R. Min-Fan and H. Sharfiden, "Mobile robot navigation using deep reinforcement learning," *Chinese Control And Decision Conference*, 2019.

[104] Z. Lu and R. Huang, "Autonomous mobile robot navigation in uncertain dynamic environments based on deep reinforcement learning," in *International Conference on Real-time Computing and Robotics*, 2021.

[105] Y. Sasaki, S. Matsuo, A. Kanezaki, and H. Takemura, "A3c based motion learning for an autonomous mobile robot in crowds," in *on Systems, Man and* (I. Conference, ed.), Cybernetics (SMC, 2019.

[106] I. A. Aranda and G. Pérez-Zúñiga, "Highly maneuverable target tracking under glint noise via uniform robust exact filtering differentiator with intrapulse median filter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 58, no. 3, pp. 2541–2559, 2021.

[107] O. Robotics, "Setup and configuration of the navigation stack on a robot," 2023. Consultado: el 23 de junio de [On line]. Available in: http://wiki.ros.org/navigation/Tutorials/RobotSetup.

[108] K. Zheng, "Ros navigation tuning guide," *Robot Operating System (ROS) The Complete Reference (Volume 6)*, pp. 197–226, 2021.

[109] M. Everett, Y. F. Chen, and J. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *International Conference on Intelligent Robots and Systems (IROS*, 2018.

[110] L. Kästner, T. Buiyan, L. Jiao, T. A. Le, X. Zhao, Z. Shen, and J. Lambrecht, "Arena-rosnav: Towards deployment of deep-reinforcement-learning-based obstacle avoidance into conventional autonomous navigation systems," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6456–6463, IEEE, 2021.

[111] L. Kästner, C. Marx, and J. Lambrecht, "Deep-reinforcement-learning-based semantic navigation of mobile robots in dynamic environments," in *International Conference on Automation Science and Engineering (CASE*, 2020.

[112] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," *IROS*, 2017.

[113] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[114] A. Ezquerro, M. Rodriguez, and R. Tellez, "openai ros - ros wiki," *Open Robotics*.

[115] I. Naotunna and T. Wongratanaphisan, "Comparison of ros local planners with differential drive heavy robotic system," in *2020 International Conference on Advanced Mechatronic Systems (ICAMechS)*, pp. 1–6, IEEE, 2020.

[116] B. Cybulski, A. Wegierska, and G. Granosik, "Accuracy comparison of navigation local planners on ros-based mobile robot," in *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, pp. 104–111, IEEE, 2019.

[117] J. Wen *et al.*, "Mrpb 1.0: A unified benchmark for the evaluation of mobile robot local planning approaches," *ICRA*, 2021.

[118] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multi-agent collision avoidance with deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 285–292, IEEE, 2017.

[119] J. Jin, N. M. Nguyen, N. Sakib, D. Graves, H. Yao, and M. Jagersand, "Mapless navigation among dynamics with social-safety-awareness: a reinforcement learning approach from 2d laser scans," 2023. ICRA, 2020, Accessed: Sep. 28 [On line]. Available in: https://sites.google.com/view/ssw-batman.