RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Automatique

End-of-studies project dissertation

for obtaining the State Engineer's degree in Automation and Control

Deep Reinforcement Learning based mapless navigation and control of mobile robots.

RABIA Khalil & KHELFAOUI Abderaouf

Under the direction of:

Dr. ACHOUR Hakim ENP and **Dr. KHELOUAT Samir** ENSTA

Publicly presented and defended on the 21st of June, 2025, in front of the

Jury composed of :

President:	M. STIHI Omar	ENP
Promoter:	Dr. ACHOUR Hakim	ENP
Promoter:	Dr. KHELOUAT Samir	ENSTA
Examiner:	Pr. BERKOUK El Madjid	ENP

ENP 2025

ENP, 10 Rue des Frères OUDEK, El Harrach 16200 Alger Algérie.

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

Département d'Automatique

End-of-studies project dissertation

for obtaining the State Engineer's degree in Automation and Control

Deep Reinforcement Learning based mapless navigation and control of mobile robots.

RABIA Khalil & KHELFAOUI Abderaouf

Under the direction of:

Dr. ACHOUR Hakim ENP and **Dr. KHELOUAT Samir** ENSTA

Publicly presented and defended on the 21st of June, 2025, in front of the

Jury composed of :

President:	M. STIHI Omar	ENP
Promoter:	Dr. ACHOUR Hakim	ENP
Promoter:	Dr. KHELOUAT Samir	ENSTA
Examiner:	Pr. BERKOUK El Madjid	ENP

ENP 2025

ENP, 10 Rue des Frères OUDEK, El Harrach 16200 Alger Algérie.

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Automatique

Mémoire de projet de fin d'études

pour l'obtention du diplôme d'ingénieur d'état en Automatique

Navigation autonome sans carte basée sur l'Apprentissage par Renforcement Profond et commande des robots mobiles.

RABIA Khalil & KHELFAOUI Abderaouf

sous la direction de :

Dr. ACHOUR Hakim ENP and **Dr. KHELOUAT Samir** ENSTA

Présenté et soutenu publiquement le 21/06/2025

Composition du jury :

Président:	M. STIHI Omar	ENP
Promoteur:	Dr. ACHOUR Hakim	ENP
Promoteur:	Dr. KHELOUAT Samir	ENSTA
Examinateur:	Pr. BERKOUK El Madjid	ENP

ENP 2025

ENP, 10 Rue des Frères OUDEK, El Harrach 16200 Alger Algérie.

ملخص

يقدم هذا المشروع سلسلة معالجة لـ **الملاحة بدون خريطة** للروبوتات المتنقلة، حيث يتم الفصل بين اتخاذ القرار والتحكم. يقوم وكيل التعلم المعزز العميق (DRL)، المدرب باستخدام خوارزمية TD3 والشبكات العصبية، بتوليد أوامر السرعة التي تسمح للروبوت بالوصول إلى الهدف وتجنب العقبات، اعتماداً فقط على بيانات المستشعرات المُدمجة. ثم يتم تمرير هذه الأوامر إلى متحكم غامض من نوع تاكاغي-سوجينو (T-S)، والذي يضمن تتبعًا دقيقًا ومسؤولًا للمسار. في حالة الروبوت الواحد، تتم مقارنة الملاحة القائمة على DRL مع نهج ملاحة كلاسيكي. يتم بعد ذلك توسيع هذا الإطار ليشمل سيناريو متعدد الروبوتات، مما يُظهر التنسيق اللامركزي في بيئات مشتركة. تؤكد نتائج المحاكاة فعالية وقابلية تكيّف الحل المقترح.

الكلمات المفتاحية: الملاحة بدون خريطة، التعلم المعزز العميق، الشبكات العصبية، المتحكم الغامض T-S، تتبع المسار,TD,3 الروبوتات المتنقلة، الملاحة الكلاسيكية.

Résumé

Ce mémoire présente une chaîne de traitement pour la **navigation sans carte** de robots mobiles, où la prise de décision et le contrôle sont gérés de manière distincte. Un agent d'apprentissage par renforcement profond (DRL), entraîné avec l'algorithme TD3 et des réseaux de neurones, génère des commandes de vitesse permettant au robot d'atteindre un objectif tout en évitant les obstacles, en se basant uniquement sur les données des capteurs embarqués. Ces commandes sont ensuite transmises à un contrôleur flou de type Takagi-Sugeno (T-S), qui assure un suivi de trajectoire précis et robuste. Dans le cas mono-agent, la navigation basée sur le DRL est comparée à une approche de navigation classique. Le cadre proposé est ensuite étendu à un scénario multi-robots, démontrant une coordination décentralisée dans des environnements partagés. Les résultats de simulation valident l'efficacité et l'adaptabilité de la solution proposée.

Mots-clés : navigation sans carte, DRL, réseaux de neurones, contrôleur flou T-S, suivi de trajectoire, robots mobiles, navigation classique, TD3.

Abstract

This thesis presents a pipeline for **mapless navigation** of mobile robots, where decision-making and control are handled in separate stages. A Deep Reinforcement Learning (DRL) agent, trained with artificial neural networks, generates velocity commands that allow the robot to reach a goal while avoiding obstacles, using only onboard sensor data. These commands are then passed to a fuzzy Takagi-Sugeno (T-S) controller, which ensures accurate and robust trajectory tracking. In the single-agent case, the DRL-based navigation is compared with a classical navigation approach. The framework is further extended to a multi-robot setup, demonstrating decentralized coordination in shared environments. Simulation results validate the effectiveness and adaptability of the proposed pipeline.

Keywords: mapless navigation, Deep Reinforcement Learning, artificial neural networks, fuzzy T-S controller, trajectory tracking, mobile robots, classical navigation, TD3.

Dedication

To my dear parents,
To my dear sister,
To my dear brother and sister in law,
To my friends,

- Khalil

Dedication

To my beloved parents, who got me to where I am now,
To my dear sisters,
To my brother, the shoulder that never failed me,
To my dude Abderrahman, the man of impossible missions,
To all my family and friends,
To our brothers and sister in Palestine, those who defended and still defending the muslims indignity, may Allah be with you,

- Abderaouf

Acknowledgment

We would like to express our sincere gratitude to our supervisors, Dr. A. Hakim and Dr. K. Samir, for their valuable guidance, support, and continuous encouragement throughout the development of this work. We are also thankful to Mr. S.Omar for accepting to serve as the president of the jury, and to Professor B. El Madjid for kindly agreeing to examine our thesis. Our special thanks go to Mr. B. Mohamed and Mr. L. Karim for granting us access to the simulation station, which was essential to our experiments. We are also deeply grateful to our friends from the Control Engineering Department for their help and support, and to everyone who contributed to this work in any form, even through words of encouragement.

Contents

List of Tables

List of Figures

General Introduction

Mobile robot navigation is a crucial capability for a wide range of applications, including autonomous delivery, surveillance, rescue, and service robotics. It consists of guiding a robot from an initial position to a target location while avoiding static and dynamic obstacles. As robotic systems are increasingly deployed in real world settings, the demand for reliable, safe, and intelligent navigation strategies becomes more critical than ever.

Traditionally, robot navigation has been tackled using a modular approach that includes mapping, localization, path planning. and control. Classical method such as A*, Dijkstra and Rapidly-exploring Random Trees (RRT) have been widely used for global path planing [2]. For local path execution and obstacle avoidance, algorithms like the Dynamic Window Approach (DWA) [3] or vector field histograms have proven effective in structured environments. These pipelines are often complemented by low-level control laws, including PID or MPC, to ensure accurate trajectory following.

While classical navigation methods offer interpretability and robustness in known environments, they rely heavily on accurate maps. Their performance typically degrades in the presence of dynamic scenarios where full knowledge is unavailable or where uncertainties and sensor noise are present. Moreover, the sequential computation of these pipelines can create bottlenecks that make real-time decision-making and adaptability difficult.

Recent advances in machine learning, particularly in reinforcement learning (RL), have opened new possibilities for learning end-to-end navigation policies directly from interaction with the environment. RL allows an agent to learn optimal actions by maximizing cumulative reward signal over time. In the context of robot navigation, RL enables policies that enables the robot to goal-seeking behavior while learning to avoid collisions. Algorithms such as Deep Q-Networks (DQN) [4] and actor critic methods like DDPG [5], TD3 [6], and SAC [7] have shown promising results in various problems other than robot navigation.

In this project, we focus on applying deep reinforcement learning techniques—specifically the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm—to enable autonomous navigation of a mobile robot in indoor environments. The agent will be trained using a curriculum learning approach, gradually progressing from simple tasks to more complex ones. To better evaluate generalization capabilities at a certain stage, an agent trained in environment A will be tested in environment B. Its performance in unseen scenarios will then be compared to that of a classical navigation pipeline using the Nav2 stack from the ROS2 framework.

Beyond single-robot navigation, we extend our approach to a collaborative setup involving two robots navigating simultaneously toward individual or shared goals. This two-robot system introduces additional challenges, such as coordination and collision avoidance between the robots. To address this, each robot will communicate its pose to the other, laying the groundwork for potential generalization to multi-robot systems.

Moreover, to enhance control robustness and ensure better interpretability, we incorporate a fuzzy logic controller, which effectively handles imprecise inputs and system nonlinearities. In our setup, the velocity commands generated by the deep reinforcement learning agent serve as reference inputs to a trajectory tracking system based on fuzzy control logic. To further mitigate the effects of sensor noise, we design a fuzzy observer-based control system, which functions as a filter and contributes to robust and reliable decision-making.

This work aims to contribute a scalable, flexible, and intelligent navigation framework that leverages the learning capabilities of deep reinforcement learning while enhancing robustness and interoperability through fuzzy logic and observer-based control strategies. In this architecture, the velocity commands generated by the DRL agent are used as reference inputs for a fuzzy logic-based trajectory tracking system. The inclusion of a fuzzy observer further improves noise resilience and decision stability. The proposed framework is validated through extensive simulations, with performance assessed using metrics such as success rate, distance traveled, time to goal, and safety.

The First chapter explores how a fuzzy controller, built using a Takagi-Sugeno modeling approach, can serve as a trajectory tracking controller, providing stability and noise rejection to complement the DRL-based decision-making framework.

Building upon these considerations, Chapter 2 provides a comprehensive overview of deep reinforcement learning (DRL), from its foundations, classification, and the most widely used agents in DRL research.

Chapter 3 focuses on the application of DRL in the context of mapless navigation for mobile robots. It explores the design of training environments, the development of custom reward functions, and the progression through various experiments aimed at building a robust single agent navigation policy capable of obstacle avoidance and goal-reaching behaviors.

Based on the approach taken in chapter 3, chapter 4 extends the single agent setup to a double agent scenario, where a double agent architecture is introduced. The logic and insights gained from the initial experiments are reused and adapted to the double agent setup.

Finally, the conclusion offers an overview of the key results obtained throughout the project. It highlights the advantages and limitations of using deep reinforcement learning for mapless navigation, including its adaptability and learning capabilities versus challenges like training time and generalization. The chapter also outlines potential future directions to improve the current framework.

Chapter 1

Fuzzy modeling and control

A fuzzy controller or model employs fuzzy rules, which consist of linguistic if-then statements involving fuzzy sets, fuzzy logic, and fuzzy inference. These rules serve as a crucial representation of expert control and modeling knowledge, connecting input variables of fuzzy controllers or models to output variables. The two primary types of fuzzy rules are Mamdani fuzzy rules and Takagi-Sugeno (TS) fuzzy rules.

In this chapter, we are mainly interested in the TS fuzzy modeling and will explore the construction of a T-S fuzzy model via sector non-linearity method and its stabilization using a fuzzy controller as well.

In other hand, this approach consists, more precisely, in reducing the complexity of the system by decomposing its operating space into a finite number of operating regions. Since the system behavior is less complex in each region, a simple-structure sub-model can be used. Thus, depending on the region in which the system operates, the output of each sub-model is more or less utilized to approximate the overall behavior of the system. The contribution of each sub-model is quantified by a weighting function associated with each operating region.

1.1 State representation of a nonlinear model

Any physical system with continuous evolution can be written in the form of a state representation. This allows describing the input-output relationships of a system through modeling in the form of ordinary differential equations. The general form of a representation is given by [8]:

$$\begin{cases} g(\dot{x}(t), x(t), u(t)) = 0 \\ y(t) = k(x(t), u(t)) \end{cases} \tag{1.1}$$

where $x(t)$ is the state vector of the system, $u(t)$ is the input vector, and $y(t)$ is the output vector. The first equation is called the state equation, and the second one is the output equation. Note that the system 1.1 is given in a general form and includes the class of models written in the form of a state-space representation, known as affine in control, given by [8]:

$$\begin{cases} \dot{x}(t) = f(x(t)) + g(x(t))u(t) \\ y(t) = h(x(t)) + m(x(t))u(t) \end{cases} \tag{1.2}$$

where $f(x(t))$ is the state function, $g(x(t))$ is the input function, $h(x(t))$ is the output function, and $m(x(t))$ is the input-output coupling matrix. This type of system, commonly encountered in control problems, will constitute the main subject of this study.

1.2 T-S Fuzzy modeling

A TS-type fuzzy model consists of a set of linear models (sub-models) linked by an interpolation structure represented by nonlinear membership functions. Indeed, in 1985, based on the fuzzy logic formalism, Takagi and Sugeno proposed an approach to modeling nonlinear systems based on a set of fuzzy rules of the type "If...Then" whose conclusions represent a set of linear dynamics. Thus, if r is the number of rules describing a TS model, the i-th rule R^i is given by [8]:

$$\text{If } z_1(t) \text{ is } K_1^i(z_1(t)),\ z_2(t) \text{ is } K_2^i(z_2(t)),\ \ldots,\ z_p(t) \text{ is } K_p^i(z_p(t)) \text{ then } \begin{cases} \dot{x}(t) = A_i x(t) + B_i u(t) \\ y(t) = C_i x(t) \end{cases} \tag{1.3}$$

Where, for $j = 1, \ldots, p$, $K_j^i(z_j(t))$ are fuzzy subsets that partition the discourse universe, $z_j(t)$ are the premise variables dependent on the inputs, outputs and/or state of the system, $x(t) \in \mathbb{R}^n$ is the state vector of the system, $u \in \mathbb{R}^m$ is the input vector and $y(t) \in \mathbb{R}^s$ is the output vector. A_i, B_i, and C_i are matrices describing the dynamics of the system .

For each fuzzy rule R^i, a weight function $w_i(z(t))$ can be attributed, determining the contribution of each linear dynamic composing the multi-model as a whole. This weight function depends on the degree of membership of the premise variables $z_j(t)$ to the fuzzy subsets $K_j^i(z_j(t))$ and the choice of the AND operator. Such that:

$$w_i(z(t)) = \prod_{j=1}^{p} K_j^i(z_j(t)) \quad \text{for } i = 1, \ldots, r \tag{1.4}$$

With: $\forall t, w_i(z(t)) \geq 0$. We define:

$$\mu_i(z(t)) = \frac{w_i(z(t))}{\sum_{i=1}^{r} w_i(z(t))} \tag{1.5}$$

The activation function $\mu_i(z(t))$ of the i-th rule of the fuzzy model verifies the convex sum properties:

$$0 < \mu_i(z(t)) < 1 \tag{1.6}$$

$$\sum_{i=1}^{r} \mu_i(z(t)) = 1 \tag{1.7}$$

Thus, after defuzzification, the state representation of a TS multi-model, in its entirety, can be written as follows:

$$\dot{x}(t) = \sum_{i=1}^{r} \mu_i(z(t))(A_i x(t) + B_i u(t)) \tag{1.8}$$

$$y(t) = \sum_{i=1}^{r} \mu_i(z(t))(C_i x(t) + D_i u(t)) \tag{1.9}$$

1.2.1 Construction of T-S Fuzzy Models

We need a T-S fuzzy model for a nonlinear system to design a fuzzy T-S controller. Therefore, constructing a fuzzy model represents a crucial and basic procedure in this method. In general, there are three approaches to constructing fuzzy models [9]:

1. Identification (Fuzzy modeling) using input-output data.
2. Linearization around multiple operating points.
3. The third approach is based on the formalism of nonlinear sectors. This technique directly relies on the analytical knowledge of the nonlinear model. Unlike the two previous approaches, which provide an approximation of the nonlinear model, this third method offers a TS model that exactly represents the initial nonlinear model.

Fuzzy modeling based on input-output data has been widely explored since the works of Takagi, Sugeno and Kange [7]. This approach involves two steps, identifying the structure and identifiying the parameters. It is useful for systems that are hard to model using analytical or pysical models[8].

On the other hand, nonlinear models for mechanical systems can be readily obtained by, for example, the Lagrange method and the Newton-Euler method. In such cases, the third approach, which derives a fuzzy model from given nonlinear dynamical models, is more appropriate.

This chapter focuses on the third approach that utilizes the idea of ”sector nonlinearity”.

1.2.2 Sector non-linearity

Sector nonlinearity is based on the idea that a simple nonlinear system $\dot{x} = f(x(t))$, where $f(0) = 0$ aiming to find the global sector such that $\dot{x} = f(x) \in [a_1 a_2]x(t)$, where a_1 and a_2 are real constants.

Figure 1.1a [8] illustrates the global sector nonlinearity idea.

This approach guarantees an exact fuzzy model construction. However, finding a global sector for general nonlinear systems is sometimes difficult. In such cases, we consider local sector nonlinearity, which is reasonable since the variables of physical systems are always bounded.

Figure 1.1b [8] illustrates the local sector nonlinearity, where two lines define the local sectors under the condition $-d < x(t) < d$.
The fuzzy model exactly represents the behavior within this local region, i.e., $-d < x(t) < d$.

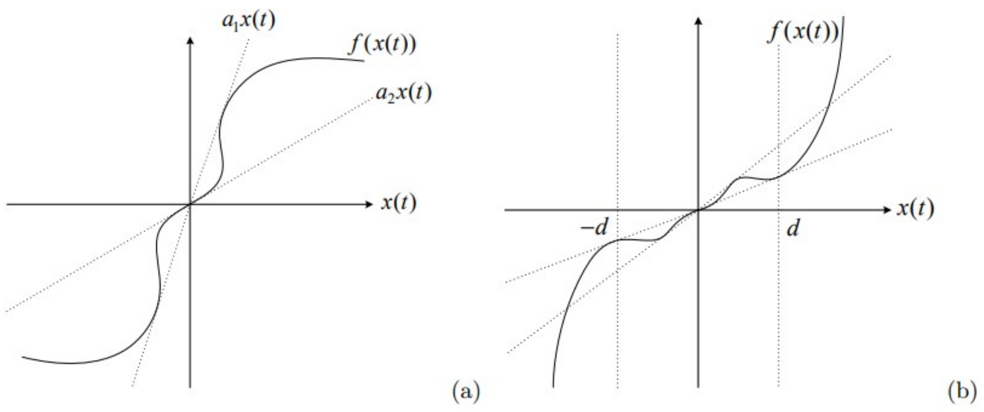


Figure 1.1: (a) Global sector nonlinearity. (b) Local sector nonlinearity.

The advantage of such a method is that it does not generate approximation errors and reduces the number of models compared to the linearization method.

Let's consider the continuous nonlinear system :

$$\dot{x} = f(x(t)) + g(x(t)) \cdot u(t) \tag{1.10}$$

Where: $x(.) \in \mathbb{R}^n$, $u(.) \in \mathbb{R}^m$, $f((x).) \in \mathbb{R}^p$, $g \in \mathbb{R}^{p\times m}$
Based on the following lemma, we can get the T-S model starting from a non-linear model.

Lemma 1.1[9]. Let $z(x(t))$ be a bounded function from $[a,b] \to \mathbb{R}$ for all $x \in [a,b]$ with $[a,b] \in \mathbb{R}^2_+$. Then there exist two functions $F_1(x(t))$ and $F_2(x(t))$ as well as two scalars α and β such that:

$$z(x(t)) = \alpha \cdot F^1(x(t)) + \beta \cdot F^2(x(t)) \tag{1.11}$$

With: $F_1(x(t)) + F_2(x(t)) = 1$, $F_1(x(t)) \geq 0$ and $F_2(x(t)) \geq 0$. A decomposition of $z(x(t))$ is considered on $[a,b]$ as follows:

$$\begin{cases} \beta = \min_{x\in[a,b]} z(t) \\ \alpha = \max_{x\in[a,b]} z(t) \\ F_1(x(t)) = \frac{z(t)-\beta}{\alpha-\beta} \\ F_2(x(t)) = \frac{\alpha-z(t)}{\alpha-\beta} \end{cases} \tag{1.12}$$

Under the assumptions of continuity and boundedness of the functions $f(x(t))$ and $g(x(t))$ in the model 1.8 and the output 1.9 with $f(0) = 0$ and $g(0) = 0$, they can be rewritten as follows:

$$\begin{cases} f(x(t)) = \sum_{i=1}^{r} \mu_i(z(t)) A_i x(t) \\ g(x(t)) = \sum_{i=1}^{r} \mu_i(z(t)) B_i x(t) \\ h(x(t)) = \sum_{i=1}^{r} \mu_i(z(t)) C_i x(t) \\ m(x(t)) = \sum_{i=1}^{r} \mu_i(z(t)) D_i x(t) \end{cases} \tag{1.13}$$

The model 1.8 and the output 1.9 become:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{r} \mu_i(z(t))(A_i x(t) + B_i u(t)) \\ y(t) = \sum_{i=1}^{r} \mu_i(z(t))(C_i x(t) + D_i u(t)) \end{cases} \tag{1.14}$$

In this case, the obtained multi-model representation corresponds exactly to the nonlinear model over the considered compact interval.

Figure 1.2 [8] shows the detailed diagram of a standard T-S model, also designed as coupled T-S structure in the literature. Indeed, they allow reducing the complexity of a nonlinear problem to be addressed (stability, stabilization, observation, diagnosis, etc.) by decomposing it into a set of local linear issues. The set of local solutions corresponding to these latter then constitutes the global solution of the initial nonlinear problem.

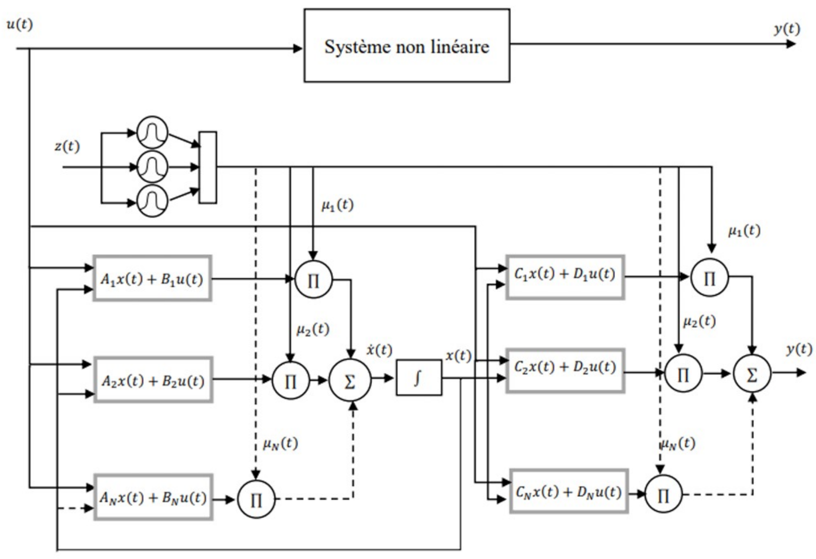


Figure 1.2: Coupled T-S structure

Example:

We consider a nonlinear system given by the following equation:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix}=\begin{bmatrix}1 & \sin(x_1)\\ f(x_1,x_2) & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}0\\ -\sin(x_1)\end{bmatrix}u \tag{1.15}$$

This system includes two nonlinearities: $f(x_1, x_2)$ and $\sin(x_1)$. By applying the previously defined lemma to each of these two nonlinear functions, a TS fuzzy model with four rules is obtained ($2^{nl} = 4$, where nl is the number of nonlinearities).

Suppose the function $f(x_1, x_2)$ takes values in the interval $[\underline{f}, \overline{f}]$ when $x = (x_1, x_2)$ belongs to a subset Ω of $\mathbb{R}^2$. By applying Lemma 1.1, the membership functions of the two nonlinearities are given by the following equations:

For $f(x_1, x_2)$:

$$\omega_0^1=\frac{\overline{f}-f(x_1,x_2)}{\overline{f}-\underline{f}},\quad \omega_1^1=\frac{f(x_1,x_2)-\underline{f}}{\overline{f}-\underline{f}} \tag{1.16}$$

For $\sin(x_1)$:

$$\omega_0^2=\frac{1-\sin(x_1)}{2},\quad \omega_1^2=\frac{\sin(x_1)+1}{2} \tag{1.17}$$

The four rules (or sub-models) that represent the TS fuzzy model of the nonlinear model (1.15) are given as follows:

- R^1: If $x_1(t)$ is ω_0^2 and $f(x_1(t), x_2(t))$ is ω_0^1, then

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \underline{f} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{1.18}$$

- R^2: If $x_1(t)$ is ω_0^2 and $f(x_1(t), x_2(t))$ is ω_1^1, then

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \underline{f} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} u \tag{1.19}$$

- R^3: If $x_1(t)$ is ω_1^2 and $f(x_1(t), x_2(t))$ is ω_0^1, then

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \overline{f} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{1.20}$$

- R^4: If $x_1(t)$ is ω_1^2 and $f(x_1(t), x_2(t))$ is ω_1^1, then

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \overline{f} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} u \tag{1.21}$$

The main interest of this method is that the obtained TS fuzzy model exactly represents the nonlinear model in the subset Ω of the state space.

1.3 Control and observation of TS models

In order to stabilize the T-S fuzzy model in 1.14, the PDC (Parallel Distributed Compensation) control is utilized and given by the following formula :

$$u(t) = -\sum_{i=1}^{r} \mu_i(z(t)K_i x(t) \tag{1.22}$$

Mainly, when the complete state of the model is not measurable, or for filtering conditions, a state observer is added to the control scheme. In this case, the state observer has the following form :

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{r} \mu_i(\hat{z}(t))(A_i \hat{x}(t) + B_i u(t) + L_i(y(t) - \hat{y}(t))) \\ y(t) = \sum_{i=1}^{r} \mu_i(\hat{z}(t))(C_i \hat{x}(t)) \end{cases} \tag{1.23}$$

With $\hat{x}(t)$ the state estimate, $\hat{z}(t)$ the premise estimate and L_i, $i \in \{1, .., r\}$, the observation gains. In this work we consider the premise variables are measurable, $\hat{z}(t) = z(t)$. In this case to compute the control law 1.22 and the observer 1.23, the same $\mu_i()$ as the model's are used. This approach allows for the development of various relaxation methods to determine the stabilization conditions of the state of the model and the observation error, which will be addressed in the following paragraph.

1.3.1 Relaxation to resolve LMI Constraints

The stabilization of the T-S model and the observation error is based on Lyapunov theory. Suitable quadratic Lyapunov functions are used and conduct to LMI (Linear Matrix Inequalities) that are resolved using LMI TOOLBOX on Matlab.

The stabilization problem conducts to double sum μ_i as follows :

$$\sum_{i=1}^{r}\sum_{j=1}^{r}\mu_i(z(t))\mu_j(z(t))\Upsilon_{ij} \tag{1.24}$$

Where Υ_{ij} are constant matrices to find.

The goal is to find an LMI problem that verifies the conditions in 1.24. Many relaxations are proposed in literature, only Tanaka's and Tuan's are considered in this chapter. Relaxations are based on the fact that the functions $\mu_i(z(t))$, $i \in \{1, ..., r\}$ and the products $\mu_i(z(t))\mu_j(z(t))$ and $\mu_j(z(t))\mu_i(z(t))$ are equal, so we impose that all matrices Υ_{ij} are negative defined.

Lemma 1.2 *(Tanaka, 1994)*:
Let Υ_{ij} be matrices of appropriate size. Inequality 1.24 is satisfied if:

$$\begin{cases} \Upsilon_{ii} < 0 & \forall i \in \{1, \dots, r\} \\ \Upsilon_{ij} + \Upsilon_{ji} < 0 & \forall (i,j) \in \{1, \dots, r\}^2, \quad i < j \end{cases}$$

Lemma 1.3 *(Tuan, 2001)*:
Let Υ_{ij} be matrices of appropriate size. Inequality (2.11) is satisfied if:

$$\begin{cases} \Upsilon_{ii} < 0 & \forall i \in \{1, \dots, r\} \\ \frac{2}{r-1}\Upsilon_{ii} + \Upsilon_{ij} + \Upsilon_{ji} < 0 & \forall (i,j) \in \{1, \dots, r\}^2, \quad i \neq j \end{cases}$$

In this work, we are only interested in Tuan's relaxation, that ensures the same number of unkown variables as Tanaka's with more relaxed LMI conditions.

1.4 Fuzzy T-S model of the mobile robot and trajectory tracking

1.4.1 Kenimatic model of the robot

The kinematic model of a non-holonomic unicycle mobile robot in the X-Y plane (Figure 1.3) can be represented by formula 1.25 in the case of rolling without slipping [10]:

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \omega \end{cases} \tag{1.25}$$

where v and ω are the control inputs of the mobile robot, representing the linear and angular velocities, respectively. The output variables are x and y (the coordinates of the robot's center of gravity) and θ which represent the orientation of the robot (angle between the X-axis and the velocity vector of the robot). While r is the radius of the wheel and l is the distance between the wheel and the center of the robot.

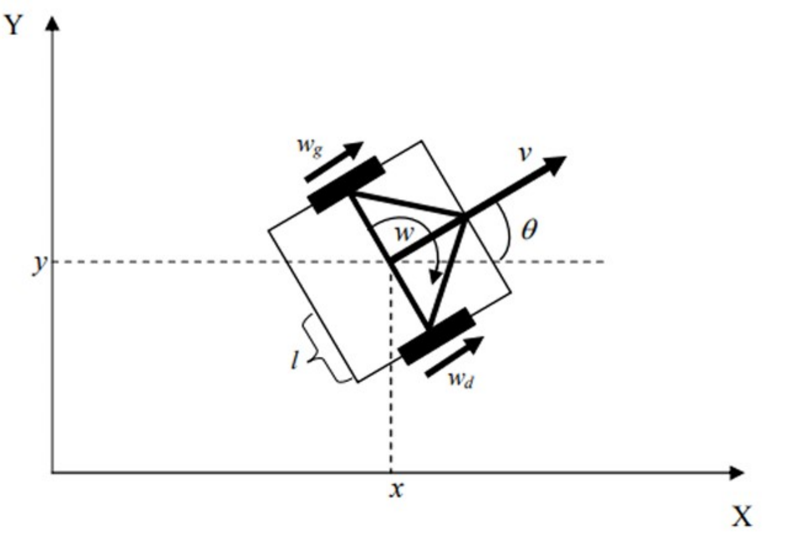


Figure 1.3: Uni-cycle mobile robot

The angular velocities of left wheel ω_g and right wheel ω_d are given by :

$$\begin{bmatrix}\omega_d\\ \omega_g\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & l\\ l & -1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

1.4.2 Trajectory tracking error model

Tracking error $e = [e_x, e_y, e_\theta]$ is defined as the difference between the pose of the real robot R and the reference robot model R$_r$. The tracking error is generally expressed in the robot frame as follows :

$$\begin{bmatrix}e_x\\ e_y\\ e_\theta\end{bmatrix} = \begin{bmatrix}cos(\theta) & sin(\theta) & 0\\ -sin(\theta) & cos(\theta) & 0\\ 0 & 0 & 1\end{bmatrix}(q_r - q) \tag{1.26}$$

While $q = \begin{bmatrix}x & y & \theta\end{bmatrix}^T$ is the actual pose of the robot, and $q_R = \begin{bmatrix}x_r & y_r & \theta_r\end{bmatrix}^T$ is the pose of reference.

Using both 1.25 and 1.26 equations, supposing that the kinematic model of the real robot and the virtual one are identical give the follow tracking error dynamics :

$$\begin{bmatrix}\dot{e}_x\\ \dot{e}_y\\ \dot{e}_\theta\end{bmatrix} = \begin{bmatrix}cos(e_\theta) & 0\\ sin(e_\theta) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v_r\\ \omega_r\end{bmatrix} + \begin{bmatrix}-1 & e_y\\ 0 & -e_x\\ 0 & -1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{1.27}$$

Where v_r and ω_r are the reference velocities. The control law of the mobile robot is given by $u = \begin{bmatrix}u\\ \omega\end{bmatrix}^T$. In order to be able to obtain a T-S model for the error dynamics, Kanayama in 1990 first proposed an anticipatory action u_f that would be applied as $u_f = \begin{bmatrix}v_r cos(e_\theta) & \omega_r\end{bmatrix}^T$.

Let $u = u_f + u_B$, we replace u by its value and simplify so the model in 1.27 will be rewritten as follows :

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} 0 & \omega_r & 0 \\ -\omega_r & 0 & v_r sinc(e_\theta) \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_r \\ \omega_r \end{bmatrix} + \begin{bmatrix} -1 & e_y \\ 0 & -e_x \\ 0 & -1 \end{bmatrix} u_B \tag{1.28}$$

Where u_B is of dimension 2 as same as u and u_f.

This non-linear model is the start point to compute the fuzzy control law without and with observer. The main interest of this method is to prove the stability of the global system with or without observer in a predefined state space [10].

Figure 1.4 illustrate the error on pose.

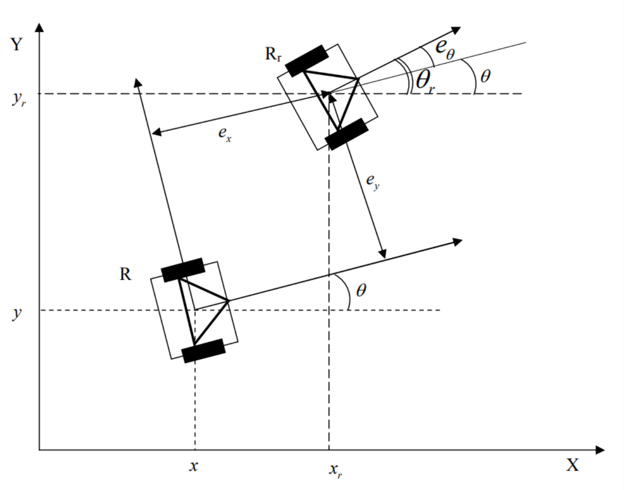


Figure 1.4: Pose error.

1.4.3 T-S fuzzy model of pose error

Considering the model in 1.27, four borned nonlinearities are $\eta_1 = \omega_r$,$\eta_2 = v_r sinc(e_\theta)$,$\eta_3 = e_y$ and $\eta_4 = e_x$ if we consider the following inequalities :

$$|e_x| \leq e_{\max}, \quad |e_y| \leq e_{\max}, \quad |e_\theta| \leq \frac{\pi}{2} \text{ rad.}$$

Based on the nonlinear sector method that contains 2^4 rules (i.e 2^4 fuzzy sub-models)

Applying the lemma 1.1 we obtain the following global model, the sixteen sub-models and the membership functions :

$$\dot{e}(t) = \sum_{i=1}^{16} \mu_i(z(t)) \left(A_i e(t) + B_i u_B(t) \right) \tag{1.29}$$

with the state and control matrices for $i \in \{1, \ldots, 16\}$:

$$A_i = \begin{bmatrix} 0 & -\varepsilon_i^1 w_{r,\max} & 0 \\ \varepsilon_i^1 v_{r,\max} & 0 & \kappa_i \\ 0 & 0 & 0 \end{bmatrix}, \quad B_i = \begin{bmatrix} -1 & \varepsilon_i^2 e_{\max} \\ 0 & \varepsilon_i^3 e_{\max} \\ 0 & -1 \end{bmatrix}$$

such that:

$$e_{\max} = 0.1\,(\mathrm{m}), \quad \varepsilon_i^1 = \begin{cases} +1 & \text{for } 1 \leq i \leq 8 \\ -1 & \text{otherwise} \end{cases}, \quad \varepsilon_i^2 = \begin{cases} +1 & \text{for } i \in \{3,4,7,8,11,12,15,16\} \\ -1 & \text{otherwise} \end{cases}$$

$$\varepsilon_i^3 = (-1)^{i+1}, \quad \kappa_i = \begin{cases} \frac{2}{\pi} v_{r,\min} & \text{for } 1 \leq i \leq 4 \text{ and } 9 \leq i \leq 12 \\ v_{r,\max} & \text{otherwise} \end{cases}$$

$$\begin{cases} \mu_1 = \omega_{01}\omega_{02}\omega_{03}\omega_{04}, & \mu_2 = \omega_{01}\omega_{02}\omega_{03}\omega_{14}, & \mu_3 = \omega_{01}\omega_{02}\omega_{13}\omega_{04}, & fra\mu_4 = \omega_{01}\omega_{02}\omega_{13}\omega_{14}, \\ \mu_5 = \omega_{01}\omega_{12}\omega_{03}\omega_{04}, & \mu_6 = \omega_{01}\omega_{12}\omega_{03}\omega_{14}, & \mu_7 = \omega_{01}\omega_{12}\omega_{13}\omega_{04}, & \mu_8 = \omega_{01}\omega_{12}\omega_{13}\omega_{14}, \\ \mu_9 = \omega_{11}\omega_{02}\omega_{03}\omega_{04}, & \mu_{10} = \omega_{11}\omega_{02}\omega_{03}\omega_{14}, & \mu_{11} = \omega_{11}\omega_{02}\omega_{13}\omega_{04}, & \mu_{12} = \omega_{11}\omega_{02}\omega_{13}\omega_{14}, \\ \mu_{13} = \omega_{11}\omega_{12}\omega_{03}\omega_{04}, & \mu_{14} = \omega_{11}\omega_{12}\omega_{03}\omega_{14}, & \mu_{15} = \omega_{11}\omega_{12}\omega_{13}\omega_{04}, & \mu_{16} = \omega_{11}\omega_{12}\omega_{13}\omega_{14} \end{cases}$$

with:

$$\begin{cases} \omega_{01} = \dfrac{w_{r,\max} - w_r}{w_{r,\max} - w_{r,\min}}, \quad \omega_{11} = 1 - \omega_{01} \\ \omega_{02} = \dfrac{v_{r,\max} - v_r \operatorname{sinc}(e_\theta)}{v_{r,\max} - v_{r,\min} \operatorname{sinc}\left(\frac{\pi}{2}\right)}, \quad \omega_{12} = 1 - \omega_{02} \\ \omega_{03} = \dfrac{e_{\max} - e_x}{2e_{\max}}, \quad \omega_{13} = 1 - \omega_{03} \\ \omega_{04} = \dfrac{e_{\max} - e_y}{2e_{\max}}, \quad \omega_{14} = 1 - \omega_{04} \end{cases}$$

1.4.4 PDC Control law

To stabilize our error dynamics in 1.29, the following PDC control law is synthesized :

$$u_B(t) = -\sum_{i=1}^{16} \mu_i(z(t)) K_i e(t) \tag{1.30}$$

The stability of the closed loop system is proven choosing a quadratic Lyapunov function $V(e(t)) = e(t)^T P e(t)$, where P is a $(n \times n)$ symmetrical and positive defined matrix.

The solution $e = 0$ of the system 1.29 is asymptotically stable if the derivative of the chosen Lyapunov function is negative defined all along the trajectory of the model :

$$\dot{V}(e(t)) = \dot{e}(t)^T P e(t) + e(t)^T P \dot{e}(t) < 0 \tag{1.31}$$

i.e

$$\sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(z(t))\mu_j(z(t)) \left[(A_i - B_i K_j)^T P + P(A_i - B_i K_j)\right] < 0 \tag{1.32}$$

Multiplying 1.32 on the right and on the left by X, we obtain the equivalent condition:

$$\sum_{i=1}^{r}\sum_{j=1}^{r}\mu_i(z(t))\mu_j(z(t))\left[XA_i^T - M_j^T B_i^T + A_i X - B_i M_j\right] < 0 \tag{1.33}$$

where $M_j = K_j X$.

Let us define:

$$\Upsilon_{ij} = XA_i^T - M_j^T B_i^T + A_i X - B_i M_j \tag{1.34}$$

According to Lemma 1.3, if there exist matrices $X > 0$, M_j (for $j = 1, \ldots, r$) satisfying the following linear matrix inequalities:

$$\begin{cases} \Upsilon_{ii} < 0 & \forall i \in \{1, \ldots, r\} \\ \frac{2}{r-1}\Upsilon_{ii} + \Upsilon_{ij} + \Upsilon_{ji} < 0 & \forall (i,j) \in \{1, \ldots, r\}^2,\ i \neq j \end{cases} \tag{1.35}$$

then, for the gain values

$$K_j = X^{-1} M_j, \quad j = 1, \ldots, r \tag{1.36}$$

the solution $e = 0$ of system 1.29 is asymptotically stable.

To ensure more rapid convergence, the inequality in 1.31 is modified by adding an exponential term as follows :

$$\dot{V}(e(t)) \leq -\gamma V(e(t)) \tag{1.37}$$

so that now the solution $e = 0$ of the system 1.29 converges and it is asymptotically stable with a convergence rate of γ if :

$$\Upsilon_{ij} = XA_i^T - M_j^T B_i^T + A_i X - B_i M_j + \gamma X \tag{1.38}$$

1.4.5 Synthesis of a T-S observer

In practice, most of the time, the states are not fully measurable or are affected by environmental noise. Therefore, a state observer is typically added to the control scheme.

In our case, all the states are measurable, however, they are influenced by noise. To address this, a T–S state observer is introduced.

The estimated error dynamics is given by:

$$\dot{\hat{e}}(t) = \sum_{i=1}^{16} h_i(z(t))\left(A_i \hat{e}(t) + B_i u_B(t) + L_i\left(e(t) - \hat{e}(t)\right)\right) \tag{1.39}$$

with $\hat{e} = \left[\hat{e}_x, \hat{e}_y, \hat{e}_\theta\right]^T$, where L_i (for $i \in \{1, \ldots, 16\}$) are the observer gain matrices to be determined.

The PDC (Parallel Distributed Compensation) control law is thus modified as:

$$u_B(t) = -\sum_{i=1}^{16} h_i(z(t)) K_i \hat{e}(t) \tag{1.40}$$

The estimation error dynamics $\tilde{e}(t) = e(t) - \hat{e}(t)$ becomes:

$$\dot{\tilde{e}}(t) = \sum_{i=1}^{16} h_i(z(t))(A_i - L_i)\tilde{e}(t) \tag{1.41}$$

The global closed-loop system dynamics with the observer is given by:

$$\begin{bmatrix} \dot{e} \\ \dot{\tilde{e}} \end{bmatrix} = \sum_{i=1}^{16}\sum_{j=1}^{16} h_i(z(t))h_j(z(t)) \begin{bmatrix} A_i - B_iK_j & B_iK_j \\ 0 & A_i - L_j \end{bmatrix} \begin{bmatrix} e \\ \tilde{e} \end{bmatrix} \tag{1.42}$$

The observer gain computation is performed using a procedure similar to that used for computing the control gains. The Lyapunov function chosen in this case is

$$V(\tilde{e}(t)) = \tilde{e}(t)^T P_{\text{obs}} \tilde{e}(t) \tag{1.43}$$

where P_{obs} is a symmetric and positive definite matrix of dimension $n \times n$.

Using Lemma 1.3, the estimation error dynamics 1.41 is stable if there exist matrices N_i, $i \in \{1, \ldots, 16\}$, and $P_{\text{obs}} > 0$, such that the following LMI conditions are satisfied:

$$\Upsilon_i < 0 \tag{1.44}$$

with

$$\Upsilon_i = A_i^T P_{\text{obs}} + P_{\text{obs}} A_i - N_i - N_i^T \tag{1.45}$$

The gains of the fuzzy TS observer in equation 1.41 are then given by:

$$L_i = P_{\text{obs}}^{-1} N_i \tag{1.46}$$

1.4.6 Simulation result for trajectory tracking

In order to validate the PDC control law, a circular reference trajectory is generated with a velocity reference of $v_r = 0.2m/s$ and $\omega_r = 1rad/s$. The goal is to join the circular trajectory and maintain it with and without noises on measurement.

Based on LMI 1.37, the stabilization gains K_i, $i \in \{1, ..., 16\}$ with $\gamma = 0.1$.

$$K_1 = \begin{bmatrix} -2.3664 & 0.3278 & -0.3953 \\ 0.2172 & -2.0620 & -1.1002 \end{bmatrix} \quad K_2 = \begin{bmatrix} -2.3327 & 0.4102 & -0.3825 \\ 0.1517 & -1.9008 & -1.0098 \end{bmatrix}$$

$$K_3 = \begin{bmatrix} -2.2669 & 0.1142 & -0.5047 \\ 0.2579 & -2.0330 & -1.0839 \end{bmatrix} \quad K_4 = \begin{bmatrix} -2.2383 & 0.2390 & -0.4783 \\ 0.1696 & -1.8729 & -0.9956 \end{bmatrix}$$

$$K_5 = \begin{bmatrix} -2.4291 & 0.5181 & -0.3215 \\ 0.2877 & -3.1678 & -1.4806 \end{bmatrix} \quad K_6 = \begin{bmatrix} -2.4304 & 0.5427 & -0.3023 \\ 0.1611 & -3.1616 & -1.4506 \end{bmatrix}$$

$$K_7 = \begin{bmatrix} -2.3004 & 0.4526 & -0.4049 \\ 0.4167 & -3.1629 & -1.4693 \end{bmatrix} \quad K_8 = \begin{bmatrix} -2.3289 & 0.4905 & -0.3687 \\ 0.2361 & -3.1674 & -1.4437 \end{bmatrix}$$

$$K_9 = \begin{bmatrix} -2.2669 & -0.1142 & 0.5047 \\ -0.2580 & -2.0330 & -1.0839 \end{bmatrix} \quad K_{10} = \begin{bmatrix} -2.2383 & -0.2390 & 0.4783 \\ -0.1696 & -1.8729 & -0.9956 \end{bmatrix}$$

$$K_{11} = \begin{bmatrix} -2.3664 & -0.3278 & 0.3953 \\ -0.2172 & -2.0620 & -1.1002 \end{bmatrix} \quad K_{12} = \begin{bmatrix} -2.3327 & -0.4102 & 0.3825 \\ -0.1517 & -1.9008 & -1.0098 \end{bmatrix}$$

$$K_{13} = \begin{bmatrix} -2.3004 & -0.4526 & 0.4049 \\ -0.4167 & -3.1630 & -1.4693 \end{bmatrix} \quad K_{14} = \begin{bmatrix} -2.3289 & -0.4905 & 0.3687 \\ -0.2361 & -3.1674 & -1.4437 \end{bmatrix}$$

$$K_{15} = \begin{bmatrix} -2.4291 & -0.5181 & 0.3215 \\ -0.2877 & -3.1678 & -1.4806 \end{bmatrix} \quad K_{16} = \begin{bmatrix} -2.4304 & -0.5427 & 0.3023 \\ -0.1611 & -3.1616 & -1.4506 \end{bmatrix}$$

1.4.6.1 Non-noisy measurement case

The following simulation results are obtained based on initial pose of the robot $\begin{bmatrix} x & y & \theta \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{\pi}{3} \end{bmatrix}$ and initial conditions of the reference model $\begin{bmatrix} x_r & y_r & \theta_r \end{bmatrix} = \begin{bmatrix} 0.1 & -0.3 & 0 \end{bmatrix}$ knowing that the velocities of reference are defined as constants, $v_r = 0.2m/s$ and $\omega_r = 1rad/s$

Figure 1.5 shows the trajectory tracking with exactitude.

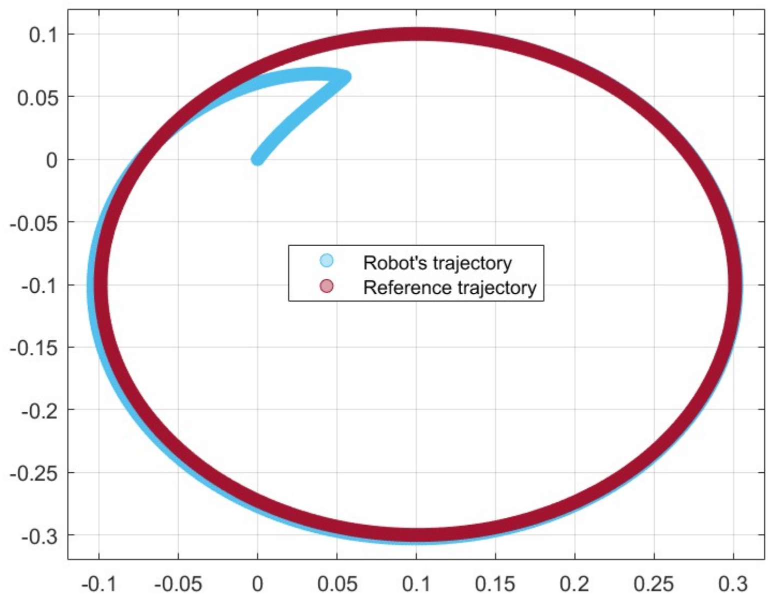


Figure 1.5: Trajectory tracking based on PDC control

The linear and angular velocities both converge to the velocities of reference as shown in figure 1.6

Figure 1.7 illustrates that the error dynamics is stabilized around the origin.

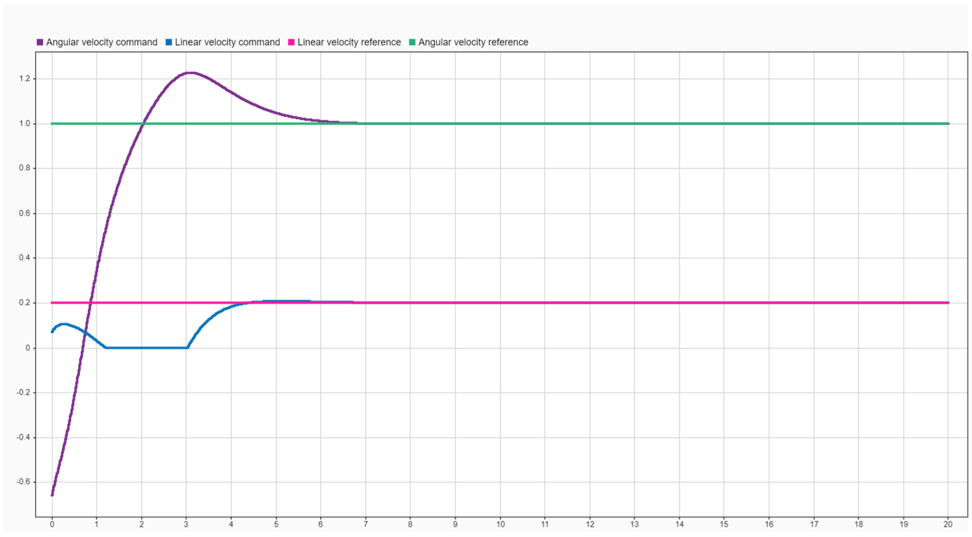


Figure 1.6: Real velocity convergence to reference velocity

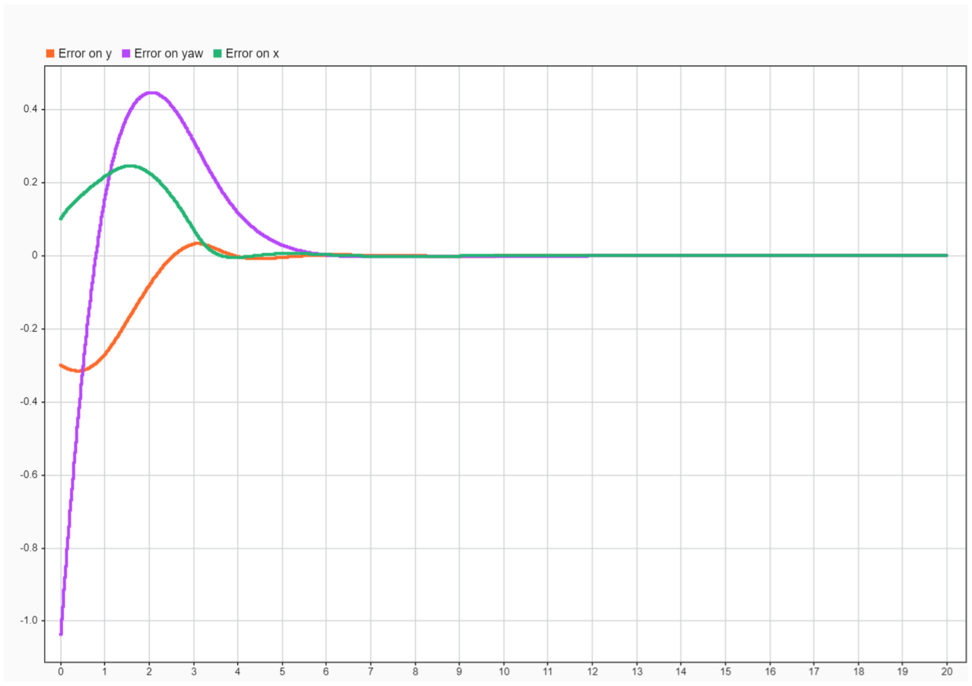
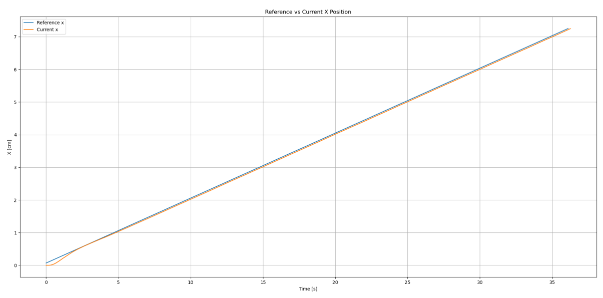
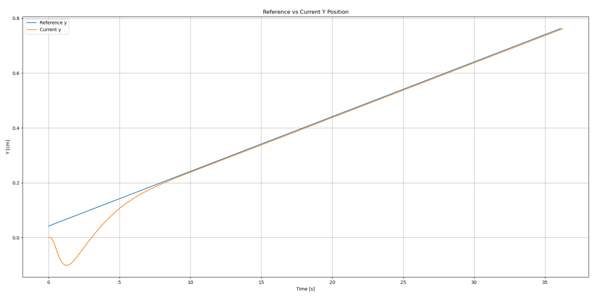


Figure 1.7: Error dynamics stabilization using PDC control

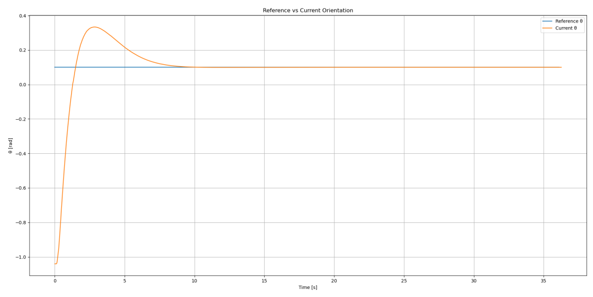
To validate the fuzzy controller, simulations were conducted using Gazebo, which provides realistic conditions that closely mimic real-world environments.



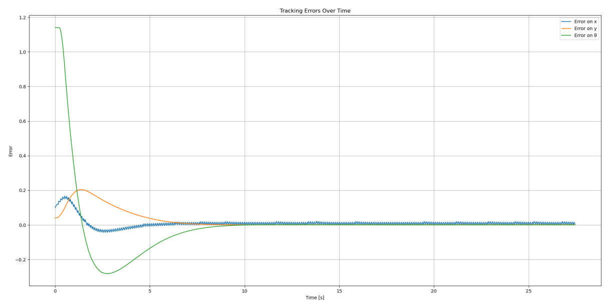
(a) Tracking on x position for linear trajectory



(b) Tracking on y position for linear trajectory.

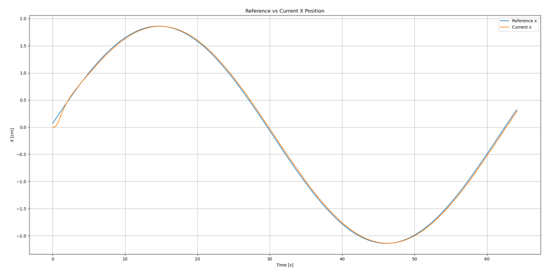


(c) Heading tracking for linear trajectory.

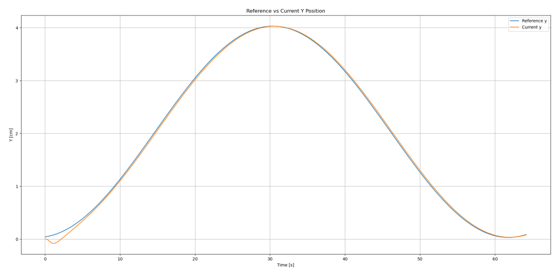


(d) Tracking error for linear trajectory.

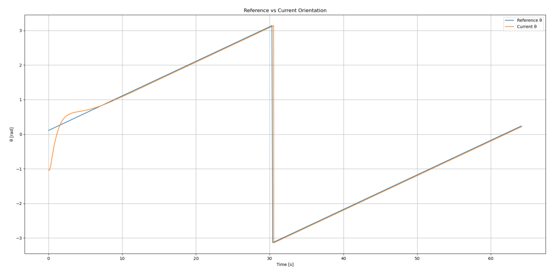
Figure 1.8: Linear trajectory tracking using fuzzy controller.



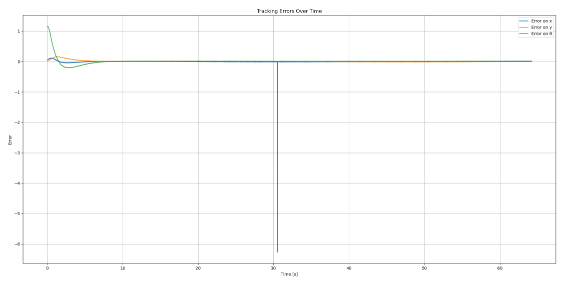
(a) Tracking on x position for circular trajectory



(b) Tracking on y position for circular trajectory.



(c) Heading tracking for circular trajectory.



(d) Tracking error for circular trajectory.

Figure 1.9: Circular trajectory tracking using fuzzy controller

Comments :

- In both trajectories, the errors in x, y, and θ stabilize around zero, with some negligible fluctuations in the x component for the linear trajectory.

- The fuzzy controller demonstrated satisfactory tracking performance, even on wavy or curved paths.

- The discontinuity observed in the heading error in the circular trajectory (Figure 1.9 (d)) is due to keeping the angle within the $[-\pi, \pi]$ range.

1.4.6.2 Noisy measurement

In a real environment, noises affect sensors measurement and lead to errors in control and navigation if it is not well addressed. As a solution, a T-S observer is added in order to filter these noises. Relying on the stabilization gains obtained in 1.4.6.1, and the observed model in 1.39, the following gains are obtained :

$$L_1 = L_3 = L_4 = L_9 = L_{10} = L_{11} = L_{12} = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.0000 & 0.0159 \\ 0.0000 & 0.0159 & 1.0000 \end{bmatrix}$$

$$L_2 = L_5 = L_6 = L_7 = L_8 = L_{13} = L_{14} = L_{15} = L_{16} = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.0000 & 0.2250 \\ 0.0000 & 0.2250 & 1.0000 \end{bmatrix}$$

Figure 1.10 represents the trajectory tracking while keeping the same initial conditions on 1.4.6.1 in addition to error dynamics observer initial conditions defined as $\begin{bmatrix} \hat{e}_x & \hat{e}_y & \hat{e}_\theta \end{bmatrix} = \begin{bmatrix} 0.2 & -0.2 & \pi \end{bmatrix}$ and adding white noise signals of 0.1 amplitude to the states.

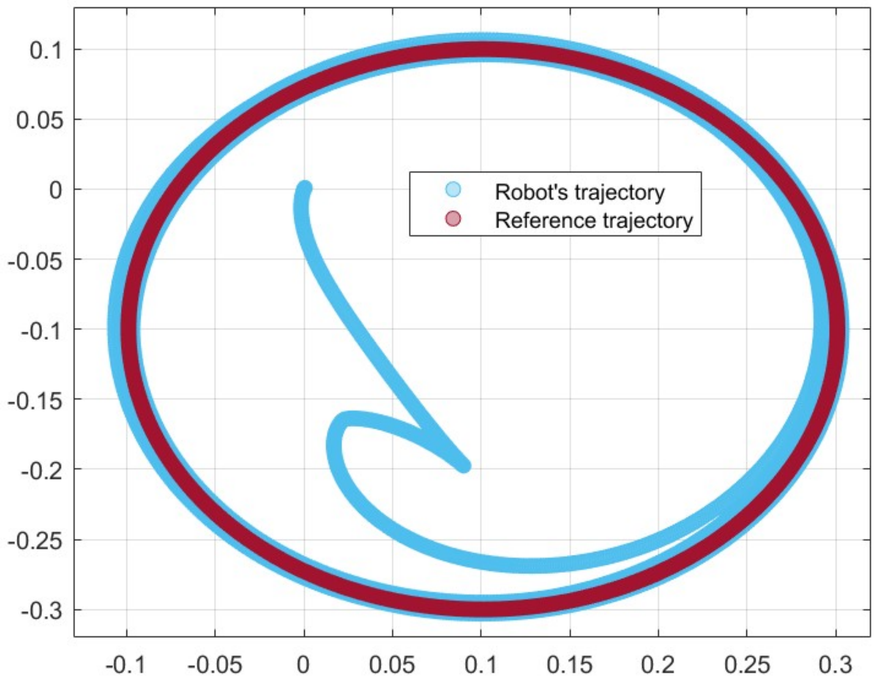


Figure 1.10: Trajectory tracking based on PDC control and T-S state observer.

Control signals, error dynamics stabilization ,and observation error are shown in figures 1.11, 1.12, and 1.13.

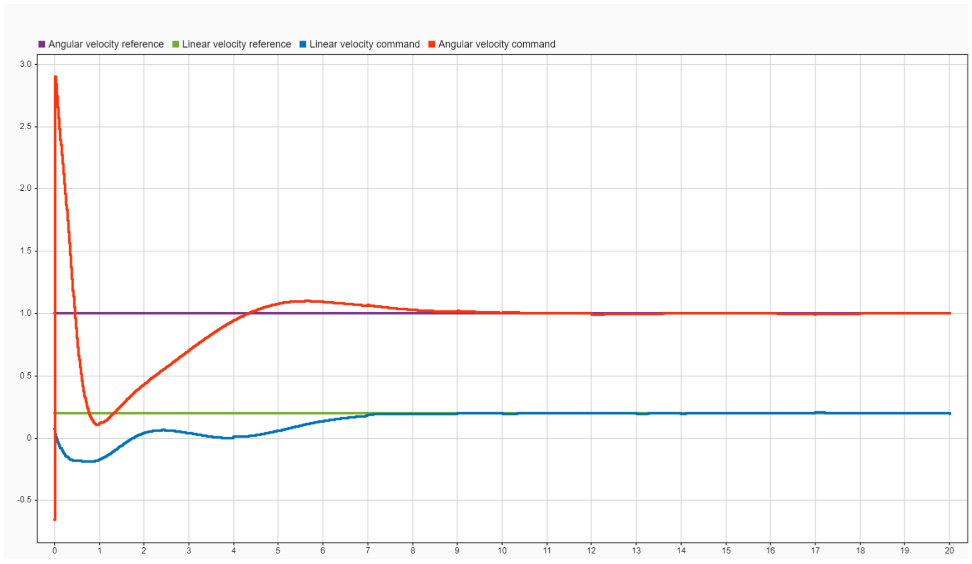


Figure 1.11: Control signal in presence of noise.

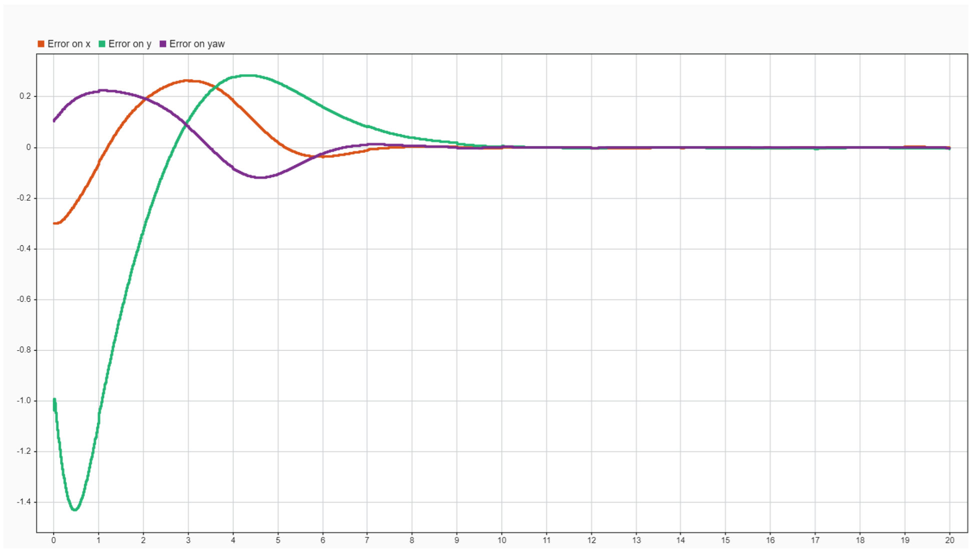


Figure 1.12: Error dynamics stabilization using PDC control and filter.

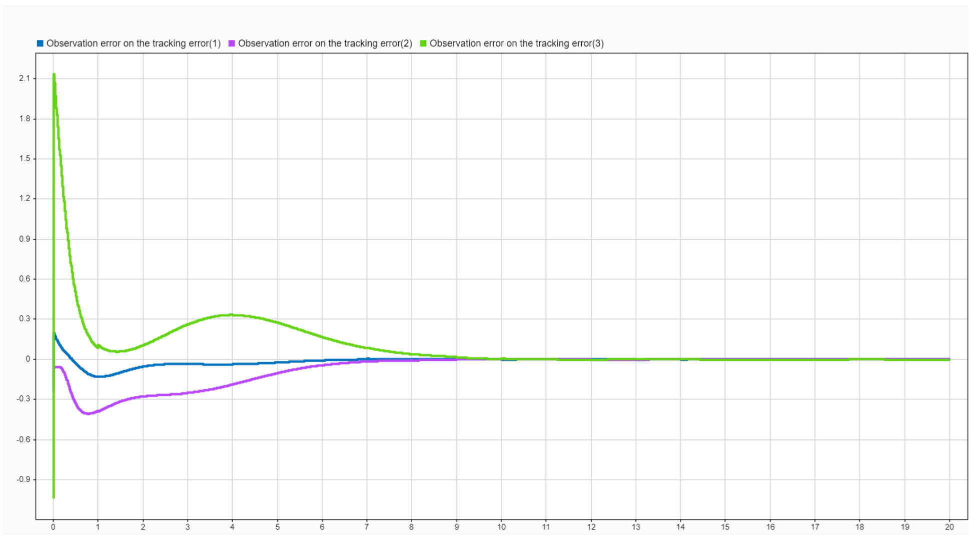


Figure 1.13: Observation error in presence of noises using PDC control and filter.

Comments

In the previous section, a Takagi-Sugeno (T-S) fuzzy controller was employed for trajectory tracking of the mobile robot. Two simulation scenarios were studied:

1. **Non-noisy case** : The T-S fuzzy controller was applied directly without any filtering. The simulation results demonstrated accurate trajectory tracking, with the robot successfully following the reference trajectory and stabilizing the error dynamics.
2. **Noisy case** : To simulate a more realistic environment, a white noise of amplitude 0.1 was added to the sensor measurements. To mitigate the effect of noise, a T-S observer was integrated with the controller to act as a filter. This observer estimated the system states accurately as the observation error tends to converge to the origin as shown in figure 1.13, leading to improved control performance. The results showed that the combined controller and observer maintained stable error dynamics and effective trajectory tracking despite the presence of noise.

To further support the choice of the T-S fuzzy controller, a kinematic trajectory tracking controller from [1] was tested on the system described in 1.27, under conditions where the initial yaw error exceeded $\frac{\pi}{2}$. The controller failed to stabilize the yaw error around the origin, as illustrated in 1.14. In contrast, the fuzzy controller successfully handled larger initial errors, demonstrating its superior robustness in such scenarios.

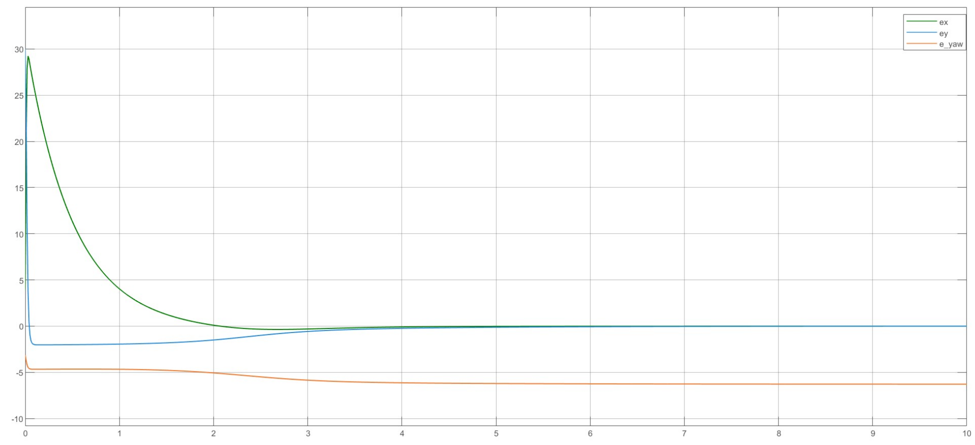


Figure 1.14: Error dynamics using controller in [1]

These simulations validate the robustness of the T-S fuzzy control approach, particularly when enhanced with a state observer for real-world scenarios involving sensor uncertainty.

1.5 Conclusion

In this chapter, a trajectory tracking controller is designed to allow mobile robots to navigate effectively in their environment. In order to ensure the desired performances, a fuzzy controller based on Takagi-Sugeno fuzzy models is considered. In this context, the method of T-S fuzzy modeling and control are well established. The controller demonstrated good tracking performance in simulation, making it a suitable component for the mapless navigation pipeline that will be addressed in the next chapters.

Chapter 2

Deep Reinforcement learning

2.1 Introduction

Reinforcement learning (RL) represents the third paradigm of machine learning in conjunction with supervised and unsupervised learning. RL is a field of machine learning (ML) that is experiencing a period of great success in the world of research applied to robotics, video games, recommendation systems, multi-agent systems..., enhanced by the advances in deep learning that allowed function approximation developing what we call now days Deep Reinforcement Learning (DRL).Figure 2.1 illustrates the paradigm of ML and where RL stands in this. Both RL and DRL consist of solving a decision-making problem passing by a sequence of trial and error where the RL or DRL agent could discover and recognize valuable decisions by penalizing the agent for taking bad actions and reinforcing it for taking good actions given by a reward signal. This interaction is similar to what humans and animals do in the real world to correct their behavior.

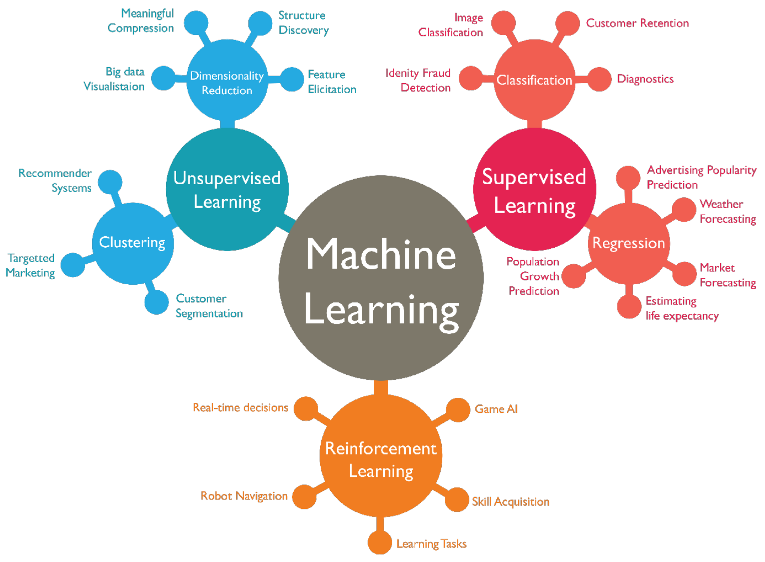


Figure 2.1: Paradigm of ML

Before diving into the discussion of the results of this thesis, we will focus on the exploration of the field of reinforcement learning. The first section begins with the definition of the notation used and with the theoretical foundations behind traditional RL. In the second it goes progressively towards DRL through an introduction to the fundamentals of DL paying more attention to algorithms used in this project.

2.2 Reinforcement Learning Framework and Example

The framework behind RL explains that problems of related learning can be reduced to three signals passing between an agent and the environment it interacts with. There are :

1. A signal to represent the choices made by the agent (actions).
2. Another signal to represent the basis on which choices are made (states).
3. A scalar signal to define the agent's goals (rewards).

States, actions, and associated rewards depend on each RL problem, while the framework associated remains the same. The goal of the RL agent is to maximize the rewards it obtains over a predefined period of time when doing the experiment. Most RL tasks can be broken down into sequences of agent-environment interactions between initial and terminal states where each sub-sequence between initial and terminal state is called an episode. The environment resets to the initial state upon reaching a terminal state.

We use the cart pole balancing problem described by Barto, Sutton, and Anderson in "Neuron-like Adaptive Elements That Can Solve Difficult Learning Control Problem". A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

States are summarized by four parameters: the position of the cart from the center, the velocity of the cart, the pole angle, and the angular velocity of the pole. The system is controlled by applying one of two discrete actions: a force of +1 to push the cart to the right, or -1 to push it to the left. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is given at every timestep the pole remains upright, encouraging the agent to keep the pendulum balanced as long as possible. An episode ends when the pole deviates more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. Thus, the longer the cart-pole remains upright within the defined limits, the more rewards the agent collects.

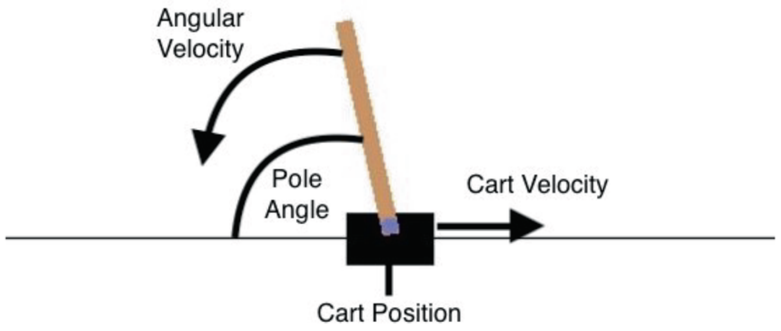


Figure 2.2: Screen capture of the OpenAI Gym CartPole problem with annotations showing the cart position, cart velocity, pole angle, and pole angular velocity

The goal of the RL agent is therefore to maximize the cumulative rewards it obtains over time.

2.3 Fundamentals of Reinforcement Learning

In a reinforcement learning (RL) problem, the decision maker is the learner and is referred to as the agent. Everything outside the agent that it interacts with constitutes the environment. The interaction occurs in discrete time steps $t = 0, 1, 2, ...$ where at each time step the agent receives a representation of the environment state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is a set of all possible states. On this basis the agent selects an action, $a_t \in \mathcal{A}(s_t)$ where $\mathcal{A}(s_t)$ is the set of actions that can be taken when in state s_t. As a consequence of the execution of this selected action, that agent transitions into a new state s_{t+1} for which it receives a reward R_{t+1}. The RL framework is shown in figure2.3 .

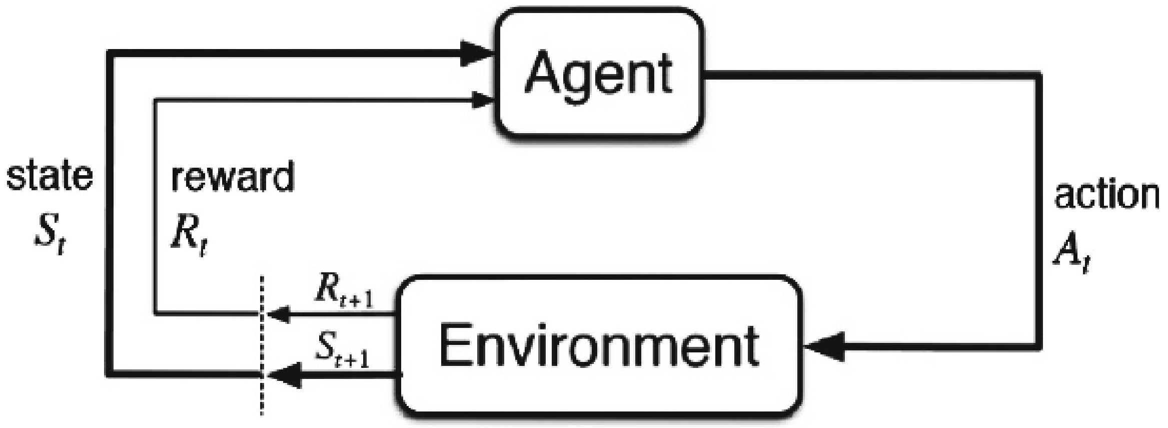


Figure 2.3: Diagram of Reinforcement Learning (RL) with main elements

Following are a number of elements that make up RL algorithms. Understanding these elements is crucial to understand the nature of the problem addressed, as well as to understand the solutions to the problem.

1. **Policy :** The policy, noted π, is a mapping from perceived states of the environment to the actions taken in those states. A policy may be a lookup table or a stochastic process. A policy is an analogous to stimulus response in psychology [11].

2. **Reward function :** A reward function, $\mathcal{R}_t(s_t, a_t)$ defines the goal in the RL problem. It gathers each state, or state-action pair to a single number that indicates the desirability of reaching a certain state. Depending on the task, the rewards can also be spare, means that the agent receives a large reward once it reaches the terminal state, and zero elsewhere. [11].

3. **Markov Decision Process (MDP) :** MDP formally describes an environment for RL where the environment is fully observable, i.e. the current state completely characterizes the process. Almost all RL problems can be formalized as MDPs even partially observable problems can be converted into MDPs[12].
   It is ideal to have a state signal that summarizes past operations compactly while retaining all the relevant information. A state s_t is said Markov if and only if :

$$\mathcal{P}[s_{t+1}|s_t] = \mathcal{P}[s_{t+1}|s_1, ..., s_t] \tag{2.1}$$

   Where $\mathcal{P}$ denotes the probability operator.
   If the Markov property holds, then given state s and action a, the probability of each possible pair of next state and reward, s', r is given by :

$$\mathcal{P}[s_{t+1} = s', \mathcal{R}_{t+1} = r|s_0, a_0, R_1, ..., s_t, a_t, R_t] = \mathcal{P}[s_{t+1} = s', \mathcal{R}_{t+1} = r|s_t, a_t, R_t] \tag{2.2}$$

If the Markov property holds true, then the environment and task as a whole is said Markov Decision Process (MDP). Taking one step in such a problem enables us to predict the next state and expected reward from the next state just by knowing the current state and action [11].

An MDP can be defined as a 5-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}_a, \mathcal{R}_a, \gamma >$ with :

- $\mathcal{S}$ is the set of possible states.
- $\mathcal{A}$ is the set of actions.
- $\mathcal{P}_a(s, s')$ is the probability of transition from state s to state s' taking action a.
- $\mathcal{R}_a(s, s')$ is the immediate reward obtained after transitioning from state s to s' taking action a.
- γ is a discount factor that prioritizes the importance of immediate reward and future rewards [13].
- R_{t+1} is the reward taken from taking one step from t to $t+1$

4. **Value function :** The value of a state represents the expected cumulative discounted reward that an agent can achieve in the future by following a specific policy starting from that state. It reflects how desirable a state is in the long term by considering both immediate rewards and the likely future states that will be encountered. Value functions, therefore, guide the agent in prioritizing actions that maximize long-term reward. Formally, the value function for a state s under a policy π is defined as [13]:

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid s_t = s\right] \tag{2.3}$$

where γ is the discount factor ($0 \leq \gamma \leq 1$), a discount factor near to zero consider maximizing one step ahead reward, on the other hand, a discount rate close to one signifies that further future rewards are also taken into consideration. R_{t+1} is the reward received at time step $t+1$, and $\mathbb{E}_\pi[\cdot]$ refers to the expected value of a random variable given a policy π with t being the time step.

5. **Action-Value function :** Also named Q-value, it estimates the expected return for each taken action given a state s following a policy π. It is represented as [13] :

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid s_t = s, a_t = a\right] \tag{2.4}$$

Where a is the action sampled from the policy π to satisfy the reward goal.

6. **Bellman equations :** Both equations. 2.3 and 2.4 satisfy recursive relationship between value of a state (pair of state-action) and the values of its successor states (pair of states-actions). These relationships are shown in 2.5 and 2.6 where a' from $\mathcal{A}$ is the next action sampled from the policy π.

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\left[G_t \mid s_t = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| s_t = s\right] \\ &= \mathbb{E}_\pi\left[\left(R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}\right) \middle| s_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} P(s', r|s, a)\left[R + \gamma V_\pi(s')\right] \end{aligned} \tag{2.5}$$

$$\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a\right] \\
&= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| s_t = s, a_t = a\right] \\
&= \mathbb{E}_\pi \left[\left(R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}\right) \middle| s_t = s, a_t = a\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} P_a(s', r|s,a) \left[R + \gamma Q_\pi(s', a')\right]
\end{aligned} \tag{2.6}$$

The goal is to find the optimal policy π^* to exploit. It can be done using Bellman optimality equations defined in 2.7 and 2.8.

$$\begin{aligned}
v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*} \left[G_t \mid S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi_*} \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E} \left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \left[R_{t+1} + \gamma v_*(s')\right]
\end{aligned} \tag{2.7}$$

$$\begin{aligned}
Q^*(s_t, a_t) &= \mathbb{E}_{s_{t+1}} \left[R_{t+1} + \gamma \max_{a'} Q^*(s', a')\right] \\
&= \sum_{s' \in \mathcal{S},\, r \in \mathcal{R}} P(s', r \mid s, a) \left[r + \gamma \max_{a'} Q^*(s', a')\right]
\end{aligned} \tag{2.8}$$

2.4 Classifying RL methods

Reinforcement Learning algorithms can be categorized in various ways depending on the underlying learning strategy and the components they rely on. The most common categorizations include:

2.4.1 Model-based vs Model-free

It is based on whether the algorithm relies on an environment model during learning.

- ***Model-based*** involves constructing a model of the environment's dynamics by collecting transition data through interaction. This model, which estimates the outcomes of actions in different states, is then used to plan future actions. The primary strength of model-based approaches lies in the agent's ability to perform virtual simulations within the state-action space, enabling it to select an optimal action based on the current state before executing it in the real environment. However, this approach comes with several limitations. Firstly, it is computationally intensive due to the overhead introduced by the planning process. Secondly, learning an accurate model can be particularly challenging in complex or stochastic environments. Inaccuracies in the learned model can lead to suboptimal decision-making and significantly degrade overall performance.

- ***Model-free*** approaches do not require learning a model of the environment to generate experiences. Their primary objective is to learn an optimal policy that maximizes a numerical reward signal. These methods are generally less computationally demanding, as they bypass the need for simulating virtual experiences. Model-free techniques such as *Monte Carlo learning* and *Temporal-Difference learning* learn directly from interactions with the environment.

- ***Monte-Carlo learning***
  Monte-Carlo based methods learn directly from complete episodes of experiences. They can only be applied to episodic tasks in which there exist a clearly defined terminal state, which would initiate a reset like board games. MC methods typically have high variances, zero bias and theoretically assured convergence properties.
- ***Temporal-Difference learning***
  TD learning is a formulation of RL algorithms. Contrary to MC learning, it schemes learn from incomplete episodes by bootstrapping existing estimates, updating a guess towards a guess[12]. It allows learning online from incomplete sequences after every step without knowing the final outcome. TD methods work for non-terminating environments. They also have low-variance and some bias, and are statically more efficient than MC methods, and sensitive to initial values. TD(0) is the simplest of TD algorithms, which updates the current estimate of the value function $V_t(s_t)$ towards the sum of the observed return $\mathcal{R}_{t+1}$ and the discounted estimated value function at the next step $V_{t+1}(s_t)$ respecting the following rule.

  $$V_{t+1}(s_t) \leftarrow V_t(s_t) + \alpha \left(R_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\right) \tag{2.9}$$

  The quantity $G_t^{(1)} = R_{t+1} + \gamma V_t(s_{t+1})$ represents the one-step target, while the corresponding one-step TD error is given by: $\delta(t) = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$.
  Since this update is directed towards the target derived from a single step, the approach is classified as a one-step TD method. Extending the temporal difference learning to n steps leads to n-step TD methods, where updates are performed towards a target representing the discounted return over n steps. The n-step returns serve as approximations of the total return over an experiment and are mathematically expressed as:

  $$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(s_{t+n}), \quad n \geq 1, \quad 0 \leq t < T - n$$

  [14].
  Algorithms such like DQN that relies on Q-learning [15] is an example of TD learning algorithms in literature.
  NB : It is often easier to learn a good policy than a good model [16], that is why model free learning is chosen in practice. Moreover TD learning methods are preferred over MC methods for their applicability to continuing tasks.

2.4.2 On and off-policy learning

RL algorithms are also categorized on the basis whether they are on-policy or off policy learning methods.

- *On-policy learning* - An on-policy method requires learning the policy π from the experience sampled from the policy itself. This means that the exploration need to be built into the policy and determine the speed of the policy improvements. [14]
- *Off-policy learning* - An off-policy learning method learns the desired target policy from experience sampled from experience sampled from a behavioral policy that the agent can

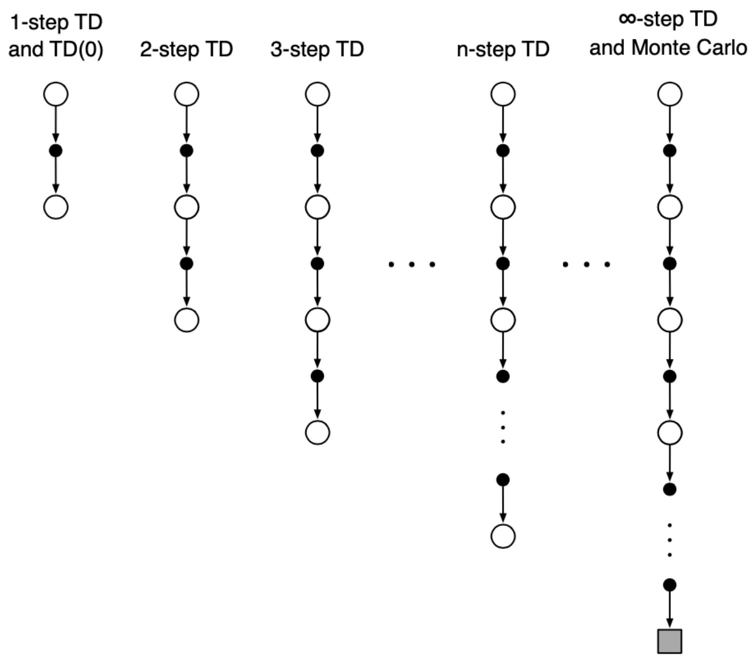


Figure 2.4: Temporal Difference Learning in Reinforcement Learning

follow. This enables a powerful exploration-exploitation balance because the agent is utilizing one policy for exploration and the another for learning.

Off-policy algorithms relying on function approximators as a representation of value functions or policy have no prove or guarantees of convergence. [13]

Experience Replay buffer
The main concept in RL is that the agent can learn form its past experiences. Past experiences include how the environment change to agent's action and are stored in from of a tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ in $D_t = \{e_1, ..., e_t\}$ where e_t contains all the steps of an episode and D_t is the data structure storing the tuples. These experiences will be accessed later with a low computational cost. During the training, samples are randomly chosen from D_t in a mini-batch and are used in off-policy learning. This breaks the correlation due to sequential observations. It also reduces simulation time and improve sample efficiency by sampling experiences from the memory instead of simulation.

2.4.3 Classifying RL - Actor/Critic

RL algorithms can also be categorized based on them being a critic-only approach where they optimize a value function, an actor-only approach which memorizes and optimizes a policy, or whether they are an actor-critic based approach, optimizing both the value and policy.

2.4.3.1 Critic-Based RL

The value function for a particular state s, when following a policy π is defined in 2.3. The state-action value function which gives information in taking a particular action being in a state s is defined in 2.4.
Value or action-value based approaches rely on optimizing for specific value function that depends on the environment's state for a better value function iteratively and implicitly producing a better policy for the agent to follow.

2.4.3.2 Actor-Based RL

Actor-only methods aim to optimize the policy function directly by adjusting its parameters to maximize expected cumulative rewards. This is typically achieved through gradient ascent on the policy's performance measure. Unlike critic-only methods, which derive policies from value functions, actor-only methods do not estimate value functions and instead focus solely on policy optimization.

2.4.3.3 Actor-Critic RL

The Actor-Critic algorithm is a reinforcement learning method that combines two approaches: the Actor, which chooses actions, and the Critic, which evaluates them. This combination helps overcome the weaknesses of using each method alone.
At the same time, the Critic checks how good the chosen actions are by estimating their value. This teamwork helps the algorithm balance trying new actions (exploration) and using what it already knows (exploitation), taking advantage of both policy and value methods.

The actor makes decisions by selecting actions based on the current policy. Its responsibility lies in exploring the action space to maximize expected cumulative rewards. By continuously refining the policy, the actor adapts to the dynamic nature of the environment.

The critic evaluates the actions taken by the actor. It estimates the value or quality of these actions by providing feedback on their performance. The critic's role is pivotal in guiding the actor towards actions that lead to higher expected returns, contributing to the overall improvement of the learning process.

Figure 2.5 illustrates the architecture of the actor-critic algorithms.

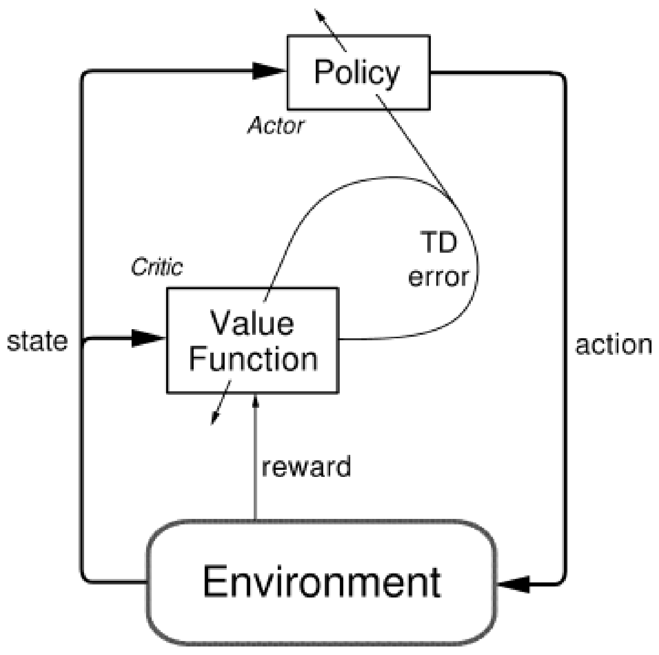


Figure 2.5: Actor-critic architecture

2.5 Deep Reinforcement Learning - DRL

The easiest way to represent a value (action-value) function is by using a lookup table that stores each state (state-action pair) along side with its entry $V(s)$ $(Q(s,a))$. A problem of storage occurs when the state-action spaces are large as it requires heavy computational resources. To solve this problem, function approximators can be used instead of lookup tables to represent values functions, thus reducing memory usage and accelerating the learning process and also generalize from seen states to unseen states[12] and [11].

Both value and action-value function can be represented as follows in equation 2.10.

$$\begin{aligned} \hat{v}(s,w) &\approx V_\pi(s) \\ \hat{q}(s,a,w) &\approx Q_\pi(s,a) \end{aligned} \tag{2.10}$$

Where w is a parameter vector for the function approximators given by $\hat{v}$ and $\hat{q}$. The extra parameter w is updated by a learning algorithm such as the *Monte-Carlo* method or the *Temporal Difference Learning* both disccussed in the next section. The function approximator can be linear or neural networks.

The linearly approximated value function is defined as:

$$\hat{v}(s,w) = x(s)^\top w = \sum_{j=1}^{n} x_j(s) w_j \tag{2.11}$$

where $x(s) = \Big(x_1(s), \ldots, x_n(s)\Big)^\top$ is the feature vector representing state s.

The goal of training is to find a parameter vector w that minimizes the mean squared error between the true value $V_\pi(s)$ and the approximated value $\hat{v}(s,w)$. This can be formulated as the following objective function:

$$J(s,w) = \mathbb{E}_\pi \left[(V_\pi(s) - \hat{v}(s,w))^2 \right] \tag{2.12}$$

To minimize $J(s,w)$, we apply gradient descent. The update rule for the parameters is:

$$\Delta w = -\frac{1}{2} \alpha \nabla_w J(w) \tag{2.13}$$

Expanding the gradient, the update becomes:

$$\Delta w = \alpha \, \mathbb{E}_\pi \left[(V_\pi(s) - \hat{v}(s,w)) \, \nabla_w \hat{v}(s,w) \right] \tag{2.14}$$

Using a single sample estimate, the update simplifies to:

$$\Delta w = \alpha \left(V_\pi(s) - \hat{v}(s,w) \right) \nabla_w \hat{v}(s,w) \tag{2.15}$$

In practice, the expected value is estimated using *Monte-Carlo* or *T-D* learning methods.

Nowdays, since they are widely used in research, neural networks are the most intuitive option to take as function approximator. It reduces training time for large MDPs and requires less computations and memory usage. Thanks to the advancement in deep learning, neural networks become the fundamental tool to exploit as function approximator to develop what we call today DRL which achieved remarkable results.

2.5.1 Fundamentals of Deep Learning - DL

Artificial neural networks

DL is an approach to learning based of a function $f\ :\ x \rightarrow y$ parametrised with $w \in R^n |\ n \in N$ such that $y = f(x, w)$.
The neuron elaborates the inputs by taking the weighted sum, adding a bais b and applying an activation function f following the relation $y = f(\sum_n wx_i + b)$. The set of parameters w and b need to be adjusted to find a good parameter set, this is learning.
Figure 2.6 describes the best what a deep neural network is with one fully-connected hidden layer.

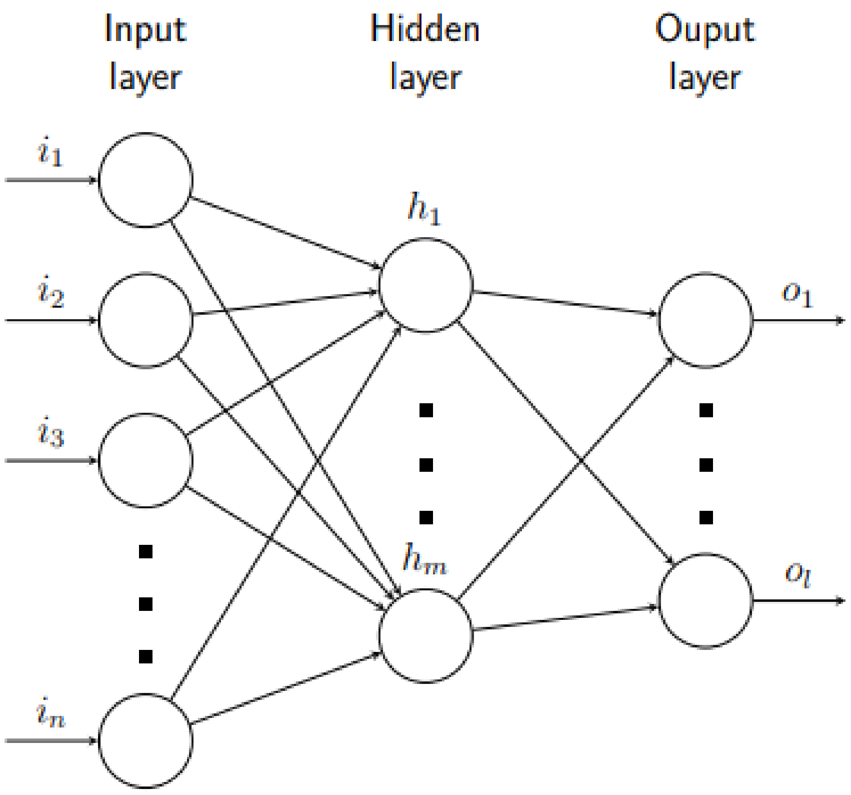


Figure 2.6: An example representation of a deep neural network with one fully-connected hidden layer

$$\begin{aligned} h &= g(W_1 \cdot i + B_1) \\ o &= W_2 \cdot h + B_2 \end{aligned} \tag{2.16}$$

The input layer receives as input a column vector of input features **i** of size $n \in \mathbb{N}$. Each value in the hidden layer, represented by a vector **h** of size $m \in \mathbb{N}$, is obtained through a transformation of the input values as described by equation 2.16 where, W_1 is a weight matrix of size $m \times n$ and $\mathbf{b}_1$ is a bias vector of size m. The function g denotes a non-linear parametric activation function, which forms the core of neural network computations. Subsequently, the hidden layer output **h** undergoes a second transformation to produce the output layer values, as described in equation 2.16 using weights $W_2 \in \mathbb{R}^{n_o \times n_h}$ and bias $b_2 \in \mathbb{R}^{n_o}$, where n_o denotes the size of the output layer and n_h the size of the hidden layer.

Activation functions

Activation functions play a key role in training deep neural networks by deciding whether a neuron should be activated based on its input. They are mathematical functions applied to each neuron's output. In deep networks, non-linear activation functions are commonly used to enable the model to learn complex, non-linear relationships between inputs and outputs. This is especially important when dealing with data like images, video, or audio, where such relationships are not simple or linear [17].
There exists several activation functions in literature, only those used in this project will be listed :

- **Sigmoid function** known as logistic function maps any input value to a value between 0 and 1. It has the advantage of leading to clear divisions, as it tends to produce results near the limits of its range. Its output can be directly interpreted as a probability. The sigmoid function is given by :
$$s(x) = \frac{1}{1 + e^{-x}}$$
where x is the output value of the neuron.

- **Hyperbolic tangent function** (tanh) maps any input value to a value between -1 and 1. It is used in problems where the output can be negative. It is given by :
$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
The advantages mentioned for the sigmoid function also apply to this one, as it is a scaled and shifted version of the former.
$$tanh(x) = 2s(2x) - 1$$
Figure 2.7 illustrates both sigmoid and tanh plots and the likeness between the two functions.

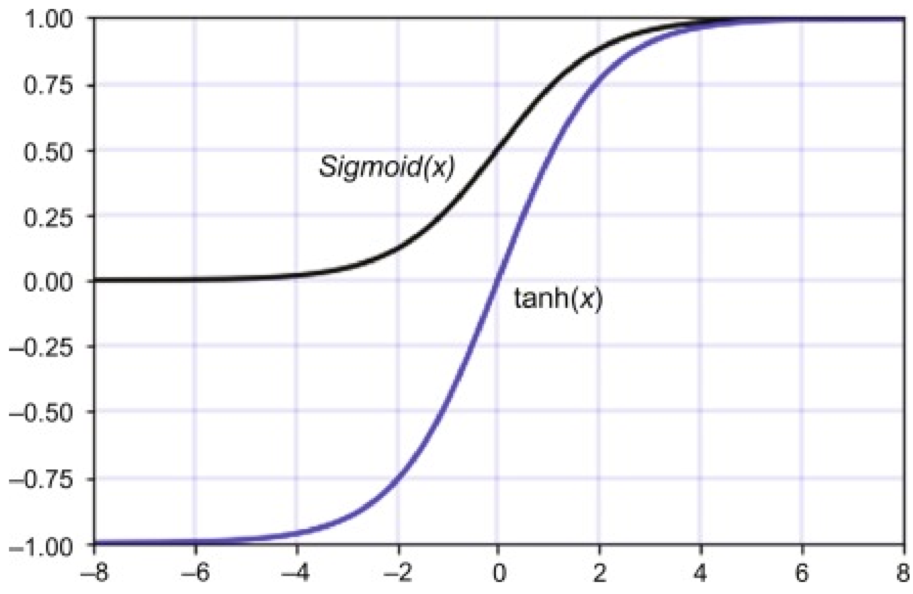


Figure 2.7: Sigmoid and Hyperbolic tangent functions plot

Learning process

The learning process of a neural network is an iterative process in which the computations are performed forward and backward through the network's layers until the loss function is minimized in order to get the best parameters W_i and B_i.
The learning process has three main steps illustrated in figure 2.8 :

- **Forward propagation**.
- **Loss function calculation**.
- **Backward propagation**.

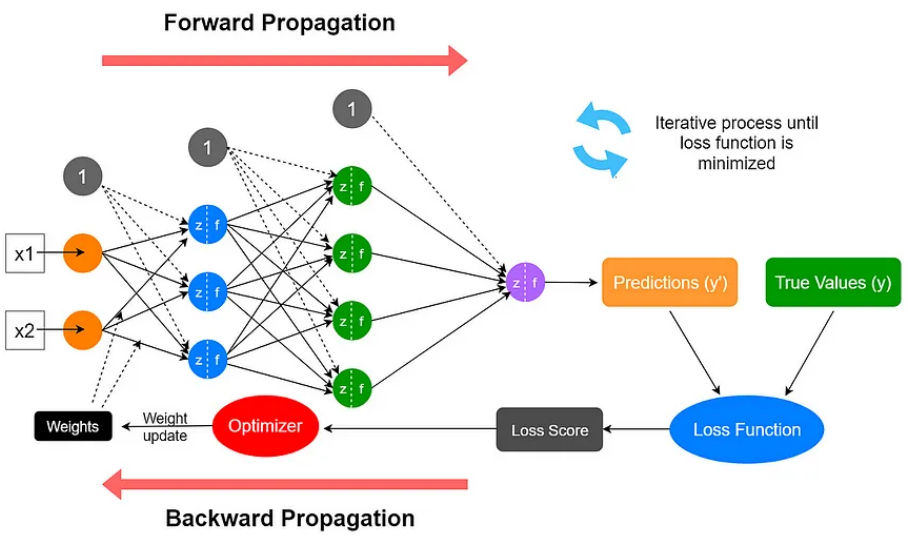


Figure 2.8: Learning process of a neural network

Forward Propagation
In forward pass, input data is fed into the neural network, calculations are performed layer by layer until the output is computed. Each neuron recieves input signals, applies an activation function, and passes the result to neurons in the next layer. This continues to the last layer that generates the final output. This step is responsible for predictions based on the current parameters (weights w and bias b).

Loss function calculation
Once the output is computed, the next step is to evaluate the performance of the model by comparing its predictions with the actual target values from training data. This is not possible without a loss function also known as a cost function, which quantifies the predicted action and the truth. Most used loss function is mean squared error (MSE). Minimizing the loss function is the goal during the training, hence improving the model accuracy.

Backward propagation
After computing the loss function, the network needs to adjust its parameters to minimize the loss and improve performance. Backward propagation, also known as backpropagation, is the process of these updates by calculating the gradient of the loss with respect to each parameter. This propagates the error backward through the network, layer by layer. The gradients are then used to update the parameters using optimization algorithms like gradient descent or its

variants. By iterating this process, the network progressively learn to make more accurate predictions and minimize errors.

Overall, the three crucial steps together enable the network to learn from data, optimize its parameters, and improve its performance overtime, making the model capable of solving complex problems.

2.5.2 DRL Agents

There are two main types of reinforcement learning agents: those designed for discrete action spaces and those for continuous action spaces. In discrete tasks, the agent chooses from a fixed set of actions. But in continuous tasks, the agent must select actions from a range of values, which makes the problem more complex because the number of possible actions is infinite.

This project focuses mainly on agents that handle continuous control tasks, special algorithms are used, such as Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO). These agents combine neural networks with value functions or/and policy optimization to learn how to act in continuous environments.The DQN algorithm, which is made for discrete action spaces, will also be tested. It uses a deep neural network to learn the action-value function, helping the agent choose the best action from a fixed set.

Deep Q-Network Agent

The Deep Q-Network (DQN) algorithm is an off-policy reinforcement learning method designed for environments with discrete action spaces. It learns a Q-value function that estimates the expected long-term reward for each action when following the optimal policy [18]. DQN improves standard Q-learning by using a target network for stability learning and an experience buffer to store past transitions [4]. During training, the agent performs the following actions:

- Updates the critic network parameters at every time step.
- Explores the action space using an epsilon-greedy strategy: with probability ϵ, it selects a random action; otherwise, it chooses the action with the highest estimated value.
- Stores past experiences in a circular buffer and learns from random mini-batches sampled from it to update the critic.

To estimate the value of the optimal policy, a DQN agent uses two action-value functions, each represented by a neural network:

- **Critic** $Q(S, A; \phi)$: This network takes the current state S and action A, and estimates the expected cumulative future reward. It represents the value of the optimal policy.
- **Target Critic** $Q_t(S, A; \phi_t)$: To make training more stable, a second network is used. It is periodically updated with the latest parameters from the main critic network.

Both networks have the same architecture and are trained using function approximation.

DQN Algorithm [4] is detailed in algorithm 1.

Deep Deterministic Policy Gradient Agent

The Deep Deterministic Policy Gradient (DDPG) algorithm is an off-policy actor-critic method used in environments with continuous action spaces. It learns a deterministic policy with the help of a Critic that estimates the value of actions. As an actor-critic method, it trains two networks: one for choosing actions (actor) and one for evaluating them (critic)[18].

During training, a DDPG agent performs the following steps:

- Updates the actor and critic learnable parameters at each time step during learning.
- Stores past experiences using a circular experience buffer. The agent updates the actor and critic using a mini-batch of experiences randomly sampled from the buffer.
- Perturbs the action chosen by the policy using a stochastic noise $\mathcal{N}$ at each training step.

To learn the policy and value function, a DDPG agent uses four neural networks:

- **Actor** $\mu(s|\theta^{\mu})$: This network takes the current observation S and returns the best action to take, aiming to maximize long-term reward. Here, π is a deterministic policy (not a probability distribution).
- **Target Actor** $\mu'(s|\theta^{\mu'})$: A copy of the actor network, updated slowly to improve training stability by using the latest actor parameters.
- **Critic** $Q(s,a|\theta^{Q})$: This network estimates how good a given action A is in a given state S, using parameters ϕ.
- **Target Critic** $Q'(s,a|\theta^{Q'})$: A copy of the critic network, also updated gradually to stabilize the training.

Both the actor and target actor networks have the same structure. The same applies to the critic and target critic networks. DDPG algorithm [5] is detailed in algorithm 2.

Twin Delayed DDPG

The Twin-Delayed Deep Deterministic Policy Gradient (TD3) algorithm is an off-policy actor-critic method designed for environments with continuous action spaces. A TD3 agent learns a deterministic policy through an actor network and uses two critic networks to estimate the value of the optimal policy more accurately. To improve stability, it also includes target networks for both the actor and critics. Past experiences are stored in a replay buffer and used for training. Additionally, TD3 supports offline learning from previously collected data without needing real-time interaction with the environment [18].
The TD3 algorithm is an improved version of the DDPG algorithm. DDPG agents can sometimes overestimate the value function, which can lead to poor policies. To address this, TD3 introduces several changes. First, it uses two Q-value networks and takes the smaller of the two estimates to reduce overestimation during updates. Second, TD3 updates the policy and target networks less often than the Q-value networks, which helps stabilize learning. Finally, when updating the policy, TD3 adds noise to the target action to prevent the policy from exploiting overly optimistic value estimates [18].

A TD3 agent maintains the following function approximators to estimate the policy and value functions:

- **Actor network** $\pi(S;\theta)$: A deterministic policy network parameterized by θ, which maps the current state S to an action A. This policy does not represent a distribution, but a function that outputs the action that maximizes the expected return.
- **Target actor network** $\pi_t(S;\theta_t)$: A copy of the actor network used to stabilize training. Its parameters θ_t are periodically or softly updated using the main actor's parameters.
- **Critic networks** $Q_k(S,A;\phi_k)$: One or two critic networks (e.g., Q_1, Q_2) parameterized by ϕ_k, which estimate the expected return for a given state-action pair (S,A). Each critic may have a different structure, though TD3 performs best when they are structurally identical but initialized with different weights.
- **Target critic networks** $Q_{t,k}(S,A;\phi_{t,k})$: Corresponding target networks for each critic, updated from the main critic parameters ϕ_k to reduce variance and improve stability. The number of target critics matches the number of main critics.

Both the actor and target actor share the same architecture and parameterization. Similarly, each critic and its target have the same structure. When two critics are used, they should have different initial parameters even if their architectures match, to avoid overestimation bias.

TD3 algorithm [6] is detailed in algorithm 4.

2.5.2.1 Soft Actor-Critic

The Soft Actor-Critic (SAC) algorithm is an off-policy actor-critic method that works with discrete, continuous, and hybrid action spaces. It learns a stochastic policy that aims to maximize both the expected long-term reward and the policy entropy. Entropy measures how uncertain or random the policy is; higher entropy encourages more exploration. By maximizing both reward and entropy, SAC balances exploration and exploitation. The agent uses two critic networks to estimate the value of the optimal policy, along with target critics and a replay buffer [18].
During training, a Soft Actor-Critic (SAC) agent performs several steps.

- The SAC agent regularly updates the actor and critic network parameters during training.
- It models a probability distribution over actions and selects actions randomly from this distribution to encourage exploration.
- The agent adjusts an entropy temperature parameter to keep the policy entropy close to a target value, maintaining a balance between exploration and exploitation.
- Past experiences are stored in a circular replay buffer.
- The agent uses mini-batches randomly sampled from this buffer to update the actor and critic networks, improving learning stability.

To estimate the policy and value function, a DDPG agent maintains four function approximators:

- **Actor** $\pi(S;\theta)$ — The actor, with parameters θ, takes observation S and returns the corresponding action that maximizes the long-term reward. Note that π here does not represent a probability distribution, but a function that returns an action.

- **Target actor** $\pi_t(S;\theta_t)$ — To improve the stability of the optimization, the agent periodically updates the target actor learnable parameters θ_t using the latest actor parameter values.
- **Critic** $Q(S,A;\phi)$ — The critic, with parameters ϕ, takes observation S and action A as inputs and returns the corresponding expectation of the long-term reward.
- **Target critic** $Q_t(S,A;\phi_t)$ — To improve the stability of the optimization, the agent periodically updates the target critic learnable parameters ϕ_t using the latest critic parameter values.

Both $Q(S,A;\phi)$ and $Q_t(S,A;\phi_t)$ have the same structure and parameterization, and both $\pi(S;\theta)$ and $\pi_t(S;\theta_t)$ have the same structure and parameterization.

SAC algorithm [7] is detailed in algorithm 3.

Figure 2.9 illustrates the classification of DRL algorithms and provides representative examples for each category.

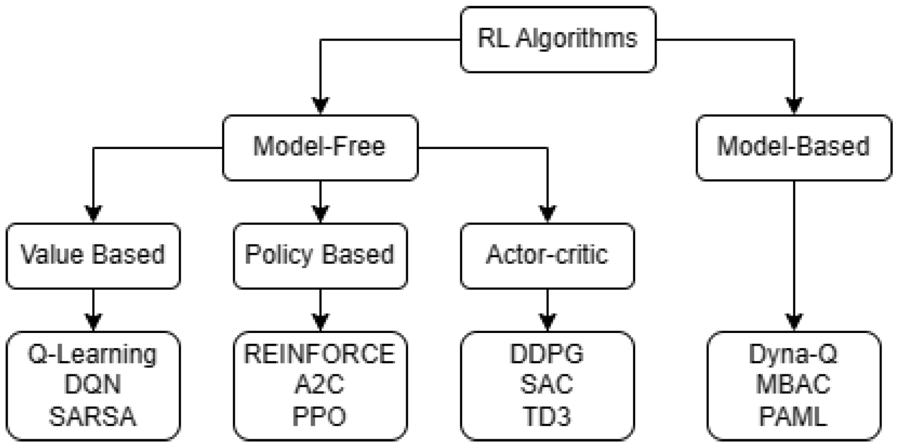


Figure 2.9: DRL Algorithms classification

2.5.2.2 DRL Agents algorithms

Algorithm 1 Deep Q-learning with experience replay

```
 1: Initialize replay memory $D$ to capacity $N$
 2: Initialize action-value function $Q$ with random weights $\theta$
 3: Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
 4: for episode = 1 to $M$ do
 5:     Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
 6:     for $t = 1$ to $T$ do
 7:         With probability $\epsilon$ select a random action $a_t$
 8:         otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
 9:         Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
10:         Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
11:         Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
12:         Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
13:         Set
                $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
14:         Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network
            parameters $\theta$
15:         Every $C$ steps reset $\hat{Q} = Q$
16:     end for
17: end for
```

Algorithm 2 Deep Deterministic Policy Gradient (DDPG)

```
1: Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
2: Initialize target networks $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
3: Initialize replay buffer $R$
4: for episode = 1 to $M$ do
5:     Initialize a random process $\mathcal{N}$ for action exploration
6:     Receive initial observation state $s_1$
7:     for $t = 1$ to $T$ do
8:         Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
9:         Execute action $a_t$, observe reward $r_t$ and new state $s_{t+1}$
10:        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
11:        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
12:        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13:        Update critic by minimizing the loss:
               $$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$
14:        Update the actor policy using the sampled policy gradient:
               $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
15:        Update the target networks:
               $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
               $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
16:    end for
17: end for
```

Algorithm 3 Soft Actor-Critic

```
1: Input: $\theta_1, \theta_2, \phi$
2: $\hat{\theta}_1 \leftarrow \theta_1$, $\hat{\theta}_2 \leftarrow \theta_2$
3: $\mathcal{D} \leftarrow \emptyset$
4: for each iteration do
5:     for each environment step do
6:         $a_t \sim \pi_\phi(a_t|s_t)$
7:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
8:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
9:     end for
10:    for each gradient step do
11:        $\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
12:        $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
13:        $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha J(\alpha)$
14:        $\hat{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\hat{\theta}_i$ for $i \in \{1, 2\}$
15:    end for
16: end for
17: Output: $\theta_1, \theta_2, \phi$
```

Algorithm 4 TD3

```
1: Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$ with random parameters $\theta_1$, $\theta_2$, $\phi$
2: Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
3: Initialize replay buffer $\mathcal{B}$
4: **for** $t = 1$ to $T$ **do**
5:     Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$
6:     $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
7:     Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$
8:     **Sample** mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
9:     $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
10:    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
11:    Update critics $\theta_i \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
12:    **if** $t$ mod $d = 0$ **then**
13:        Update $\phi$ by the deterministic policy gradient:
               $$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$$
14:        Update target networks:
               $$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i, \quad \phi' \leftarrow \tau\phi + (1 - \tau)\phi'$$
15:    **end if**
16: **end for**
```

2.6 Conclusion

This chapter introduced the theoretical basis of Deep Reinforcement Learning (DRL), summarizing its core components, such as the agent- environment interaction, reward, and the use of deep neural networks to approximate value functions and policies. We classified the main categories of reinforcement learning algorithms and provided a quick overview of the most widely used DRL agents. These concepts will serve later as a prerequisite for implementing DRL in mapless navigation which will be addressed in the following chapters.

Chapter 3

Deep Reinforcement Learning Based Mapless Navigation

3.1 Intoduction

Deep Reinforcement Learning (DRL) is a powerful learning paradigm that merges the principles of reinforcement learning with deep neural networks, enabling agents to learn complex decision-making tasks in high-dimensional, continuous, and unstructured environments. In DRL, an agent interacts with the environment by perceiving a state, selecting an action, and receiving a reward based on the consequences of its actions. Over time, through trial and error, the agent learns a policy that maximizes cumulative rewards. The training process involves foundational concepts such as value functions, policy gradients, and experience replay buffers, which together enable efficient learning and generalization [13].

As discussed in the previous chapter, DRL algorithms such as Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), and Soft Actor-Critic (SAC) have emerged as state-of-the-art tools for learning control policies directly from raw sensory inputs. These algorithms are particularly effective in continuous control tasks often encountered in robotics, where actions must be selected in real time and under uncertainty or partial observability [5]. The ability of DRL to operate directly on noisy sensor data and adapt to dynamic environments makes it a suitable choice for autonomous mobile robots.[19] One particularly challenging and increasingly relevant application of DRL in robotics is mapless navigation. In contrast to traditional methods that rely on a pre-built map of the environment—such as those using Simultaneous Localization and Mapping (SLAM)—mapless navigation requires the robot to make decisions solely based on current sensor readings and a goal position, without any prior knowledge of the environment. This is critical in scenarios such as search and rescue missions or dynamic public spaces, where the environment is either unknown, constantly changing, or too complex to model in advance.

Traditional path planning methods—such as A*, D*, or sampling-based planners—have been widely used for robotic navigation. However, these methods often depend on hand-crafted heuristics or constraint functions that need to be tuned for specific scenarios. While effective in well-structured environments, such engineered pipelines tend to lack the adaptability required for diverse, unpredictable conditions. Excessive reliance on hand-engineered rules can severely limit a robot's ability to generalize and perform reliably in unseen environments.

To address these limitations, learning-based navigation methods have emerged as a promising alternative. Supervised learning approaches, which train models from expert demonstrations, have shown some success but require extensive labeled datasets, which are often impractical to collect. In contrast, reinforcement learning allows a robot to autonomously learn navigation strategies by interacting directly with the environment. Through reward feedback, the agent gradually improves its behavior, learning to reach its target while avoiding collisions and unsafe paths.

Recent research in DRL-based mapless navigation demonstrates that agents can learn to navigate efficiently in unknown environments without relying on global maps. By learning end-to-end policies from sensor data such as sparse LiDAR readings or monocular images—robots can develop behaviors that generalize across different environments and obstacle configurations, eliminating the need for extensive hand-coded logic or rule sets.

3.2 Related works

In recent years, significant progress has been made in the application of Deep Reinforcement Learning (DRL) to mapless navigation tasks. Various research efforts have explored how DRL can be utilized to enable mobile robots to autonomously navigate in unknown environments without relying on a global map.

In the context of mobile robotics, autonomous navigation through unknown environments remains a challenging and critical task. The robot must be capable of identifying a collision-free path without prior knowledge of the environment. Traditionally, classical navigation pipelines rely on three main components: localization, map construction, and path planning. These systems typically depend on dense and precise laser range sensors, such as LiDAR, for accurate localization and Simultaneous Localization and Mapping (SLAM). However, such approaches are not only computationally intensive but also rely on expensive hardware and precise sensory data, which can limit their scalability and deployment in resource-constrained settings.

Current reinforcement learning (RL) methods are often computationally intensive and time-consuming, as they typically require millions of interactions with the environment to learn complex tasks effectively. Conducting such extensive training directly in the real world is generally infeasible due to practical constraints, safety risks, and hardware wear. As a result, RL agents are commonly trained in simulated environments where the training process can be automated, parallelized, and executed safely. Simulation not only accelerates the learning process but also prevents physical damage to robotic platforms that might otherwise occur during trial-and-error interactions, such as collisions with obstacles.[20]

In the following discussion, we review laser sensor-based reinforcement learning approaches for mapless autonomous navigation of mobile robots. These methods leverage sparse or dense laser range data to enable real-time perception and decision-making without the need for an explicit map of the environment.

Authors in [21] used Advantage Actor Critic (A2C) RL method, and 30 values from laser ranging sensors along with target relative distance and angle to target as observation input to directly map actions commands from a set of discrete actions space. The RL agent is trained using PyGames and Chipmunk 2D simulator that lacks of real physical properties. A lack of stability is noticed while training for autonomous mobile robots using DRL methods. Authors in [22] use efficient policy gradient method, Proximal Policy Optimization (PPO) to learn navigation policy. A convolution deep neural network was used to point 720 laser scan data and relative pose to target to discrete actions to reduce. However the learning was not stable hence the learnt policy will not generalize in outdoor environments.

In the study by Taheri et al.[23], the authors propose a mapless navigation strategy for mobile robots using an enhanced Proximal Policy Optimization (PPO) algorithm. The agent receives a poor observation space compared to the study in [22].Tahari et al composed their state observation of laser scan data, previous velocity commands, and the relative position of the goal. Training is performed in the Gazebo simulation environment, integrated with ROS, and the authors design a task-specific reward function to encourage safe and efficient goal-reaching behavior. The policy directly outputs continuous velocity commands without relying on global maps. While the approach demonstrates strong performance in avoiding collisions and navigating in cluttered static environments, the learned policy generalizes only to static obstacles and does not handle dynamic environments effectively.

Although recent studies have shown promising results using deep reinforcement learning for mapless navigation, most approaches still face generalization limitations, particularly in dynamic environments. These limitations form the core motivation of our work: to develop a more adaptable and robust navigation strategy capable of operating effectively in dynamic settings. The objective, consistent with previous works, is to learn a policy that maps real-time sensor observations to motion commands, allowing the robot to navigate safely and efficiently toward a target location in the presence of both static and dynamic obstacles. To this end, a formal problem formulation is presented in the following section.

3.3 Problem formulation

In this work, we address the problem of autonomous mapless navigation for a mobile robot using DRL. The goal is to enable the robot to navigate safely and efficiently in unknown environments toward a defined target location using LiDAR sensor observations assuming that localization system is accurate without relying on any pre-existing map or global localization system. This problem is formalized as a Markov Decision Process (MDP), where the agent interacts with its environment through decision making based on partial observations.

3.3.1 Simulation environment

The robot used in this study is a differetial drive mobile robot equipped with a 2D LiDAR sensor from turtlebot serie shown in figure 3.1. TurtleBot was chosen as it is widely used in

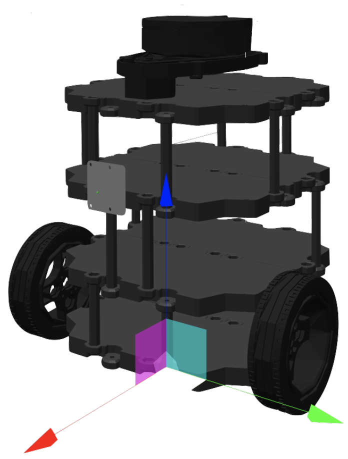


Figure 3.1: Turtle bot 3 burger gazebo model

the research which helps in comparing the findings. The LiDAR provides laser scan data for obstacle perception. The robot can only move forward and rotate, and its motion is controlled through linear and angular velocity commands. Odemetry data is supposed ideal but not used to build a map.

Before diving into environment definition, we will first justify the choice of the couple Gazebo-ROS2. Gazebo offers a high-fidelity physics engine and realistic sensor simulation which are essential for accurately modeling robot-environment interactions such as collision dynamics, motion constraints, and sensor noise. This is particularly important for RL, where the success of the approach using real robot depends on the accuracy of simulated interactions.
Before diving into the environment definition, we first justify the choice of the Gazebo–ROS 2 combination.

Gazebo offers a high-fidelity physics engine and realistic sensor simulation, which are essential for accurately modeling robot–environment interactions such as collision dynamics, motion constraints, and sensor noise. This level of realism is particularly important in reinforcement learning, where the success of transferring a trained agent to a real robot depends heavily on the accuracy of the simulated interactions [24].

ROS 2 provides robust tools for robot control, sensor data handling, and modular system integration, making it easier to build, test, and transition navigation systems from simulation to real-world deployment. Compared to its predecessor, ROS 2 improves communication reliability, supports real-time constraints, and offers a scalable architecture suitable for complex robotic applications [25].

Together, Gazebo and ROS 2 form a powerful simulation and control stack that supports seamless interaction between the trained agent and future real-world robot applications.

Since training can not be conducted using a real robot platform, training and evaluation will be conducted entirely in simulation using Gazebo simulator integrated with ROS2. To ensure scalability and generalization of the learned navigation policy, a multi stage environments are designed from the simplest to the more complex.

- **Stage 1**: Indoor environment with no obstacles.
- **Stage 2**: Indoor environment with static cylinders as obstacles.
- **Stage 3**: Indoor environment with the cylinders from stage 2 moving in a circular trajectory.
- **Stage 4**: Indoor environment with statical walls placed arbitrary in the square area.
- **Stage 5**: Indoor environment with the statical walls from stage 4 in addition to moving cylinders with a random trajectory.

Each stage is chosen to increase difficulty incrementally to enable curriculum-style learning.

figure 3.2 shows the environments chosen for training.

The architecture of our mapless navigation system using DRL, illustrated in figure 3.3, serves as the foundation for the experiments conducted in this study. The system is based on a simulated environment integrated with OpenAi Gym und utilizes Stable Baselines for the implementation of the DRL algorithms. All tools, libraries, and frameworks used in this project are open-source.

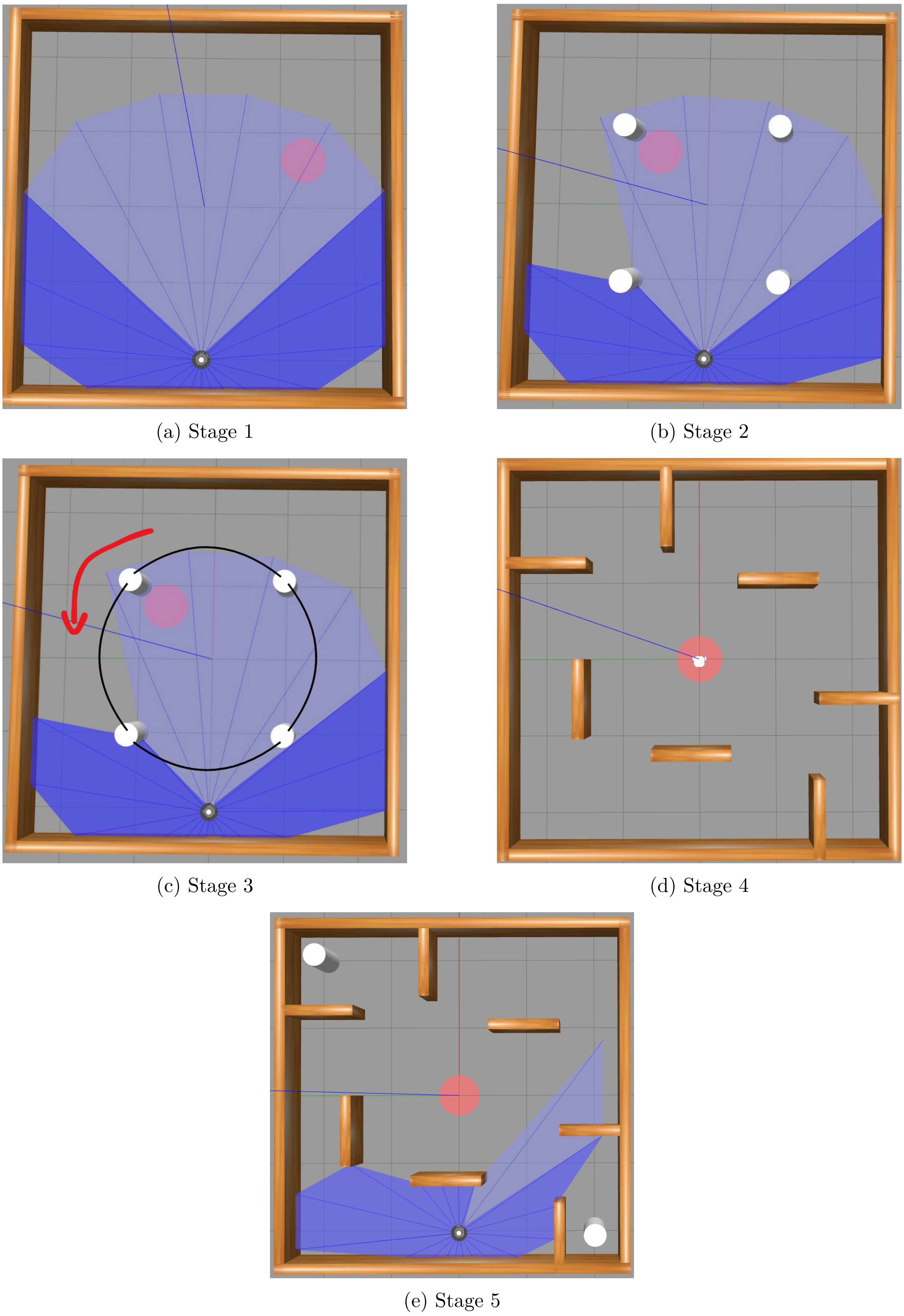


Figure 3.2: Illustration of the five training environments used in the study. The red disk denotes the random target location.

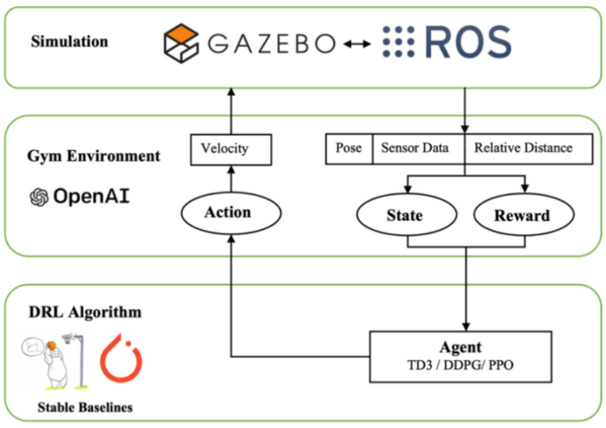


Figure 3.3: System architecture for training the mobile robot

3.3.1.1 OpenAi GYM and Stable-Baselines

The OpenAI Gym python3 framework offers standardized APIs that facilitate interaction between reinforcement learning environments and algorithms. In our work, we utilize the Stable-Baselines3 (SB3) library alongside OpenAI Gym to train a mobile robot using deep reinforcement learning (DRL) techniques. SB3 is an open-source library that delivers robust, PyTorch-based implementations of advanced DRL algorithms.

3.4 Learning experience and agent setup

The main goal of the mapless navigation robot is to navigate to a given goal while avoiding obstacles and taking an optimal path. To achieve this goal, the robot must take the appropriate actions give a state.

3.4.1 State and action spaces

The agent receives observation data from sensor (LiDAR in our case), relative distance and orientation to goal and robot's last timestep velocity as input.

The goal then is to find a function to map raw input data to velocity command as follows :

$$v_t = f(d_t, g_t, v_{t-1})$$

Where d_t is a 20-dimensional information vector from the Li-DAR, and covers a 360 degree field of view unlike common works that limit the field of view only to the robot's forward direction only, thereby missing obstacles approaching from behind. Value from Li-DAR are normalized between 0 and 1 before passing the to the agent.

The vector g_t represents the two-dimensional normalized relative position of the goal, where the distance to the target is scaled by the radius of the map, and the angular component included in $[-\pi, \pi]$ is normalized to the range $[-1, +1]$, this information helps the robot to move closer to the target [20]

The term v_{t-1} refers to the robot's velocity at the previous time step. It represents the last action taken and helps the robot decide the next move by considering its current motion and inertia [26].

As for the current action, the agent determines the appropriate control command v_t at each time step based on the current state, with the objective of reaching the target efficiently. The action space can be defined as either discrete or continuous, specifying the allowable range for each control parameter.
In this project, we adopt a continuous action space, where the agent outputs real-valued commands for both linear and angular velocity. Specifically, the action space is defined as follows:

$$\text{Linear velocity} \in [0.0, 0.45]\ \text{m/s}, \quad \text{Angular velocity} \in [-2.0, 2.0]\ \text{rad/s}$$

This configuration allows the agent to select any value within these ranges at each time step, enabling smoother and more precise motion control compared to discrete action settings.

3.5 Experiments and reward function design summary

In this section, we describe the training experiments and the design of the reward function, which is tailored to each training stage. The aim is to enable the agent to learn the navigation task progressively, starting from simple environments and advancing toward more complex scenarios.

Experiment 1 For this initial experiment, we evaluated several state-of-the-art Deep Reinforcement Learning (DRL) algorithms, including PPO, DQN, SAC, TD3, and DDPG. These algorithms were selected based on their adoption and demonstrated success in the DRL literature. Each model was trained using the observation space described previously, which includes laser scan data, the relative position of the goal, and the robot's previous velocity.

Training was conducted in *Stage 1*, a simplified environment containing no dynamic or static obstacles, aside from the fixed indoor walls that define the environment's boundaries. The purpose of this setup was to assess the basic learning capabilities of each algorithm in a low-complexity setting and to establish a baseline for performance comparison in later, more challenging stages in order to chose a single algorithm to train in very complex environment.

The reward function used in this stage was designed to encourage progress toward the goal and penalize undesirable outcomes. It is defined as follows:

$$r = \begin{cases} +100, & \text{if the goal is reached } (r_g) \\ -100, & \text{if a collision occurs } (r_c) \\ -100, & \text{if the episode times out } (r_t) \end{cases}$$

$$\text{Otherwise,} \quad r = 10 \cdot (d_{t-1} - d_t) - \frac{|\omega_t|}{2}$$

Where d_t is the current distance to the goal, d_{t-1} is the previous distance to the goal, and ω_t is the angular velocity at time step t. The term $10 \cdot (d_{t-1} - d_t)$ rewards the agent for moving closer to the goal and penalizes it if the distance increases. The second term, $\frac{|\omega_t|}{2}$, promotes smooth rotational movements by discouraging excessive turning, while the overall reward formulation still applies strong penalties for collisions and timeouts.

Experiment 2 In this stage, a set of static cylindrical obstacles was introduced into the environment, as defined in *Stage 2*. The same set of pretrained DRL algorithms from Experiment 1 were retrained in this new setting. The objective was to assess the agents' ability to extend their learned goal-reaching behavior by incorporating obstacle avoidance skills. While the agents were already proficient at navigating toward the goal, they had not yet learned to avoid obstacles.

The reward function from Stage 1 was retained but augmented with an additional penalty term to discourage the agent from approaching obstacles too closely. This term is defined as:

$$r_a = \begin{cases} -3 \cdot e^{-d_t}, & \text{if } \min(d_t) < 0.4 \\ 0, & \text{otherwise} \end{cases}$$

where $\min(d_t)$ represents the minimum distance to the nearest obstacle at time step t. This exponential penalty increases as the robot gets closer to an obstacle, thereby encouraging safer navigation paths and improved collision avoidance.

At the end of this stage, the performance of all algorithms was compared. The best-performing model—based on success rate, collision rate, and learning stability—was selected to proceed to the more advanced training stages.

Experiment 3 In this stage, the environment complexity was further increased by introducing dynamic obstacles. The same cylindrical objects from Stage 2 were now programmed to move along predefined circular trajectories, simulating real-world scenarios such as pedestrians or moving machinery. This posed a greater challenge, as the agent not only had to reach the goal and avoid static obstacles but also anticipate and react to moving ones.

The reward structure from Stage 2 was maintained, with a modification to the obstacle-avoidance penalty to reflect the increased risk posed by dynamic obstacles. Specifically, the avoidance penalty term was scaled as follows:

$$r'_a = 2 \cdot r_a = \begin{cases} -6 \cdot e^{-d_t}, & \text{if } \min(d_t) < 0.4 \\ 0, & \text{otherwise} \end{cases}$$

By doubling the penalty when approaching moving obstacles, the agent is strongly discouraged from unsafe proximity to dynamic entities, promoting anticipatory and cautious behavior.

Following the evaluation of all algorithms, the best-performing one, selected based on its success rate, and safety in statical environment, was chosen to continue from this experiment.

Experiment 4 In this experiment, exploring stage 4, we shift to an environment with complex static structures. Multiple wall segments are placed randomly varying in length and orientation, creating irregular navigation routes.
The goal is to train the previous model from experiment 2 so the robot can overcome obstacles that require more than simple sidestepping, letting the agent learn how to avoid large continuous barriers.

Experiment 5 In this experiment, we test how well the trained model can work in a completely new environment that it has never seen before. We take the best model from the previous experiments and run it in a different map with a new layout and different obstacle placements.

The goal is to check if the model can still reach the target and avoid obstacles without being trained again. To measure this, we compare its performance with a classical navigation method that also does not have any prior knowledge of the map.

We compare both methods based on their success rate, number of collisions, and how efficiently they reach the goal. This helps us understand whether the trained DRL model can generalize to new environments.

Experiment 6 In stage 5, we build on the model trained in Stage 4 by introducing dynamic elements into the environment. Specifically, cylindrical obstacles are reintroduced and programmed to move along random trajectories. These moving obstacles are placed among the static walls already present in the environment.

The goal of this experiment is to train the agent to navigate in mixed environments that contain both static and dynamic obstacles. The agent must now not only plan around fixed structures like walls but also react to the unpredictable movement of dynamical obstacles.

Training continues using the same reward function as in previous stage, which penalizes proximity to obstacles and rewards progress toward the goal. This stage serves to further enhance the agent's ability to operate in realistic scenarios that resemble complex, populated indoor environments.

3.6 Training and results

This section presents both training dynamics and performance evaluation of the DRL-based navigation models, as defined in the experimental stages of the previous section. Each experiment was designed to evaluate specific aspects of the agent's learning capabilities, including goal reaching, obstacle avoidance, and policy generalization.

The training process was conducted using the Stable-Baselines3 implementations of selected DRL algorithms. All training was performed in simulation using the Gazebo–ROS 2 framework, ensuring realistic physics and reliable system integration.

The first part focuses on training metrics such as actor loss, critic loss, average episode reward, and convergence behavior over time. These metrics provide insights into the learning stability and efficiency of the selected DRL algorithms.

The second part presents the evaluation results for each experimental stage.
The following evaluation metrics were used to assess the performance of the trained models:

- **Success rate**: Percentage of episodes in which the agent successfully reaches the goal without any collision.
- **Collision rate**: Percentage of episodes that end due to collision with an obstacle or wall.
- **Average reward**: Mean cumulative reward obtained per episode, reflecting overall learning efficiency.

3.6.1 Summary of Algorithms hyper-parameters

Before presenting the training dynamics and results, the hyperparameters selected for each algorithm are summarized in the following tables.

Hyperparameter	**Value**
Learning Rate	1×10^{-4}
Buffer Size	100,000
Learning Starts	1,000
Batch Size	64
Gamma (γ)[1]	0.99
Train Frequency	1 step
Gradient Steps	1
Target Update Interval	1,000
Action Noise	–
Network Architecture	[256, 256]
Activation Function	ReLU

Table 3.1: DQN Hyperparameters

Hyperparameter	**Value**
Learning Rate	1×10^{-4}
Buffer Size	1,000,000
Learning Starts	1,000
Batch Size	64
Gamma (γ)	0.99
Tau (τ)[2]	0.001
Train Frequency	1 step
Gradient Steps	1
Action Noise	Yes
Network Architecture	[400, 300]
Activation Function	ReLU

Table 3.2: DDPG Hyperparameters

Hyperparameter	**Value**
Learning Rate	3×10^{-4}
Buffer Size	1,000,000
Learning Starts	10,000
Batch Size	256
Gamma (γ)	0.99
Tau (τ)	0.005
Train Frequency	1 step
Gradient Steps	1
Target Update Interval	1
Entropy Coefficient start	1
Network Architecture	[256, 256]
Activation Function	ReLU

Table 3.3: SAC Hyperparameters

Hyperparameter	**Value**
Learning Rate	1×10^{-4}
Buffer Size	500,000
Learning Starts	10,000
Batch Size	64
Gamma (γ)	0.99
Tau (τ)	0.005
Train Frequency	1 step
Gradient Steps	1
Policy Delay	2
Action Noise	Yes
Network Architecture	[400, 300]
Activation Function	ReLU

Table 3.4: TD3 Hyperparameters

[1]The discount factor γ determines how much future rewards are valued compared to immediate rewards.

[2]The soft update parameter τ controls the rate at which the target network is updated towards the main network.

3.6.2 Experiment 1 training dynamics and results

Training dynamics

As stated earlier, DQN, DDPG, SAC, and TD3 algorithms are trained in Experiment 1 and evaluated to determine the most suitable candidate for adaptation to more complex environments.

All training dynamics—including reward, overall mean reward, actor and critic loss for actor–critic methods, and loss for the DQN algorithm—are plotted and discussed.

DQN

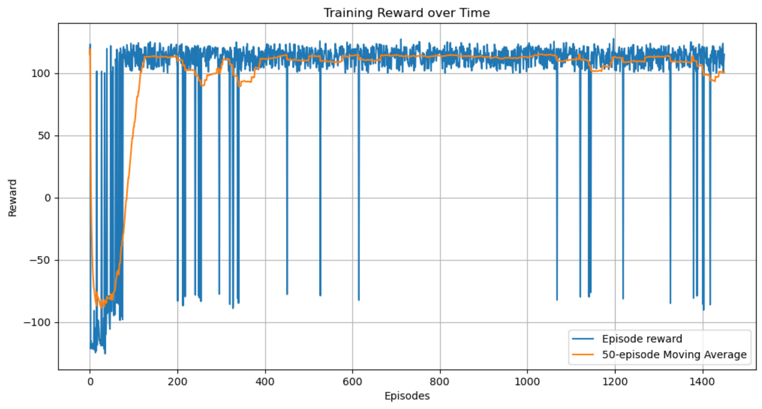


Figure 3.4: DQN Stage 1 – Episode Reward and moving average over last 50 episodes

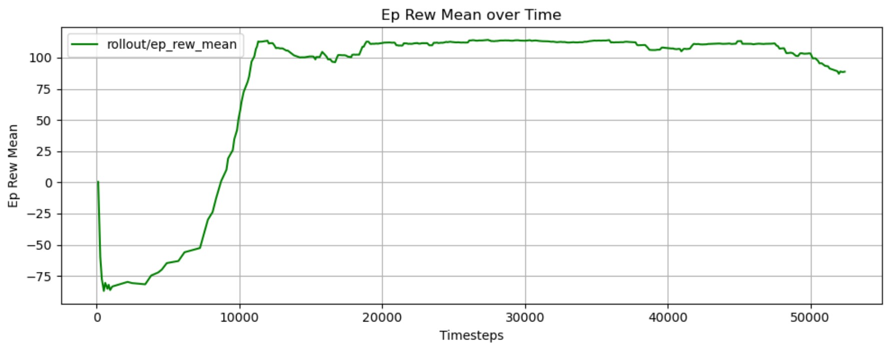


Figure 3.5: DQN Stage 1 – Mean Reward overtime

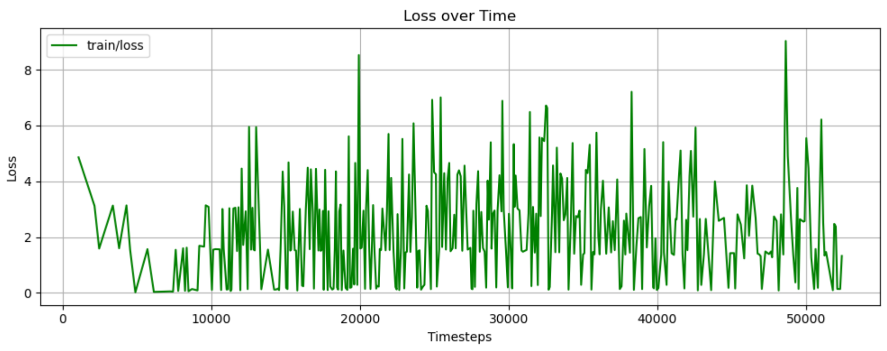


Figure 3.6: DQN Stage 1 – Training Loss

- The episodic reward shown in Figure 3.5 indicates unstable learning, with consistently high negative values across episodes.
- A smoothing trend begins to appear after approximately 100 episodes, but the training remains unstable overall.
- The loss curve in Figure 3.6 is highly irregular, which is a known issue associated with the DQN algorithm.
- This instability is attributed to poor learning performance, primarily caused by an inadequate balance between exploration and exploitation.

DDPG

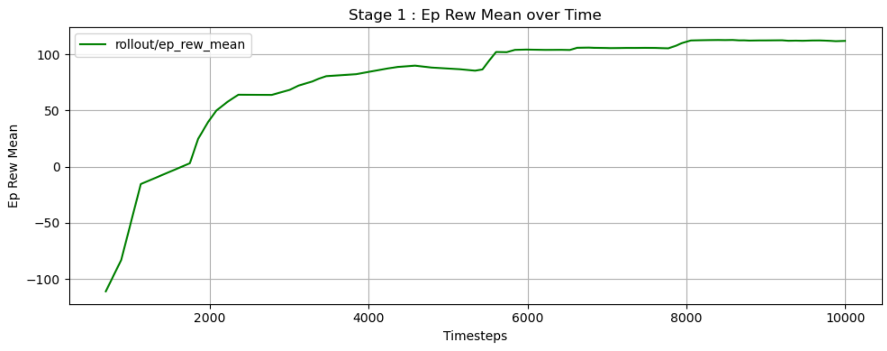


Figure 3.7: DDPG Stage 1 – Mean Reward

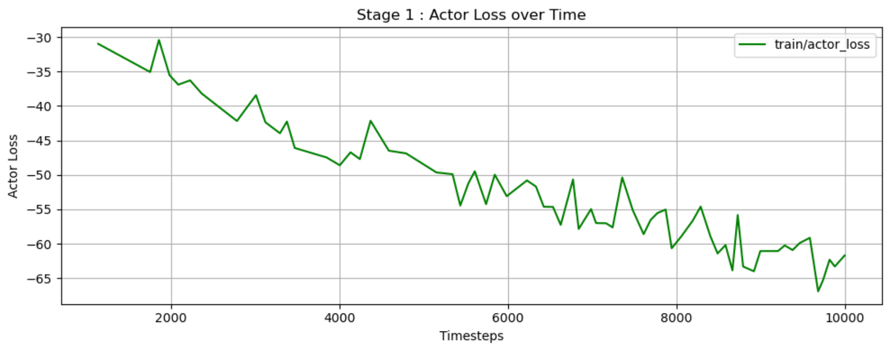


Figure 3.8: DDPG Stage 1 – Actor Loss

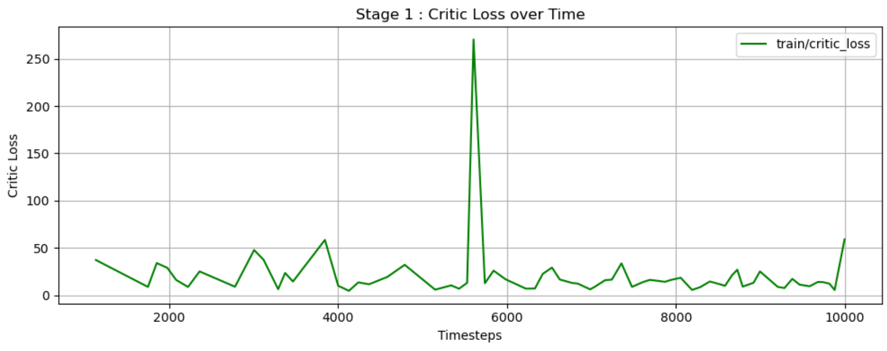


Figure 3.9: DDPG Stage 1 – Critic Loss

- Figure 3.7 shows a clear and steady improvement in the mean reward, indicating progressive convergence.
- Actor loss in figure 3.8decreases smoothly and tend to stabilize. Critic loss is showing stability in figure 3.9 reflecting stable learning.

SAC

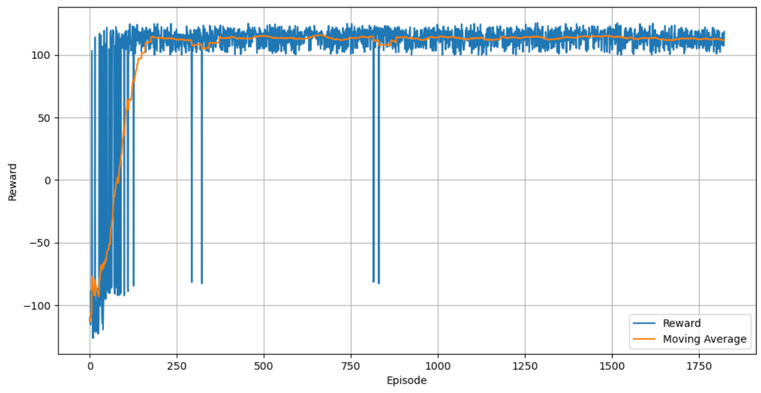


Figure 3.10: SAC Stage 1 – Episode Reward per Episode

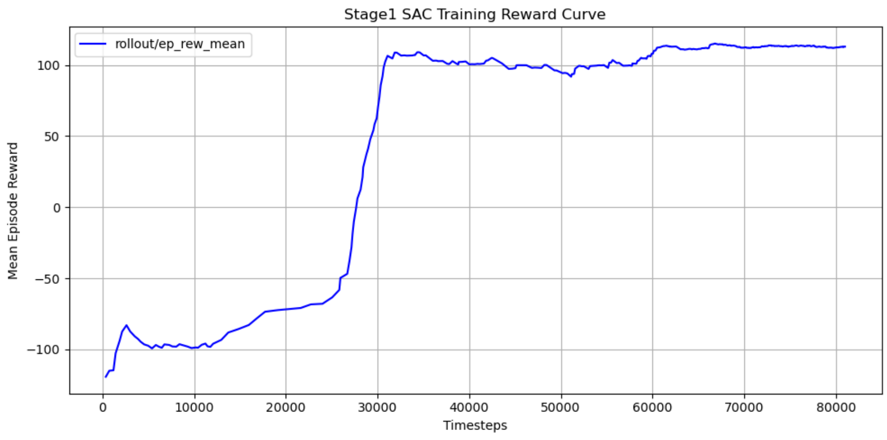


Figure 3.11: SAC Stage 1 – Mean Reward (Smoothed)

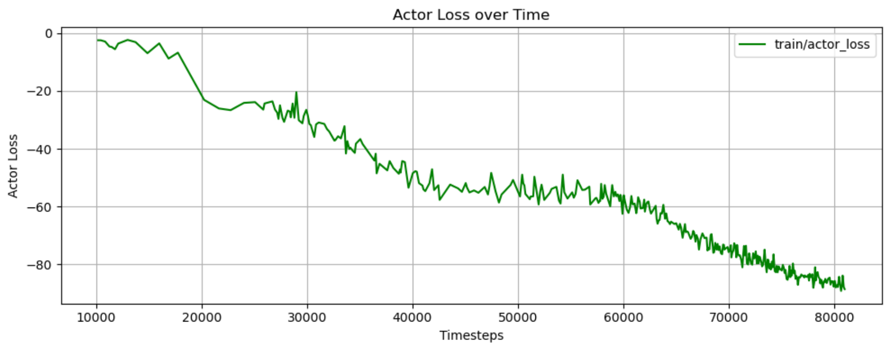


Figure 3.12: SAC Stage 1 – Actor Loss

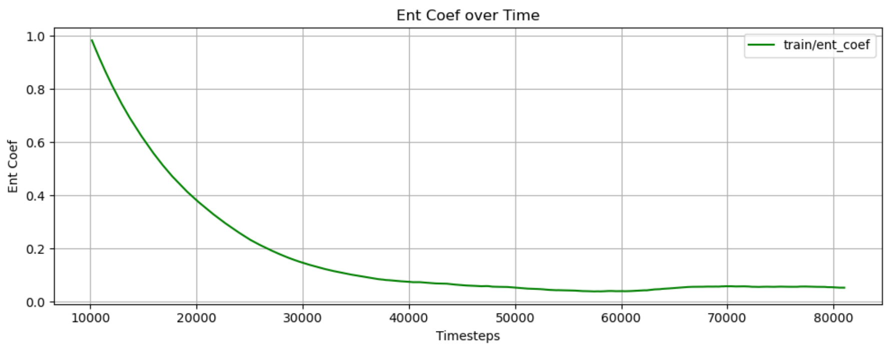


Figure 3.13: SAC Stage 1 – entropy coefficient dynamics

- Figures 3.10 and 3.11 show a smooth and consistent reward increase, indicating effective learning.
- The entropy term in SAC ensures a balanced exploration-exploitation trade-off, leading to stable policy improvement.
- The actor loss remains stable post-convergence, with no major spikes, suggesting robust learning due to entropy regularization [7].
- Figure 3.13 shows a smooth entropy coefficient decay from 1.0 to 0.05, marking a proper transition from exploration to exploitation.

TD3

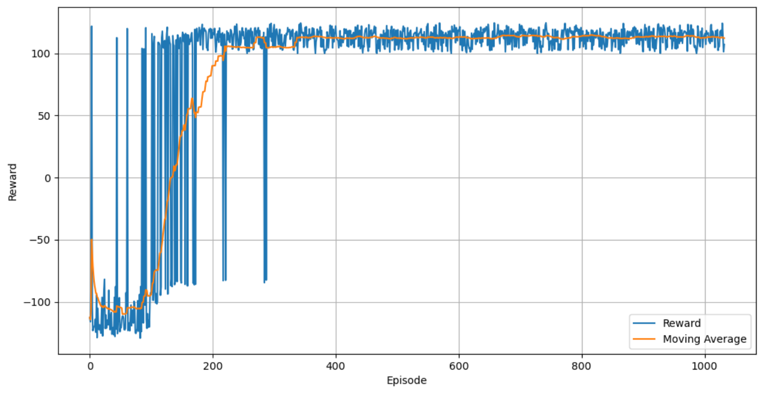


Figure 3.14: TD3 Stage 1 – Episode Reward per Episode

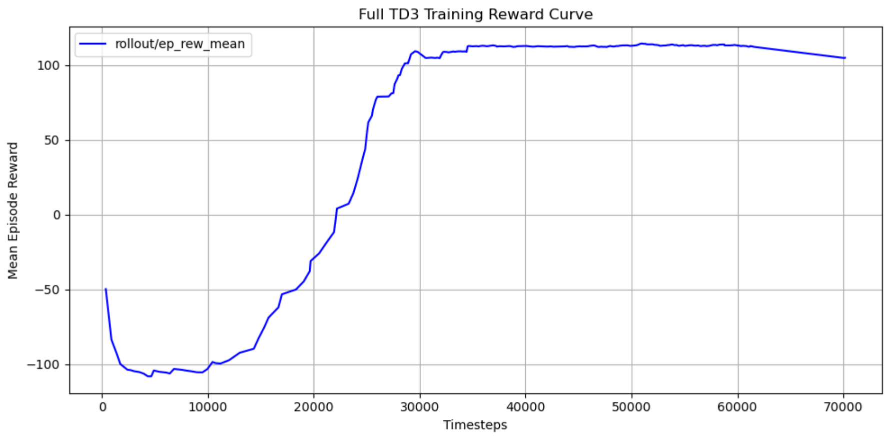


Figure 3.15: TD3 Stage 1 – Mean Reward (Smoothed)

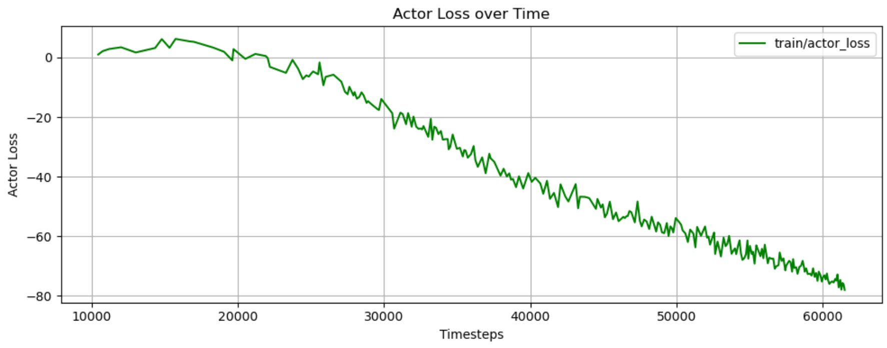


Figure 3.16: TD3 Stage 1 – Actor Loss

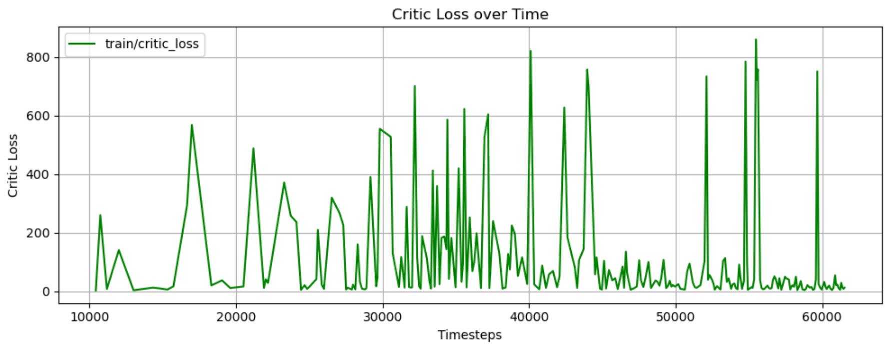


Figure 3.17: TD3 Stage 1 – Critic Loss

- TD3 shows the highest reward among all algorithms, with a consistent progression toward convergence.
- The stability mechanisms of TD3 are reflected in the smooth actor and critic loss curves.
- The critic loss stabilizes after around 40,000 timesteps, with only minor noise-related spikes that do not affect overall training.

Results

Each trained model is tested over 30 goal-reaching trials. During these tests, the success rate and the collision rate are computed to evaluate the agent's performance.

The summary of the testing experiment is presented in table 3.5

Algorithm	Success Rate	Collision Rate	Mean Reward
TD3	100%	0%	98.68
SAC	96.97%	3.03%	84.016
DDPG	100%	0%	82.35
DQN	76.67%	23.33%	60.47

Table 3.5: Comparison of DRL algorithms on success rate, collision rate, and mean reward in experiment 1.

Conclusion

As a conclusion of this first experiment, DQN demonstrates limited performance and unstable learning behavior, as reflected in its higher collision rate and lower average reward compared to the other algorithms. This instability makes it less suitable for more complex navigation tasks.

In contrast, DDPG, SAC, and TD3 show significantly better performance in terms of reward progression, learning stability, and success rate. Their ability to learn efficient policies and adapt to the environment makes them strong candidates for future experiments involving more complex scenarios and generalization tasks. These algorithms will therefore be retained for the next stages of evaluation.

3.6.3 Experiment 2 training dynamics and results

In this experiment, DDPG, SAC, and TD3 from experiment 1 that learned how to reach a gaol are retrained in stage 2 in presence of obstacles. The task now it to reach a goal while avoiding obstacle in an optimal way.

The following section will illustrate both training dynamics and experiment results so we can chose the best algorithm for generalization.

Training dynamics

DDPG

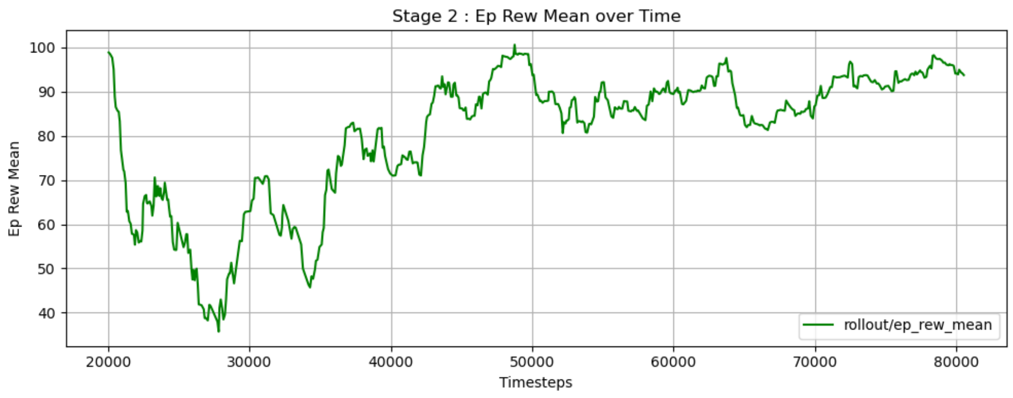


Figure 3.18: DDPG Stage 2 - Mean Reward

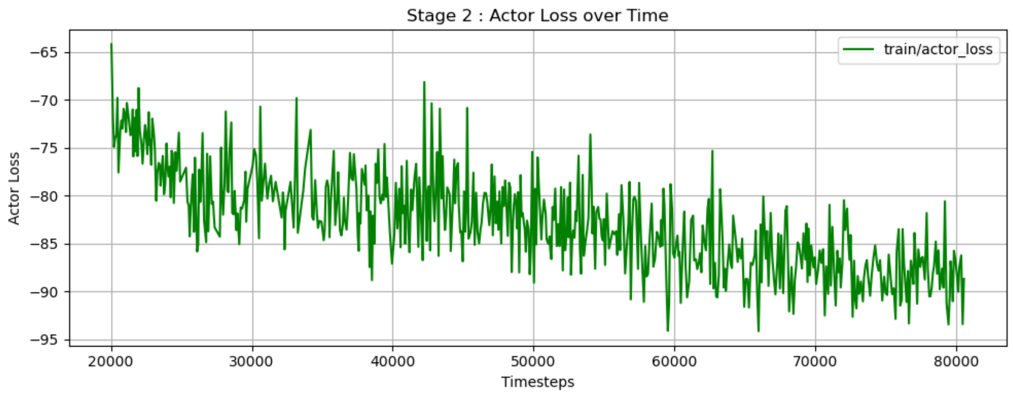


Figure 3.19: DDPG Stage 2 - Actor Loss

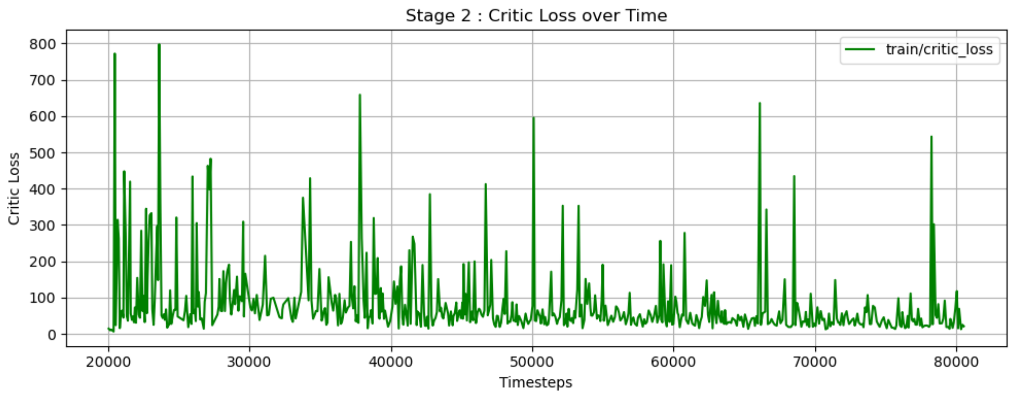


Figure 3.20: DDPG Stage 2 - Critic Loss

- The mean reward in figure 3.18 increases toward a low level reward with some oscillations, indicating limited learning.

- Actor and critic losses shows some stability after a certain number of training time steps. The reward fluctuation in addition to stability suggests that the model converged to a sub optimal policy and is no longer improving its policy.

SAC

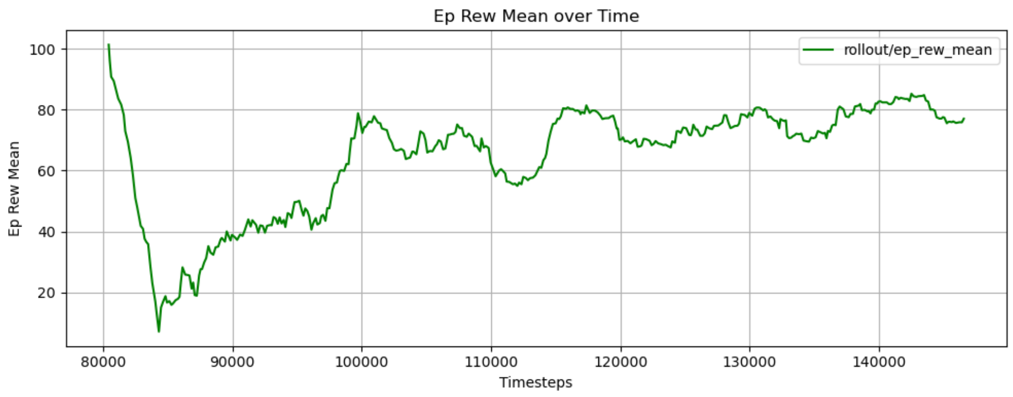


Figure 3.21: SAC Stage 2 - Mean Reward

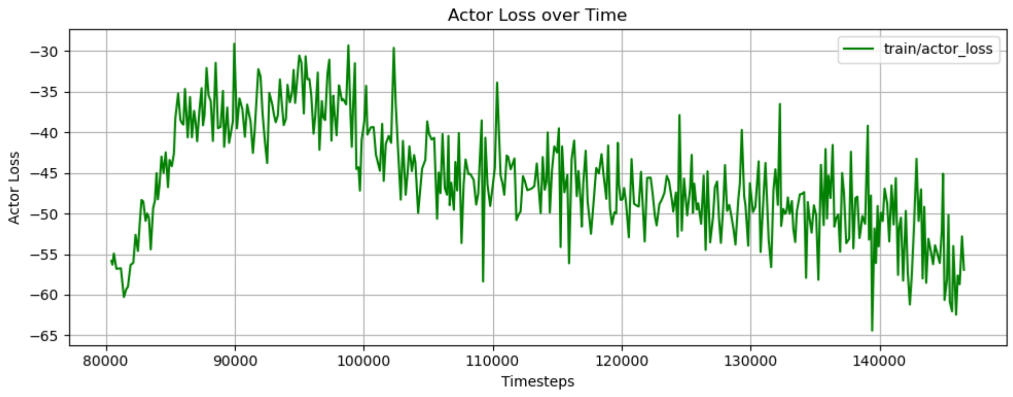


Figure 3.22: SAC Stage 2 - Actor Loss

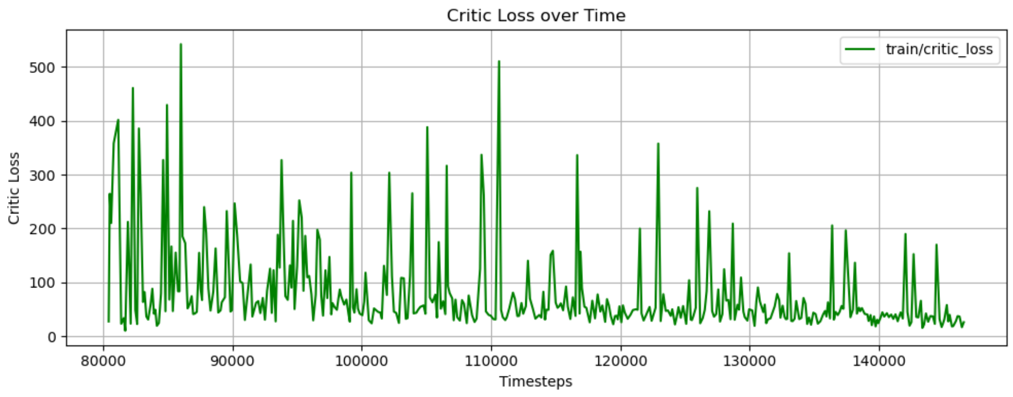


Figure 3.23: SAC Stage 2 - Critic Loss

- The mean reward in figure 3.21 increases well but with no smoothness and converges to a low reward.
- Actor and critic losses in figures 3.22 and 3.23 are relatively stable, though the critic loss shows small irregularities due to SAC's stochastic behavior.

TD3

- The mean reward in figure 3.24 shows a steady and stable increas, indication smooth learning and efficient policy improvement. The few fluctuations and due to the closeness of sone of the targets to obstacles so the agent will collect some negative reward for approaching the obstacle but prioritize reaching the goal.

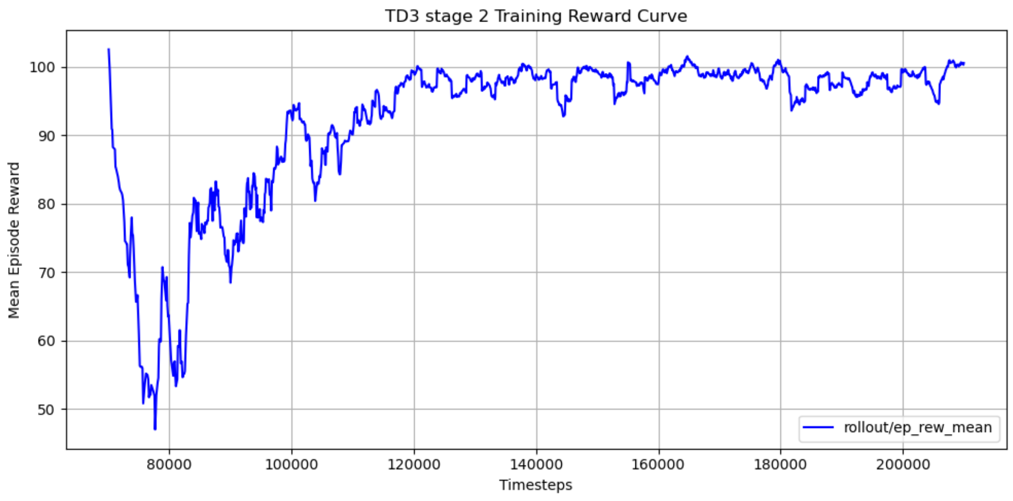


Figure 3.24: TD3 Stage 2 - Mean Reward

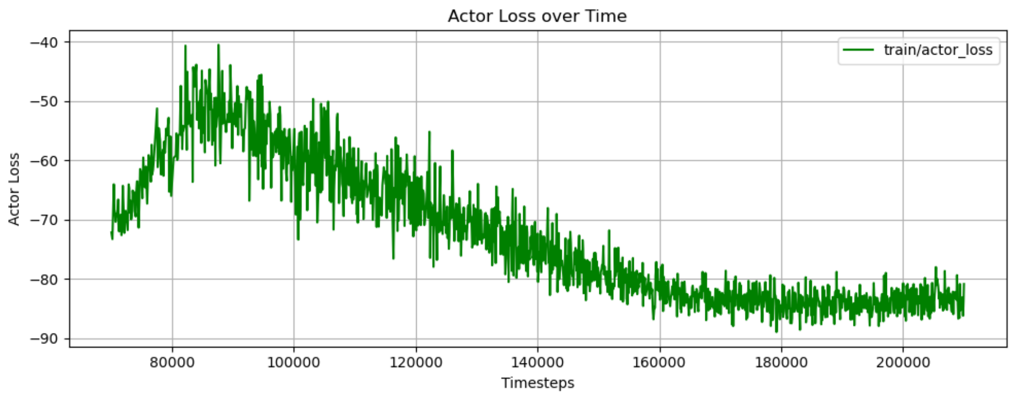


Figure 3.25: TD3 Stage 2 - Actor Loss

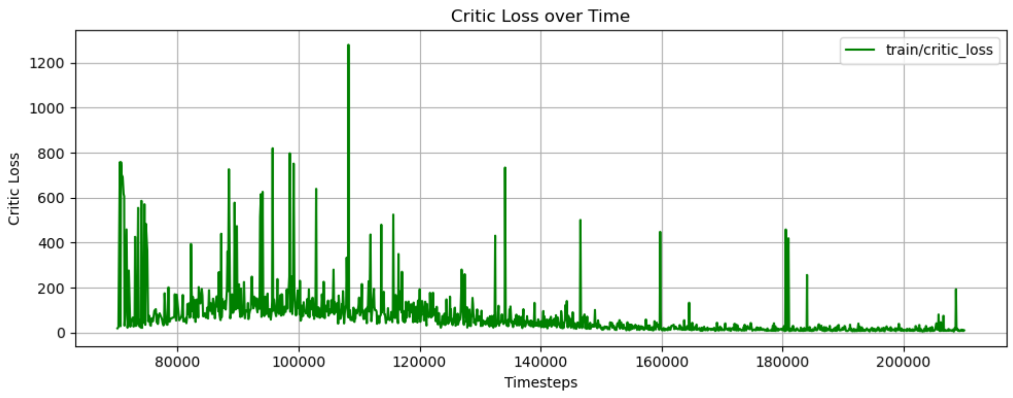


Figure 3.26: TD3 Stage 2 - Critic Loss

- The actor loss in figure 3.25 remains low and stable, a sign of effective policy updates.
- The critic loss in figure 3.26 fluctuates slightly but remains within a reasonable range, showing robustness against noise. After 40k steps, it tends to stabilize with only minor spikes due to noises.

Algorithm	**Success Rate**	**Collision Rate**	**Mean Reward**
SAC	96.67%	3.33%	85.41
TD3	**100%**	**0%**	**99.19**
DDPG	70.00%	30.00%	34.25

Table 3.6: Comparison of SAC, TD3, and DDPG algorithms in experiment 2

Results

Conclusion

Among the three algorithms, DDPG shows the weakest performance, with a low success rate, and poor mean reward compared to TD3. DDPG's training curves, including reward and loss functions, shows that the agent converged to a sub-optimal policy. In addition to that, it showed a low performance in the testing phase that make him out of the discussion for later experiments.

SAC performs better than DDPG, achieving higher mean reward and a very strong success rate, although its training dynamics shows some fluctuations, its stochastic nature made him a promising candidate for the next experiments.

TD3 is ultimately selected for the next stages, as it offers both high performance and the most stable learning behavior, with smooth reward progression and stable actor and critic losses.

3.6.4 Experiment 3 training dynamics and results

As stated before, in this experiment, the agent will be trained on stage 3 environment where the obstacles are now dynamic.

Training dynamics

The TD3 pretrained agent from experiment 2 is now trained for additional 90k time steps to enhace its capability to devlop a navigation strategy in presence of dynamic obstacles.

Training curves are shown bellow.

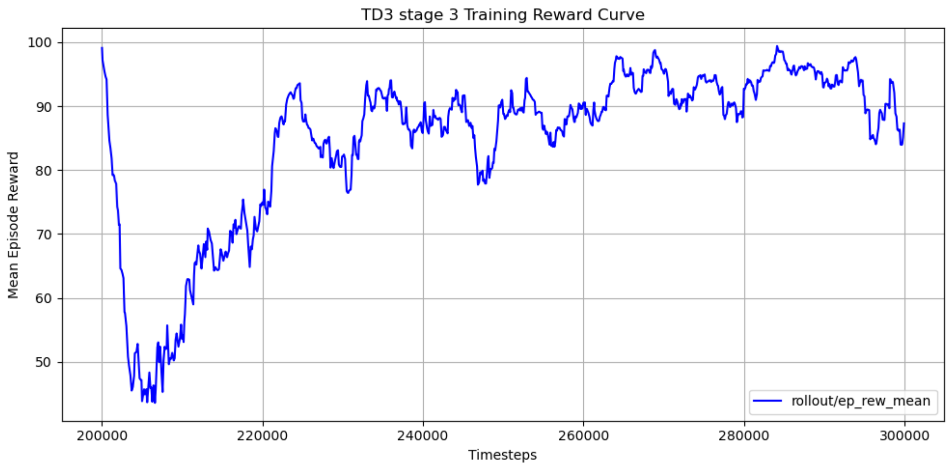


Figure 3.27: TD3 Mean reward in stage3

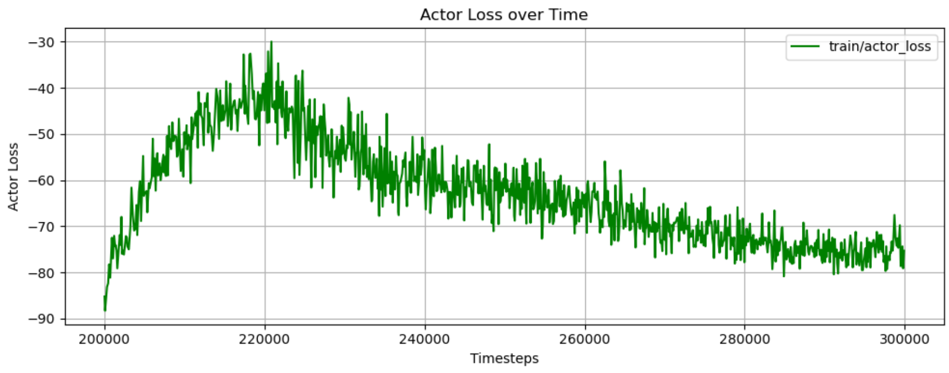


Figure 3.28: TD3 Actor loss in stage3

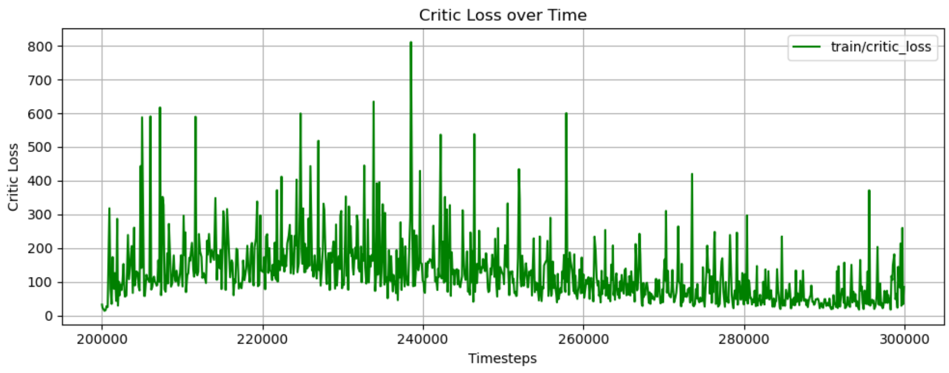


Figure 3.29: TD3 Critic loss in stage 3

In stage 3, the TD3 agent continues to show consistent and stable training behavior. As shown in figure 3.27, the mean reward increases progressively, with minimal fluctuation, due to unexpected positions of the obstacles and the randomness of the spawning position of the robot. The reward shows a steady improvement of the learned policy.
The actor loss curve in figure 3.28 remains smooth an relatively flath with minor variations, this suggests that the policy updates are well controlled and do not suffer from instability. In figure 3.29, the critic loss is initially high but stabilizes after around 40k time steps with some visible fluctuations due to noises. However these fluctuations are not disruptive and do not impact the training performance as confirmed in the experiment results.

Results

In this experiment, 30 goal positions were randomly generated in an environment containing dynamic obstacles. The agent successfully reached all target positions, achieving a success rate of 100% with no collisions. The average reward obtained across all episodes was 100.49, demonstrating the robustness and effectiveness of the proposed navigation strategy.

Conclusion

To conclude this stage, TD3 confirms its status as the most reliable and stable algorithm among those tested. It consistently achieves a high average reward, a high success rate, and demonstrates effective adaptation to more complex scenarios introduced in this phase. These results reinforce the selection of TD3 as a reference algorithm for tackling more advanced robotic navigation tasks in future experiments.

3.6.5 Experiment 4 training dynamics and results

Training dynamics

In this fourth experiment, the reward function was modified to encourage the robot to overcome walls. Specifically, the agent now receives a double penalty for collisions and a triple penalty for proximity to obstacles, compared to experiment 2. These adjustments are reflected in the little observed fluctuations of the training curve, as shown in figure 3.30, as the robot receives more negative reward but still achieves the overall goal.

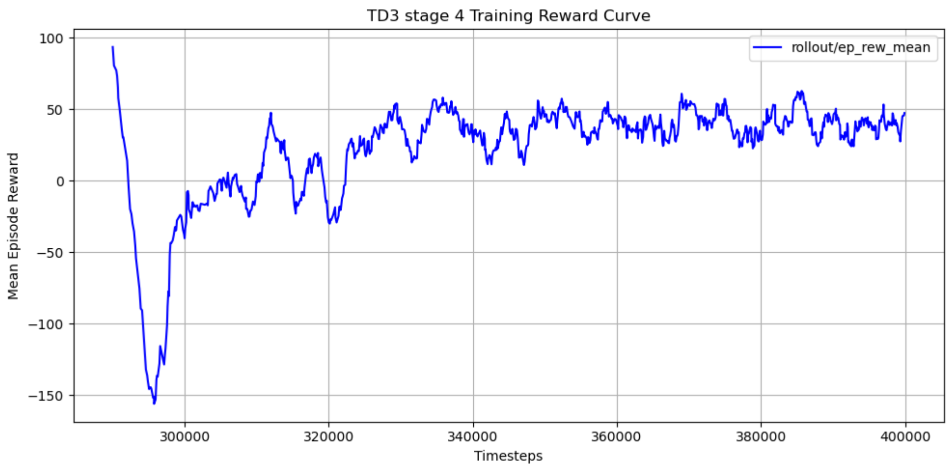


Figure 3.30: Experiment 4 mean reward

Despite the increased penalties, the agent successfully learned a stable policy. The mean reward tends to coverage to a high value, indicating the effective learning. Furthermore, the actor loss remains stable without significant spikes, as illustrated in Figure 3.31, which confirms the policy's stability during training.

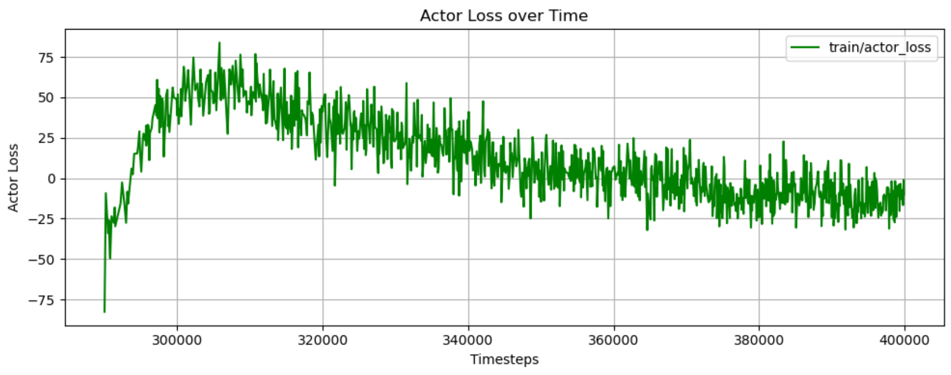


Figure 3.31: Experiment 4 actor loss

Results

As in the previous experiments, the TD3 agent continued to yield promising results. In this test, 30 goal positions were randomly generated behind and between walls to evaluate the agent's ability to navigate challenging environments. The agent successfully reached all targets, achieving a 100% success rate once again.

Conclusion

This experiment demonstrated the robustness and adaptability of the TD3 agent in more constrained and penalized environments. Despite the harsher reward conditions designed for safer navigation near walls, the agent maintained a strong learning capability. This experiment confirms once more the effectiveness of the modified reward function for curriculum learning strategy.

3.6.6 Experiment 5 results and conclusion

Generalization Test in a New Environment

In this fifth experiment, we evaluate the generalization capability of the trained TD3 agent from Experiment 4 in a completely new environment that it has never encountered during training. The goal is to assess whether the learned policy can achieve safe and efficient navigation in unseen scenarios.

To visualize the test setup, the new environment is shown in Figure 3.32. To support our analysis, we also compare the agent's performance with a classical navigation method based on the ROS 2 Nav2 stack, which uses the Dijkstra algorithm for global planning and the Dynamic Window Approach (DWA) for local planning. The classical method is provided with the same static map used in Experiment 4 to ensure fair comparison conditions.

In the classical approach, a map of the environment is first constructed using SLAM techniques. This map, along with the target goal pose, is then provided to the navigation algorithm, which computes a global path and continuously adjusts it during execution based on detected obstacles.

To assess which method, DRL or classical navigation, performs better under unknown conditions, we compare their performance based on the total distance covered and the time required to reach the goal. These metrics are particularly relevant in scenarios where speed and energy efficiency are critical, such as in emergency rescue missions or hospital environments.

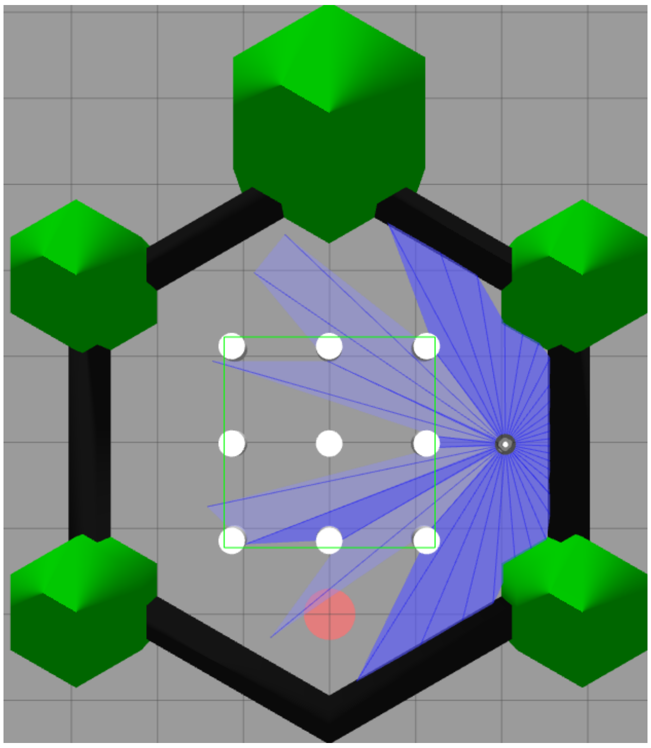


Figure 3.32: Unknown environment for the agent

Results

For the first part of this evaluation, the TD3 agent was tested on 30 randomly generated goal positions within the new environment. All target positions were selected such that the path to reach them require navigating around obstacles. Impressively, the agent successfully reached all of 30 goals, demonstrating strong generalization capability and robust obstacle avoidance, even in an environment it had never encountered before.

For the second part, the robot with classical navigation approach also demonstrated a strong robust navigation within all goals generated.

The table below summarizes the average distance traveled and average time required to reach the goal over 9 trials for each method. The exact same 9 goals are generated in both approach.

Method	**Average Distance (m)**	**Average Time (s)**
DRL Agent Appraoch	**3.47**	**11.48**
Classical Approach	3.52	24.38

Table 3.7: Average distance and time to reach the goal over 9 trials

The summary of this experiment is found in the following table

Trial	**DRL Agent**		**Classical Approach**	
	Distance (m)	Time (s)	Distance (m)	Time (s)
1	2.91	9.53	3.13	19.81
2	2.84	8.26	3.49	28.87
3	3.15	10.49	3.15	22.34
4	3.16	9.66	3.21	22.72
5	3.25	10.11	3.07	19.52
6	2.97	8.80	3.21	20.76
7	4.62	13.98	4.45	28.12
8	4.82	15.06	4.29	29.84
9	4.45	13.40	4.46	29.40
Mean	**3.47**	**11.48**	**3.52**	**24.38**

Table 3.8: Performance comparison of DRL and Classical navigation over 9 trials

Conclusion

The evaluation demonstrated that both the DRL agent and the classical navigation approach were capable of successfully reaching al target goals in an unfamiliar environment, indicating robust obstacle avoidance and reliable navigation. While both methods showed comparable performance in terms of distance optimization, the DRL agent significantly outperformed the classical approach in terms of time efficiency.

This difference is attributed to the mechanisms of each method. The classical navigation stack relies on constructing a global path using the Dijkstra algorithm and then continuously adjusting it using the local planner (DWA) as new obstacles are detected. This process involves constant re-evaluation and can introduce computational delays.

In conclusion, the DRL agent, trained to maximize reward based on reaching the goal quickly and safely, reacts more directly towards the goal. It doesn't not require a prior map or global planning and instead makes real-time decisions that balance goal reaching and collision avoidance. As a result, it achieves faster navigation, making it more suitable for time-critical applications such as emergency response or hospital devilry tasks.

3.6.7 Experiment 6 Training dynamics and results

Training dynamics

Results

The agent trained in Experiment 6, which involved both static walls and dynamic cylinders in the indoor environment, demonstrated its ability to navigate in the presence of both static and dynamic obstacles. However, it occasionally struggled with dynamic obstacles, succeeding in 22 out of 30 trials and failing in the remaining 8.

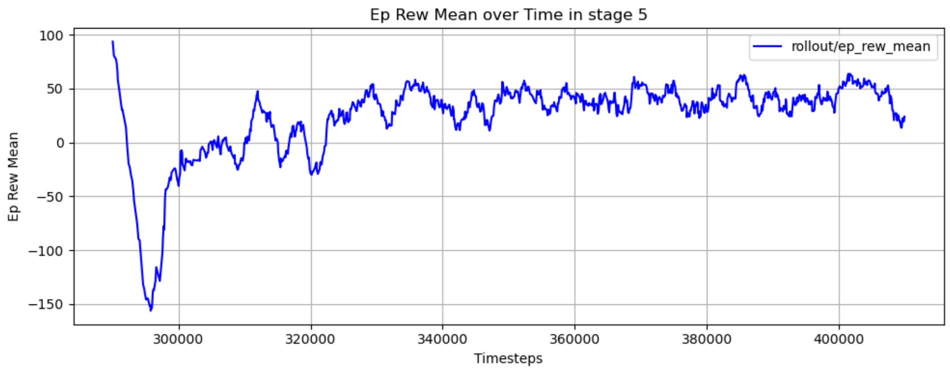


Figure 3.33: Experiment 6 mean reward

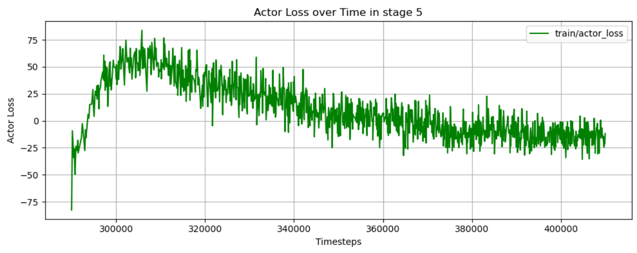


Figure 3.34: Experiment 6 actor loss

Conclusion

Experiment 6 demonstrated stable learning, as evidenced by the reward progression and the stability of the actor loss shown in Figures 3.33 and 3.34. The agent exhibited effective navigation capabilities; however, further refinement is needed to handle situations where dynamic obstacles approach rapidly. This could potentially be addressed by designing a more suitable reward function.

3.6.8 Training time

The training was performed on a machine equipped with an Intel Core i9 14th generation processor and an NVIDIA RTX 3060 GPU to accelerate computation.

The table below summarizes the training duration for each stage and algorithm.

Algorithm	**Stage**	**Training Time**
DQN	Stage1	4h 57min
DDPG	Stage 1	57min
	Stage 2	4h 58min
SAC	Stage 1	5h 28min
	Stage 2	4h 16min
TD3	Stage 1	3h 55min
	Stage 2	10h 31min
	Stage 3	7h 34min
	Stage 4	7h 41min
	Stage 5	5h 01min

Table 3.9: Summary of Training Time per Algorithm and Stage

3.7 Conclusion

This chapter presented a series of six experiments designed to evaluate and compare, then choose the best-performing algorithm in mapless navigation from a set of DRL algorithms — namely, DQN, DDPG, SAC, and TD3 — to continue for later increasingly complex navigation tasks.

The results highlighted the limitations of **DQN**, which showed unstable learning and poor navigation performance, making it unsuitable for complex tasks. **DDPG** showed moderate capabilities but failed in maintaining consistent success, particularly in the presence of obstacles. Contrary to **DDPG**, **SAC** showed better and more consistent success even though some fluctuations appeared in its losses due to its stochastic nature.

Among all the tested algorithms, **TD3** consistently outperformed the others, showing robust and stable learning behavior across all stages. Its smooth reward progression, high success rate, and adaptability to curriculum-based difficulty made it the most reliable candidate for real-world navigation challenges.

Comparing the DRL approach with a classical navigation method, where both achieved similar success rates, the DRL agent significantly outperformed the classical stack in terms of time efficiency, showing the benefits of learning-based decision making in real-time mapless navigation.

Overall, this chapter validates the suitability of DRL, and particularly the **TD3** agent, for autonomous navigation in environments with increasing complexity. It also shows the importance of reward shaping and curriculum design in achieving efficient and generalizable policies.

Demonstrating videos

To facilitate access to the demonstration videos, a set of QR codes has been generated. These QR codes can be scanned to view the corresponding demonstrations. They are provided below.

(a) Experiment 1

(b) Experiment 2

(c) Experiment 3

(d) Experiment 5

(e) Experiment 6

Figure 3.35: QR codes to demonstration videos.

Chapter 4

Multi Robot System

4.1 Introduction

In recent years, single-robot navigation systems have found widespread applications in real-world scenarios such as home cleaning robots, autonomous delivery, and self-driving cars. While effective in isolated use cases, these systems often face limitations in complex indoor environments, where the presence of multiple agents is increasingly common. Modern industrial and service settings frequently require coordinated efforts between multiple robots to accomplish tasks efficiently and safely.

Industry leaders like Amazon, FedEx, and Tesla have adopted multi-robot systems in their warehouses and manufacturing facilities to automate transport, cleaning, and pick-and-place operations, achieving significant gains in productivity and operational robustness.

In this work, we extend our initial research on single-robot mapless navigation by developing a multi-robot system using Deep Reinforcement Learning (DRL). Specifically, we consider a two-robot setup in which both agents must reach their respective goals without colliding with walls, obstacles, or each other. This setup reflects real-world challenges in shared, dynamic spaces and aims to promote safe and efficient autonomous cooperation.

4.2 Centrelized and Decentrelized Multi Robot System

When it comes to choosing between centralized and decentralized architectures, it usually depends on the specific task, the environment, how big the system is, and any limits on communication or processing power. Centralized systems have the benefit of a global view, which can lead to more optimal decisions since everything is handled in one place. This can make co-ordination and planning easier, but it also creates potential problems like bottlenecks, a single point of failure, and trouble scaling up especially in complex or changing environments.

On the other hand, decentralized systems spread out the decision-making across the robots themselves. This makes the system more scalable and resilient, since each robot can work independently using local data or limited shared info. That said, it can be tricky to get the group to act in a coordinated way, especially when local actions are supposed to lead to a global goal.

4.2.1 Centralized MRS

A centralized multi-robot system is characterized by a single control entity, typically referred to as a central coordinator or master controller, that maintains global knowledge of the system state and makes all critical decisions for the robot team [27]. In this architecture, individual robots act primarily as executors of commands issued by the central authority. The central controller collects sensor data from all robots, processes this information to maintain a comprehensive world model, plans coordinated actions, and distributes specific task assignments to each robot in the team.

This hierarchical structure ensures that all robots operate under a unified control strategy, with the central controller serving as the sole decision-making authority. Communication flows predominantly from the periphery to the center for data collection and from the center to the periphery for command distribution.

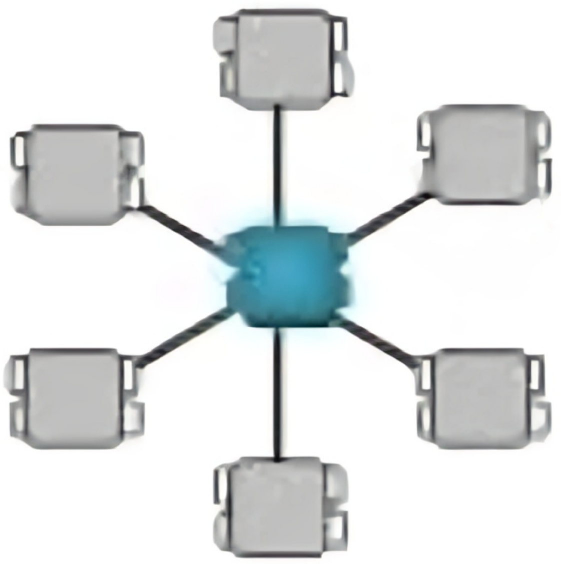


Figure 4.1: Centrelized MRS

4.2.2 Decentralized MRS

In contrast, decentralized multi-robot systems distribute decision-making authority across multiple robots, with each agent possessing some degree of autonomy and local intelligence. These systems rely on peer-to-peer communication and distributed consensus mechanisms to achieve coordination without a single point of control. Each robot maintains its own local perception of the environment, makes independent decisions based on local information and communication with neighboring robots, and contributes to the overall system behavior through emergent collective intelligence.

Decentralized architectures can range from fully distributed systems where all robots have equal authority, to hybrid approaches that incorporate multiple levels of hierarchy while maintaining distributed decision-making capabilities. communication.

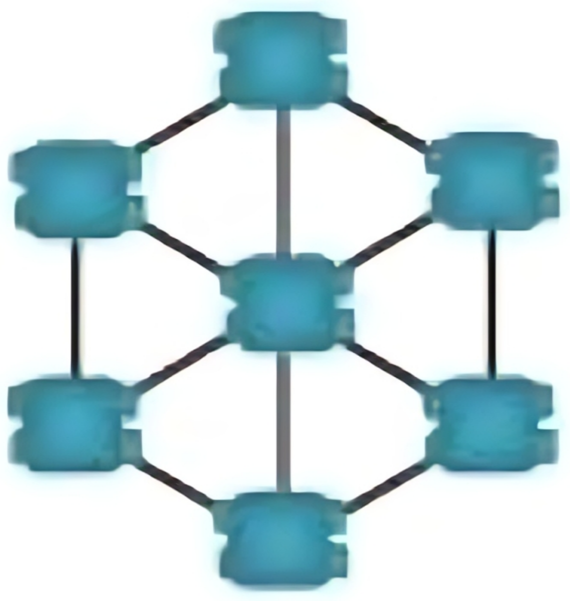


Figure 4.2: Decentrelized MRS

4.2.3 Comparative Analysis

According to [27], [28] and [29] following table provides a systematic comparison between centralized and decentralized multi-robot systems across key performance dimensions:

Characteristic	**Centralized Systems**	**Decentralized Systems**
Decision Making	Single central authority makes all major decisions	Distributed decision making across multiple agents
Scalability	Poor scalability due to central bottleneck	Good scalability with distributed processing
Fault Tolerance	Single point of failure vulnerability	Robust to individual robot failures
Optimality	Can achieve globally optimal solutions	Local optimization may lead to suboptimal global performance
Communication	Hub-and-spoke topology with high central traffic	Peer-to-peer communication with distributed load
Implementation	Simpler design and debugging process	Complex coordination algorithms required
Real-time Performance	Limited by central processing delays	Better real-time response through local processing
Predictability	Highly predictable system behavior	Emergent behaviors can be unpredictable

Table 4.1: Comparison of Centralized and Decentralized Multi-Robot Systems

4.3 Problem formulation

This chapter presents an extension of the previously developed single-robot navigation framework to a multi-robot system comprising two autonomous agents. Building upon the established deep reinforcement learning (DRL) methodology for mapless navigation, the system architecture incorporates inter-robot communication protocols that enable real-time data exchange, particularly goal position information, to facilitate coordinated task execution. Each robot within the system operates under a multi-objective optimization framework that simultaneously addresses three critical navigation constraints: obstacle avoidance with respect to static environmental features such as walls and dynamic obstacles, collision avoidance between the two robotic agents, and successful path planning to achieve the individually assigned target destinations. The distributed control strategy ensures that both robots can navigate autonomously while maintaining situational awareness of their counterpart's intentions and trajectories, thereby enabling safe and efficient multi-agent coordination in unknown environments without reliance on prior environmental mapping.

Chosen Approach
Building on the initial experience with the single-robot system developed in Chapter 3, and informed by the five experimental stages that followed, we explored both centralized and decentralized control strategies. Through this comparative experimentation, it became clear that centralized control was not well-suited to the specific nature of our tasks and system constraints. Consequently, we transitioned to a decentralized multi-robot system, which better aligned with the functional requirements and research objectives of this thesis.

4.4 Environment Setup

For this multi-robot navigation study, we adopted the same simulation environment described in Section 3.3.1, with the key modification of incorporating a second robot into the system. Both robots share identical physical characteristics and sensor configurations, maintaining the same differential drive kinematics and 2D LiDAR specifications as the single-robot setup. For gazebo environement, same ones used in 3.3.1 are used for the 2 robot system with both robots spawn at the initial poses {[2, 2], [-2, -2]}

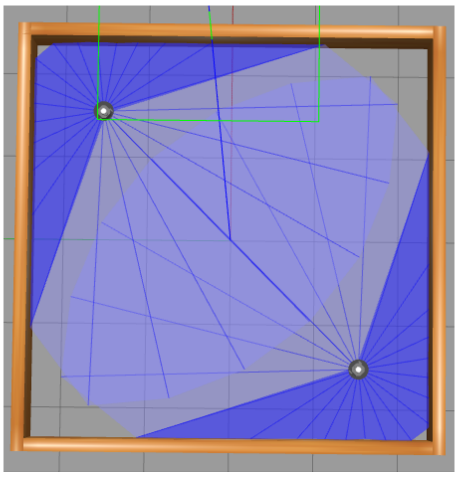


Figure 4.3: stage1 MRS

4.4.1 Robot-Robot Detection Challenge

A critical challenge emerged from the physical configuration of the robots: since both TurtleBot3 platforms are equipped with LiDAR sensors mounted at identical heights (approximately 0.18m above ground), the laser beams pass over each robot without detecting them. This geometric limitation means that robots can not perceive each other through their primary sensing modality, creating a significant obstacle detection blind spot that would inevitably lead to robot-robot collisions during close encounters. To address this fundamental sensing limitation, we implemented a simple yet effective communication protocol that enables explicit coordination between the robots. The solution leverages the ROS 2 communication infrastructure already present in our simulation environment.

Solution

To address this fundamental sensing limitation, we implemented a simple yet effective communication protocol that enables explicit coordination between the robots as shown in figure 4.4.

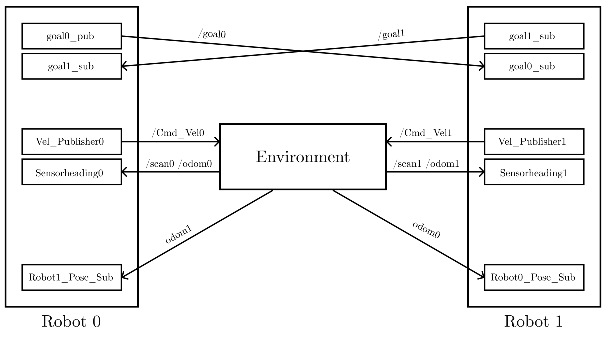


Figure 4.4: Robots communication diagram.

The solution leverages the ROS 2 communication infrastructure already present in our simulation environment. the communication architecture is seated-up as follows:

- Each robot publishes its current pose (position and orientation) to its own dedicated ROS 2 topic
- A communication node is implemented for each robot that subscribes to the other robot's pose topic
- This creates a unidirectional subscription pattern where each robot can access the other's pose information
- Real-time pose data is processed locally by each robot to compute relative distances and positions using following equation :

$$\begin{aligned}\Delta x &= x_{bot1} - x_{bot0}\\ \Delta y &= y_{bot1} - y_{bot0}\\ \text{relative\_d} &= \sqrt{(\Delta x)^2 + (\Delta y)^2}\\ \theta &= \arctan 2(\Delta y, \Delta x)\end{aligned} \tag{4.1}$$

4.4.2 Reward Desing

In this experiment, the same reward used in 3.5 is sophisticated to fulfill the needs we are imposing. A term is added to the reward which penalizes the robots if they approach each other:

$$r_{\text{relative\_d}} = \begin{cases} -3 \cdot \exp(-\text{relative\_d}), & \text{if relative\_d} < 0.7 \\ 0, & \text{otherwise} \end{cases}$$

where **relative_d** is the distance between robots defined in 4.1.

Overall, the basic reward function used in the first stage for each robot is now defined as:

$$r = \begin{cases} +100, & \text{if the goal is reached } (r_g) \\ -100, & \text{if a collision occurs } (r_c) \\ -100, & \text{if the episode times out } (r_t) \end{cases}$$

$$\text{Otherwise,} \quad r = 10 \cdot (d_{t-1} - d_t) - \frac{|\omega_t|}{2} + r_{\text{relative\_d}}$$

As the complexity of the environment incrementally increases with each progressive stage of our experimental framework, we introduce a sophisticated **negative reward mechanism** designed to penalize robots for approaching dangerous zones, obstacles, and environmental boundaries. This penalty-based approach serves as a crucial safety measure that actively discourages the autonomous agents from venturing too close to potential collision sources, including static obstacles, dynamic barriers, and the perimeter walls of the operational environment.

The implementation of this negative reward system ensures that robots develop risk-averse navigation strategies while simultaneously pursuing their primary objectives. The negative reward signal becomes progressively stronger as robots approach danger zones, creating a natural repulsive force that guides them toward safer trajectories. This collision avoidance strategy is particularly effective in multi-robot scenarios where coordination between multiple autonomous agents becomes critical, as each robot learns to maintain safe operating distances from both environmental hazards and other robots in the shared workspace. More comprehensive details and experimental validation of this approach are thoroughly discussed in Section 3.5.

4.4.3 Networks and Learning

Given the development of a two-robot system and the specific objectives and requirements outlined in the Problem Formulation section, we analyzed the performance results from our single-agent experiments presented in Tables 3.5, 3.6, and 3.7. Based on these findings, we adopted a decentralized control approach utilizing two independent Twin Delayed Deep Deterministic Policy Gradient (TD3) network architectures to control each robot autonomously. Both networks share the same hyper parameters displayed in 3.4 and the actions space in 3.4.1, although each one have its own input tensor formed as shown in figure 4.5.

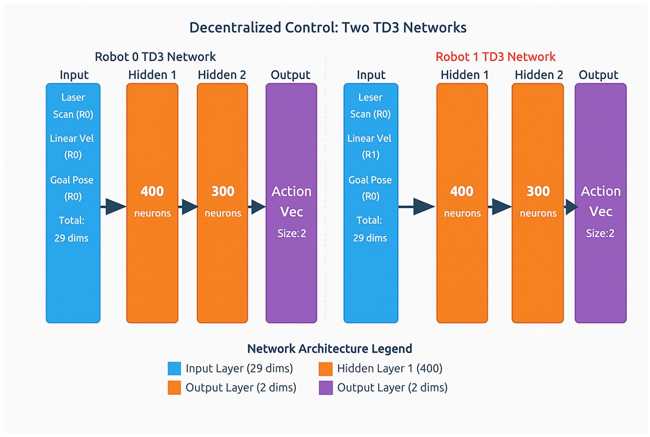


Figure 4.5: MRS Networks

4.5 Simulation and Results

The training process was strategically divided into four distinct stages, each progressively increasing in complexity and environmental challenges. To effectively manage this hierarchical learning structure and ensure optimal knowledge transfer, we adopted a comprehensive transfer learning approach that leverages the accumulated experience from simpler scenarios to tackle more complex navigation tasks.

Training commenced with Stage 1, which presented the most basic navigation scenario with minimal obstacles and straightforward path planning requirements. Upon successful completion of each stage, we carefully preserved the learned model parameters and utilized them as initialization weights for the subsequent, more challenging stage. This methodical approach to knowledge transfer ensured that the agent could build upon previously acquired navigation skills rather than starting from scratch at each new complexity level.

This transfer learning strategy provided several significant advantages throughout the training process. First, it enabled continuous and cumulative learning, allowing the agent to retain valuable navigation behaviors and decision-making patterns learned in simpler environments. Second, it substantially reduced the overall training time required for each stage, as the model could leverage pre-existing knowledge rather than exploring the entire action space from random initialization. Third, it improved training stability by providing a solid foundation of learned behaviors that could be refined and adapted to new challenges.

The progressive complexity structure was designed to systematically introduce new challenges while maintaining learning stability. Stage 1 focused on basic obstacle avoidance and goal-reaching in sparse environments. Stage 2 introduced additional static obstacles and more complex spatial arrangements. Stage 3 incorporated dynamic elements and increased environmental variability. Finally, Stage 4 presented the most challenging scenarios with dense obstacle configurations and complex navigation requirements.

In the following sections, we provide a comprehensive discussion and interpretation of the simulation results obtained across all four training stages. Our evaluation focuses on several key performance metrics that collectively provide insight into the learning dynamics and model behavior. These metrics include the cumulative reward progression, which indicates the agent's improving ability to successfully navigate and reach goals; the actor loss evolution, which reflects the stability and convergence of the policy network; and the critic loss patterns, which demonstrate the accuracy of value function estimation throughout the learning process. Additionally, we analyze convergence rates, training stability, and the effectiveness of knowledge transfer between stages. These comprehensive indicators provide valuable insight into the learning behavior, adaptation capabilities, and overall stability of the model throughout the entire training process, enabling us to assess both the individual stage performance and the cumulative benefits of our progressive learning approach.

4.5.1 Stage 1

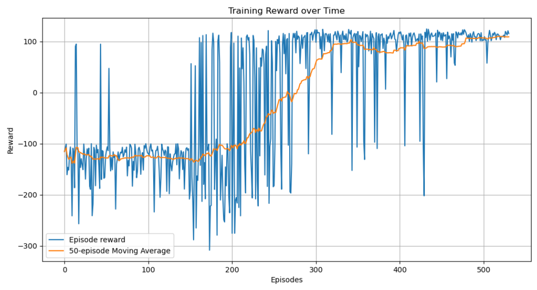


Figure 4.6: Robot 0 reward

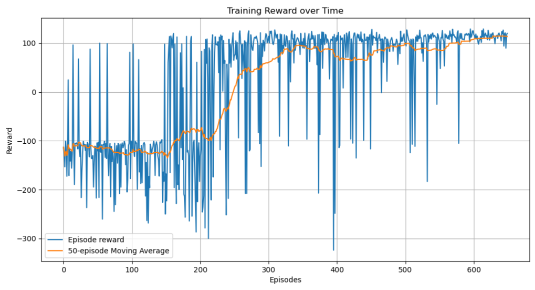


Figure 4.7: Robot 1 reward

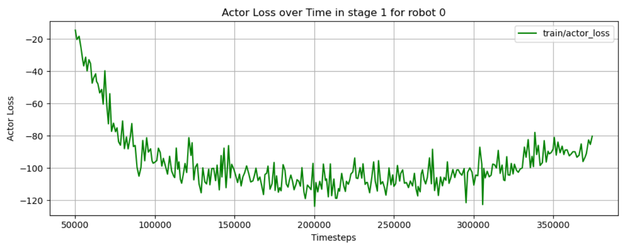


Figure 4.8: Robot 0 actor loss

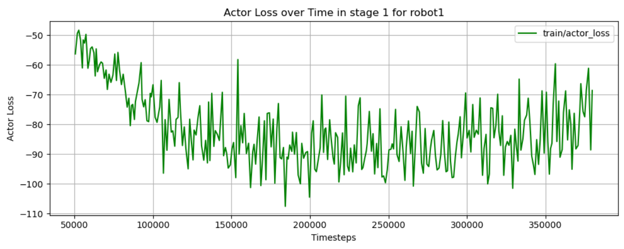


Figure 4.9: Robot 1 actor loss

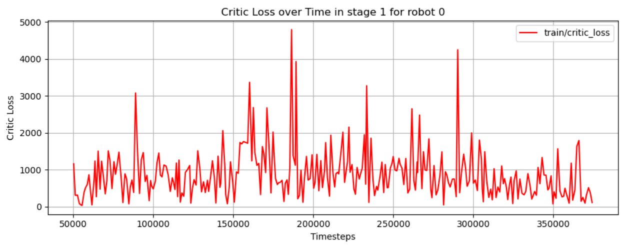


Figure 4.10: Robot 0 critic loss

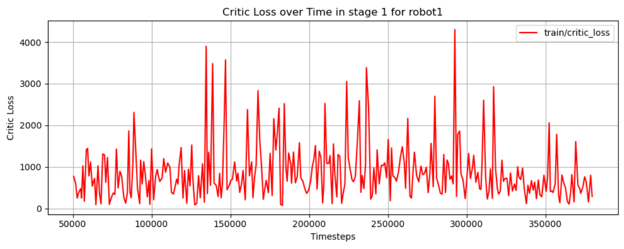


Figure 4.11: Robot 1 critic loss

Figure 4.12: Stage 1 results: Rewards, actor losses, and critic losses for both robots

Comments

Reward : Both robots follow nearly identical learning curves in Stage 1, improving from 125 to 420+ within 350 episodes, then stabilizing around 420–450. The synchronized progress and consistent rewards reflect effective decentralized training and solid early-stage performance.

Actor Loss Actor losses for both robots show similar high-variance patterns, dropping from -50 to fluctuating between -70 and -105. This lack of convergence indicates active exploration and incomplete policy learning, typical in early training stages.

Critic Loss Critic loss is highly variable, spiking past 4000 with baseline around 500–1000. These fluctuations reflect early-stage TD learning and large prediction errors as critics adapt to fast-changing policies.

4.5.2 Stage 2

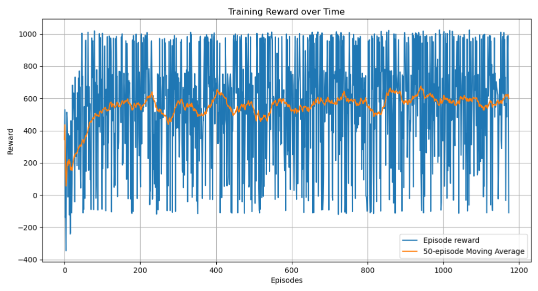


Figure 4.13: Robot 0 reward

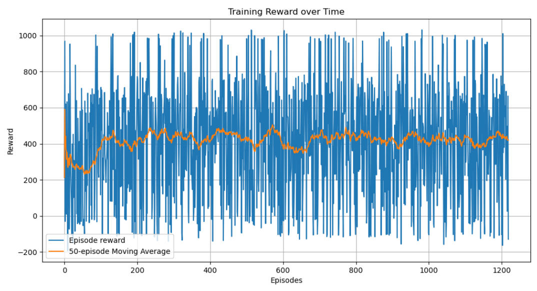


Figure 4.14: Robot 1 reward

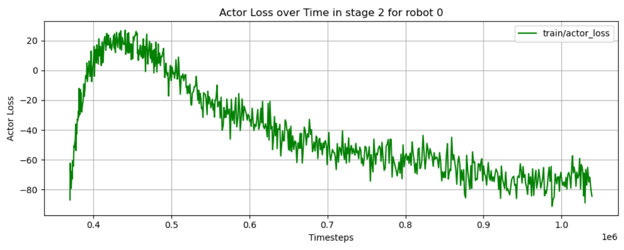


Figure 4.15: Robot 0 actor loss

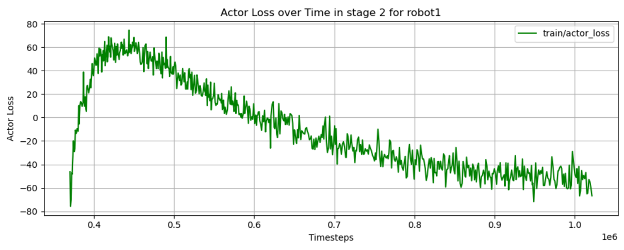


Figure 4.16: Robot 1 actor loss

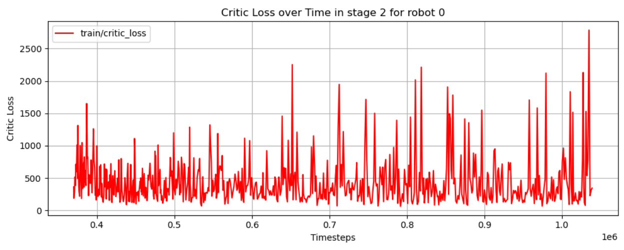


Figure 4.17: Robot 0 critic loss

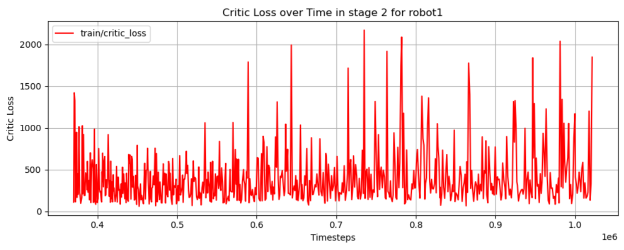


Figure 4.18: Robot 1 critic loss

Figure 4.19: Stage 2 results: Rewards, actor losses, and critic losses for both robots

Comments :
Reward : Robot 0 improves from -60 to $\tilde{4}70$, while Robot 1 reaches $\tilde{4}20$. Both follow three phases: exploration, rapid learning (0.4–0.6M), and stable performance. Robot 0 slightly outperforms, likely due to better interaction or initialization. Unlike other stages strong fluctuations can be remarked.

Actor Loss : Actor loss steadily decreases for both. Robot 0 converges to -70, Robot 1 to -50. The trends are smooth, showing stable policy updates. Robot 1's lower loss suggests more efficient policy learning.

Critic Loss : Critic loss is variable, with spikes over 2500 and baseline around 200–500. This reflects active learning and proper value updates. Both critics adapt well without signs of instability.

4.5.3 Stage 3

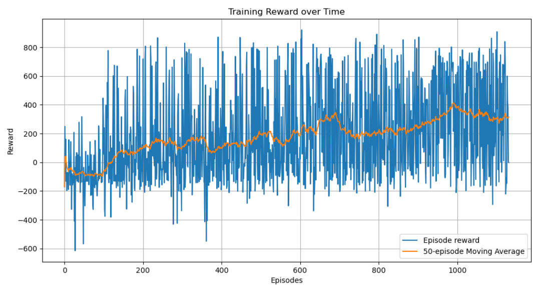


Figure 4.20: Robot 0 reward

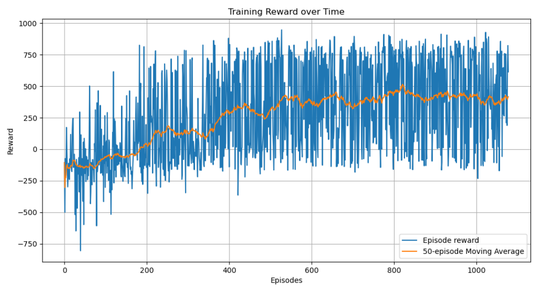


Figure 4.21: Robot 1 reward

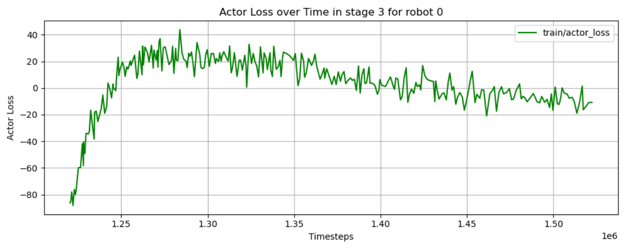


Figure 4.22: Robot 0 actor loss

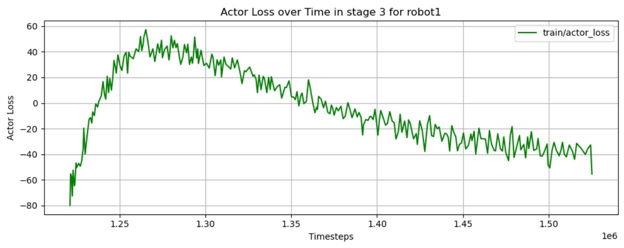


Figure 4.23: Robot 1 actor loss

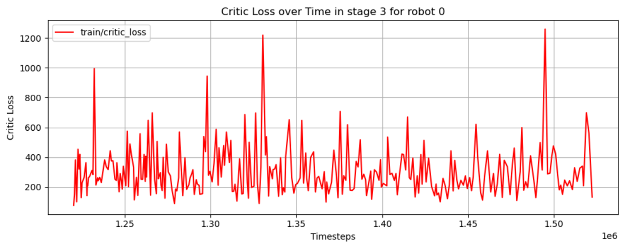


Figure 4.24: Robot 0 critic loss

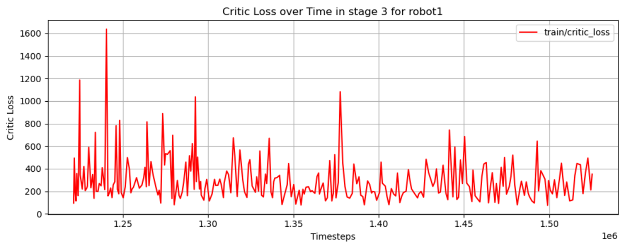


Figure 4.25: Robot 1 critic loss

Figure 4.26: Stage 3 results: Rewards, actor losses, and critic losses for both robots

Comments :
Reward : Both robots had continuous increment in their average reward although robot 1 had faster increase comparing its counterpart.

Actor Loss : Similar behavior is remarked though, robot 0 converges rapidly where robot 1 took more time to get to its convergence.

Critic Loss High strikes are seen in both plots, despite that it decreases over time for robot1 unlike robot 0 which sometimes had large strikes that exceeded the 4000 limit.

4.5.4 Stage 4

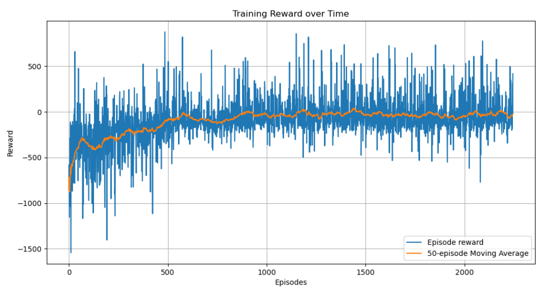


Figure 4.27: Robot 0 reward

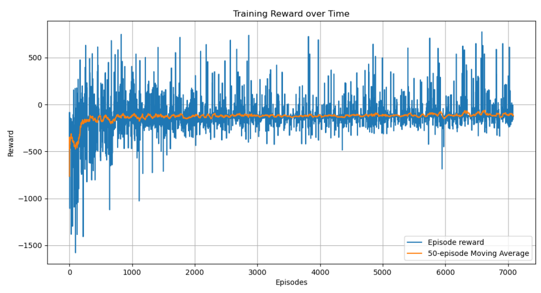


Figure 4.28: Robot 1 reward

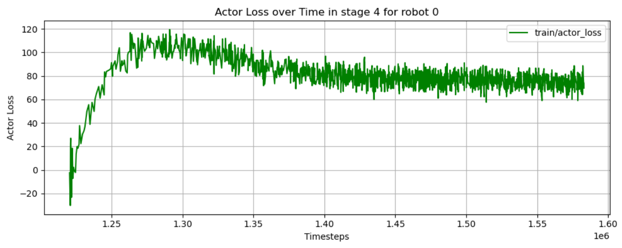


Figure 4.29: Robot 0 actor loss

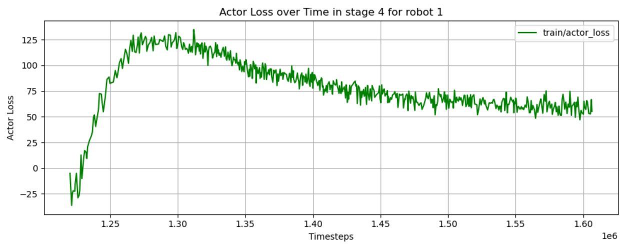


Figure 4.30: Robot 1 actor loss

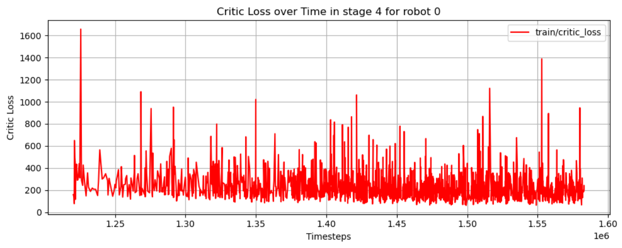


Figure 4.31: Robot 0 critic loss

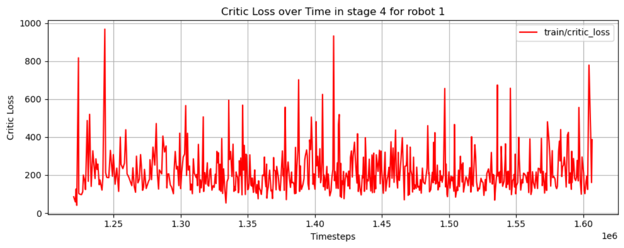


Figure 4.32: Robot 1 critic loss

Figure 4.33: Stage 4 results: Rewards, actor losses, and critic losses for both robots

Comments :
Reward : This last stage marked the smoothest values in terms of reward comparing to other stages regardless of the final value which was low for the reason that numerous walls existed in stage 4.

Actor Loss : Rapid convergence with small fluctuations for both robots.

Critic Loss Strikes still exists despite the long time of learning from stage 1 to 4 although lower values were reached comparing to previous stages.

Stage	Learning Time
1	8h 15min
2	12h 25min
3	14h 05min
4	16h 00min

Table 4.2: Training duration for each stage

Stage	Success Rate (%)
1	93
2	95
3	90
4	80

Table 4.3: Success rate for each stage

4.6 Conclusion

Based on our experimental results, the decentralized multi-robot system demonstrated successful coordinated navigation across all four stages, confirming that our TD3-based approach works effectively for multi-agent mapless navigation. We managed to solve the critical robot-robot detection problem through our communication protocol, which compensated for the LiDAR height limitation that prevented robots from sensing each other directly. Both robots showed remarkably similar learning patterns and reached stable performance levels, with rewards consistently hitting 420-470 points in the later stages. This confirms that our decentralized approach allows each robot to make autonomous decisions while still coordinating effectively through minimal communication.

Our progressive training strategy proved successful in transferring knowledge between stages. We observed reduced training instability as we moved through the stages, and the robots maintained their performance levels even as environments became more complex. The pose-sharing mechanism we implemented worked well - it solved the detection problem and enabled collision avoidance through our relative distance calculations and penalty rewards. Throughout all stages, the robots mostly maintained safe distances from each other without compromising their individual navigation goals.

Looking at the training progression, we found some interesting patterns. Stage 1 gave us a solid foundation with both robots learning at similar rates and stabilizing around 420-450 reward points. In Stage 2, we saw better policy convergence with smoother actor losses, though the robots started showing some individual differences in performance. Stage 3 was particularly interesting because Robot 1 converged faster than Robot 0, while Stage 4 showed the smoothest reward curves despite being our most complex environment. Our success rates (93%, 95%, 90%, 80%) were generally strong, though we did see the expected drop in Stage 4 due to increased environmental complexity.

Comparing our results to the single-robot case, we achieved slightly lower success rates, which makes sense given the added complexity of coordinating two agents in the same space. This reduction reflects the natural trade-off in multi-agent systems, each robot now has to balance reaching its own goal with avoiding the other robot and maintaining cooperative behavior. The loss patterns we observed were healthy throughout training. Actor losses settled between -50 and -70, and while critic losses had some spikes above 4000, this variability is typical for TD learning in multi-agent environments and didn't prevent convergence.

Our decentralized architecture showed good fault tolerance since each robot can operate independently, and it scales better than centralized approaches would. The communication overhead remained manageable, and both robots learned in parallel effectively. Training times increased reasonably with complexity (from 8h 15min to 16h), showing our approach is computationally efficient. The progressive training method handled the complexity scaling well while keeping learning stable.

This work validates that decentralized multi-robot navigation using DRL is feasible in unknown environments. We achieved good coordination between autonomous agents, and our transfer learning approach provides a foundation for scaling to larger robot teams. The slightly lower success rates compared to single robots point to areas we can improve - better communication protocols, more sophisticated reward shaping, and maybe incorporating explicit cooperation strategies into the learning process. Our experimental framework gives us a solid platform for future multi-robot research, with proven methods for progressive training and effective decentralized coordination.

Demonstating Videos

(a) Experiment 1

(b) Experiment 2

(c) Experiment 3

Figure 4.34: QR codes to demonstration videos.

General Conclusion

The Takagi-Sugeno fuzzy modeling approach employed for trajectory tracking provided exact representation of nonlinear dynamics through sector nonlinearity methods, eliminating approximation errors common in traditional linearization techniques. The Parallel Distributed Compensation control strategy, optimized through Linear Matrix Inequalities, ensured robust stabilization with guaranteed convergence rates even under challenging initial conditions that would cause conventional controllers to fail. The integration of T-S observers successfully addressed real-world sensor noise and measurement uncertainties, demonstrating superior performance and reliability for autonomous navigation applications.

Artificial Intelligence (AI) has become an increasingly vital tool in autonomous robotics, offering sophisticated techniques for navigation, learning, and optimization of complex multi-agent systems. In this thesis, we explored several AI-driven approaches to enhance mapless navigation capabilities for mobile robots, with particular emphasis on deep reinforcement learning algorithms and multi-robot coordination strategies.

Through comprehensive chapters, we have detailed our approach, findings, and the challenges encountered in developing robust autonomous navigation systems without prior environmental knowledge. We began by introducing the theoretical foundations of deep reinforcement learning, examining its potential applications and advantages in robotic navigation scenarios. Building upon this foundation, we investigated DRL-based approaches for mapless navigation, conducting extensive simulations on single robot systems to identify and select the most effective algorithm for our navigation framework.

Our comparative analysis of various DRL algorithms demonstrated promising results in accurately modeling navigation behaviors and decision-making processes. The TD3 algorithm was selected and subsequently implemented and rigorously tested in multi-robot system architectures, showing robust capability to handle complex coordination tasks and dynamic environments. Finally, we developed a fuzzy control approach to ensure reliable trajectory tracking, providing a dependable method for real-time navigation control and performance assurance across diverse operational scenarios.

The results demonstrated the efficiency of our integrated approach, with the multi-robot coordination strategies adding flexibility or capabilities that significantly enhanced the overall navigation performance. This finding underscores the importance of adaptable learning algorithms and intelligent coordination mechanisms in optimizing autonomous navigation for various robotic applications.

Looking forward, there are several avenues for improving and extending the capabilities of our mapless navigation system. Future work will focus on real-time implementation of these methods to further validate their practical applicability in dynamic environments. Additionally, efforts will be made to improve the algorithms with the goal of achieving full autonomy in more complex scenarios, including advanced dynamic obstacle avoidance and adaptive behavior learning. Furthermore, investigating the integration of additional sensor modalities could provide richer environmental perception, enhancing the algorithm's accuracy and robustness in various operational contexts. In the future, we also aim to apply this system to multiple robot configurations and design state observers to reduce the dependency on physical sensors, thereby optimizing cost and system complexity.

In summary, this thesis demonstrates the powerful capabilities of AI in advancing autonomous mobile robotics, offering innovative solutions for complex navigation, coordination, and optimization challenges. The promising results in mapless navigation and multi-robot systems pave the way for future research and practical applications, ultimately contributing to the development of smarter, more efficient autonomous robotic systems capable of operating reliably in unknown and dynamic environments.

Bibliography

[1] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. A stable tracking control method for an autonomous mobile robot. In *Proceedings of the International Conference on Artificial Life and Robotics*, Cincinnati, OH, USA, 13-18 May 1990.

[2] Steven M. Lavalle. *Planning Algorithms.* Cambridge University Press, 2006.

[3] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[6] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[7] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018.

[8] Kamyar Mehran. Takagi-sugeno fuzzy modeling for process control. Tutorial report, Newcastle University, School of Electrical, Electronic and Computer Engineering, 2008. Industrial Automation, Robotics and Artificial Intelligence (EEE8005).

[9] Mohammed Chadli. *Stabilité et commande de systèmes décrits par des multimodèles.* Thèse de doctorat, Institut National Polytechnique de Lorraine, Nancy, France, 2002. Thèse de doctorat en Automatique / Robotique.

[10] El-Hadi Guechi. *Suivi de trajectoires d'un robot mobile non holonome : approche par modèle flou de Takagi-Sugeno et prise en compte des retards.* Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis, Valenciennes, France, 2010. Spécialité Automatique et Génie Informatique.

[11] Siddharth Ravi. Reinforcement learning across timescales. Master thesis, Delft University of Technology, 2017. Accessed: 2025-05-27.

[12] David Silver. Reinforcement learning course, 2015.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[14] Jens Kober and Jan Peters. *Reinforcement Learning in Robotics: A Survey*, pages 9–67. Springer International Publishing, Cham, 2014.

[15] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[16] Shixiang Gu, Timothy P. Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. *CoRR*, abs/1603.00748, 2016.

[17] Islem Kobbi and Abdelhak Benamirouche. Reinforcement and deep learning-based optimization of pose estimation techniques in single and multi-agent systems: application to aerial and mobile robots. Mémoire de projet de fin d'Études, École Nationale Polytechnique, Département Automatique, Alger, Algeria, 2023.

[18] MathWorks. *Reinforcement Learning Toolbox.* The MathWorks, Inc., Natick, Massachusetts, United States, 2024.

[19] Diego Arce, Jans Solano, and Cesar Beltrán. A comparison study between traditional and deep-reinforcement-learning-based algorithms for indoor autonomous navigation in dynamic scenarios. *Sensors*, 23(24), 2023.

[20] Ameer Hamza. Deep reinforcement learning for mapless mobile robot navigation. Master's thesis, Blekinge Institute of Technology, 2021.

[21] Matej Dobrevski and Danijel Skočaj. Deep reinforcement learning for map-less goal-driven robot navigation. *International Journal of Advanced Robotic Systems*, 18(1):1729881421992621, 2021.

[22] Nguyn Đc Toàn and Gon-Woo Kim. Mapless navigation with deep reinforcement learning based on the convolutional proximal policy optimization network. In *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 298–301. IEEE, 2021.

[23] Hamid Taheri, Seyed Rasoul Hosseini, and Mohammad Ali Nekoui. Deep reinforcement learning with enhanced ppo for safe mobile robot navigation, 2024.

[24] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154, 2004.

[25] Steven Macenski, Tully Foote, Brian Gerkey, William Lalancette, and Louise Woodall. The robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), 2022.

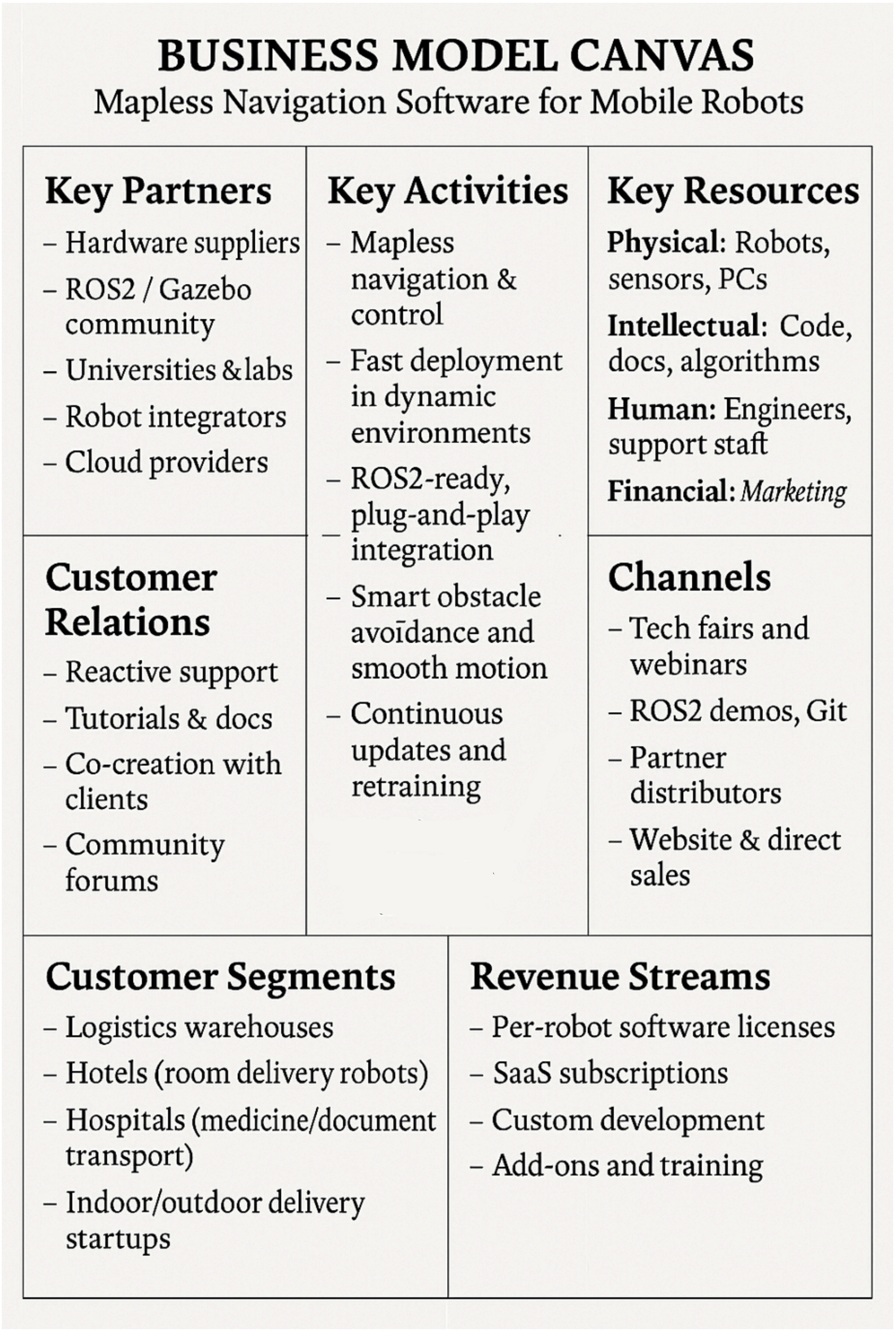
[26] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. *CoRR*, abs/1703.00420, 2017.

[27] M. Bensouici and M. Bouchoucha. Multi-agent deep reinforcement learning for autonomous exploration. *arXiv preprint arXiv:2401.xxxxx.*

[28] S. Zhang, Z. Zheng, and S. Zu. A comprehensive research about multi-robot control models. In *Proceedings of the 2023 International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2023)*, volume 108 of *Advances in Computer Science Research*, 2023.

[29] Fiveable. Centralized vs. decentralized control - swarm intelligence and robotics study guide. Fiveable Library, 2024. Accessed 2025.

Business Model Canvas



BUSINESS MODEL CANVAS

Mapless Navigation Software for Mobile Robots

Key Partners

- Hardware suppliers
- ROS2 / Gazebo community
- Universities & labs
- Robot integrators
- Cloud providers

Key Activities

- Mapless navigation & control
- Fast deployment in dynamic environments
- ROS2-ready, plug-and-play integration
- Smart obstacle avoīdance and smooth motion
- Continuous updates and retraining

Key Resources

Physical: Robots, sensors, PCs

Intellectual: Code, docs, algorithms

Human: Engineers, support staft

Financial: *Marketing*

Customer Relations

- Reactive support
- Tutorials & docs
- Co-creation with clients
- Community forums

Channels

- Tech fairs and webinars
- ROS2 demos, Git
- Partner distributors
- Website & direct sales

Customer Segments

- Logistics warehouses
- Hotels (room delivery robots)
- Hospitals (medicine/document transport)
- Indoor/outdoor delivery startups

Revenue Streams

- Per-robot software licenses
- SaaS subscriptions
- Custom development
- Add-ons and training

Figure 4.35: Business Model Canvas for the proposed appraoch.