People’s Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
National Polytechnic School of Oran Maurice Audin
Electrical Engineering department

GRADUATION THESIS

In fulfillment of the requirements for the **INGENIORAT**’s **degree**

Major : Control Systems Engineering

By :

BOUARICHE Iheb

LAROUSSI Maroua

Subject

Deep Reinforcement Learning Control of a manipulator robot

Publicly defended on MM/DD/YYYY, in front of the jury composed of :

M.	NOM Prénom	Professeur	à	Président
Mr.	LITIM EL Mostafa	Maître de Conférences A	at ENPO	Supervisor
Mr.	BELHERAZEM Adel	Maître de Conférences B	at USTO	Co-supervisor
M.	NOM Prénom	Maître de Conférences A	à	Examinateur

I would like to dedicate this thesis to ... …

Acknowledgement

With a whisper of gratitude and a melody of appreciation, we embark upon the sacred task of acknowledgment.

We would like to express our deepest gratitude and appreciation to all those who have contributed to the completion of this thesis. Without their invaluable support, guidance, and encouragement, this accomplishment would not have been possible.

In the realm of academic exploration, where ideas soar and knowledge blooms, we are grateful to two outstanding people who have played crucial roles in determining the direction of our study, our esteemed supervisor Mr. LITIM El Mostafa, and co-supervisor Mr. BELHERAZEM Adel.

We extend our sincere appreciation to the members of our thesis committee for their genuine interest in our humble work and their diligent evaluation. we are grateful for their time, assistance, knowledge, and essential contributions to our study.

I acknowledge the countless scholars, researchers, and thinkers who have paved the way in our field of study. Our thesis has been inspired and motivated by their revolutionary work and commitment to knowledge advancement.

In the tapestry of my life's journey, our parents and families stand as pillars of unwavering support and unconditional love. We express our deepest gratitude for their endless sacrifices, boundless support, and everlasting faith in our capabilities. Their presence and constant support have been the driving force behind our academic aspirations.

I, LAROUSSI Maroua, am indebted to my remarkable thesis partner, BOUARICHE Iheb, for introducing me to the captivating realm of AI. His passion, knowledge, and guidance have been instrumental in shaping my understanding of this field and igniting a newfound curiosity within me. Thank you for sharing your expertise and being a motivating and inspiring partner in this AI adventure.

Words cannot express my gratitude to my friends, especially NADJI Loubna, TELLI Fatna and RAJAA Rachida, for the support they have shown me throughout my journey. Your presence and assistance have been a constant source of strength and motivation, uplifting me during challenging times. Your friendship has enriched my life in immeasurable ways, and for that, I am truly grateful.

ملخص

تتناول هذه الأطروحة تحليلاً لعدة تقنيات تحكم لنظام ذراع روبوتي. من بين هذه التقنيات تقنية التحكم بالنسبة و التكامل و التفاضل، تقنية التحكم بالعزم المحسوب، تقنية التحكم بالوضع المنزلق، و أيضا تقنيات التحكم الذكي باستخدام التعلم العميق المعزز. الهدف من هذه الأطروحة هو تقييم أداء استراتيجيات التحكم في تنفيذ مهام معينة وبطريقة فعالة من خلال التجارب و المحاكاة. تساعد النتائج على توضيح قدرات هذه الإستراتيجيات وقيودها وحدودها، وكذلك إمكانية تطبيقها العملي بناءً على الاحتياجات الخاصة لتطبيقات الذراع الروبوتي. إضافة على ذلك قد توجه هذه النتائج المهندسين والباحثين في اختيار أفضل طريقة تحكم لهذه الأنظمة.

الكلمات المفتاحية:تقنيات التحكم متقدمة، التعلم العميق المعزز، التحكم بالوضع المنزلق، ذراع روبوتي.

Abstract

This thesis investigates and analyzes several control techniques for the robotic arm system Kuka LBR IIWA 14, including conventional approaches, namely Proportional-Integral-Derivative (PID) control, Computed Torque Control (CTC), Sliding Mode Control (SMC), and intelligent control techniques using Deep Reinforcement Learning (DRL). The objective is to assess these control strategies' performance, in executing accurate and effective manipulation tasks through simulations and experiments. The results help to clarify their capabilities, constraints, and limits as well as their potential for practical implementation. Based on the particular needs of robotic arm applications, these insights may guide engineers and researchers in choosing the best control method.

Keywords: Advanced control, Deep Reinforcement Learning, Sliding Mode Control, IIWA, robotic

Résumé

Ce mémoire examine et analyse plusieurs aspects de commande pour le bras robotique Kuka LBR IIWA 14, à savoir la commande Proportionnel-Intégral-Derivée (PID), la commande par couple calculé (CTC), la commande par mode glissant (SMC) ainsi que les commandes intelligentes basées sur l'apprentissage par renforcement profond (DRL). L'objectif est d'évaluer les performances de ces méthodes de commande dans l'exécution précise et efficace des tâches de manipulation à travers des expériences et des simulations. Les résultats permettent de mettre en évidence leurs capacités, contraintes et limites. Ils montrent également le potentiel de leur mise en oeuvre. Sur la base des besoins spécifiques des applications des bras manipulateurs, ces informations peuvent guider les ingénieurs et les chercheurs dans le choix de la meilleure stratégie de commande.

Mots Clée : Commande avancée, Apprentissage par Renforcement Profond, commande par mode glissant, IIWA, robotique.

Contents

List of Figures

List of Tables

List of Algorithms

General Introduction

Industrial robots were formerly designed to perform specific tasks with an objective in mind, consisting in boosting production speed, saving costs and reducing risks to human partners. These machines were viewed as electro-mechanical devices that operate largely in industrial manufacturing processes and adhere to pre-programmed directives. However, it is expected that, as time goes on, robotic systems will grow more automated and digitalized with the emergence of Artificial Intelligence, since teaching and traditional control techniques are insufficient to accomplish difficult tasks and fulfill the fourth industrial revolution's objectives.

Industry 4.0, denoted I4.0, is characterized by the incorporation of digital technologies such as Artificial Intelligence (AI), the Internet of Things (IoT) and robotics into conventional manufacturing processes. It is built on the idea of creating « Smart Factories », relying on a seamless interconnection and communication between machines, products and people along with the deployment of autonomous systems combining AI and robotics in order to achieve highly flexible and efficient manufacturing processes. AI-powered robots will be able to oversee every stage of manufacturing, from the input of raw materials to the output of finished goods, and control complicated procedures: Based on the information gathered by their sensors, the robots will choose for themselves what tasks to perform, how long to complete them, and in what order, replicating consequently the human intelligence.

The success of Industry 4.0 depends on continuous research and innovation. To satisfy human needs and handle the constraints and issues that may arise from developing these technologies, it is crucial to conduct research through which novel applications are explored, potential flaws are identified, more advanced robots are developed and AI algorithms are improved.

Problematic

In terms of intelligence and decision-making, conventional robotic control techniques are lacking. They can't adapt to changing settings since they use rigid and clear pre-programmed instructions that require accurate models, which makes them unable to manage complicated tasks or unexpected and ambiguous situations and reduces the robotic system's flexibility and autonomy. For informed decision-making, traditional control techniques may not be able to handle large amounts of data or do sophisticated data analysis. These limits show that robots require more intelligent and adaptable control systems to improve decision-making and performance in real-world contexts. This is where Deep Reinforcement Learning can help.

Objective

This thesis aims to explore an advanced approach involving artificial intelligence, known as Deep Reinforcement Learning, from a control system engineer's point of view, exploit it to control a 7 Degree of Freedom robotic arm named Kuka LBR IIWA 14, and compare its performance to the conventional-control-methods'.

Manuscript organization

This thesis is divided into five chapters in order to provide a thorough study. The key chapters are summarised as follows:

- The 1^{st} chapter entitled state of the art, examines the existing research on the Deep Reinforcement Learning control in robotics.
- The 2^{nd} chapter is about modeling the Kuka LBR IIWA robot. It covers various aspects such as kinematics and dynamics.
- In the 3^{rd} chapter, three different conventional control strategies are explored.
- The 4^{th} chapter analyses the key concepts, algorithms and frameworks which constitute the foundations of Deep Reinforcement Learning that will be used for the robot's control.

- The 5^{th} chapter presents the simulations' results, including their interpretation and analysis.

The main results are outlined in the conclusion, along with a recap of the study's goals, a discussion of its contributions, and suggestions for further research.

Chapter 1

State of the art

1.1 Introduction

Deep reinforcement learning (DRL) is a rapidly growing field that combines deep learning and reinforcement learning to enable intelligent agents to learn complex behaviors and perform tasks, which can result in more efficient and effective automation in many sectors. In the realm of robotics, DRL has emerged as a promising area of research with the potential to revolutionize the way robots operate and learn. This chapter aims to review the existing literature, highlighting the significant developments, limitations, and possible applications of DRL-based control for manipulator robots.

1.2 Deep Reinforcement Learning Control literature

In recent years, there was significant interest in the applications of Deep Reinforcement Learning in robotics, and several promising works have been published, including the research paper titled "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates" [1] in 2016, which investigates the use of an off-policy deep reinforcement learning algorithm for robotic manipulation tasks, with asynchronous updates to improve the performance of the robot in accomplishing complex tasks. The experiments involved training a 7 DoF robotic arm to perform various manipulation tasks, such as grasping and moving objects, in a simulated environment. The result has shown that the proposed method outperforms previous approaches in terms of both speed and accuracy since the robot was able to complete the tasks in a short amount of time and with high success rates. Consequently, it has the advantage of being able to handle complex and diverse tasks and allowing faster convergence. However, the reliance on simulation and high computational requirements may limit practical applications. The research suggests that deep reinforcement learning with off-policy algorithms and asynchronous updates can be effective for robotic manipulation tasks.

Another recent work, entitled "Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods" [2] published in 2018 offered a novel approach as it assesses the performance of three off-policy deep reinforcement learning methods for vision-based robotic grasping tasks. The main objective is to train a simulated 7 Dof robotic arm to grasp objects from different positions and orientations and evaluate the performance of these methods which are Soft Actor-Critic (SAC), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Deep Deterministic Policy Gradient (DDPG), with different input representations. The study reveals that SAC and TD3 outperformed DDPG by achieving the highest success rates. It is important to note that augmenting the state representation with depth images improved the performance of all the tested algorithms. The benefits of this approach include reproducible experiments and efficient testing of different algorithms, while the potential lack of translation to real-world scenarios and the inability to capture all nuances of real-world environments in the simulation constitute its limitations.

The research paper titled "A Comparison of Action Spaces for Manipulation Tasks" [3] released in 2019 provided a new insight by focusing on comparing the performance of an on-policy (Proximal Policy Optimization, PPO) and off-policy (Soft Actor-Critic, SAC) RL algorithms across four action spaces for different manipulation tasks in robotics. The authors carried out experiments using a simulated model of a 7 Dof Kuka IIWA 14 arm to test the performance of the action spaces on hammering, object pushing and peg-insertion tasks. In all the experiments, the impedance controller action space outperformed the other three action spaces which are PD, inverse dynamics and direct torque, in terms of task completion time, smoothness of motions and success rate, which constitute its advantages. However, it requires more computational resources. The study concludes that the choice of action space can significantly impact the performance of manipulation tasks and that it should be considered in the design of robotic systems.

A notable contribution has been made in this field with the research paper issued in 2020 and titled "Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization" [4] that aims to investigate the possibility of transferring a Reinforcement

Learning policy trained in simulation to a real-world robotic system without the need for dynamics randomization. The experiments were conducted using a simulated environment where the robotic arm Kuka LBR IIWA 14 was trained with a deep reinforcement learning algorithm to perform a reaching task. The trained policy was then successfully transferred to a real-world robotic system to test its performance, without the need for randomization as the real robot performs the task similarly to the simulated one. The major advantages of the method reside in simplifying the training process and applicability to a wide range of tasks. However, accurate modeling of dynamics and environment in simulation and high computational requirements are potential drawbacks. As a result, sim2real transfer without dynamics randomization can be a promising approach for reinforcement learning in robotics, with potential applications in areas such as manufacturing and healthcare.

1.3 Conclusion

Overall, these research papers mentioned above have demonstrated the potential of Deep Reinforcement Learning in enhancing the performance of robotic manipulation tasks, especially in vision-based grasping, action space optimization, asynchronous off-policy updates, and sim2real transfer learning. They have illustrated the strengths and weaknesses of various Deep Reinforcement Learning algorithms, input representations and training techniques, emphasizing the significance of reproducibility, efficiency, and precision in assessing robotic systems' performance. The promising results have provided a strong motivation for our work, which is based to a certain extent on these studies and aims to explore other approaches and methods for training and evaluating DRL algorithms in the area of robotic manipulation.

Chapter 2

System's modeling

2.1 Introduction

In order to analyze the robotic arm's behavior, it is necessary, to begin with its modeling. Basic physical laws that govern robot dynamics provide the necessary understanding of the mathematical model which enables us to develop efficient control techniques, improve performance, and guarantee accurate and safe operation. In order to forecast the robotic arm's motion, dynamics, and reaction to various inputs, this chapter concentrates on its mathematical representation. The modeling method including topics like kinematics and dynamics modeling will be covered.

2.2 System's description: Kuka LBR IIWA 14

The robotic arm chosen for this study is Kuka LBR IIWA 14. It is a versatile and sophisticated industrial 7-axis (degree of freedom (DoF)) serial manipulator designed for numerous control engineering applications such as human-robot collaboration as it offers many security elements that have granted its certification to work close to people. Kuka LBR IIWA 14 [5].

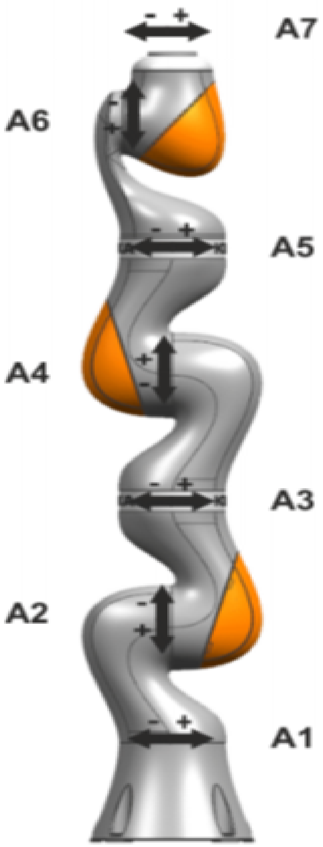


Figure 2.1: Kuka LBR IIWA 14 robot

2.3 Kinematics modeling

2.3.1 Denavit-Hartenberg convention

The Denavit-Hartenberg (DH) convention is frequently utilized in order to define the kinematic characteristics of robotic manipulators [6, 7]. Each joint of the robot is given a coordinate frame, enabling a systematic representation of the robot's geometry and kinematic interactions, as shown in the following figure:

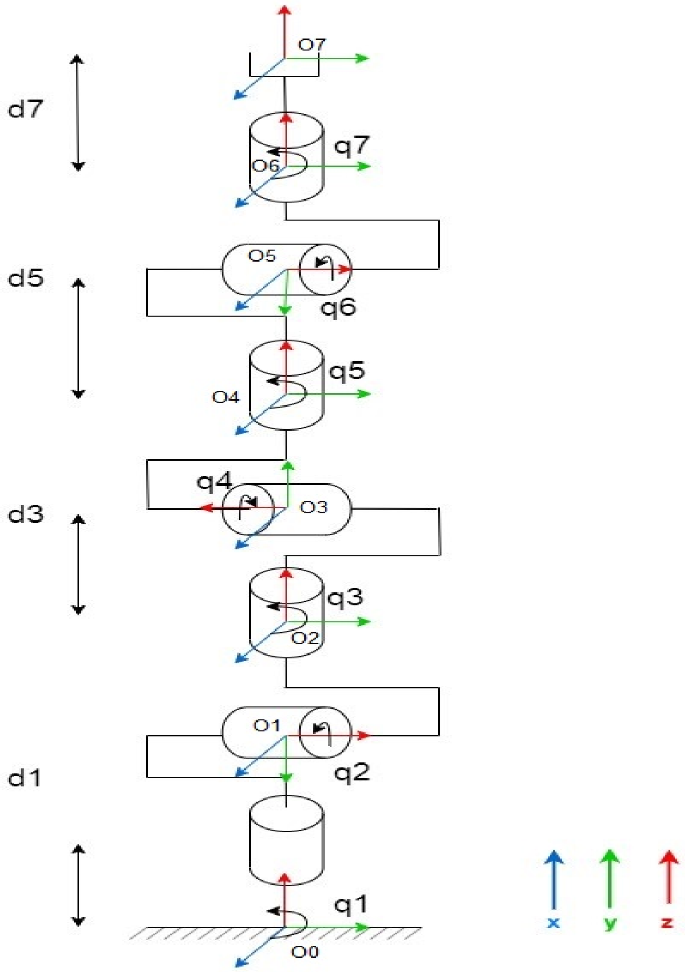


Figure 2.2: Kuka LBR IIWA 14 robot's DH frames

The set of DH parameters that define the kinematic and geometric characteristics of the joints in the Kuka robot are enumerated in the table below [8]:

Table 2.1: DH parameters

link	α(rad)	a(m)	d(m)	q(rad)
1	α_1	0	d_1	q_1
2	α_2	0	0	q_2
3	α_3	0	d_3	q_3
4	α_4	0	0	q_4
5	α_5	0	d_5	q_5
6	α_6	0	0	q_6
7	α_7	0	d_7	q_7

such that:

Table 2.2: DH parameters values

Link	1	2	3	4	5	6	7
α_i	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	0
d_i	0.36	0	0.42	0	0.4	0	0.12

where:

– Link Length (a): distance between z_{i-1} and z_i along x_i.

– Link Offset (d): distance between x_{i-1} and x_i along z_{i-1}

– Joint Angle (q): angle between axes x_{i-1} and x_i corresponding to a rotation about z_{i-1}

– Joint Twist (α): angle between axes z_{i-1} and z_i corresponding to rotation about x_i [7].

Using these parameters, the transformation matrix representing the rotation and translation between successive coordinate frames R_{i-1} and R_i can be computed as follows:

$$^{i-1}T_i = \begin{bmatrix} \cos q_i & -\sin q_i \cos \alpha_i & \sin q_i \sin \alpha_i & a_i \cos q_i \\ \sin q_i & \cos q_i \cos \alpha_i & -\cos q_i \sin \alpha_i & a_i \sin q_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3.1.1 Forward kinematics

The forward kinematics provides the relation that determines the end-effector's position based on the joint angles of the robotic arm relative to the base coordinate frame [9,10]:

$$P = f(q) \tag{2.1}$$

where:

- $q = [q_1 \quad q_2 \quad q_3 \quad \ldots \quad q_n]^T$ represents the joint angles' vector. In our case, $n = 7$.
- $P = [x_e \quad y_e \quad z_e]^T$ represents the end-effector's position vector.

After calculating the final transformation matrix ${}^0T_7$ given by:

$$ {}^0T_7 = {}^0T_1 {}^1T_2 {}^2T_3 {}^3T_4 {}^4T_5 {}^5T_6 {}^6T_7 \tag{2.2} $$

we may obtain the end-effector's pose using the following equation [10]:

$$ \mathbf{p}_e = \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} =^0 T_7 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} $$

2.3.1.2 Inverse kinematics

In contrast to forward kinematics, Inverse kinematics determines the joint angles of the manipulator robot required to achieve the desired position and orientation of its end-effector. In other words, inverse kinematics solves the problem of finding the joint configurations that result in a specific end effector pose [9, 10].

$$ q = f^{-1}(P) \tag{2.3} $$

Having a number of degrees of freedom greater than the operational space limited to 6, the Kuka LBR IIWA robot is considered a redundant robot [7–11]. As a result, there is an infinite number of solutions that can solve the inverse kinematics problem [11]. To address this issue, various techniques have been developed. In this thesis, one of the numerical optimization methods known as the "Quasi-Newton method" will be considered and used to perform the task [12, 13].

2.3.2 Dynamics

Dynamics aids in identifying how forces or torques exerted on the robot impact its motion as well as how the motion of the robot affects the forces or torques that are applied to it.

In what follows, the actuators' dynamics and the friction torques will be neglected for simplification purposes.

2.3.2.1 Forward dynamics

The forward dynamics determines the joint accelerations $\ddot{q}$ given the joint angles q and velocities $\dot{q}$ as well as the applied torques τ, which means that it predicts the resulting motion of the robot's links and joints based on the torques exerted on them [9, 10].

2.3.2.2 Inverse dynamics

On the contrary, inverse dynamics computes the necessary torques to follow the desired trajectory or motion. In other words, it calculates the torques that reproduce a desired motion or trajectory based on the dynamic properties of the robot.

The equations of motion for a robot are often derived using the Lagrange formulation [7, 10, 14–16]. The Lagrangian function is thus defined as the difference between the system's kinetic energy $\mathcal{T}$ and potential energy $\mathcal{U}$:

$$\mathcal{L}(q, \dot{q}) = \mathcal{T}(q, \dot{q}) - \mathcal{U}(q) \tag{2.4}$$

Where the kinetic energy equation is given by:

$$\mathcal{T} = \sum_{i=1}^{7} \frac{1}{2}\dot{p}_i^T m_i \dot{p}_i + \frac{1}{2}\omega_i^T R_i I_{l_i}^i R_i^T \omega_i \tag{2.5}$$

and the potential energy is calculated using:

$$\mathcal{T} = \sum_{i=1}^{7} m_i g^T p_i \tag{2.6}$$

with:

- $\dot{p}_i$ and ω_i representing the linear and angular velocities.
- p_i: the center of mass position of the link i in the base frame.
- $I_{l_i}^i$ representing the inertia tensor relative to the center of mass of the link i expressed in the base frame.
- R_i representing the rotation matrix from the link i frame to the base frame.
- m_i: the mass of the link i.

– g: the gravity acceleration homogeneous vector in the base frame.

The Lagrange equation is then computed as follows:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \qquad i = 1, 2, ..., 7 \tag{2.7}$$

The inverse dynamics derived from the equation 2.7 is expressed by:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau \tag{2.8}$$

where:

– $M(q)$: the mass matrix $(n \times n)$.

– $C(q, \dot{q})$: the Coriolis matrix $(n \times n)$.

– $G(q)$: the gravity term vector.

– τ : the torque vector.

By rearranging the inverse dynamics equations to isolate the joint acceleration $\ddot{q}$, the forward dynamics is obtained:

$$\ddot{q} = M(q)^{-1}(\tau - C(q, \dot{q})\dot{q} - G(q)) \tag{2.9}$$

Remark. It is not uncommon for the calculations involving the homogeneous transformation matrix for a 7-degree-of-freedom (7-DOF) robot to be complex and time-consuming. The symbolic equations representing the elements of the matrix can be quite large and involve various trigonometric functions, making manual calculations cumbersome. Therefore, MATLAB was used to perform these calculations numerically as it is a powerful numerical computing environment of complex equations involving matrices, vectors, and mathematical functions. By using numerical calculations, the time required to obtain the homogeneous transformation matrix is significantly reduced compared to manual symbolic calculations.

2.4 conclusion

By exploring kinematics and dynamics in this chapter, we have developed a mathematical representation of our robotic arm's behavior. This modeling framework will serve as a solid foundation for the subsequent chapters, where we will delve into various control strategies and experimental validation.

Chapter 3

Advanced control of Kuka robot

3.1 Introduction

The control of robotic arms has emerged as a dominating and prominent research area in the realm of robotics. This is mainly attributed to the complex dynamics of the robots, which exhibit non-linearity, strong couplings, and varying parameters over time such as changing loads. The problems encountered in both theoretical studies and practical applications lie in the fact that these systems do not have general tools and techniques to synthesize their controls. Consequently, considerable efforts have been devoted to tackling these challenges, resulting in the emergence of diverse control approaches that have evolved over the past few decades.

In this chapter, three control approaches will be explained, namely Proportional-Integral-Derivative, Computed Torque and Sliding Mode Control, which aim to find the combination of torques that brings the system from an initial configuration to the desired one.

3.2 Proportional-Integral-Derivative (PID) controller

Although many control algorithms are developed to meet the objectives outlined in the specifications, there is a certain reluctance to adopt non-linear control techniques considered difficult to master, complicated to implement and whose performance analysis is complex. Therefore, linear and classical control methods like PID control are employed to solve non-linear problems similar to those posed by robot manipulators. Our study will focus on the method proposed by Khalil and Dombre in [17] where robotic arms are considered linear systems and each of their joints is controlled by a decentralized PID control with constant gains. In other words, the n-DoF manipulator is thought of as a linear n-independent second-order system and is controlled using n-independent SISO (Single Input-Single Output) control systems [17, 18]. Such control is implemented according to the following scheme:

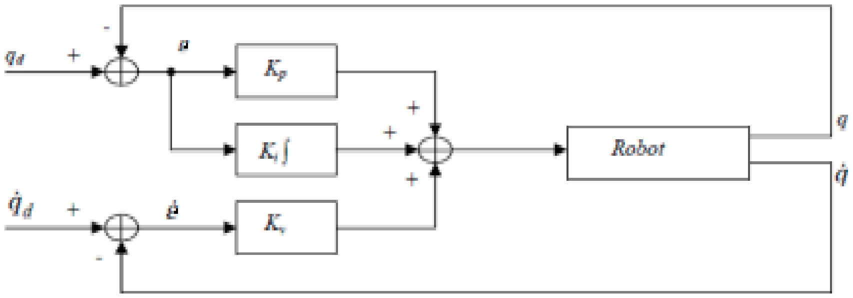


Figure 3.1: PID control scheme.

The control law is given by:

$$\tau = K_p(q_d - q) + K_v(\dot{q}_d - \dot{q}) + K_I \int (q_d - q)d\tau \tag{3.1}$$

Where q_d and $\dot{q}_d$ are the desired positions and velocities in the joint space. K_p, K_v and K_I are $(n \times n)$ positive definite diagonal matrices.

The gains are calculated by taking into account the joint j model represented by the following second-order linear system with constant coefficients:

$$\tau_j = M_j \ddot{q}_j + F_{vj} \dot{q}_j + \gamma_j \tag{3.2}$$

with $M_j = M_{jj_{max}}$ representing the maximum value of the inertia matrix diagonal elements M_{jj}. F_{vj} is the viscous friction and γ_j is a disturbance torque.

For $\gamma_j = 0$, the closed-loop transfer function is obtained by setting the equation3.1

taken for a joint j equal the equation 3.2 and applying Laplace transform:

$$\frac{q_j(s)}{q_{dj}(s)} = \frac{K_{vj}s^2 + K_{pj}s + K_{Ij}}{M_j s^3 + (K_{vj} + F_{vj})s^2 + K_{pj}s + K_{Ij}} \tag{3.3}$$

with :

$$D(s) = M_j s^3 + (K_{vj} + F_{vj})s^2 + K_{pj}s + K_{Ij} \tag{3.4}$$

being the characteristic equation.

The most typical option is to select the gains in a way that results in a negative real triple pole, which provides a quick response without oscillations. The equation 3.4 could be consequently factored out as follows:

$$D(s) = M_j(s + \omega_j)^3 \tag{3.5}$$

with $\omega_j > 0$. The largest possible value of ω_j is selected. However, in order to prevent the system from becoming unstable, this pulse must not be greater than the resonance pulsation [17].

Through identification, we deduce the PID gains:

$$K_{pj} = 3M_j w_j^2 \tag{3.6}$$

$$K_{dj} + F_{vj} = 3M_j w_j \tag{3.7}$$

$$K_{Ij} = M_j w_j^3 \tag{3.8}$$

In our case, $F_{vj} = 0$

Choosing a negative triple pole ensures the stability of the system in the closed loop. This can also be verified using Ruth criterion (linear control).

3.3 Computed Torque Control (CTC)

The fundamental idea behind computed torque control, a potent nonlinear controller, is to linearize a system by canceling its non-linearities namely gravitational, centrifugal, Coriolis and friction torques before applying linear control theory to it, simplifying that way the design of the controller [10, 16, 17, 19]. The computed torque's expression is therefore given by:

$$\tau = M(q)u + C(q,\dot{q})\dot{q} + G(q) \tag{3.9}$$

By equaling the expressions 2.8 and 3.9, the result is as follows:

$$M(q)(\ddot{q} - u) = 0 \tag{3.10}$$

The equation 3.10 shows that the problem is reduced to the control of n second-order, linear, invariant, decoupled systems (double integrators) [9, 10]. Several options can be considered for u. The case of control defined in terms of acceleration and achieved by a Proportional Derivative PD-type compensator is examined in what follows. the control signal u is thus given by [10, 16, 19–21]:

$$u = \ddot{q}_d - K_v\dot{e} - K_p e \tag{3.11}$$

where: $e = q - q_d$ and $\dot{e} = \dot{q} - \dot{q}_d$ are the tracking error (error between the desired and current joint positions) and velocity error (error between the desired and current joint velocities) respectively. K_v and K_p are positive definite diagonal matrices representing the PD controller gains.

Note that the equation 3.10 is equivalent to:

$$\ddot{q} - u = 0 \tag{3.12}$$

By injecting the control signal expression u in 3.12, we derive the governing equation of the error dynamics in the closed loop:

$$\ddot{e}(t) + K_v\dot{e}(t) + K_p e = 0 \tag{3.13}$$

The gain matrices K_v and K_p are chosen to impose the desired dynamics of a second-order system on the error of each axis j, with damping ξ_j and pulsation ω_j. Hence, they are written as follows:

$$K_v = \mathrm{diag}\{K_{vj}\} \qquad \textit{and} \qquad K_p = \mathrm{diag}\{K_{pj}\} \tag{3.14}$$

such that:

$$K_{vj} = 2\xi_j\omega_j \qquad \textit{and} \qquad K_{pj} = {\omega_j}^2 \tag{3.15}$$

In order to prevent overshoot and oscillations from affecting the robot's response, it is recommended to opt for a damping $\xi_j = 1$. The final expression of the torque that the actuators must generate is :

$$\tau = M(q)\ddot{q}_d - M(q)(K_v\dot{e} + K_p e) + C(q,\dot{q})\dot{q} + G(q) \tag{3.16}$$

The following figure represents the Computed Torque Control scheme [22].

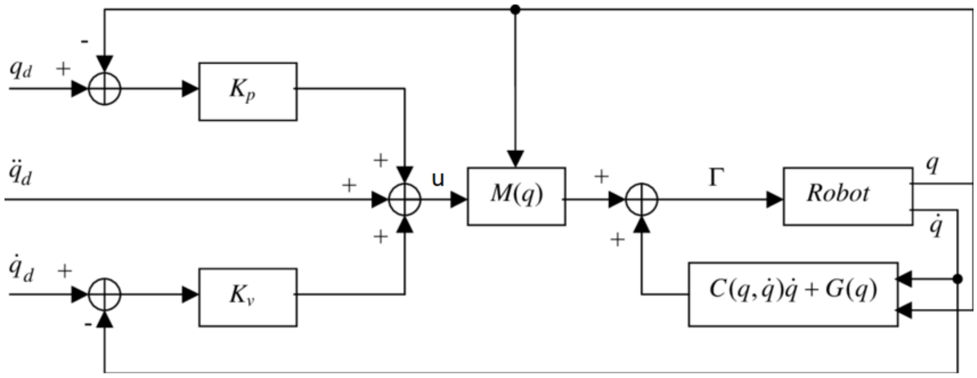


Figure 3.2: Computed Torque Control (CTC) scheme.

By putting $x = [e \quad \dot{e}]^T$, the dynamics of the error governed by the equation 3.12 can be represented in a state space form as:

$$\dot{x} = \begin{bmatrix} 0 & I \\ -K_p & -K_v \end{bmatrix} x = Ax \tag{3.17}$$

Lyapunov's theorem may be used to demonstrate the global and asymptotic stability of the equilibrium point, the following Lyapunov candidate function is selected [9]:

$$V(x) = x^T P x \tag{3.18}$$

whose derivative with respect to time is:

$$\dot{V}(x) = \dot{x}^T P x + x^T P \dot{x} \tag{3.19}$$

such that $P = P^T > 0$, a symmetric positive definite matrix, is the solution of the Lyapunov equation:

$$A^T P + PA = -Q \tag{3.20}$$

With $Q > 0$

By substituting 3.17 in 3.20:

$$\dot{V}(x) = x^T(A^T P + PA)x = -x^T Q x < 0 \tag{3.21}$$

The Lyapunov stability conditions are verified, which implies that the global asymptotic stability of the equilibrium point is guaranteed.

3.4 Sliding Mode Control (SMC)

One of the most crucial issues in automated industrial plants is the control of dynamic systems, particularly robots, with non-linearities, modeling uncertainties, and disturbances like dynamic parameters and effects [23, 24]. Consequently, adopting robust control approaches, such as sliding mode control, has advanced significantly.

Sliding mode control is a particular type of variable structure system. The basic idea behind this control strategy is to bring the trajectory of the system's state variable, regardless of the initial conditions, onto a hyper-surface known as a sliding surface which represents the desired dynamical behavior in the phase plane (reaching mode) and to maintain it on the surface until it tends towards the origin of that plane (the sliding mode) [25].

The sliding mode control law synthesis involves three main steps which are:

3.4.1 Sliding surface design

The general form of the sliding surface that was proposed by Slotine [26, 27] is given by:

$$S(x) = \left(\frac{\partial}{\partial x} + \lambda\right)^{n-1} e(x) \tag{3.22}$$

where: n represents the system's order.
x is the state's vector.
$e(x)$ is the error between the current variable x and the desired one x_d.
λ is a positive coefficient representing the slope of the sliding surface.

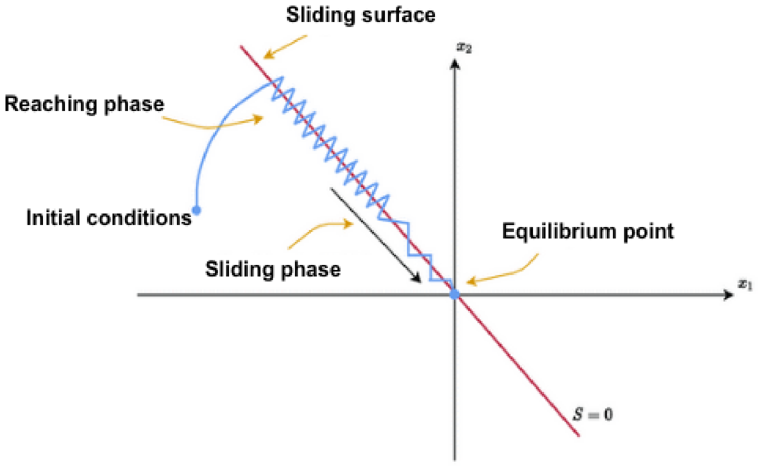


Figure 3.3: Trajectory modes in the phase plane.

In our case, n=2. Therefore, the surface's equation becomes:

$$S(x) = \dot{e} + \lambda e \tag{3.23}$$

where:

$$e = q - q_d = x_1 - x_{1d} \tag{3.24}$$

and

$$\dot{e} = \dot{q} - \dot{q}_d = x_2 - x_{2d} \tag{3.25}$$

By taking $x = [q \quad \dot{q}]^T$, the equation 2.8 can be put in the form of a state representation:

$$\left\{ \begin{array}{l} \dot{x_1} = x_2 \\ \dot{x_2} = M^{-1}(x_1)\tau - M^{-1}(x_1)(C(x_1, x_2)x_2 + G(x_1)) \end{array} \right. = \left\{ \begin{array}{l} \dot{x_1} = x_2 \\ \dot{x_2} = g(x)u + f(x) \end{array} \right. \tag{3.26}$$

$$\text{with}: \left\{ \begin{array}{l} u = \tau \\ g(x) = M^{-1}(x_1) \\ f(x) = -M^{-1}(x_1)(C(x_1, x_2)x_2 + G(x_1)) \end{array} \right. \tag{3.27}$$

3.4.2 The convergence and existence conditions

The conditions that are commonly considered in sliding mode analysis are:

1. A Lyapunov candidate function must be chosen in order to guarantee the convergence and the stability of the state's trajectory on the sliding surface. We usually opt for the following function [24, 25, 27]:

$$V(x) = \frac{1}{2}S^2 \tag{3.28}$$

whose first derivative is:

$$\dot{V}(x) = S(x).\dot{S}(x) \tag{3.29}$$

2. To guarantee the existence and convergence of the sliding mode on the surface, the following equation must be verified [27–29]:

$$S(x).\dot{S}(x) < 0 \tag{3.30}$$

3.4.3 Determination of the control law

The sliding mode control employs a switching control law to direct the system's state trajectory onto the sliding surface. It consists of two components:

$$u = u_{eq} + u_n \tag{3.31}$$

- Continuous control law: The equivalent control term, denoted as u_{eq}, aims to eliminate the non-linearities and uncertainties present in the system dynamics and is in charge of directing the state trajectory of the system along the sliding surface. It is issued from ([29]):

$$\dot{S} = 0 \tag{3.32}$$

After differentiating the equation 3.23, setting it equal to zero and using the state representation of the system along with equations 3.24 and 3.25, we obtain:

$$u_{eq} = g^{-1}(x)(\ddot{q}_d - \lambda\dot{e} - f(x)) \tag{3.33}$$

- Discontinuous control law: When the state of the system deviates from the sliding surface, the discontinuous control law is triggered. In order to make sure the system follows the required reference trajectory, it generates a control action that brings back the system's state to the sliding surface. It is derived from the condition 3.30 which implies that:

$$\dot{S} = -K.sgn(S) \tag{3.34}$$

where: K is a positive definite diagonal matrix.
and *sgn* is a function defined by:

$$sng(S) = \begin{cases} +1 & \text{if } S(x) > 0 \\ -1 & \text{if } S(x) < 0 \end{cases}$$

By setting the derivative of the sliding surface equation 3.23 equal to 3.34, we can determine the control signal expression:

$$u = g^{-1}(x)(\ddot{q}_d - \lambda\dot{e} - f(x)) - g^{-1}(x)K.sgn(S) \tag{3.35}$$

with:

$$u_n = -g^{-1}(x)K.sgn(S) \tag{3.36}$$

3.4.4 The chattering phenomenon

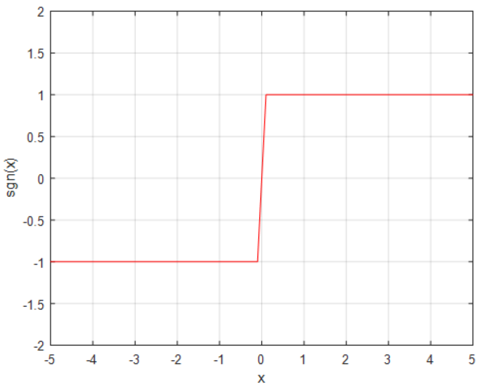
Sliding mode control essentially employs discontinuous control laws to drive the state trajectory of the system on a sliding surface and confine it within its vicinity. However, it exhibits high-frequency oscillations known as Chattering when the system state reaches the sliding surface as shown in the figure 3.3. This causes negative effects on the controlled actuators and activates unwanted dynamics.

The system can achieve smoother control action, lessen components' wear and tear, and enhance the overall system performance and stability by reducing or eliminating chattering. To mitigate it, various techniques can be used:

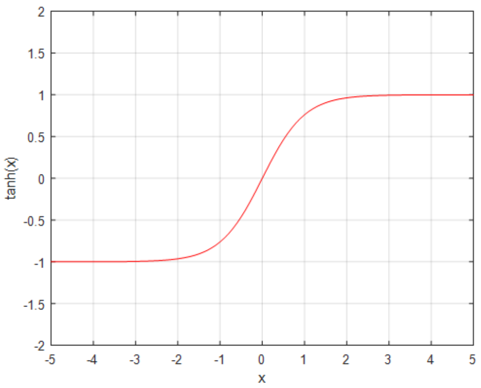
- Replacing the sign function that forms the discontinuous component of the control law by a function that can be thought of as an approximation to the former one such as the sigmoid function and hyperbolic tangent function [23].
- Using a high-order Sliding Mode approach [25].

- Using the Boundary Layer approach introduced by Slotine and Li [26].

The one that is considered in our study is the substitution of the sign function for the hyperbolic tangent.



(a) Sign function



(b) Hyperbolic tangent function

3.5 Conclusion

This chapter has examined three distinct control strategies for a robotic arm: PID, Computed Torque, and Sliding Mode control. Each technique has its own advantages and considerations. The PID control provides a simple and widely-used approach, while the computed torque control offers improved tracking accuracy by compensating for dynamic effects. On the other hand, sliding mode control excels in handling uncertainties and disturbances. However, it is important to note that each approach has its disadvantages in relation to the specific application requirement and constraints.

Chapter 4

Deep Reinforcement Learning

4.1 Introduction

As one of the broad categories of Machine Learning, Reinforcement learning has made remarkable progress in recent years. From a control system engineer's perspective, it provides potent means of searching for optimal controllers for linear and nonlinear systems characterized by deterministic or even stochastic dynamics that are unknown or highly uncertain [30]. Nowadays, Reinforcement Learning has gained more popularity among control engineers after the introduction of Deep Reinforcement Learning, which combines it with Deep Learning to enable agents to solve more complex tasks.

In this chapter, we will explore the basic concepts of reinforcement learning starting with the fundamentals of Markov Decision Processes and then dive into the details of its popular algorithms. We will also explore advanced topics related to deep reinforcement learning giving particular attention to the Deep deterministic policy gradient algorithm, on which this thesis focuses.

4.2 Markov Decision Process

The Markov Decision Process (MDP) is a mathematical framework used to formalize sequential decision-making in uncertain conditions. In other words, MDP offers a formal way to model an RL problem which consists of [4, 31–40]:

1. A set of states, denoted S, including all possible discrete or continuous measurable outputs of the environment that the decision maker, called agent, interacts with.

2. A set of actions, A(s), which clusters all possible control inputs that can be applied to the environment in each state by the agent.

3. A transition model, $P = p(s(t+1)|s(t), a(t))$, which represents the probability of reaching state s_{t+1} by taking a specific action a_t in state s_t.

4. A reward function, $R = E[r(t+1)|s(t), a(t)]$, which is the received reward after transitioning from state s_t to s_{t+1} with action a_t.

The Markov property is assumed to hold for the transition states of the environment. In other words, the probability of reaching state s_{t+1} only depends on s_t, without being influenced by any earlier states.

4.3 Trial and Error

While interacting with the environment, the agent improves its behaviors through a trial-and-error learning cycle. Starting from an initial state, it selects an action following rules it uses to make decisions in each time step and applies it to the environment. A numerical reward r is received along with new observations after the environment transitions to the next state s_{t+1}. The agent, then, makes adjustments based on the feedback until achieving the terminal state in the case of a finite sequence that is usually called a training episode. Reinforcement learning methods are used in this case to help the agent learn from each episode and optimize policy [4, 31–40]. The following figure represents Reinforcement learning's principle [39].

4.4 Policy

A policy is a mapping between states and actions that an agent uses to decide what action to take in a particular state. There are two types of policies [31–37]:

- Deterministic Policy which maps each state to a single action and is defined by $\pi(s) = a$

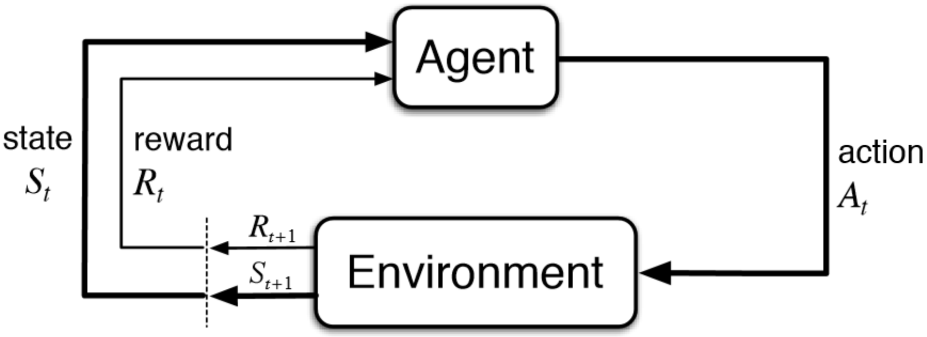


Figure 4.1: The principle of Reinforcement Learning.

- Stochastic Policy that maps each state to a probability distribution over possible actions and is defined by $\pi(a|s) = a$

4.5 Model-free and model-based

It is important to distinguish between model-free and model-based approaches when working with RL algorithms. The model-based approach requires a model that can be used by the agent to learn a policy. On the other hand, in the model-free approach, the agent learns directly from experiences by interacting with the environment through trial and error, without having any previous knowledge about it or a model representing it [31, 32],[[35, 36], [41]. Our thesis will mainly focus on model-free RL methods.

4.6 Cumulative Reward

Also known as a return and generally denoted as G(t), the cumulative reward refers to the total reward an agent accumulates over time as it interacts with the environment [31–35], [37]. The return G(t) on each episode and at each step t can be defined by:

$$G = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.1}$$

Where γ is a discount factor that determines the importance of immediate and future rewards.

4.7 Value Functions

Value functions are functions of states or state-action pairs which quantify how beneficial it is for an agent to be in a particular state, or to perform a particular action in a particular state. These functions can be of two types [31–38], [41]:

– State value: which is the expected reward from a particular state s_t and following the policy π. It indicates how a state is better than other states in the environment and is mathematically defined as :

$$\nu_\pi(s) \overset{\text{def}}{=} \mathbb{E}_\pi \left[G_t \,|\, S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right], \tag{4.2}$$

for all $s \in S$.

– Action value: is the expected reward conditioned by taking a particular action a_t in a particular state st following a policy π. It estimates how an action in a particular state is better than other actions in the same state. Also known as Q-value, it can be calculated as follows:

$$Q_\pi(s,a) \overset{\text{def}}{=} \mathbb{E}_\pi \left[G_t \,|\, S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right] \tag{4.3}$$

4.8 Optimal Policy

It is important to recall that reinforcement learning aims to find the policy which yields the most outcome in the long term. Finding the optimal strategy corresponds to solving the MDP problem and determining the optimal state or action values of an environment [31–39, 41]. The previous ideas can be linked and represented mathematically by the following equation:

$$\pi_*(s) = \arg\max_\pi V_\pi(s) = \arg\max_\pi Q_\pi(s,a) \tag{4.4}$$

4.9 Bellman Equation

Bellman Equation is a fundamental concept in Reinforcement Learning and uncertain decision-making as it enables us to compute the value of a state in a Markov Decision Process (MDP) by iteratively updating the value function based on the immediate reward obtained in that state and the expected value of the successor states. In simpler terms, it represents the recursive relationship between the value of a state and the value of the following state [31, 32], [35–37], [41]. The Bellman equation defines the expected values as follows:

$$\nu_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma\nu_\pi(s_{t+1}) \,|\, S_t = s\right] \tag{4.5}$$

It can also be used to define the action values:

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \,|\, S_t = s, A = a\right] \tag{4.6}$$

4.10 On-Policy and Off-Policy

On-policy and Off-policy are two distinct and essential approaches to reinforcement learning. They differ in the way the agent's policy is updated during training, based on the feedback received from the environment. In fact, On-policy methods update and improve the policy that is used to make decisions and interact with the environment while off-policy methods use data generated by a different policy for updating [30], [34].

The Agent is composed of an RL algorithm that it utilizes to learn the optimal policy and hence maximize the reward it receives from the environment it interacts with. It is important to note that there exists a wide variety of RL algorithms today and that each one of them has its particularities that meet the requirements of a particular problem.

4.11 Temporal Difference Learning TD(0)

Temporal Difference Learning is a value-based, On-policy iterative method that estimates the action values using the Bellman equation. The idea behind TD is to learn by updating the predicted state value using the immediate reward and the difference between the current state value and the next state value, also known as temporal difference error

[31,32], [34,35], [38,39]. TD(0) updates state values from experiences in the environment as follows:

- at each step t:

$$V_t \leftarrow V_t + \alpha(r_{t+1} + \gamma V_{t+1} - V_t) \tag{4.7}$$

Where: α is the learning rate and γ is the discount factor.

TD learning has proven to be a highly effective tool for learning in complex and dynamic environments as it mostly converges on optimal values. It should be mentioned that TD is a general term which refers to a class of Reinforcement Learning algorithms that leverage the temporal difference error to learn from experiences, such as Q-learning.

4.11.1 Q-Learning

Q-learning is one of the most important value-based and Off-Policy methods used in RL to learn action values taking into account the received reward and the maximum estimated Q-value of the next state regardless of the action that the agent takes next [31,32], [34,35], [38], [41]. Q-learning, which uses a similar update rule as TD is defined as follows:

- At each step t

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{4.8}$$

In practical applications, action values are often used rather than state values since the value of the next state cannot be known before performing an action. However, it is possible to estimate the value of each action in a given state, which enables the agent to make more informed decisions.

4.12 Exploration and Exploitation

In order to maximize rewards, the agent must take actions that it knows yield high rewards. On the other hand, to discover new and potentially more effective actions, it also needs to explore actions that haven't been taken yet and observe the result. Therefore, finding the right balance between exploration that refers to the process of seeking out new information about the environment, and exploitation which consists in following the best policy that produces the highest rewards based on past experiences is one of the crucial challenges in Reinforcement Learning. One way to ensure good training is to utilize a decayed noise that starts with a high value, allowing the agent to explore more in the early stages, and gradually diminishes over time to focus on exploiting its knowledge [31,32], [35,36], [39].

4.13 Limitation of Reinforcement Learning

The TD methods that were previously discussed are designed for systems with low-dimensional state and action spaces in simple environments. These problems are solved by creating a Q-value function table with a state-action pair corresponding to each. However, dealing with high dimensional or continuous action and state spaces results in memory and computation difficulties [32,36,40]. To address this issue, function approximation methods such as neural networks can be used. The combination of Neural Networks theory and Reinforcement Learning has led to the emergence of Deep Reinforcement Learning which will be explained after introducing the fundamental concepts of Deep Learning in the following section.

4.14 Deep Learning

Deep learning is a sub-field of Machine Learning that is based on the use of Artificial Neural Networks with multiple layers. These networks are designed to learn from large amounts of data and to generalize that learning to new situations [31–33], [35, 36], [40,42].

One way to address the issue of memory when dealing with large dimensional environments in the context of TD methods is to make use of function approximation techniques. By using a Neural Network as a function approximator, the agent can learn to generalize its value estimates across similar states, even when the number of states is too large to store explicitly in memory and in complex and high-dimensional environments. Additionally, the use of Deep Neural Networks may enable the agent to learn more sophisticated representations of the environment, leading to better perception and decision-making capabilities [32], [36].

Artificial neural networks have turned out to be the heart of deep learning and a key element in the deep reinforcement learning theory. Consequently, it is necessary to understand its fundamental concepts in order to draw a complete picture of the DRL algorithms [32], [36].

4.14.1 Artificial Neural Network

Inspired by the human brain, Artificial Neural Networks are comprised of multiple interconnected nodes, arranged in three different categories of layers in a way that allows them to be universal function approximator: the input layer where input data is fed, the output one that yields the output values and one or more hidden layers. Each node, or neuron, is connected to another and has an associated weight and threshold which activates the node [32], [35, 36].

The neuron first receives one or more weighted inputs, sums them and applies a non-linear function known as activation function to that sum to obtain an output value which can be mathematically represented by :

$$x_j = f\left(\sum_{i=0}^{n} w_{i,j} x_{i,j} + b\right) \tag{4.9}$$

Here, a_j represents the output value of neuron j, $w_{i,j}$ represents the weights associated with the connections, $x_{i,j}$ represents the output value of neuron i connected to neuron j, b represents the bias, and f represents the activation function.

This process is repeated until the last layer, which is the output layer. The resulting estimated output value is then compared with the real one by calculating the loss function.

The learning begins by computing the gradient of the loss function with respect to the network parameters from the output layer and propagating backward through the whole NN (back-propagation).

All the weights of the neural network are then updated based on the gradient descent rule to optimize the loss function.

There exist many activation functions which are meant to introduce nonlinearity to the neural network and make it perform more complex tasks such as approximating functions. ReLU and hyperbolic tangent activation functions can be held as an example.

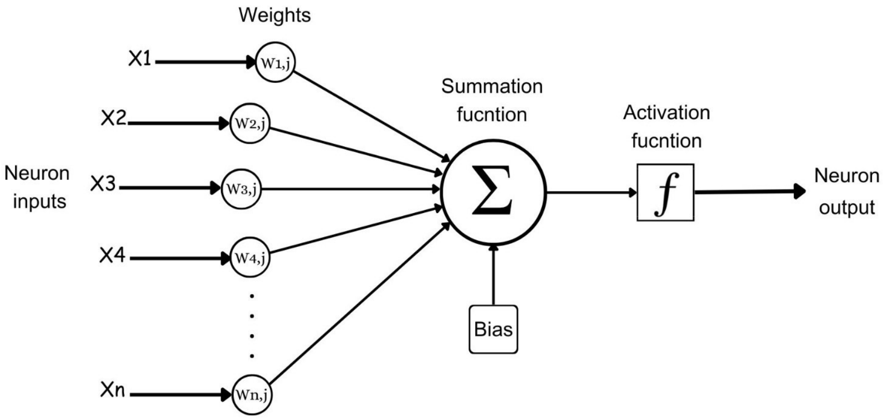


Figure 4.2: The basic element of a Neural Network (neuron).

Many different types of neural network architectures have been developed for various applications. The architecture on which our study focuses is Feed-forward Neural Network. It constitutes the simplest type of Neural Network architecture, which consists of one or more layers of interconnected nodes, with no loops or cycles in the network structure.

4.15 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a sub-field of modern Artificial Intelligence that has greatly progressed over time. Integrating both Reinforcement Learning and Deep Neural Networks methods to enable machines to learn and make decisions in complex environments has led to the development of powerful algorithms and agents that have accomplished impressive tasks so far [31, 32], [40].

4.15.1 Deep Q-Learning

The research paper entitled [Playing Atari with Deep Reinforcement Learning" by Volodymyr Mnih et al., published in 2013] [43] was the explosion of Deep Reinforcement Learning when Deep Q-learning solves ATARI games and surpasses the human expert. The basic idea of this algorithm is estimating the action values in an environment using the Q-learning and the Bellman equation to make the Deep Neural Network learn the action values. This network is called Q-Network. It can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ (Q-learning loss) that changes at each step i:

$$L_i(\theta_i) = \Big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a, \theta_i)\Big)^2 \tag{4.10}$$

with θ_i representing the parameters of the Q-Network. s' is the next state observed after taking action a in state s. a' represents the action taken in the next state observed s'

This algorithm uses an experience replay memory to train the neural network. During the training steps, it selects mini batch random samples from the experience memory and applies the Q-learning update using the gradient descent. This method provides data efficiency, breaks the correlation and reduces the variance of the updates. By using experience replay, the behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.

DQN uses two neural networks. The target network is a copy of the Q-network that is frozen for a certain number of steps and is only updated periodically with the weights from the Q-network. A target network is used to calculate the TD error target value and stabilize the learning process and prevent over-fitting of the Q-network. It also helps to reduce the variance and improve convergence [31], [33, 41], [43]. The Deep Q-learning algorithm is defined as follows [31]:

Algorithm 4.1: Deep Q-Learning Algorithm (adapted)

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $\mathcal{Q}$ with random weights $\theta$
Initialize target action-value function $\widehat{\mathcal{Q}}$ with weights $\theta^- = \theta$
for $episode = 1$ to $M$ do
    Initialize sequence $s_1$
    for $t = 1$ to $T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = argmax_a \mathcal{Q}(s_t, a; \theta)$
        Execute action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
        Sample random mini-batch of transitions $(s_t, a_t, r_t, s_{t+1})$ from $D$
        Set
            $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \widehat{\mathcal{Q}}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - \mathcal{Q}(s_j, a_j; \theta))^2$ with respect to the
          network parameters $\theta$
        Every $C$ steps reset $\widehat{\mathcal{Q}} = \mathcal{Q}$
    end
end
```

Value-based methods are designed for discrete-action space which makes them inadequate for continuous action space challenges. Therefore, other methods were developed to enable Deep Reinforcement Learning in continuous control problems.

4.15.2 Actor-Critic

Actor-Critic methods address the limitations mentioned in the previous section by simultaneously learning a policy and a state value function. The actor and the critic are both represented using neural networks as function approximators: the first one controls the policy and the latter estimates the value function. The actor adjusts the parameters of the policy while the critic approximates the value function for the current policy and criticizes the actions taken by the agent at each time step.

In order to assure a better performance in the long run, the Actor-critic method uses a technique called Policy iteration which consists in alternating between policy evaluation and policy improvement. In policy evaluation, the value function of the current policy is estimated from the outcomes of sampled trajectories whereas in policy improvement, the current value function is used to generate a better policy [31], [36], [38], [41].

Combining policy-based and value-based methods through actor-critic architecture has become a widely spread approach on which modern and more sophisticated techniques, such as the Deep Deterministic Policy Gradient algorithm, rely for solving Reinforcement learning problems.

4.15.3 Deep Deterministic Policy Gradient (DDPG)

Although Deep Q-network has proven to be effective with high-dimensional observation space problems, it can only deal with discrete and low-dimensional action spaces which poses significant problems as many real-world applications, particularly physical control tasks, have continuous and high-dimensional action spaces. A Deep Q-network cannot be straightforwardly applied to continuous domains since it relies on finding the action that maximizes the action-value function. Luckily, the DDPG algorithm, which is chosen for the work covered by this thesis, addresses this limitation: It maximizes the action value function to update the actor NN and minimizes the TD error to update the critic NN.

DDPG is an actor-critic algorithm that learns a Q-function and a policy simultaneously using off-policy data and the Bellman equation to learn the Q-function which is subsequently taken into account in learning the policy. Like deep Q-learning, uses a replay buffer memory and separate target networks for calculating the target value of the TD error.

- Replay buffer memory: consists in storing a finite number of previous experience tuples (state, action, reward, next state). Each time step, a uniformly uncorrelated sampled mini-batch of these tuples is utilized to update both the actor and critic. When the buffer is full, old tuples are replaced by new ones.
- Target networks: copies of the actor and critic networks, called target networks, are created to compute the target value and ensure stable learning with deep learning. However, their weights are updated at a slower rate.

The difference from the Q-network is that we have the actor action and the state as an input to the Q-network and its output is an action value.

DDPG algorithm learns a deterministic actor policy $\mu_\theta(s)$ which gives the action that maximizes the critic $Q_\phi(s, a)$. By sampling a mini-batch from the buffer experiences, the actor network's $(\mu_\theta(s))$ weights are updated using the gradients of the Q-function (with a direction that maximizes the values of the action). On the other hand, Q-network (the critic network $Q_\phi(s, a)$) will be updated by the bellman equation and target networks [30], [32], [35], [37–41], [44].

To encourage exploration and exploitation, a noise is added to the action of the actor and is usually decayed over episodes.

The DDPG algorithm is defined as follows [37]:

Algorithm 4.2: DDPG Algorithm

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for $episode = 1$ to $M$ do
    Initialize a random process $\mathcal{N}$ for action exploration Receive initial observation state $s_1$;
    for $t = 1$ to $T$ do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random mini-batch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss:
            $\mathcal{L} = \frac{1}{N}\sum_i (y_i - Q(s,a_i|\theta^Q))^2 \quad (4.11)$
        Update actor policy using the sampled policy gradient: $\nabla_\theta \mathcal{J}(\theta)$
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \quad (4.12)$
        Update target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \quad (4.13)$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \quad (4.14)$
    end
end
```

4.16 Conclusion

In summary, this chapter has given a thorough overview of the theoretical foundations of Deep Reinforcement Learning. By analyzing the key concepts, algorithms, and frameworks, we have deepened our understanding of the guiding principles of DRL, its advantages and limitations, and its potential to improve the control capabilities of robotic arms. The knowledge learned from this chapter serves as a basis for the experimentation in the subsequent chapter, advancing DRL control in robotics applications.

Chapter 5

Simulations' results

5.1 Introduction

This chapter presents the simulation setup and the obtained results for the control of the manipulator robot Kuka LBR IIWA 14 using various control techniques, namely PID, computed torque, sliding mode, and Deep Reinforcement Learning (DRL) control. The objective is to evaluate the performance and compare the effectiveness of these control strategies in achieving precise and robust robot manipulations.

For this purpose:

- The desired position was set to $P_d = [0.2 \quad 0.5 \quad 0.8]^T$.
- The end-effector initial position is $P_i = [0 \quad 0 \quad 1.3]^T$.
- In the PID control, CTC and SMC case, a trajectory was generated between the two previous positions starting from the initial joints configuration using a 5-degree polynomial. The resolution of the polynomial is contingent upon the absence of velocities and accelerations at the desired point.

5.2 Robot's parameters

SolidWorks was used to collect the robot's parameters. As for the model's validation and simulation, it was done with MATLAB using Simulink and Robotics Systems Toolbox.

Table 5.1: Kuka robot's parameters

Body	Mass(kg)	G_x(m)	G_y(m)	G_z(m)	I_{xx}	I_{yy}	I_{zz}	I_{yz}	I_{xz}	I_{xy}
0	5	0	-0.03	0.12	NaN	NaN	NaN	NaN	NaN	NaN
1	4	0.0003	0.059	0.1995	0.0745	0.1345	0.08	0	0.035	0
2	4	0	0.13	0.39	0.1612	0.1476	0.0236	0.0144	0	0
3	3	0	0.067	0.5985	0.0710	0.0251	0.0579	-0.001	0	0
4	2.7	0.0001	-0.076	0.8010	0.1334	0.1257	0.0127	-0.0117	0	0
5	1.7	0	-0.0006	0.9649	0.0452	0.0131	0.0411	-0.0062	0	0
6	1.8	0	0.02	1.18	0.0306	0.0278	0.0057	-0.0027	0	0
7	0.3	0	0	1.281	0.005	0.0036	0.0047	0	0	0

5.3 PID Control

PID gains were tuned according to the method suggested by Khalil and Dombre [17] which was presented in the 3^{rd} chapter. For:

- all $\omega_j = 20$ rad/s.
- $M_1 = 0.8 \quad M_2 = 0.34 \quad M_3 = 0.12 \quad M_4 = 0.55 \quad M_5 = 0.01 \quad M_6 = 0.01 \quad M_7 = 0.005$.

and Hence:

- $K_p = \text{diag}(960, 4080, 144, 660, 12, 12, 6)$
- $K_v = \text{diag}(48, 204, 7, 33, 0.6, 0.6, 0.3)$
- $K_I = \text{diag}(6400, 27200, 960, 4400, 80, 80, 40)$

the following results were obtained.

5.3.1 Results

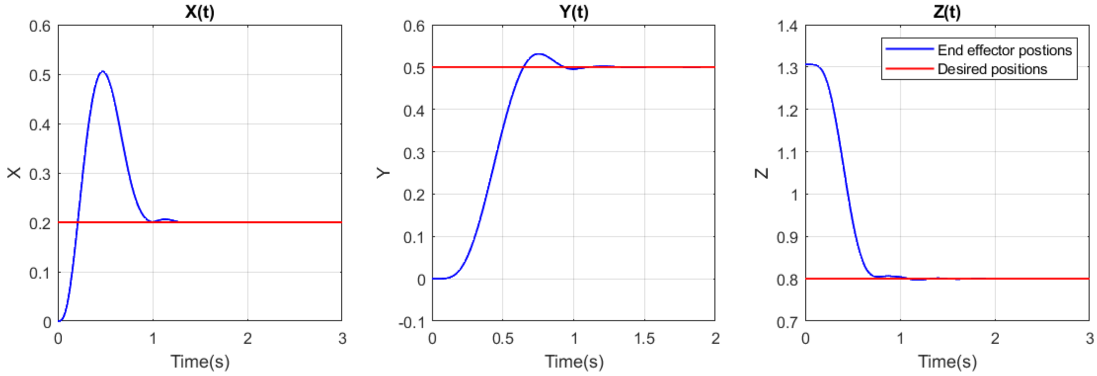


Figure 5.1: PID: End effector's position with respect to the desired position.

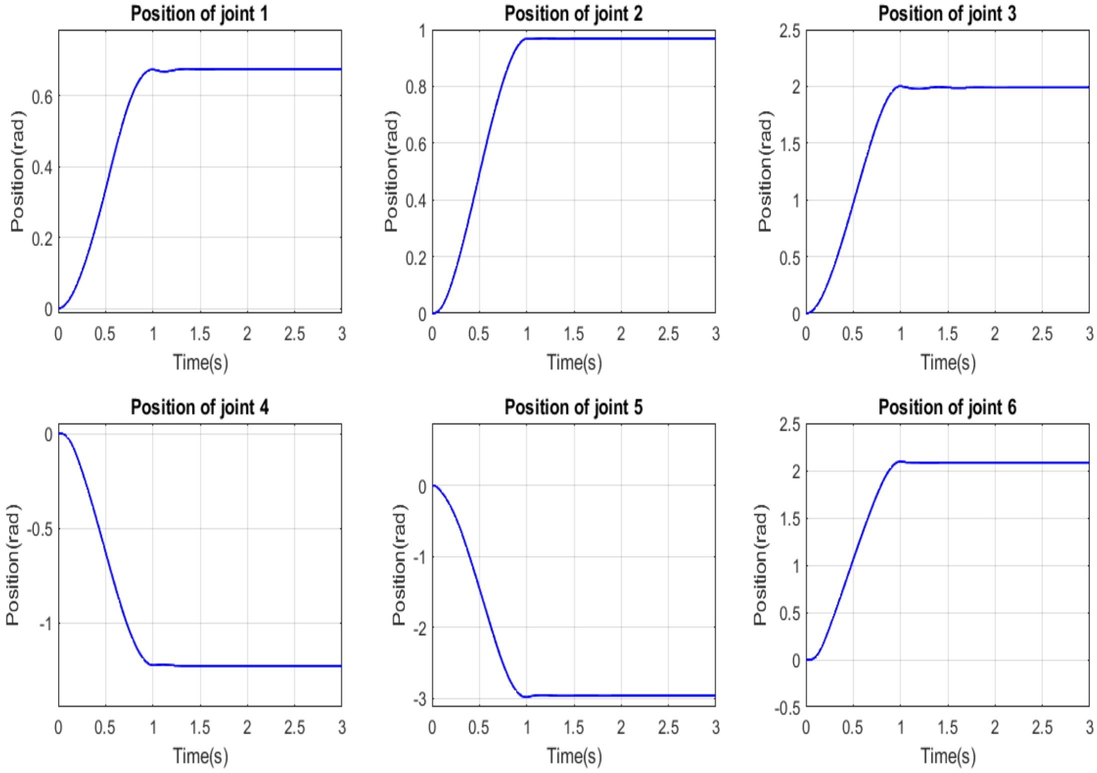


Figure 5.2: PID: joint positions.

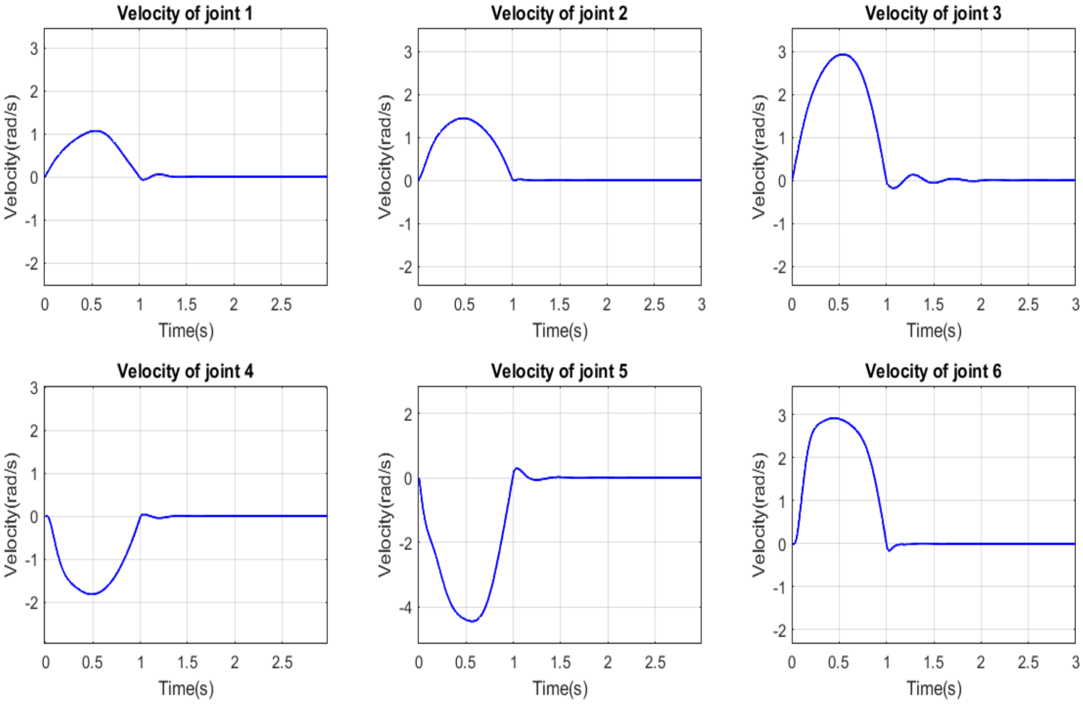


Figure 5.3: PID: joint velocities.

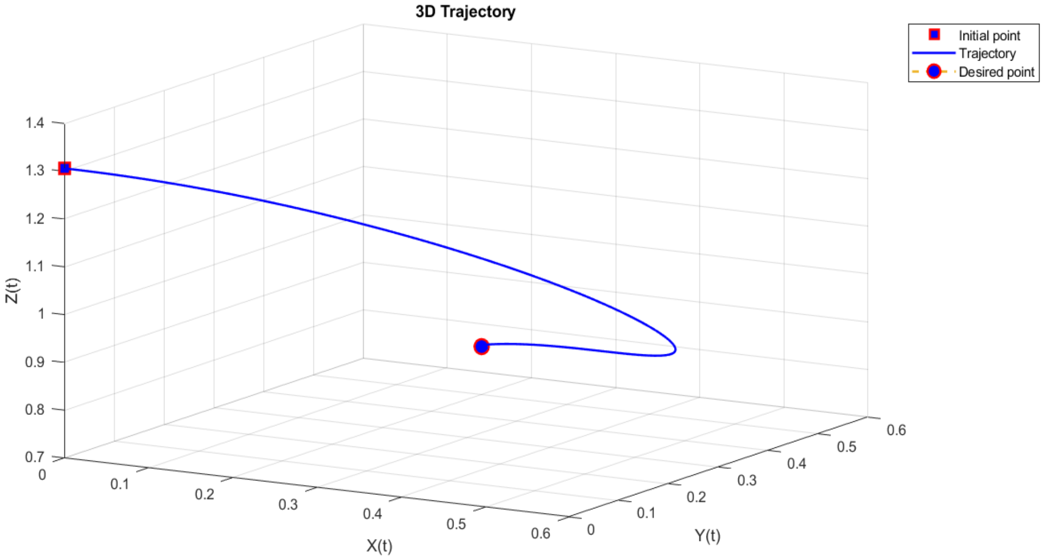


Figure 5.4: PID: 3D Trajectory.

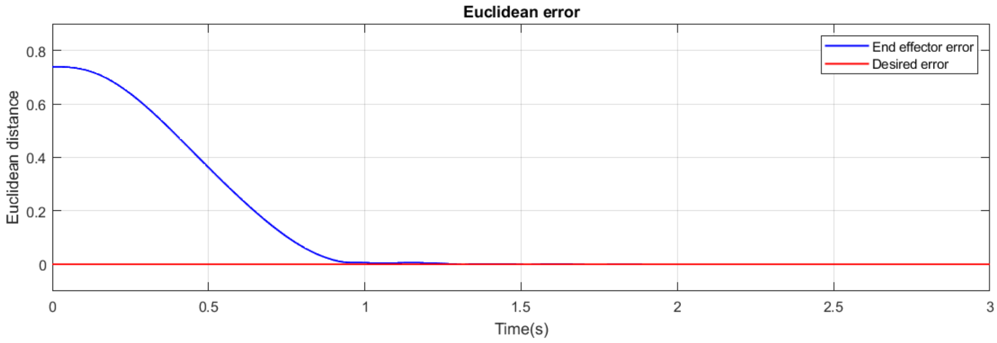


Figure 5.5: PID: Euclidean error.

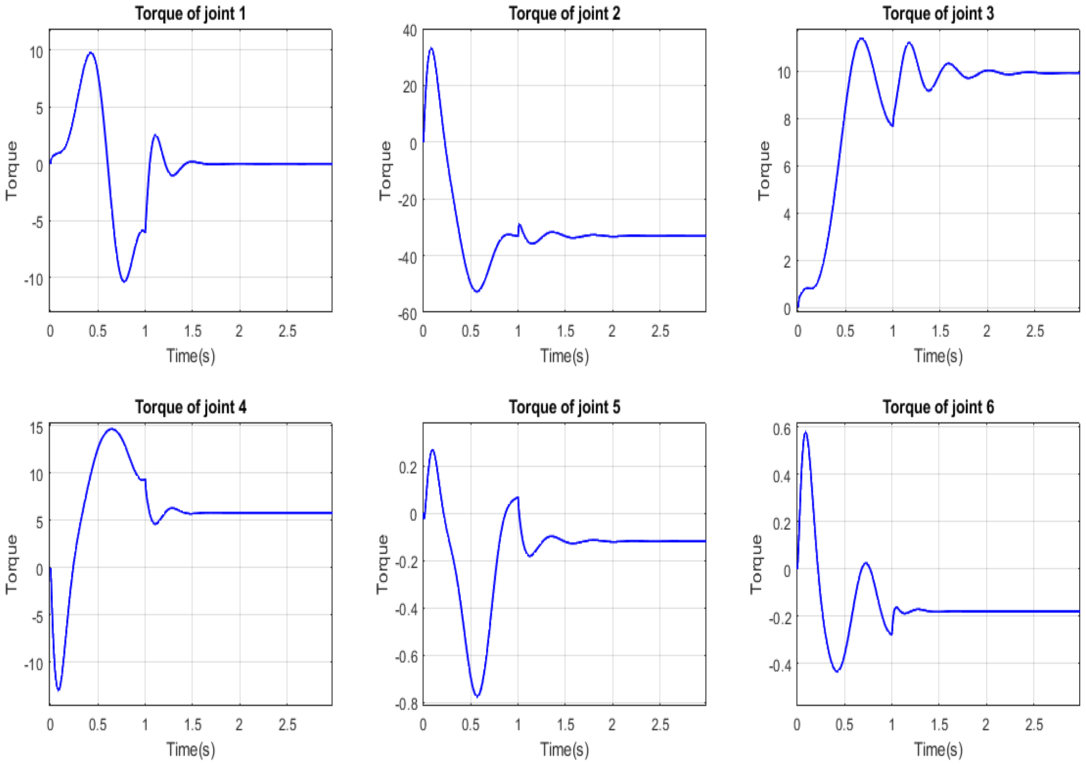


Figure 5.6: PID: torques (control signals).

5.3.2 Interpretation

Figure 5.1 shows that the end effector reaches the specified position. Its Cartesian coordinates x, y and z are tending to the references within 1.5s with overshooting and slight oscillations on x and y components due to the fact that it first adjusts its z component accurately to the desired one before the two others. Consequently, good precision is achieved as the Euclidean error tends to zero at around t=1.5s (figure 5.5), meaning that the integral action has accomplished its role. The joint positions depicted in figure 5.2 converge smoothly on the configuration that leads to the target position and represents the solution of the inverse kinematics problem with the corresponding velocities displayed in figure 5.3. The velocities have the form of a bi-quadratic polynomial that goes to 0 oscillating slightly once the target position is reached, maintaining the end-effector at it. Conditions on joint positions and velocities are thus respected. The previous figures along with 5.4 that represents the 3D trajectory indicate that the robot follows the twists without veering. It moves smoothly without any noticeable oscillations or erratic behavior which means that the closed-loop system is stable. As for the torques signals shown in figure 5.6, most of them vary frequently due to the fact that the linear control method tries to bring the non-linear system to the desired configuration and maintain it there. However, the torques do not exceed the value of 12 except for the one applied to the second joint that attains a value of 50, which is expected since it lifts the rest of the bodies. Hence, PID control does not require great effort to accomplish the task. It is important to remind that the friction torques and actuators' dynamics were not taken into account in modeling, which explains the control strategy's good performance.

5.4 Computed Torque Control

We wish to control the system so that the error follows the dynamics governed by:
$\Delta = \ddot{e} + 20\dot{e} + 100e$
with all $\xi_j = 1$ and $\omega_j = 10$ rad/s, which implies that $K_{vj} = 20$ and $K_{pj} = 100$.
For a choice of $K_v = \text{diag}\{20\}$ and $K_p = \text{diag}\{100\}$, we obtain the following results.

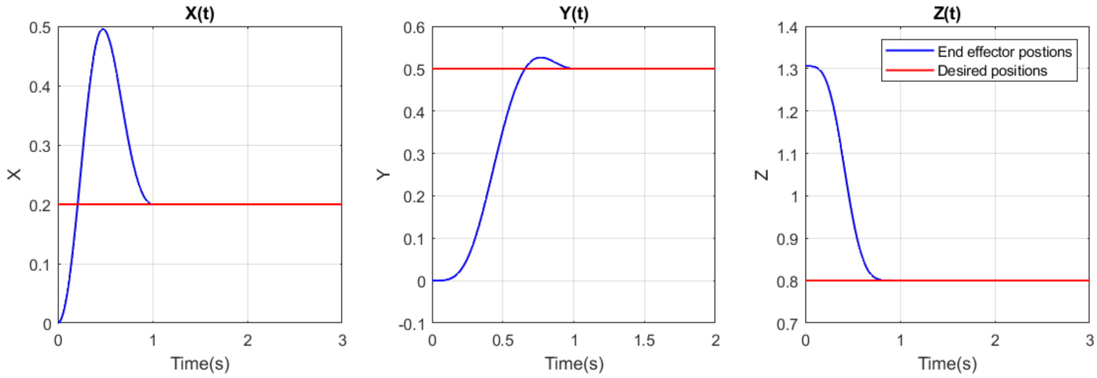


Figure 5.7: CTC: End effector's position with respect to the desired position.

5.4.1 Results

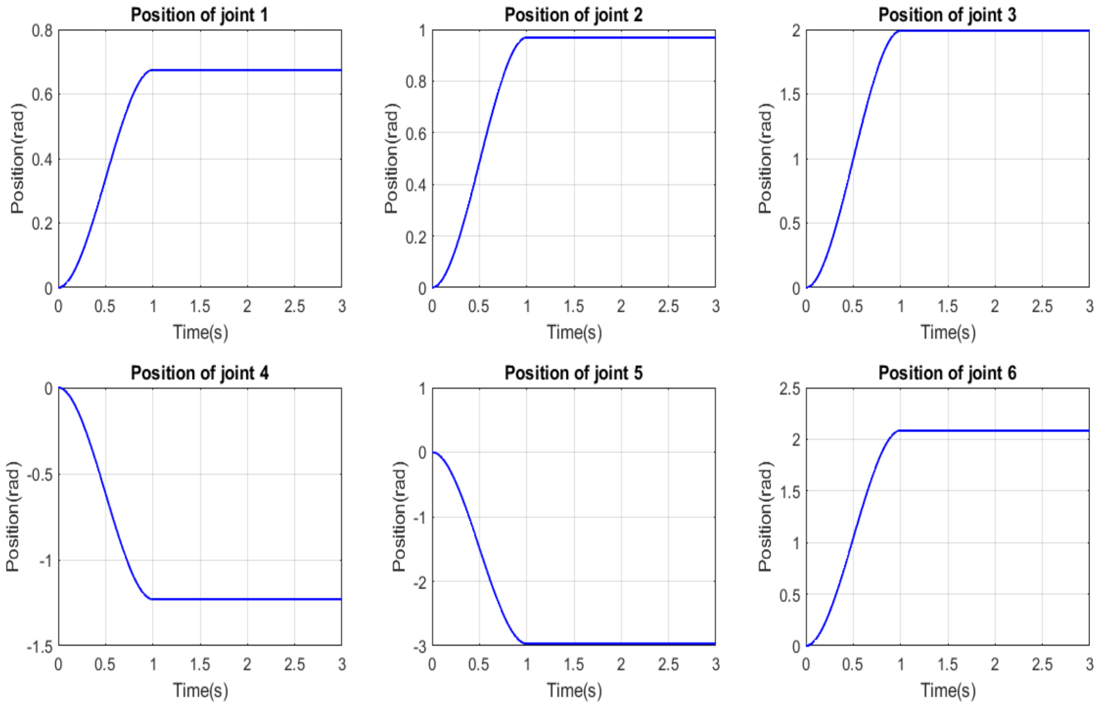


Figure 5.8: CTC: joint positions.

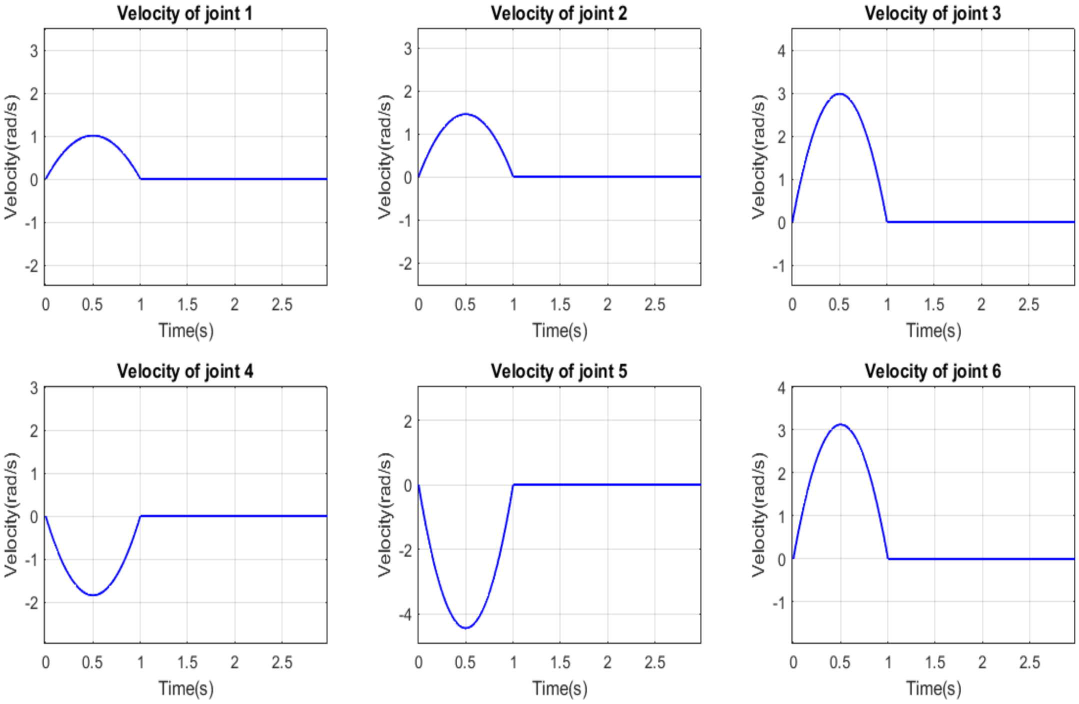


Figure 5.9: CTC: joint velocities.

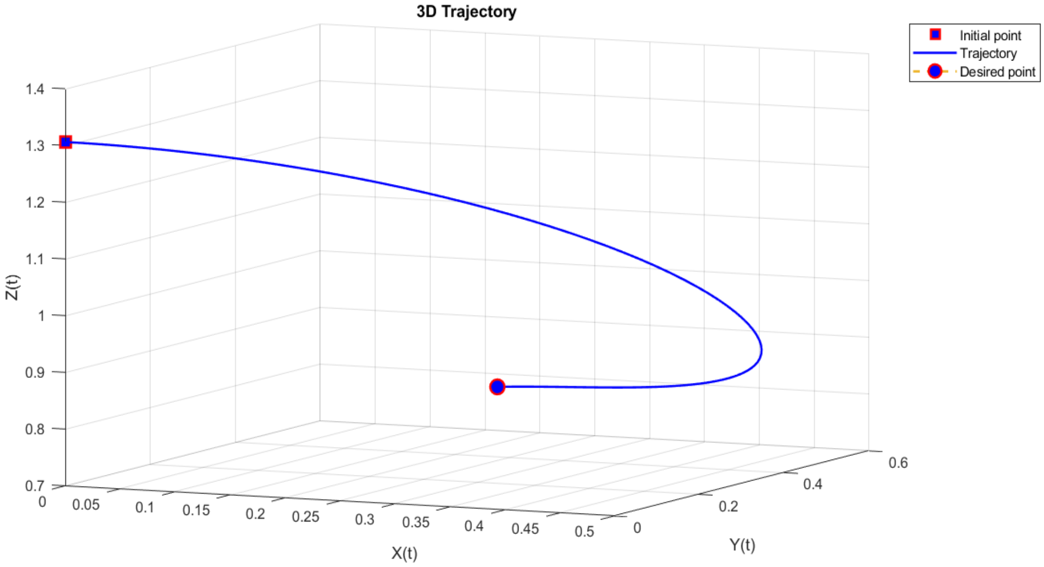


Figure 5.10: CTC: 3D Trajectory.

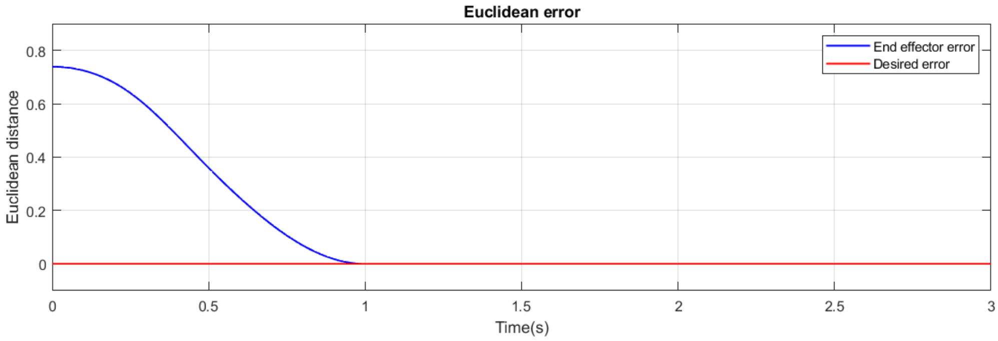


Figure 5.11: CTC: Euclidean error.

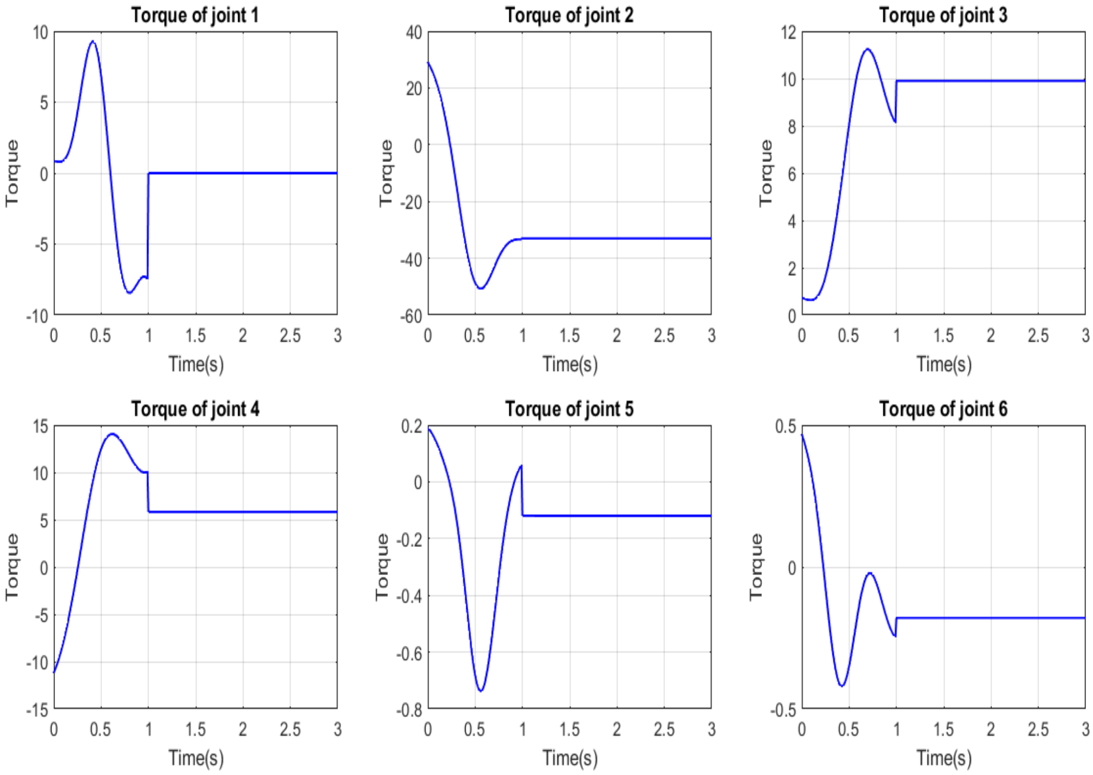


Figure 5.12: CTC: torques (control signals).

5.4.2 Interpretation

Figure 5.7 demonstrates that the end effector successfully reaches the desired position. The Cartesian coordinates x, y, and z converge to their respective references within 1s. There are overshoots observed in the x and y components, attributed to the initial adjustment of the z component to match the desired value before fine-tuning the other two. As a result, this control method achieves high precision, as indicated by the Euclidean error approaching zero at approximately t=1s in figure 5.11 which is faster than the PID control convergence. In figure 5.8, the joint positions smoothly converge towards the configuration that leads to the target position, representing the solution of the inverse kinematics problem. The corresponding velocities shown in figure 5.9, follow a bi-quadratic polynomial pattern that gradually diminishes to zero without oscillating once the target position is reached. This ensures that the end-effector remains at the desired position, while satisfying the conditions on joint positions and velocities. The aforementioned figures with figure 5.10 illustrating the 3D trajectory collectively demonstrate the robot's accurate tracking of the desired path without deviation that is slightly different from the PID's one. The robot moves seamlessly, devoid of oscillations or irregular behavior, indicating the stability of the closed-loop system. Examining the torque signals depicted in figure 5.12, it is notable like in the PID control case, the majority of the signals remain below a value of 12, except for the torque applied to the second joint, which reaches 50. Since it lifts the remaining components of the system, this result is anticipated. Consequently, the CTC control mechanism performs the task more efficiently than PID without requiring excessive effort or fluctuation as it cancels out the non-linearities and commands the resulting system using a linear control (PD). Note that the model is considered perfect because friction torques and actuators dynamics were neglected in the modeling process.

5.5 Sliding Mode Control

For a choice of $\lambda = \text{diag}\{400\}$ and $K = \text{diag}\{400\}$ and by replacing the sign function with tanh function, We obtain the following results.

5.5.1 Results

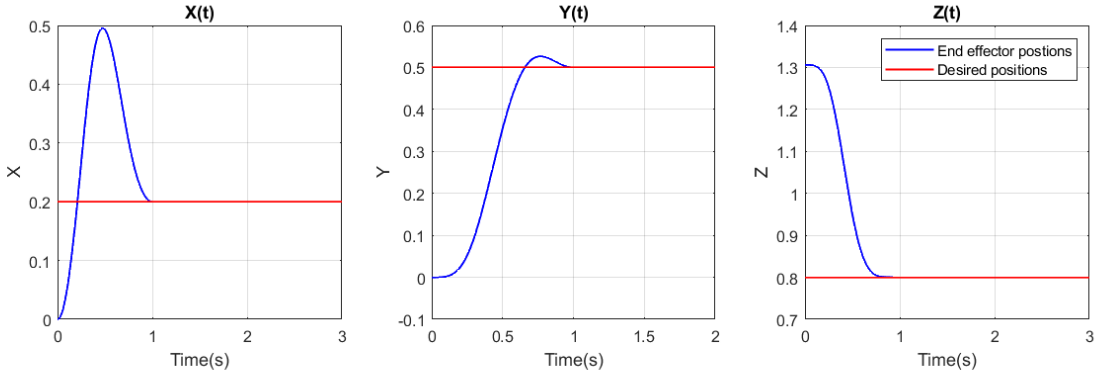


Figure 5.13: SMC: End effector's position with respect to the desired position.

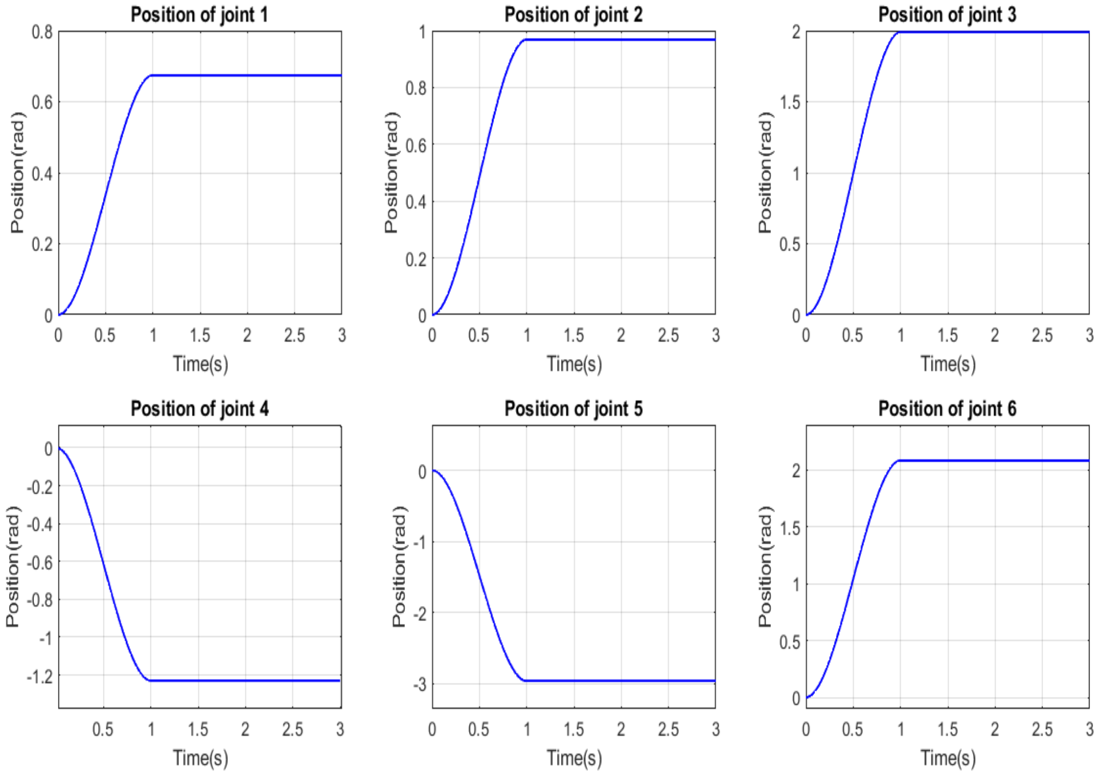


Figure 5.14: SMC: joint positions.

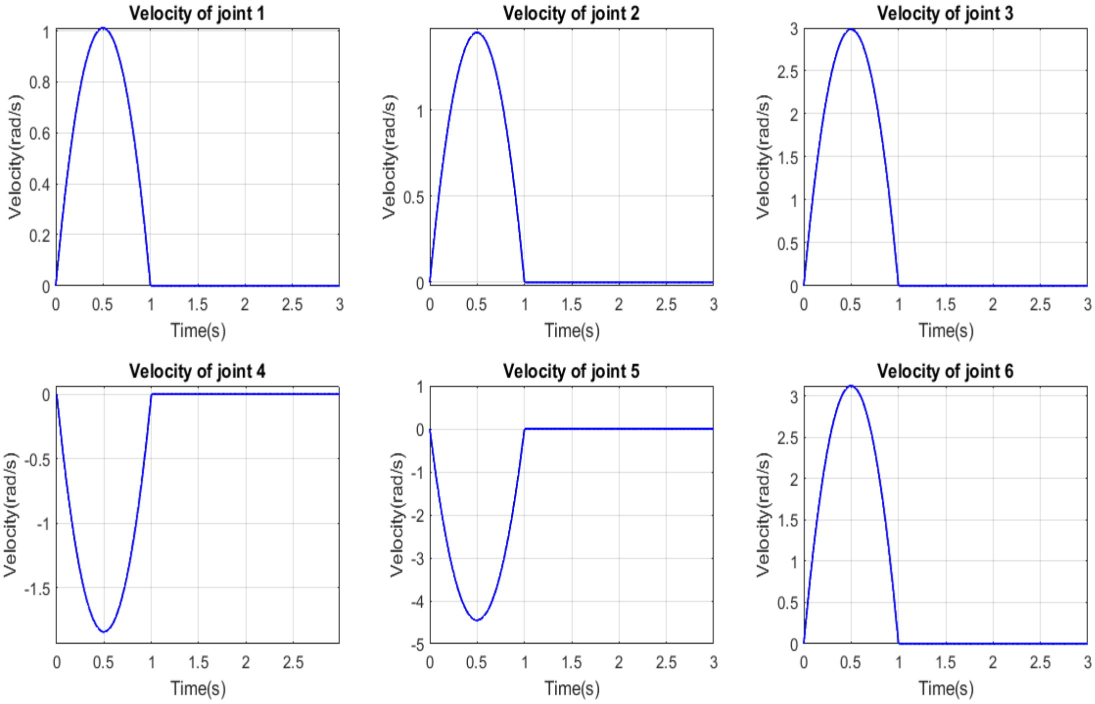


Figure 5.15: SMC: joint velocities.

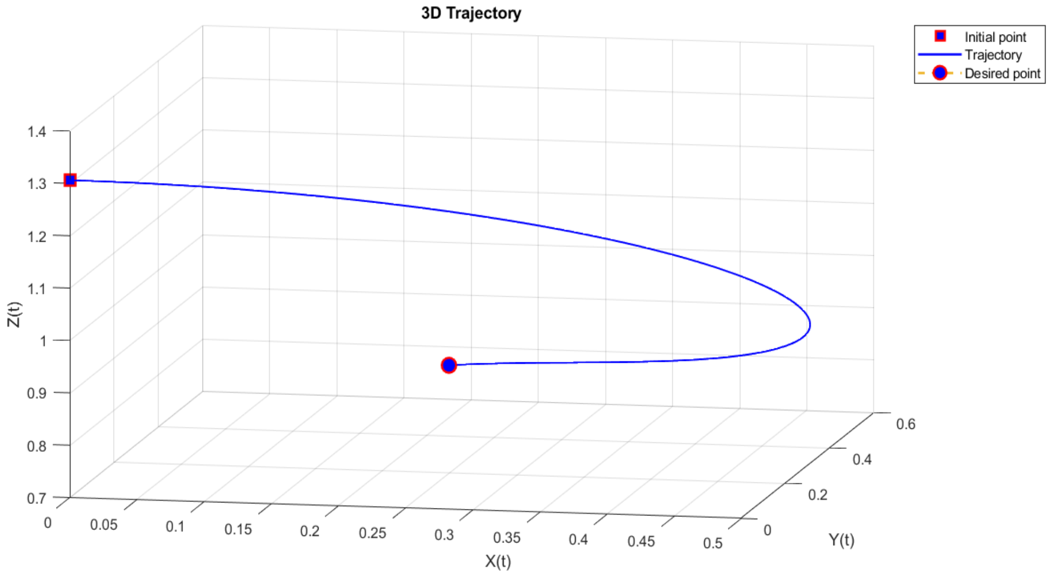


Figure 5.16: SMC: 3D Trajectory.

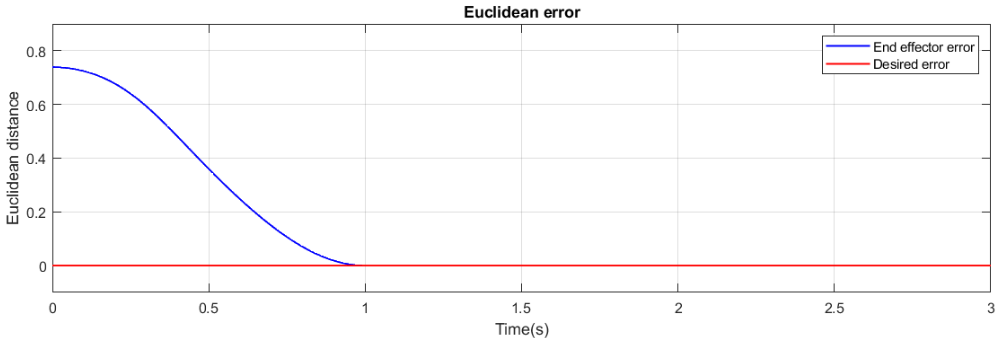


Figure 5.17: SMC: Euclidean error.

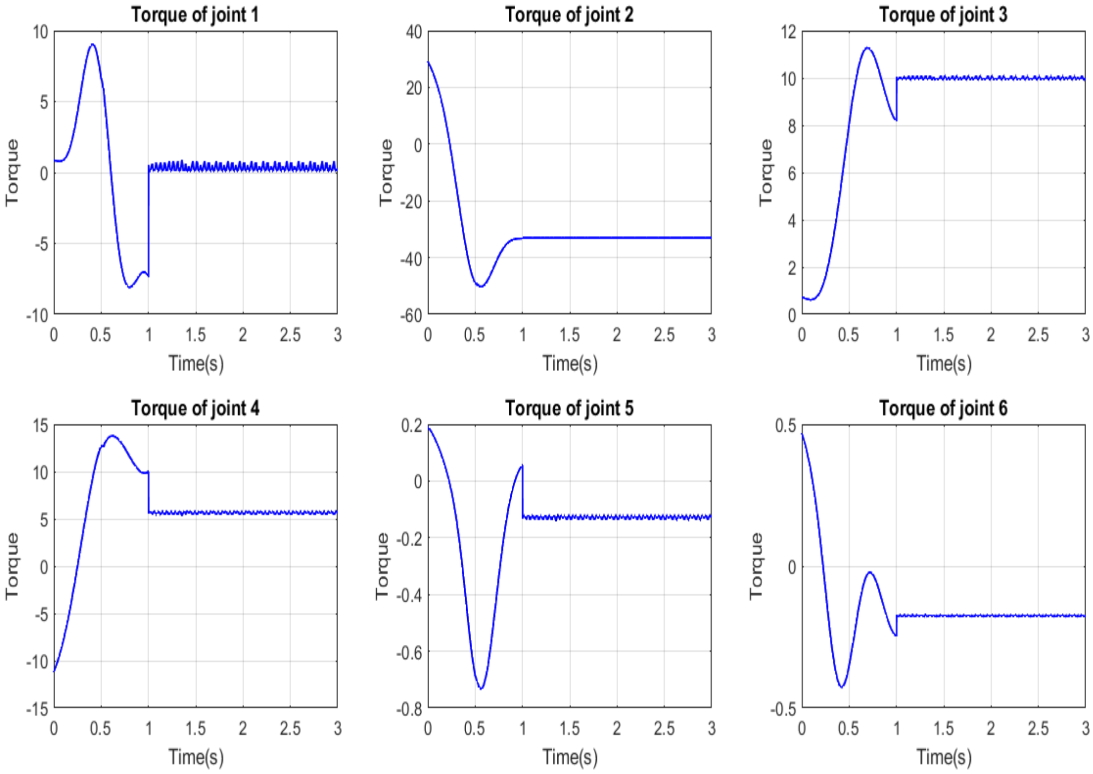


Figure 5.18: SMC: torques (control signals).

5.5.2 Interpretation

Figure 5.13 provides evidence of the successful achievement of the desired position by the end effector. Within a time frame of 1s, the Cartesian coordinates x, y, and z converge toward their respective references. Notably, there are overshoots in the x and y components, which results from the initial adjustment of the z component to align with the desired value before adjusting the other two. This control method exhibits remarkable precision, as depicted by the Euclidean error approaching zero at around t=1s in figure 5.17, which is as fast as the convergence achieved by the CTC approach. In figure 5.14, the joint positions smoothly converge towards the configuration necessary to reach the target position, corresponding to the solution of the inverse kinematics problem. The corresponding velocities, shown in figure 5.15, follow a bi-quadratic polynomial pattern that gradually decreases to zero without oscillations once the target position is attained. This ensures that the end effector remains precisely at the desired position while satisfying the prescribed conditions on joint positions and velocities. The combination of the previous figures, along with figure 5.16 illustrating the 3D trajectory, provides compelling evidence of the robot's accurate tracking of the desired path without deviations. It moves seamlessly, devoid of oscillations or irregular behavior, signifying the stability of the closed-loop system. By analyzing the torque signals depicted in figure 5.18, similar to the PID and CTC cases, the majority of the signals remain below a value of 12, except for the torque exerted on the second joint, reaching 50. This outcome is expected since it lifts other components of the system. Despite the substitution of the sign function with a hyperbolic tangent function, small and high-frequency oscillation may be observed because the SMC drives the state's trajectory on the sliding surface and confines it to its vicinity. This was predicted as the substitution does not eliminate the chattering but it reduces it. Hence, the SMC control mechanism exhibits remarkable performances in terms of efficiency and stability. Friction torques and actuator dynamics being neglected in the modeling, the robot's model is assumed to be perfect which explains such good results.

5.6 Deep Reinforcement Learning

5.6.1 Trajectory optimization

This experience aims to generate and optimize the trajectory that the robot driven by computed torque control should follow from the initial position until it reaches the target one, using the DDPG Algorithm to train a NN which receives 20 inputs (joints positions and velocities, current end-effector's position and the desired one) and whose 21 outputs (joint positions, velocities and accelerations) are applied to the environment.

5.6.1.1 Experience details

- **Environment:** The environment consists of the computed torque controller and the robot system in Simulink. Despite the existence of 21 inputs and 21 outputs in this environment, only the joints' positions and velocities are taken into account as speed limits are set in this training.

- **Neural network architecture:** All 7 positions and velocities of the robot's joints along with the end effector's position and the target one in the Cartesian space are taken as inputs for the actor NN and the state NN of the critic.
  - **Critic Network:**
    1. **States NN:** it comprises:
       * Normalization input layer with 20 inputs that will fall between -1 and 1.
       * Fully connected layer with 200 neurons and ReLU activation function applied to each neuron's output.
       * Fully connected layer with 200 neurons without an activation function.
    2. **Actions NN:** it includes:
       * Normalization input layer with 7 inputs (actions).
       * Fully connected layer with 200 neurons without an activation function.
    3. **Additional neural network:** it is made up of:
       * Additional layer that combines the outputs of the previous NNs. It has 200 outputs to which a ReLU activation function is applied.

  - ∗ Fully connected layer with 1 neuron outputting Action Values which will be utilized for the actor optimization with back-propagation.

- – **Actor Network:** it involves:
  - ∗ Normalization input layer with 20 inputs.
  - ∗ Fully connected layer with 200 neurons and a ReLU activation function applied to each neuron's output.
  - ∗ Fully connected layer with 200 neurons and a ReLU activation function applied to each neuron's output.
  - ∗ Fully connected layer with 21 neurons and Tanh layer.
  - ∗ Scaling layer between -3.14 and 3.14.

- **Hyper-parameters:**
  - – Discount Factor = 0.99
  - – Mini Batch Size = 64
  - – Experience Buffer memory = 10^6
  - – Target smooth factor = 10^{-3}
    This factor is used for the updating of the target network in a smooth way to ensure training stability: $\theta_{target} = (1-\tau)\theta_{targetold} + \tau\theta_{main}$
    where τ is the target smooth factor and θ_{target} is the target weights for the actor and the critic,
  - – Noise Standard Deviation = $2\sqrt{Ts}$.
  - – Noise Standard Deviation Decay Rate = 10^{-3}.
    The noise will decay on each episode with this value. In each episode, we have $NewStd = OldStd.(1-10^{-3})$.
  - – Learning Rate for the critic network = 10^{-3}.
  - – Learning rate for Actor network = 10^{-4}.

- **Training parameters:**
  - – Max episodes = 10000
  - – Steps on each episode = 500
  - – Score average window reward = 100
  - – Ts = 0.01 (100 steps in one second)
  - – Te = 5s (each episode)

- **Reward function:**

$$\text{reward} = -2\cdot(\text{error\_euc})^2 - 0.1\cdot\left(\frac{\text{sum of (torque\_signal)}}{7}\right) - (\text{if velocity of any joint} > 30) \tag{5.1}$$

The reward's expression indicates that the training focuses on increasing precision and minimizing efforts.

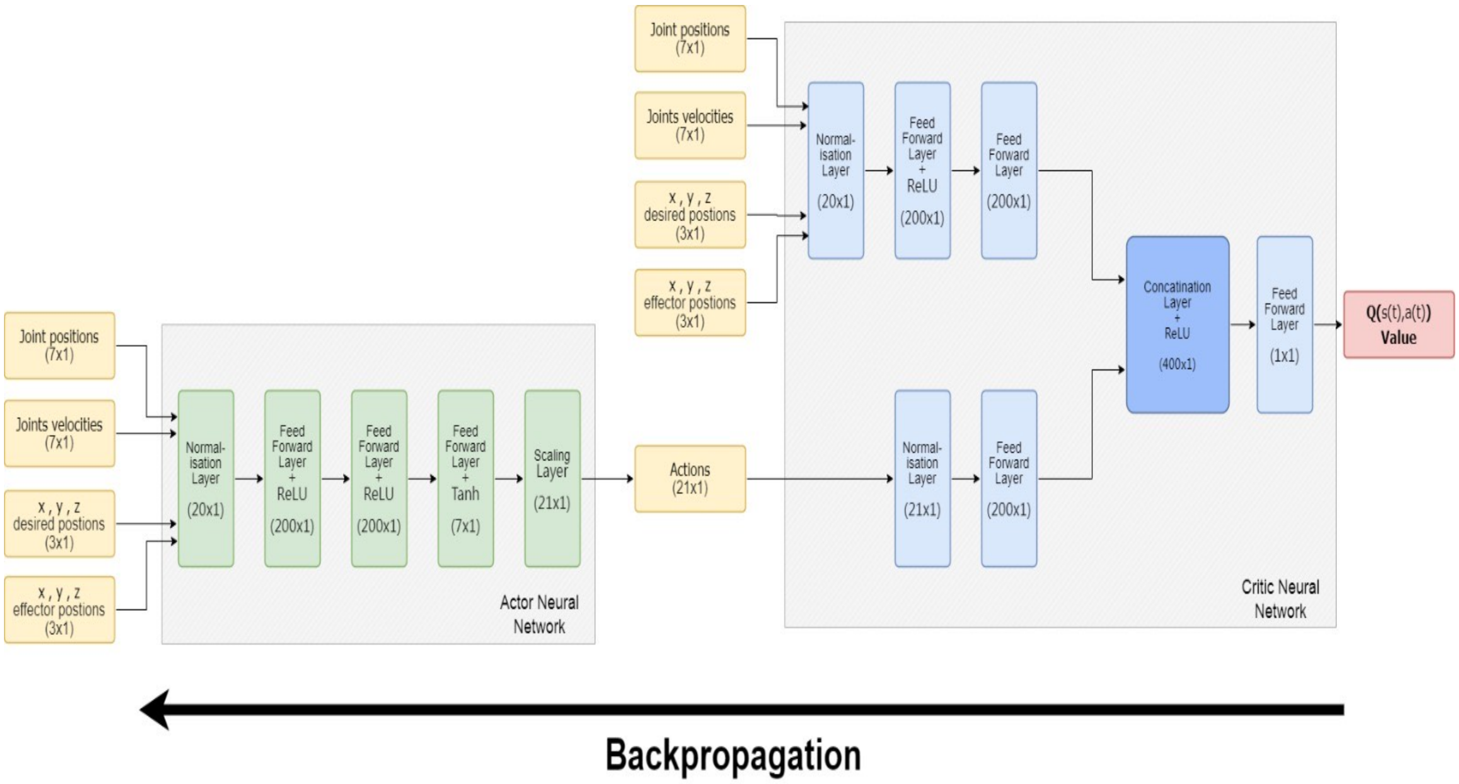

Joint positions (7x1)
Joints velocities (7x1)
x , y , z desired postions (3x1)
x , y , z effector postions (3x1)
Normal-isation Layer (20x1)
Feed Forward Layer + ReLU (200x1)
Feed Forward Layer (200x1)
Concatination Layer + ReLU (400x1)
Feed Forward Layer (1x1)
Q(s(t),a(t)) Value
Actions (21x1)
Normal-isation Layer (21x1)
Feed Forward Layer (200x1)
Critic Neural Network
Joint positions (7x1)
Joints velocities (7x1)
x , y , z desired postions (3x1)
x , y , z effector postions (3x1)
Normal-isation Layer (20x1)
Feed Forward Layer + ReLU (200x1)
Feed Forward Layer + ReLU (200x1)
Feed Forward Layer + Tanh (7x1)
Scaling Layer (21x1)
Actor Neural Network
Backpropagation

Figure 5.19: Trajectory optimization: Actor's and critic's architecture.

5.6.1.2 Training

When the training starts, the robot's actions are taken randomly due to the random initialization of the neural networks' weights and the application of noise to the action selection process.

For each episode, 500 tuples will be stored and utilized to train the NN to select the best actions -that yield the greatest reward- using the DDPG Algorithm which optimizes the actor and critic networks. The updating is done after each step from the beginning of the training by taking 64 tuples from the memory. The Target network will be updated using the target smooth equation. The Actor network is thus optimized with this loop to choose the actions that generate the best values (critic's output), which leads to the optimization of the system in order to obtain good results, depending on its complexity, the training's time and quality, the number of episodes and other factors. The training in this experience lasted two and a half days.

5.6.1.3 Results

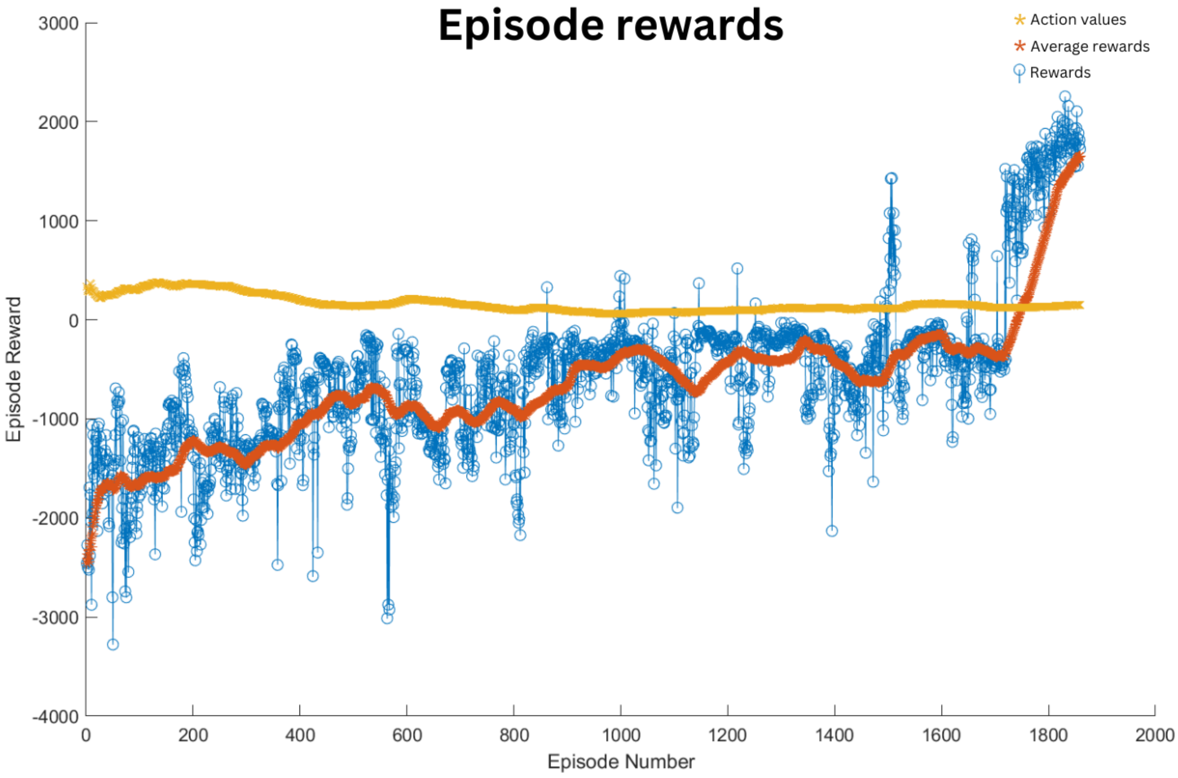


Figure 5.20: Trajectory optimization: Reward.

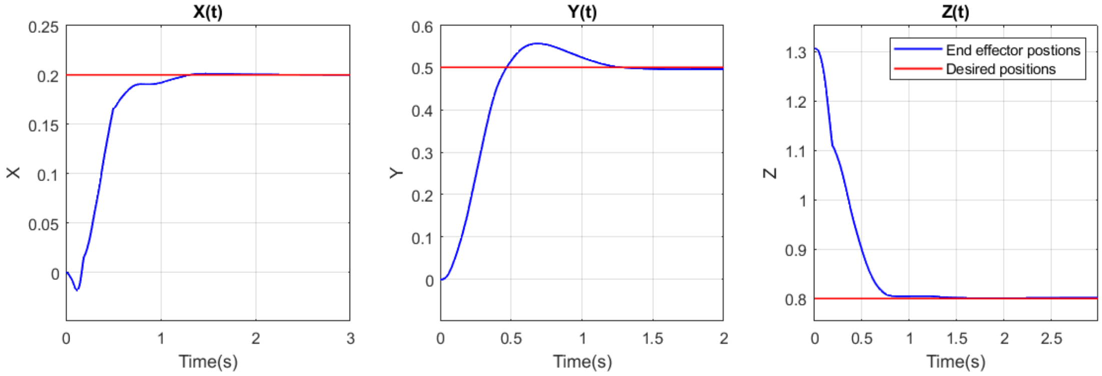


Figure 5.21: Trajectory optimization: End effector's position with respect to the desired position.

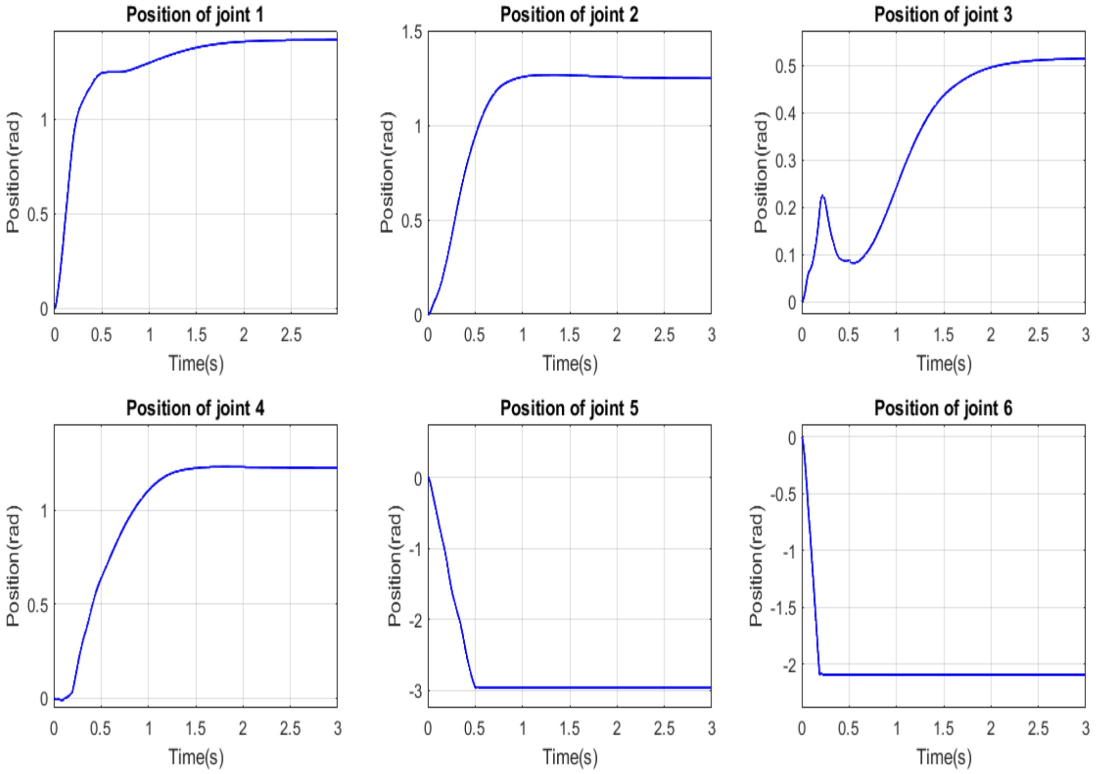


Figure 5.22: Trajectory optimization: joint positions.

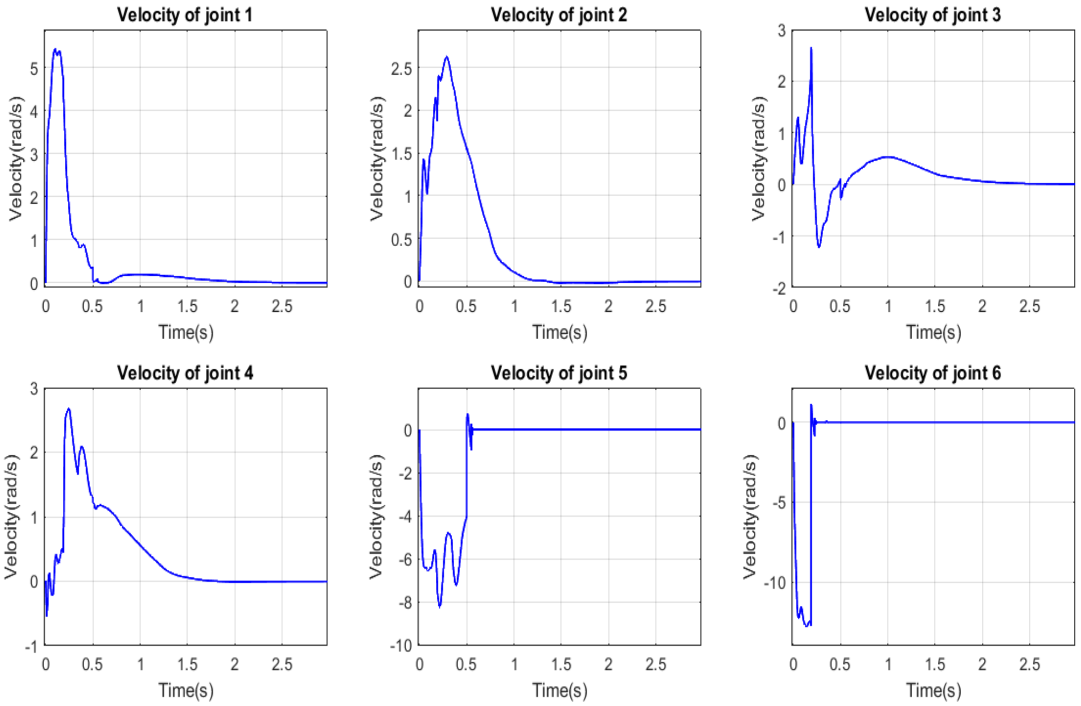


Figure 5.23: Trajectory optimization: joint velocities.

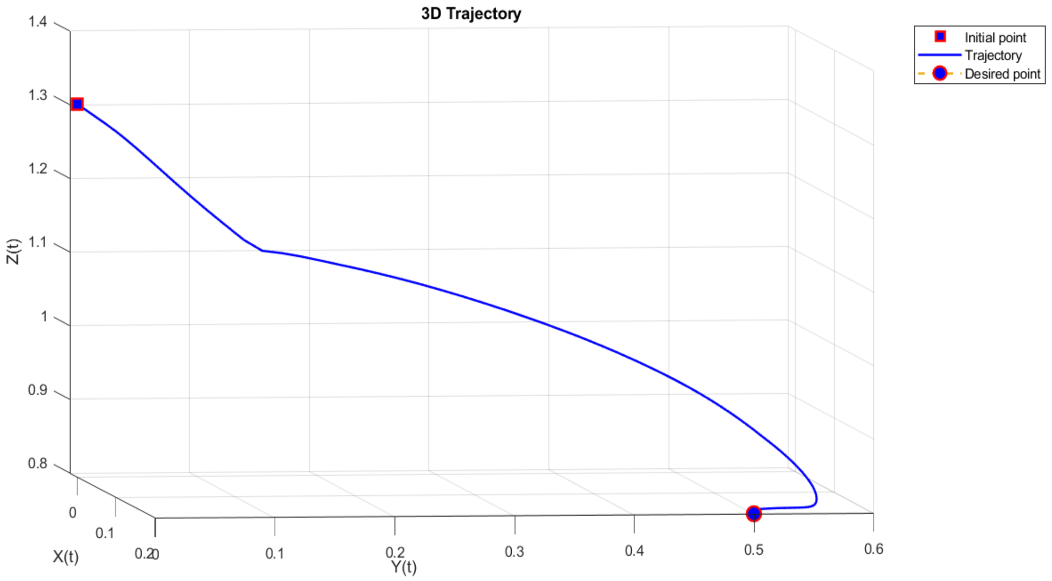


Figure 5.24: Trajectory optimization: 3D Trajectory.

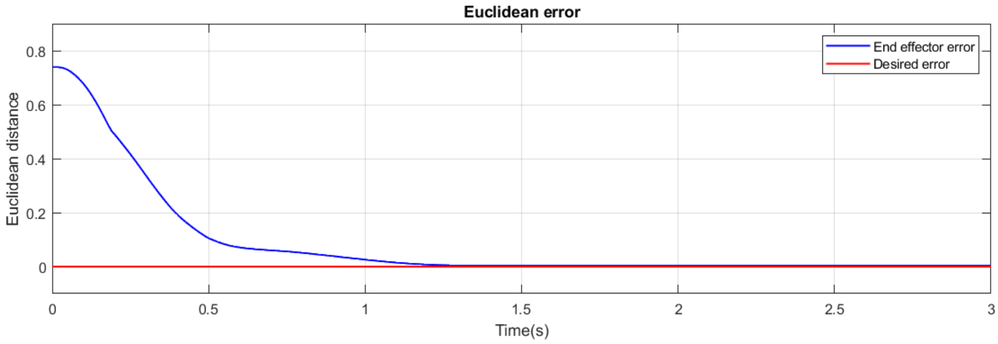


Figure 5.25: trajectory optimization: Euclidean error.

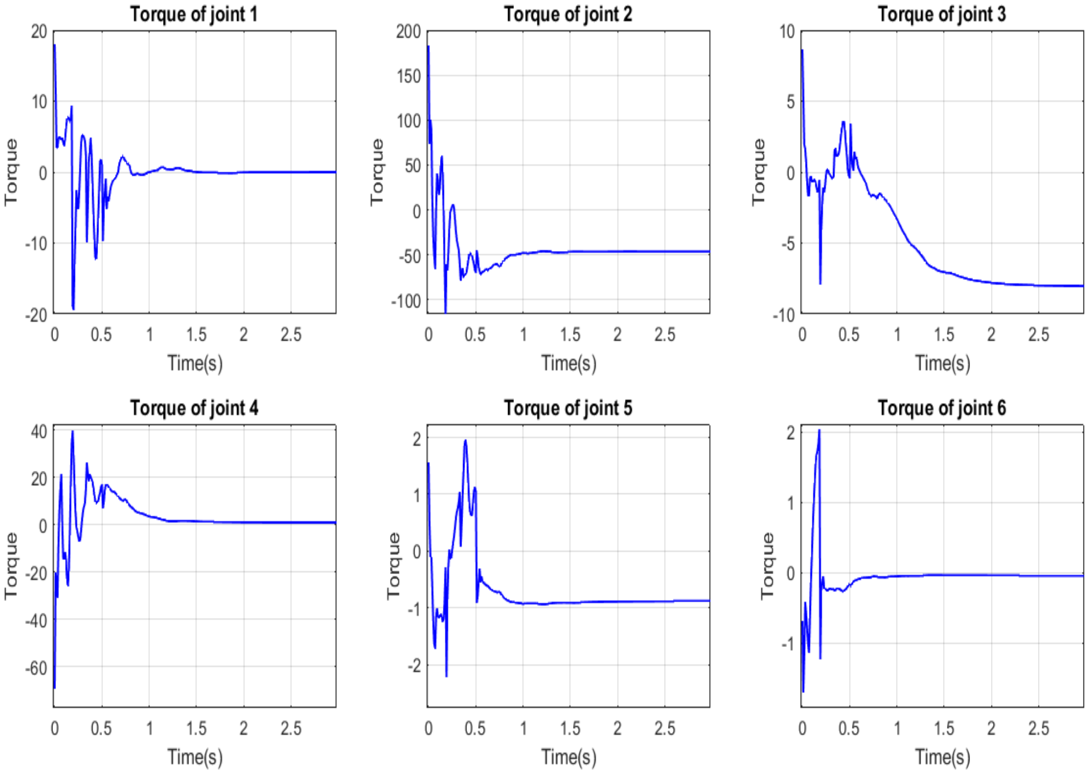


Figure 5.26: Trajectory optimization: torques (control signals).

5.6.1.4 Interpretation

Figure 5.20 displaying the progression of the cumulative reward over time provides insights into the learning process. It depicts both the episode reward and the average reward (calculated over 100 episodes). Initially, during the early stages of training, the system (consisting of the robot and the CTC) performs poorly in reaching the target point. This is expected as the agent possesses no prior knowledge of the environment, resulting in a reward of -2000 for the initial actions taken. However, as the agent continues its training cycle, the weights are adjusted, and the cumulative reward gradually increases. Eventually, it reaches a maximum value of 2000 after approximately 1900 episodes, indicating that the agent has discovered solutions to the optimization problem and selected the optimal one. In other words, the agent has learned, acquired new skills, and successfully achieved the objective. It is important to note that fluctuations and instability in the cumulative reward are observed due to the updating of neural networks (NNs) and the application of noise, along with the emergence of new states that the agent discovers. The duration of training is relatively long because the agent engages in a balance between exploration and exploitation to effectively learn and improve its policy. The use of limited training resources justifies the extended learning process.

Examining the data in the figures 5.21, 5.22, 5.23 and 5.25, it is evident that after training, the robot achieves precise goal attainment. The error is significantly reduced to 5mm, and the joint positions and velocities adjust to the configuration necessary to bring the end-effector's position as close as possible to the target position in less than 1.5s. Figure 5.26 demonstrates how the agent is trained to optimize the reward function, aiming for efficiency: Most of the torques remain below 100 (except for the torque applied on joint 2 which requires greater effort), and some converge to zero. This indicates that the agent learns to minimize energy consumption effectively. Under the specific training conditions mentioned above, the agent has learned the best trajectory, as depicted in figure 5.24. It is important to highlight that with further extended training and modifications to the reward function, the trajectory can be further refined and improved.

The attained outcomes are very good, particularly when taking into account the limitations imposed by the scarcity of training resources, the constrained training duration, and the agent's absence of prior knowledge regarding the environment.

5.6.2 Control and trajectory optimization

The experiment's purpose is to control the robotic arm on the trajectory chosen by the agent. The objective is achieved by training a NN receiving the same inputs as the previous experience and outputs 7 signals representing the torques.

5.6.2.1 Experience details

The experience details are almost similar to the previous one except for a few changes.

- **Environment:** The robotic arm constitutes the environment in this case.
- **NN architecture:**
  1. For the critic, the architecture and the learning rate remain unchanged.
  2. For the actor, the same architecture and learning are considered. however, the number of outputs was reduced from 21 to 7 outputs.
- **Hyper-parameters:**
  - Discount Factor = 0.98.
  - Mini Batch Size = 64.
  - Experience Buffer memory = 10^6.
  - Target smooth factor = 10^{-3}.
  - Noise Standard Deviation = $2\sqrt(Ts)$.
  - Noise Standard Deviation Decay Rate = 10^{-5}.
    Note that in this experience, a very small decay rate for the noise was chosen for more exploration since the environment has become more complex compared to the previous experience.
- **Training parameters:**
  - Max episodes = 10000
  - Steps on each episode = 200
  - Score average window reward = 100
  - Ts = $\frac{1}{40} = 0.025$
  - Te = 5s
- **Reward function:** It is identical to the previous experience except for the condition on the velocities which was omitted. The same objective must be achieved.

$$\text{reward} = -2 \cdot (\text{error\_euc})^2 - 0.1 \cdot \left(\frac{\text{sum of (torque\_signal)}}{7}\right) \tag{5.2}$$

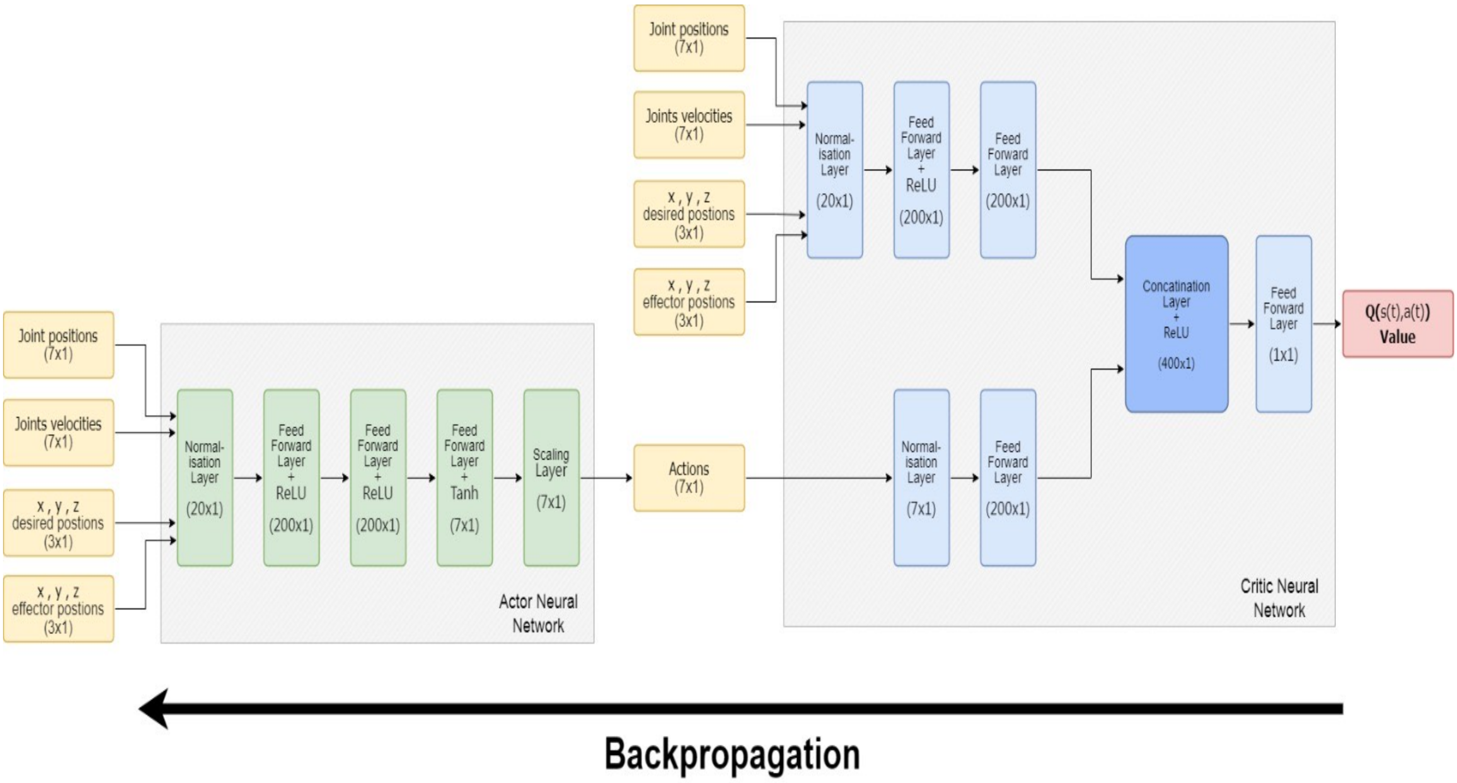

Joint positions (7x1)
Joints velocities (7x1)
x , y , z desired postions (3x1)
x , y , z effector postions (3x1)
Normal-isation Layer (20x1)
Feed Forward Layer + ReLU (200x1)
Feed Forward Layer + ReLU (200x1)
Feed Forward Layer + Tanh (7x1)
Scaling Layer (7x1)
Actor Neural Network
Actions (7x1)
Joint positions (7x1)
Joints velocities (7x1)
x , y , z desired postions (3x1)
x , y , z effector postions (3x1)
Normal-isation Layer (20x1)
Feed Forward Layer + ReLU (200x1)
Feed Forward Layer (200x1)
Normal-isation Layer (7x1)
Feed Forward Layer (200x1)
Concatination Layer + ReLU (400x1)
Feed Forward Layer (1x1)
Q(s(t),a(t)) Value
Critic Neural Network
Backpropagation

Figure 5.27: Control and trajectory optimization: Actor's and critic's architecture.

Note that the experiment's training lasted four days.

5.6.2.2 Results

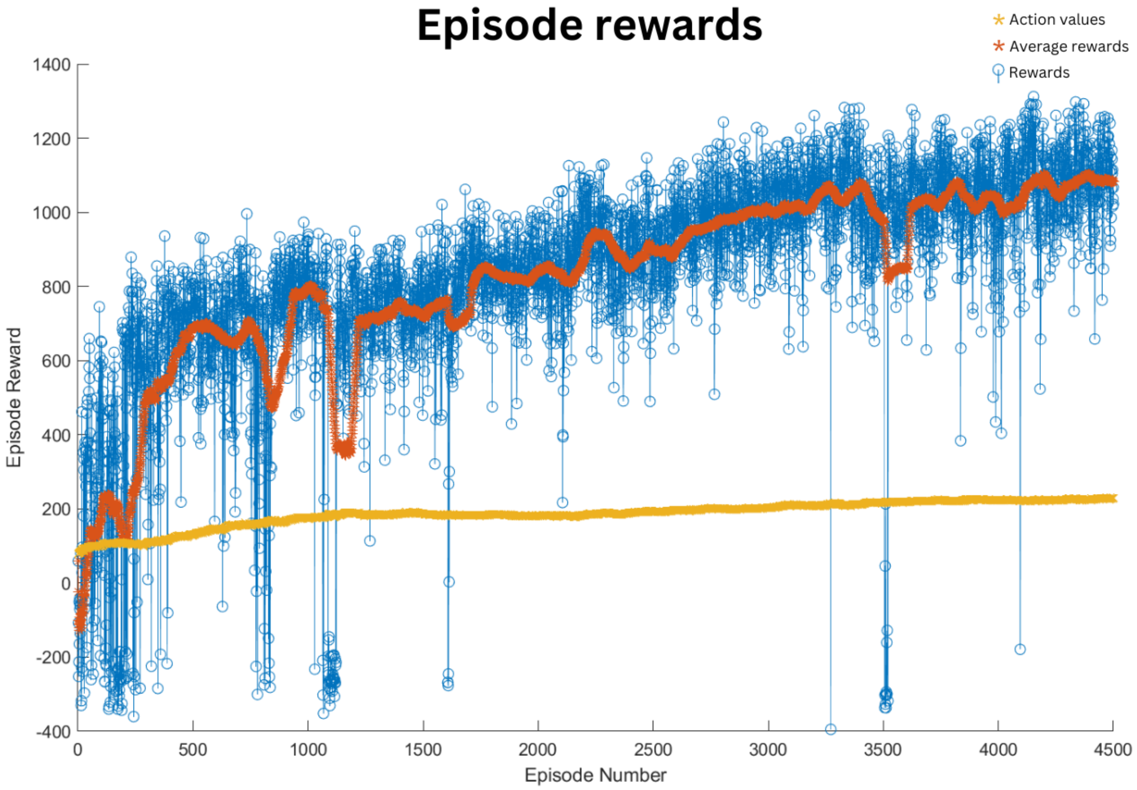


Figure 5.28: Control and trajectory optimization: Reward.

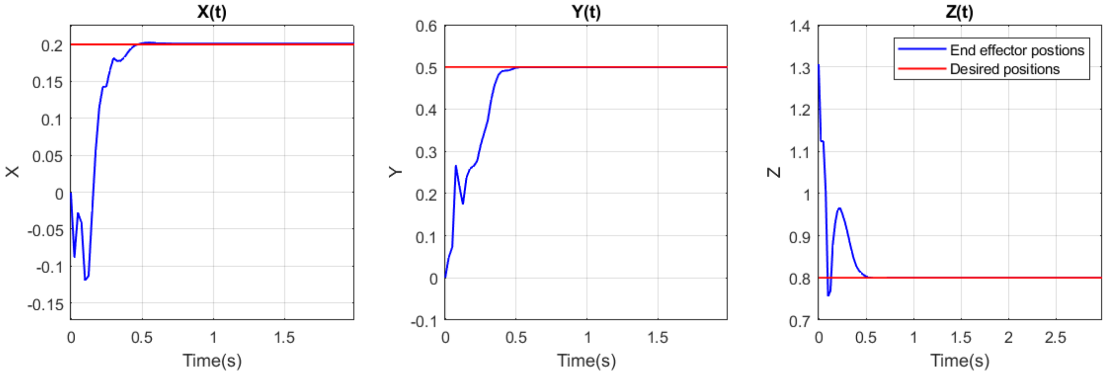


Figure 5.29: Control and trajectory optimization: End effector's position with respect to the desired position.

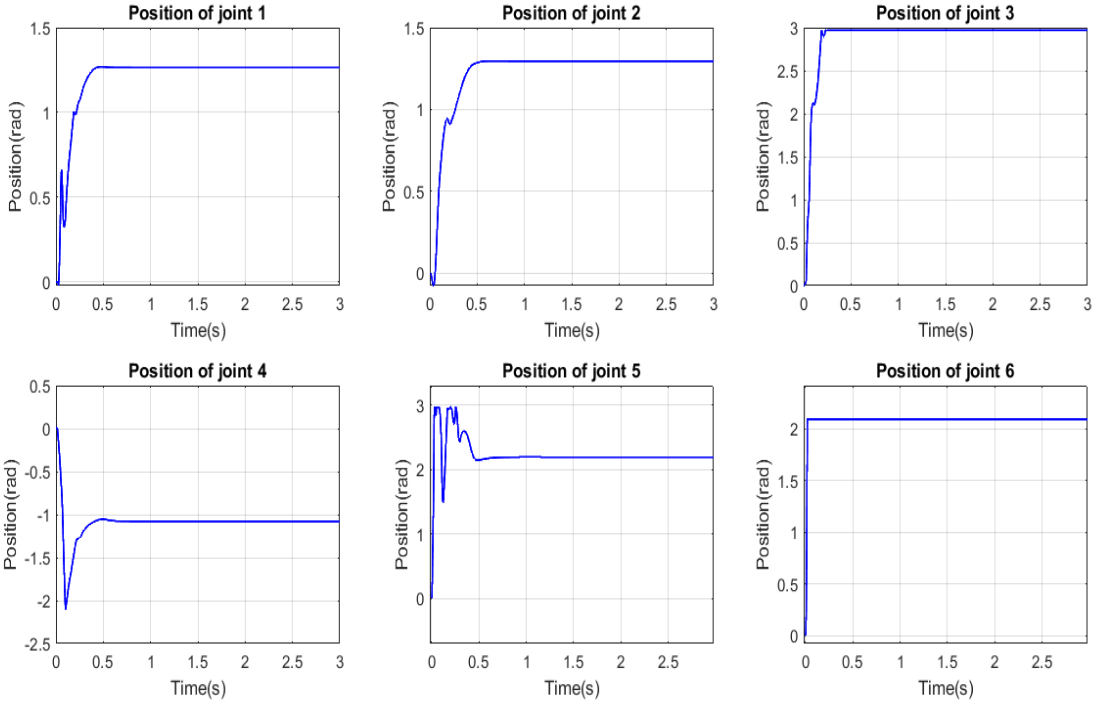


Figure 5.30: Control and trajectory optimization: joint positions.

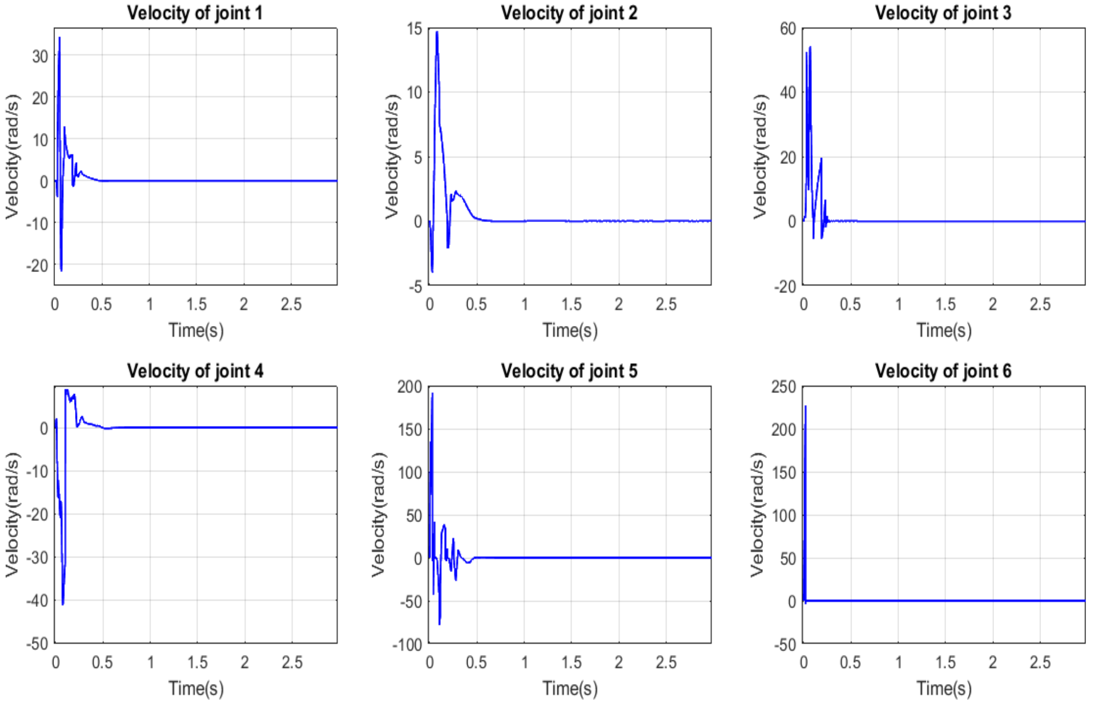


Figure 5.31: Control and trajectory optimization: joint velocities.

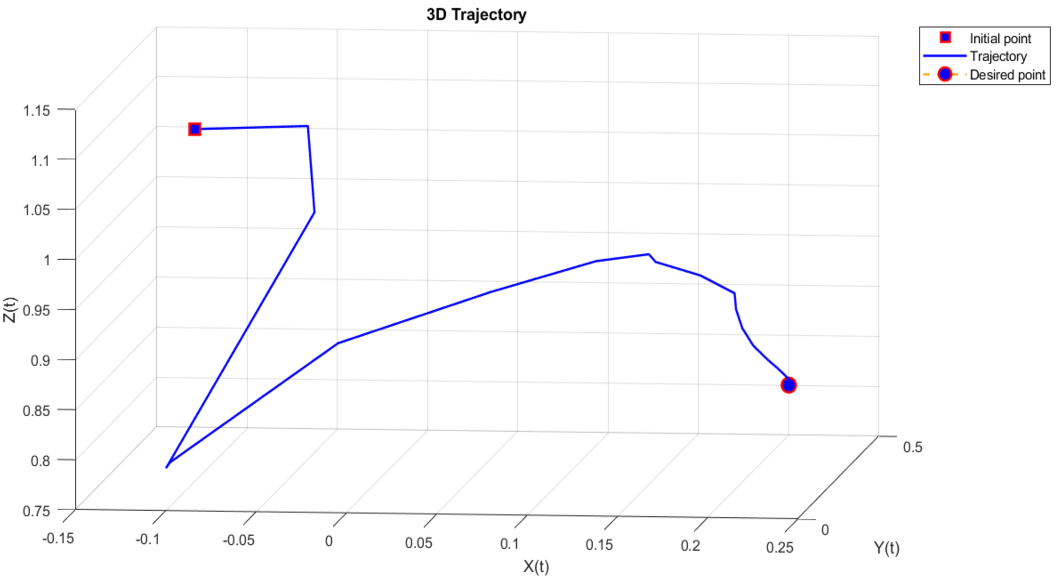


Figure 5.32: Control and trajectory optimization: 3D Trajectory.

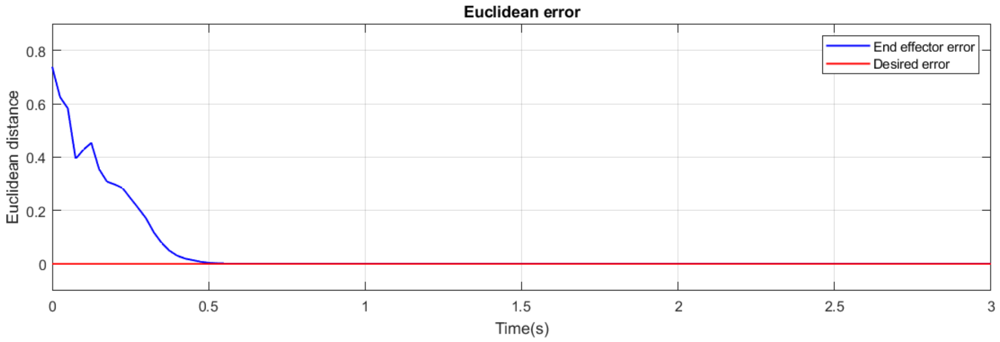


Figure 5.33: Control and trajectory optimization: Euclidean error.

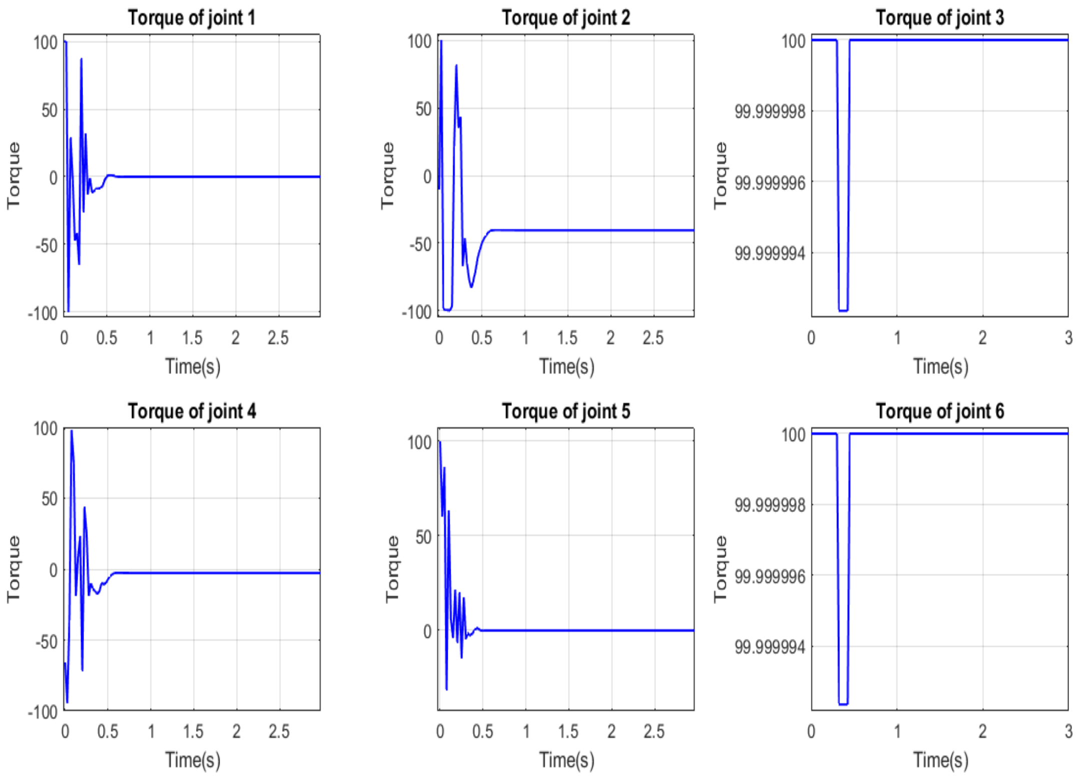


Figure 5.34: Control and trajectory optimization: torques (control signals).

5.6.2.3 Interpretation:

The figure 5.28 representing the progression of the cumulative reward over time provides valuable insights into the learning process. It showcases both the episode reward and the average reward (calculated over 100 episodes). Initially, during the early stages of training, the system (consisting of the robot) struggles to reach the target point, which is expected since the agent lacks prior knowledge of the environment. As a result, the initial actions yield a reward of -100 for failing to generate a trajectory leading to the desired position and control the robot in the tracking. However, as the agent continues its training cycle, the weights are adjusted, leading to a gradual increase in the cumulative reward. Eventually, it reaches its peak value of 1000 after approximately 4500 episodes, indicating that the agent has successfully identified solutions to the optimization problem and consistently selects the optimal one. In essence, the agent learns, acquires new skills, and successfully achieves its objective. It is crucial to mention that fluctuations and instability in the cumulative reward arise due to the updating of neural networks (NNs) and the application of noise, along with the emergence of new states that the agent encounters. The training duration is relatively long due to the agent striking a balance between exploration and exploitation, essential for effective learning and policy improvement. The lengthy learning process can be justified considering the limited availability of training resources.

Analyzing the data in figures 5.29, 5.30, 5.31, and 5.33, it becomes evident that the robot achieves precise goal attainment following the training. The error is significantly reduced to 2mm, and the joint positions and velocities adjust (in an irregular way because the agent only focuses on optimizing the reward function which includes precision and energy minimization) as well as possible to the required configuration in approximately 0.5s due to the absence of speed limits. Unlike other experiences, velocities shown in figure 5.31 are greater. In figure 5.34, it is illustrated how the robot's movements are optimized, emphasizing efficiency. The majority of torques remain below 100 (except the torque applied to joint 2, which requires greater effort), and some torques converge to zero, indicating the agent's successful training on minimization of energy consumption. Under the specified training conditions and with the absence of conditions on the trajectory, the agent has learned the one depicted in figure 5.32. It is important to emphasize that further extended training and modifications to the reward function can lead to further refinement and improvement of the trajectory.

The achieved results are remarkable, especially considering the constraints of limited training resources, a restricted training period, and the agent's lack of prior knowledge about the environment.

5.7 Robustness analysis

In order to investigate the robustness of the previous control strategies, a load of 3kg is added.

5.7.1 PID Control

5.7.1.1 Results

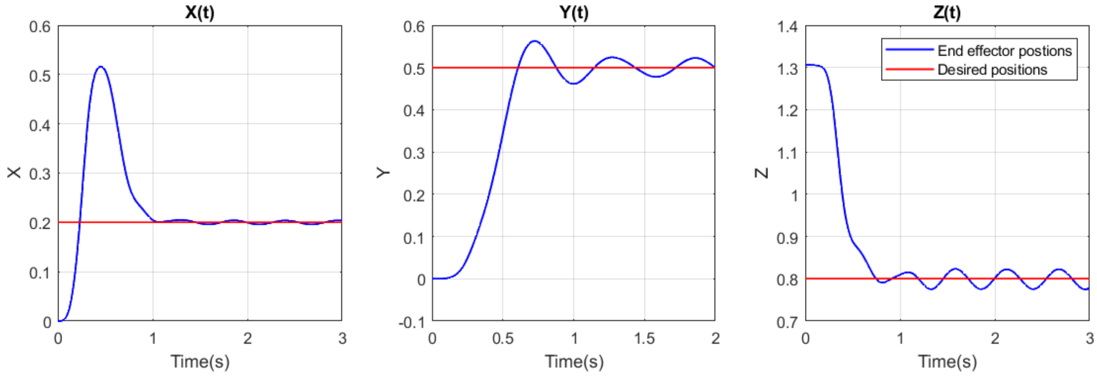


Figure 5.35: PID: End effector's position with respect to the desired position (adding 3kg).

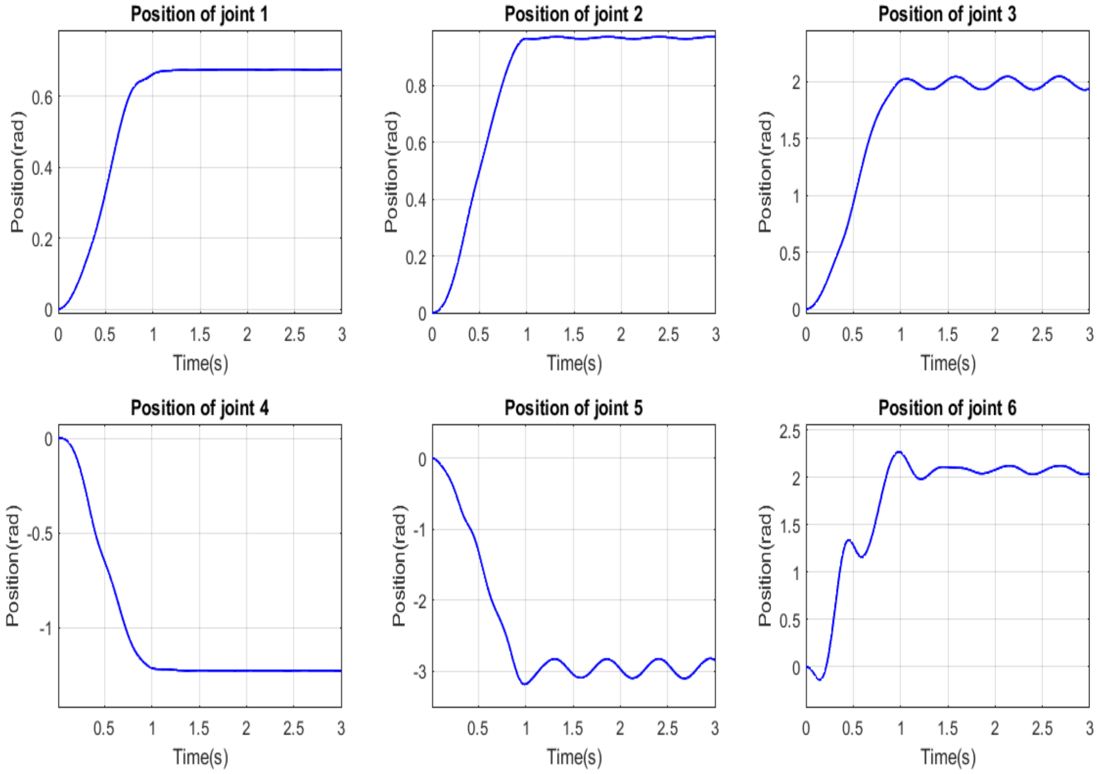


Figure 5.36: PID: joint positions (adding 3kg).

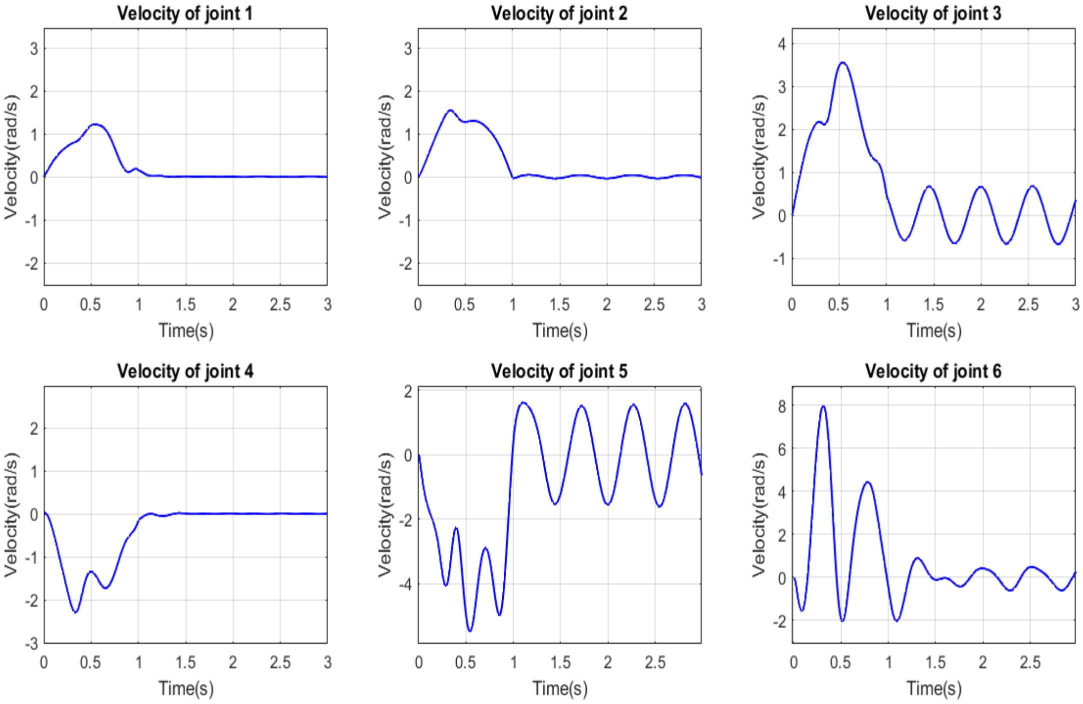


Figure 5.37: PID: joint velocities (adding 3kg).

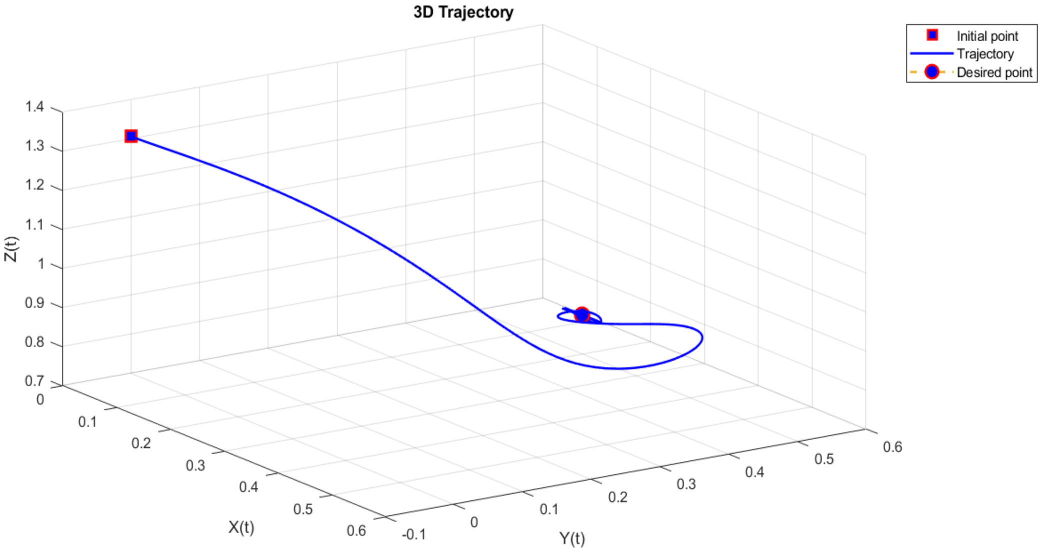


Figure 5.38: PID: 3D Trajectory (adding 3kg).

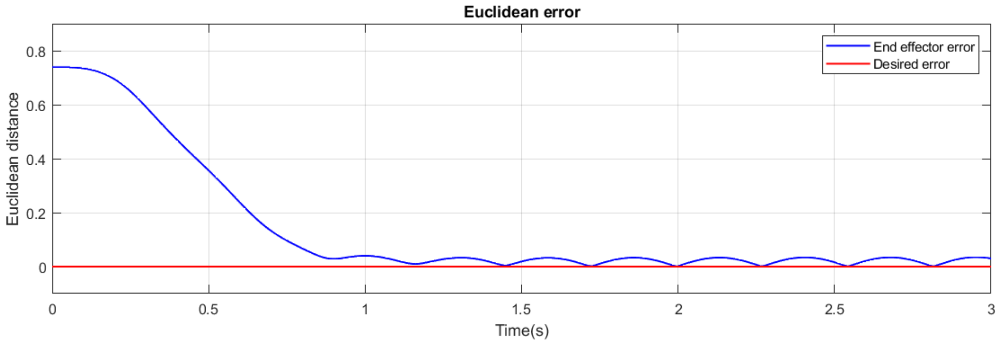


Figure 5.39: PID: Euclidean error (adding 3kg).

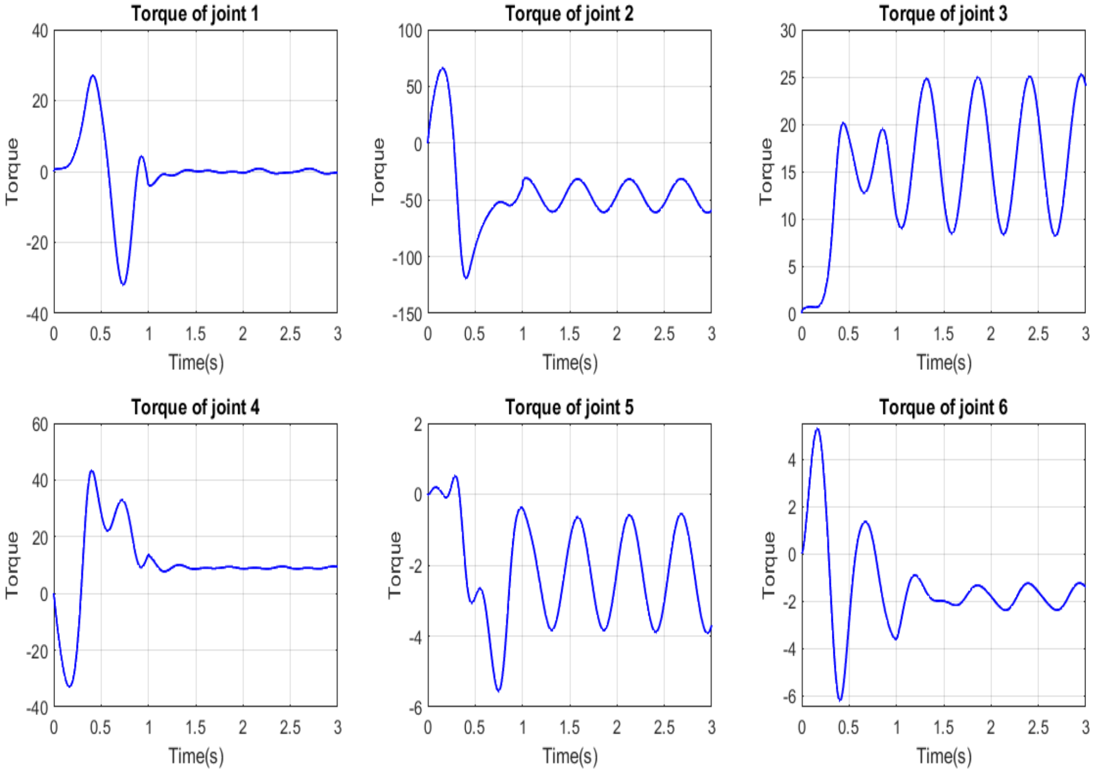


Figure 5.40: PID: torques (control signals) after adding 3kg.

5.7.1.2 Interpretation

The aforementioned figures provide compelling evidence that the performance of the PID control law has significantly deteriorated. Notably, substantial oscillations are observed in the joint positions (figure 5.36), velocities (figure 5.37), error (figure 5.39), and torques signals which have increased (figure 5.40) as the control law attempts to adjust the position of the end-effector to the desired target with the additional 3kg load applied to the robotic arm. Although the end-effector eventually reaches the target position, it continues to oscillate in its vicinity, as demonstrated in figures 5.35 and 5.38. Furthermore, a discernible change in the trajectory is observed due to the presence of the added load. Consequently, both the stability and precision of the PID control law are compromised, raising concerns about its robustness in this scenario.

5.7.2 Computed Torque Control

5.7.2.1 Results

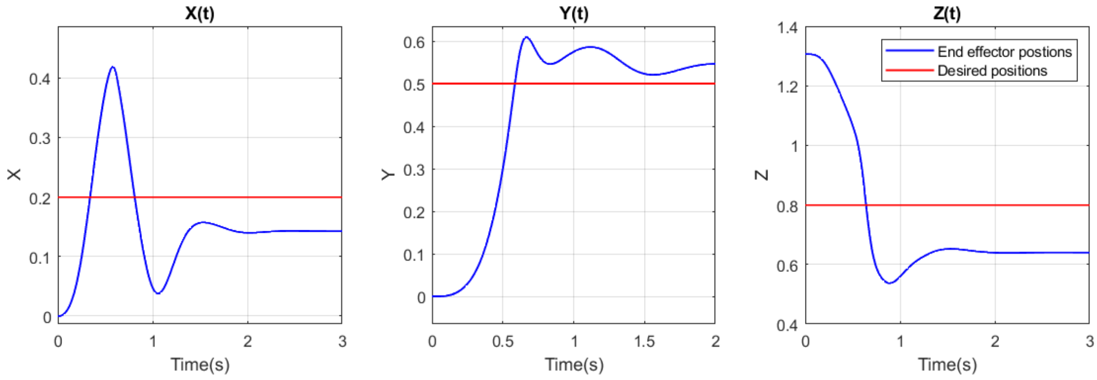


Figure 5.41: CTC: End effector's position with respect to the desired position (adding 3kg).

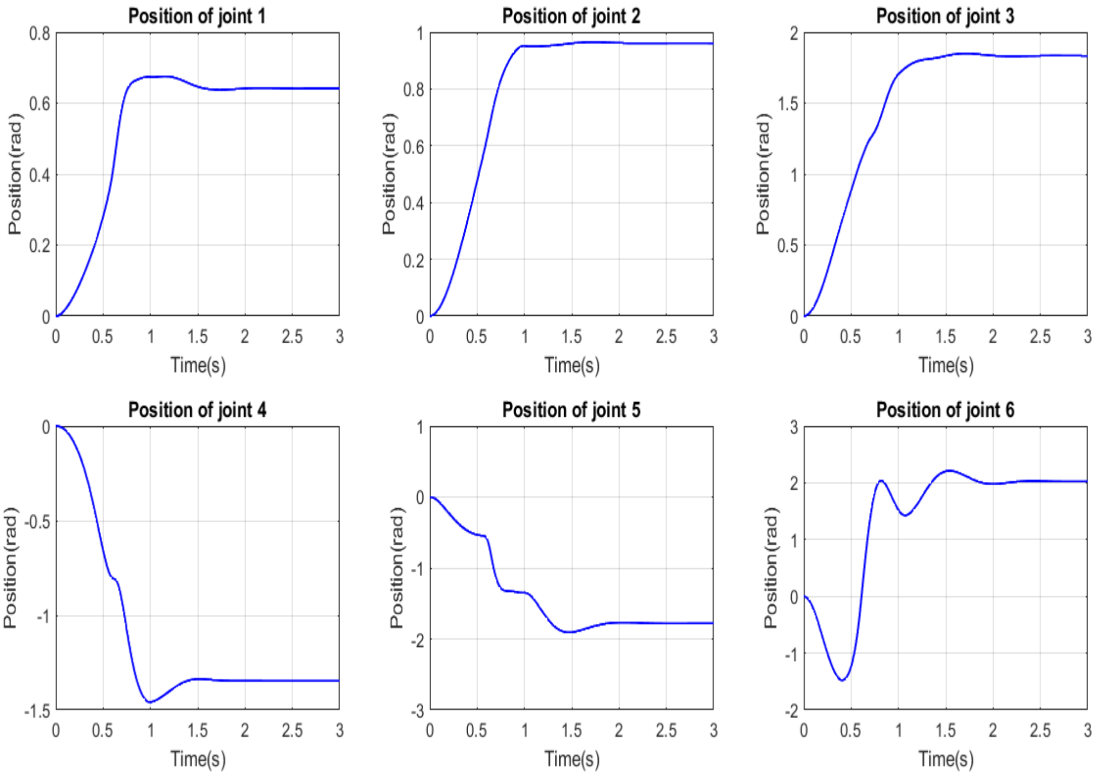


Figure 5.42: CTC: joint positions (adding 3kg).

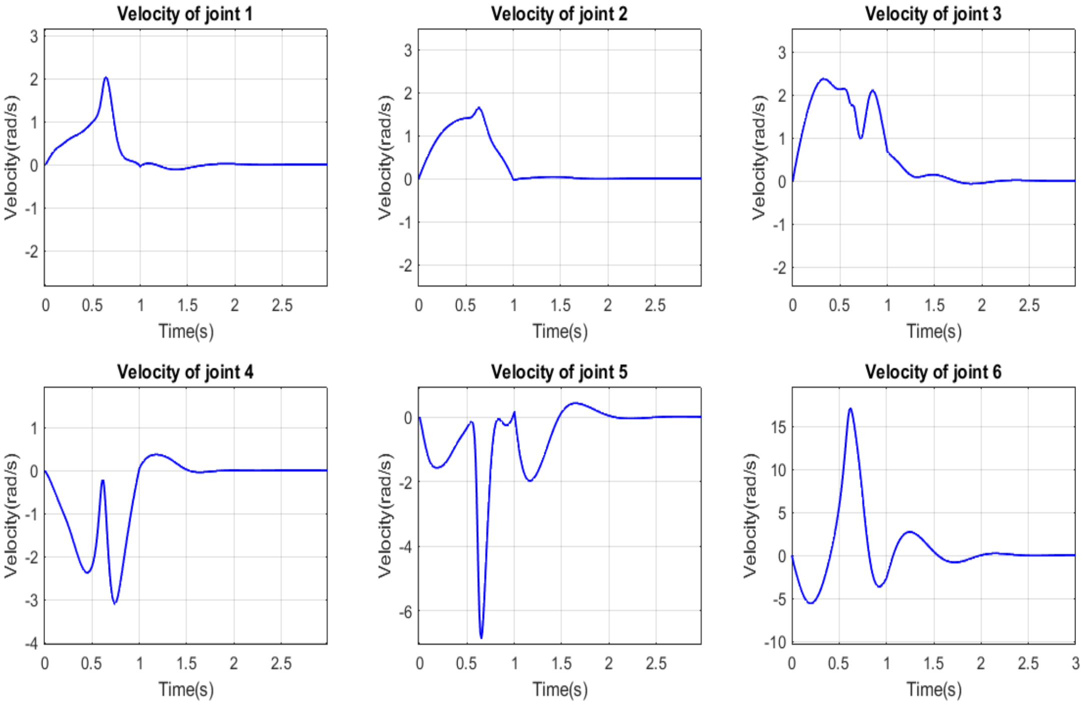


Figure 5.43: CTC: joint velocities (adding 3kg).

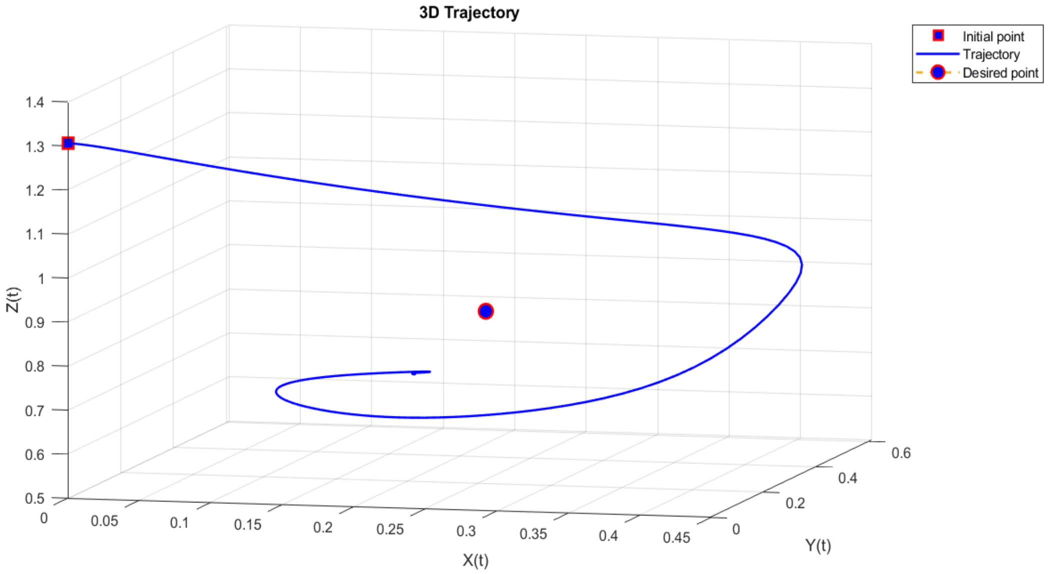


Figure 5.44: CTC: 3D Trajectory (adding 3kg).

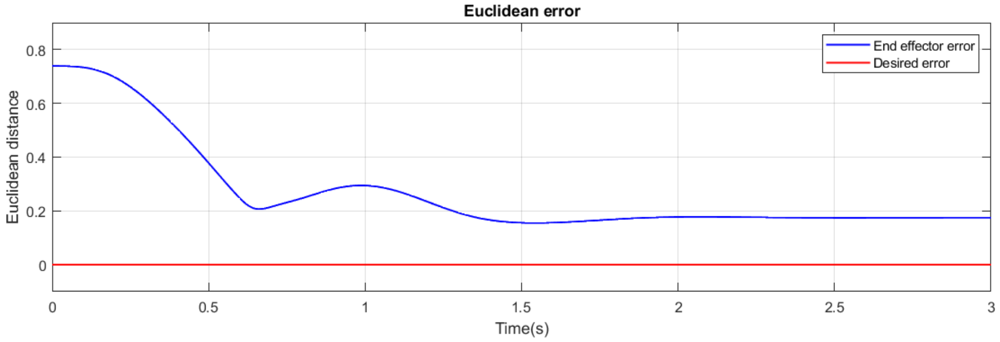


Figure 5.45: CTC: Euclidean error (adding 3kg).

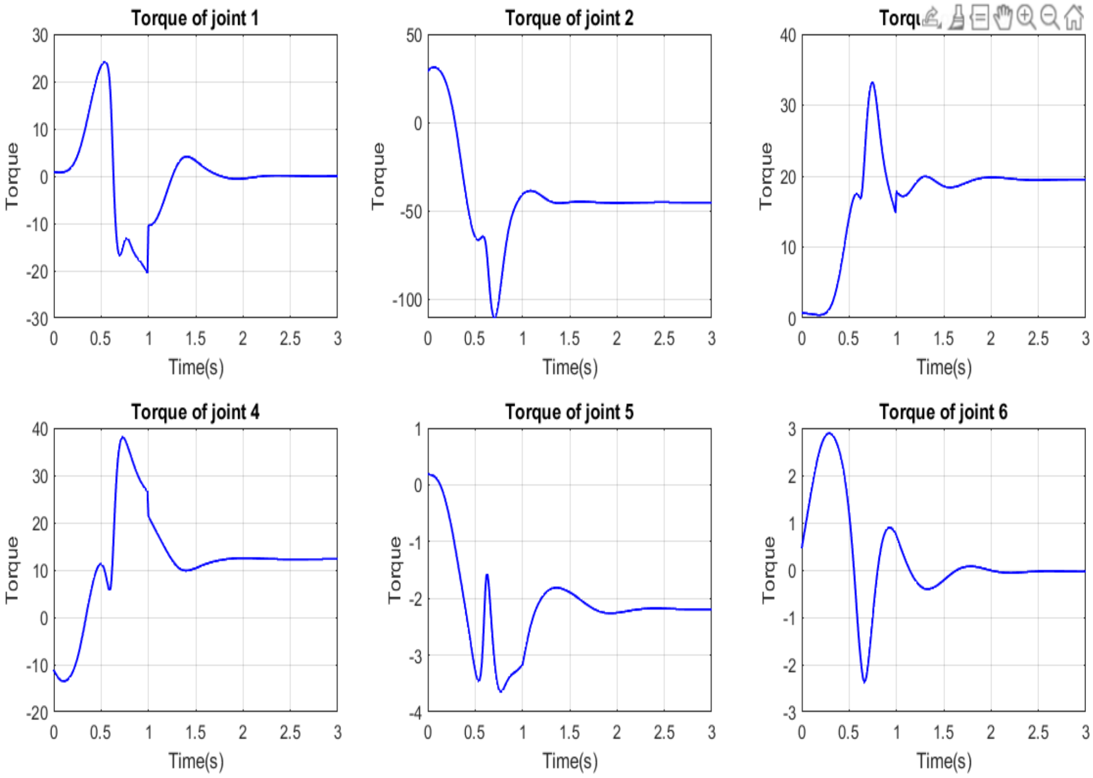


Figure 5.46: CTC: torques (control signals) after adding 3kg.

5.7.2.2 Interpretation

The figures presented above illustrate the impact on the performance of the computed torque control law. In contrast to PID control, the joint positions (figure 5.42), velocities (figure 5.43), error (figure 5.45), and torques (figure 5.46) signals exhibit damped oscillations as the control law mitigates non-linearities and applies a Proportional-Derivative (PD) control approach to adjust the end-effector's position to the desired target while accommodating the additional 3kg load on the robotic arm. While stability in the presence of the load is achieved, the control law falls short in attaining the desired precision. The error signal depicted in figure 5.45 stabilizes at a value of 0.2m within 2 seconds, indicating a considerable deviation from the target position, as shown in figures 5.41. Moreover, a noticeable deviation in the trajectory is observed due to the presence of the added load. Consequently, both the precision and speed of the computed torque control law are compromised in this scenario.

5.7.3 Sliding Mode Control

5.7.3.1 Results

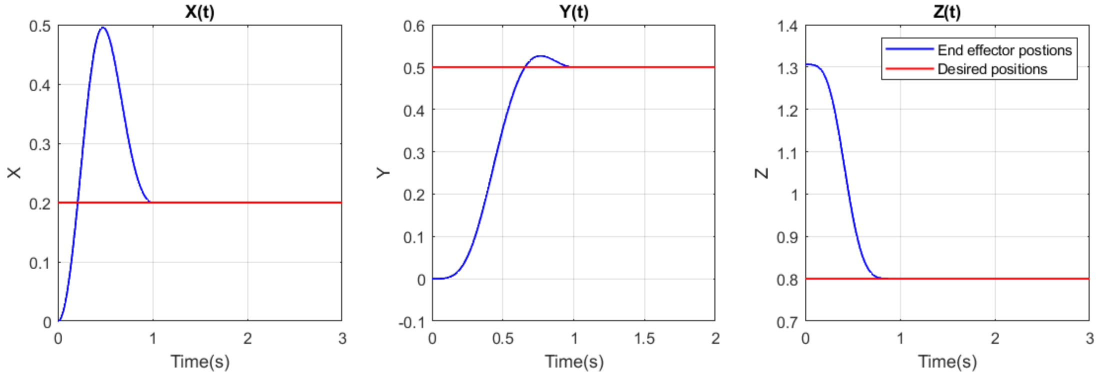


Figure 5.47: SMC: End effector's position with respect to the desired position (adding 3kg).

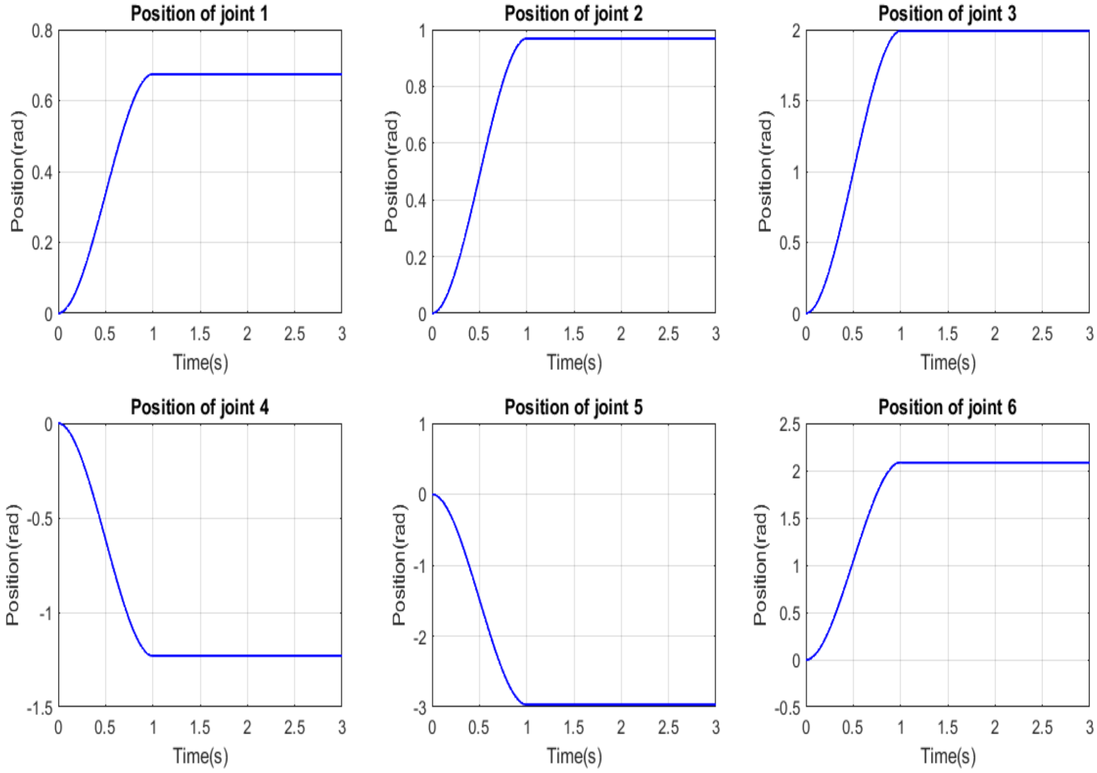


Figure 5.48: SMC: joint positions (adding 3kg).

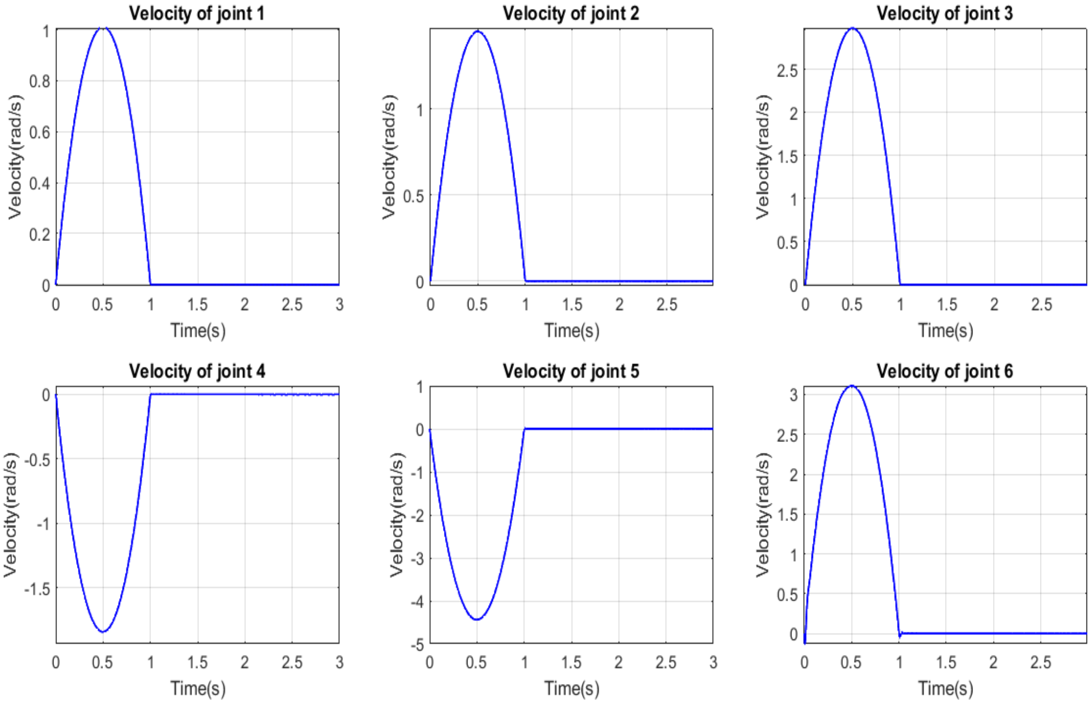


Figure 5.49: SMC: joint velocities (adding 3kg).

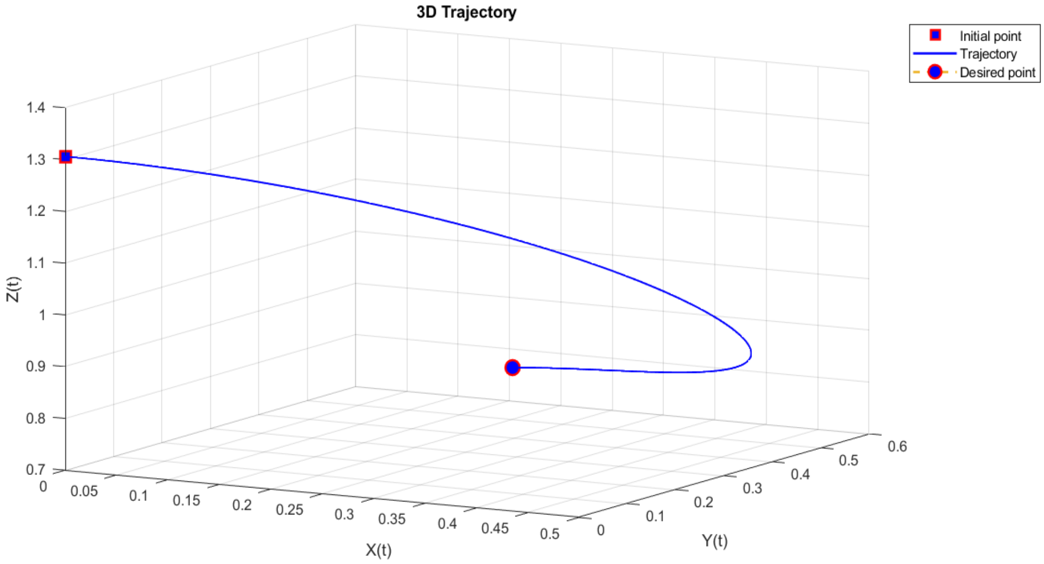


Figure 5.50: SMC: 3D Trajectory (adding 3kg).

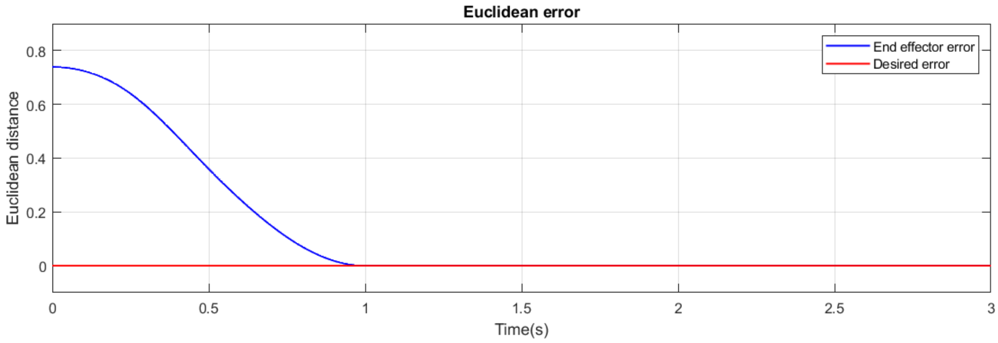


Figure 5.51: SMC: Euclidean error (adding 3kg).

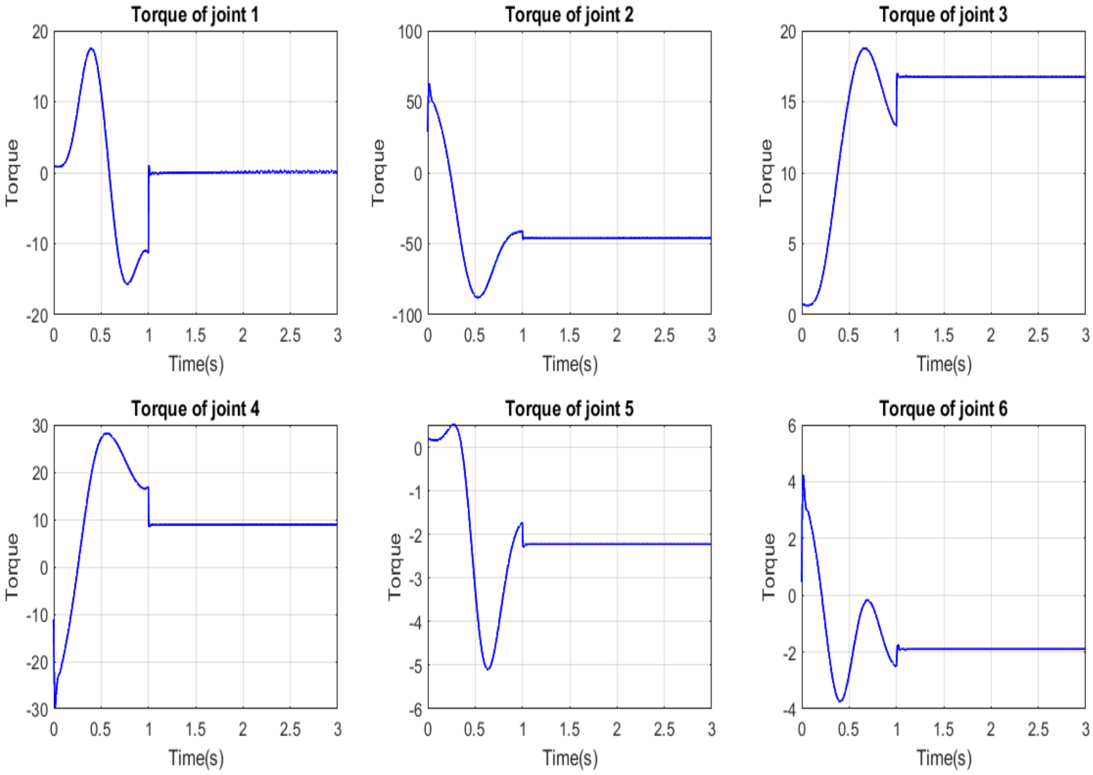


Figure 5.52: SMC: torques (control signals) after adding 3kg.

5.7.3.2 Interpretation

The figures presented above provide compelling evidence of the influence of the additional load on the performance of the sliding mode control law. They demonstrate that the sliding mode control law is a robust strategy compared to PID and computed torque control. Unlike the other control methods, the joint positions (figure 5.48) and velocities (figure 5.49) signals do not exhibit oscillations and converge to the desired configuration, indicating the effectiveness of the control law in adjusting the end-effector's position to the target while accommodating the additional 3kg load on the robotic arm. Stability in the presence of the load is confirmed. Additionally, the robot achieves the desired precision, as evidenced by the convergence of the end-effector's position to its reference accurately (figure 5.47). The error signal depicted in figure 5.51 stabilizes at zero within 1 second, indicating that the control law maintains its rapid response. Furthermore, figure 5.50 shows no deviation in the trajectory due to the presence of the added load. However, slight oscillations, known as chattering, are observed in the torques (figure 5.52), which is expected as the control law adjusts to handle the added load. Overall, the sliding mode control law maintains its performance and robustness in this scenario.

5.7.4 Deep Reinforcement Learning Control

5.7.4.1 Results

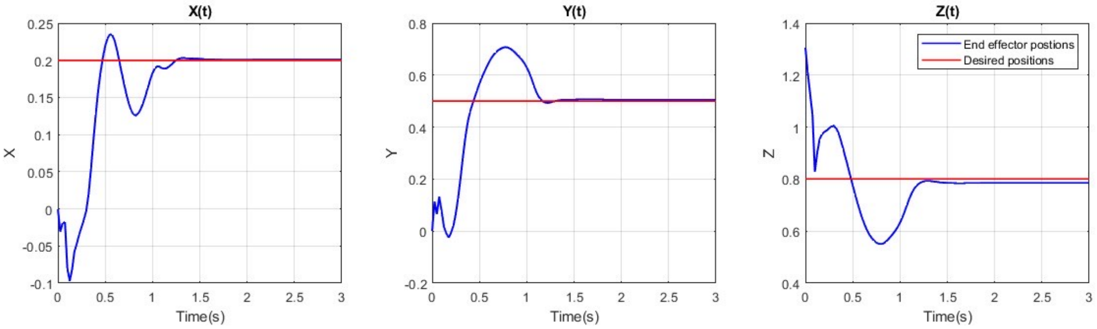


Figure 5.53: DRLC: End effector's position with respect to the desired position (adding 3kg).

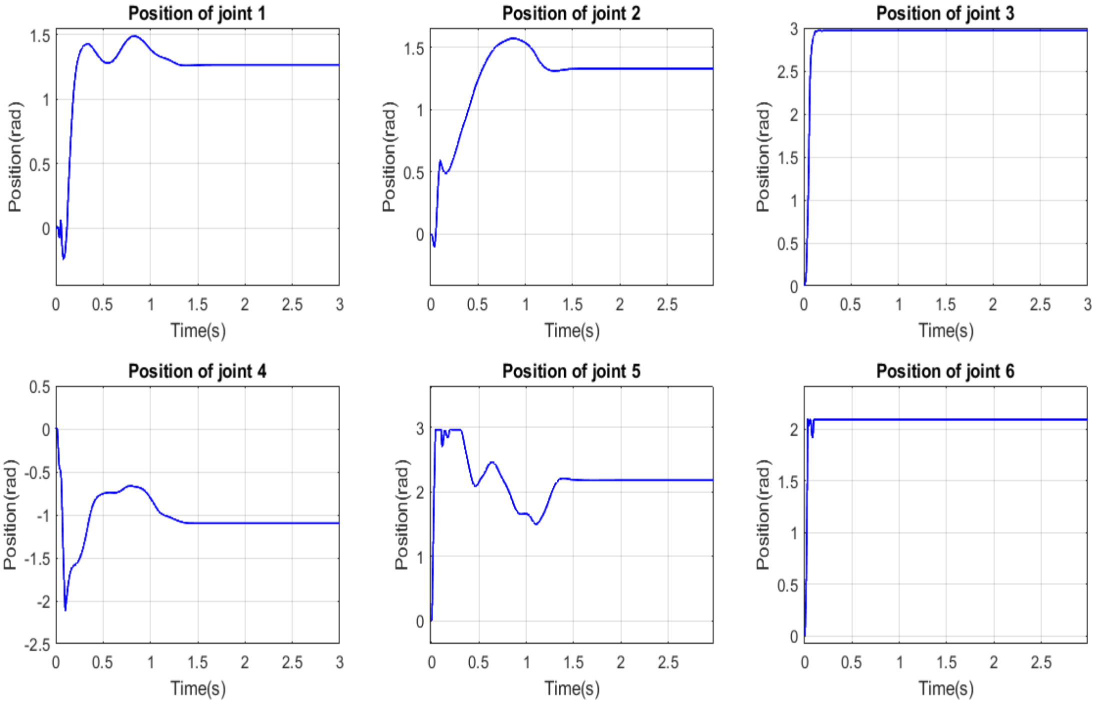


Figure 5.54: DRLC: joint positions (adding 3kg).

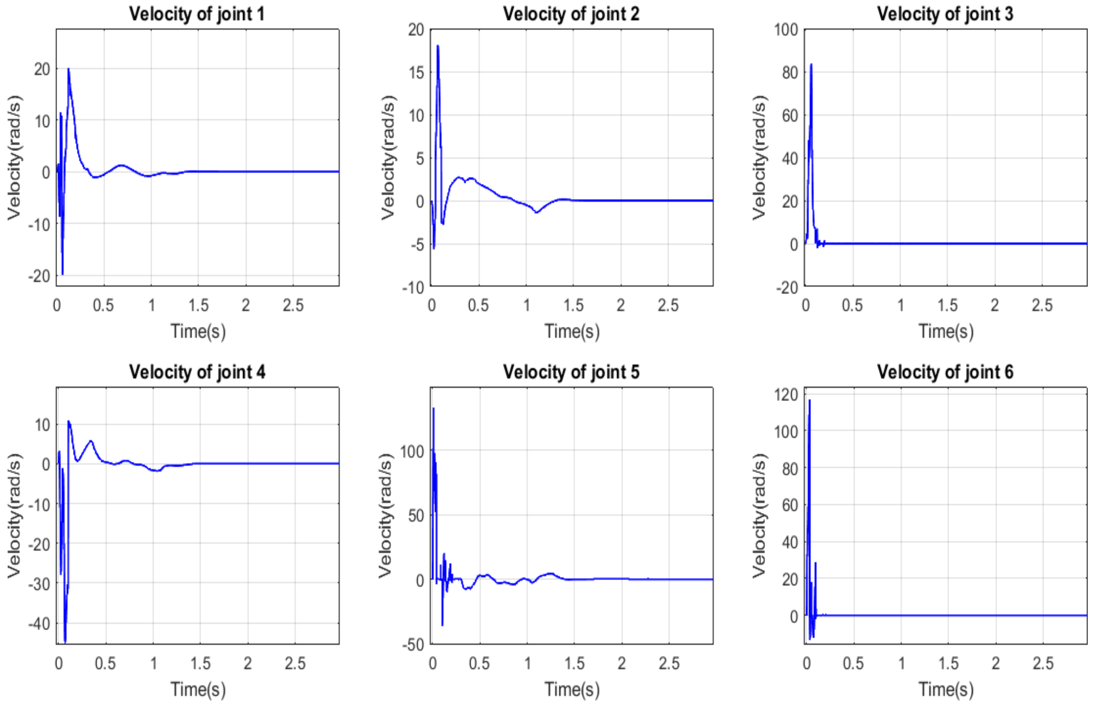


Figure 5.55: DRLC: joint velocities (adding 3kg).

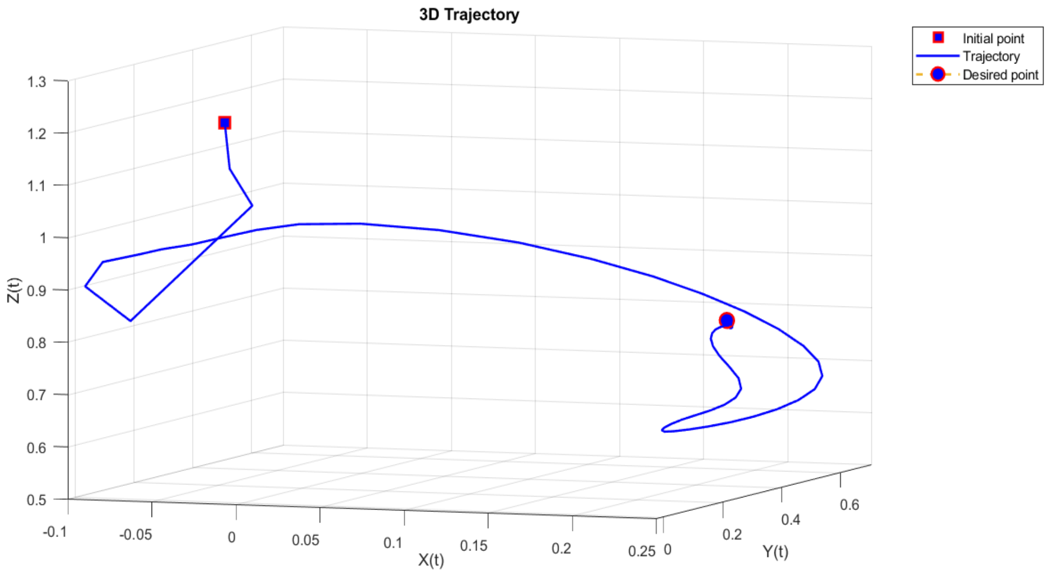


Figure 5.56: DRLC: 3D Trajectory (adding 3kg).

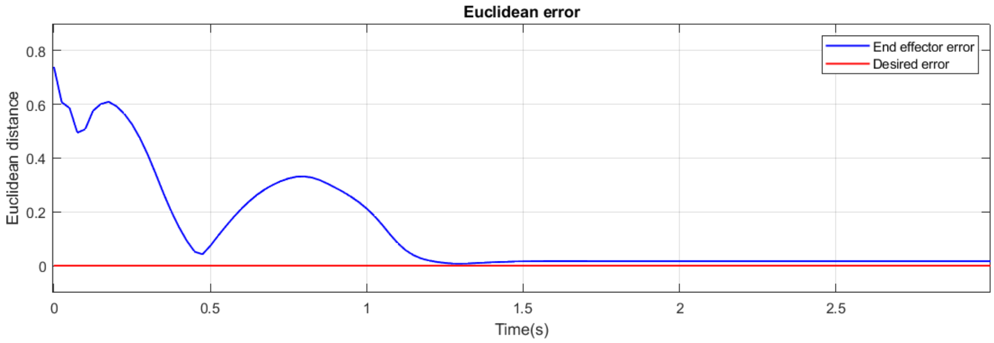


Figure 5.57: DRLC: Euclidean error (adding 3kg).

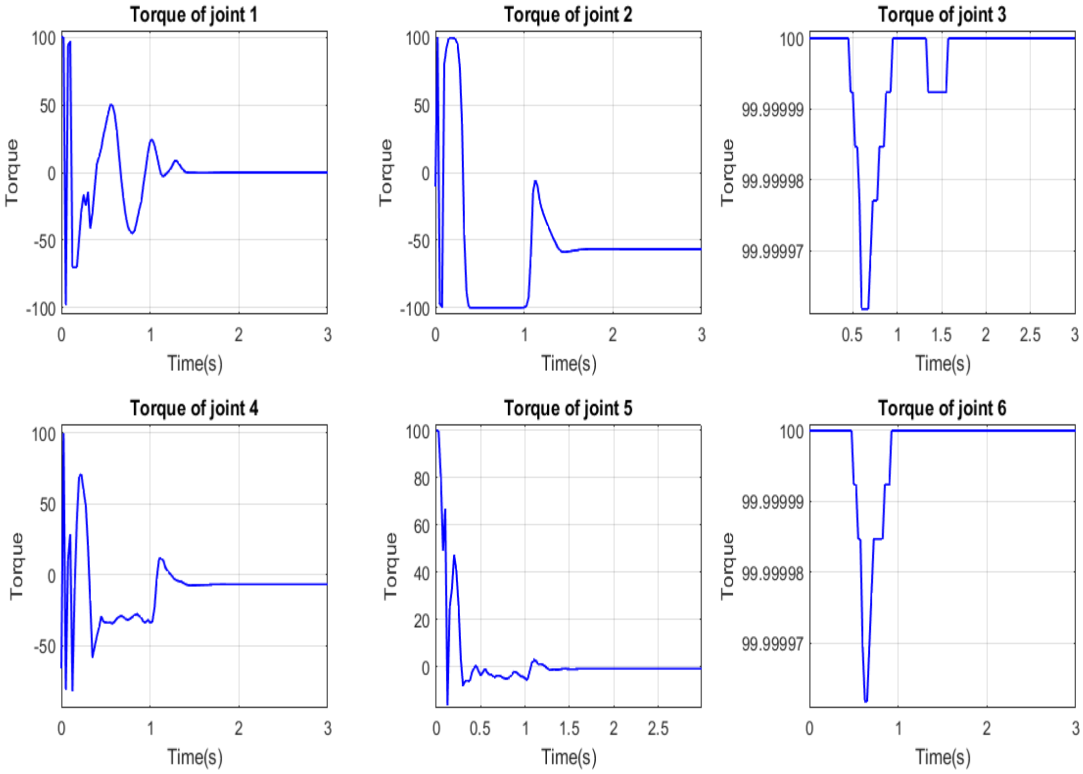


Figure 5.58: DRLC: torques (control signals) after adding 3kg.

5.7.4.2 Interpretation

According to the figures presented above, the DRL control strategy maintains its performance in the presence of the additional load. The joint positions (figure 5.54) and velocities (figure 5.55) signals do not exhibit oscillations and converge towards the desired configuration, indicating the effectiveness of the control law in adjusting the end-effector's position to the target while accommodating the additional 3kg load on the robotic arm. Stability in the presence of the load is confirmed. Additionally, the robot achieves a high level of precision, as evidenced by the convergence of the end-effector's position to its reference (figure 5.53). The error signal depicted in figure 5.57 stabilizes at a value of 1cm within 1.5 seconds, indicating a slightly altered rapid response of the control law. Furthermore, figure 5.56 shows a smoother trajectory that adapts to the presence of the added load. The torque signals remain below 100, and most of them tend towards 0 (figure 5.58), providing evidence of the control strategy's effectiveness in minimizing effort. The achieved results demonstrate generalization capabilities, as the agent successfully adapts to the new conditions of the additional load, despite not being specifically trained for it.

Therefore, the Deep Reinforcement Learning control approach can be considered robust in this scenario.

5.8 Comparative Analysis

The following table summarizes the comparison between Proportional-Integral-Derivative control (PID), Computed Torque Control (CTC), Sliding Mode Control (SMC) and Deep Reinforcement Learning Control (DRLC)

Table 5.2: Comparison of Control Strategies

Criteria	PID	CTC	SMC	DRL
Performance and Tracking Accuracy	Good, but may have steady-state error and overshoot	Excellent and low steady-state error	Excellent, but potential for chattering	Excellent performance (depending on training factors)
Robustness	Sensitive to disturbances and parameter variations	Less sensitive to disturbances and uncertainties	Robust against uncertainties but may suffer from chattering	Robustness depends on training and generalization
Control Effort	Depends on tuning parameters	Low	Moderate (depends on design)	Depends on training and NN complexity
Computational Requirements	Low	Moderate	Moderate	High (for training)
Adaptability and Learning	Limited	Limited	Moderate (to uncertainties)	High adaptability and learning capability
Implementation Complexity	simple	Moderate	Moderate	Complex (requires neural network infrastructure)

Criteria	PID	CTC	SMC	DRL
Model Dependency	Relies on a mathematical model of the system	Requires knowledge of system dynamics for controller design	Relies on a mathematical model of the system	Model-free approach, doesn't rely on a specific system model
Training Data	Does not require training data	Does not require training data	Does not require training data	Requires a sufficient amount of training data for learning
Explainability	Transparent and interpretable	Transparent and interpretable	Transparent and interpretable	Less transparent (considered as a black-box)
Intelligence	N/A	N/A	N/A	Learns and improves its control behavior over time through training
Generalization	N/A	N/A	N/A	capable of Generalizing learned behavior to unseen situations
Application-Specific Considerations	May require task-specific tuning	Suitable for precise control tasks	Suitable for applications requiring good tracking accuracy and robustness against uncertainties with careful design to avoid chattering	Potential for solving complex control tasks (good training)

5.9 Conclusion

This Chapter examined the robotic arm control methods of Proportional-Integral-Derivative (PID), Computed Torque Control (CTC), Sliding Mode Control (SMC), and Deep Reinforcement Learning (DRL). Several major conclusions from considerable study and testing have shown the merits and weaknesses of each robotic arm control approach.

- PID control performs well in robotic arm trajectory tracking tasks but it struggles with severe nonlinearities and uncertainty as well as great disturbances.
- CTC control demonstrates good robotic arm trajectory tracking and disturbance rejection. For high-precision control applications, it handled nonlinearities and disturbances quite well. CTC control requires a good knowledge of the system's model to produce precise control.
- SMC robotic arm control mitigated uncertainties and disturbances. SMC control outperforms the other control techniques in terms of accuracy, rapidity, stability, effort control and robustness while performing the task. However, chattering may cause mechanical wear and tear.
- DRL control demonstrated robotic arm autonomy and adaptability. Deep neural networks and reinforcement learning algorithms taught DRL control optimum control policies via environmental interaction. It handled complicated, high-dimensional systems without explicit models, making it adaptive. DRL control requires significant training time and resources for excellent results and is also considered a "black box".

Understanding each method's advantages and disadvantages helps in selecting the adequate strategy according to the application requirements

General Conclusion

This thesis focuses on developing and evaluating intelligent control techniques for robust robotic systems. The research involves modeling, controlling, and optimizing a modern and redundant robotic system with over 6 degrees of freedom. The aim is to overcome the limitations of traditional control methods and demonstrate the potential of intelligent control approaches in achieving superior performance and adaptability.

The thesis begins with the mathematical modeling of the robotic system using numerical calculations and algorithms, ensuring an accurate representation of its geometry and dynamics. Computer-aided design techniques are employed to create a detailed 3D representation.

Various control strategies, including classical methods like PID, CTC, and SMC, are tested under unloaded and loaded conditions, with a focus on evaluating the performance of the end effector.

Building on the insights gained from classical control, the thesis explores the integration of intelligent control techniques based on artificial intelligence and reinforcement learning algorithms. These approaches leverage AI's power to solve complex control and intelligence problems without requiring extensive domain knowledge or manual system analysis. Training an agent to interact with the physical environment enhances the system's performance and optimization capabilities. Experimental results validate the effectiveness and potential of intelligent control techniques in achieving set objectives and exhibiting robustness in scenarios where traditional methods fall short.

Through a comprehensive comparative analysis, the research emphasizes the advantages of intelligent control techniques over traditional approaches. It is anticipated that these advanced control strategies will play a pivotal role in future robotic systems, overcoming the challenges posed by complex physical systems. Ultimately, the utilization of intelligent control will enable the achievement of control and intelligence objectives that were previously unattainable using manual methods.

Perspectives

Building upon these findings and for future research, we will shift our focus toward the following perspectives to explore and unlock new possibilities:

- Exploring the hybrid control approach which combines both traditional and DRL control techniques.
- Integrating perception and vision systems.
- Extending the scope of this study to analyze the effects of friction torques and actuator dynamics on the control system.
- Investigating methods to transfer learned control policies from a simulated environment to a physical robotic arm.

Bibliography

[1] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396, IEEE, 2017.

[2] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine, "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6284–6291, IEEE, 2018.

[3] P. Varin, L. Grossman, and S. Kuindersma, "A comparison of action spaces for learning manipulation tasks," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6015–6021, IEEE, 2019.

[4] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4383–4388, IEEE, 2020.

[5] V. Chawda and G. Niemeyer, "Toward torque control of a kuka lbr iiwa for physical human-robot interaction," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6387–6392, IEEE, 2017.

[6] E. Dombre and W. Khalil, *Robot manipulators: modeling, performance analysis and control*. John Wiley & Sons, 2013.

[7] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Science & Business Media, 2010.

[8] C. Faria, F. Ferreira, W. Erlhagen, S. Monteiro, and E. Bicho, "Position-based kinematics for 7-dof serial manipulators with global configuration control, joint limit and singularity avoidance," *Mechanism and Machine Theory*, vol. 121, pp. 317–334, 2018.

[9] M. H. Boussamet and L. I. Saadi, "Contribution à la commande non linéaire d'un bras manipulateur flexible," Master's thesis, University of Science and Technology, 2019.

[10] A. M. Shuman, "Modeling and control of 6-axis robot arm," 2020.

[11] B. Benhabib, A. A. Goldenberg, and R. G. Fenton, “A solution to the inverse kinematics of redundant manipulators,” *Journal of robotic systems*, vol. 2, no. 4, pp. 373–385, 1985.

[12] Y. Bestaoui, “An unconstrained optimization approach to the resolution of the inverse kinematic problem of redundant and nonredundant robot manipulators,” *Robotics and autonomous systems*, vol. 7, no. 1, pp. 37–45, 1991.

[13] K. Erleben and S. Andrews, “Inverse kinematics problems with exact hessian matrices,” in *Proceedings of the 10th International Conference on Motion in Games*, pp. 1–6, 2017.

[14] A. Mueller, “Modern robotics: Mechanics, planning, and control [bookshelf],” *IEEE Control Systems Magazine*, vol. 39, no. 6, pp. 100–102, 2019.

[15] M. W. Spong, S. Hutchinson, M. Vidyasagar, *et al.*, *Robot modeling and control*, vol. 3. Wiley New York, 2006.

[16] J. J. Craig, *Introduction to Robotics Mechanics and Control.* Pearson Prentice Hall, 2005.

[17] W. KHALIL and E. DOMBRE, “Bases de la modélisation des robots,”

[18] H. M. NOUARI Ferial, “Commande pid adaptative d’un robot manipulateur à 2ddl,”

[19] A.-N. Sharkawy and P. Koustoumpardis, “Dynamics and computed-torque control of a 2-dof manipulator: Mathematical analysis,” *International Journal of Advanced Science and Technology*, vol. 28, no. 12, pp. 201–212, 2019.

[20] M. I. Ullah, S. A. Ajwad, R. U. Islam, U. Iqbal, and J. Iqbal, “Modeling and computed torque control of a 6 degree of freedom robotic arm,” in *2014 International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE)*, pp. 133–138, IEEE, 2014.

[21] R. Kelly, V. S. Davila, and J. A. L. Perez, *Control of robot manipulators in joint space.* Springer Science & Business Media, 2005.

[22] K. Abdelkader, “Synthese des commandes robustes des robots manipulateurs rigides,” *Memoire, Universite d’Oran Es-Senia*, vol. 5, 2004.

[23] F. L. Lewis, D. M. Dawson, and C. T. Abdallah, *Robot manipulator control: theory and practice.* CRC Press, 2003.

[24] Z. Abbas, *Motion control of robotic arm manipulator using pid and sliding mode technique*. PhD thesis, Capital University, 2018.

[25] M. L. Dahmani, *Commande par mode glissant d'ordre supérieur dans Le domaine de position: application au robot manipulateur*. PhD thesis, 2017.

[26] J.-J. E. Slotine, W. Li, *et al.*, *Applied nonlinear control*, vol. 199. Prentice hall Englewood Cliffs, NJ, 1991.

[27] A. Yahdou, M. Bounadja, Z. Boudjemaà, and B. Belmadani, "Commande à structure variable par mode glissant d'une machine asynchrone double-etoile,"

[28] W. Perruquetti and J. P. Barbot, *Sliding mode control in engineering*, vol. 11. Marcel Dekker New York, 2002.

[29] M. Belhocine, M. Hamerlain, and K. Bouyoucef, "Robot control using a sliding mode," in *Proceedings of 12th IEEE International Symposium on Intelligent Control*, pp. 361–366, IEEE, 1997.

[30] T. De Bruin, J. Kober, K. Tuyls, and R. Babuska, "Experience selection in deep reinforcement learning for control," *Journal of Machine Learning Research*, vol. 19, 2018.

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

[32] G. C. Andrei, "Deep reinforcement learning for dynamical systems," Master's thesis, Université de Lorraine, 2022.

[33] T. de Bruin, "Sample efficient deep reinforcement learning for control," 2020.

[34] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[35] T. J. Stray, "Application of deep reinforcement learning for control problems," Master's thesis, NTNU, 2019.

[36] Organization offering the course, "Reinforcement learning with matlab."

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

[38] M. A. S. CID, *REINFORCEMENT LEARNING ON CONTROL SYSTEMS WITH UNOBSERVED STATES.* PhD thesis, UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA, 2017.

[39] D. Mehta, "State-of-the-art reinforcement learning algorithms," *International Journal of Engineering Research and Technology*, vol. 8, pp. 717–722, 2020.

[40] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more.* Packt Publishing Ltd, 2020.

[41] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, 2021.

[42] I. Goodfellow, "Deep learning-ian goodfellow, yoshua bengio, aaron courville- google books," 2016.

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

[44] OpenAI, "Openai."

Appendix A

Quasi-Newtonian method

The following algorithm represents the BFGS (Broyden-Fletcher-Goldfarb-Shanno) method, which is a quasi-Newton method used for optimization problems. It approximates the Hessian matrix and the gradient of the error function to find the direction of convergence towards the minimum. The BFGS method iteratively performs the following steps for N number of iterations to approach the minimum of the error function and update the joint positions accordingly.

1. Initialization: initializing the joint values θ_i and the initial Hessian matrix $B(0)$.

2. Calculation of the direction: The direction $P(k)$ is computed as the inverse of the Hessian matrix $B(k)$ multiplied by the negative gradient of the error function $\nabla f(\theta(k))$. This direction represents the update step to move closer to the minimum.

$$P(k) = -B(k)^{-1} \nabla f(\theta(k)) \tag{A.1}$$

3. Updating joint values: The joint values θ_i are updated using the formula:

$$\theta_i(k+1) = \theta_i + \alpha \cdot P(k) \tag{A.2}$$

   where α is a factor that determines the step size or rate of change for each iteration

4. Calculation of $y(k)$: It represents the change in the gradient.

$$y(k) = \nabla f(\theta(k+1)) - \nabla f(\theta(k)) \tag{A.3}$$

5. Update of the Hessian matrix: The Hessian matrix $B(k+1)$ is updated using the formula:

$$B(k+1) = B(k) + \frac{y(k) \cdot y(k)^T}{y(k)^T \cdot s(k)} - \frac{B(k) \cdot s(k) \cdot s(k)^T \cdot B(k)^T}{s(k)^T \cdot B(k) \cdot s(k)} \tag{A.4}$$

   $s(k)$ represents the change in joint positions, given by $s(k) = \alpha P(k)$