People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Oum El Bouaghi
Faculty of Sciences and Applied Sciences

Thesis

Presented to obtain

3rd Cycle Doctorate

Branch: Electronics

Specialty: Electronics of Embedded Systems

Title :

Contribution to Motion Planning and Control of Robotic Systems

Presented by :

Akram GHEDIRI

Publicly defended on 17/01/2024 in front of the following committee members:

N°	first and last name	Grade	University	Quality
01	Mohamed LASHAB	Prof.	University of Oum El Bouaghi	President
02	Kheireddine LAMAMRA	Prof.	University of Oum El Bouaghi	Supervisor
03	Abdelaziz AIT KAKI	MCA	University of Oum El Bouaghi	Co-reporter
04	Sofiane BOUOUDEN	Prof.	University of Khenchela	Examiner
05	Abderrahim Fayçal MEGRI	MCA	ENP Constantine	Examiner
06	Abdelhafid ZEROUAL	MCA	University of Skikda	Examiner

Declaration

I hereby declare that this dissertation is a product of my independent work completed after enrolling in the third cycle PhD program at Larbi Ben M'hidi University, Oum El Bouaghi, Algeria. It has not been previously included in any other thesis or dissertation presented to this institution or any other academic establishment for a degree, diploma, or any other qualification.

Akram GHEDIRI

Dedication

All Praise be to Allah Almighty for his blessing, for the strength and the patience he gave to me to accomplish this humbled work.

To my Mom & Dad
I would not be the man I am today without Allah and your lifetime of unfailing love, care, and sacrifices . I cannot repay you for all that you have done for me. From the bottom of my heart, I say thank you, and God bless you.

To my beloved wife and daughter
Through the ups and downs, you have been my unwavering support, providing encouragement, guidance, and friendship. Your belief in me and your presence, I am very lucky to have you in my life. Thank you, and God bless you.

To my beloved brothers and friends
I am forever grateful for the bond we share. Your presence has been a source of solace and encouragement, reminding me that I am never alone. Together, we have created memories that will forever be cherished.

Acknowledgement

My deepest appreciation to my exceptional supervisor, **Prof. Kheireddine LAMAMRA**. It is through his wise guidance that I discovered the true value of this field of study, and my appreciation for it has grown with each passing day. His words have served as a constant source of motivation, inspiring me to push myself further. His boundless wisdom and insightful guidance have been the guiding light that has helped me navigate through this journey and complete this work.

I also would like to express my sincere appreciation to all my professors and the dedicated members of the Laboratory of Electronics and New Technology (LENT). Their significant contributions and unwavering support have played a crucial role in the success of this humble work. I am truly grateful for their guidance and assistance throughout this endeavor.

Abstract

This study presents a novel design for adaptive PID gain tuning in the context of robot manipulators' PID Computed-Torque control. The approach incorporates Deep Deterministic Policy Gradient (DDPG) reinforcement learning to account for unmodeled dynamics and external disturbances. The main objective is to dynamically compute the outer-loop PID controller gains, ensuring minimal trajectory tracking errors while effectively rejecting disturbances and maintaining stable closed-loop dynamics.

To implement the control scheme, a comprehensive understanding of the robot's dynamics is necessary. Therefore, the study develops both the kinematic and dynamic equations for an n-link serial manipulator. The UR5e robot manipulator model is utilized, and the dynamic and kinematic parameters provided by the manufacturer and related works are considered to ensure accuracy.

Simulation results demonstrate the effectiveness and robustness of the proposed approach. It successfully handles bounded internal and external disturbances, showcasing its ability to adapt and mitigate their effects. Additionally, the adaptive gain tuning offers improved trajectory tracking performance compared to conventional PID controllers.

Overall, this work contributes to the advancement of control strategies for robot manipulators by integrating adaptive PID gain tuning with DDPG reinforcement learning. By addressing unmodeled dynamics and external disturbances, the proposed approach enhances the stability and tracking capabilities of robot manipulator systems.

Keywords: UR5e robot manipulator; Adaptive PID control; Computed-torque control; DDPG reinforcement learning; Trajectory tracking; Disturbance rejection.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Robot manipulators are crucial elements in modern industrial automation, where they are tasked with performing various precise and repeatable tasks along desired trajectories. However, achieving accurate trajectory tracking in robot manipulators poses a significant challenge due to their characteristics as Multi-Input Multi-Output (MIMO) nonlinear and hard-coupled systems [**1**][**6**]. Manipulators are subject to both structured uncertainties, such as imprecise inertial link properties and unknown payloads, and unstructured uncertainties caused by unmodeled dynamics like nonlinear friction, backlash, joint elasticity, and external disturbances [**7**][**8**]. Consequently, implementing model-based control techniques such as conventional PD and PID computed-torque controllers becomes challenging, especially in critical and demanding applications [**11**].

The computed torque control technique [**10**] emerges as a special application of feedback linearization that utilizes the system's exact dynamics to cancel nonlinear effects. PD computed torque control is effective and guarantees stability in the absence of external disturbances, while PID computed torque control proves useful in eliminating steady-state errors when disturbances are constant [**9**]. However, in practice, most robotic manipulators experience unmodeled dynamics and payload variations that can be perceived as unobserved bounded and time-varying disturbances [**17**][**18**]. Consequently, the performance of PD/PID computed torque controllers deteriorates under such conditions [**15**][**16**].

To overcome these challenges, researchers have explored advanced control techniques that can effectively handle the uncertainties and disturbances encountered in robot manipulators. By developing robust control strategies and integrating adaptive algorithms, it becomes possible to enhance the performance of manipulator control systems and achieve more reliable trajectory tracking in critical applications.

In summary, conventional PD and PID computed-torque controllers face difficulties in addressing uncertainties and disturbances in robot manipulators. Advanced control techniques and adaptive algorithms offer promising avenues to achieve robust and accurate trajectory tracking in the presence of unmodeled dynamics and payload variations. These advancements pave the way for improved performance and increased applicability of robot manipulators in industrial automation settings.

1.1 State of the Art

In this study, a novel control strategy is proposed to address the limitations of conventional PID computed-torque controllers in dealing with uncertainties. The approach involves designing an adaptive gain tuner that dynamically adjusts the controller gains online to maintain the desired control system performance. The objective is to enable the robot manipulator to track a given trajectory while minimizing tracking errors and ensuring bounded closed-loop dynamics and internal signals.

The key advantage of the adaptive PID gain tuner is its ability to adaptively handle time-variant uncertainties [**9**]. Previous studies have explored various fuzzy-based PID gain tuning methods for robotic manipulators, which achieved good performance in error minimization. However, these methods had drawbacks such as user-defined fuzzy rules and system variables, requiring human intervention and resulting in different outcomes based on different designs [**28**][**31**]. Additionally, they incurred high control costs, which are undesirable for optimal control [**3**][**5**].

With the advancements in machine learning and artificial intelligence, deep reinforcement learning algorithms, such as deep deterministic policy gradient (DDPG), have gained popularity in robot applications and control problems. These algorithms have proven to be effective, and high-quality tools are available for training and implementing neural networks [**2**][**4**][**14**]. Hence, in this work, the PID gain adjustment method is based on DDPG reinforcement learning. This approach combines the deterministic policy gradient algorithm, which operates over continuous action spaces, and the actor-critic approach of Deep Q-Network, which trains target networks using experience replay.

The adopted method takes the joint position and velocity errors of the robot as states and computes optimal PID controller gains as actions. The DDPG agent learns the mapping between tracking errors and the corresponding optimal PID gains automatically, without requiring human intuition as in fuzzy-based tuning. Moreover, DDPG agent-based tuning ensures guaranteed asymptotic convergence of tracking errors.

Several studies have successfully applied DDPG in robotics control, achieving fast convergence and good trajectory tracking with adaptive gains [**19**][**21**]. However, the main difference between those previous works and the current study lies in the implementation of the PID controller with computed-torque control, which incorporates the robot dynamics. Therefore, the UR5e robot is selected as a case study in this work due to the availability of kinematic and dynamic parameters of its joints and links. The UR5e robot is a lightweight six-degree-of-freedom industrial manipulator with revolute joints, actuated by Strain wave gears-equipped brushless AC servo motors.

1.2 Outline

Chapter 2 focuses on deriving the forward kinematics and dynamics of n-link serial robot manipulators. The Denavit-Hartenberg notation is utilized for the forward kinematics, while the Lagrange-Euler formulation is applied for the dynamics analysis.

In Chapter 3, the robot's tracking control problem is formulated, and a detailed discussion and analysis of the PID Computed-torque control stability are presented.

Chapter 4 delves into the fundamentals of deep feedforward neural networks and DDPG reinforcement learning, exploring their principles and applications.

Chapter 5 is dedicated to the development of a DDPG reinforcement learning-based PID gain tuner for PID-Computed torque control of robot manipulators. The chapter elaborates on the design and implementation of the proposed method. Furthermore, the efficiency and robustness of the approach are validated through computer simulation tests on a UR5e robot model under bounded time-varying disturbances.

Lastly, conclusions from the study's results, perspectives, and future research directions in this domain are suggested for further exploration.

Chapter 2

Modeling of Industrial Manipulators

2.1 Introduction to Industrial Manipulators

Industrial manipulators are automated systems that are designed to carry out various tasks in industrial settings. They are used in a variety of industries, including manufacturing, automotive, aerospace, and pharmaceuticals. Industrial manipulators are used to lift, move, and position heavy objects or components, and they can be programmed to perform repetitive tasks with great precision and accuracy. They are used to improve the efficiency and productivity of manufacturing processes and to reduce the risk of injury to workers.

The development of industrial manipulators has been driven by a number of factors, including the need to improve manufacturing efficiency, reduce costs, and enhance worker safety. Industrial manipulators are particularly well-suited to tasks that are repetitive or involve heavy lifting, as they can perform these tasks without becoming fatigued or requiring breaks. They are also able to work in environments that are hazardous or difficult for humans to access, such as high temperatures, radiation, or toxic substances.

2.1.1 Types of Industrial Manipulators

There are several different types of industrial manipulators, each designed to perform specific tasks in different industries. Some of the most common types of industrial manipulators include:

1. Articulated Robots: These are the most common type of industrial manipulators, and they are used in a wide range of industries. Articulated robots consist of a series of connected arms that can be moved in different directions, enabling them to reach a variety of positions.

2. Cartesian Robots: Cartesian robots consist of three linear axes that are arranged perpendicular to each other. They are used for tasks that require precise positioning, such as pick-and-place operations.

3. SCARA Robots : SCARA robots are designed for tasks that require high speed and precision, such as assembly and packaging. They consist of two parallel arms that can move in the horizontal plane.

4. Collaborative Robots : Collaborative robots, also known as cobots, are designed to work alongside humans in industrial settings. They are equipped with sensors and safety features that allow them to work safely alongside humans, making them ideal for tasks that require a high degree of precision and accuracy.

2.1.2 Application Area

Industrial manipulators are used in a wide range of applications across different industries. Some of the most common applications of industrial manipulators include:

1. Material Handling - Industrial manipulators are used to lift, move, and position heavy objects or components in manufacturing processes. They are used in a variety of industries, including automotive, aerospace, and construction..

2. Assembly - Industrial manipulators are used in assembly processes to position components and to perform precise operations such as welding or riveting. They are particularly useful for assembling complex products such as automobiles or aircraft.

3. Packaging - Industrial manipulators are used in packaging processes to pick and place products into containers or onto pallets. They can be programmed to handle a wide range of products, from small electronic components to large appliances.

4. Inspection - Industrial manipulators are used in inspection processes to detect defects or anomalies in products or components. They can be equipped with sensors and cameras that enable them to detect and respond to changes in their environment.

2.2 Rigid-body Motions and Transformations

When it comes to the kinematic modeling of robot manipulators, rigid-body motions in three-dimensional space play an essential role in all aspects of robotic manipulation, including the establishment of various coordinate frames to represent the positions and orientations (configurations) of rigid bodies, and the transformations among these coordinate frames through operations of rotation and translation. Homogeneous transformations combine the operations of rotation and translation into a single matrix multiplication, and are used in section 2.3 to derive the forward kinematic equations of rigid manipulators.

In this section, we begin by representing translations of points and vectors in a Euclidean space associated with several coordinate frames. Following this, we introduce the concept of a rotation matrix to perform rotations and represent relative orientations among coordinate frames. We then combine these two concepts to construct homogeneous transformation matrices, which can be used to simultaneously describe the position and orientation of one coordinate frame with respect to another. Furthermore, homogeneous transformation matrices can be used to perform coordinate transformations. Such approach allow us to represent various quantities in different coordinate frames. which are very useful and straightforward to derive both kinematic and dynamic equations of robot manipulators.

2.2.1 Translations in space

In robotics, trajectories are often defined using Cartesian coordinates, which makes analytic reasoning the typical used representation. This approach requires the choice of a reference coordinate frame that consists of an origin point in space, and three orthogonal coordinate axes for three-dimensional spaces. Then, the reasoning is performed via algebraic manipulations (matrices operations), to obtain the transformed coordinates. Consider a real vector denoted by ${}^{i}\boldsymbol{p}_1$, pointing to an arbitrary point in space p_1, and relative to frame $\{i\}$. Then, the transformed coordinates of vector ${}^{i}\boldsymbol{p}_1$, via a translation vector ${}^{i}\boldsymbol{t}$, can be expressed by the vector ${}^{i}\boldsymbol{p}_2$ relative to frame $\{i\}$ as

$$ {}^{i}\boldsymbol{p}_2 = {}^{i}\boldsymbol{p}_1 + {}^{i}\boldsymbol{t}. \tag{2.1} $$

Hence, the inverse transformation can be simply obtained by subtracting the translation vector from the two sides of equation (2.1) as

$$^{i}\boldsymbol{p}_1 = {}^{i}\boldsymbol{p}_2 - {}^{i}\boldsymbol{t}. \tag{2.2}$$

2.2.2 Rotation matrix

Rotations in space can be represented by a rotation matrix $\boldsymbol{R} \in SO(3)$, where the set $SO(3)$ is called special orthogonal group in three-dimensions space, and defined as

$$SO\left(3\right) = \left\{ \boldsymbol{R} \in \mathbb{R}^{3\times 3} \,\middle|\, \boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{I}, \det \boldsymbol{R} = 1 \right\}, \tag{2.3}$$

which satisfy the following properties:

Proposition 2.2.1. *The inverse of a rotation matrix $\boldsymbol{R} \in SO(3)$ is also a rotation matrix, such that $\boldsymbol{R}^{-1} = \boldsymbol{R}^{T}$.*

Proof. The condition $\boldsymbol{R}^{T}\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^{T} = \boldsymbol{I}$ implies that $\boldsymbol{R}^{T} = \boldsymbol{R}^{-1}$. Since $\det \boldsymbol{R}^{T} = \det \boldsymbol{R} = 1$, $\boldsymbol{R}^{-1}$ is also a rotation matrix. □

Proposition 2.2.2. *The product of two rotation matrices is a rotation matrix.*

Proof. Given $\boldsymbol{R}_1, \boldsymbol{R}_2 \in SO(3)$, $\boldsymbol{R}_1\boldsymbol{R}_2$ satisfies $\left(\boldsymbol{R}_1\boldsymbol{R}_2\right)\left(\boldsymbol{R}_1\boldsymbol{R}_2\right)^{T} = \left(\boldsymbol{R}_1\boldsymbol{R}_2\right)^{T}\left(\boldsymbol{R}_1\boldsymbol{R}_2\right) = \boldsymbol{I}$. Further, $\det \boldsymbol{R}_1\boldsymbol{R}_2 = \det \boldsymbol{R}_1 \det \boldsymbol{R}_2 = 1$. Thus $\boldsymbol{R}_1\boldsymbol{R}_2$ satisfies the conditions for a rotation matrix. □

Proposition 2.2.3. *Multiplication of rotation matrices is associative, $\left(\boldsymbol{R}_1\boldsymbol{R}_2\right)\boldsymbol{R}_3 = \boldsymbol{R}_1\left(\boldsymbol{R}_2\boldsymbol{R}_3\right)$, but generally not commutative, $\boldsymbol{R}_1\boldsymbol{R}_2 \neq \boldsymbol{R}_2\boldsymbol{R}_1$.*

Proof. Associativity and nocommutativity follows from the properties of matrix multiplication in linear algebra. □

Proposition 2.2.4. *For any vector $\boldsymbol{u} \in \mathbb{R}^3$ and $\boldsymbol{R} \in SO(3)$, the vector $\boldsymbol{R}\boldsymbol{u}$ has the same length as $\boldsymbol{u}$.*

Proof. This follows from $\|\boldsymbol{R}\boldsymbol{u}\|^2 = \left(\boldsymbol{R}\boldsymbol{u}\right)^{T}\boldsymbol{R}\boldsymbol{u} = \boldsymbol{u}^{T}\boldsymbol{R}^{T}\boldsymbol{R}\boldsymbol{u} = \boldsymbol{u}^{T}\boldsymbol{u} = \|\boldsymbol{u}\|^2$. □

Uses of Rotation Matrices

The uses of a rotation matrix can be divided to three essential actions, which are representing an orientation; changing the reference frame in which a vector or a frame is represented and rotating a vector or a frame. To illustrate the effect of each action, consider a point in space p and three different arbitrary coordinate frames – $\{i\}$, $\{j\}$, and $\{k\}$ – located at the same origin and representing the same space. The orientation of frame $\{j\}$ relative to frame $\{i\}$ can be represented with a rotation matrix notated as $^{i}\boldsymbol{R}_j$. Similarly, $^{j}\boldsymbol{R}_k$ is referring to the orientation of frame $\{k\}$ relative to frame $\{j\}$. The vectors that represent the location of point p relative to frames $\{i\}$ and $\{j\}$ are notated as $^{i}\boldsymbol{p}$ and $^{j}\boldsymbol{p}$, respectively. The orientation of a frame relative to another can be obtained using successive rotations as

$$^{i}\boldsymbol{R}_k = {}^{i}\boldsymbol{R}_j {}^{j}\boldsymbol{R}_k. \tag{2.4}$$

Furthermore, the reference frame of a vector can be changed by a rotation matrix as

$$^{i}\boldsymbol{p} = {}^{i}\boldsymbol{R}_j {}^{j}\boldsymbol{p}. \tag{2.5}$$

Finally, a rotation matrix $\boldsymbol{R}$ can be used to rotate a vector $\boldsymbol{v}$ as

$$\boldsymbol{v}' = \boldsymbol{R}\boldsymbol{v}. \tag{2.6}$$

Rodrigues Formula of Rotation

Rotation matrices can constructed using Rodrigues formula of rotation. By defining a rotation angle θ, and a rotation axis represented by unit vector $\boldsymbol{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$, the correspondent rotation matrix according to the right-hand rule can be expressed as

$$\boldsymbol{R}_{\boldsymbol{u}}(\theta) = \boldsymbol{I} + \sin(\theta)\,\boldsymbol{S}(\boldsymbol{u}) + [1 - \cos(\theta)]\,\boldsymbol{S}^2(\boldsymbol{u}), \tag{2.7}$$

or in terms of the matrix exponential as

$$\boldsymbol{R}_{\boldsymbol{u}}(\theta) = e^{\theta \boldsymbol{S}(\boldsymbol{u})}, \tag{2.8}$$

where $\boldsymbol{S}(\boldsymbol{u}) \in \mathbb{R}^{3\times 3}$ is a skew-symmetric matrix that satisfies $\boldsymbol{S}^T(\boldsymbol{u}) = -\boldsymbol{S}(\boldsymbol{u})$, and given by

$$\boldsymbol{S}(\boldsymbol{u}) = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}. \tag{2.9}$$

Skew-symmetric matrices possess useful properties that will be exploited to derive velocity kinematics. Among these properties are

1. Linearity, that is

$$\boldsymbol{S}(\alpha\boldsymbol{a} + \beta\boldsymbol{b}) = \alpha\boldsymbol{S}(\boldsymbol{a}) + \beta\boldsymbol{S}(\boldsymbol{b}), \tag{2.10}$$

   for any vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$ and scalars $\alpha, \beta \in \mathbb{R}$.

2. For any vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$,

$$\boldsymbol{a} \times \boldsymbol{b} = \boldsymbol{S}(\boldsymbol{a})\,\boldsymbol{b} = -\boldsymbol{S}(\boldsymbol{b})\,\boldsymbol{a}, \tag{2.11}$$

   where the operator $\times$ denotes the vector cross product.

3. For any rotation matrix $\boldsymbol{R} \in SO(3)$ and vector $\boldsymbol{a} \in \mathbb{R}^3$,

$$\boldsymbol{S}(\boldsymbol{R}\boldsymbol{a}) = \boldsymbol{R}\boldsymbol{S}(\boldsymbol{a})\,\boldsymbol{R}^T. \tag{2.12}$$

Time Derivative of Rotation Matrix

Consider a rotation matrix $\boldsymbol{R}$ that is a function of time varying angle $\theta(t)$ and a fixed rotation axis $\boldsymbol{u}$. Time derivative of $\boldsymbol{R}_{\boldsymbol{u}}(\theta(t))$ can be written as

$$\dot{\boldsymbol{R}}_{\boldsymbol{u}}(\theta(t)) = \frac{d\boldsymbol{R}_{\boldsymbol{u}}(\theta(t))}{dt} = \frac{\partial \boldsymbol{R}_{\boldsymbol{u}}(\theta(t))}{\partial\theta(t)}\dot{\theta}(t). \tag{2.13}$$

Using the matrix exponential form in (2.8), we get

$$\dot{\boldsymbol{R}}_{\boldsymbol{u}}(\theta(t)) = \dot{\theta}(t)\boldsymbol{S}(\boldsymbol{u})\,e^{\theta(t)\boldsymbol{S}(\boldsymbol{u})} = \boldsymbol{S}\left(\dot{\theta}(t)\boldsymbol{u}\right)\boldsymbol{R}_{\boldsymbol{u}}(\theta(t)). \tag{2.14}$$

2.2.3 Homogeneous transformation matrix

Homogeneous transformation matrix simplifies the handling of rigid motions that consist of pure translations together with pure rotations, as it reduces the computation of rigid motions compositions in the form of matrix multiplications. A homogeneous transformation matrix $\boldsymbol{T}$, belongs to Special Euclidean group in three-dimensions space $SE(3)$, which is defined as

$$SE\left(3\right)=\left\{\boldsymbol{T}\left|\boldsymbol{T}=\left[\begin{array}{c|c}\boldsymbol{R} & \boldsymbol{d}\\ \hline \boldsymbol{0} & 1\end{array}\right],\boldsymbol{R}\in SO\left(3\right),\boldsymbol{d}\in\mathbb{R}^{3}\right.\right\}. \tag{2.15}$$

Using Banachiewicz-Schur identity, and the fact that $\boldsymbol{R}^{-1}=\boldsymbol{R}^{T}$ according to Proposition 2.2.1, we can easly show that the inverse of a homogeneous transformation matrix $\boldsymbol{T}^{-1}$ is given by

$$\boldsymbol{T}^{-1}=\left[\begin{array}{c|c}\boldsymbol{R}^{T} & -\boldsymbol{R}^{T}\boldsymbol{d}\\ \hline \boldsymbol{0} & 1\end{array}\right]. \tag{2.16}$$

Given that $\boldsymbol{R}^{T}\in SO(3)$, and $\boldsymbol{R}^{T}\boldsymbol{d}\in\mathbb{R}^{3}$, then" $\boldsymbol{T}^{-1}\in SE(3)$. In other words, $\boldsymbol{T}^{-1}$ is also a homogeneous transformation matrix.

We can simply prove that the product of two homogeneous transformation matrices is a homogeneous transformation matrix. Given $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$, we have

$$\boldsymbol{T}_1\boldsymbol{T}_2=\left[\begin{array}{c|c}\boldsymbol{R}_1 & \boldsymbol{d}_1\\ \hline \boldsymbol{0} & 1\end{array}\right]\left[\begin{array}{c|c}\boldsymbol{R}_2 & \boldsymbol{d}_2\\ \hline \boldsymbol{0} & 1\end{array}\right]=\left[\begin{array}{c|c}\boldsymbol{R}_1\boldsymbol{R}_2 & \boldsymbol{R}_1\boldsymbol{d}_2+\boldsymbol{d}_1\\ \hline \boldsymbol{0} & 1\end{array}\right]. \tag{2.17}$$

Since $\boldsymbol{R}_1\boldsymbol{R}_2\in SO(3)$, and $\boldsymbol{R}_1\boldsymbol{d}_2+\boldsymbol{d}_1\in\mathbb{R}^{3}$, then, the conditions of $SE(3)$ are satisfied.

Uses of Homogeneous Transformation Matrices

Analogically to a rotation matrix, homogeneous transformation matrices can be used to represent a configuration (position and orientation); changing the reference frame in which a vector or a frame is represented and transforming a vector or a frame. Consider a point in space p and three different arbitrary coordinate frames – $\{i\}$, $\{j\}$, and $\{k\}$ – located at different origins and representing the same space. The configuration of frame $\{j\}$ relative to frame $\{i\}$ is notated as ${}^{i}\boldsymbol{T}_{j}$. Similarly, ${}^{j}\boldsymbol{T}_{k}$ is referring to the configuration of frame $\{k\}$ relative to frame $\{j\}$. The vectors that represent the location of point p relative to frames $\{i\}$ and $\{j\}$ are notated as ${}^{i}\boldsymbol{p}$ and ${}^{j}\boldsymbol{p}$, respectively. The configuration of a frame relative to another can be obtained using successive matrix multiplication as

$${}^{i}\boldsymbol{T}_{k}={}^{i}\boldsymbol{T}_{j}{}^{j}\boldsymbol{T}_{k}. \tag{2.18}$$

Furthermore, the reference frame of a vector can be changed by a homogeneous transformation matrix as

$$\left[\begin{array}{c}{}^{i}\boldsymbol{p}\\ \hline 1\end{array}\right]={}^{i}\boldsymbol{T}_{j}\left[\begin{array}{c}{}^{j}\boldsymbol{p}\\ \hline 1\end{array}\right]. \tag{2.19}$$

Finally, a homogeneous transformation matrix $\boldsymbol{T}$ can be used to transform a vector $\boldsymbol{v}$ as

$$\left[\begin{array}{c}\boldsymbol{v}'\\ \hline 1\end{array}\right]=\boldsymbol{T}\left[\begin{array}{c}\boldsymbol{v}\\ \hline 1\end{array}\right]. \tag{2.20}$$

2.3 Forward kinematics

Forward kinematics of a robot manipulator determine the position and orientation of end-effector in terms of joints variables without considering the forces and torques of the actuators. In order to derive the forward kinematics, a coordinate system has to be rigidly assigned to each joint and the link that is attached to it. The Denavit-Hartenberg convention and other definitions facilitate the assignment of the coordinate frames in a systematic manner and are presented in this section.

2.3.1 Definitions

A serial chain robot manipulator consists of n joints and and $n+1$ links, where each joint connects two links. Joints are enumerated from 1 to n, and links from 0 to n, starting from the inertial frame or the robot's base. Following this convention, joint i connects link $i-1$ to link i. When joint i is under actuation, link i will move, except for the base, which does not move when the joints are actuated. Each joint i, is associated with a joint variable, denoted by q_i. In case the joint is revolute, then q_i is the angle of rotation θ_i. Similarly, q_i represents the joint displacement d_i, if the joint is prismatic. We can write

$$q_i = \sigma_i \theta_i + (1 - \sigma_i)\, d_i, \tag{2.21}$$

where

$$\sigma_i = \begin{cases} 1 & \text{If joint } i \text{ is revolute} \\ 0 & \text{If joint } i \text{ is prismatic} \end{cases}.$$

Homogeneous transformation matrices are essential approach for deriving kinematics equations. ${}^{i-1}\boldsymbol{T}_i\,(q_i)$ is a homogeneous transformation matrix that is a function of joint variable q_i, which describes the position and orientation of the frame $\{i\}$ relative to the frame $\{i-1\}$, and is therefore of the form

$${}^{i-1}\boldsymbol{T}_i\,(q_i) = \left[\begin{array}{c|c} {}^{i-1}\boldsymbol{R}_i\,(q_i) & {}^{i-1}\boldsymbol{o}_i\,(q_i) \\ \hline \boldsymbol{0} & 1 \end{array}\right]. \tag{2.22}$$

${}^{i}\boldsymbol{T}_j\,(q_{i+1}, q_{i+2}, \ldots, q_j)$ is a homogeneous transformation matrix that expresses the position and orientation of the frame $\{j\}$ relative to the frame $\{i\}$, and written as

$${}^{i}\boldsymbol{T}_j\,(q_{i+1}, q_{i+2}, \ldots, q_j) = \begin{cases} {}^{i}\boldsymbol{T}_{i+1}\,(q_{i+1})\,{}^{i+1}\boldsymbol{T}_{i+2}\,(q_{i+2}) \cdots {}^{j-1}\boldsymbol{T}_j\,(q_j) & i < j \\ \boldsymbol{I} & i = j \\ {}^{i}\boldsymbol{T}_j^{-1}\,(q_{i+1}, q_{i+2}, \ldots, q_j) & i > j \end{cases}. \tag{2.23}$$

The rotation matrix ${}^{i}\boldsymbol{R}_j\,(q_{i+1}, q_{i+2}, \ldots, q_j)$ that expresses the orientation of frame $\{j\}$ relative to the frame $\{i\}$ is given by

$${}^{i}\boldsymbol{R}_j\,(q_{i+1}, q_{i+2}, \ldots, q_j) = \begin{cases} {}^{i}\boldsymbol{R}_{i+1}\,(q_{i+1})\,{}^{i+1}\boldsymbol{R}_{i+2}\,(q_{i+2}) \cdots {}^{j-1}\boldsymbol{R}_j\,(q_j) & i < j \\ \boldsymbol{I} & i = j \\ {}^{i}\boldsymbol{R}_j^{T}\,(q_{i+1}, q_{i+2}, \ldots, q_j) & i > j \end{cases}. \tag{2.24}$$

The vector ${}^{i}\boldsymbol{o}_j\,(q_{i+1}, \ldots, q_j)$ that expresses the position of frame $\{j\}$ relative to the frame $\{i\}$ is given by the recursive formula as

$${}^{i}\boldsymbol{o}_j\,(q_{i+1}, \ldots, q_j) = {}^{i}\boldsymbol{o}_{j-1}\,(q_{i+1}, \ldots, q_{j-1}) + {}^{i}\boldsymbol{R}_{j-1}\,(q_{i+1}, \ldots, q_{j-1})\,{}^{j-1}\boldsymbol{o}_j\,(q_j). \tag{2.25}$$

2.3.2 Denavit-Hartenberg Convention

The Denavit-Hartenberg (D-H) Convention is a commonly used approach to describe the kinematics of robot manipulators. It was introduced to represent the kinematic structure of articulated mechanical systems. The D-H method is based on the idea of assigning coordinate systems to each link of the robot in a systematic manner, and using these coordinate systems to describe the position and orientation of each link relative to the previous one [**8**]. This method is widely used due to its simplicity, generality, and applicability to a broad range of robot configurations. The coordinate frame $\{i\}$ that are assigned to the joint j_i and link l_i must meet the following requirements:

1. The axis $\boldsymbol{z}_i$ is the axis of revolution of joint j_{i+1}.
2. The axis $\boldsymbol{x}_i$ is perpendicular to the axis $\boldsymbol{z}_{i-1}$ and $\boldsymbol{z}_i$.
3. The axis $\boldsymbol{x}_i$ intersects the axis $\boldsymbol{z}_{i-1}$.
4. All frames must be right-handed coordinates.

The D-H parameters define the geometry and kinematics of a a robot manipulator, and they are used to establish the kinematic model of the robot. The parameters are defined as follows:

1. Link length a_i: The distance along $\boldsymbol{x}_i$ from the intersection of the $\boldsymbol{x}_i$ and $\boldsymbol{z}_{i-1}$ axes to $\boldsymbol{o}_i$.
2. Link twist α_i: the angle from $\boldsymbol{z}_{i-1}$ to $\boldsymbol{z}_i$ taken about $\boldsymbol{x}_i$.
3. Link offset d_i: distance along $\boldsymbol{z}_{i-1}$ from $\boldsymbol{o}_{i-1}$ to the intersection of the $\boldsymbol{x}_i$ and $\boldsymbol{z}_{i-1}$ axes.
4. Joint angle θ_i: the angle from $\boldsymbol{x}_{i-1}$ to $\boldsymbol{x}_i$ measured about $\boldsymbol{z}_{i-1}$.

Using these four parameters, the position and orientation of each link relative to the previous link can be described. The homogeneous transformation matrix associated with the i^{th} link is given by [**35**]

$$
\begin{aligned}
{}^{i-1}\boldsymbol{T}_i &= \mathrm{Trans}_{\boldsymbol{z}_{i-1}}(d_i)\mathrm{Rot}_{\boldsymbol{z}_{i-1}}(\theta_i)\mathrm{Trans}_{\boldsymbol{x}_i}(a_i)\mathrm{Rot}_{\boldsymbol{x}_i}(\alpha_i) \\
&= \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha_i)\sin(\theta_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix},
\end{aligned} \tag{2.26}
$$

where

$$
\mathrm{Trans}_{\boldsymbol{z}_{i-1}}(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathrm{Rot}_{\boldsymbol{z}_{i-1}}(\theta_i) = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
\mathrm{Trans}_{\boldsymbol{x}_i}(a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathrm{Rot}_{\boldsymbol{x}_i}(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.
$$

Finally, the homogeneous transformation matrix ${}^0\boldsymbol{T}_n$ that describes the configuration of the end-effoctor relative to the base coordinate system can be computed according to (2.23).

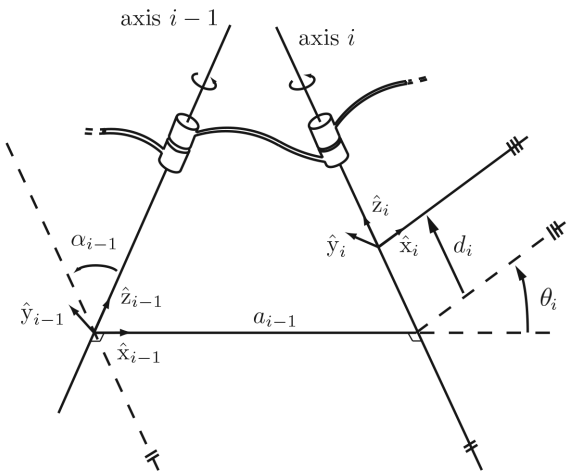

Figure 2.1: Illustration of the Denavit–Hartenberg parameters.

2.4 Velocity kinematics

Velocity kinematics, also known as geometric or differential kinematics, are a fundamental concept in robotics that describes the motion of a robot's end effector as a function of its joint variables. It deals with the study of how the end effector's velocity and acceleration are related to the joint velocities and accelerations. Velocity kinematics of a robot manipulator can be obtained using the forward kinematic equations as a basis, which define a mapping function between the end effector configuration and the joint positions. The velocity relationships are then determined by the Jacobian of this function. The manipulator Jacobian represents the instantaneous transformation between the joint velocities and the linear and angular velocities of the end effector.

2.4.1 Angular Velocity

When a rigid body purely rotates about a fixed axis, every point of the body moves in a circle. The centers of these circles belong to the axis of rotation. As the body rotates, a perpendicular from any point of the body to the axis sweeps out an angle θ, and this angle is the same for all body point . If $\boldsymbol{u} \in \mathbb{R}^3$ is a unit vector in the direction of the axis of rotation, then the angular velocity can be expressed in terms of the time derivative of θ as

$$\boldsymbol{\omega} = \dot{\theta}\boldsymbol{u}. \tag{2.27}$$

Generally, We are seeking to find the composition of angular velocity due to the relative rotation of several coordinate frames. We donete ${}^i\boldsymbol{\omega}_j$ the angular velocity vector corresponding to the derivative of ${}^i\boldsymbol{R}_j$ according to (2.14), expressed as

$${}^i\dot{\boldsymbol{R}}_j = \boldsymbol{S}\left({}^i\boldsymbol{\omega}_j\right){}^i\boldsymbol{R}_j. \tag{2.28}$$

If we take the time derivative of the product ${}^i\boldsymbol{R}_j{}^j\boldsymbol{R}_k$, we get

$$\tfrac{d}{dt}\left({}^i\boldsymbol{R}_j{}^j\boldsymbol{R}_k\right) = {}^i\dot{\boldsymbol{R}}_j{}^j\boldsymbol{R}_k + {}^i\boldsymbol{R}_j{}^j\dot{\boldsymbol{R}}_k = \boldsymbol{S}\left({}^i\boldsymbol{\omega}_j + {}^i\boldsymbol{R}_j{}^j\boldsymbol{\omega}_k\right){}^i\boldsymbol{R}_k \ . \tag{2.29}$$

On the other hand, we have

$${}^i\dot{\boldsymbol{R}}_k = \boldsymbol{S}\left({}^i\boldsymbol{\omega}_k\right){}^i\boldsymbol{R}_k. \tag{2.30}$$

Taking the linearity property of $\boldsymbol{S}$ as in (2.10), the resultant composition of angular velocity vectors can be given as

$$^{i}\boldsymbol{\omega}_{k} = {}^{i}\boldsymbol{\omega}_{j} + {}^{i}\boldsymbol{R}_{j}{}^{j}\boldsymbol{\omega}_{k}. \tag{2.31}$$

The equivalent recursive formula can be written as

$$^{i}\boldsymbol{\omega}_{j} = {}^{i}\boldsymbol{\omega}_{j-1} + {}^{i}\boldsymbol{R}_{j-1}{}^{j-1}\boldsymbol{\omega}_{j}. \tag{2.32}$$

According to the D-H convention, rotation occurs about the $\boldsymbol{z}_i$ axis associated to each link in case of a revolute joint (i.e. $q_i = \theta_i$). The angular velocity vector of the i^{th}-link relative to frame $\{i-1\}$ can then be expressed as

$$^{i-1}\boldsymbol{\omega}_{i} = \sigma_i \dot{q}_i \boldsymbol{z}_{i-1}^{i-1} = \sigma_i \dot{q}_i \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.33}$$

2.4.2 Geometric Jacobian

The geometric Jacobian matrix maps the joint velocities to the linear and angular velocities of the end-effector in Cartesian space. It is essential tool in many aspects of robotics, including constructing the dynamics equations of the robot, motion planning, trajectory optimization, and control. It can also be used to study the singularity of the robot, which occurs when the Jacobian becomes rank-deficient, meaning that there are certain joint configurations where the robot loses control over its end-effector. The angular part of the geometric Jacobian matrix $\boldsymbol{J}_{{}^{i}\boldsymbol{\omega}_{j}} \in \mathbb{R}^{3\times n}$ can be recursively obtained as

$$\boldsymbol{J}_{{}^{i}\boldsymbol{\omega}_{j}} = \boldsymbol{J}_{{}^{i}\boldsymbol{\omega}_{j-1}} + {}^{i}\boldsymbol{R}_{j-1}\boldsymbol{J}_{{}^{j-1}\boldsymbol{\omega}_{j}}, \tag{2.34}$$

where

$$\boldsymbol{J}_{{}^{j-1}\boldsymbol{\omega}_{j}} = \begin{bmatrix} \boldsymbol{0}_{3\times(j-1)} & \sigma_j \boldsymbol{z}_{j-1}^{j-1} & \boldsymbol{0}_{3\times(n-j)} \end{bmatrix}. \tag{2.35}$$

Similarly, the linear part of the geometric Jacobian matrix $\boldsymbol{J}_{{}^{i}\boldsymbol{o}_{j}} \in \mathbb{R}^{3\times n}$ can be optained by differenting (2.25) with respect joint positions vector, which give a recursive form as

$$\boldsymbol{J}_{{}^{i}\boldsymbol{o}_{j}} = \boldsymbol{J}_{{}^{i}\boldsymbol{o}_{j-1}} + \boldsymbol{S}\left({}^{i}\boldsymbol{o}_{j-1} - {}^{i}\boldsymbol{o}_{j}\right)\boldsymbol{J}_{{}^{i}\omega_{j}} + {}^{i}\boldsymbol{R}_{j-1}\boldsymbol{J}_{{}^{j-1}\boldsymbol{o}_{j}}, \tag{2.36}$$

where

$$\boldsymbol{J}_{{}^{j-1}\boldsymbol{o}_{j}} = \begin{bmatrix} & -\sigma_j a_j \sin(\theta_j) & \\ \boldsymbol{0}_{3\times(j-1)} & \sigma_j a_j \cos(\theta_j) & \boldsymbol{0}_{3\times(n-j)} \\ & 1-\sigma_j & \end{bmatrix}. \tag{2.37}$$

2.4.3 Jacobian of an Arbitrary Point on a Link

For the derivation of dynamics equations of motion, it is necessary to derive the jacobian matrix associated with the center of mass of each link. Given a vector $^{i}\boldsymbol{c}_{j} \in \mathbb{R}^{3}$ donating the position of center of mass attached to j^{th}-link with respect the frame $\{i\}$, the transformation equation can be written as

$$^{i}\boldsymbol{c}_{j} = {}^{i}\boldsymbol{o}_{j} + {}^{i}\boldsymbol{R}_{j}{}^{j}\boldsymbol{c}_{j}. \tag{2.38}$$

Taking the partial derivative with respect to joint positions vector, the linear part of the geometric jacobian $\boldsymbol{J}_{{}^{i}\boldsymbol{c}_{j}} \in \mathbb{R}^{3\times n}$ associated to the center of mass can be expressed as

$$\boldsymbol{J}_{{}^{i}\boldsymbol{c}_{j}} = \boldsymbol{J}_{{}^{i}\boldsymbol{o}_{j}} + \boldsymbol{S}\left({}^{i}\boldsymbol{o}_{j} - {}^{i}\boldsymbol{c}_{j}\right)\boldsymbol{J}_{{}^{i}\omega_{j}}. \tag{2.39}$$

Note that the center of mass vector must be computed sparely for each link as it is not directly given using the homogeneous transformation matrices.

2.4.4 Velocity Transformation

The transformation of velocities between two coordinate frames $\{i\}$ and $\{j\}$ can be obtained as

$$ {}^{i}\boldsymbol{H}_{j} = \left[\begin{array}{c|c} {}^{i}\boldsymbol{R}_{j} & -{}^{i}\boldsymbol{R}_{j}^{T}\boldsymbol{S}\left({}^{i}\boldsymbol{r}_{i,j}\right) \\ \hline \boldsymbol{0} & {}^{i}\boldsymbol{R}_{j}^{T} \end{array}\right], \tag{2.40} $$

where ${}^{i}\boldsymbol{r}_{i,j} \in \mathbb{R}^3$ is a vector pointing from frame $\{i\}$ to $\{j\}$, expressed in frame $\{i\}$.

2.5 Dynamics

The dynamics of a robot manipulator refer to describing the relationship between the robot's motion and the forces, torques that cause the joints to move. Understanding the dynamics is essential for controlling the robot's movement accurately and efficiently. Two most commonly used methods for the derivation of the dynamic equations of motion are the Euler-Lagrange equations and the Newton-Euler formulation. The Euler-Lagrange is an energy-based method (i.e. kinetic and potential energy) which is derived from D'Alembert's principle and the principle of virtual work, whereas the Newton-Euler method is a recursive formulation of the dynamic equations that is based on Newton's second law of motion (i.e. $F = ma$).

The Euler-Langrange method is adopted in this work. due to it's useful advantages over Newton-Euler formulation. The Euler-Lagrange method allows for the use of generalized coordinates, which can simplify the mathematical representation of complex systems. Generalized coordinates are independent variables that describe the configuration of a system, and they can be chosen to simplify the equations of motion, especially for systems with constraints or complex geometries. In addition, Euler-Lagrange method naturally incorporates symmetries and conservation laws into the derivation of the equations of motion. By utilizing the Lagrangian formalism, one can exploit symmetries such as translational invariance, rotational invariance, and time invariance to derive corresponding conservation laws, such as the conservation of linear momentum, angular momentum, and energy.

2.5.1 Inertia Tensor

To derive the dynamic equations of a robot manipulator, the inertia tensor associated to each link must be defined. Generally, an inertia tensor has the form

$$ \boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}, \tag{2.41} $$

where

$$ \begin{aligned} I_{xx} &= \int\int\int \left(y^2 + z^2\right)\rho\left(x, y, z\right) dx\, dy\, dz, \\ I_{yy} &= \int\int\int \left(x^2 + z^2\right)\rho\left(x, y, z\right) dx\, dy\, dz, \\ I_{zz} &= \int\int\int \left(x^2 + y^2\right)\rho\left(x, y, z\right) dx\, dy\, dz, \end{aligned} $$

and

$$I_{xy} = I_{yx} = -\int\int\int xy\rho\,(x,y,z))dx\,dy\,dz,$$
$$I_{xz} = I_{zx} = -\int\int\int xz\rho\,(x,y,z))dx\,dy\,dz,$$
$$I_{yz} = I_{zy} = -\int\int\int yz\rho\,(x,y,z))dx\,dy\,dz,$$

where $\rho\,(x,y,z)$ is the mass density of the link, expressed as a function of position. The diagonal terms of the inertia tensor I_{xx}, I_{yy} and I_{zz} are the principal moments of inertia about the x, y, and z-axes, respectively, while the off-diagonal terms I_{xy}, I_{xz}, I_{yz}, etc, are the cross products of inertia. If the mass distribution of the link is symmetric with respect to the link-attached frame, then the cross products of inertia are equal to zero.

The inertia matrix expressed in the link-attached frame is a constant matrix, meaning that it is independent of the motion of the manipulator. Given $\boldsymbol{I}$ the inertia tensor expressed in the link-attached frame, the inertia tensor $\boldsymbol{\mathcal{I}}$ expressed relative the base frame can be obtained via a similarity transformation as

$$\boldsymbol{\mathcal{I}} = \boldsymbol{R}\boldsymbol{I}\boldsymbol{R}^T. \tag{2.42}$$

2.5.2 Kinetic Energy for an n-Link Robot

In order to obtain the dynamics model, the kinetic energy and potential energy terms of a robot with rigid links are derived using the Denavit–Hartenberg joint variables as generalized coordinates of the concerned system. Based on the parallel axis theorem, the kinetic energy of a rigid body is the sum of two terms, the translational kinetic energy obtained by considering the entire mass of the body to be a point mass located at the center of mass, and the rotational kinetic energy of the body about the center of mass. The kinetic energy of the rigid body is then expressed as

$$\mathcal{K} = \frac{1}{2}m\boldsymbol{v}^T\boldsymbol{v} + \frac{1}{2}\boldsymbol{\omega}^T\boldsymbol{\mathcal{I}}\boldsymbol{\omega}, \tag{2.43}$$

where m is the mass of the body, $\boldsymbol{v}$ and $\boldsymbol{\omega}$ are the linear and angular velocity vectors, respectively, and $\boldsymbol{\mathcal{I}}$ is the inertia tensor of the body. Using this expression, the total kinetic energy of n-link serial manipulator can be generalized as [**6**]

$$\begin{aligned}\mathcal{K} &= \frac{1}{2}\sum_{i=1}^{n}\left(m_i\dot{\boldsymbol{q}}^T\boldsymbol{J}_{v,c_i}^T\left(\boldsymbol{q}\right)\boldsymbol{J}_{v,c_i}\left(\boldsymbol{q}\right)\dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T\boldsymbol{J}_{\omega,c_i}^T\left(\boldsymbol{q}\right)\boldsymbol{R}_i\left(\boldsymbol{q}\right)\boldsymbol{\mathcal{I}}_i\boldsymbol{R}_i{}^T\left(\boldsymbol{q}\right)\boldsymbol{J}_{\omega,c_i}\left(\boldsymbol{q}\right)\dot{\boldsymbol{q}}\right)\\ &= \frac{1}{2}\dot{\boldsymbol{q}}^T\left[\sum_{i=1}^{n}\left(m_i\boldsymbol{J}_{v,c_i}^T\left(\boldsymbol{q}\right)\boldsymbol{J}_{v,c_i}\left(\boldsymbol{q}\right) + \boldsymbol{J}_{\omega,c_i}^T\left(\boldsymbol{q}\right)\boldsymbol{R}_i\left(\boldsymbol{q}\right)\boldsymbol{\mathcal{I}}_i\boldsymbol{R}_i{}^T\left(\boldsymbol{q}\right)\boldsymbol{J}_{\omega,c_i}\left(\boldsymbol{q}\right)\right)\right]\dot{\boldsymbol{q}}\\ &= \frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{M}\left(\boldsymbol{q}\right)\dot{\boldsymbol{q}}.\end{aligned} \tag{2.44}$$

where $\boldsymbol{q} = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}^T$ and $\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{q}_1 & \dot{q}_2 & \cdots & \dot{q}_n \end{bmatrix}^T$ are vectors of joints position and velocity, respectively; m_i is the mass of the i^{th}-link; $\boldsymbol{J}_{v,c_i}\left(\boldsymbol{q}\right)$, $\boldsymbol{J}_{\omega,c_i}\left(\boldsymbol{q}\right) \in \mathbb{R}^{3\times n}$ are the linear and angular parts of the geometric jacobian matrix from base frame to the mass center of the i^{th}-link as described in equations (2.34)(2.39); $\boldsymbol{R}_i\left(\boldsymbol{q}\right) \in \mathbb{R}^{3\times 3}$ is the rotation matrix that expresses the orientation of frame $\{i\}$ relative to the base frame as defined in (2.24); $\boldsymbol{\mathcal{I}}_i \in \mathbb{R}^{3\times 3}$ is the inertia tensor expressed in the body attached frame of the i^{th}-link; $\boldsymbol{M}\left(\boldsymbol{q}\right) \in \mathbb{R}^{n\times n}$ is symmetric positive-definite mass matrix of the robot manipulator.

2.5.3 Potential Energy for an n-Link Robot

In the case of rigid-body dynamics, the potential energy resulted from elasticity are ignored. Hence, gravitational force is the only considered source of potential energy. The potential energy of the i^{th}-link can be obtained by assuming that the entire mass of the link to be a point mass located at the center of mass and is given by

$$\mathcal{P}_i = m_i \boldsymbol{g}_0 \boldsymbol{c}_i, \tag{2.45}$$

where m_i is the mass of the i^{th}-link; $\boldsymbol{g}_0$ is the vector giving the direction of gravity in the base frame; the vector $\boldsymbol{c}_i$ represents the coordinates of the center of mass of the i^{th}-link relative to base frame as described in (2.38). The total potential energy of the n-link robot manipulator is therefore expressed as

$$\mathcal{P} = \sum_{i=1}^{n} \mathcal{P}_i = \sum_{i=1}^{n} m_i \boldsymbol{g}_0 \boldsymbol{c}_i. \tag{2.46}$$

Note that the potential energy is a function of the joints position and not their velocity.

2.5.4 Equations of Motion

The equations of motion of n-link robot manipulator can be derived using The Euler–Lagrange formulation as

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right)}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial \mathcal{L}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right)}{\partial \boldsymbol{q}} = \boldsymbol{\tau} - \boldsymbol{f}\left(\dot{\boldsymbol{q}}\right) - \boldsymbol{\tau}_d, \tag{2.47}$$

where $\mathcal{L} = \mathcal{K} - \mathcal{P}$ is the Lagrangian of the system; $\boldsymbol{\tau} \in \mathbb{R}^n$ is a vector of input forces and torques provided by joint actuators; $\boldsymbol{\tau}_d \in \mathbb{R}^n$ is a vector of disturbance forces and torques from the external effects and modeling uncertainties; $\boldsymbol{f}\left(\dot{\boldsymbol{q}}\right) \in \mathbb{R}^n$ is a vector representing joints friction. The equations of motion can be rearranged to a common form as [**6**]

$$\boldsymbol{M}\left(\boldsymbol{q}\right)\ddot{\boldsymbol{q}} + \underbrace{\boldsymbol{C}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right)\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{f}\left(\boldsymbol{q}\right)}_{\boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}})} + \boldsymbol{\tau}_d = \boldsymbol{\tau} \tag{2.48}$$

where $\boldsymbol{C}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right) \in \mathbb{R}^{n \times n}$ is the coriolis and centrifugal matrix; $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ is the gravity vector. The elements c_{ij} of the matrix $\boldsymbol{C}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right)$ are defined as [**8**]

$$c_{ij} = \frac{1}{2}\sum_{k=1}^{n}\left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ik}}{\partial q_j} - \frac{\partial m_{kj}}{\partial q_i}\right)\dot{q}_k, \tag{2.49}$$

which are known as the Christoffel symbols of the first kind. Finally, the gravity vector is defined by

$$\boldsymbol{g} = \frac{\partial \mathcal{P}}{\partial \boldsymbol{q}}. \tag{2.50}$$

Different friction models that can be adopted for the robot dynamic equations are discussed in the following subsection.

2.5.5 Friction Models

Friction forces and torques in gearboxes and bearings of joints and actuators may have a significant impact on the manipulator performance. Friction is a complex phenomenon that is the subject of current research. Friction models are a gross attempt to capture the micromechanics behavior of surface contact. Friction models often include a static and a velocity-dependent terms. The presence of a static friction term means that a nonzero torque is required to cause the joint motion. Velocity-dependent friction term indicates that the amount of friction torque increases with increasing velocity of the joint.

Other factors may contribute to the friction, including the load carried by the joint bearings, the time the joint has been at rest, the temperature, etc . The friction in a gearbox often increases as the transmission ratio increases [**36**].

Static Friction

Static friction model can be defined as

$$f_s \leq \mu_s N, \tag{2.51}$$

where f_s is the static friction force; μ_s is the coefficient of static friction, which depends on the nature of the surfaces in contact; N is the normal force, which is the force exerted by a surface perpendicular to the contact area. This model indicates that the static friction force can take any value from zero up to the maximum static friction force. Once the applied force exceeds the maximum static friction force, the body will start moving, and the static friction will be replaced by kinetic friction.

Viscous Friction

Viscous friction model is typically described by

$$f_v = \mu_v \dot{q}, \tag{2.52}$$

where f_v is the force due to viscous friction; μ_v is the coefficient of viscosity or the viscosity of the medium; $\dot{q}$ is the velocity of the body moving through the medium. The magnitude of the force is directly proportional to the velocity of the body and the coefficient of viscosity. This equation represents a simplified model of viscous friction, assuming a linear relationship between the force and velocity. In reality, the relationship between frictional force and velocity can be more complex, depending on factors such as the shape of the body and the properties of the medium.

Coulomb Friction

Coulomb friction describes the frictional force between two surfaces in relative motion. It is given by:

$$f_c = f_s \mathrm{sgn}\left(\dot{q}\right), \tag{2.53}$$

where f_c is the force due to Coulomb friction; f_s is the static friction force; $\dot{q}$ is the velocity of the body moving through the medium. The magnitude of the force is affected by the direction of motion rather than the velocity of the body. Coulomb friction model assumes that the frictional force remains constant as long as the applied force does not exceed the maximum static friction force. Once motion is initiated, the frictional force is described by the kinetic friction coefficient

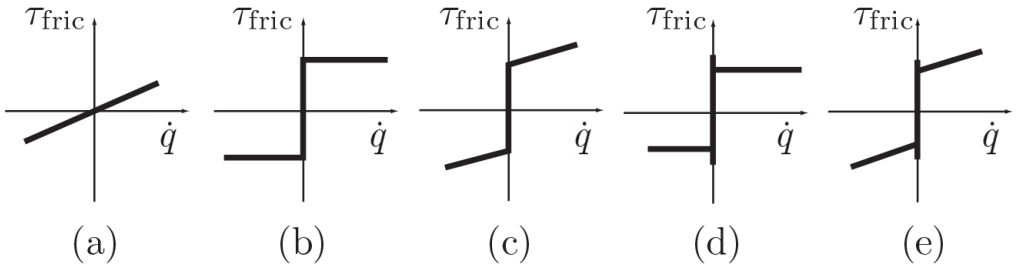

Figure 2.2: Friction models: (a) Viscous friction. (b) Coulomb friction. (c) Static plus viscous friction. (d) Static plus Coulomb friction. (e) Static, Coulomb and viscous friction.

2.5.6 Properties of Robot Dynamic Equations

The equations of motion for an n-link robot contain some important structural properties that can be exploited for developing control algorithms. The most important of which are the skew symmetry, passivity and global bounds on the inertia matrix.

Skew Symmetry and Passivity

The skew symmetry property refers to the relationship between the inertia matrix $\boldsymbol{M}(\boldsymbol{q})$ and the coriolis and centrifugal matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ described in Equation (2.48).

Proposition 2.5.1. *Given an inertia matrix for an n-link robot $\boldsymbol{M}(\boldsymbol{q})$; a coriolis and centrifugal matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ in terms of the elements of $\boldsymbol{M}(\boldsymbol{q})$ according to Equation (2.49). Then, the matrix $\boldsymbol{H}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{M}}(\boldsymbol{q}) - 2\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is skew symmetric, that is, $\boldsymbol{H}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{H}^T(\boldsymbol{q}, \dot{\boldsymbol{q}}) = 0$*

Related to the skew symmetry is the passivity property, which means in this context, that there exists a constant, $\beta \geq 0$, such that

$$\int_0^T \dot{\boldsymbol{q}}^T(t)\boldsymbol{\tau}(t)dt \geq -\beta, \quad \forall\, T > 0, \tag{2.54}$$

which means that the amount of energy dissipated by the system over time interval $[0, T]$ has a lower bound given by $-\beta$.

Bounds on the Inertia Matrix

For a fixed value of the generalized coordinate q, let $0 < \lambda_1(q) < \cdots < \lambda_n(q)$ denote the eigenvalues of inertia matrix $\boldsymbol{M}(\boldsymbol{q})$. These eigenvalues are positive as a result of the positive definiteness property of inertia matrix. Then, it can easily be proven that [**8**]

$$\lambda_1(\boldsymbol{q})\boldsymbol{I}_n < \boldsymbol{M}(\boldsymbol{q}) < \lambda_n(\boldsymbol{q})\boldsymbol{I}_n, \tag{2.55}$$

where $\boldsymbol{I}_n$ is $n \times n$ identity matrix. If all the manipulator's joints are revolute, then the inertia matrix contains only terms of sine and cosine functions. Therefore, the inertia matrix is bounded as a function of the generalized coordinates. As a result, there exist constants independent of $\boldsymbol{q}$, namely, λ_{min} and λ_{max}, that satisfy [**8**]

$$\lambda_{min}\boldsymbol{I}_n < \boldsymbol{M}(\boldsymbol{q}) < \lambda_{max}\boldsymbol{I}_n. \tag{2.56}$$

2.6 Trajectory Planning

Trajectory generation plays a vital role in the field of robotics, particularly in the robot control. Robot manipulators are designed to perform a wide range of tasks, from simple pick-and-place operations to complex assembly processes. To execute these tasks successfully, it is crucial to generate trajectories that guide the manipulator's end-effector through desired positions and orientations accurately and efficiently.

The trajectory generation process involves determining the optimal path that a robot manipulator should follow to accomplish a specific task. This path should consider various factors such as obstacle avoidance, joint limits on velocities, accelerations, or torques, and the manipulator's dynamics. By defining a well-planned trajectory, robots can perform tasks with precision, safety, and improved performance.

In this section, trajectory is considered to be a combination of configurations sequence achieved by the robot (i.e. path), and a time scaling function, which specifies the timing of those configurations . We will discuss point-to-point interpolation in joint space, and optimal-time trajectories along specified paths taking into account the joint actuator limits.

2.6.1 Definitions

A path $\boldsymbol{q}(s(t))$ maps a scalar time function $s(t)$, assumed to be 0 at the start and 1 at the end of the path, reaching a desired point in the robot's configuration space $\boldsymbol{\Theta}$, that is $\boldsymbol{q}:[0,1]\rightarrow\boldsymbol{\Theta}$. Time scaling function $s(t)$ assigns a value of a scale s, such that $s:[0,T]\rightarrow[0,1]$. So as $s(t)$ increases from 0 to 1, the robot moves along the desired path, where $t=0$ and $t=T$ correspond to the timing of initial and final configurations of the desired path. Together, a path and a time scaling function Constitute a trajectory $\boldsymbol{q}(s(t))$, or $\boldsymbol{q}(t)$ for short. The velocity and acceleration terms along the trajectory can be obtained using time differentiation and chain rule as

$$\dot{\boldsymbol{q}}=\frac{d\boldsymbol{q}}{dt}=\frac{d\boldsymbol{q}}{ds}\dot{s}, \tag{2.57}$$

$$\ddot{\boldsymbol{q}}=\frac{d\dot{\boldsymbol{q}}}{dt}=\frac{d\boldsymbol{q}}{ds}\ddot{s}+\frac{d^2\boldsymbol{q}}{ds^2}\dot{s}^2. \tag{2.58}$$

Therefore, each of $\boldsymbol{q}(s(t))$ and $s(t)$ must be twice differentiable to ensure that the acceleration (and hence robot dynamics) is well defined. A path $\boldsymbol{q}(t)$ that starts from initial configuration $\boldsymbol{q}_s$ to an end configuration $\boldsymbol{q}_e$ could be defined in joint space as a linear function in terms of time scaling function $s(t)$ as

$$\boldsymbol{q}(t)=\boldsymbol{q}_s+s(t)(\boldsymbol{q}_e-\boldsymbol{q}_s),\quad s\in[0,1], \tag{2.59}$$

with derivatives

$$\dot{\boldsymbol{q}}(t)=\dot{s}(t)(\boldsymbol{q}_e-\boldsymbol{q}_s), \tag{2.60}$$

$$\ddot{\boldsymbol{q}}(t)=\ddot{s}(t)(\boldsymbol{q}_e-\boldsymbol{q}_s). \tag{2.61}$$

In order for a path to be executable by the robot manipulator, the desired positions, velocities, and accelerations must be within the acceptable range of the joints actuators capabilities, that is for each joint i, we have [**6**]

$$q_{i,\min}\leq q_i(t)\leq q_{i,\max}, \tag{2.62}$$

$$|\dot{q}_i(t)|\leq\dot{q}_{i,\text{limit}}, \tag{2.63}$$

$$|\ddot{q}_i(t)|\leq\ddot{q}_{i,\text{limit}}. \tag{2.64}$$

2.6.2 Polynomial Time Scaling

A convenient form for the time scaling $s(t)$ is a third-order polynomial time function as [**6**]

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3. \tag{2.65}$$

A point-to-point motion in time T implies the initial constraints $s(0) = \dot{s}(0) = 0$ and the final constraints $s(T) = 1, \dot{s}(T) = 0$. Solving Equation (2.65) for constants a_i with these four constraints, we get

$$a_0 = 0,\ a_1 = 0,\ a_2 = \frac{3}{T^2},\ a_3 = -\frac{2}{T^3}. \tag{2.66}$$

Substituting the solution into Equations (2.60) and (2.61) yields

$$\dot{\boldsymbol{q}}(t) = \left(\frac{6t}{T^2} - \frac{6t^2}{T^3}\right)(\boldsymbol{q}_e - \boldsymbol{q}_s), \tag{2.67}$$

$$\ddot{\boldsymbol{q}}(t) = \left(\frac{6}{T^2} - \frac{12t^2}{T^3}\right)(\boldsymbol{q}_e - \boldsymbol{q}_s), \tag{2.68}$$

The maximum joint velocities are reached at $t = \frac{T}{2}$, which give

$$\dot{\boldsymbol{q}}_{\max} = \frac{3}{2T}(\boldsymbol{q}_e - \boldsymbol{q}_s). \tag{2.69}$$

The maximum joint accelerations and decelerations are reached at $t = 0$ and $t = T$, which gives

$$\ddot{\boldsymbol{q}}_{\max} = -\ddot{\boldsymbol{q}}_{\min} = \frac{6}{T^2}\left|(\boldsymbol{q}_e - \boldsymbol{q}_s)\right|. \tag{2.70}$$

These maximums must fulfill the conditions specified in Equations (2.63) and (2.64). Plots of third-order polynomial time scaling are shown in Figure 2.3.

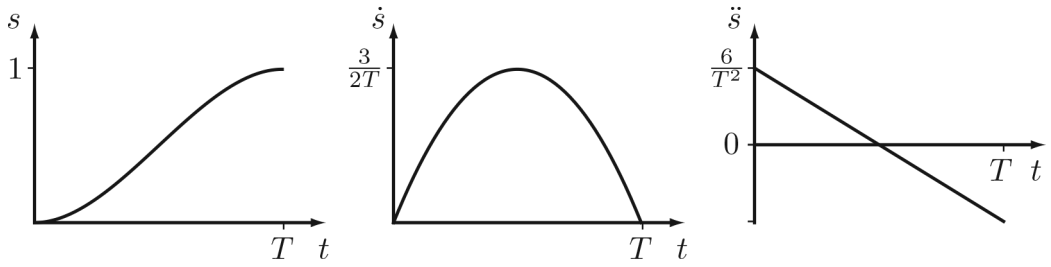

Figure 2.3: Third-order polynomial time scaling.

third-order polynomial time scaling does not constrain the initial and final accelerations to be zero, which means the the robot is required to follow discontinuous jump in acceleration at endpoint of the path, inducing vibration in the robot. Fifth-order polynomial can be used to eliminate this problem, by adding two more design freedoms in the polynomial, which gives a smoother motion with a higher maximum velocity than a third-order polynomial. Plots of fifth-order polynomial time scaling are shown in Figure 2.4.

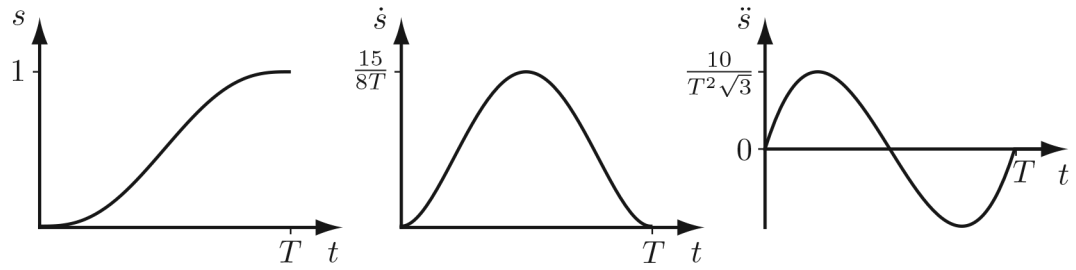

Figure 2.4: Fifth-order polynomial time scaling.

2.6.3 Trapezoidal Motion Profile

Trapezoidal time scaling is common in industrial applications. The trapezoidal motion profile consists of a constant acceleration stage $\ddot{s} = a$ over a time period t_a, followed by a constant velocity stage $\dot{s} = v$ over a time period $T - t_a$, and ending with constant deceleration stage $\ddot{s} = -a$ of time t_a. The trapezoidal motion profile is not as smooth as the polynomial-based ones, but it has the advantage that is for the largest velocity v, and acceleration a, that satisfy [**6**]

$$|(q_e - q_s)\, v| \leq \dot{q}_{\text{limit}}, \tag{2.71}$$

$$|(q_e - q_s)\, a| \leq \ddot{q}_{\text{limit}}, \tag{2.72}$$

result the fastest straight-line motion possible. Plots of trapezoidal motion profile are shown in Figure 2.5.

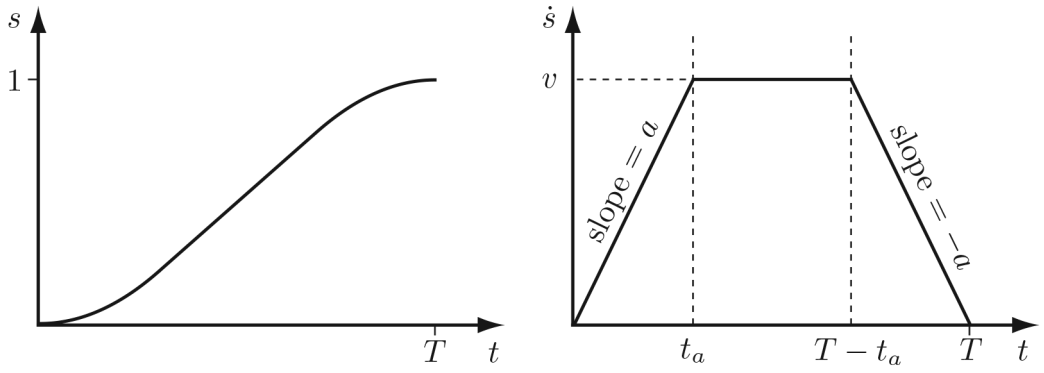

Figure 2.5: $s(t)$ and $\dot{s}(t)$ for a trapezoidal motion profile.

2.6.4 S-Curve Motion Profile

S-curve time scaling is widely used approach for high performance industrial applications, due its feature of avoiding vibrations or oscillations induced by sudden jumps in acceleration. S-curve motion profile consists of seven phases: (1) constant jerk until the desired acceleration is reached; (2) constant acceleration until the required velocity is being approached; (3) constant negative jerk until acceleration equals zero at the time of reaching the desired velocity; (4) coasting at constant velocity; (5) negative constant jerk; (6) constant deceleration; (7) constant positive jerk until velocity and acceleration reach zero exactly at the time when $s = 1$ [**6**].

The acceleration profile $\ddot{s}(t)$ is given by

$$\ddot{s}(t) = \begin{cases} \frac{at}{t_a} & ,0 \leq t < t_a \\ a & ,t_a \leq t < t_b \\ \frac{a(t_a+t_b-t)}{t_a} & ,t_b \leq t < t_a + t_b \\ 0 & ,t_a + t_b \leq t < T - (t_a + t_b) \\ -\frac{a(t_a+t_b-T+t)}{t_a} & ,T - (t_a + t_b) \leq t < T - t_b \\ -a & ,T - t_b \leq t < T - t_a \\ -\frac{a(T-t)}{t_a} & ,T - t_a \leq t \leq T \end{cases} \tag{2.73}$$

Then, $s(t)$ and $\dot{s}(t)$ can be derived from the time integration relationship. The velocity profile $\dot{s}(t)$ for an S-curve is shown in Figure 2.6.

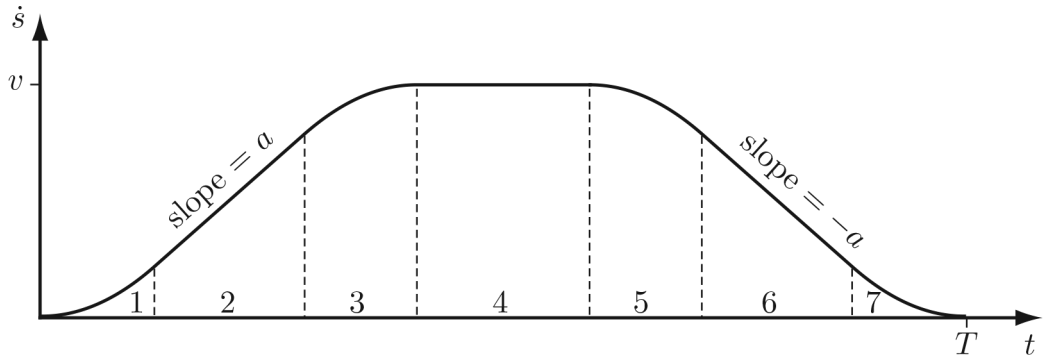

Figure 2.6: $\dot{s}(t)$ for a seven stages S-curve motion profile.

2.7 Chapter Summery

The existence of clearly defined rigid motions and the utilization of Denavit-Hartenberg (DH) parameters play a crucial role in simplifying the derivation of forward and velocity kinematics. By employing these techniques, the process becomes more straightforward and manageable.

To obtain the velocity kinematics and dynamic equations, it is essential to derive the geometric Jacobian. This mathematical tool provides the necessary framework for understanding the relationship between joint velocities and end-effector velocities. By obtaining the geometric Jacobian, one can effectively analyze the system's motion and make informed decisions about control and optimization.

Moreover, having a firm grasp of the forward and velocity kinematics, along with the inertia tensors, enables the derivation of equations of motion using Euler-Lagrange equations. These equations form the foundation for studying the dynamics of mechanical systems. By applying Euler-Lagrange equations to the given system, it becomes possible to describe and analyze the system's behavior, including its response to external forces and torques.

Lastly, trajectory planning techniques, namely polynomial, trapezoidal, and S-Curve motion profiles have been explored, specifically designed for robot manipulators, taking into account the mechanical limitations of the joints' actuators.

Chapter 3

Computed-Torque Control of Industrial Manipulators

3.1 Introduction

The field of robotics has experienced remarkable advancements over the past few decades, revolutionizing industries ranging from manufacturing and healthcare to exploration and entertainment. The ability to control robotic systems with precision and accuracy is of utmost importance to ensure their safe and efficient operation in various applications. In this context, control techniques play a pivotal role in enabling robots to perform complex tasks and interact with their environments effectively.

One prominent control technique that has garnered significant attention in the robotics community is Computed Torque Control (CTC). CTC is a model-based control approach that aims to regulate the robot's joint torques to achieve desired motion and trajectory tracking. It utilizes the dynamics model of the robot to compute the torque inputs necessary to counteract external disturbances and achieve precise control.

In this chapter, the theoretical foundations of full-feedback linearizarion and Computed Torque Control will be investigated to refine and optimize the control strategy, by leveraging the mathematical model of the robot's dynamics developed in Chapter 2. Additionally, the integration of advanced techniques, such as state feedback, PID and optimal control, will be explored to enhance the capabilities of CTC in dealing with uncertainties, nonlinearities, and disturbances.

3.2 Feedback Linearization Control

Feedback linearization is a control technique that allows the control of nonlinear systems as if they were linear. The idea behind feedback linearization is to find a suitable transformation that cancels the nonlinear dynamics of the system, making it behave as a linear system. This transformation is achieved by carefully selectinga feedback control law. The resulting transformed system can then be controlled using standard linear control techniques such as pole placement, optimal control, PID or robust control. By transforming a nonlinear system into a linear one, it becomes easier to analyze its stability, controllability, and observability properties. Moreover, linear control methods are generally well-established, extensively studied, and have a wide range of available design tools.

However, feedback linearization has certain limitations. The technique relies on accurate knowledge of the system's dynamics, including its mathematical model. This information may not always be available or may be subject to uncertainty. Additionally, feedback linearization

cannot overcome fundamental limitations of nonlinear systems, such as unbounded control effort or instability in certain regions of the state space [**22**][**23**]. Feedback linearization control problem can be stated as follows: Given nonlinear a single-input single-output (SISO) system of the form

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\, u, \\ y &= h(\boldsymbol{x}), \end{aligned} \tag{3.1}$$

find a state feedback control

$$u = \alpha(\boldsymbol{x}) + \beta(\boldsymbol{x})v, \tag{3.2}$$

and a change of variables

$$\boldsymbol{z} = \boldsymbol{T}(\boldsymbol{x}), \tag{3.3}$$

that transform the system (3.1) into an equivalent linear system of the form

$$\begin{aligned} \dot{\boldsymbol{z}} &= \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}v, \\ y &= \boldsymbol{C}\boldsymbol{z}, \end{aligned} \tag{3.4}$$

where $\boldsymbol{f} : \boldsymbol{D} \to \mathbb{R}^n$ and $\boldsymbol{g} : \boldsymbol{D} \to \mathbb{R}^n$ are sufficiently differentiable on domain $\boldsymbol{D} \subset \mathbb{R}^n$; $\alpha : \boldsymbol{D} \to \mathbb{R}$; $\beta : \boldsymbol{D} \to \mathbb{R}$ is nonzero for all $\boldsymbol{x} \in \boldsymbol{D}$; $\boldsymbol{T} : \boldsymbol{D} \to \mathbb{R}^n$ is a diffeomorphism such that $\boldsymbol{D}_{\boldsymbol{z}} = \boldsymbol{T}(\boldsymbol{D})$ contains the origin; $\boldsymbol{A} \in \mathbb{R}^{n\times n}$; $\boldsymbol{B} \in \mathbb{R}^n$; $\boldsymbol{C} \in \mathbb{R}^n$; $(\boldsymbol{A}, \boldsymbol{B})$ controllable.

3.2.1 Full-State Linearization

The key idea behind full-state linearization focuses on transforming the entire state dynamics of a nonlinear system into an equivalent linear system. In this context, linearization can be achieved using Lie derivative, which is an essential mathematical tool for analyzing the dynamics of a nonlinear system and guide the transformation process. It measures how the system's state changes along the vector field defined by its dynamics. Considering the system (3.1), the time derivative of the output $\dot{\boldsymbol{y}}$ is defined as [**7**]

$$\dot{y} = \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} \left[\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\, u\right] := L_f h(\boldsymbol{x}) + L_g h(\boldsymbol{x}) u, \tag{3.5}$$

where

$$L_f h(\boldsymbol{x}) = \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}),$$

and

$$L_g h(\boldsymbol{x}) = \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}).$$

are called the Lie derivative of $h(\boldsymbol{x})$ along the fields $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$, respectively, The k^{th} time derivative of the output can be expressed as

$$y^{(k)} = L_f^k h(\boldsymbol{x}) + L_g L_f^{k-1} h(\boldsymbol{x}), \tag{3.6}$$

where

$$\begin{aligned} L_f^0 h(\boldsymbol{x}) &= h(\boldsymbol{x}), \\ L_f^k h(\boldsymbol{x}) &= \frac{\partial \left[L_f^{k-1} h(\boldsymbol{x})\right]}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}), \\ L_g L_f^{k-1} h(\boldsymbol{x}) &= \frac{\partial \left[L_f^{k-1} h(\boldsymbol{x})\right]}{\partial \boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}). \end{aligned} \tag{3.7}$$

If $h(\boldsymbol{x})$ satisfies

$$L_g L_f^{k-1} h(\boldsymbol{x}) = 0, \;\; k = 1, 2, \ldots, \rho - 1; \quad L_g L_f^{\rho-1} h(\boldsymbol{x}) \neq 0, \tag{3.8}$$

then, the derivatives $\dot{y}, \ddot{y}, \ldots, y^{(\rho-1)}$ are independent of input u, and the input only appears in $y^{(\rho)}$ with a nonzero coefficient as

$$y^{(\rho)} = L_f^{\rho} h(\boldsymbol{x}) + L_g L_f^{\rho-1} h(\boldsymbol{x}) u, \tag{3.9}$$

where ρ is the relative degree of the system, and $1 \leq \rho \leq n$. In the case of full-state linearization, the relative degree is equal to the order of the system (i.e. $\rho = n$). By choosing the state feedback control u to be

$$u = \alpha(\boldsymbol{x}) + \beta(\boldsymbol{x}) v, \quad \beta(\boldsymbol{x}) = \left[L_g L_f^{n-1} h(\boldsymbol{x}) \right]^{-1}, \quad \alpha(\boldsymbol{x}) = -\beta(\boldsymbol{x}) L_f^{n} h(\boldsymbol{x}), \tag{3.10}$$

and substituting in Equation (3.9), yields

$$y^{(n)} = v, \tag{3.11}$$

where v is the outer-loop input. We can clearly see that the system is fully linearizable, forming a chain of n integrators. By defining a diffeomorphism transformation as

$$\boldsymbol{z} = \boldsymbol{T}(\boldsymbol{x}) := \begin{bmatrix} y \\ \dot{y} \\ \vdots \\ y^{(n-1)} \end{bmatrix}, \tag{3.12}$$

then, the linearized system dynamics can be rearranged as

$$\begin{aligned} \dot{\boldsymbol{z}} &= \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B} v, \\ y &= \boldsymbol{C}\boldsymbol{z}, \end{aligned} \tag{3.13}$$

where $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ is the controllable canonical form representation of n integrators chain. More specifically, the system is given in the Brunovsky canonical form [**5**] , that is,

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \vdots & & & 0 & 1 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}, \;\; \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \;\; \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{3.14}$$

In a multi-input multi-output (MIMO) nonlinear system, achieving full-state feedback linearization can be done in a manner analogous to the single-input single-output (SISO) system, without any loss of generality.

3.2.2 State Feedback Stabilization

Consider the linearized system (3.14). The poles of a linear system represented in the Brunovsky canonical form are the eigenvalues λ_i, $i = 1, 2, \ldots, n$, of the matrix $\boldsymbol{A}$, which can be obtained by solving the characteristic polynomial of the system, that is,

$$\det\left(\lambda \boldsymbol{I} - \boldsymbol{A}\right) = 0. \tag{3.15}$$

The arrangement of matrix $\boldsymbol{A}$, characterized by a sequence of n integrators, leads to the eigenvalues of the system attaining a value of zero, yielding the following relationship:

$$\lambda_i = 0, \quad i = 1, 2, \ldots, n, \tag{3.16}$$

which makes the system to be unstable. In order to stabilize the dynamics, a full-state feedback controller can be employed. By choosing the outer-loop control law as

$$v = -\boldsymbol{K}\boldsymbol{z}, \tag{3.17}$$

and substituting in Equation (3.13), the closed-loop dynamics can be expressed as

$$\begin{aligned} \dot{\boldsymbol{z}} &= (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})\boldsymbol{z}, \\ y &= \boldsymbol{C}\boldsymbol{z}, \end{aligned} \tag{3.18}$$

where $\boldsymbol{K} = \begin{bmatrix} k_1 & k_2 & \cdots & k_n \end{bmatrix}$, is the state feedback gain matrix. The closed loop state matrix can then be written as

$$\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \vdots & & & 0 & 1 \\ -k_1 & -k_2 & -k_3 & \cdots & -k_n \end{bmatrix}. \tag{3.19}$$

Therefore, the transfer function of the closed-loop system is given by

$$H(s) = \frac{1}{s^n + k_n s^{n-1} + \cdots + k_2 s + k_1}, \tag{3.20}$$

with the characteristic polynomial

$$\Delta_c(s) = s^n + k_n s^{n-1} + \cdots + k_2 s + k_1. \tag{3.21}$$

Routh-Hurwitz Stability

Ensuring the stability of a closed-loop system can be achieved by employing the Routh-Hurwitz stability criterion. The Routh-Hurwitz stability criterion encompasses both a necessary and a sufficient condition for determining the stability of a control system. Failure to satisfy the necessary condition indicates system instability. Conversely, meeting the necessary condition implies the possibility of system stability, but further analysis is required [**24**][**25**]. The sufficient condition assists in ascertaining the stability of the control system.

The necessary condition entails the requirement for the coefficients of the characteristic polynomial to be positive. This condition ensures that all the roots of the characteristic equation possess negative real parts. The sufficient condition involves examining the elements of the first column in the Routh array. For system stability, all elements in the first column should exhibit the same sign, either positive or negative. Routh array for a $n^{th}-$order linear system is shown in Table 3.1.

Table 3.1: Routh array

s^n	$a_0^{(0)}$	$a_2^{(0)}$	$a_4^{(0)}$	$\cdots$
s^{n-1}	$a_1^{(0)}$	$a_3^{(0)}$	$a_5^{(0)}$	$\cdots$
s^{n-2}	$a_1^{(1)}$	$a_3^{(1)}$	$a_5^{(1)}$	$\cdots$
s^{n-3}	$a_2^{(1)}$	$a_4^{(1)}$	$a_6^{(1)}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

where $a_i^{(0)}$ is coefficient of s^i in the characteristic polynomial equation, and $a_j^{(k)}$ is intermediate coefficient given by

$$a_j^{(k)} = a_j^{(k-1)} - \frac{a_{j+1}^{(k-1)} a_{j-2}^{(k-1)}}{a_{j-1}^{(k-1)}}. \tag{3.22}$$

If any coefficient in the first column changes sign, it indicates the presence of poles with positive real parts, indicating system instability.

Pole Placement

Another approach to guarantee the stability of a closed-loop dynamics is by positioning the system's poles in a manner that makes the matrix $\boldsymbol{A} - \boldsymbol{BK}$ Hurwitz, that is all the eigenvalues λ_i possess negative real parts. Ackermann's formula provides a straightforward procedure for computing the state feedback controller gains required to achieve the desired pole locations. Given the closed-loop system described in (3.18), state feedback gain can be computed as

$$\boldsymbol{K} = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \boldsymbol{\mathcal{C}}^{-1} \Delta_d(\boldsymbol{A}), \tag{3.23}$$

where $\boldsymbol{\mathcal{C}}$ is the controllability matrix, given by

$$\boldsymbol{\mathcal{C}} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{AB} & \cdots & \boldsymbol{A}^{n-1}\boldsymbol{B} \end{bmatrix}, \tag{3.24}$$

and $\Delta_d(\boldsymbol{A})$ is the desired characteristic polynomial evaluated at matrix $\boldsymbol{A}$, that is

$$\Delta_d(\boldsymbol{A}) = (\boldsymbol{A} - \lambda_1 \boldsymbol{I})(\boldsymbol{A} - \lambda_2 \boldsymbol{I}) \ldots (\boldsymbol{A} - \lambda_n \boldsymbol{I}). \tag{3.25}$$

In order to compute $\boldsymbol{\mathcal{C}}^{-1}$, the controllability matrix must have a full-rank, which implies that the system has to be controllable in order to employ full-state feedback control.

3.3 Computed-Torque Control

One fundamental challenge in robot control is ensuring that the manipulator accurately tracks a predetermined trajectory. Achieving this objective is crucial as it determines the robot's ability to perform meaningful tasks. Prior to engaging in any productive activity, it is essential to position the robot manipulator precisely at the required locations and at the appropriate time intervals. Thus, The tracking control problem of a robot manipulator can be defined as follows: Given a desired position vector $\boldsymbol{q} \in \mathbb{R}^n$, velocity vector $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, and acceleration vector $\ddot{\boldsymbol{q}} \in \mathbb{R}^n$, compute the necessary joint input $\boldsymbol{u}$ based on the measured states $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, to ensure that the robot's states follow the intended trajectory, while all signals involved in the closed-loop system remain bounded.

Computed torque control, as described by Lewis[**5**], Spong,[**8**] and Slotine [**1**], is a special case of full-feedback liearization control, which combines the benefits of feedback control and model-based control to compensate for uncertainties and disturbances in the system to achieve precise trajectory tracking and robust performance by robot manipulators.

The feedback torque is determined by a feedback control law that adjusts the system's response based on the error between the desired and actual states. The feedback control law aims to minimize this error and improve the tracking performance. It typically employs proportional-integral-derivative (PID) or more advanced control techniques, such as optimal and adaptive controllers, to achieve accurate control of the manipulator.

3.3.1 Linearization Loop

The linearization loop is responsible for transforming the dynamics equations of the robot manipulator into an equivalent linear system considering the manipulator's physical properties, such as mass, inertia, and friction. To generate the Linearization input, we begin with rearranging the manipulator's dynamics equations in the form of nonlinear state space representation. Given the equation of motion of n-link serial manipulator

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \underbrace{\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{f}(\boldsymbol{q})}_{\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}})} + \boldsymbol{\tau}_d = \boldsymbol{\tau}. \tag{3.26}$$

the corresponding state space representation can be written as

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{G}(\boldsymbol{x})\,\boldsymbol{u} + \boldsymbol{w}(t), \tag{3.27}$$

where

$$\boldsymbol{x} := \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}, \quad \boldsymbol{u} := \boldsymbol{\tau}, \quad \boldsymbol{f}(\boldsymbol{x}) := \begin{bmatrix} \dot{\boldsymbol{q}} \\ -\boldsymbol{M}^{-1}(\boldsymbol{q})\,\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}}) \end{bmatrix},$$

$$\boldsymbol{G}(\boldsymbol{x}) := \begin{bmatrix} 0 \\ \boldsymbol{M}^{-1}(\boldsymbol{q}) \end{bmatrix}, \quad \boldsymbol{w}(t) := \begin{bmatrix} 0 \\ -\boldsymbol{M}^{-1}(\boldsymbol{q}) \end{bmatrix} \boldsymbol{\tau_d}(t).$$

The inverse of mass matrix $\boldsymbol{M}(\boldsymbol{q})$ always exists due to the fact that it is symmetric and positive-definite matrix (discussed in Chapter 2). By defining the input as

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{v} + \boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}}), \tag{3.28}$$

and substituting in (3.27), we get

$$\begin{bmatrix} \dot{\boldsymbol{q}} \\ \ddot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} 0 & \boldsymbol{I} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{I} \end{bmatrix} \boldsymbol{v} + \boldsymbol{w}, \tag{3.29}$$

which is a linear system in the Brunovsky canonical form with outer-loop input $\boldsymbol{v}$ as discussed in Section 3.2. Linearization loop scheme is shown in Figure 3.1.

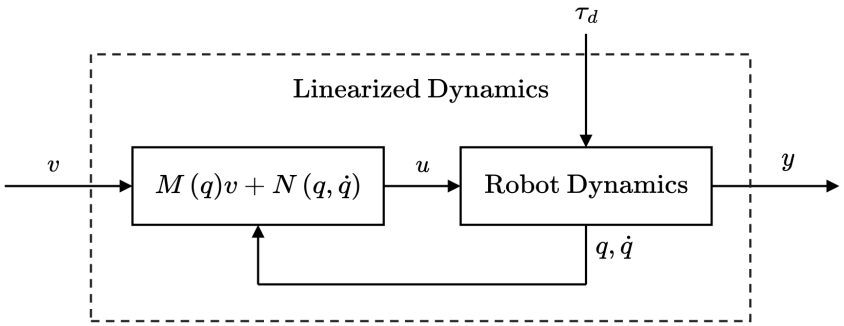

Figure 3.1: Linearization loop

The time-varying disturbance $\boldsymbol{\tau}_d$ can be interpreted as the dynamic impacts arising from both structured and unstructured uncertainties within the system. Structured uncertainties include factors like imprecise inertial link properties, unknown payloads, while unstructured uncertainties arise from unmodeled dynamics such as nonlinear friction, backlash, joint elasticity, and various external disturbances. Hence, the presence of the time-varying disturbance vector $\boldsymbol{w}$ can be attributed to the combined effects of these structured and unstructured uncertainties.

3.3.2 Proportional-Derivative Outer-loop Controller

Consider a desired trajectory $\boldsymbol{q}_d(t)$ that the robot manipulator is required to follow. To ensure trajectory tracking by the joint variable,we define a position tracking error as

$$\boldsymbol{e} = \boldsymbol{q}_d - \boldsymbol{q}. \tag{3.30}$$

To obtain velocity and acceleration tracking errors, we differentiate with respect to time as

$$\begin{aligned} \dot{\boldsymbol{e}} &= \dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}, \\ \ddot{\boldsymbol{e}} &= \ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}. \end{aligned} \tag{3.31}$$

By defining the outer-loop control function as [**5**]

$$\boldsymbol{v} = \ddot{\boldsymbol{q}}_d + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e}, \tag{3.32}$$

and substituting in (3.29), yields closed-loop error dynamics as

$$\begin{bmatrix} \dot{\boldsymbol{e}} \\ \ddot{\boldsymbol{e}} \end{bmatrix} = \begin{bmatrix} 0 & \boldsymbol{I} \\ -\boldsymbol{K}_P & -\boldsymbol{K}_D \end{bmatrix} \begin{bmatrix} \boldsymbol{e} \\ \dot{\boldsymbol{e}} \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{I} \end{bmatrix} \boldsymbol{w}, \tag{3.33}$$

where $\boldsymbol{K}_P$ and $\boldsymbol{K}_P$ are the proportional and derivative gain matrices, respectively. Usually, PD controller gains are arranged as $n \times n$ diagonal matrices, so that

$$\boldsymbol{K}_P = \operatorname{diag}\left\{k_{P,1}, k_{P,1}, \ldots, k_{P,n}\right\}, \quad \boldsymbol{K}_D = \operatorname{diag}\left\{k_{D,1}, k_{D,1}, \ldots, k_{D,n}\right\}. \tag{3.34}$$

The closed-loop error dynamics then are given in Laplace domain as

$$\boldsymbol{e}(s) = \left(s^2\boldsymbol{I} + \boldsymbol{K}_D s + \boldsymbol{K}_P\right)^{-1}\boldsymbol{w}(s) := \boldsymbol{H}(s)\,\boldsymbol{w}(s), \tag{3.35}$$

with characteristic polynomial as

$$\Delta_c(s) = \prod_{i=1}^{n}\left(s^2 + k_{D,i}s + k_{P,i}\right). \tag{3.36}$$

By applying the Routh-Hurwitz stability criterion as disscussed in Section 3.2, we can clearly proof that as long as $k_{P,i}$ and $k_{D,i}$ are all positive, the closed-loop error dynamics are asymptotically stable. Therefore, as long as the time-varying disturbance $\boldsymbol{w}(t)$ is bounded, so is the tracking error $\boldsymbol{e}(t)$, that is

$$\|\mathbf{e}(s)\| \leq \|\boldsymbol{H}(s)\| \cdot \|\boldsymbol{w}(s)\| < \infty. \tag{3.37}$$

The boundedness of $\boldsymbol{w}(t)$ is equivalent to the boundedness of $\boldsymbol{\tau}_d(t)$ and $\boldsymbol{M}(\boldsymbol{q})$ as discussed in Section 2.5. PD computed-torque control scheme is shown in Figure 3.2.

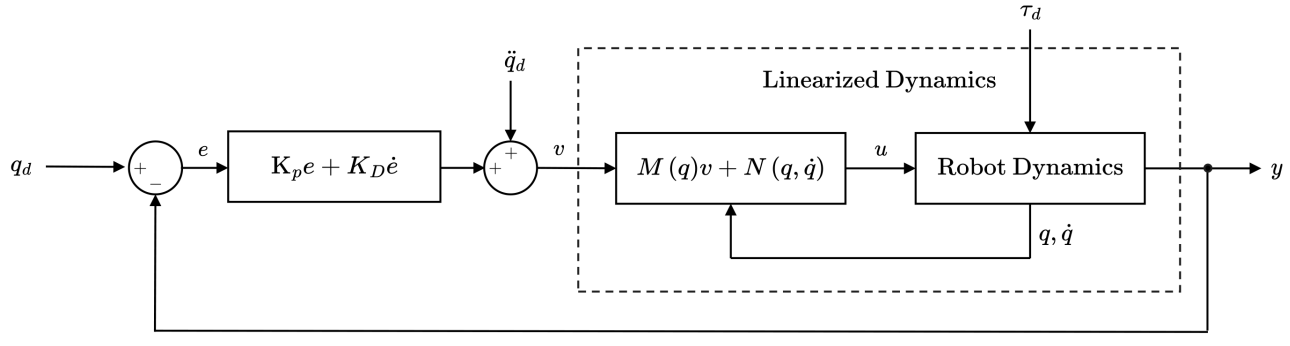

Figure 3.2: PD computed-torque control

3.3.3 Proportional-Integral-Derivative Outer-loop Controller

PD computed-torque control is effective if the disturbance $\boldsymbol{w}(t)$ is negligibly small. However, from control theory viewpoint, it is known that in the presence of significant disturbances, PD controller yields a nonzero steady-state error. PID controller extends the PD controller by adding an integral term, which continuously integrates the error and gradually adjusts the control signal to compensate for any steady-state errors caused by disturbances. Therefore. we are motivated to employ PID computed-torque controller. By introducing an integral tracking error term as

$$\boldsymbol{\varepsilon} = \int_0^t \boldsymbol{e}(t)dt, \tag{3.38}$$

and PID computed-torque control law as [**5**]

$$\boldsymbol{v} = \ddot{\boldsymbol{q}}_d + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} + \boldsymbol{K}_I\boldsymbol{\varepsilon}, \tag{3.39}$$

the closed-loop error dynamics becomes

$$\begin{bmatrix} \boldsymbol{e} \\ \dot{\boldsymbol{e}} \\ \ddot{\boldsymbol{e}} \end{bmatrix} = \begin{bmatrix} 0 & \boldsymbol{I} & 0 \\ 0 & 0 & \boldsymbol{I} \\ -\boldsymbol{K}_I & -\boldsymbol{K}_P & -\boldsymbol{K}_D \end{bmatrix} \begin{bmatrix} \boldsymbol{\varepsilon} \\ \boldsymbol{e} \\ \dot{\boldsymbol{e}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{I} \end{bmatrix} \boldsymbol{w}, \tag{3.40}$$

with characteristic polynomial as

$$\Delta_c(s) = \prod_{i=1}^{n} \left(s^3 + k_{D,i}s^2 + k_{P,i}s + k_{I,i}\right), \tag{3.41}$$

where $\boldsymbol{K}_I$ is defined analogously to (3.34). The steady-state response of the tracking error then can be written as

$$\boldsymbol{e}(s) = (\boldsymbol{K}_{I,i})^{-1}\boldsymbol{w}(s). \tag{3.42}$$

According to the Routh-Hurwitz criterion, the closed-loop error system stability conditions requires that

$$k_{P,i} > 0, \quad k_{D,i} > 0, \quad 0 < k_{I,i} < k_{P,i}k_{D,i}. \tag{3.43}$$

When dealing with unobserved time-varying disturbances, selecting appropriate PID gains becomes challenging. In such cases, adaptive tuning of the gains is necessary to ensure the controller effectively rejects the disturbance and achieves the desired performance. The goal is to make the error (e(t)) asymptotically converge to zero, indicating the system's stability and accurate response. By adaptively adjusting the gains, the controller can dynamically respond to changing disturbance characteristics and maintain optimal performance despite the unpredictable nature of the disturbances. PID computed-torque control is shown in Figure 3.3.

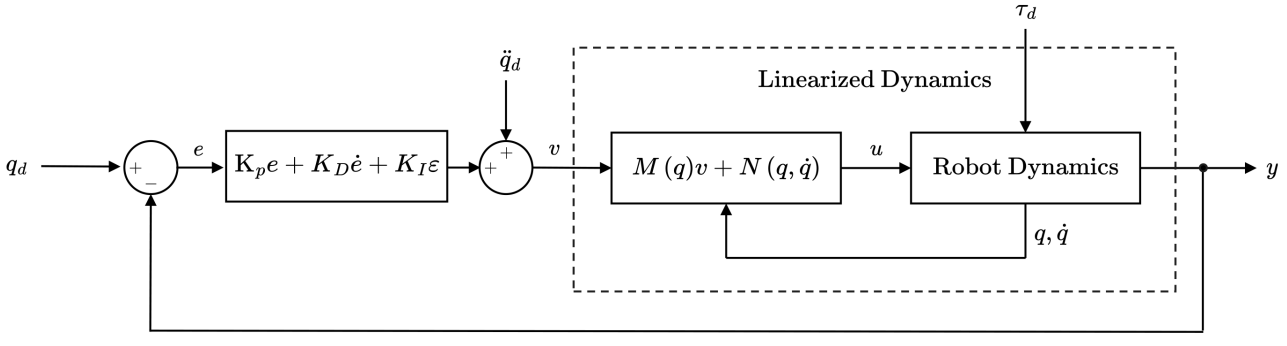

Figure 3.3: PID computed-torque control

3.4 Chapter Summery

In this chapter, the focus was on computed torque control applied to robot manipulators, specifically investigating the effectiveness of PD (Proportional-Derivative) and PID (Proportional-Integral-Derivative) controllers. Computed torque control is a widely used approach in robotics for achieving accurate trajectory tracking and disturbance rejection.

The chapter began by an introduction to the concept of full-feedback linearization control, which serves as the fundamental principle underlying computed-torque control. Subsequently, the computed-torque control scheme was investigated, emphasizing its key features of decoupling the robot dynamics and implementing a feedback control strategy. The primary objective of this approach is to compute the control torque necessary to achieve the desired motion while effectively compensating for disturbances and uncertainties.

Next, the PD controller was presented as a fundamental control technique. Its basic structure, consisting of proportional and derivative terms, allows for responsive and stable control. The benefits and limitations of the PD controller in the context of computed torque control were discussed, along with stability requirements.

Finally, the PID controller, an extension of the PD controller, was introduced. The addition of an integral term enables the controller to handle steady-state errors and improve tracking accuracy. The chapter examined the advantages and challenges associated with PID control, emphasizing stability requirements and the importance of gain tuning to achieve optimal performance.

Chapter 4

Deep Deterministic Policy Gradient

4.1 Introduction

Reinforcement Learning (RL) is a branch of machine learning that focuses on training agents to make sequential decisions in dynamic environments. Unlike supervised learning, where labeled examples guide the learning process, RL agents learn through trial and error, optimizing their behavior based on the feedback received from the environment.

DDPG is a powerful algorithm that combines deep neural networks with the actor-critic framework to handle continuous action spaces efficiently. The actor network learns a policy that directly outputs continuous actions based on the observed state, while the critic network evaluates the quality of the actions taken by the actor. By using deep neural networks, DDPG can handle high-dimensional state and action spaces, allowing agents to learn complex behaviors [**2**][**20**].

In this section, we will delve into the fundamentals of deep feedforward neural networks and reinforcement learning. We will explore the underlying principles of these concepts and thoroughly examine the mechanics and inner workings of the DDPG algorithm.

4.2 Deep Feedforward Neural Networks

Deep feedforward neural networks are a fundamental class of artificial neural networks. which are particularly effective in solving complex problems by learning hierarchical representations of data.

At its core, a deep feedforward neural network consists of multiple layers of interconnected neurons, organized in a sequential manner. The network is called "feedforward" because the information flows in one direction, from the input layer through the hidden layers to the output layer, without any loops or recurrent connections.

Each neuron in a deep feedforward neural network receives inputs from the neurons in the previous layer, performs a computation, and passes the output to the neurons in the next layer. The computations within each neuron involve a weighted sum of inputs, followed by applying an activation function. The activation function introduces non-linearity and allows the network to model complex relationships between inputs and outputs.

In this Section, the main components of deep feedforward neural network will be investigated, including multilayer perceptron, activations, loss functions, backpropagation and numerical optimizers.

4.2.1 Multilayer Perceptron

A multilayer perceptron (MLP) is a powerful and widely used feedforward neural network architecture. It is composed of multiple layers of interconnected neurons, each performing specific computations to process and transform input data. The structure of an MLP allows it to learn complex patterns and make predictions based on the input it receives.

In an MLP, each neuron performs a two-step process. First, it calculates a weighted sum of its inputs, taking into account the importance of each input based on its corresponding weight. This weighted sum represents the combined influence of the inputs on the neuron's activation. Then, an activation function is applied to this sum, introducing non-linearity and determining whether the neuron should be activated or not.

The result of the activation function is then transmitted to the neurons in the subsequent layer, where the process is repeated. This sequential propagation of information through the layers is known as the feed-forward process, as the data flows from the input layer through the hidden layers to the output layer. The output layer provides the final prediction or classification based on the learned patterns and relationships within the data. MLP feed-forward equation can be expressed as [**37**]

$$\boldsymbol{a}^{[l]} = \boldsymbol{f}^{[l]} \left(\underbrace{\boldsymbol{W}^{[l]} \boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}}_{\boldsymbol{z}^{[l]}} \right), \tag{4.1}$$

where $\boldsymbol{a}^{[l]} \in \mathbb{R}^{n^{[l]}}$ is the activation vector of the l^{th}-layer, $\boldsymbol{b}^{[l]} \in \mathbb{R}^{n^{[l]}}$ is the bias vector of the l^{th}-layer, $\boldsymbol{W}^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$ is the weight matrix of the l^{th}-layer, $\boldsymbol{f}^{[l]} \in \mathbb{R}^{n^{[l]}}$ is element-wise activation function of the l^{th}-layer Figure 4.1 illustrates a typical feedforward layer in an MLP.

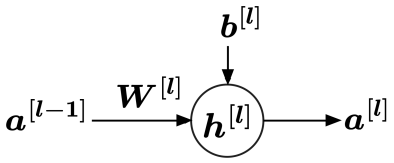

Figure 4.1: MLP feedforward layer

It is important to note that $\boldsymbol{a}^{[0]}$ simply represents the input data vector $\boldsymbol{x}$, while $\boldsymbol{a}^{[L]}$ corresponds to the predicted output vector $\boldsymbol{y}$.

4.2.2 Activation Functions

Activation functions play a pivotal and vital role in determining the activation status of a neuron within a MLP. Their primary purpose is to introduce non-linearity, enabling MLPs to learn and approximate highly nonlinear functions and capture intricate patterns and features in the data [**38**].

By utilizing elementary mathematical operations, activation functions effectively determine the significance of the neuron's input within the network's prediction process. Without non-linear activation functions, MLPs would be limited to representing only linear transformations of the input data.

Furthermore, activation functions also facilitate the flow of gradients during the training process. During backpropagation, the gradients are computed and propagated through the network to update the model's weights. Activation functions that are differentiable ensure the availability of derivatives, allowing for efficient calculation and backpropagation of gradients. This crucial functionality enables the network to optimize its parameters effectively and learn from prediction errors. Some Commonly used activation functions in MLPs are:

Linear Activation

The linear activation function, also known as the identity function, is a simple activation function that preserves the weighted sum of inputs without introducing any non-linearity. Its equation is defined as

$$f(x) = x. \tag{4.2}$$

Sigmoid Activation

The sigmoid activation function maps the weighted sum of inputs to a value between 0 and 1, providing a smooth and bounded non-linearity. It is commonly used in binary or logistic classification problems. The equation for the sigmoid function is expressed as

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{4.3}$$

Hyperbolic Tangent (tanh) Activation

The hyperbolic tangent (tanh) activation function is similar to the sigmoid function but maps the weighted sum of inputs to a value between -1 and 1. It provides a symmetric non-linearity around the origin. The equation for the tanh function is given by

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{4.4}$$

Rectified Linear Unit (ReLU) Activation

The rectified linear unit (ReLU) activation function is a popular choice due to its simplicity and ability to handle sparse activation. It returns the input if it is positive, and zero otherwise. The equation for the ReLU function is

$$f(x) = \max(0, x). \tag{4.5}$$

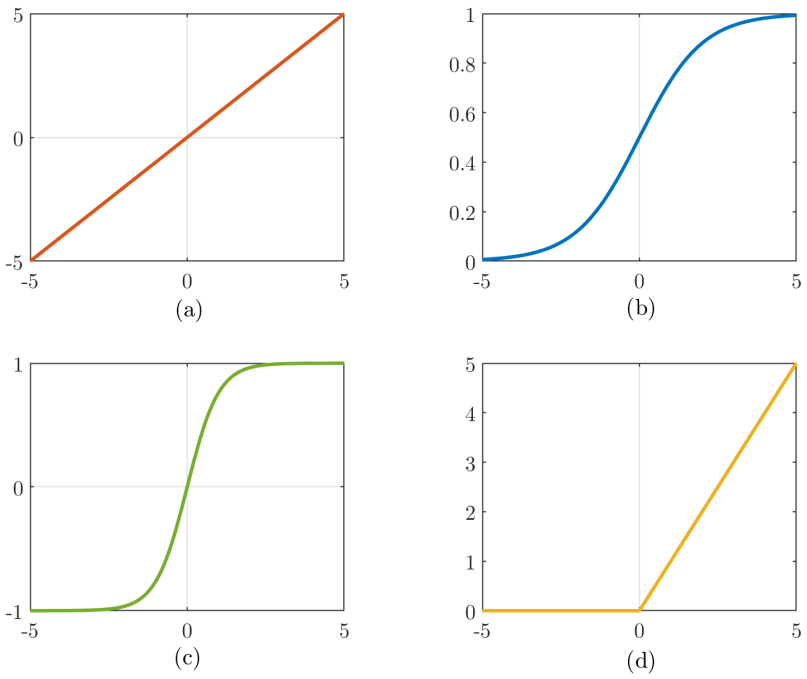

Figure 4.2: Activation functions: (a) Identity. (b) Sigmoid. (c) tanh. (d) ReLU.

4.2.3 Loss Functions

Loss functions play a crucial role in neural networks as they serve as a measure of the discrepancy between the predicted output and the actual target values. These functions quantify the error or loss incurred by the network during training, providing a means to guide the learning process and optimize the model's parameters [**39**].

In a neural network, the ultimate goal is to minimize the loss function, which drives the network to make accurate predictions. Different types of loss functions are used depending on the task at hand, such as regression or classification. The choice of the loss function depends on the specific problem and the desired behavior of the network.

For regression problems, common loss functions include mean squared error (MSE), Root Mean Squared Error (RMSE), and mean absolute error (MAE).

Mean Squared Error

Mean squared error is a popular loss function in neural networks, particularly for regression tasks. It computes the average squared difference between predicted and true values. MSE is continuous and differentiable, making it suitable for gradient-based optimization algorithms like backpropagation. It emphasizes larger errors due to its squared nature, but this can make it sensitive to outliers. It is also sensitive to the scale of the target variable. MSE is given by

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{4.6}$$

Root Mean Squared Error

Root mean squared error is another commonly used evaluation metric for regression tasks. It measures the average magnitude of prediction errors by taking the square root of the average of the squared differences between predicted and true values. RMSE provides interpretability, sensitivity to large errors, and efficient optimization. However, it is sensitive to outliers and the scale of the target variable, which should be considered when using RMSE in deep learning. RMSE can be computed as

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}. \tag{4.7}$$

Mean Absolute Error

Mean absolute error is also common evaluation metric and loss function. It measures the average absolute difference between predicted and true values. MAE is robust to outliers, insensitive to the scale of the target variable, and provides a straightforward interpretation. However, it treats all errors equally and is not differentiable at zero, which can affect its sensitivity to small errors and optimization using gradient-based algorithms. MAE equation is given by

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|. \tag{4.8}$$

The selection of an appropriate loss function is crucial for effective training and optimizing the network's performance. It guides the learning process by providing feedback signals to adjust the network's weights and biases. By minimizing the loss function through techniques like gradient descent and backpropagation, the network learns to make better predictions and improve its overall performance.

4.2.4 Backpropagation

In a feedforward neural network, information flows from the input layer to the output layer, a process known as forward propagation. The input data is passed through the hidden units at each layer until it produces the prediction output. During training, forward propagation continues until it generates a scalar loss or cost function, $E(\boldsymbol{\theta})$ which represents the discrepancy between the predicted and actual output. On the other hand, backward propagation or backpropagation for short, enables the neural network to update its parameters by propagating the error backwards from the output layer to the input layer.

The main objective is to minimize the loss function by finding the optimal set of parameters. This involves calculating the gradient of the loss function to determine optimal network parameters.

However, finding the global minimum of the loss function can be challenging in complex neural networks. Iterative optimization algorithms are employed to update the parameters and converge to a local minimum, but guaranteeing the discovery of the global minimum is typically not feasible.

In order to compute the gradient, we start by defining a parameters vector as

$$\boldsymbol{\theta}^{[l]} = \begin{bmatrix} \boldsymbol{w}^{[l]} \\ \boldsymbol{b}^{[l]} \end{bmatrix}, \tag{4.9}$$

where

$$\boldsymbol{w}^{[l]} = \operatorname{vec}\left(\boldsymbol{W}^{[l]}\right). \tag{4.10}$$

Then, gradient of the loss function $E(\boldsymbol{\theta})$ with respect to l^{th} layer learnable parameters, can be obtained according to

$$\nabla_{\boldsymbol{\theta}^{[l]}} E = \left(\frac{\partial E}{\partial \boldsymbol{\theta}^{[l]}}\right)^T. \tag{4.11}$$

Using the chain rule, we get

$$\frac{\partial E}{\partial \boldsymbol{\theta}^{[l]}} = \frac{\partial E}{\partial \boldsymbol{a}^{[l]}} \frac{\partial \boldsymbol{a}^{[l]}}{\partial \boldsymbol{z}^{[l]}} \frac{\partial \boldsymbol{z}^{[l]}}{\partial \boldsymbol{\theta}^{[l]}}. \tag{4.12}$$

Knowing that

$$\boldsymbol{W}^{[l]} \boldsymbol{a}^{[l-1]} = \left[\left(\boldsymbol{a}^{[l-1]}\right)^T \otimes \boldsymbol{I}_{n^{[l]}}\right] \boldsymbol{w}^{[l]}, \tag{4.13}$$

equation (4.12) can be simplified to [**40**]

$$\frac{\partial E}{\partial \boldsymbol{\theta}^{[l]}} = \frac{\partial E}{\partial \boldsymbol{a}^{[l]}} \operatorname{diag}\left(\mathrm{f}'^{[\mathrm{l}]}\left(\boldsymbol{z}^{[\mathrm{l}]}\right)\right) \begin{bmatrix} \left(\boldsymbol{a}^{[l-1]}\right)^T \otimes \boldsymbol{I}_{n^{[l]}} & \boldsymbol{I}_{n^{[l]}} \end{bmatrix}. \tag{4.14}$$

Once the gradients are computed, optimization algorithms like gradient descent can be used to update the parameters of the network. The gradients provide information on how the loss function changes as the parameters are adjusted, allowing the network to move towards the direction of lower loss.

Its important to note that real-world implementations of backpropagation require appropriate hardware and software setups to handle the complexity and intensity of computation as the the number of learnable parameters in the network increases.

4.2.5 Optimization Algorithms

Optimization algorithms play a vital role in improving the performance of deep learning models. These algorithms adjust the weights and minimize the loss function during the model training process. An optimizer is a function or algorithm that modifies attributes of a neural network, including weights, biases and learning rates. The primary objective is to minimize the overall loss and enhance accuracy.

Deep learning models are characterized by non-linearities and a large number of parameters, making it challenging to determine the optimal parameter values that maximize prediction accuracy. In this context, optimization algorithms are indispensable in all aspects of deep learning applications. They provide the means to efficiently explore the parameter space, search for the best configurations, and achieve optimal model performance. Optimization algorithms empower deep learning practitioners to fine-tune their models and extract the best possible insights from their data [**41**]. In this work, we focus on the gradient descent algorithm and several of its variants for our purposes.

Gradient Descent

Gradient descent (GD) is an iterative optimization algorithm utilized to find a local minimum of a differentiable function. The main concept involves taking consecutive steps in the direction opposite to the gradient of the function at the current point, as it represents the path of steepest descent. On the contrary, moving in the direction of the gradient would result in reaching a local maximum of the function, which is referred to as gradient ascent. In the context of deep learning, GD is highly valuable for minimizing the cost or loss function. This iterative process can be expressed as [**41**]

$$\theta_{k+1} = \theta_k - \gamma \nabla_\theta E\left(\theta_k\right), \tag{4.15}$$

where $\gamma > 0$ is the learning rate, which acts as a scaling factor for the gradient, determining the magnitude of each step and effectively controlling the size of parameters updates.

To converge towards the local minimum effectively, it is crucial to select an appropriate learning rate. It should neither be excessively low nor excessively high. This is important because if the steps taken are too large, the algorithm may not successfully reach the local minimum, instead bouncing back and forth within the convex function of gradient descent. On the other hand, setting the learning rate to an extremely small value will eventually guide the gradient descent towards the local minimum, but it may require a considerable amount of time to converge.

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is an iterative technique used to optimize an objective function with desirable smoothness properties, such as differentiability or subdifferentiability. It can be seen as a stochastic approximation of gradient descent optimization, as it substitutes the true gradient (computed from the complete dataset) with an estimation (computed from a randomly chosen subset of the data). This approach is particularly valuable in high-dimensional optimization problems, as it alleviates the considerable computational burden associated with processing the entire dataset. While it leads to faster iterations, it generally comes at the expense of a slower convergence rate.

Gradient estimation equation has the form of a sum as

$$\nabla_\theta E(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_i(\theta), \tag{4.16}$$

where each summand function $\nabla_\theta E_i(\theta)$ is typically associated with the i^{th} observation in the training dataset. Using this estimation, stochastic gradient descent parameter update can be computed as [**41**]

$$\theta_{k+1} = \theta_k - \frac{\gamma}{N}\sum_{i=1}^{N} \nabla_\theta E_i(\theta_k). \tag{4.17}$$

During the execution of the algorithm, it iterates through the training set and applies the aforementioned update for each training sample. Multiple passes can be made over the training set until the algorithm converges. To prevent cycles and introduce further randomness, the data can be shuffled before each pass.

SGD With Momentum

Although stochastic gradient descent is widely used as an optimization strategy, it can sometimes result in slow learning. To address this issue, the method of momentum is employed to accelerate the learning process, particularly in scenarios involving high curvature, small yet consistent gradients, or noisy gradients. The momentum algorithm tackles this by accumulating an exponentially decaying moving average of past gradients and utilizes this information to continue moving in the direction of the accumulated gradients.

The momentum algorithm incorporates a variable v, which represents the velocity of the parameters as they traverse the parameter space. This velocity determines both the direction and speed of the parameter updates. Specifically, the velocity is established as an exponentially decaying average of the negative gradient. The term "momentum" is derived from a physical analogy, wherein the negative gradient corresponds to a force propelling a particle through the parameter space in accordance with Newton's laws of motion. The update rule is expressed as [**41**]

$$\theta_{k+1} = \theta_k + v_{k+1}, \tag{4.18}$$

where

$$v_{k+1} = \alpha v_k - \frac{\gamma}{N}\sum_{i=1}^{N} \nabla_\theta E_i(\theta_k). \tag{4.19}$$

In this context, $\alpha \in [0, 1)$ is a hyperparameter that governs the rate at which the influence of previous gradients diminishes exponentially. The impact of previous gradients on the current direction is more pronounced when α is larger relative to γ.

Adaptive Gradient

Adaptive gradient (Adagrad) is an optimization algorithm that adjusts the learning rate of each parameter based on their historical gradients. Its primary purpose is to handle sparse and noisy data by individually adapting the learning rate for each parameter. This adaptation considers the frequency and intensity of past parameter updates. Adagrad decreases the learning rate for frequently updated parameters and increases it for parameters with infrequent updates,

enabling effective handling of such data scenarios. Updating rule for the network's parameters is given by [**41**]

$$\theta_{k+1} = \theta_k - \gamma_{k+1} \nabla_\theta E\left(\theta_k\right), \tag{4.20}$$

where

$$\gamma_k = \frac{\gamma}{\sqrt{\alpha_k + \varepsilon}}, \tag{4.21}$$

$$\alpha_k = \sum_{i=1}^{k} \left(\nabla_\theta E\left(\theta_i\right)\right)^2. \tag{4.22}$$

To prevent division-by-zero, the small positive scalar ε is incorporated. In this context, the learning rate is then adapted to decrease automatically as the sum of squared gradients continues to increase after each time step.

Adaptive Momentum

The Adam optimizer, short for Adaptive Moment Estimation, is an advanced variant of stochastic gradient descent (SGD) that merges the principles of adaptive learning rates and momentum to optimize models effectively and efficiently. In Adam, the learning rate is adaptively adjusted for each parameter by considering the magnitude of their gradients. By scaling the learning rate with the estimated standard deviation of the gradients, Adam can effectively handle parameters with different sensitivities. This adaptive learning rate approach enables Adam to converge rapidly and achieve efficient optimization of the model.

To employ Adam algorithm, we start with defining an update rule for aggregates of gradients according to [**41**]

$$m_{k+1} = \beta_1 m_k + \frac{(1-\beta_1)}{N} \sum_{i=1}^{N} \nabla_\theta E_i\left(\theta_k\right), \tag{4.23}$$

$$v_{k+1} = \beta_2 v_k + \frac{(1-\beta_2)}{N^2} \left(\sum_{i=1}^{N} \nabla_\theta E_i\left(\theta_k\right) \right)^2, \tag{4.24}$$

where $\beta_i \in [0, 1)$ is constant. Initially, both m_k and v_k are set to 0. As β_i approaches 1, there is a tendency for m_k and v_k to become more biased towards 0. To address this issue, bias correction can be applied as

$$\hat{m}_k = \frac{m_k}{(1-\beta_1)}, \tag{4.25}$$

$$\hat{v}_k = \frac{v_k}{(1-\beta_2)}. \tag{4.26}$$

Ultimately, the network parameters are updated according to the following equation

$$\theta_{k+1} = \theta_k - \frac{\alpha \hat{m}_k}{\sqrt{\hat{v}_k + \varepsilon}}, \tag{4.27}$$

where the small positive scalar ϵ is utilized to prevent division-by-zero scenarios.

To select the optimal optimizer for deep learning training, several factors should be considered. These factors include the size and complexity of the dataset, the architecture of the network, convergence speed, sensitivity to hyperparameters, incorporation of regularization techniques, prior knowledge, and empirical results. It is essential to experiment with different optimizers and hyperparameter settings to identify the most suitable one for a specific task.

4.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a powerful reinforcement learning (RL) algorithm that combines deep neural networks with the actor-critic framework to solve continuous control problems. It is a variation of the popular Deep Q-Network (DQN) algorithm, specifically designed to handle continuous action spaces.

In many real-world applications, such as robotics, finance, and autonomous systems, the action space is continuous, meaning that actions are represented by real-valued vectors rather than discrete choices. Traditional RL algorithms struggle to handle such continuous action spaces due to the need for discretization or the computational challenges they pose. DDPG addresses this limitation by employing deep neural networks and leveraging the actor-critic architecture [**32**].

DDPG has demonstrated remarkable performance in a variety of complex control tasks, including robotic manipulation, robotic locomotion, and continuous control in simulated environments. Its ability to handle continuous action spaces and leverage the power of deep neural networks makes it a popular choice for tackling real-world problems[**33**].

4.3.1 Agent-Environment Interaction

In the realm of RL, an agent interacts with an environment by taking actions based on the current system state and observing the subsequent responses. This iterative process enables the agent to gradually refine its behavior and optimize its performance over time.

The environment encapsulates all elements outside the agent's direct control, constituting factors that cannot be modified arbitrarily. The agent's actions encompass a diverse range of choices and strategies, representing the decisions it aims to learn and improve upon.

To enhance its effectiveness, the agent embarks on a trial-and-error exploration, experimenting with different actions and analyzing the outcomes to identify the most favorable choices. Through continuous adaptation and refinement, the agent gradually acquires the ability to make increasingly optimal decisions, ultimately attaining higher rewards within the environment.

The agent-environment interaction model, depicting this dynamic process, is illustrated in Figure 4.3.

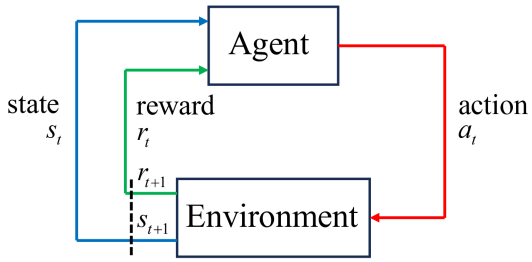

Figure 4.3: Agent-environment interaction model

In this model, the learning process occurs in several steps. At each time step t, the agent observes a state s_t and selects an action a_t to perform. Upon taking the action, the environment E and the agent transition to a new state s_{t+1} based on the current state and the chosen action. Additionally, as the environment moves to the new state, it provides the agent with a scalar reward r_{t+1} as feedback, indicating the quality of the agent's action in relation to the environment. This iterative process continues until the agent gradually approaches an optimal behavior through learning and adaptation.

4.3.2 Markov Decision Process

RL environments can be formally described as Markov decision processes (MDPs). The key principle of a Markov property in RL states that the future outcomes are independent of the past, given the current state. This property implies that once the current state is known, the past history becomes irrelevant and can be discarded. Consequently, by knowing only the initial state, it is possible to predict all future states and rewards by iteratively applying the model. An MDP can be represented by a tuple consisting of five elements (S, A, P, R, γ), where S represents a finite set of states that can be reached by the environment; A denotes the finite set of possible agent's actions; P refers to the state transition probability function, given by [**42**]

$$P\left(s', a, s\right) = \mathbb{P}\left[s_{t+1} = s' | s_t = s, a_t = a\right]; \tag{4.28}$$

R is the reward function, given by

$$R\left(s', a, s\right) = \mathbb{E}\left[r_{t+1} | s_{t+1} = s', s_t = s, a_t = a\right]; \tag{4.29}$$

$\gamma \in [0, 1]$ is the discount factor, which determines the relative importance placed on immediate rewards compared to future rewards. When γ is set to a lower value, more weight is given to immediate rewards. On the other hand, when γ is close to 1, greater significance is attributed to future rewards in the decision-making process.

The goal in RL is to maximize the cumulative reward obtained from the environment, often referred to as the return function G_t. The return function represents the total sum of rewards accumulated by the agent over a certain time horizon or episode. The objective is to find a policy or strategy that leads to the highest possible value. The expected return function is formulated as [**2**]

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{4.30}$$

To maximize the expected return, the agent aims to learn a policy π that prescribes the optimal actions to maximize the expected return in every state. In general, the policy π defines a probability distribution over actions for each state, indicating the likelihood of taking different actions given a particular state. The policy π can be expressed as [**2**]

$$\pi\left(s|a\right) = \mathbb{P}\left[s_t = s | a_t = a\right]. \tag{4.31}$$

The optimal policy π^* represents the set of actions to be taken in each state, leading to the highest cumulative reward across all states. By discovering and following the optimal policy, the RL agent aims to maximize its overall performance and achieve the most favorable outcomes in the given environment. The optimal policy can be obtained as [**2**]

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[G_t | \pi\right]. \tag{4.32}$$

4.3.3 Value Functions

The value of a state or state-action pair in reinforcement learning reflects the expected return when starting from that state or state-action pair and subsequently following a specific policy indefinitely. A value function serves as an estimate of the desirability of being in a particular state. Similarly, it measures the quality of performing a specific action within that state. The notion of goodness is quantified in terms of an optimality criterion, typically defined as the expected return. In this study, our focus is on the expected infinite-horizon discounted return, allowing us to omit the time as an explicit parameter. This simplification eliminates the need for additional notation when dealing with terminal states. There are two primary categories of value functions, namely:

State-Value Function

Considering a specific state s, and a policy π, the expected return according to policy π can be computed as [**43**]

$$V^{\pi}(s) = \mathbb{E}\left[G_t | s_t = s\right]. \tag{4.33}$$

Action-Value Function

Considering a specific state s, arbitrary action a, and a policy π, the expected return according to policy π can be computed as [**43**]

$$Q^{\pi}(s,a) = \mathbb{E}\left[G_t | s_t = s, a_t = a\right], Q : S \times A \to \mathbb{R}. \tag{4.34}$$

4.3.4 Actor-Critic Networks

In practical scenarios, it is rare for an agent to learn an optimal policy. Especially in the continuous action domain, generating optimal policies can be computationally expensive. To mitigate this computational cost, we inevitably resort to approximations. The Q-learning algorithm is a popular method for estimating the optimal action-value function, Q^*, in reinforcement learning. Q-functions, also known as critic functions, play a central role in this algorithm.

The goal is to learn the optimal action-value function, Q^*, which represents the expected cumulative reward for taking a particular action in a given state and following the optimal policy thereafter. The Q-function takes both the current state and action as inputs and outputs the expected cumulative reward. By estimating the optimal policy, an agent can make informed decisions by selecting actions that maximize the expected cumulative reward. The optimal Q-function can be computed as [**44**]

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{4.35}$$

Using the Bellman equation [], the optimal Q-function can be recursively updated [**45**]

$$Q^*(s_{t+1}, a_{t+1}) = Q^{\pi}(s_t, a_t) + \alpha\left[R_t + \gamma Q^*(s_t, a_t) - Q^{\pi}(s_t, a_t)\right], \tag{4.36}$$

where α is the learning rate. Consequently, the parameters of the Q- function, namely θ^Q, are updated by minimizing a loss function of the form [**45**]

$$L(\theta^Q) = \frac{1}{M}\sum_{i=1}^{M}\left(Q(s_t, a_t|\theta^Q) - y_i\right)^2 \tag{4.37}$$

where M is the mini-batch size of random samples, and

$$y_i = r(s_t, a_t) + \gamma Q(s_{i+1}, \mu(s_{i+1}|\theta^Q)), \tag{4.38}$$

where μ is the actor deterministic policy according to the Q-value function, that is

$$a_t = \mu(s_t|\theta^Q). \tag{4.39}$$

The actor is updated using policy gradient for maximizing the expected discounted reward with respect to the actor [**2**] parameters θ^{μ} as

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{M}\sum_{i=1}^{M} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_i}. \tag{4.40}$$

Finally, the parameters of the actor and critic target networks are updated using a smooth update technique [**2**][**45**], employing a factor $\rho \ll 1$ as

$$\begin{aligned} \theta^{\mu'} &= \rho\theta_{\mu} + (1-\rho)\theta^{\mu'}, \\ \theta^{Q'} &= \rho\theta_{Q} + (1-\rho)\theta^{Q'}. \end{aligned} \tag{4.41}$$

Figure 4.4 depicts the illustration of the Actor-Critic scheme.

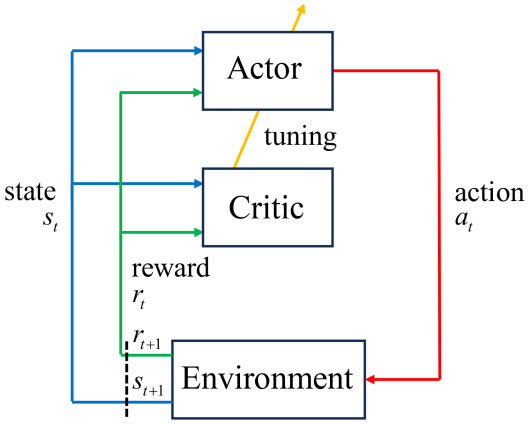

Figure 4.4: Actor-Critic scheme: The actor serves as the policy, transforming the input state into an output action, while the critic serves as the value function. Both networks can be updated by incorporating the input from the critic. It is worth noting that the actor relies on the critic throughout the learning process.

4.3.5 Replay Buffers

In the DDPG algorithm, the replay buffer plays a crucial role in training the agent by storing and providing a pool of past experiences. These experiences typically consist of tuples containing the current state, action taken, reward received, next state, and whether the episode terminated.

During training, the agent samples mini-batches of experiences from the replay buffer to learn from a diverse set of transitions. This process helps mitigate the issues of correlated and non-stationary data that can arise when using consecutive experiences during online training.

By randomly sampling from the replay buffer, the agent can break any temporal correlations in the data and learn from a broader range of experiences. This enables more efficient and stable learning as the agent is not solely reliant on the most recent experiences.

The replay buffer also allows for experience replay, which is crucial for achieving off-policy learning in DDPG. Off-policy learning means the agent can learn from experiences generated by a different policy than the one currently being trained. This feature enables the agent to learn from older and more diverse experiences, leading to better sample efficiency and generalization [**46**].

4.3.6 Exploration

In RL with discrete action spaces, exploration is typically achieved by randomly selecting actions, often following an epsilon-greedy strategy. However, in the case of continuous action spaces, exploration is accomplished by adding noise directly to the action itself. Ornstein-Uhlenbeck process [**26**][**27**], is a commonly utilized in RL to introduce noise to the action output. Following the addition of noise, the action is then clipped within the appropriate range

to ensure it remains valid and feasible. An exploration policy μ' can be defined as

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + N_t, \tag{4.42}$$

where N is noise produced by the Ornstein-Uhlenbeck process, which can be discretized and approximated via Euler-Maryuama discretization as

$$N_{t+1} = N_t + \beta_1(\beta_2 - N_t)\Delta t + \beta_3 \Delta W_t, N_0 = n_0, \tag{4.43}$$

where $\beta_1 > 0$, β_2, and $\beta_3 > 0$ are process parameters, and W_t represent the Wiener process.

4.4 Chapter Summery

In this chapter, we delve into two important topics in the field of artificial neural networks: deep feedforward neural networks and Deep Deterministic Policy Gradient (DDPG) algorithm.

First, we explore deep feedforward neural networks, which are also known as feedforward neural networks or multilayer perceptrons (MLPs). We start by discussing the basics of neural networks, including neurons, activation and loss functions. Then, we delve into the concept of deep neural networks, which are neural networks with multiple hidden layers. We explain the motivation behind using deep networks and highlight their ability to learn complex patterns and representations.

Next, we cover the training process for deep feedforward neural networks. We discuss the backpropagation algorithm, which is the key method for optimizing the network's weights. We explain how backpropagation computes the gradients and updates the weights using gradient descent. Additionally, we touch upon optimization techniques, such as gradient descent, scholastic gradient decent, momentum, adaptation, and finally, Adam algorithm, which makes the learning process more efficient in terms of preference and convergence speed.

Moving on, we introduce the concept of reinforcement learning and dive into the DDPG algorithm. We discuss the fundamental components of DDPG, including markov decision process, value functions, actor and critic networks, and experience replay. We explain how DDPG combines the advantages of deep neural networks with the policy gradient algorithm to solve continuous control problems. We also highlight the importance of exploration and exploitation in reinforcement learning and how DDPG addresses this through the use of noise and exploration policies.

Overall, this chapter provides a comprehensive overview of deep feedforward neural networks and the DDPG algorithm. It covers the theoretical foundations, training processes, practical considerations, and real-world applications of these powerful techniques in the field of artificial intelligence and reinforcement learning.

Chapter 5

DDPG-Based Adaptive PID-CTC of UR5e Robot

This chapter delves into the development of a simulation model for the UR5e robot, involving a comprehensive discussion on the kinematic model and dynamic parameters. Utilizing the most reliable data sourced from the manufacturer and relevant literature [**12**][**13**], the model is built to accurately represent the behavior of the UR5e robot.

The focus then shifts to the implementation of the proposed control strategy. This strategy entails training a DDPG agent, which is tasked with adaptively updating the gains of a PID-Computed torque controller designed specifically for the UR5e robot model. The ultimate goal is to achieve high-performance trajectory tracking, thereby ensuring precise and stable motion execution.

Simulation results are presented and analyzed to evaluate the effectiveness of the proposed method. The outcomes demonstrate the remarkable capability of the DDPG agent in adapting to reject unobservable time-varying disturbances. Through its learning process, the DDPG agent identifies the optimal policy that yields gains ensuring closed-loop stability and superior trajectory tracking performance with minimal errors.

The findings underscore the potential of the proposed control approach in handling uncertainties and disturbances encountered in the robot's operational environment. By dynamically updating the PID gains based on real-time feedback, the DDPG agent showcases its adaptability and robustness, empowering the robot to maintain precise and stable motion even in challenging situations.

5.1 UR5e Robot Manipulator

The UR5e robot is a collaborative robotic arm developed by Universal Robots. It features a compact and lightweight design, making it easy to install and move around different work environments. It consists of six joints that provide it with a high degree of flexibility and precision in its movements. With it's collaborative Operation, he UR5e is designed to work collaboratively alongside humans, ensuring safe interaction without the need for safety barriers or cages. It employs advanced safety features like power and force limiting to protect human operators in case of accidental collisions [**29**][**30**].

The robot can be programmed using Universal Robots' intuitive programming interface called Polyscope. Polyscope allows users to program the robot through a touch screen interface with icons and visual cues, making it accessible to both beginners and experienced operators. It supports various programming options, including graphical flowchart-based programming and script programming.

Due to it's versatile applications, the UR5e robot can be equipped with a wide range of end

effectors and grippers to suit different application needs. Universal Robots offers a variety of devices and sensors, that can be easily integrated with the robot arm, enabling it to perform tasks such as picking, placing, and manipulating objects.

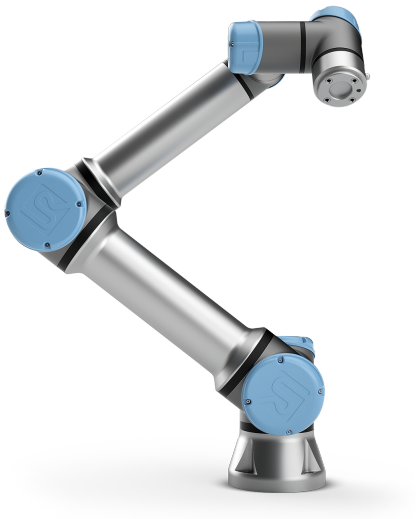

Figure 5.1: UR5e collaborative robot

The UR5e is commonly used in applications such as pick and place operations, machine tending, assembly, packaging, testing, and quality inspection. It is suitable for scenarios where human-robot interaction is required.

5.1.1 Forward Kinematics

Figure 5.1 illustrates the structure of UR5e manipulator, which comprises seven links and six revolute joints. Each revolute joint provides one degree of freedom (DOF), resulting in a total of six DOF for the UR5e. To derive the forward kinematics, the initial step involves determining the Denavit-Hartenberg (DH) parameters.

DH Parameters

To obtain the DH parameters for the UR5 manipulator, the DH convention, as described in Section 2.3.2, is applied. The process begins by creating a diagram of the manipulator, illustrating its links and joints (Figure 5.2). The dimensions of the links are provided by the manufacturer.

Based on the diagram and following the DH convention, coordinate frames o_i, x_i, y_i, z_i are assigned for each link, where $i = 0, 1, \ldots, 6$. The assigned coordinate frames are depicted in Figure 5.2. Notably, the coordinate frame o_2, x_2, y_2, z_2 is not located on the second link. This placement minimizes the number of non-zero DH parameters and ensures the subsequent transformation matrices are concise.

Due to the revolute nature of UR5e robot joints, joint position variables are defined according to (2.21) as

$$q_i = \theta_i, \tag{5.1}$$

where in this case $\sigma_i = 1$, as discussed in Section 2.3.1. The DH parameters are visualized in Figure 5.2 and summarized in Table 5.1.

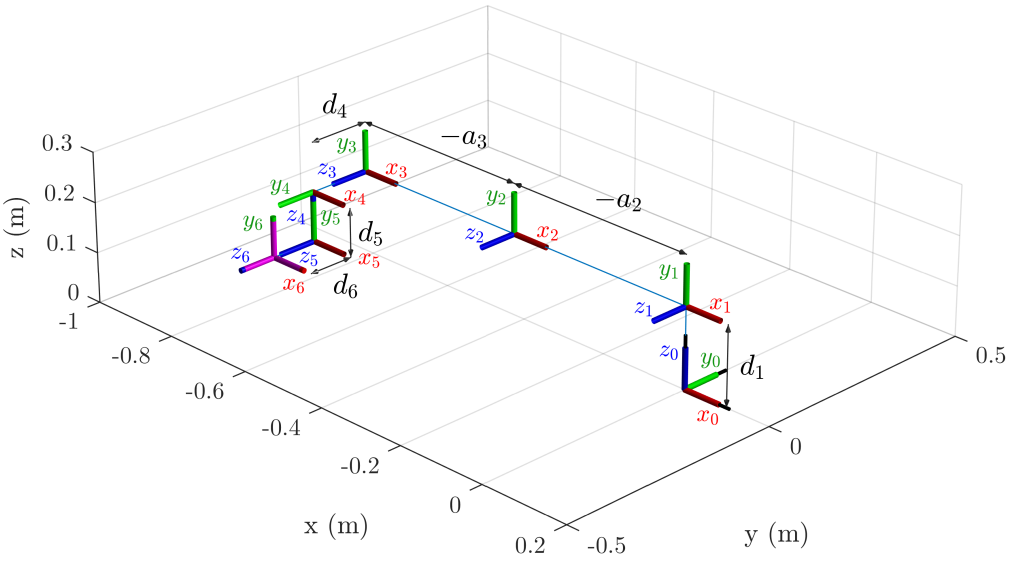

Figure 5.2: DH coordinates assignment for UR5e robot

Table 5.1: DH parameters of UR5e robot

i	$a_i(m)$	$\alpha_i(rad)$	$d_i(m)$	$\theta_i(rad)$
0	0	0	–	–
1	0	$\frac{\pi}{2}$	0.1625	θ_1
2	-0.425	0	0	θ_2
3	-0.3922	0	0	θ_3
4	0	$\frac{\pi}{2}$	0.1333	θ_4
5	0	$-\frac{\pi}{2}$	0.0997	θ_5
6	0	0	0.0996	θ_6

Transformation Matrices

The transformation matrices ${}^{i-1}\boldsymbol{T}_i\left(\theta_i\right)$ can be obtained by substituting the parameters from Table 5.1 into Equation (2.26) as

$$ {}^{0}T_1 = \begin{bmatrix} \cos\left(\theta_1\right) & 0 & \sin\left(\theta_1\right) & 0 \\ \sin\left(\theta_1\right) & 0 & -\cos\left(\theta_1\right) & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.2} $$

$$ {}^{1}T_2 = \begin{bmatrix} \cos\left(\theta_2\right) & -\sin\left(\theta_2\right) & 0 & a_2\cos\left(\theta_2\right) \\ \sin\left(\theta_2\right) & \cos\left(\theta_2\right) & 0 & a_2\sin\left(\theta_2\right) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.3} $$

$$ {}^{2}T_3 = \begin{bmatrix} \cos\left(\theta_3\right) & -\sin\left(\theta_3\right) & 0 & a_3\cos\left(\theta_3\right) \\ \sin\left(\theta_3\right) & \cos\left(\theta_3\right) & 0 & a_3\sin\left(\theta_3\right) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.4} $$

$$
{}^{3}T_{4} = \begin{bmatrix} \cos(\theta_4) & 0 & \sin(\theta_4) & 0 \\ \sin(\theta_4) & 0 & -\cos(\theta_4) & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.5}
$$

$$
{}^{4}T_{5} = \begin{bmatrix} \cos(\theta_5) & 0 & -\sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & \cos(\theta_5) & 0 \\ 0 & -1 & 0 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.6}
$$

$$
{}^{5}T_{6} = \begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & 0 \\ \sin(\theta_6) & \cos(\theta_6) & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.7}
$$

The forward kinematics of the UR5e manipulator are represented by the transformation matrix ${}^{0}\boldsymbol{T}_{6}(\boldsymbol{\theta})$, which defines the position and orientation of the end effector with respect to the base frame o_0, x_0, y_0, z_0 and the joint positions vector $\boldsymbol{\theta_i}$. ${}^{0}\boldsymbol{T}_{6}$ is derived by substituting Equations (5.2)-(5.7) into Equation (2.23), yielding

$$
{}^{0}T_{6} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.8}
$$

with its components defined as

$$
\begin{aligned}
p_x &= d_6(s_1c_5 - c_1c_{234}c_5) + d_4s_1 + a_2c_1c_2 + d_5\mathrm{s}_{234}\mathrm{c}_1 \\
&\quad + a_3c_1c_2c_3 - a_3c_1s_2s_3, \\
p_y &= a_2s_1c_2 - d_4c_1 - d_6(c_1c_5 + c_{234}s_1s_5) + d_5s_1s_{234} \\
&\quad + a_3s_1c_2c_3 - a_3s_1s_2s_3, \\
p_z &= d_1 + d_5(s_{23}s_4 - c_{23}c_4) + a_3s_{23} + a_2s_2 \\
&\quad - d_6s_5(c_{23}s_4 + s_{23}c_4), \\
r_{11} &= c_6(s_1s_5 + c_1c_{234}c_5) - c_1s_{234}s_6, \\
r_{12} &= -s_6(s_1s_5 + c_1c_{234}c_5) - c_1s_{234}c_6, \\
r_{13} &= s_1c_5 - c_1c_{234}s_5 \\
r_{21} &= -c_6(c_1s_5 - s_1c_{234}c_5) - s_1s_{234}s_6, \\
r_{22} &= s_6(c_1s_5 - s_1c_{234}c_5) - s_1s_{234}c_6, \\
r_{23} &= -c_1c_5 - s_1c_{234}s_5, \\
r_{31} &= c_{234}s_6 + s_{234}c_5c_6, \\
r_{32} &= c_{234}c_6 - s_{234}c_5s_6, \\
r_{33} &= -s_{234}s_5,
\end{aligned} \tag{5.9}
$$

where s_{234} represents $\sin(\theta_2 + \theta_3 + \theta_4)$ and c_{234} for the $\cos(.)$ of the same.

It is important to highlight that the nonlinear behavior of the robot dynamics is a result of the inherent nonlinearities present in the equations of forward kinematics.

Euler Angles Representation

The orientation of the end-effector frame o_6, x_6, y_6, z_6 relative to the base frame o_i, x_i, y_i, z_i can be described using three angles φ, ϑ, ψ, known as Euler angles, which can be obtained form the rotational part of the transformation matrix ${}^0\boldsymbol{T}_6$ as

$$\begin{aligned} \vartheta &= \operatorname{atan2}\left(r_{33}, \sqrt{1-r_{33}^2}\right), \\ \varphi &= \operatorname{atan2}\left(r_{13}, r_{23}\right), \\ \psi &= \operatorname{atan2}\left(-r_{31}, r_{32}\right), \end{aligned} \tag{5.10}$$

for $r_{13} \neq 0$ and $r_{23} \neq 0$, where atan2(.) refers to the two-argument arctangent function.

In case of $r_{13} = r_{23} = 0$, the sum $\varphi + \psi$ can be obtained if $r_{13} = 1$ as

$$\varphi + \psi = \operatorname{atan2}\left(r_{11}, -r_{12}\right). \tag{5.11}$$

Similarly, the difference $\varphi - \psi$ can be obtained for $r_{13} = -1$ as

$$\varphi - \psi = \operatorname{atan2}\left(-r_{11}, -r_{12}\right). \tag{5.12}$$

In this scenario, where only the sum or difference can be determined, there exist an infinite number of solutions. To resolve this, a convention can be adopted where φ is chosen to be 0 in order to find a solution.

5.1.2 Velocity Kinematics

Geometric Jacobian

The angular component of the geometric Jacobian matrix that is attached to the i^{th} link frame and relative to the base frame $\boldsymbol{J}_{{}^0\boldsymbol{\omega}_i}$, can be determined by considering the axes ranging from ${}^0\boldsymbol{z}_0$ to ${}^0\boldsymbol{z}_5$. These axes are obtained by applying Equation (2.33), yielding

$$\begin{aligned} {}^0\boldsymbol{z}_0 &= \boldsymbol{k}, \\ {}^0\boldsymbol{z}_1 &= {}^0\boldsymbol{R}_1\boldsymbol{k}, \\ {}^0\boldsymbol{z}_2 &= {}^0\boldsymbol{R}_1\,{}^1\boldsymbol{R}_2\boldsymbol{k}, \\ {}^0\boldsymbol{z}_3 &= {}^0\boldsymbol{R}_2\,{}^2\boldsymbol{R}_3\boldsymbol{k}, \\ {}^0\boldsymbol{z}_4 &= {}^0\boldsymbol{R}_3\,{}^3\boldsymbol{R}_4\boldsymbol{k}, \\ {}^0\boldsymbol{z}_5 &= {}^0\boldsymbol{R}_4\,{}^4\boldsymbol{R}_5\boldsymbol{k}, \end{aligned} \tag{5.13}$$

where $\boldsymbol{k} = [0, 0, 1]^T$, and ${}^{i-1}\boldsymbol{R}_i$ is the rotational part of transformation matrix ${}^{i-1}\boldsymbol{T}_i$ specified in Equations (5.2)-(5.7). The linear part of the Jacobian matrix that is attached to the i^{th} link frame and relative to the base frame, namely $\boldsymbol{J}_{{}^0\boldsymbol{o}_i}$, can be obtained as described in Equation (2.36), where ${}^{i-1}\boldsymbol{o}_i$ is the linear or displacement part of transformation matrix ${}^{i-1}\boldsymbol{T}_i$.

5.1.3 Dynamics

Centers of Mass

The manufacturer provides the positions of the mass center points for each link, denoted as ${}^i\boldsymbol{c}_i$, which are expressed in a frame attached to each respective link. The positions of these mass center points are illustrated in Figure 5.3 and provided in Table 5.2 [**14**], along with the corresponding mass of each link.. To compute the dynamics equations as described in Section 2.5, it is necessary to convert the vector representing the center of mass coordinates into the inertial frame. This transformation can be accomplished using Equation (2.38).

After transforming the center of mass coordinates into the inertial frame, the linear component of the geometric Jacobian matrix for a specific center of mass can be determined by applying Equation (2.39).

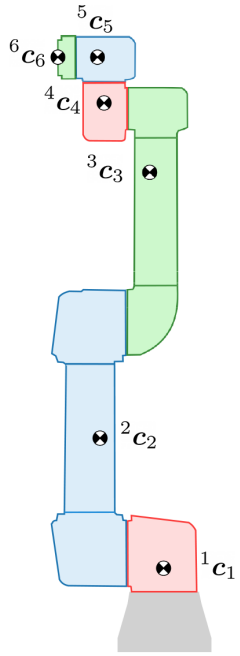

Figure 5.3: UR5e mass center points

Table 5.2: Link dynamic parameters of UR5e

i	$m_i(kg)$	$[c_{x,i}, c_{y,i}, c_{z,i}](m)$
1	3.761	[0, -0.02561, 0.00193]
2	8.058	[0.2125, 0, 0.11336]
3	2.846	[0.15, 0, 0.0265]
4	1.37	[0, -0.0018, 0.01634]
5	1.3	[0, 0.0018, 0.01634]
6	0.365	[0, 0, -0.001159]

The coordinates provided by the manufacturer for the center of mass attached to each link assume the absence of any external load. These coordinates represent the equilibrium position of the center of mass under the inherent weight and distribution within the link itself. However, it is essential to consider that the center of mass may shift when external loads are applied to the system. When external loads are present, such as forces or moments acting on the links, the position of the center of mass may deviate from the initially provided coordinates. The magnitude and direction of the external loads can influence the displacement of the center of mass, leading to a new equilibrium position.

Inertia Tensors

In order to compute the dynamics equations, particularly the kinetic energy of the UR5e robot, it is necessary to have knowledge of the inertia tensors in the body-attached frame for each respective link. However, the manufacturer does not provide this specific data.

There are two approaches to calculate the inertia tensors. One approach involves approximating the link structure using elementary volumes, as demonstrated in [1]. Another approach

is to compute the inertia tensors using CAD software, such as Solidworks, which utilizes finite element methods.

In this work, the chosen method for obtaining the inertia tensors is through finite element analysis. This method provides more accurate data compared to geometric approximation, ensuring the reliability of the calculated inertia tensors. The inertia tensors of UR5e robot about the attached link's center of mass are provided based on finite element analysis in SI units ($kg.m^2$), and given by

$$\begin{aligned}
\boldsymbol{I}_1 &= \begin{bmatrix} 92 & 0 & 0 \\ 0 & 81 & -2 \\ 0 & -2 & 78 \end{bmatrix} \times 10^{-4}, \boldsymbol{I}_2 = \begin{bmatrix} 172 & 0 & 0 \\ 0 & 3273 & 0 \\ 0 & 0 & 3246 \end{bmatrix} \times 10^{-4} \\
\boldsymbol{I}_3 &= \begin{bmatrix} 43 & 0 & 56 \\ 0 & 856 & 0 \\ 56 & 0 & 848 \end{bmatrix} \times 10^{-4}, \boldsymbol{I}_4 = \begin{bmatrix} 23 & 0 & 0 \\ 0 & 18 & 1 \\ 0 & 1 & 14 \end{bmatrix} \times 10^{-4} \\
\boldsymbol{I}_5 &= \begin{bmatrix} 17 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 15 \end{bmatrix} \times 10^{-4}, \boldsymbol{I}_6 = \begin{bmatrix} 2035 & 0 & 0 \\ 0 & 2074 & 3 \\ 0 & 3 & 2532 \end{bmatrix} \times 10^{-7}.
\end{aligned} \tag{5.14}$$

Having this, the inertia tensors relative to the base frame can be computed according to Equation (2.42).

Joint Parameters

Considerations for joint dynamics parameters and mechanical constraints play a vital role in the design of robust controllers. These parameters are particularly important in achieving optimal trajectory tracking performance. For example, during trajectory planning, it is crucial to take into account the maximum positions, velocities, and accelerations that the joint actuators can handle. By considering these limits, it is possible to avoid overloading the robot, which could potentially cause damage.

Additionally, friction is an undesirable effect in robot joints that can impact performance. To achieve optimal trajectory tracking performance, it is necessary to minimize or eliminate the effects of friction as much as possible. This ensures smoother and more accurate movement of the UR5e manipulator.

Universal Robots provides users with various parameters for the UR5e manipulator, which comprises individual joints equipped with brushless AC servo motors and harmonic drive reducers. Table 5.3 offers valuable information about these joints, including details such as maximum and minimum joint velocities and accelerations [**12**]. The table also provides friction parameters and maximum torque ratings that are essential for understanding the behavior and optimizing the performance of the UR5e robot's joints.

Table 5.3: Joint parameters of UR5e

i	$f_{v,i}(\frac{N.m.s}{rad})$	$f_{c,i}(N.m)$	$\dot{\theta}_{i,max}(\frac{rad}{s})$	$\ddot{\theta}_{i,max}(\frac{rad}{s^2})$	$\tau_{i,max}(N.m)$
1	0.4	0.11	±3.2	±15	150
2	0.4	0.11	±3.2	±15	150
3	0.4	0.11	±3.2	±15	150
4	0.3	0.13	±3.2	±25	28
5	0.3	0.13	±3.2	±25	28
6	0.3	0.13	±3.2	±25	28

5.2 Trajectory Tracking Controller

5.2.1 DDPG Agent

In the given context, we consider a DDPG agent that interacts with an environment, denoted as E, in discrete time steps t. The agent receives a set of states, represented as $s_t \in \boldsymbol{S}$, from the environment. It is assumed that the environment is fully observed, meaning the agent has complete information about the current state.

The agent then provides a set of actions, denoted as $a_t \in \boldsymbol{A}$, and receives an immediate reward r_t from a reward function $r(s_t, a_t)$. The agent's actions are determined by a deterministic policy, which is defined in Equation (4.39).

In this work, the goal of the DDPG agent is to learn an optimal policy through trial and error in each learning episode, aiming to maximize the discounted future reward R_t. To begin, we define the state and action sets as

$$\boldsymbol{S} = \left[\boldsymbol{e}^T, \dot{\boldsymbol{e}}^T\right]^T, \ \boldsymbol{A} = \left[k_{P_1}, k_{D_1}, k_{I_1}, \ldots, k_{P_n}, k_{D_n}, k_{I_n}\right]. \tag{5.15}$$

Here, $S \in \mathbb{R}^{2n}$ represents the joint position and velocity errors described in Section 3.3.2, and $A \in \mathbb{R}^{3n}$ represents the optimal PID controller gains discuused in Section 3.3.3. The aim is to design the controller such that the trajectory tracking error $e(t)$ converges to zero in the presence of unobserved time-varying bounded disturbance $w(t)$ while maintaining stable closed-loop dynamics.

Reward Function

In order to enhance the efficiency of the controller, a reward function is devised to minimize the joint position and velocity errors. The proposed reward function [**34**], is defined as

$$r = -\frac{\Delta t}{2nT} \sum_{i=1}^{n} \left(|e_i| + |\dot{e}_i|\right) \leq 0. \tag{5.16}$$

In this reward function, Δt represents the time step, n denotes the number of joints, and T signifies the total episode simulation time. The reward is calculated by taking the negative average of the absolute values of the joint position and velocity errors across all joints, and then scaling it by the time step and the number of joints. This reward function choice is a modified version of the MAE loss function discussed in Section 4.2.3. The MAE loss function has been demonstrated to be robust to outliers, insensitive to the scale of the target variable, and offers a clear and direct interpretation.

By using the MAE-based reward function, the controller aims to minimize the absolute differences between the desired and actual joint positions and velocities. This choice of reward function facilitates effective learning by providing meaningful feedback to the agent regarding the performance of the controller. It helps the agent understand and adapt its actions to minimize the errors and improve trajectory tracking.

The best possible result is achieved when the trajectory tracking error $e(t)$ and its derivative $\dot{e}(t)$ both approach zero, leading to a reward of zero. Therefore, the goal in this case is to minimize the reward, indicating successful convergence of the controller.

Selecting an optimal reward function does not have precise rules. However, several factors are considered in constructing the reward, such as the speed of convergence, learning performance, and the nature of the state-action domain (continuous or discrete). These factors influence the design of the reward function to effectively guide the learning process of the controller.

Actor-Critic Networks

Figure 5.4 illustrates the proposed actor and critic networks. In this configuration, the actor network takes the observed states from the environment as input. These states correspond to the position and velocity trajectory tracking errors of the UR5e robot. The actor network generates an output that represents the optimal PID gains. These gains aim to minimize the tracking error while accounting for both external and internal time-varying disturbances.

On the other hand, the critic network receives both the states and the resulting actions as input. It calculates the critic or Q-value, which provides feedback to fine-tune the actor network and optimize its performance. By incorporating the critic's evaluation, the actor network can learn and improve its decision-making process, ultimately enhancing the overall controller performance.

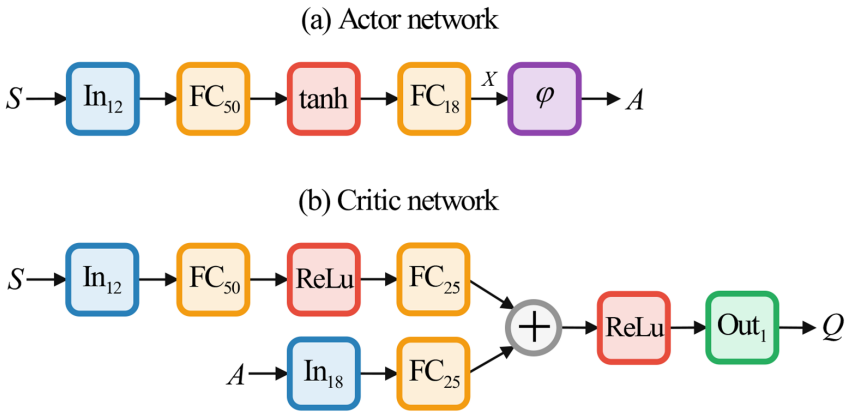

Figure 5.4: Actor-Critic networks

In order to satisfy the stability conditions of the closed-loop error system, the output function in the actor network is specifically designed. As discussed in Section 3.3.3, the stability conditions for the PID-CTC controller require that

$$k_{P,i} > 0, \quad k_{D,i} > 0, \quad 0 < k_{I,i} < k_{P,i}k_{D,i}.$$

To enforce these conditions, a restriction function $\boldsymbol{\varphi}(\boldsymbol{X}) = \boldsymbol{A}$ is proposed, where the vector $\boldsymbol{X} = [x_1, x_2, \ldots, x_{3n}]$ represents output of the second-to-last layer. and ε_1 and ε_2 are positive constants with $\varepsilon_1 > \varepsilon_2$. The restriction function is defined as [**34**]

$$\begin{cases} k_{P_i} = & \max(\varepsilon_1, x_{1+3(i-1)}), \\ k_{D_i} = & \max(\varepsilon_1, x_{2+3(i-1)}), \\ k_{I_i} = & \min(\max(\varepsilon_1, x_{3+3(i-1)}), k_{P_i}k_{D_i} - \varepsilon_2^2). \end{cases} \tag{5.17}$$

In this formulation, each PID gain $k_{P,i}$, $k_{D,i}$, and $k_{I,i}$ is determined based on the corresponding elements of the vector $\boldsymbol{X}$, ensuring that they satisfy the stability conditions. The values are constrained by a maximum value of ε_1 for $k_{P,i}$ and $k_{D,i}$, and a minimum of $\max(\varepsilon_1, x_{3+3(i-1)})$ and a maximum of $k_{P_i}k_{D_i} - \varepsilon_2^2$ for $k_{I,i}$.

5.2.2 Adaptive PID-CTC Controller

DDPG-based adaptive PID-CTC control scheme of UR5e robot is illustrated in Figure 5.5, which consists of two main sections.

The first section is the linearization loop, where the nonlinear dynamics of the UR5e robot are linearized using computed-torque control. This process utilizes the system's states, including joint positions and velocities, to generate a linearization input that cancels out the

nonlinearities. The linearization loop aims to achieve a linear model of the system for better control design and stability analysis.

The second section is the outer-loop control, responsible for trajectory tracking. It incorporates a PID controller with gains that are continuously tuned by the DDPG agent. The DDPG agent optimizes the PID gains to minimize the tracking error, considering the presence of bounded and unobservable time-varying disturbances.

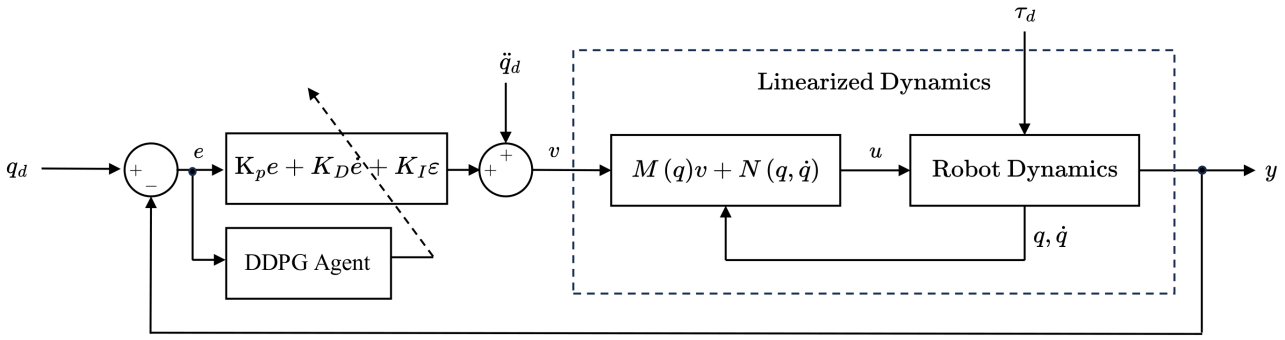

Figure 5.5: DDPG-based adaptive PID-CTC controller

5.3 Implementation

5.3.1 DDPG Agent Training

Training a DDPG agent involves an iterative process that aims to optimize the agent's policy to maximize its expected cumulative reward. The training process steps are organized as follow:

Algorithm 1: DDPG algorithm

```
Randomly initialize parameters θ^μ, θ^Q;
Initialize target network Q′, μ′;
Initialize replay buffer B;
for episode = 1, E do
    Initialize noise process N (4.43);
    Receive initial observed state s_t;
    for t = 1, T do
        Select and execute action a_t (4.39);
        Observe reward r_t (5.16) and state s_{t+1};
        Store transition (s_t, a_t, r_t, s_{t+1}) in B;
        Sample a random M transitions from B;
        Update the actor policy (4.40);
        Update the target networks (4.41);
    end
end
```

With the necessary data and tools to establish a simulation environment, it becomes possible to train the agent using the parameters specified in Table 5.4. The training process produces the outcomes depicted in Figure 5.6.

Table 5.4: DDPG agent training parameters

Parameter	Description	Value
α	Learning rate	10^{-4}
γ	Discounting factor	1
ρ	Smoothing factor	10^{-3}
B	Replay buffer size	10^6
E	Maximum number of episodes	5000
M	Mini-batch size	64
Δt	Sampling time	$0.1(s)$
T	Episode time	$10(s)$

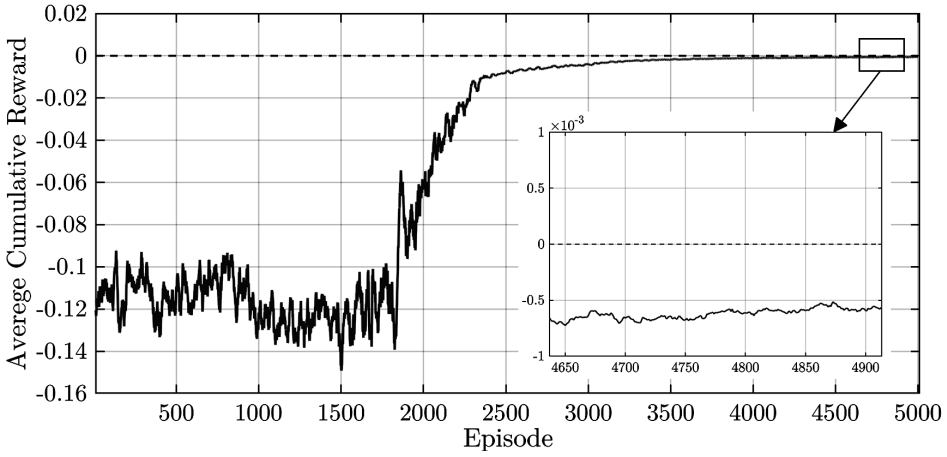

Figure 5.6: Average cumulative reward per episode

From the observations in Figure 5.6, it can be seen that after approximately 5000 episodes of training, the average cumulative reward converges to around -5×10^{-4}, indicating that both the position tracking error $e(t)$ and velocity tracking error $\dot{e}(t)$ are also converging towards zero, as mentioned earlier. This convergence implies that the DDPG agent has successfully computed PID gains that minimize the trajectory tracking errors in the presence of bounded time-varying disturbances.

Furthermore, this achievement in minimizing the tracking errors is coupled with the assurance of system stability. The stability of the system is vital for safe and accurate control. The DDPG agent's ability to generate PID gains that ensure stability underlines its effectiveness in handling the dynamic behavior of the system.

5.3.2 Simulations and Results

The effectiveness of the trained agent was assessed by selecting a desired joint configuration vector as $\theta_e = \left[\pi, -\frac{2\pi}{3}, \frac{\pi}{2}, -\frac{\pi}{3}, \frac{\pi}{4}, -\frac{\pi}{6}\right]$. The initial positions were set to $\theta_s = 0$. By applying the S-curve trajectory planning equations from Section 2.6, the desired trajectory elements θ_d, $\dot{\theta}_d$, and $\ddot{\theta}_d$ were derived.

To evaluate the agent's performance, it was tested in a simulation environment using the specified trajectory. The simulation aimed to mimic real environmental conditions, so disturbances were introduced. These disturbances were modeled as a combination of continuous and discontinuous time-varying functions with randomly generated parameters. The results of this evaluation in SI units are depicted in Figures 5.7 to 5.17 .

Joint-Space Trajectory Tracking

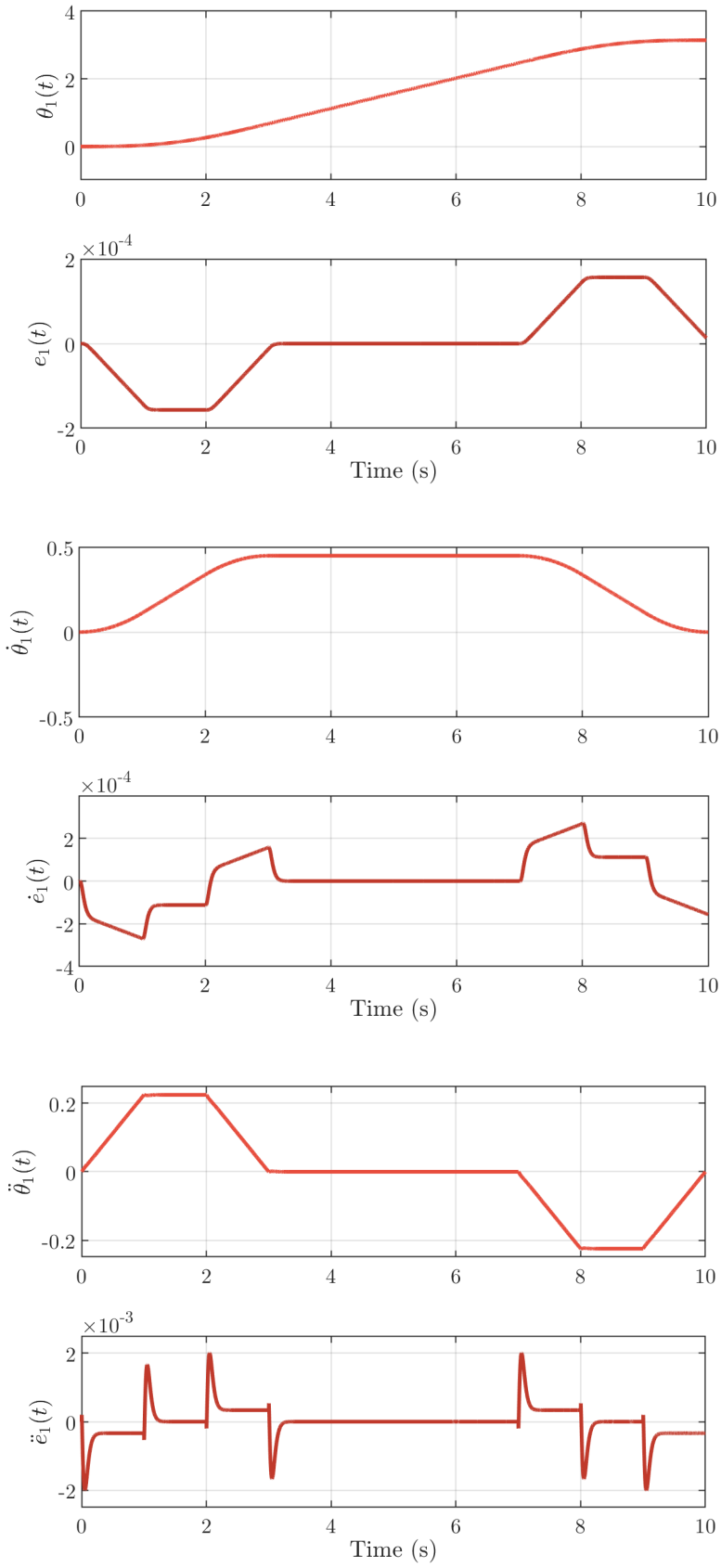

Figure 5.7: Joint-space trajectory tracking: Joint 1

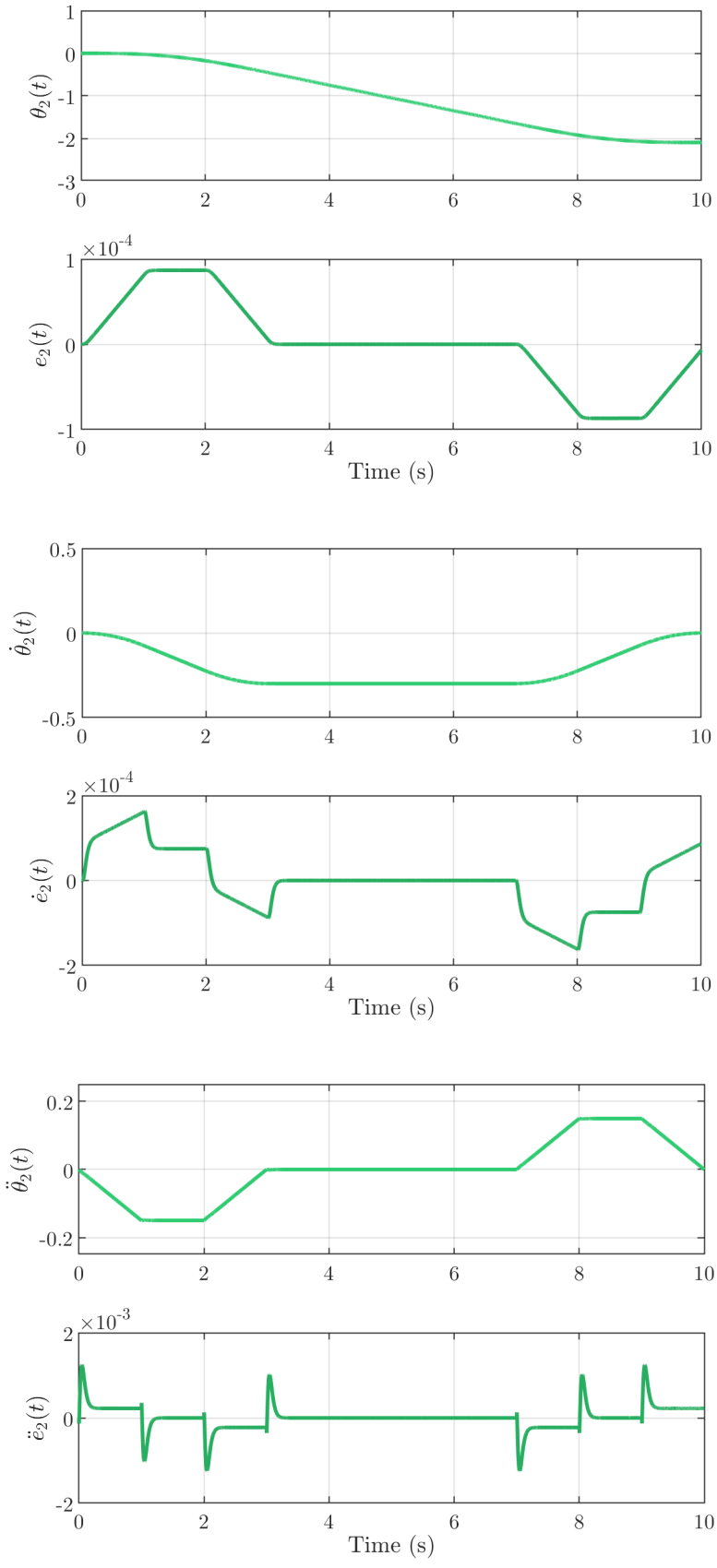

Figure 5.8: Joint-space trajectory tracking: Joint 2

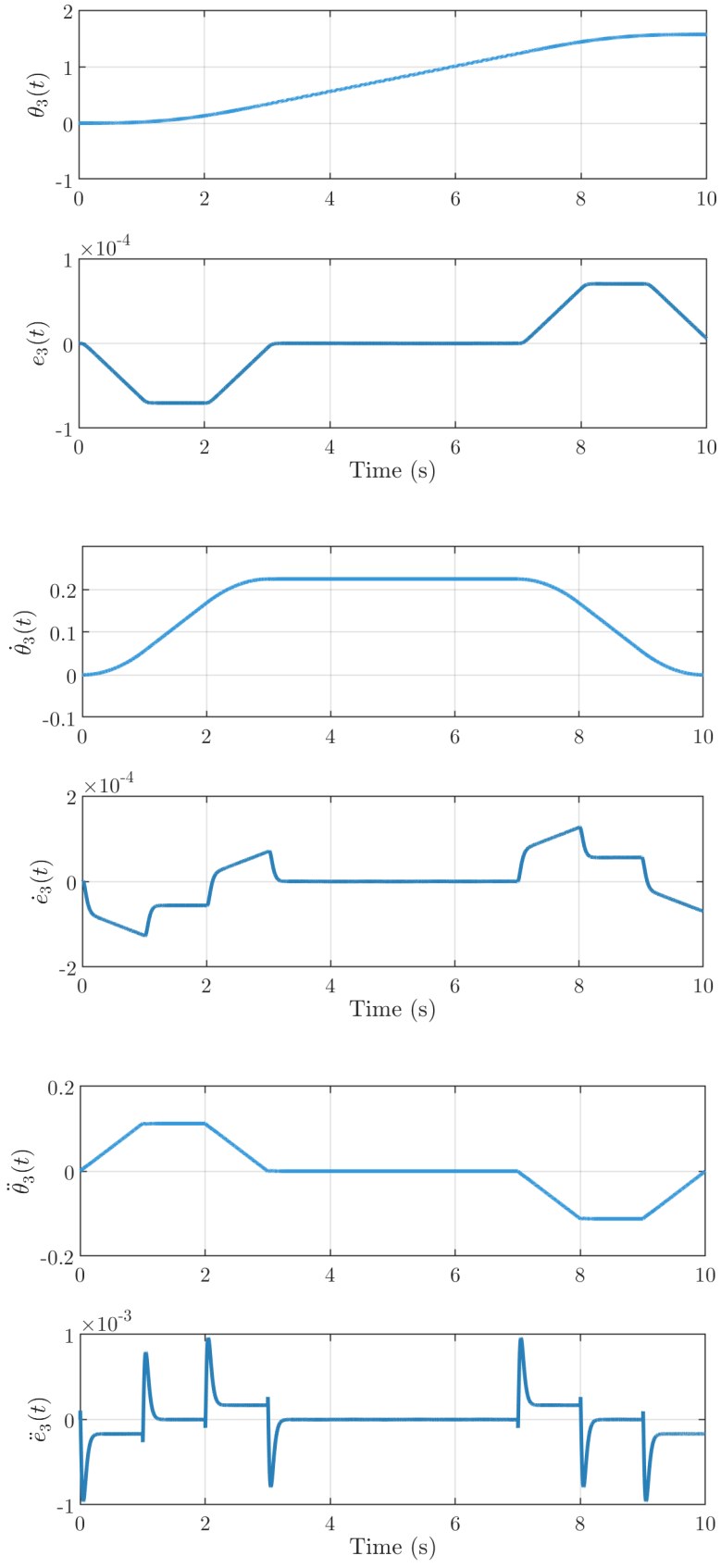

Figure 5.9: Joint-space trajectory tracking: Joint 3

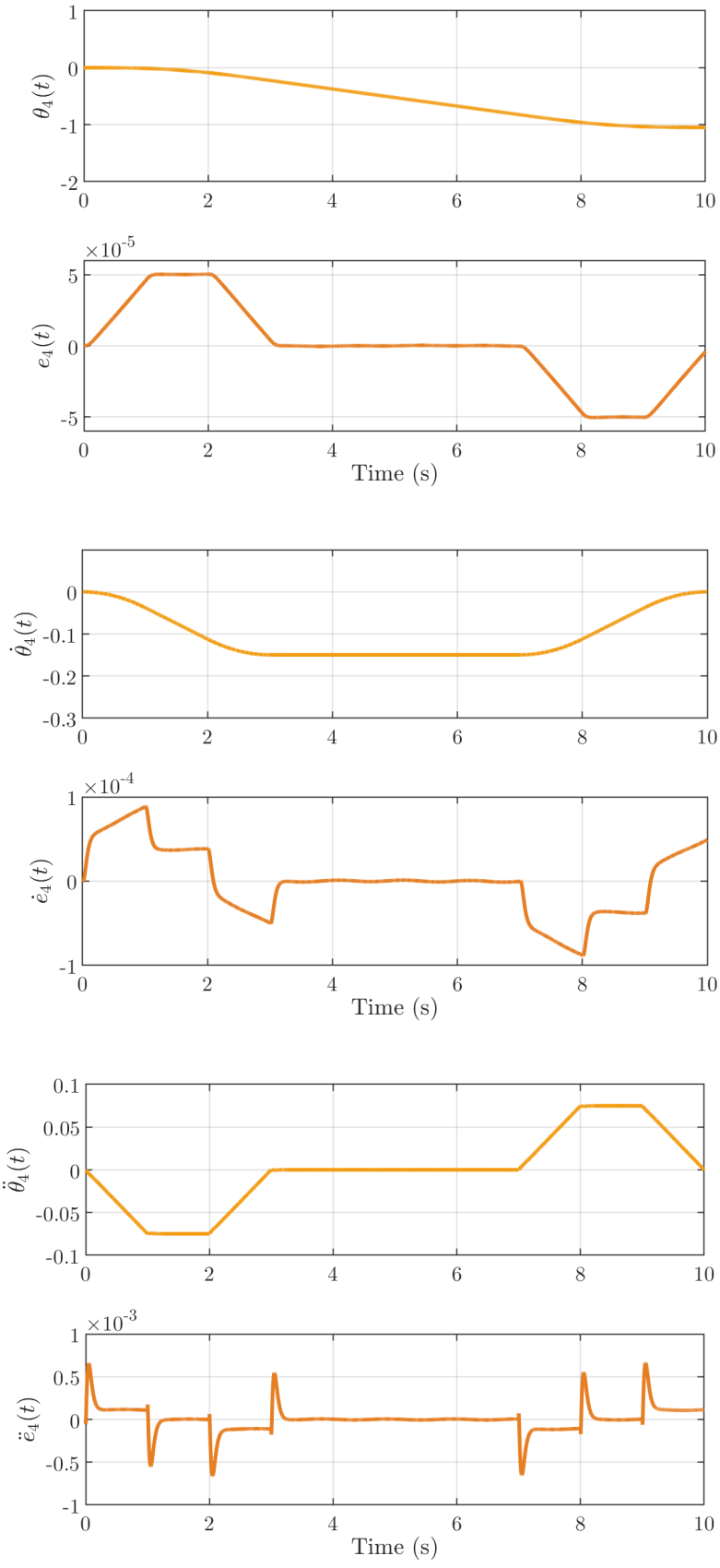

Figure 5.10: Joint-space trajectory tracking: Joint 4

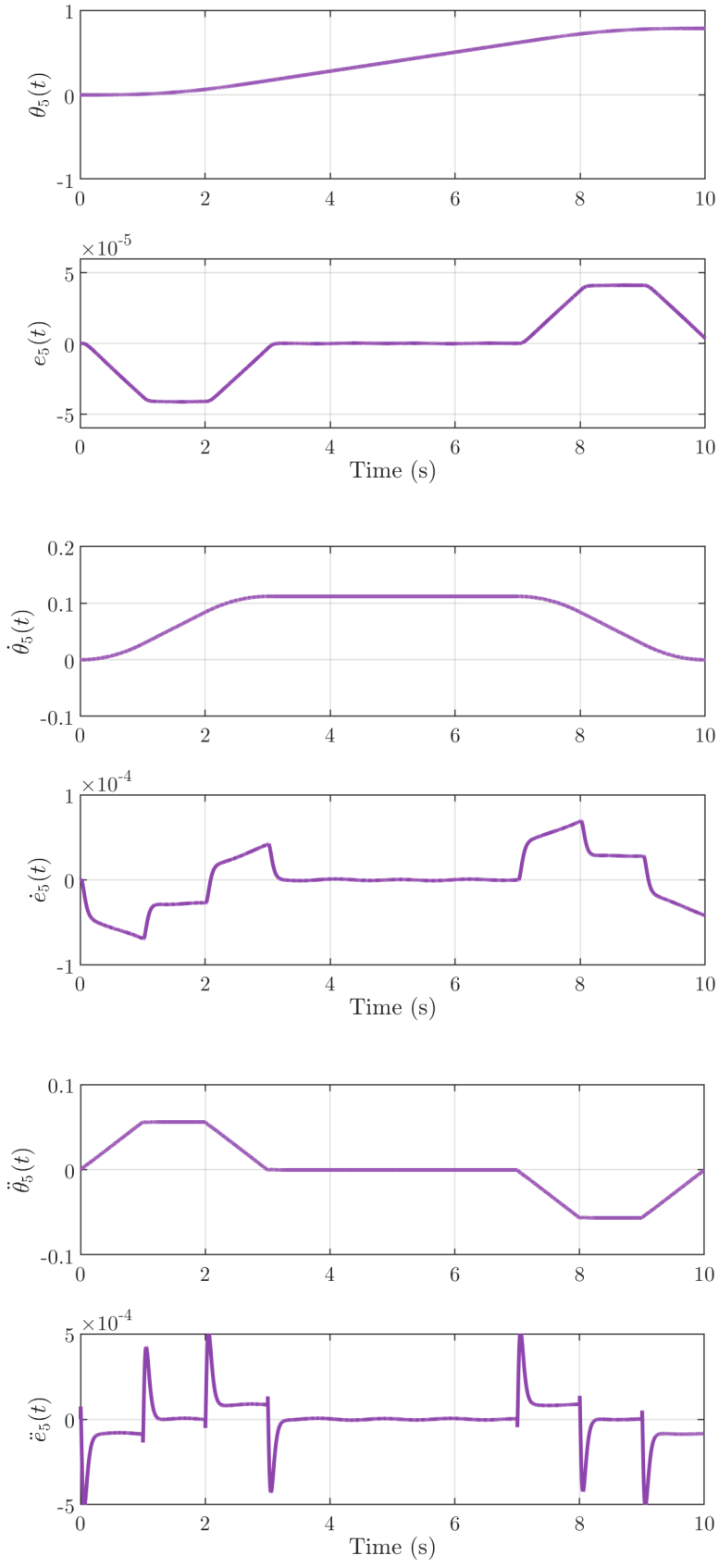

Figure 5.11: Joint-space trajectory tracking: Joint 5

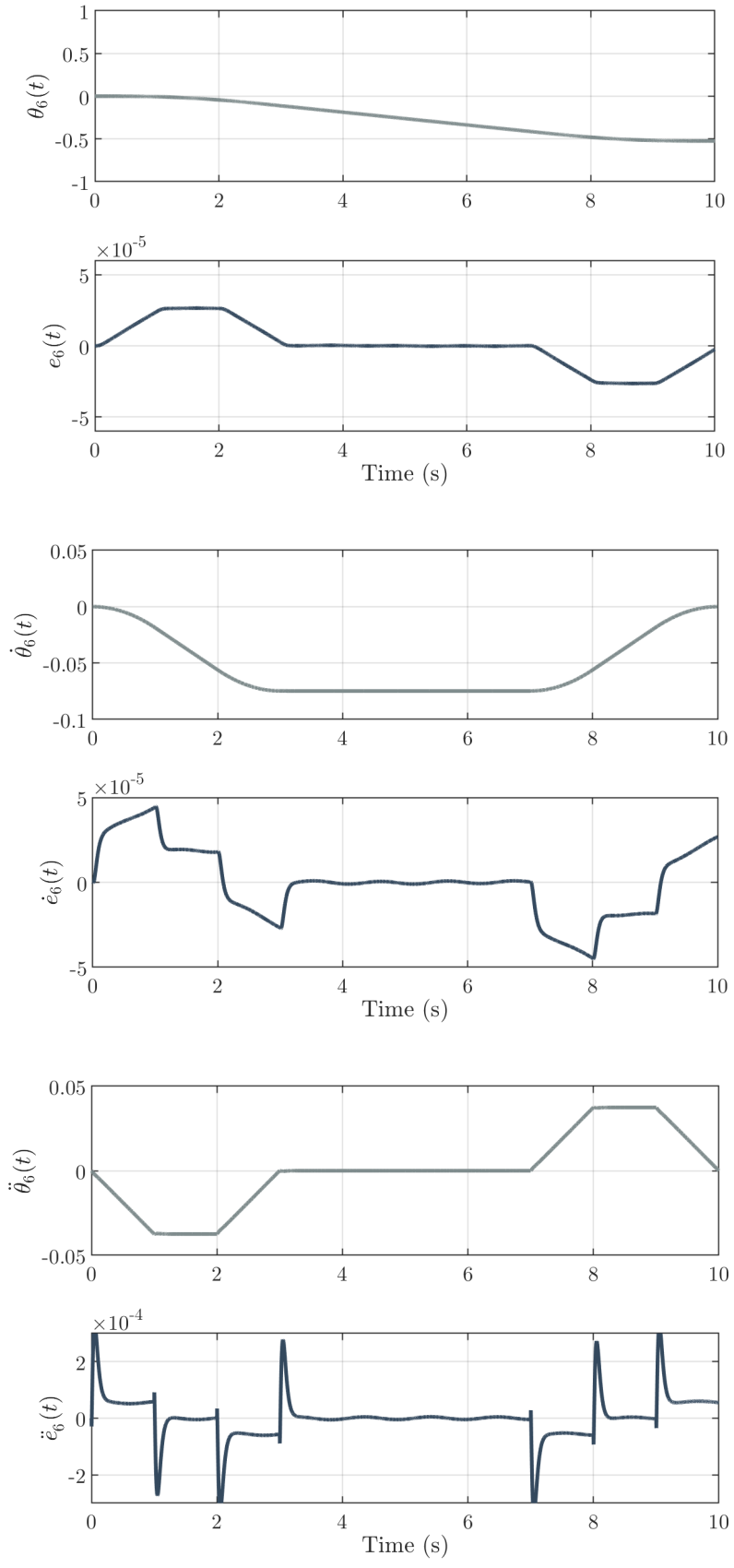

Figure 5.12: Joint-space trajectory tracking: Joint 6

Outer-loop Control Signals

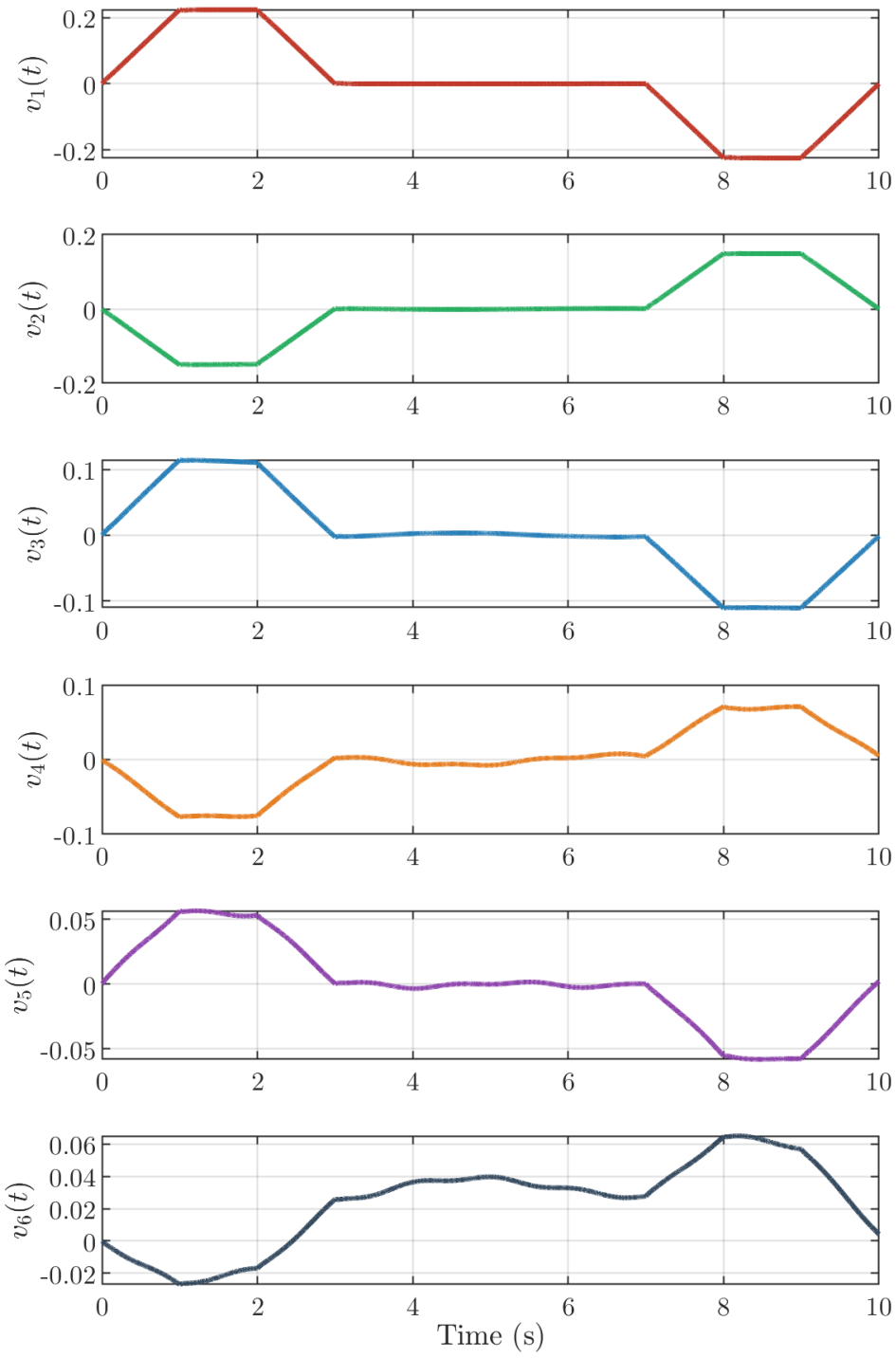

Figure 5.13: Outer-loop control signals

Joint Input Torque

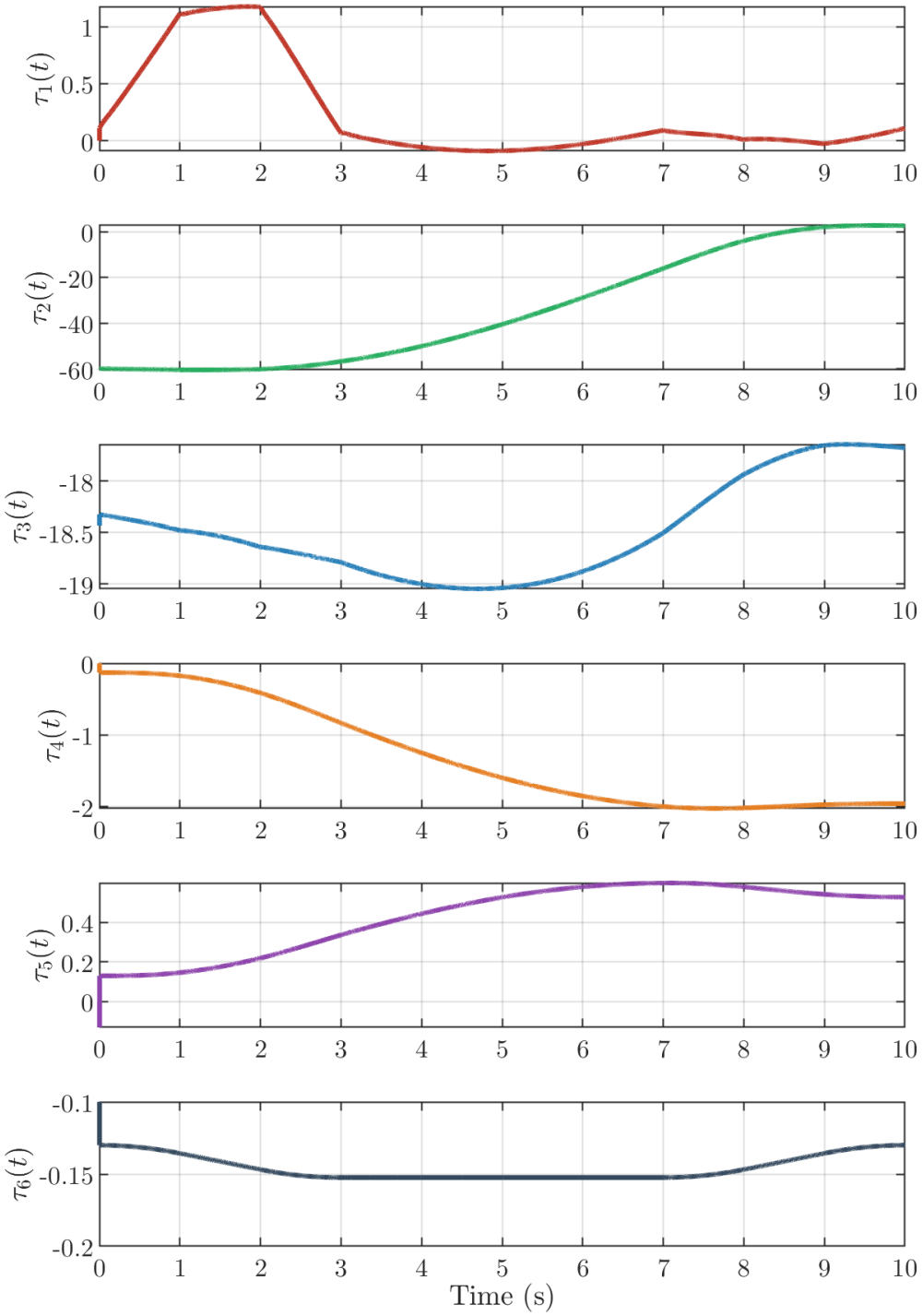

Figure 5.14: Joint input torque

Task-Space Trajectory Tracking

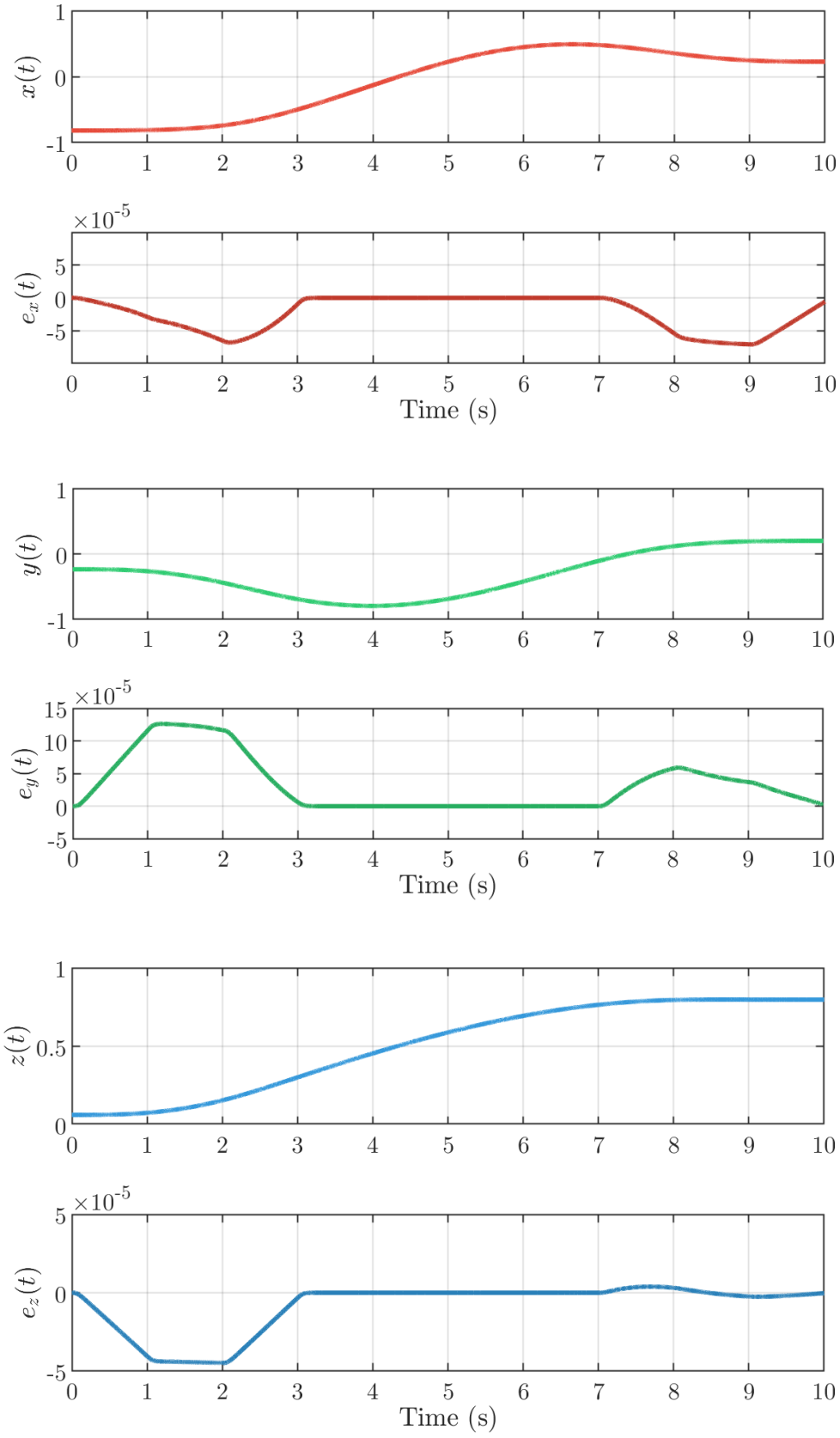

Figure 5.15: Task-space trajectory tracking: Displacement

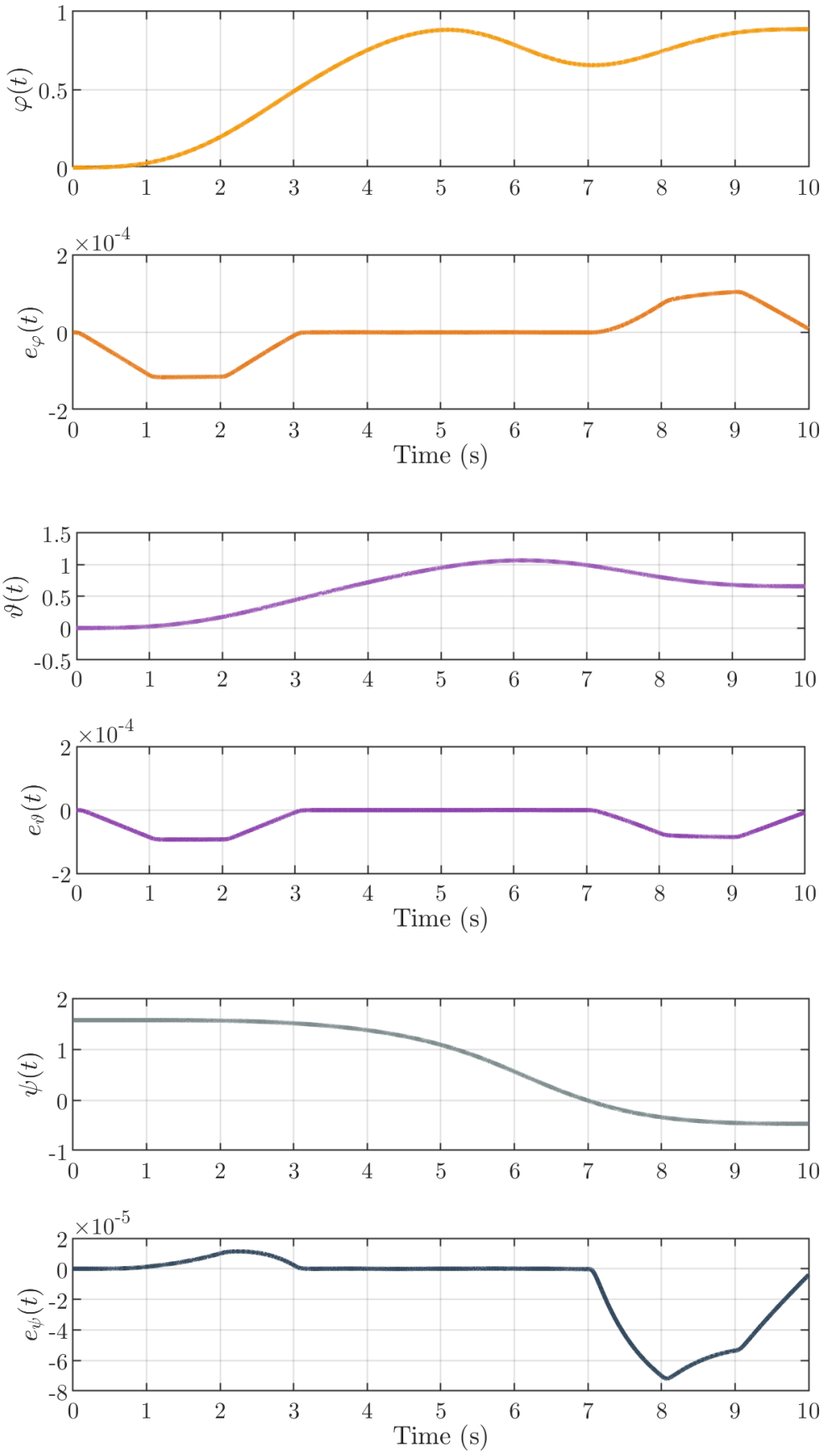

Figure 5.16: Task-space trajectory tracking: Orientation

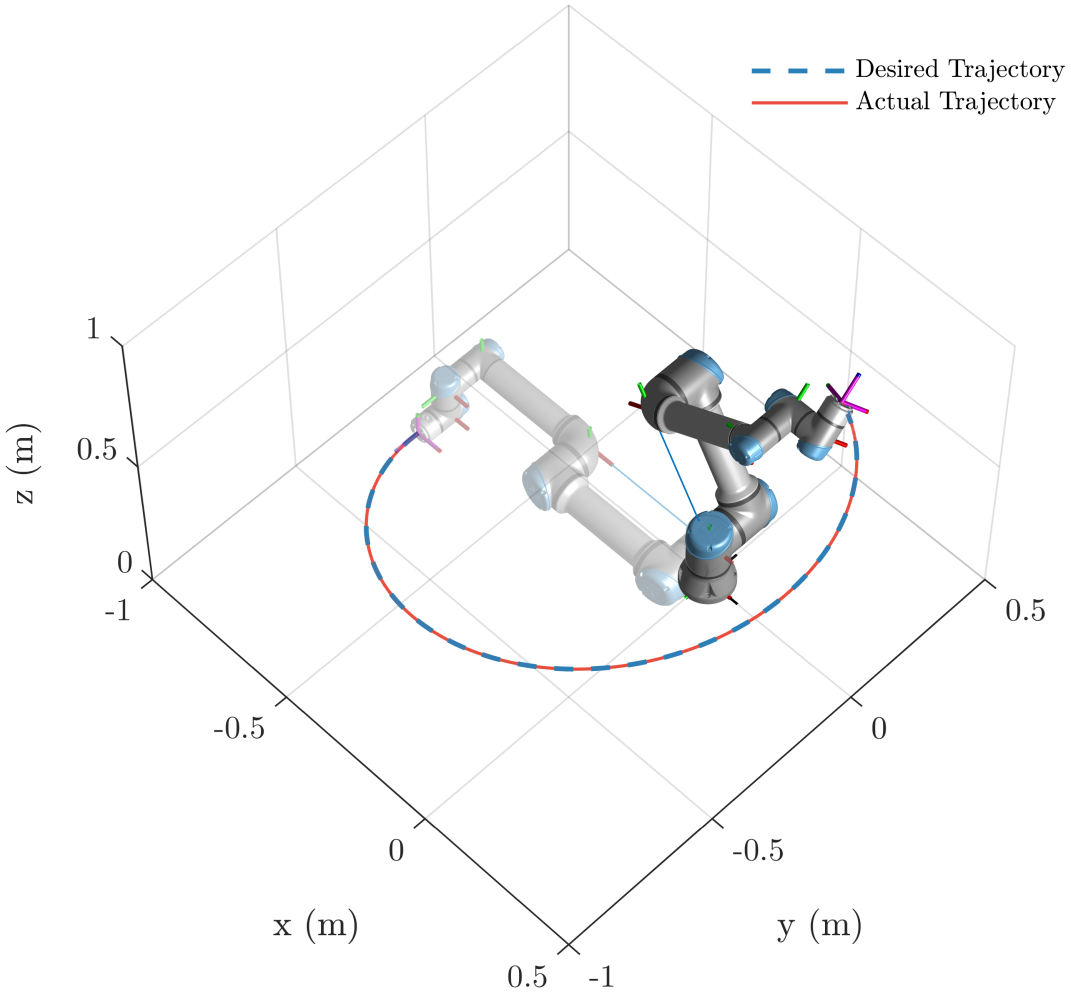

Figure 5.17: Task-space trajectory tracking: 3D representation

5.3.3 Results Discussion

As we extend our analysis further, it becomes evident that the proposed DDPG reinforcement learning-based PID gain tuner has significant advantages over conventional control approaches. The ability to adapt and fine-tune PID gains dynamically based on the robot's dynamics and task requirements leads to improved control precision and disturbance rejection capabilities. The presented results in Figures 5.7 to 5.17 provide compelling evidence of the method's effectiveness and its applicability to real-world scenarios.

In Figures 5.7 to 5.12, we observe the remarkable tracking performance of the robot's joint positions, velocities, and accelerations. The maximum errors in position, velocity, and acceleration are kept within very tight bounds, indicating a robust control system that can handle uncertainties and disturbances effectively. This level of accuracy and precision is essential in various applications, such as manufacturing, assembly, and surgical robotics, where positional accuracy directly impacts the quality and safety of the task execution.

In Figure 5.13, the outer loop control signals exhibit an adaptive response to the changing dynamics and disturbances in the environment. By dynamically adjusting the PID gains, the control system successfully compensates for uncertainties, keeping the actuators within safe operating limits (as shown in Figure 5.14). This ensures the longevity of the robot's hardware and prevents any detrimental effects caused by excessive torque demands.

Figures 5.15 and 5.16 highlight the impressive task-space trajectory tracking performance. The robot precisely follows the desired Cartesian displacements and orientations, crucial for applications like pick-and-place tasks, painting, or welding, where maintaining accuracy in the workspace is of utmost importance. The minimized tracking errors demonstrate the efficiency of the proposed method in achieving high-fidelity motion control, even in the presence of disturbances.

Furthermore, Figure 5.17 portrays the robot's trajectory tracking performance in a simulated 3D environment. The smooth execution of the S-curve profile trajectory from the initial transparent configuration to the final solid configuration exemplifies the seamless and reliable motion generated by the control system. This capability is vital in scenarios where the robot must navigate complex paths and execute intricate maneuvers while maintaining stability and accuracy.

The results presented in this study not only validate the proposed DDPG reinforcement learning-based PID gain tuner but also open up new possibilities for future research and applications in robotics and control. Building on these achievements, potential extensions could explore the integration of adaptive control techniques or hybrid control strategies that combine machine learning and classical control methods. Additionally, real-world experimentation on physical robot platforms can further validate the system's performance and its potential in various industrial settings.

Moreover, investigating the scalability of the proposed method to more complex robotic systems with higher degrees of freedom could unlock exciting avenues in advanced automation and multi-robot coordination tasks. Furthermore, the study can be extended to include analyses of different types of disturbances, uncertainties, and environmental conditions to assess the system's resilience and adaptability in diverse scenarios.

Overall, the presented research exemplifies the synergy between traditional control theories, machine learning algorithms, and robotics, paving the way for advanced intelligent control systems that can drive the development of autonomous and highly capable robotic systems in the near future. The potential benefits of such systems are far-reaching, impacting fields such as industry, healthcare, exploration, and beyond. As the field of robotics continues to evolve, this study stands as a valuable contribution, inspiring further advancements and innovations that will shape the future of robotics and automation.

Conclusions and Perspectives

The subject at hand revolves around the development and evaluation of a novel control approach for robot manipulators using a combination of classical control techniques and cutting-edge reinforcement learning algorithms. Through the exploration of forward kinematics, dynamics, and PID Computed-torque control, the dissertation lays the foundation for understanding the fundamental aspects of robot motion and control.

The key innovation lies in the proposed DDPG reinforcement learning-based PID gain tuner, which showcases remarkable adaptability and efficiency in compensating for uncertainties and disturbances. By dynamically adjusting PID gains, the control system achieves precise tracking of joint positions, velocities, and accelerations, resulting in enhanced trajectory tracking performance in both joint space and task space.

The presented simulation results, spanning joint tracking, control signals, torque inputs, and task-space trajectory tracking, consistently demonstrate the robustness and effectiveness of the proposed approach. The ability to maintain accurate motion execution even in the presence of disturbances makes this control method highly suitable for real-world applications, particularly in industrial automation, manufacturing, and other safety-critical environments.

The research highlights the promising integration of classical control methodologies with advanced machine learning techniques, leading to intelligent and adaptable robotic systems. This synergy empowers robots to navigate complex tasks with precision and resilience, showcasing their potential in various industries and domains.

As we consider the implications of this study, we recognize the broader impact of intelligent control systems on shaping the future of robotics and automation. The ability to continuously learn and optimize control strategies opens up new possibilities for autonomous and collaborative robots, enabling them to interact with uncertain environments and respond to dynamic changes effectively.

Moreover, the study serves as a stepping stone for future research in this domain. The potential extensions include exploring the integration of adaptive control techniques, sensor fusion, and multi-robot coordination to further enhance the performance and scalability of the proposed method.

In conclusion, this work represents a significant contribution to the fields of robotics, control systems, and artificial intelligence. The findings underscore the power of combining classical control knowledge with state-of-the-art machine learning algorithms to create intelligent and agile robotic systems capable of tackling real-world challenges. As robotics continues to advance, the insights from this research will continue to drive innovation, shaping the development of autonomous and capable robots in the years to come.

Bibliography

[1] Asada, H., & Slotine, J.J. (1986). Robot analysis and control. John Wiley & Sons.

[2] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[3] Bouhamed, O., Ghazzai, H., Besbes, H., & Massoud, Y. (2020, October). Autonomous UAV navigation: A DDPG-based deep reinforcement learning approach. In 2020 IEEE International Symposium on Circuits and Systems (ISCAS) (pp. 1-5). IEEE.

[4] Dong, Y., & Zou, X. (2020, October). Mobile Robot Path Planning Based on Improved DDPG Reinforcement Learning Algorithm. In 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS) (pp. 52-56). IEEE.

[5] Lewis, F.W., Jagannathan, S., & Yesildirak, A. (2020). Neural network control of robot manipulators and non-linear systems. CRC press.

[6] Spong, M.W., Hutchinson, S., & Vidyasagar, M. (2020). Robot modeling and control. John Wiley & Sons.

[7] Khalil, W., & Dombre, E. (2004). Modeling, identification and control of robots. Butterworth-Heinemann.

[8] Lynch, K.M., & Park, F.C. (2017). Modern Robotics Mechanics, Planning, and Control.

[9] Middletone, R.H., & Goodwin, G.C. (1986, December). Adaptive computed torque control for rigid link manipulators. In 1986 25th IEEE Conference on Decision and Control (pp. 68-73). IEEE.

[10] Lewis, F.L., Dawson, D.M., & Abdallah, C.T. (2003). Robot manipulator control: theory and practice. CRC Press.

[11] Rastogi, E., & Prasad, L.B. (2015, December). Comparative performance analysis of PD/PID computed torque control, filtered error approximation based control and NN control for a robot manipulator. In 2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON) (pp. 1-6). IEEE.

[12] Kebria, P.M., Al-Wais, S., Abdi, H., & Nahavandi, S. (2016, October). Kinematic and dynamic modelling of UR5 manipulator. In 2016 IEEE international conference on systems, man, and cybernetics (SMC) (pp. 004229-004234). IEEE.

[13] Kufieta, K. (2014). Force estimation in robotic manipulators: Modeling, simulation and experiments. Department of Engineering Cybernetics NTNU Norwegian University of Science and Technology.

[14] Perrusquía, A., Yu, W., & Li, X. (2021). Multi-agent reinforcement learning for redundant robot control in task-space. International Journal of Machine Learning and Cybernetics, 12(1), 231-241.

[15] Nie, J., Zhou, Y., Chen, C., Han, N., & Li, D. (2014). A PID parameter tuning method for air conditioning system based on annealing genetic algorithm. International Journal of Computer Applications in Technology, 50(3-4), 264-269.

[16] Misir, D., & Malki, H.A. (2006). Liapunov stability for a fuzzy PID controlled flexible-joint manipulator. International journal of computer applications in technology, 27(2-3), 97-106.

[17] Pham, H.V., & Lam, T.N. (2019). A new method using knowledge reasoning techniques for improving robot performance in coverage path planning. International Journal of Computer Applications in Technology, 60(1), 57-64.

[18] Burkan, R. (2019). A method for modelling of parameter estimation laws depending on functions for adaptive control of robot manipulators. International journal of computer applications in technology, 41(3-4), 205-229.

[19] Wang, H., & Yu, S. (2010). Tracking control of robot manipulators based on orthogonal neural network. International Journal of Modelling, Identification and Control, 11(1-2), 130-135.

[20] Zhang, J., Cheng, L., Wang, T., Xia, W., Yan, D., Wu, Z., & Duan, X. (2019). A welding manipulator path planning method combining reinforcement learning and intelligent optimisation algorithm. International Journal of Modelling, Identification and Control, 33(3), 261-270.

[21] Guo, W., Pan, T., Li, Z., & Li, G. (2020). A review on data-driven approaches for industrial process modelling. International Journal of Modelling, Identification and Control, 34(2), 75-89.

[22] Ayman, M., & Soliman, M. (2020). Robust pole-placer power system stabilisers design via complex Kharitonov's theorem. International Journal of Modelling, Identification and Control, 34(3), 197-207.

[23] Gupta, A., & Manocha, A.K. (2020). Performance enhancements of physical systems by reduced-order modelling and simulation. International Journal of Modelling, Identification and Control, 36(1), 14-23.

[24] Viriyapong, R., & Tavaen, S. (2020). Global stability and optimal control of melioidosis transmission model with hygiene care and treatment in human and animal populations. International Journal of Modelling, Identification and Control, 34(4), 301-315.

[25] Zhang, W., Zhang, Z., Li, Q., & Xue, Z. (2020). Multiple U-model control of uncertain discrete-time nonlinear systems. International Journal of Modelling, Identification and Control, 34(4), 293-300.

[26] Arenas-López, J.P., & Badaoui, M. (2020). The Ornstein-Uhlenbeck process for estimating wind power under a memoryless transformation. Energy, 213, p.118842.

[27] Martin, D., O'Byrne, J., Cates, M.E., Fodor, É., Nardini, C., Tailleur, J., & van Wijland, F. (2021). Statistical mechanics of active Ornstein-Uhlenbeck particles. Physical Review E, 103(3), p.032607.

[28] Lamamra, K., Batat, F., & Mokhtari, F. (2020). A new technique with improved control quality of nonlinear systems using an optimized fuzzy logic controller. Expert Systems with Applications, 145, p.113148.

[29] Raviola, A., Martin, A.D., Guida, R., Pastorelli, S., Mauro, S., & Sorli, M. (2021, June). Identification of a UR5 Collaborative Robot Dynamic Parameters. In International Conference on Robotics in Alpe-Adria Danube Region (pp. 69-77). Springer, Cham.

[30] Kovincic, N., Müller, A., Gattringer, H., Weyrer, M., Schlotzhauer, A., & Brandstötter, M. (2019, May). Dynamic parameter identification of the Universal Robots UR5. In Proceedings of the ARW & OAGM Workshop (pp. 44-53).

[31] Lamamra, K., & Belarbi, K. (2011). Comparison of neural networks and fuzzy logic control designed by multi-objective genetic algorithm. IJACT: International Journal of Advancements in Computing Technology, 3(4), 137-143.

[32] Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., & Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. arXiv preprint arXiv:1707.08817.

[33] Wu, X., Liu, S., Zhang, T., Yang, L., Li, Y., & Wang, T. (2018, August). Motion control for biped robot via DDPG-based deep reinforcement learning. In 2018 WRC Symposium on Advanced Robotics and Automation (WRC SARA) (pp. 40-45). IEEE.

[34] Ghediri, A., Lamamra, K., Ait Kaki, A., & Vaidyanathan, S. (2022). Adaptive PID computed-torque control of robot manipulators based on DDPG reinforcement learning. Int. J. Model. Identif. Control., 41(3), 173-182.

[35] Denavit, Jacques, Hartenberg, Richard Scheunemann (1955). "A kinematic notation for lower-pair mechanisms based on matrices". Journal of Applied Mechanics. 22 (2): 215–221.

[36] Urbakh, M., Klafter, J., Gourdon, D., & Israelachvili, J. (2004). The nonlinear nature of friction. Nature, 430(6999), 525-528.

[37] Baum, E. B. (1988). On the capabilities of multilayer perceptrons. Journal of complexity, 4(3), 193-215.

[38] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. Towards Data Sci, 6(12), 310-316.

[39] Janocha, K., & Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. arXiv preprint arXiv:1702.05659.

[40] LeCun, Y., Touresky, D., Hinton, G., & Sejnowski, T. (1988, June). A theoretical framework for back-propagation. In Proceedings of the 1988 connectionist models summer school (Vol. 1, pp. 21-28).

[41] Soydaner, D. (2020). A comparison of optimization algorithms for deep learning. International Journal of Pattern Recognition and Artificial Intelligence, 34(13), 2052013.

[42] Filar, J., & Vrieze, K. (2012). Competitive Markov decision processes. Springer Science & Business Media.

[43] Littman, M. L. (2001). Value-function reinforcement learning in Markov games. Cognitive systems research, 2(1), 55-66.

[44] Xu, Z., Yang, D., Tang, J., Tang, Y., Yuan, T., Wang, Y., & Xue, G. (2020). An actor-critic-based transfer learning framework for experience-driven networking. IEEE/ACM Transactions on Networking, 29(1), 360-371.

[45] Pfau, D., & Vinyals, O. (2016). Connecting generative adversarial networks and actor-critic methods. arXiv preprint arXiv:1610.01945.

[46] Zhang, S., & Sutton, R. S. (2017). A deeper look at experience replay. arXiv preprint arXiv:1712.01275.