UNIVERSITÀ
DI PAVIA

Faculty of Engineering

Department of Electrical, Computer and Biomedical Engineering

Ph.D. Thesis

Self-configuring Robotic Systems: the use of Deep Reinforcement Learning as a tool for Industrial Manipulators

A.Y. 2019/2020

Cycle XXXIII

Candidate:
Bianca Sangiovanni, M.Eng.

Advisor:
Professor Antonella Ferrara, Ph.D.

Abstract

The present Thesis aims to present different examples of self-configuring systems involving robotic manipulators. With self-configuring systems, we refer to systems featuring the ability to autonomously adapt their control strategy or general configuration in order to overcome possible limitations encountered during their operation. To this end, the use of Deep Reinforcement Learning (DRL), a type of machine learning that enables the system to autonomously discover the best strategy to solve a task, is discussed as a tool for decision making or for controlling robotic systems in situations in which a model-based solution is not readily achievable. End-to-End, model-free control approaches are studied and used contextually with - or alternatively to - model-based control and decision strategies. Specifically, the work focuses on industrial manipulators operating in uncertain environments to perform motion and collision avoidance tasks.

For robot motion, a novel switched-structure scheme to achieve both centralized and decentralized control, using the perturbation estimation feature of the Integral Sliding Mode controller, is presented and discussed in detail. Different approaches for decision-making are illustrated.

For collision avoidance, a framework enabling end-to-end, model-free control of robotic manipulators operating in cluttered environments is introduced and tested on several case studies. A novel hybrid algorithm combining DRL-based strategies and conventional planning methods is proposed. The presented approaches are then deployed on different robotic systems, and experimental results are reported.

Sommario

Lo scopo della presente Tesi è quello di presentare diversi esempi di sistemi autoconfiguranti *(self-configuring)* che coinvolgono manipolatori robotici. Con sistemi autoconfiguranti, si intendono sistemi che hanno la capacità di adattare la propria strategia di controllo, o la propria configurazione generale, per superare eventuali limitazioni incontrate durante il loro funzionamento. A questo scopo, l'uso del Deep Reinforcement Learning (DRL), un tipo di machine learning che permette al sistema di scoprire autonomamente la strategia migliore per compiere un determinato task, viene introdotto e discusso come strumento per prendere decisioni o per controllare sistemi robotici in situazioni in cui una soluzione con controlli convenzionali *model-based* non è facilmente raggiungibile. Gli approcci di controllo *end-to-end, model-free*, sono studiati e utilizzati contestualmente - o alternativamente - a strategie di controllo e decisione convenzionali. Il lavoro si concentra, in particolare, su manipolatori industriali che operano in condizioni incerte, per eseguire operazioni di moto e *collision avoidance.*

Per il controllo del moto, viene presentato e discusso in dettaglio un nuovo schema a struttura switched per ottenere un controllo sia centralizzato che decentralizzato, utilizzando la proprietà di stima delle perturbazioni del controllore basato su Integral Sliding Mode. Vengono illustrati diversi approcci per il processo decisionale, con cui si determina la scelta del controllore.

Per la collision avoidance, viene introdotto un framework per il Deep Reinforcement Learning che permette il controllo end-to-end e model-free di manipolatori robotici che operano in ambienti occupati da ostacoli. L'approccio proposto viene testato su diversi casi di studio. Viene proposto inoltre un nuovo algoritmo ibrido che combina strategie basate su DRL e metodi di pianificazione convenzionali. Le soluzioni presentate sono poi testate su diversi sistemi robotici reali, per cui vengono alcuni risultati sperimentali.

Acknowledgements

This manuscript represents the culmination of my three-years journey as a Ph.D. student, a journey that allowed me to mature as a scholar and as a human. This would have not been possible without the many people that I met and those who supported me. First and foremost, I wish to express the most profound gratitude to my Ph.D. advisor Professor Antonella Ferrara for her mentorship and for allowing me to pursue my doctoral studies, but mostly for her support and inspiration. I wish to deeply thank Professor Gian Paolo Incremona and Professor Marco Piastra for the very meaningful collaborations during these years and for being great examples of professionalism and competence. Thank you to all the incredible lab mates at the *ICDS Lab* for sharing all the laughs, struggles, many memorable moments, and overall great days. Massimo, Giulia, Emanuele, Andrea, Alessandro, but also Federica, Enrico, Eleonora, Jack, Alessio, Francesca, Giacomo, Diego and Behrouz, you are all incredibly brilliant and I wish you all the best. Special thanks to Professor Chiara Toffanin for always being so patient, lab technician Gianluca De Felici for his assistance, and Nikolas Sacchi for his support in setting up the experiments and the fun times discussing robotics. I wish to thank all the students I had the honor to co-tutor: Simone Colosi, Alessio Stefanini and Marco Guido. I also wish to express sincere gratitude to all the members of the *Robotics and Perception Group* at the University of Zurich/ETH, especially Professor Davide Scaramuzza, Mrs. Tamar Tolcachier, Davide Falanga, and Antonio Loquercio for our collaboration during my visiting semester: it was a pleasure and an honor for me to join such incredible lab. Furthermore, I would like to thank Professor Paolo Rocco and Professor Matteo Rubagotti for agreeing to review my thesis and providing valuable feedback and suggestions to improve the manuscript.

Last but not least, I want to thank with all my heart my wonderful friends and family for always supporting me during the pursuit of my Ph.D.; honorable mention to Fra, Alesja, Angelo, Zambi, and Max for the epic moments, and to my mom, dad and sister Sonia for being amazing. Finally, my most profound acknowledgement goes to Cre: thank you for always being with me. You are the love and joy of my life.

Contents

List of Figures

List of Tables

Part I

Introduction and Background

Chapter 1

Introduction

Since the development of the first industrial manipulators in 1959 by George Devol and Joseph Engelberger, robots have seen an ever-increasing presence in industries: the consistent development in the automation field has, indeed, created a fertile environment for the deployment of robotics solutions that are now able to perform a plethora of tasks, both independently and alongside humans. As reported by the *International Federation of Robotics (IFR)*, in 2019, a record of 2.7 million robots were operative in factories worldwide [1]: this is consistent with the trend of the last 10 years (Figure 1.1), in which the operational stock of industrial robots has seen a steady rise, accelerated by the advent of the fourth industrial revolution (*Industry 4.0)*. Furthermore, although still in its early stages, the application of collaborative robotics (*cobots*) in industry is on the rise, with a 4.8% market share of newly installed industrial robots of 2019, still according to the IFR World Robotics Report 2020 [1].

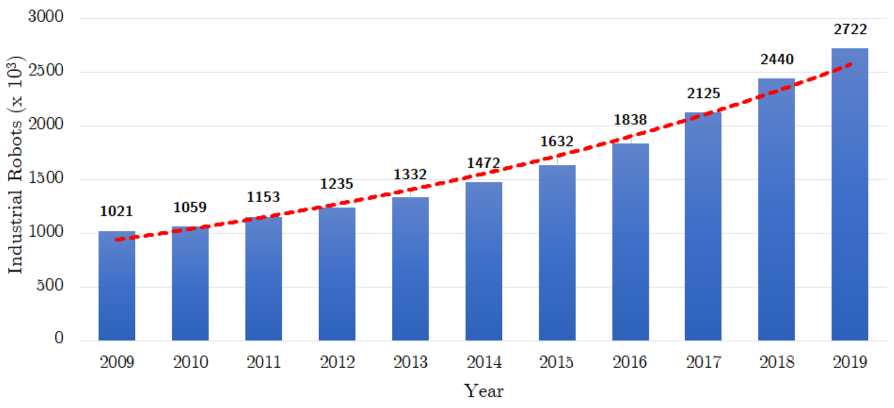


Figure 1.1: The worldwide operational stock of industrial robots

Figure 1.2: Industrial manipulators working in a factory (photo credits to KUKA Roboter GmbH, Bachmann)

Indeed, as robots and automated systems see increasing popularity, there is a necessity for them to coexist and adapt to other elements in the environment, such as humans, machinery, or, more generally, to unpredictable variations of their working conditions. As such, a high degree of flexibility is required in order to guarantee satisfactory performance during operations. For these reasons, many proposals have been made available by research on collaborative robotics: the requirements for human-robot interaction ensuring safety are discussed in [2]; in [3], then, several approaches are presented to ensure safety during the robot's task execution, as well as in [4], where a framework for maximizing the productivity of collaborative robotics while enforcing safety is introduced; in [5] instead the problem is tackled from a task-allocation perspective, proposing a hierarchical framework in which humans and machines coexist.

Besides the vast dedicated research field of collaborative robotics, an interesting perspective on systems flexibility, even when the system is not natively designed for cooperation like 'classical' manipulators, can be given by the concept of *self-configuring systems*. Generally speaking, with self-configuring systems we refer to those systems that can autonomously adapt their control strategy or general configuration (such as morphology or type of operation). The research field of hybrid control and switched systems, for example, presents extended literature on the matter: in [6] a general strategy to control different types of hybrid systems as optimal control problems is introduced; again in [7], a switching logic is implemented

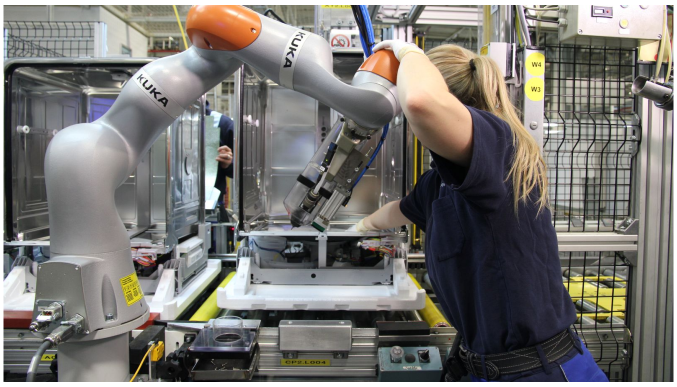


Figure 1.3: A human operator working closely with a KUKA cobot (photo credits to the International Federation of Robotics)

in order to choose among different controllers, depending on arbitrary metrics, while in [8] a switching control framework is designed to stabilize a system with changing dynamics. Apart from hybrid control or dynamics, robots that self-configurate by autonomously changing their shape to adapt to environments are then vastly researched in the fields of soft robotics, as in [9], where a robot can change its shape to achieve locomotion over different types of surfaces, or exploration, such as in [10] where aerial robots are designed in order to adapt their morphology during flight.

Although many notable examples and contributions using state of the art methodologies are provided in the very lively and broad field of robotics, it is not always possible to have an accurate enough knowledge of the system into consideration or the environment in which it operates. Thus, conventional model-based methods can present shortcomings when facing behaviors that were not taken into account during the control or model design phase. To this end, in recent years, the field of Reinforcement Learning, and its evolution in *Deep* Reinforcement Learning, has been gaining significant momentum in robotic applications. In fact, thanks to its generalization capabilities, it enables model-free, end-to-end control of systems without the need to hand-engineer specific procedures.

1.1 Deep Reinforcement Learning for Robotics

Deep Reinforcement Learning (DRL), following the work of Mnih et al. in 2015 [11], has been researched as a promising approach for solving hard to hand-engineer tasks, such as cases where it is difficult to have an accurate enough model description of the considered system. For robotic applications, this translates into letting the robot interact with its environment so that, for any given control operation, it is possible to find an (ideally) optimal strategy to conclude the task successfully. Similarly to conventional optimal control, Reinforcement Learning also aims to solve the problem of finding the optimal policy to maximize a given cost function (or reward), relying on measured data [12, 13]. Nevertheless, many challenges rise when dealing with reinforcement learning for systems that have a high number of states and actions: the so-called *curse of dimensionality* [14] refers to the infeasibility to map every state and action to their *values*, needed to find the policy to be used. Furthermore, although RL approaches have been successfully applied to systems with discrete action space, many real-world applications require an agent to select optimal actions from continuous spaces. Indeed, discrete actions could not be adequate for devising strategies where a tiny change in action can significantly affect the outcome. A possible solution to some problems arising with the use of reinforcement learning in robotics is given by Deep Learning [15], which can be used to train general-purpose deep neural networks in place of hand-defined policies. Thanks to this, and the improved computational capabilities available, in recent years reinforcement learning has been successfully adapted to systems featuring continuous state and action spaces. Actor-critic, model-free methods built on Q-learning have been proposed, such as Deep Deterministic Policy Gradient (DDPG) [16], Twin Delayed DDPG (TD3) [17] and Normalized Advantage Function (NAF) [18], which have since been employed for end-to-end control of robotics systems. In [19, 20], for instance, such algorithms have been used to train the agent to grasp sparse objects via convolutional neural networks for pose estimation. In [21], and then in [22], data-efficient methodologies for dexterity operations have been applied. In [23], further improvements have been made for enabling a robot to accomplish a stacking task by using soft Q-learning. Yet, despite the recent improvements, DRL for robotics systems raises concerns over safety issues: long training times are required, and a high level of domain randomization is needed to prevent undesirable effects and unexpected behaviors due to the systems' complexity, which could enter a combination of states and actions that were not explored during training, entailing unpredictable outcomes. Nevertheless, training on a physical system is often impossible due to possible damages caused by hardware wearing and significant energy consumption. For this reason, also known as the *curse of real-world samples* [12], the majority of the research in the field is carried out in simulation [24]. Therefore, despite the recognized potential, it is still difficult and expensive to achieve near-optimal behaviors using fully autonomous, end-to-end control strategies with DRL. A possible solution to alleviate these problems and obtain *"the best of both worlds"* - in which the "worlds" are standard, conventional control and decision methods and model-free, end-to-end control with DRL - would be to find a way to combine different approaches, using DRL as a tool to support conventional methods, and vice-versa.

1.2 Thesis contribution

The aim of this Thesis is to present and discuss different ways in which robotic applications, with a focus on robotic manipulators, can be adapted so that the system achieves self-configuring capabilities; more specifically, the topics of motion control and collision avoidance will be explored in this context, as summarized in the following subsections. Furthermore, Deep Reinforcement Learning is applied in the proposed studies in order to enable both decision and control of systems operating under uncertain conditions.

Robot Motion

A novel switched structure control scheme based on Integral Sliding Mode control is presented and discussed in detail. Specifically, the designed architecture allows switching between a centralized approach and a decentralized one in order to control the motion of a robot manipulator. Two case studies, one involving a variable-gear-actuator robotic model and one including a robot identified on the basis of real data, are presented and the results are discussed. The same switched-structure control scheme is then adapted in order to enable online decision-making with an agent trained with DRL.

Collision Avoidance

A novel approach based on DRL, which allows end-to-end, model-free control of a robot operating in a cluttered environment is introduced and discussed in detail. The elements of the DRL framework and the training procedures are illustrated, discussing the effects of the so-called transfer learning applied to the considered scenarios. Simulation results for different case studies are reported and commented on. In addition to the end-to-end strategy, a hybrid dual-mode algorithm that combines conventional motion planning techniques and DRL-based control is introduced. The proposed solution aims to improve the tracking performances of the robot manipulator while allowing model-free control for collision avoidance. A switching metric, which confers self-configuring capabilities to the considered robotic system, is introduced, and results are discussed for different case studies. The proposed end-to-end framework is then applied to a teleoperation application. The manipulator is tasked with tracking a trajectory generated in real-time by a human operator while avoiding obstacles encountered during motion; experimental results are compared with a Model Predictive Control (MPC) approach. Lastly, the interface built in order to deploy the proposed approaches to an industrial manipulator present at the University of Pavia is introduced and described with preliminary experimental results.

1.3 Thesis structure

This dissertation, divided into four Parts, is structured as follows.

(I) Introduction: preliminary concepts on the main topics of interest of this Thesis are introduced. A background on robot models, motion planning, and the basics of Deep Reinforcement Learning are briefly discussed.

(II) Self-configuring control schemes for robot motion: the self-configuring, switched structure control architecture for motion control of robot manipulators is introduced and discussed in detail. Different applications of the proposed approach and the use of DRL for decision-making are presented. This Part of the thesis is based on the results published in the following papers.

- A. Ferrara, G.P. Incremona, and B. Sangiovanni. *Integral sliding mode based switched structure control scheme for robot manipulators.* In 2018 15th International Workshop on Variable Structure Systems (VSS), pages

168–173. Graatz, Austria, 2018.

- B. Sangiovanni, G.P. Incremona, A. Ferrara, and M. Piastra. *Deep reinforcement Learning based self-configuring integral sliding mode control scheme for robot manipulators.* In 2018 57th IEEE Conference on Decision and Control (CDC), pages 5969–5974, Miami Beach (FL), USA, December 2018.
- A. Ferrara, G. P. Incremona, and B. Sangiovanni. *Tracking control via switched integral sliding mode with application to robot manipulators.* Control Engineering Practice, 90, pages 257–266, 2019.

(III) Self-configuring approaches for robot collision avoidance: the proposed DRL framework for robot collision avoidance and its applications are introduced and discussed. The framework is then used in the context of a self-configuring approach to motion planning with obstacle avoidance, which combines conventional planning methods with end-to-end control. This Part of the thesis, especially **Chapters 9, 10, 11**, is based on the results published in the following papers.

- B. Sangiovanni, A. Rendiniello, G.P. Incremona, A. Ferrara, and M. Piastra. *Deep reinforcement learning for collision avoidance of robotic manipulators.* In 2018 16th European Control Conference (ECC), pages 2063–2068. Lymassol, Cyprus, July 2018.
- B. Sangiovanni, G. P. Incremona, M. Piastra, and A. Ferrara. *Self-configuring robot path planning with obstacle avoidance via deep reinforcement learning.* IEEE Control Systems Letters, 5(2), pages 397–402, 2021.

The work presented in **Chapter 12** was developed in collaboration with the *Astana Laboratory for Robotic and Intelligent Systems* of the Nazarbayev University, Nur-Sultan, Kazakhstan.

(IV) Conclusion: some concluding remarks are gathered, and some insights on possible future work stemming from the research presented are proposed.

1.4 List of peer-reviewed scientific publications

The complete list of peer-reviewed publications produced during the Ph.D. course as of January 2021 is reported.

Conference proceedings

- B. Sangiovanni, A. Rendiniello, G.P. Incremona, A. Ferrara, and M. Piastra. *Deep reinforcement learning for collision avoidance of robotic manipulators.* In 2018 16th European Control Conference (ECC), pages 2063–2068. Lymassol, Cyprus, July 2018.
- A. Ferrara, G.P. Incremona, and B. Sangiovanni. *Integral sliding mode based switched structure control scheme for robot manipulators.* In 2018 15th International Workshop on Variable Structure Systems (VSS), pages 168–173. Graatz, Austria, 2018.
- B. Sangiovanni, G.P. Incremona, A. Ferrara, and M. Piastra. *Deep reinforcement Learning based self-configuring integral sliding mode control scheme for robot manipulators.* In 2018 57th IEEE Conference on Decision and Control (CDC), pages 5969–5974, Miami Beach (FL), USA, December 2018.

Journals

- A. Ferrara, G.P. Incremona, and B. Sangiovanni. *Tracking control via switched integral sliding mode with application to robot manipulators.* Control Engineering Practice, 90, pages 257–266, 2019.
- B. Sangiovanni, G. P. Incremona, M. Piastra, and A. Ferrara. *Self-configuring robot path planning with obstacle avoidance via deep reinforcement learning.* IEEE Control Systems Letters, 5(2), pages 397–402, 2021.

Book chapters

- A. Ferrara, G.P. Incremona, B. Sangiovanni, *Sliding Mode Fault Diagnosis with Vision in the Loop for Robot Manipulators*, New Trends in Robot Control, pages 81-105, Springer, 2020.

Chapter 2

Preliminaries of Robotics

In this Chapter, preliminaries on the theory of robot modeling and planning will be introduced. First, basic definitions will be recalled in Section 2.1. Then, a more detailed description of a robotic model will be given in Sections 2.2 and 2.3, where basic concepts on both kinematic and dynamical modeling are provided. Then, the main concepts on the topic of motion planning for robots will be recalled in Section 2.4.

2.1 Basic Definitions

A *Robot* is a servo-mechanical system able to perform complex operations, often heavy, repetitive, or dangerous for human beings. There are different kinds of robots, [25], but a first distinction can be made between manipulation (industrial) robots and mobile robots. The former feature a fixed base and a certain Degree of Freedom (DoF), that is the number of independent parameters that define its physical configuration; fixed robots are typically designed to fit into industrial settings to perform tasks such as grasping and assembling. Mobile robots, on the contrary, are not linked to the ground and are designed to move around, using wheels, mechanical legs, rotors (in case of aerial robotics), or thrusters (in case of ROVs) to perform navigation tasks; they are usually employed for search and rescue operations, exploration, and surveillance tasks. The focus of this dissertation will be on industrial manipulators.

The main elements composing the mechanical structure of a robot manipulator, such as the one represented in Figure 2.1, are:

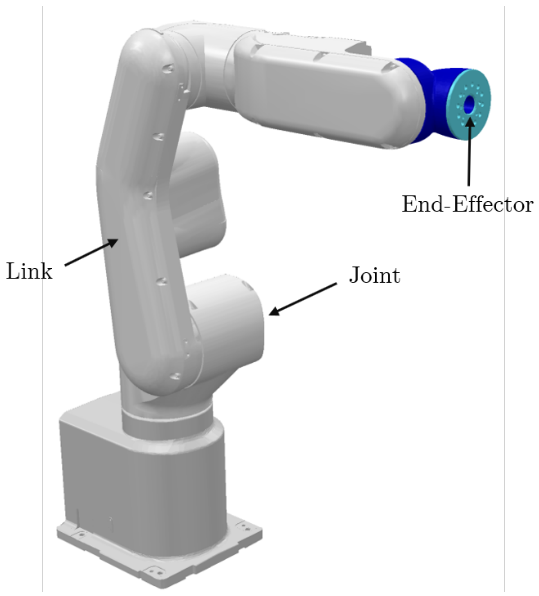
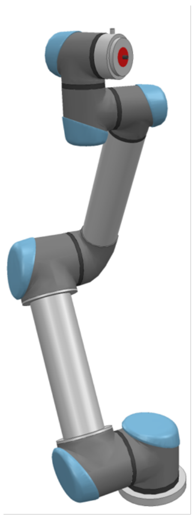


Figure 2.1: Main elements of the mechanical structure of an industrial manipulator.

- **Joints**: actuated elements that allow the mobility of the manipulator, and each joint confers a degree of freedom; they can be revolute, for rotational movements, or prismatic, for translations along the axes. Joints can be implemented with different types of actuators (i.e. DC, AC or brushless motors, depending on the application).
- **Links**: rigid elements composing the overall robot body, connected through joints.
- **End-effector**: the utensil at the tip of the structure, connected to the wrist, that is the last joint of the kinematic chain, conferring dexterity.

It is then possible to describe a robot using a *kinematic model*, which represents the relations between the end-effector and the joints configuration, or a *dynamical model*, which details the equations that describe the robot motions in terms of forces acting on its body and actuators. Whether a kinematic or dynamic representation is required usually depends on the specific application considered. In the following sections, both robot's kinematic and dynamical models will be introduced and described.


(a)

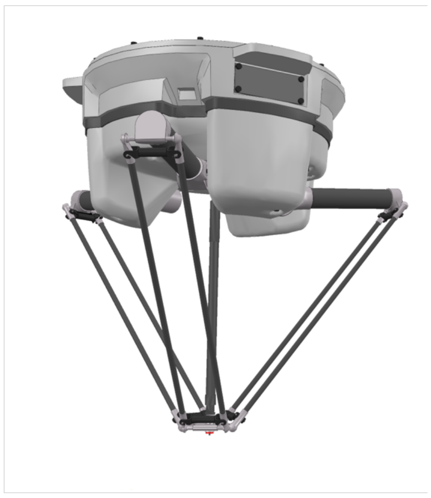

(b)

Figure 2.2: (a) Anthropomorphic robot, with an open kinematic chain (b) Parallel robot, with a closed kinematic chain.

2.2 Kinematic Model

The mechanical structure of a robot manipulator can be represented by a kinematic chain, which describes the connection of each rigid link of the robot with its joints, in relation to the pose of the end-effector: the composition of the motion of each joint and the connected links, thus, results in the overall motion of the whole structure. The number of joints influencing the motion is called the *Degree of Freedom* (DoF) of the robot. The kinematic chain can be either open, if a single sequence of links connects the base and the tip of the robot, or closed, if the links describe a ring (i.e., more kinematic chains connect the base to the end-effector). In Figure 2.2 two robots with an open and closed kinematic chain are represented, respectively. The focus of this work will be on robots featuring an open kinematic chain. To this end, let us introduce, for a generic n-DoF manipulator, the vector of the joint variables

$$q(t) = \begin{bmatrix} q_1(t) \\ q_2(t) \\ \vdots \\ q_n(t) \end{bmatrix}$$

which contains the values of each joint composing the kinematic chain (expressed in radians/degrees or meters, depending on whether the joint is revolute or prismatic), and the end-effector's pose

$$x_e(t) = \begin{bmatrix} p_e(t) \\ \varphi_e(t) \end{bmatrix}$$

where $p_e(t)$ represents the position $O - xyz$ of the end-effector in the cartesian space with respect to a base reference frame, and $\varphi_e(t)$ is the orientation of the end-effector, expressed in Euler angles (or quaternions). Intuitively, the velocities and accelerations of the robots' joints can be expressed as $\dot{q}(t) = \frac{dq}{dt}$ and $\ddot{q}(t) = \frac{d^2q}{dt^2}$. For the sake of convenience, the dependence of variables on time t will be omitted when obvious.

The *operative space* (or *working space*) is the portion of the surrounding space that is accessible to the manipulator (i.e., where $x_e(t)$ exists). The manipulator's physical structure, namely the length of each link and its joints ranges, must be known to define the operative space.

To determine the position and orientation of the end-effector in an operative space $\in \mathbb{R}^3$ we generally need 6 DoF: 3 to move the object in the desired point in space, and 3 to orient it with respect to a reference frame. If the robot has more degrees of freedom than the ones required to accomplish the task, it is called *redundant*.

In the following, different ways to express the robot kinematics are recalled from [26] by B. Siciliano, L. Sciavicco, and L. Oriolo, and briefly discussed.

2.2.1 Direct Kinematics

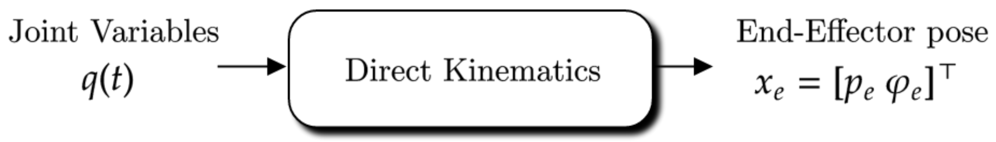


Figure 2.3: Direct Kinematics

The goal of *direct kinematics* (Figure 2.3) in robotics is to determine the position and orientation of the end-effector as a function of the joint variables i.e., it computes a transformation from the robot's joint space to the robot's operative space. Taking

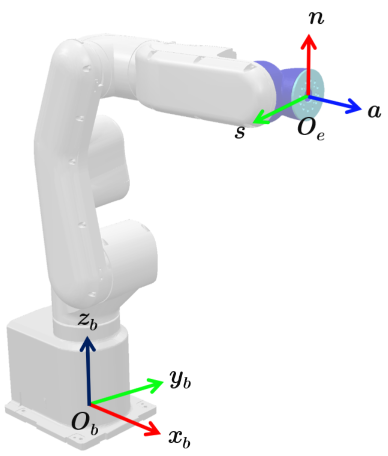


Figure 2.4: Base reference frame and tool reference frame on a 6-DoF robot Epson VT6

as an example the robot illustrated in Figure 2.4, let us define a *base reference frame* $O_b - x_b, y_b, z_b$ and an end-effector frame (or *tool frame*) $O_e - n, s, a$, where O_e is placed on the last virtual joint of the end-effector and the orthogonal axes n, s, $a \in \mathbb{R}^3$ represent the normal, sliding and approach vectors, respectively. Then, the direct kinematics can be expressed as the homogeneous transformation

$$T_e^b(q) = \begin{bmatrix} n(q) & s(q) & a(q) & p_e(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

where q is the vector of the joint variables and p_e the position of the end-effector with respect to the base frame. The matrix $T_e^b(q) \in \mathbb{R}^{4\times4}$ can be obtained by recursively compute the transformation matrix between each element of the kinematic chain, up to the tool frame. Therefore, by considering the individual reference frames O_i, $i = 0...n$ attached to each link of the robot, it is possible to compute the transformations

$$T_n^0(q) = A_1^0(q_1)\, A_2^1(q_2) \ldots A_n^{n-1}(q_n), \tag{2.2}$$

where

$$A_i^{i-1} = \begin{bmatrix} R_i^{i-1}(q_i) & p_i(q_i) \\ 0 & 1 \end{bmatrix} \tag{2.3}$$

contains the rotation matrix $R_i^{i-1} \in \mathbb{R}^{3\times3}$ and the position $p_i \in \mathbb{R}^3$ of a reference O_i with respect to the previous one O_{i-1}. The direct kinematic is then

$$T_e^b(q) = T_0^b\, T_n^0(q)\, T_e^n. \tag{2.4}$$

One way to assign reference frames O_i to each component of the robot in order to compute the transformation matrix is by applying the *Denavit-Hartenberg* (DH) convention [27]. For the sake of simplicity, in the following, let us express the direct kinematic as a non-linear function

$$x_{\mathrm{e}} = k(q). \tag{2.5}$$

2.2.2 Inverse Kinematics

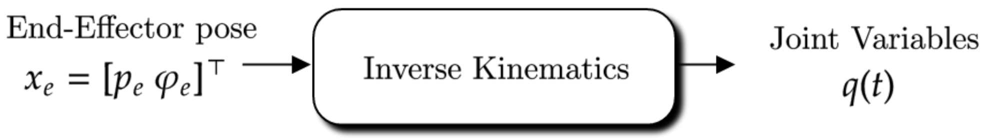


Figure 2.5: Inverse Kinematics

As opposed to the direct kinematics, the goal of inverse kinematics (Figure 2.5) is to define the joint variables q given the pose of the end-effector x_e. It performs a transformation from the operating space to the joint space. The computation of inverse kinematics is a crucial aspect for robotic applications, especially for tasks that require motion planning for reaching and manipulation, in which the desired trajectory is naturally expressed in the cartesian space. While the computation of the direct kinematics is generally easy and feasible, for inverse kinematics a series of complications arise:

1. **Multiple solutions exist:** a point in space can be reached by the end-effector with many (potentially infinite) possible configurations, especially if the manipulator is redundant.

2. **No solution exists:** it is possible that a given position and orientation does not belong to the operative space, and therefore can not be reached by any configuration.

3. **High complexity of computation:** equations can be non-linear and a solution in closed form may not always exists.

.

2.2.3 Differential Kinematics

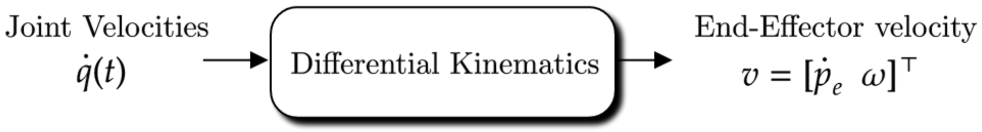


Figure 2.6: Differential Kinematics

Similarly to direct kinematics, *differential kinematics* (Figure 2.6) establishes a relation between the joint velocities $\dot{q}$ and the velocity of the robot's end-effector. To this end, let $v \in \mathbb{R}^3$ be the end-effector's velocity, such that

$$v = \begin{bmatrix} \dot{p}_e \\ \omega \end{bmatrix} \tag{2.6}$$

where $\dot{p}_e \in \mathbb{R}^3$ and $\omega \in \mathbb{R}^3$ denote its translation and rotational velocities, respectively. Note that said quantities are referred to the reference frame $O_e - n, s, a$, previously introduced for the discussion of direct kinematics. The relation between $\dot{q}$ and v can then be expressed as

$$v = \begin{bmatrix} J_p(q) \\ J_o(q) \end{bmatrix} \dot{q} = J(q)\,\dot{q} \tag{2.7}$$

where the matrix $J(q) \in \mathbb{R}^{6\times n}$ is the so-called *geometric Jacobian* of the manipulator. More in detail, $J_p(q) \in \mathbb{R}^{3\times n}$ encapsulates the contribution of the joints velocities to the end-effector's linear velocity, while $J_o(q)$ that of the angular one. Thus, each joint contributes to the end-effector velocity, and the result is the total roto-translational velocity component in the operating space. Specifically, for each joint i, each column of the geometric Jacobian matrix can be computed as

$$J_i = \begin{bmatrix} j_{pi} \\ j_{oi} \end{bmatrix} = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} \tag{2.8}$$

if the joint is prismatic, otherwise

$$J_i = \begin{bmatrix} j_{pi} \\ j_{oi} \end{bmatrix} = \begin{bmatrix} z_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix} \tag{2.9}$$

if the joint is revolute. In the previous equations, z_i refers to the *axis* and p_i to the *position vector* of the i-th joint, respectively. As it can be noted, similarly to the transformation matrix (2.1), the Jacobian strictly depends on the structure of the manipulator, and specifically on the type of its joints. Alternatively, the Jacobian can be computed analytically as the time derivative of the direct kinematics. The so-called *analytical Jacobian* J_A is then computed as

$$\dot{x}_e = \begin{bmatrix} \dot{p}_e \\ \dot{\varphi}_e \end{bmatrix} = \begin{bmatrix} \frac{\partial p_e}{\partial q}\dot{q} \\ \frac{\partial \varphi}{\partial q}\dot{q} \end{bmatrix} \tag{2.10}$$

hence

$$\dot{x}_e = J_A(q)\dot{q}. \tag{2.11}$$

It is worth noticing that not always one has $J_A = J$, as it can be that $\omega \neq \dot{\varphi}$

2.3 Dynamical Model

Let us now introduce the dynamical equations that regulate the behavior of a robotic system in the joint space. As a first definition, the dynamical model of the robot expresses the relation between the forces τ, acting on the joints, and the motion of the overall arm, as in

$$\tau = D(q, \dot{q}, \ddot{q}) \tag{2.12}$$

where $D(q, \dot{q}, \ddot{q})$ generically describes the robot's dynamics. The focus of this Thesis is primarily on motion tasks performed by an industrial manipulator. It is assumed, therefore, that the end-effector does not actively come in contact with the environment. Furthermore, the closed-form of the dynamical model based on the *Lagrangian* formulation is considered. In the following subsections, making reference to the theory in [26] (Chapter 7) by B. Siciliano, L. Sciavicco, and L. Oriolo and the references therein, the main steps to describe the model of a generic industrial manipulator will be reported.

2.3.1 Lagrangian formulation

Let us assume that the robot is composed of rigid links, and let $q \in \mathbb{R}^n$ be the *generalized coordinates* describing the motion of said links in a n-DoF manipulator.

In the case of an open-chain manipulator, those refer to the joint variables. The Lagrangian of the system is generally defined as

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{2.13}$$

where $\mathcal{T}$ and $\mathcal{U}$ are the total kinematic and potential energy of the system, respectively. In the case of an actuated manipulator, the kinematic energy is expressed in a quadratic form as

$$\mathcal{T} = \frac{1}{2}\dot{q}^\top B(q)\dot{q} \tag{2.14}$$

where $B(q) \in \mathbb{R}^{n\times n}$ is the *inertia matrix*, which is dependant on the masses, the lengths, and the inertias of the links in the manipulator, and represents the inertial contribute of each joint to the motion of the manipulator.

The potential energy, instead, is expressed as

$$\mathcal{U} = -\sum_{i=1}^{n}(m_{l_i} g_0^\top p_{l_i} + m_{m_i} g_0^\top p_{m_i}) \tag{2.15}$$

where m_{l_i} and m_{m_i} represent the masses of the i-th link and rotor, respectively, $g_0 \in \mathbb{R}^3$ is the vector of gravitational torques, and p_{l_i} and p_{m_i} are the position of the center of gravity of the i-th link and the rotor.

Let us now introduce the *Lagrange equations*, expressed as

$$\frac{d}{dt}\left(\frac{\mathcal{L}}{\partial \dot{q}}\right)^\top - \left(\frac{\mathcal{L}}{\partial q}\right)^\top = \xi \tag{2.16}$$

where $\xi \in \mathbb{R}^n$ is the vector of *generalized forces* associated with the coordinates q. By substituting Eq. (2.14)-(2.15) into (2.16) and computing the derivatives, one obtains

$$B(q)\ddot{q} + n(q,\dot{q}) = \xi \tag{2.17}$$

$$n(q,\dot{q}) = \dot{B}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^\top B(q)\dot{q}\right)\right)^\top + \left(\frac{\partial \mathcal{U}(q)}{\partial q}\right)^\top \tag{2.18}$$

Omitting further computation, for which the reader is referred to [26], assuming no physical interaction with the environment is performed, one obtains the dynamical model expressed as

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s sign(\dot{q}) + g(q) = \tau \tag{2.19}$$

in which

- $B(q) \in \mathbb{R}^{n\times n}$ is the inertia matrix introduced in (2.14).
- $C(q,\dot{q}) \in \mathbb{R}^{n\times n}$ represents the centripetal and Coriolis torques and depends on the robot's configuration and the angular velocities of the links. Like for the inertia matrix, masses, lengths, and inertias of rigid bodies composing the manipulator should be known or suitably estimated.
- $F_v \in \mathbb{R}^{n\times n}$ and $F_s \in \mathbb{R}^{n\times n}$ represent the viscous and static frictions, respectively.
- $g(q) \in \mathbb{R}^n$ is the vector of gravitational torques. It is especially relevant for the dynamical behavior of the system, as it needs to be compensated if the robot stops in a specific configuration. It depends on mass and length of the links.
- $\tau \in \mathbb{R}^n$ represents the motor torques.

2.4 Motion Planning

Motion planning and control is a crucial aspect of robotic applications. Its goal is to generate suitable reference signals for the system to obtain a predefined motion, required to perform a task. More specifically, it provides the robot with a time sequence of values that result in continuous motion from a starting point to a final one, in a finite time interval. Preliminary concepts on how to generate suitable references to perform robot motion will be presented in the context of robot manipulators, although many of the presented concepts are valid in general. The following subsections attain their content from the theory in [26] (Chapter 4) by B. Siciliano, L. Sciavicco and L. Oriolo and the references therein. To this end, the first important distinction to make is that of trajectory generation in the *joint space* and in the *operative space*. As the name suggests, the first approach deals with the generation of reference signals for each joint of the robot, while the latter allows the definition of a trajectory in the operative (cartesian) space for

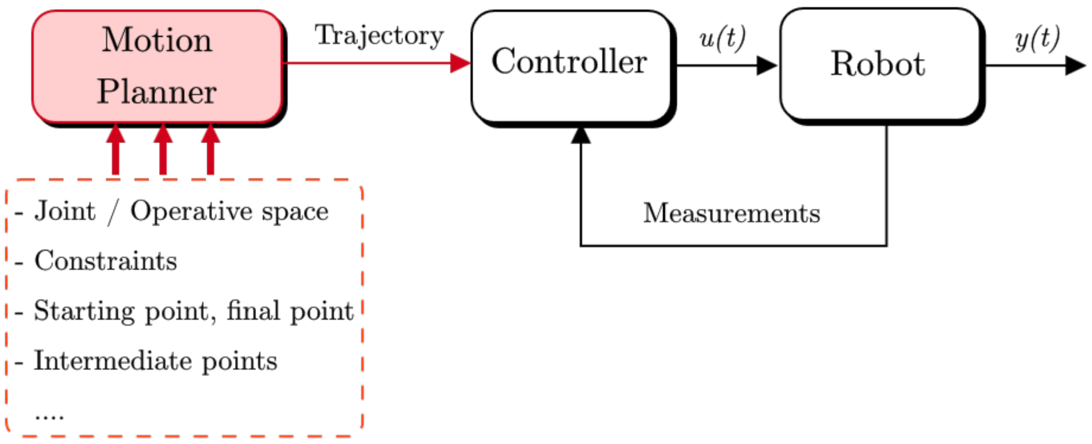


Figure 2.7: Robot control scheme with external motion planner

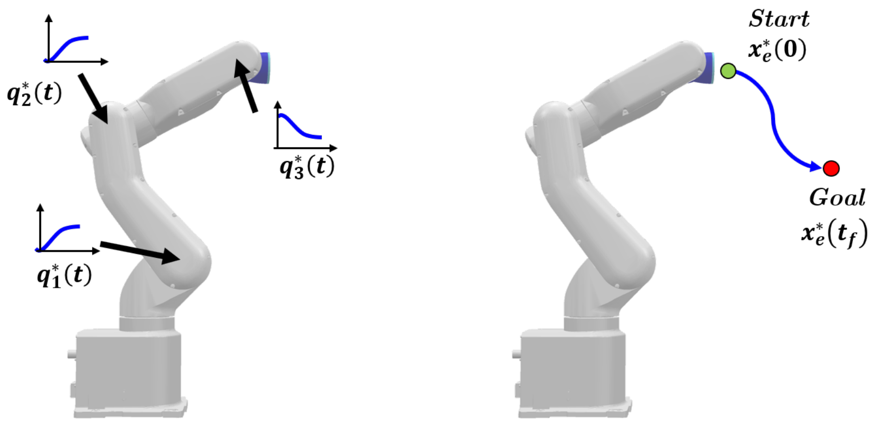


Figure 2.8: Trajectories in the joint space (left) and in the operative space (right)

the robot's end-effector, as exemplified in Figure 2.8. Depending on the problem to solve, one approach might be preferable to the other. In any case, trajectory generation should be of light computational load, ensure continuity of position, velocity, and acceleration profiles as not to damage the robot, and present low oscillatory behavior. Thus, given a set of requirements, which are usually defined a-priori during the design phase, a motion planner computes the desired trajectory and feeds it to the system under control (Figure 2.7).

2.4.1 Motion planning in the joint space

The goal of trajectory planning in the joint space is to generate the desired joint positions $q^*(t)$ to obtain a coordinated motion of the robot, respecting the imposed constraints. As trajectories for each joint can be computed independently, it does not require performing online kinematic inversion, which allows for more accountability of kinematic singularities. Different approaches may be adopted for motion planning, but all fall into two classes:

1. **Point-to-point:** only the initial and the final points are specified.
2. **Interpolation of intermediate points:** throughout the motion, several waypoints are specified between the starting one and the goal.

In the following, some of the most common approaches are presented and briefly discussed to provide a general overview. The work presented in this Thesis will primarily deal with point-to-point trajectories, so a more in-depth discussion will be given on this topic. For the sake of simplicity, and without loss of generality since planning in the joint space allows to manage each joint independently, all trajectories hereafter refer to a single joint.

Polynomial functions

Given an initial position q_i, a final position q_t, an initial time t_i and a time instant t_f such that $q(t_f) = q_f$, one possible way to define the reference sequence for the joint is by using a *polynomial function* of the type

$$q(t) = \sum_{j=0}^{n} a_j (t - t_i)^j \tag{2.20}$$

with n being the degree of the polynomial. If one wishes to ensure the minimum dissipated energy, it can be demonstrated that a quadratic velocity profile (and thus, a cubic position profile) allows a movement of a joint from q_i to q_f in time t_f that also minimizes the cost

$$\int_{t_i}^{t_f} \tau^2(t) dt. \tag{2.21}$$

Therefore, it is possible to choose a trajectory defined by a *cubic polynomial* ($n = 3$) such as

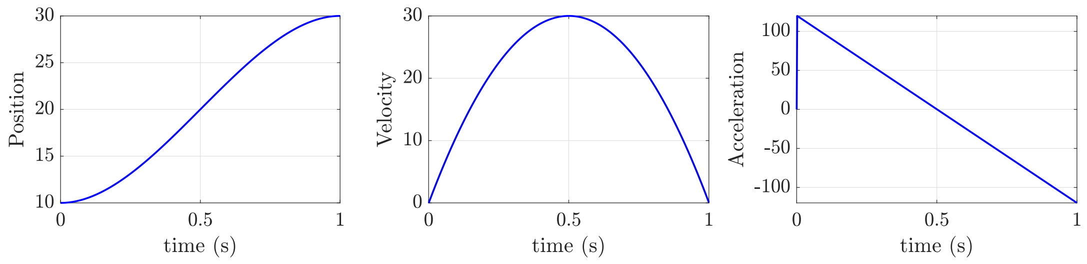


Figure 2.9: Example of a cubic trajectory with conditions $q_i = 0$, $q_f = 30\ deg$, $\dot{q}_i = 0$, $\dot{q}_f = 0\ deg/s$, $t_f = 1$ s

$$q(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 + a_3(t - t_i)^3 \tag{2.22}$$

with boundary conditions on positions and velocities

$$q(t_i) = q_i \quad \dot{q}(t_i) = \dot{q}_i \quad q(t_f) = q_f \quad \dot{q}(t_f) = \dot{q}_f. \tag{2.23}$$

Trajectory profiles for a cubic polynomial are reported in Figure 2.9. As it can be noted, this type of trajectory causes high discontinuities in acceleration at boundaries. This behavior is generally undesired, as it can damage the actuators. Such issue can be solved by imposing boundary conditions on the acceleration as well. This, however, requires to increase the degree and use a *quintic polynomial* (n=5) such as

$$q(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 + a_3(t - t_i)^3 + a_4(t - t_i)^4 + a_5(t - t_i)^5 \tag{2.24}$$

with boundary conditions on positions, velocities and accelerations:

$$q(t_i) = q_i \quad \dot{q}(t_i) = \dot{q}_i \quad q(t_f) = q_f \quad \dot{q}(t_f) = \dot{q}_f \quad \ddot{q}(t_i) = \ddot{q}_i \quad \ddot{q}(t_f) = \ddot{q}_f. \tag{2.25}$$

This results in a smoother acceleration profile, as can be seen in Figure 2.10

Trapezoidal Velocity Profile (TVP)

An alternative way to generate a joint trajectory is to use the so-called *trapezoidal velocity profiles*. The position profile $q(t)$ obtained is constituted by two parabolic arcs, at the beginning and end of the trajectory, connecting a linear segment, during which the joints move at a constant cruise velocity $\dot{q}_c$. Before reaching cruise velocity, the joint moves at constant acceleration $\ddot{q}_c$ and, when approaching the

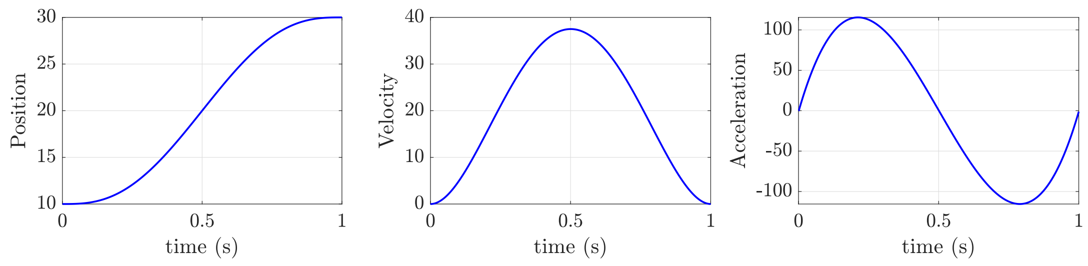


Figure 2.10: Example of a quintic trajectory with conditions $q_i = 0$, $q_f = 30\ deg$, $\dot{q}_i = 0$, $\dot{q}_f = 0\ deg/s$, $\ddot{q}_i = 0$, $\ddot{q}_f = 0\ deg/s^2$, $t_f = 1$ s

goal position q_f, constantly decelerates with equal magnitude. More in detail, the trapezoidal velocity trajectory can be described by the equations

$$q(t) = \begin{cases} q_i + \frac{1}{2}\ddot{q}_c t^2, & 0 \leq t \leq t_c \\ q_i + \ddot{q}_c(t - \frac{t_c}{2}), & t_c < t \leq t_f - t_c \\ q_f - \frac{1}{2}\ddot{q}_c(t_f - t)^2 & t_f - t_c < t \leq t_f \end{cases} \tag{2.26}$$

where t_c is the time instant at which $\dot{q}(t_c) = \dot{q}_c$. Specifically, given initial and final conditions q_i, q_f and t_f, one can impose the cruise velocity $\dot{q}_c$ such that

$$\frac{|q_f - q_i|}{t_f} < |\dot{q}_c| \leq \frac{2|q_f - q_i|}{t_f}. \tag{2.27}$$

Furthermore, it can be determined

$$t_c = \frac{q_i - q_f + \dot{q}_c t_f}{\dot{q}_c} \tag{2.28}$$

and, consequently,

$$\ddot{q}_c = \frac{\dot{q}_c^2}{q_i - q_f + \dot{q}_c t_f}. \tag{2.29}$$

Alternatively, one can directly impose the desired acceleration $\ddot{q}_c$ such that

$$\ddot{q}_c \geq \frac{4|q_f - q_i|}{t_f^2}. \tag{2.30}$$

An example of a TVP trajectory is reported in Figure 2.11.

Polynomial interpolation

In some applications, in order to obtain the desired motion, it might be necessary to define a sequence of intermediate waypoints that the joint need to pass through

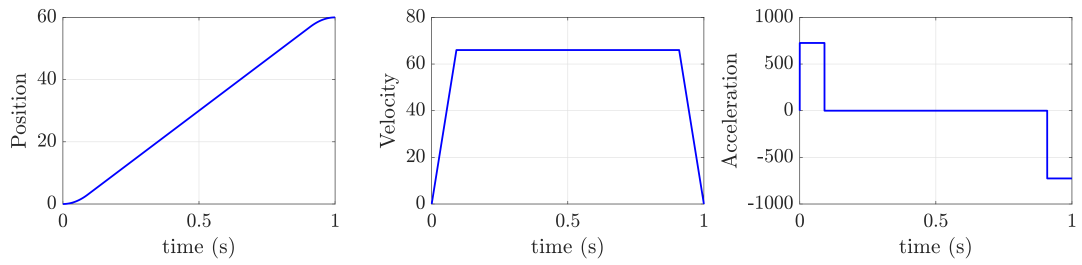


Figure 2.11: Example of a TVP trajectory with conditions $q_i = 0$, $q_f = 60\ deg$, $\dot{q}_c = 66\ deg/s$, $t_f = 1$ s

before reaching the final position q_f. This might be the case in settings where the robot is required to perform especially precise motion or adjust itself to be compliant with the environment. One possibility to achieve this is to perform cubic interpolation between each waypoint q_k. The goal is then to define a suitable sequence of cubic polynomials $\Pi_k(t)$, $k = 1...N$ such that

$$q_1 = q_i, \quad q_N = q_f, \quad t_1 = t_i, \quad t_N = t_f, \quad q(t_k) = q_k. \tag{2.31}$$

Moreover, the sequence needs to present a continuous velocity profile. To this end, intermediate velocities can be imposed arbitrarily or according to specific criteria. Nevertheless, as observed in point-to-point motion, one limitation in the use of cubic polynomials is that it presents acceleration discontinuities at boundaries. One way to solve this problem is by using a *spline* curve, which ensures the continuity of the trajectory in its derivatives by using virtual points used to impose continuity in the acceleration profile artificially.

2.4.2 Motion planning in the operative space

Alternatively to planning in the joint space, motion planning in the operative space allows for the definition of trajectories of the end-effector, instead of joints. Therefore, the goal is now to define a sequence of the desired end-effector configurations x_e^*(t) in order to generate the required motion. This approach is perhaps the most common when defining industrial tasks, such as pick and place, reaching, spot welding, etc., as it provides a more natural definition of the trajectory for humans, who also operate in the same space, and allows for easier accountability of environmental constraints (such as obstacles or prohibited areas). Nevertheless, once the trajectory is defined, it is necessary to perform real-time kinematic inver-

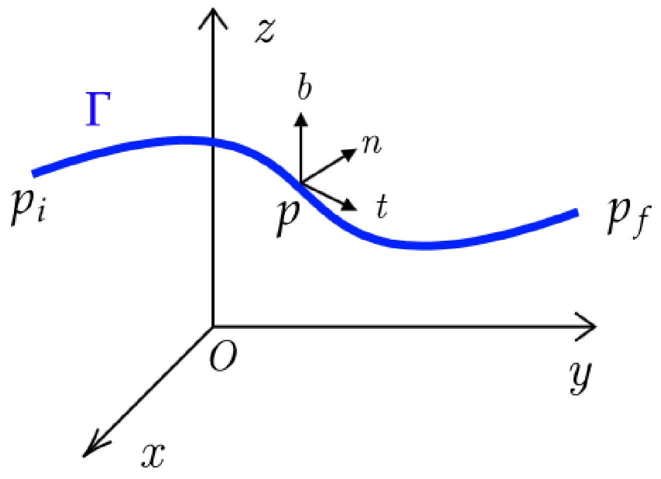


Figure 2.12: Parametric curve in the cartesian space

sion in order to convert each point to a suitable joint reference: this, besides being computationally expensive, may lead to the occurrence of kinematic singularities, which cause unexpected behavior from the manipulator. Great attention must then be posed to the definition of the trajectory, especially in those configurations that might cause singularities. In the following, the main concepts on planning for both position and orientation trajectories of the end-effector are introduced and briefly discussed.

Path primitives

Before discussing planning methods, let us recall the concepts of *parametrization* of a curve, in order to define the path primitives that describe the desired motion. Let us consider a curve Γ in the cartesian space, as represented in Figure 2.12. Then, let us consider an initial point $p_i \in \Gamma$ and a generic point $p \in \Gamma$. The *arc length* s of a point p represents the length of Γ between p_i and p. Therefore, one can write a parametrization of the curve Γ through s as

$$p = f(s). \tag{2.32}$$

Specifically, to each point p correspond three unit vectors, which are related to the representation of Γ as a function of s, defined as

$$t = \frac{dp}{ds}, \quad n = \frac{d^2p}{\left\|\frac{d^2p}{ds^2}\right\| ds^2}, \quad b = t \times n \tag{2.33}$$

where t is the tangent, n is the normal and b is the binormal vector.

Position planning

We wish to generate a reference trajectory p_e^*(t) for the position of the end-effector (i.e., regardless of its orientation). Specifically, the trajectory is defined by a starting position p_i^*, a goal position p_f^* and a finishing time t_f, so that it can be written as

$$p_e^* = f(s(t)) \tag{2.34}$$

where $s(t)$ is a specific timing law, which can be chosen, for instance, as one of those introduced for the discussion of trajectories in the joint space in Subsection 2.4.1, such as polynomials or TVP. Intuitively, one has that $s(0) = 0$ in point p_i^* and $s(t_f) = s_f$ in point p_f^*.

Considering a linear trajectory (i.e., a segment connecting the initial and final point), given a timing law s (time dependence omitted for clarity), the motion can then be defined as

$$p_e^* = \frac{s}{\|p_f^* - p_i^*\|}(p_f^* - p_i^*) + p_i^* \tag{2.35}$$

$$\dot{p}_e^* = \frac{\dot{s}}{\|p_f^* - p_i^*\|}(p_f^* - p_i^*) \tag{2.36}$$

$$\ddot{p}_e^* = \frac{\ddot{s}}{\|p_f^* - p_i^*\|}(p_f^* - p_i^*) \tag{2.37}$$

Orientation planning

Similar consideration as those made for position planning can be made for orientation planning, i.e., the sequence φ_e^* that moves the robot's end-effector from an initial configuration φ_i^* to a goal configuration φ_f^*. In order to avoid violating the orthonormality condition of the three unit vectors n_e, s_e, a_e that describe the orientation of the end-effector with respect to a base frame, one can define the robot's orientation through Euler angles

$$\varphi_e = [\phi, \vartheta, \psi]. \tag{2.38}$$

Then, one can proceed by designing a linear trajectory connecting φ_i^* to φ_f^*, with a suitable timing law, resulting in a trajectory defined as

$$\varphi_e^* = \frac{s}{\left\|\varphi_f^* - \varphi_i^*\right\|}(\varphi_f^* - \varphi_i^*) + \varphi_i^* \tag{2.39}$$

$$\dot{\varphi}_e^* = \frac{\dot{s}}{\left\|\varphi_f^* - \varphi_i^*\right\|}(\varphi_f^* - \varphi_i^*) \tag{2.40}$$

$$\ddot{\varphi}_e^* = \frac{\ddot{s}}{\left\|\varphi_f^* - \varphi_i^*\right\|}(\varphi_f^* - \varphi_i^*) \tag{2.41}$$

where the vectors n_e, s_e, a_e can then be computed using the ZYZ angles.

Chapter 3

Preliminaries on Deep Reinforcement Learning

In this Chapter, the basic concepts behind Machine Learning and Reinforcement Learning will be recalled. Specifically, more focus will be dedicated to the topic of Deep Reinforcement Learning and the available algorithms adapted to the domain of control problems, in order to highlight the aspects relevant to the content of this Thesis.

3.1 General overview on Machine Learning

In computer science, Machine learning (ML), as introduced by Arthur Samuel in 1959 [28], is the field that gives computers the ability to learn and accomplish a task without being explicitly programmed to do so, by the construction of algorithms that can learn from and make predictions on data. Thanks to increased computational power and availability of massive quantities of data, machine learning today finds application in many different domains, such as computer vision, natural language processing, identification, marketing analysis, or robotics. Its popularity is mostly due to its appealing property of being able to encode solutions for problems that are difficult to program explicitly. Furthermore, the same models can be adapted to different domains to solve other problems, with minimal hand-engineering. Nevertheless, one of the main drawbacks of machine learning is that it mostly works as a black box, meaning that given its probabilistic nature, it is hard to interpret its results, which may lead to unexpected outcomes. Furthermore, the quality of a ML model greatly depends on the quality of the data used for training, which may not always be available: one of the most critical features of machine learning is indeed that of *generalization*, i.e., the ability to perform with acceptable results even on data that were never explored during training, which is only possible if a proper data selection was made a-priori. Therefore, despite its promising potential, great care must be taken before applying machine learning to more critical domains.

Generally, the goal of a ML model $\mathcal{M}$ is to represent the relation between some input data x and an output y, by means of a parametric function

$$\mathcal{M}_\theta(x) = y. \tag{3.1}$$

The process of training is then the computation of the parameters θ of $\mathcal{M}$ in order to steer the output of the ML model as close as possible to the desired outcome. Usually, this happens through a *loss function*, used to quantify such divergence. Machine learning can then be divided into three main categories, summarized in Figure 3.1. Specifically:

- **Supervised learning** is obtained by training the model using labeled data, meaning that at each learning iteration, the output of the model is compared against the actual expected output, using some predefined metric; parameters are then updated and the procedure repeated until satisfactory results are obtained, and the model can correctly predict results for new data. The main applications of supervised learning are classification and data regression for model identification.

- **Unsupervised learning**, as the name suggests, is obtained by training the model with unlabeled data. Its main goal is to find common structures and repeated patterns in the input data, meaning it finds application in clustering, density estimation, or anomaly detection.

- **Reinforcement learning** is a type of machine learning in which, differently from the previous two, usually no dataset is built a-priori, meaning that the model autonomously collects its data for training by interacting with the environment and observing the outcome of its action.

In the following sections, a more in-depth discussion on the basic concepts of reinforcement learning will be provided before moving onto the topic of Deep

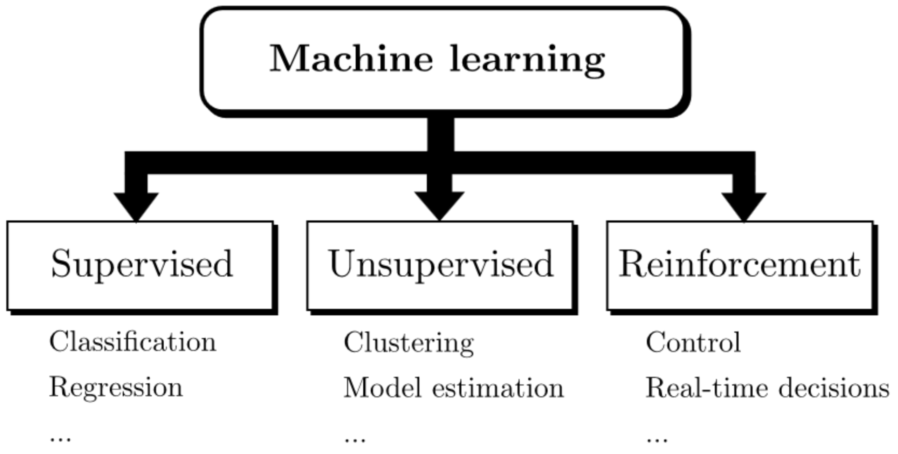


Figure 3.1: Classification of machine learning types

Reinforcement Learning (DRL), which is of interest for the work presented in this Thesis.

3.2 Reinforcement Learning

As anticipated in the previous section, Reinforcement Learning (RL) [29], is a branch of machine learning that, inspired by behavioral psychology, does not need a supervisor or a pre-built dataset in order to autonomously discover an optimal outcome for a specific task. The training process can be summarized by *"learning by doing"*, and the results are achieved through iterative trial and error while interacting with an environment. Specifically, the RL framework, summarized in Figure 3.2, relies on the concept that an *agent*, at any given time t, observes the environment, represented by a certain *state* $x_t \in \mathcal{X}$, with $\mathcal{X}$ being the state space, and according to a certain *policy* $\pi(u|x)$ performs a certain action $u_t \in \mathcal{U}$, with $\mathcal{U}$ being the action space, thus changing the environment's state. When entering a new state, the agent receives a *reward* r_t, that is a scalar indicator on 'how well' the agent has performed.

The framework is abstract and flexible: the *action space* and the *state space* are entirely arbitrary to the designer's discretion and can take a variety of forms with different degrees of complexity. The *policy* π, which serves as a mapping from states to action, can be either deterministic or probabilistic. For a given control task, the learning process is divided into episodes, where the agent interacts with

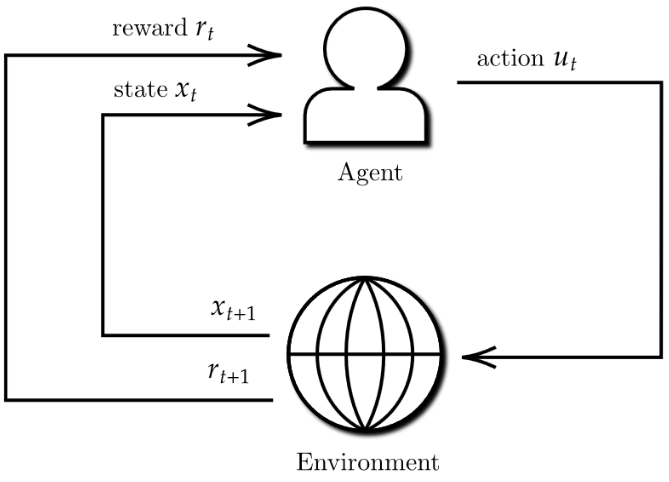


Figure 3.2: Reinforcement learning framework

the environment for either a complete attempt to perform a goal task or a fixed number of time-steps before being reset. The whole training process includes a typically large number of such episodes, up to a predefined maximum.

In the following, the main elements involved in the process of reinforcement learning, based on the theory introduced in [29] by R. Sutton and A. Barto and the references therein, will be recalled and briefly discussed.

3.2.1 Reward

A *reward* r_t is a scalar feedback signal that indicates "how well" an agent has done at step t. The agent's goal is to maximize the (expected) cumulative reward it receives in the long run. In case of episodic tasks with finite horizon T, the expected cumulative reward R_t is defined as

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{3.2}$$

where the term $0 \leq \gamma \leq 1$ is the discount rate, used to prioritize earlier rewards over later ones: if γ is close to 0, the agent will preferably choose actions that maximize immediate rewards, while a value of γ close to 1 will more likely result in the selection of a sequence of actions that will lead to long term maximization, despite possible immediate drawbacks. The reward function is perhaps the most crucial design element when setting a reinforcement learning framework, as it is the only form of 'supervision' in the training process: therefore, it must be designed with great care in order to properly represent the desired behavior that the agent must accomplish.

3.2.2 Markov Decision Process

Given any state x and action u, the *transition probability* (or model) of the process is the distribution of each possible successive state x' and reward r, i.e.,

$$p(x', r|x, u) = P(x', r|x, u). \tag{3.3}$$

Given any time t, a process is said a Markov process if and only if it satisfies the *Markov property*

$$P(x_{t+1}, r_{t+1}|x_t, u_t) = P(x_{t+1}, r_{t+1}|x_t, u_t, x_{t-1}, u_{t-1}, ..., x_0, u_0) \tag{3.4}$$

meaning that the environment's state at time $t + 1$ depends only on the current state and action ("memorylessness" property). In RL, a task in which the transition probability of the environment satisfies the Markov property (3.4) is called a *Markov Decision Process* (MDP)

$$< \mathcal{X}, \mathcal{U}, r, p, \gamma > \tag{3.5}$$

Therefore, given the current state x, action u and the next state x', the expected value of the reward is defined as:

$$r(x, u, x') = E\left[r_{i+1}|x_t, u_t, x_{t+1}\right]. \tag{3.6}$$

3.2.3 Policy

The *policy* $\pi(u|x)$ defines the agent's behavior in the environment, and is the probability that the agent will perform an action u while in state x. Depending on the task, the policy can be represented as a simple function or look-up table, or as a more complex deterministic or probabilistic function that requires extensive computation.

3.2.4 Value function

A *value function* V^π is an estimate of the value of moving into a certain state following an action provided by the policy π. Specifically, the value of a state is the expected cumulative reward the agent can gain over time, starting from that state and following the policy π thereafter, i.e.,

$$V^{\pi}(x) = E_{\pi}\left[R_t | x_t = x\right] = E_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | x_t = x\right]. \tag{3.7}$$

$V^{\pi}(x)$ is the state-value function for the policy π. Furthermore, the value of a given state can be expressed in relation to the subsequent states by the so-called *Bellman Equation*

$$\begin{aligned} V^{\pi}(x) &= E_{\pi}\left[R_t | x_t = x\right] \\ &= E_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | x_t = x\right] \\ &= \sum_{u} \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | x_{t+1} = x'\right]\right] \\ &= \sum_{u} \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^{\pi}(x')\right] \end{aligned} \tag{3.8}$$

in which $P^u_{xx'}$ and $R^u_{xx'}$ are (3.3) and (3.6), respectively. In a similar manner to the value function (3.7), one can define the *action-value function* Q^{π} (also known as Q-function) as

$$Q^{\pi}(x,u) = E_{\pi}\left[R_t | x_t = x, u_t = u\right] = E_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | x_t = x, u_t = u\right] \tag{3.9}$$

representing the expected cumulative reward of taking action u while in state x, and following π thereon.

Since the goal of reinforcement learning is to find an *optimal policy* that can maximize the cumulative reward, let us define π^* such that

$$\pi^*(x) = \operatorname*{argmax}_{\pi} V^{\pi} \qquad \forall x \in \mathcal{X}. \tag{3.10}$$

Therefore, one can define the *optimal value function*

$$V^*(x) = \max_{\pi} V^{\pi}(x). \tag{3.11}$$

and, recalling the expression of the action-value function, one can express the Bellman equation for V^* as

$$\begin{aligned}V^{\pi}(x) &= \max_{u\in\mathfrak{U}} Q^{\pi^*}(x,u)\\ &= \max_u E_{\pi^*}\left[r_{t+1}+\gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|x_t=x,u_t=u\right]\\ &= \max_u E\left[r_{t+1}+\gamma V^*(x_{t+1})|x_t=x,u_t=u\right]\\ &= \max_u \sum_{x'} P^u_{xx'}\left[R^u_{xx'}+\gamma V^*(x')\right].\end{aligned} \tag{3.12}$$

Intuitively, the Bellman optimality equation for Q^* is obtained by substituting $V^*(x')$ with $Q^*(x',u')$. This equation expresses that, under an optimal policy, the value of the state is equal to the expected reward obtained after performing the best action from thereon. Furthermore, since (3.12) is a system of N equations in N unknowns, given an environment with known dynamics, one can solve the value of V^* for each state, and then determine the optimal policy π^* to solve a task. However, the model of the environment is not always available upfront.

3.2.5 Q-learning

When dealing with complex systems, such as those found in robotic applications, the complete transition probability of the environment may not be available, requiring the use of *model-free* learning approaches. One way to achieve this is through Q-learning, introduced by C. Watkins [30], which enables the direct approximation of the optimal action-value function Q^* independently of the policy being applied. Specifically, the algorithm, summarized in Algorithm 1, performs a step-wise update of the approximator $\hat{Q}$, by computing a temporal difference between two consecutive values and updating the value of $\hat{Q}$ according to a *learning rate* $\alpha \in [0,1]$. Furthermore, as demonstrated by Watkins in [30], the following result holds:

Theorem 1. Under the assumption of performing each action infinitely often, and visiting each state infinitely often, with $\alpha \to 0$, then

$$\hat{Q}(x,u) \to Q^*(x,u) \tag{3.13}$$

with probability 1. □

Simply put, it means that given infinite steps, it is guaranteed the convergence of the approximator $\hat{Q}$ to the optimal action-value function Q^*.

Algorithm 1 Q-learning algorithm

```
1: Randomly initialize $\hat{Q}(x_t, u_t)$
2: **for** each episode do:
3:   **for** $t = 0$ to $T$ do:
4:   select with probability $(1 - \epsilon)$ $u = \text{argmax}_u \hat{Q}(x_t, u_t)$, otherwise select $u$ randomly
5:     step $x_{t+1} \leftarrow \textit{Environment}\,(x_t, u_t)$, collect $r$
6:     $\hat{Q}(x_t, u_t) \leftarrow \hat{Q}(x_t, u_t) + \alpha \left[r + \gamma \; \max_u \hat{Q}(x_{t+1}, u_{t+1}) - \hat{Q}(x_t, u_t)\right]$
7:     $x_t \leftarrow x$
8:     **end for**
9: **end for**;
```

Making reference to Line 4 of Algorithm 1, let us introduce the concepts of *exploration* and *exploitation* in reinforcement learning.

- **Exploration:** during training, the agent tends to transition to other states regardless of policy optimization. Exploration is usually achieved by either randomizing actions entirely or by adding noise to the selected action u. This enables the agent to explore states that would otherwise risk never getting evaluated, thus preventing finding a possible better policy.

- **Exploitation:** on the contrary, the agent uses already acquired data in order to make decisions. Although this may lead to finding a policy faster, it usually results in suboptimal outcomes, as the agent does not know of other possibly better solutions.

A policy that only uses exploitation is called *greedy.* Typically, one design element in a reinforcement learning framework is the balance between exploration and exploitation, in order to ensure fast convergence and fair coverage of all actions, states, and rewards of the environment.

Although Q-learning seemingly facilitates the learning procedure for the agent as it bypasses the knowledge of the model, it may become infeasible when dealing with large state and action spaces, especially if these are *continuous.* As this is often the case in robotic applications, alternative approaches for representing policies and value functions need to be considered.

3.3 Deep Reinforcement Learning

In case of problems with an overwhelmingly large number (possibly infinite) of possible states, one way to overcome the issue of mapping the relationships between states and actions is to use Deep Neural Networks (DNNs) to build a parametric approximator of the Q-function. The most notable example of DNNs being used to solve reinforcement learning problems is the work of Mnih et al. [11], where superhuman performances were achieved when training an agent to play Atari videogames. DNNs, represented for instance in Figure 3.3 are powerful parametric functions capable of modeling complex nonlinear relationships between inputs and outputs by means of *neurons*, to which certain weights w are assigned, connected in k layers; the term *deep* refers to the level of composition of the parameters and the use of multiple *hidden* layers between the input and output ones. Let us assume to have a generic target function

$$y = f(x), \quad x \in \mathbb{R}^n.$$

Then, a parametric representation of the function with a DNN is expressed as

$$\hat{y} = W \cdot g(W^{(1)} \cdot g(W^{(2)} \cdot g(...g(W^{(k)} \cdot x + b^{(k)})...) + b^{(2)}) + b^{(1)}) + b \tag{3.14}$$

where $W^{(k)}$ is the set of weights associated with neurons of the k-th layer, g is an *activation function*, and $b^{(k)}$ is the set of biases associated with the output of the k-th layer. Generally speaking, the aim of training a DNN is that of updating the set of parameters θ (i.e., the weights and bias) so that the output of the network matches the desired one.

Making now reference to Q-learning, let us consider a parametrized Q-function $\hat{Q}(x, u|\theta^Q)$. Then, let us impose a stochastic behavior policy β such that $u_t = \beta(x_t)$, with a state visitation frequency ρ^β (i.e., the number of times a state is visited using stochastic policy β). Under these assumptions, the goal is now to minimize a *loss function*

$$L(\theta^Q) = E\left[\left(\hat{Q}(x_t, u_t|\theta^Q) - y_t\right)^2\right] \tag{3.15}$$

where the term y_t is the target function such that

$$y_t = r(x_t, u_t) + \gamma\hat{Q}(x_{t+1}, \hat{\mu}(x_{t+1})|\theta^Q) \tag{3.16}$$

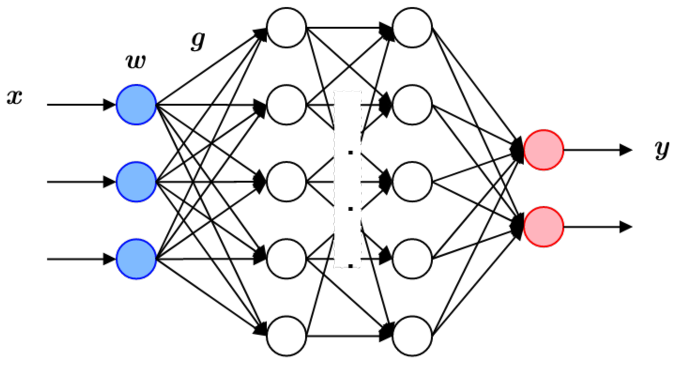


Figure 3.3: Graphic rendering of a Deep Neural Network

and

$$\hat{\mu}(x_t) = \underset{u_t}{\operatorname{argmax}}\, \hat{Q}(x_t, u_t|\theta^Q). \tag{3.17}$$

Several algorithms have been proposed with promising results for solving (3.17), such as Deep Deterministic Policy Gradients [16], T3D [17], or Soft Actor-Critic [31]. Nevertheless, when dealing with a very large number of continuous states and actions, it is especially useful to be able to rephrase the optimization problem so that the computation of the policy can be performed more efficiently. This can be achieved by the Normalized Advantage Function (NAF) algorithm, introduced in [18]; the algorithm, which will be introduced and briefly discussed in the following subsection, is the method of choice for training agents in all applications of DRL methods presented in this Thesis. The reason for this is its superior performance with respect to the state of the art method (DDPG) and its computational efficiency, which allowed training to be also performed on unoptimized hardware.

3.3.1 Normalized Advantage Function

Making reference to [18], let us consider the Q-learning problem, and let us consider a parametrized Q-function $\hat{Q}\left(x_t, u_t|\theta^Q\right)$ such that

$$\hat{Q}\left(x, u|\theta^Q\right) = \hat{A}\left(x, u|\theta^A\right) + \hat{V}\left(x|\theta^V\right) \tag{3.18}$$

where $\hat{A}$ and $\hat{V}$ are DNN approximators of the so-called *advantage function* $A^\pi(x_t, u_t) = Q^\pi(x_t, u_t) - V^\pi(x_t)$ and the value function $V^\pi(x_t)$, respectively. Specifically, the advantage term is expressed as the quadratic function

$$\hat{A}\left(x,u|\theta^{A}\right) = -\frac{1}{2}(u-\hat{\mu}(x|\theta^{\mu})^{\top}P(x|\theta^{P})(u-\hat{\mu}(x|\theta^{\mu}). \tag{3.19}$$

The function $P(x|\theta^P)$ is a positive-definite square matrix such that

$$P(x|\theta^P) = L(x|\theta^P)L(x|\theta^P)^{\top} \tag{3.20}$$

with $L(x)$ lower triangular and with entries derived from the outer layer of the dedicated neural network. The policy (3.17) can then be computed as

$$\begin{aligned}\frac{\partial}{\partial u}\hat{Q}\left(x,u|\theta^{Q}\right) &= \frac{\partial}{\partial u}\hat{A}\left(x,u|\theta^{A}\right) + \frac{\partial}{\partial u}\hat{V}\left(x,u|\theta^{V}\right)\\ &= -(u-\hat{\mu}(x|\theta^{\mu}))^{\top}P(x|\theta^{P}).\end{aligned} \tag{3.21}$$

By imposing equality to 0, one gets

$$\frac{\partial}{\partial u}\hat{Q}\left(x,u|\theta^{Q}\right) = 0 \quad \Rightarrow u = \hat{\mu}(x|\theta^{\mu}). \tag{3.22}$$

Therefore, the action maximizing the Q-function is always $u = \hat{\mu}(x|\theta^{\mu})$. The NAF algorithm is summarized in Algorithm 2.

3.3.2 Hyperparameters

In machine learning there are predefined parameters, called *hyperparameters*, whose values are not learned during the training process but rather decided upfront during the design phase. In practice, these values are determined through a preliminary search activity aiming to identify the most effective combination. When training an agent with NAF, the following hyperparameters have to be set before initiating the process.

Exploration

- **Noise type $\mathcal{D}$:** the type of stochastic process that fosters exploration by adding noise, at each time step, to the action determined by the agent, in order to keep exploring the environment.
- **Noise scale:** scaling factor for noise values.
- **Noise decay factor:** it rules the speed at which the absolute noise value decays during the training activity.

Algorithm 2 NAF algorithm for continuous Q-learning

```
Randomly initialize $\hat{Q}(x, u|\theta^Q) \quad \theta^Q = (\theta^\mu, \theta^P, \theta^V)$
Initialize the target network with $\theta^Q_{\text{TAR}} \leftarrow \theta^Q$
Initialize replay buffer $RB \leftarrow 0$
for each episode do:
    Initialize random process $\mathcal{D}$ for action exploration
    $x_0 \leftarrow Environment\,(reset)$
    for $t = 0$ to $T$ do:
        $u_t \leftarrow \mu\,(x_t|\theta^\mu) + \mathcal{D}_t$
        $r_t \leftarrow r(x_t, u_t)$
        $x_{t+1} \leftarrow Environment\,(x_t, u_t)$
        $RB \leftarrow RB \cup \{(x_t, u_t, r_t, x_{t+1})\}$ store transition in $RB$
        Sample at random and normalize the mini batch $MB$
        for each sample $i = (x_i, u_i, r_i, x_{i+1})$ in $MB$
            $y_i = r_i + \gamma \hat{V}\left(x_{i+1}|\theta^V_{TAR}\right)$
            Compute gradients
                $\frac{\partial}{\partial \theta^Q}\left(y_i - Q\left(x_i, u_i|\theta^Q\right)\right)^2$ (Loss function $L(\theta^Q)$)
            $\theta^Q \leftarrow \theta^Q - \eta\left(\overline{\frac{\partial}{\partial \theta^Q} L(\theta^Q)}\right)$
            $\theta^Q_{\text{TAR}} \leftarrow \tau\theta^Q + (1 + \tau)\theta^Q_{\text{TAR}}$
        end for
    end for
end for
```

Learning

- **Update factor τ:** soft target update at the end of each step.
- **Discount factor γ:** how much the future rewards are valued with respect to the present one.
- **Learning rate η:** learning rate for stochastic gradient descent.

Part II

Self-configuring control schemes for robot motion

Chapter 4

Motivation and state of the art

Motion planning and control are among the classical problems encountered when designing robotics systems [32, 26]. As anticipated in Chapter 2.4, motion planning consists of defining the necessary reference trajectories in terms of positions, velocities, and accelerations in order to define the desired motion of the robot. Once a suitable reference is defined for the motion task we need to solve, it is necessary to design and implement a control structure that is capable of generating the required control inputs (i.e., joint torques or accelerations) to feed the actuator in order to ensure that the motion of the robot is executed in the desired manner. Several aspects must be taken into account during the design phase when controlling a robot to perform motion, such as the robot's mechanical structure, hardware constraints, and the effect of unmodeled dynamics that may affect the way a system behaves under different conditions. In the case of industrial robotic manipulators, two approaches can be adopted depending on the type of operation required, the desired performances, and the type of actuators employed. Specifically, a decentralized control structure or a centralized one can be used. Although the main difference between the two methodologies is, respectively, whether the robotic system is regarded as a composition of linear and decoupled Single-Input-Single-Output (SISO) systems (one for each actuator), or as Multi-Input-Multi-Output (MIMO), other aspects influence the choice of the control structure to be applied to the robotic system. The decentralized approach is generally employed in applications where low performances in terms of velocity and acceleration are required, and the joints present high transmission ratios (i.e., the systems are naturally decoupled); in this case, non-linearities and coupling effects are regarded as disturbances acting on each system [33]. On the other hand, the centralized approach is used when the manipulator's joints have low transmission rates and there is a higher demand in terms of velocity and acceleration performances; such control structure usually relies on the inverse dynamics control approach, that allows to eliminate the nonlinear dynamics acting on the system, which in this case are non-negligible and must be taken into account during the design of the controllers: in fact, dynamic inversion allows to perform a global feedback linearization, resulting in a decoupling of the considered system [26].

Although the decentralized approach presents several advantages due to its simple structure and light computational weight in contrast with the centralized approach, which instead requires the extensive online computation of the inverse dynamics, it is not always suitable when high precision, high-velocity performances during the operation are needed. On the other hand, it is not always possible to have an accurate enough system model to perform the inverse dynamics exactly, resulting in unmodeled disturbances further affecting the robot's performance. Nevertheless, the system might present a non-fixed structure, such as when the robot present Variable Gear-ratios Actuators (VGA) to achieve a broader range of speed, impedance, and forces [34, 35, 36].

Because of the above reason, an a-priori choice of the control structure may result in degraded performances if the operating conditions of the industrial manipulator change during the execution of the task. The choice of the controllers employed in the control structure also plays a critical role in the resulting performances. Sliding Mode Control (SMC) [37, 38, 39] has been used in robotics from the early 1980's [40, 41, 42, 43], although with severe limitation due to the presence of chattering effects, incompatible with electro-mechanical systems. With the introduction of Higher Order Sliding Mode (HOSM) controllers in the last decades, that allow one to confine the effects of discontinuity of the control law to higher order derivatives of the sliding variable, the problem of chattering alleviation could be solved without loss of robustness [44, 45, 46, 47], thus making the application of Sliding Mode Control feasible for robotic systems. For example, [48, 49] present the design of Suboptimal Second Order Sliding Mode (SSOSM) control algorithm, while in [50] a third order switched SMC scheme is proposed. Chattering alleviation can also be successfully achieved with Integral Sliding Mode (ISM) [51]. Besides chattering alleviation, ISM enforces robustness to the controlled system starting from the initial time instant and provides the interesting feature of "perturbation estimation", which allows one to estimate the effects of coupling and unmodeled dynamics on the controlled system. For all these reasons, ISM is the control approach adopted in our application.

The main contribution illustrated in this Part of the thesis is the design of a self-configuring switched structure control architecture, implementing both the centralized and the decentralized approaches, so that the system can autonomously select the most suitable control methodology at any given time, based on an arbitrary metric. The proposed approach is introduced and discussed in detail, presenting two possible methodologies adopted for the switching rule: one based on fixed thresholds [52, 53] and one based on a policy trained with DRL for decision making [54].

4.1 Structure

This Part of the thesis is structured as follows.

- **Chapter 5** introduces the design procedure of the novel switched-structure control scheme, which is presented in detail along with the discussion of two case studies.

- **Chapter 6** presents and discusses an alternative approach based on Deep Reinforcement Learning for the proposed switched-structure control scheme: elements of the framework are detailed and preliminary results are illustrated for the considered case study.

- **Chapter 7** gathers some concluding remarks.

Chapter 5

Switched Structure Control Scheme for Robot Manipulators

In this Chapter, the switched structure scheme for motion control of industrial robot manipulators is introduced and discussed. To overcome the issues deriving from choosing a-priori a specific control scheme, which can result in limited performances when the operating condition of the system varies, the proposed architecture implements both a decentralized approach, suited for lower performance requirements and high transmission ratios, and the inverse dynamics based centralized approach, suited for higher performances in terms of velocity and acceleration. In both cases, the Integral Sliding Mode (ISM) [51] algorithm is used to control the manipulator and compensate matched disturbances. Furthermore, the *perturbation estimation* capability of the ISM controller is used to retrieve an estimate of unmodelled dynamics. This is used to design a switching rule which, based on the perturbations estimated by the ISM controllers, allows to switch from a decentralized control architecture, which typically requires high control gains, to a centralized one with reduced control gains and beneficial effects in terms of chattering and actuator saturation. The implementation of the switching rule thus gives the overall system a self-configuring capability. Such approach can then be applied to systems presenting VGAs or requiring a wider range of trajectory variations.

5.1 Basics of Sliding Mode Control

Sliding Mode Control (SMC) is a nonlinear control method, part of the so-called Variable Structure Control Systems (VSCS) theory, vastly explored in [39, 43, 37]. Let us consider a nonlinear *affine* system (i.e., nonlinear with respect to the state and linear with respect to the control variable) having state-space equation as

$$\dot{x}(t) = f(x(t)) + g(x(t))u(t) \tag{5.1}$$

where $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$ are the state and input, respectively. The main objective of SMC is to lead the state trajectories onto a so-called *sliding surface* $s(x(t)) = 0 \in \mathbb{R}^{n-m}$, where the system enters *sliding mode* in a finite time, by designing a discontinuous control input. When the sliding mode is enforced, one can obtain an *equivalent system.* While in sliding mode, the equivalent system features reduced order (from n to $n - m$) and robustness to matched disturbances (i.e., disturbance acting on the control variable). The dynamics of the equivalent systems are defined by the design of (i) the sliding manifold and, consequently, (ii) the control law.

Example: double integrator

Let us assume a SISO system, written as a perturbed double integrator as follows

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = u(t) + \eta(t) \\ y(t) = s(x(t)) \end{cases} \tag{5.2}$$

where $x \in \Omega \subset \mathbb{R}^2$ is the state vector with $x(t_0) = x_0$, $u(t) \in \mathbb{R}$ is the input and $\eta(t) \in \mathcal{H}$ is a bounded matched uncertainty, with $\mathcal{H}$ being a compact set containing the origin, and $\mathcal{H}^{\text{sup}}$ known. The output function $s(x) : \Omega \to \mathbb{R}$ is of class $\mathrm{C}(\Omega)$ and represents the so-called *sliding variable*, that has to be steered to zero in finite time. Furthermore, $s(x)$ has to be selected such that if $u(t)$ is designed in a way that guarantees $s(x(t_{\mathrm{r}})) = 0\ \forall\, x_0 \in \Omega$ and $s(x(t)) = 0\ \forall\, t > t_{\mathrm{r}}$, where t_{r} is the ideal reaching time, then $\forall\, t \geq t_{\mathrm{r}}$ the origin is an asymptotically stable equilibrium point of (5.2), constrained to $s(x(t)) = 0$. From here on, dependency on time is omitted when obvious, for the sake of simplicity.

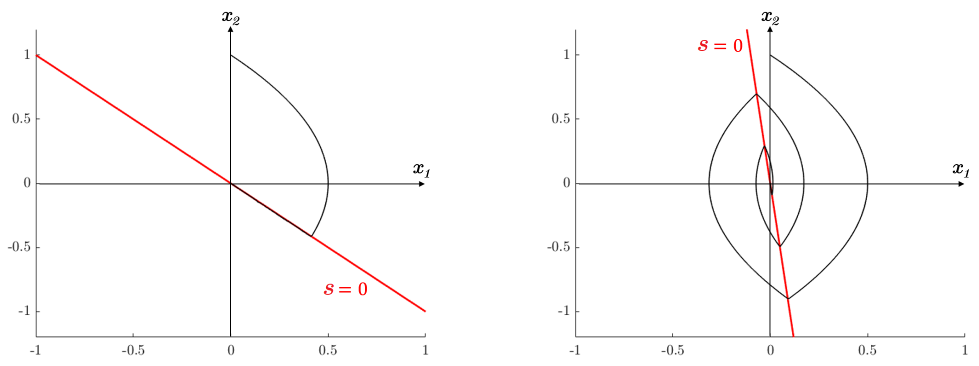


Figure 5.1: State trajectories for $c_1 = 1$ (a) and $c_1 = 10$ (b)

A typical choice for the sliding variable is that of a linear combination of the states. Therefore, let us select the sliding variable $s = c_1x_1 + x_2$, with $c_1 > 0$ so that the sliding manifold is

$$s = c_1x_1 + \dot{x_1} = 0. \tag{5.3}$$

Then, let us consider the discontinuous control law as

$$u = \begin{cases} -1, & \text{if } s > 0 \\ 1, & \text{if } s < 0 \end{cases} \tag{5.4}$$

The behavior of the controlled system, for different choice of c_1, is as depicted in Figure 5.1.

When Sliding Mode is enforced, the equivalent system obtained is

$$x_1(t) = x_1(t_r)e^{-c_1(t-t_r)}. \tag{5.5}$$

As in this case, a typical choice for the control law is a *relay form*, i.e.

$$u(t) = -K \cdot sign(s(x(t), t)), \tag{5.6}$$

where $K > 0$ is a design parameter that, like the sliding variable, has to be properly selected. Despite its desirable properties, due to non-idealities in the switch when sliding mode is enforced, SMC presents the so-called *chattering effect* [55]: in practical applications, the state trajectory does not lie precisely on the sliding manifold, but in the so-called boundary layer, which depends on the actual (finite) switching frequency. In electromechanical systems, this may lead to hardware degradation or undesirable effects. One approach to overcome this issue is the use of Integral Sliding Mode (ISM) control, which besides chattering alleviation, presents other desirable features, such as matched uncertainties rejection, perturbation estimation and the enforcement of sliding mode from the initial time instant. More details are provided in the context of the proposed approach, in Section 5.4.

5.2 Problem formulation

Let us consider a n-joints robot. As detailed in Chapter 2.3, the equations representing the dynamics of the manipulator can be described as the following MIMO nonlinear coupled model

$$B(q)\ddot{q} + n(q,\dot{q}) = \tau \tag{5.7}$$

$$n(q,\dot{q}) = C(q,\dot{q})\dot{q} + F_{\mathrm{v}}\dot{q} + g(q) \tag{5.8}$$

where $B(q) \in \mathbb{R}^{n\times n}$ is the inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{n\times n}$ represents centripetal and Coriolis torques, $F_{\mathrm{v}} \in \mathbb{R}^{n\times n}$ is the viscous friction matrix, $g(q) \in \mathbb{R}^n$ is the vector of gravitational torques, and $\tau \in \mathbb{R}^n$ represents the motor torques. In the following, the time dependency of the joint variables $q(t)$ and $\dot{q}(t)$ is omitted for the sake of simplicity.

Given the robot manipulator model in (5.7)-(5.8), let us assume that q^* and $\dot{q}^*$ $\in \mathbb{R}^n$ are pre-specified reference signals for the joint variables and their first time derivative, respectively. It is assumed that the components of q^* are bounded and $\dot{q}^*$ is Lipschitz continuous. The tracking errors are then defined as

$$e(t) = q^* - q, \quad \dot{e}(t) = \dot{q}^* - \dot{q}\ , \tag{5.9}$$

where $e_1 = e$ and $e_2 = \dot{e}$ represent the position error and the velocity error, respectively.

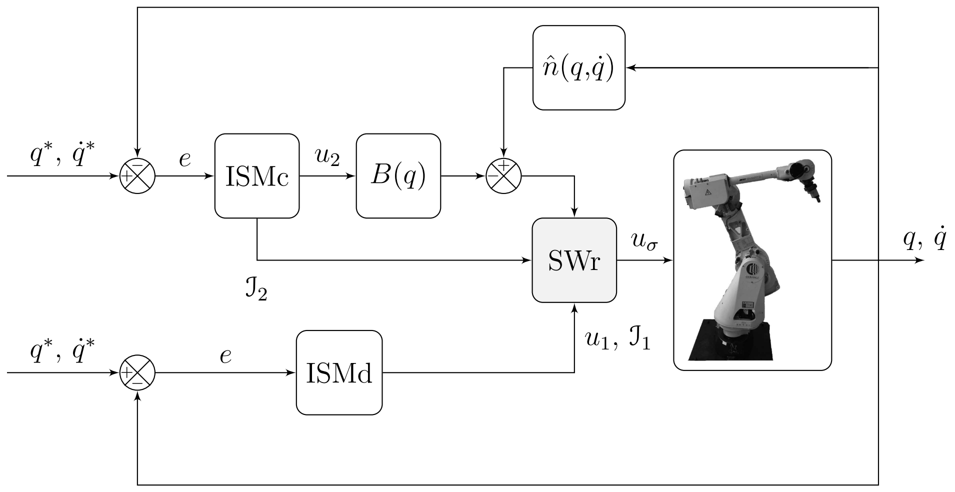


Figure 5.2: The proposed switched structure control scheme

5.3 Self-configuring switched structure scheme

The control structure used in this work is illustrated in Figure 5.2. The first loop implements the decentralized control approach described in Subsection 5.3.1; the second loop implements instead the inverse dynamics based centralized control structure, illustrated in Subsection 5.3.2. For each structure, an ISM controller, described in detail in Section 5.4, is designed to regulate the system and perform a perturbation estimation, thus producing the signals used for the switching criteria.

Remark. Since the uncertainties include centrifugal and Coriolis effects, they could be unbounded due to the quadratic dependence on the joint velocities. In order to guarantee the stability results detailed hereafter, limited velocities are considered, such that the assumptions made in the following subsections on the perturbations acting on the system hold.

5.3.1 Decentralized Control Scheme

As previously anticipated, when using a decentralized approach the robot manipulator is regarded as the composition of n linear and decoupled SISO systems. For the considered case, let us assume that a motor acts on each joint of the manipulator. For the sake of simplicity, let us define the dynamics of the joint motors as the effects related to the spinning of the motor around its own axis. Taking into account a single joint, let $J_{\mathrm{m}j}$ be the motor inertia, $D_{\mathrm{m}j}$ the coefficient of viscous friction of the motor, and let $\tau_{\mathrm{lm}j}$ be the load torque at the axis of the j-th motor, such that

$$\tau_{\mathrm{lm}j} = \frac{\tau_j}{k_{\mathrm{r}j}} \tag{5.10}$$

with $k_{\mathrm{r}j}$ being the reduction ratio defined as

$$k_{\mathrm{r}j} = \frac{q_{\mathrm{m}j}}{q_j} = \frac{\dot{q}_{\mathrm{m}j}}{\dot{q}_j} \ . \tag{5.11}$$

The equation of the jth motor is

$$J_{\mathrm{m}j}\ddot{q}_{\mathrm{m}j} + D_{\mathrm{m}j}\dot{q}_{\mathrm{m}j} = \tau_{\mathrm{m}j} - \tau_{\mathrm{lm}j} \ , \tag{5.12}$$

where $\tau_{\mathrm{m}j}$ is the torque exerted by motor j. Substituting (5.7)-(5.8) to τ_{lm} in (5.12) for each joint, considering $D_{\mathrm{m}j}$, and posing

$$J_{\mathrm{m}} = \begin{bmatrix} J_{\mathrm{m}1} & 0 & \cdots & 0 \\ 0 & J_{\mathrm{m}2} & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & J_{\mathrm{m}n} \end{bmatrix}, \ K_{\mathrm{r}} = \begin{bmatrix} k_{\mathrm{r}1} & 0 & \cdots & 0 \\ 0 & k_{\mathrm{r}2} & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & k_{\mathrm{r}n} \end{bmatrix} \tag{5.13}$$

one obtains

$$J_{\mathrm{m}}\ddot{q}_{\mathrm{m}} = \tau_{\mathrm{m}} - K_{\mathrm{r}}^{-1}\left(B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_{\mathrm{v}}\dot{q} + g(q)\right) \tag{5.14}$$

Equivalently, one can write

$$\left(J_{\mathrm{m}} + K_{\mathrm{r}}^{-1}B(q)K_{\mathrm{r}}^{-1}\right)\ddot{q}_{\mathrm{m}} = \tau_{\mathrm{m}} - K_{\mathrm{r}}^{-1}C(q,\dot{q})K_{\mathrm{r}}^{-1}\dot{q}_{\mathrm{m}} - K_{\mathrm{r}}^{-1}F_{\mathrm{v}}K_{\mathrm{r}}^{-1}\dot{q}_{\mathrm{m}} - K_{\mathrm{r}}^{-1}g(q) \tag{5.15}$$

The above equations can be interpreted as those of a linear and completely decoupled system, subjected to a disturbance deriving from the nonlinear and coupled terms of the dynamic model, i.e.,

$$M(q)\ddot{q}_{\mathrm{m}} = \tau_{\mathrm{m}} - d \ , \tag{5.16}$$

where

$$M(q) = J_{\mathrm{m}} + K_{\mathrm{r}}^{-1}B(q)K_{\mathrm{r}}^{-1}$$

and

$$d = K_{\mathrm{r}}^{-1}C(q,\dot{q})K_{\mathrm{r}}^{-1}\dot{q}_{\mathrm{m}} - K_{\mathrm{r}}^{-1}F_{\mathrm{v}}K_{\mathrm{r}}^{-1}\dot{q}_{\mathrm{m}} - K_{\mathrm{r}}^{-1}g(q)$$

As can be noted, the larger are the reduction rations $k_{\mathrm{r}j}$, the smaller is the impact of the disturbance term d on the system. The control torque τ in (5.7) can then be computed as

$$\tau = K_{\mathrm{r}} M(q) K_{\mathrm{r}} u_1, \tag{5.17}$$

such that system (5.16) becomes

$$\ddot{q} = u_1 - K_{\mathrm{r}}^{-1} M^{-1}(q) d = u_1 - \eta_1 \ . \tag{5.18}$$

where u_1 is the control law to be designed. Letting $x_1 = e_1$ and $x_2 = e_2$, consider now a Proportional-Derivative (PD) controller defined as

$$u_1 = \ddot{q}^* + K_{\mathrm{D1}} x_2 + K_{\mathrm{P1}} x_1 \ , \tag{5.19}$$

with K_{D1} and K_{P1} positive definite diagonal matrices (see [25, Chapter 6] for further details). Combining the previous equations it holds

$$\dot{x}_2 + K_{\mathrm{D1}} x_2 + K_{\mathrm{P1}} x_1 - \eta_1 = 0 \ . \tag{5.20}$$

Finally, the state space representation of the closed-loop system becomes

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -K_{\mathrm{D1}} x_2 - K_{\mathrm{P1}} x_1 + \eta_1 \ . \end{aligned} \tag{5.21}$$

which is characterized by the matched uncertain terms η_1 such that the following assumption holds.

Assumption 1. The uncertainty η_1 is such that

$$\|\eta_1\|_\infty \leq \beta_1 \tag{5.22}$$

with β_1 being a positive constant.

In order to reject η_1, a decentralized ISM (ISMd) control of the type

$$u_1 = \ddot{q}^* + K_{\mathrm{D}_1} x_2 + K_{\mathrm{P}_1} x_1 + u_{11} \ , \tag{5.23}$$

is designed, where u_{11} is typically a discontinuous control action. As a consequence, the equivalent controlled dynamics becomes

$$\dot{x}(t) = \begin{bmatrix} 0 & I \\ -K_{\mathrm{P1}} & -K_{\mathrm{D1}} \end{bmatrix} x(t) \tag{5.24}$$

or, in a compact way,

$$\dot{x}(t) = A_1 x(t),\ x(0) = x_0 \tag{5.25}$$

with A_1 being the closed-loop dynamics matrix. Then, it is possible to conclude that the error is governed by an asymptotically stable second order dynamics that can be arbitrarily assigned, on each joint, by suitably selecting the gains in the diagonal matrices K_{D1} and K_{P1}.

5.3.2 Centralized Control Scheme

In the absence of decoupling effects given by the high reduction ratios (for instance in case of direct drive motors), the use of a centralized control strategy might turn out to be the only viable solution. In the proposed scheme, the centralized approach relies on the so-called inverse dynamics controller. Let us assume again to exactly estimate the inertia matrix $B(q)$ and to have a replica of the vector $n(q,\, \dot{q})$, such that $\hat{n}(q,\, \dot{q}) \neq n(q,\, \dot{q})$. Moreover, let u_2 be an auxiliary control vector such that the control torque is selected as

$$\tau = B(q)u_2 + \hat{n}(q, \dot{q})\ . \tag{5.26}$$

Substituting (5.26) into model (5.7)-(5.8), one has

$$B(q)\ddot{q} + n(q, \dot{q}) = B(q)u_2 + \hat{n}(q, \dot{q})\,, \tag{5.27}$$

which is a chain of n decoupled double integrator systems of the type

$$\ddot{q} = u_2 - \eta_2\ . \tag{5.28}$$

Letting again $x_1 = e_1$ and $x_2 = e_2$, consider a PD controller defined as

$$u_2 = \ddot{q}^* + K_{\mathrm{D2}} x_2 + K_{\mathrm{P2}} x_1\ , \tag{5.29}$$

with K_{D2} and K_{P2} positive definite diagonal matrices, which substituted to the previous equation gives

$$\dot{x}_2 + K_{\mathrm{D2}} x_2 + K_{\mathrm{P2}} x_1 - \eta_2 = 0\ . \tag{5.30}$$

The state space representation of the closed-loop system is

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -K_{\mathrm{D}_2} x_2 - K_{\mathrm{P}_2} x_1 + \eta_2\ . \end{aligned} \tag{5.31}$$

This system is characterized by the matched uncertain terms η_2 such that the following assumption holds.

Assumption 2. The uncertainty η_2 is such that

$$\|\eta_2\|_\infty \leq \beta_2 \, . \tag{5.32}$$

with β_2 being a positive constant.

In order to reject η_2, a centralized ISM (ISMc) control of the type

$$u_2 = K_{\mathrm{D}2} x_2 + K_{\mathrm{P}2} x_1 + u_{12} \tag{5.33}$$

is designed, where u_{12} is a discontinuous control action. As a consequence, the equivalent controlled dynamics becomes

$$\dot{x}(t) = \begin{bmatrix} 0 & I \\ -K_{\mathrm{P}2} & -K_{\mathrm{D}2} \end{bmatrix} x(t) \tag{5.34}$$

or in a compact way

$$\dot{x}(t) = A_2 x(t), \; x(0) = x_0 \tag{5.35}$$

with A_2 being the closed-loop dynamics matrix. Then, one can conclude again that the error is governed by an asymptotically stable second order dynamics that can be arbitrarily assigned, on each joint, by suitably selecting the gains in the diagonal matrices $K_{\mathrm{D}2}$ and $K_{\mathrm{P}2}$.

Note that the choice of the PD controller is not mandatory, in the sense that any other stabilizing control law can be used as high level controller. Nevertheless, the use of PD controllers is quite common in robotics. As for the ISM control component, it is preferred to other sliding mode control solutions for its capability to make the controlled system insensitive to the matched uncertain terms since the initial time instant, thus eliminating the so-called reaching phase [51].

5.3.3 Switching Block

Let us introduce a *switching block* (SWr) embedding a switching rule that enables the changes between the decentralized and the centralized control scheme for the manipulator. Specifically, it relies on a given metric $\mathcal{J}_{\sigma(t)}$, where $\sigma \in \{1, 2\}$ is the so-called switching signal related to the decentralized control and the centralized one, respectively. For the proposed approach, the switching metric $\mathcal{J}_{\sigma(t)}$ is chosen as a function of the matched unknown terms η_1 and η_2, i.e.,

$$\mathcal{J}_{\sigma(t)} = \begin{cases} \mathcal{J}(\eta_1), & \text{if } \sigma(t) = 1 \\ \mathcal{J}(\eta_2), & \text{if } \sigma(t) = 2 \end{cases} . \tag{5.36}$$

The reason for this is due to the fact that the dynamics of the controlled system are affected by uncertain terms, which in turn depend on the velocity and acceleration required to the robot, affecting the performance. The logic of the switching rule then is as follows: if, while using a decentralized approach, the coupling terms are greater than a given threshold P (i.e., high velocity and acceleration performance are required), switch to the centralized control. Vice versa, if the centralized control is active and the uncertain terms are lower than P (i.e., high velocity and acceleration performance are not required), switch to the decentralized control. Therefore, the rule can be written as

$$\mathcal{J}_{\sigma\in\{1,\,2\}} \gtrless P \, . \tag{5.37}$$

Note that P is a suitably selected threshold, the value of which is determined, for instance, through a trial and error procedure, or by the knowledge of the physical quantities involved in the system and provided by the robot data sheet. Specifically, for the proposed approach, the choice for the metric $\mathcal{J}_{\sigma(t)}$ is the Root Mean Square (RMS) value of η_1 and η_2 of the considered joints, since it represents the deviation of the input signal from a given baseline, which corresponds to the case without uncertain terms.

5.4 Integral Sliding Mode Control design

It is now possible to design the ISM controllers to be used in the decentralized and centralized case, respectively. ISM is typically characterized by a control variable u_σ, $\sigma \in \{1,\, 2\}$ depending on the used structure, split into two parts, i.e.,

$$u_\sigma(t) = u_{0\sigma}(t) + u_{1\sigma}(t) \tag{5.38}$$

where $u_{0\sigma}$ is generated by a suitable PD controller designed relying on the nominal system model, while $u_{1\sigma}$ is the sliding mode control action, used in order to reject the uncertainties affecting the system.

The $u_{1\sigma}$ component has to be designed relying on the errors states x_1, x_2 previously defined. Thus, the integral sliding manifold is defined as

$$\Sigma(t) = s(t) + \varphi(t) = 0 \tag{5.39}$$

where Σ is a vector of the auxiliary sliding variables, while $s = Sx$, with S being a matrix of positive coefficients, is the actual sliding variable. Furthermore, the integral term φ is given by

$$\varphi(t) = -s(0) - \int_0^t S[x_2(z),\, u_{0\sigma}(z)]^\top dz \tag{5.40}$$

with the initial condition $\varphi(0) = -s(0)$. Then, the discontinuous control law is defined as

$$u_{1\sigma}(t) = -K_\sigma \operatorname{sgn}\left(\Sigma(t)\right)\ , \tag{5.41}$$

with K_σ being the ISM control gain.

5.4.1 Stability proof

The considered system, controlled by the selected controller, can be described by the following dynamics

$$\dot{x}(t) = A_{\sigma(t)}x(t), \quad x(t_0) = x_0 \tag{5.42}$$

where $\sigma(t)$ is *admissible*, meaning that in finite time only a finite number of switchings can occur, by virtue of the presence of a predefined dwell-time. Necessary condition for stability under the switching signal $\sigma(t)$ is that all matrices A_i, $i = 1, 2$ are Hurwitz [56]. Moreover, the dwell-time T_{dw} is such that the equilibrium point $x = 0$ of the system (5.42) is asymptotically stable with the switching signal $\sigma(t) = i \in \{1, 2\}$, $t \in [t_k, t_{k+1})$ where t_k and t_{k+1} are successive switching times with $t_{k+1} - t_k \geq T_{\mathrm{dw}}$, for all $k \in \mathbb{N}$ and index $i \in \{1, 2\}$.

Let us prove that the ISM component allows to achieve autonomous systems (5.42) where each matrix A_i is Hurwitz in spite of the uncertainties.

Theorem 2. Given the error system (5.21) (or (5.31)) such that Assumption 1 (or 2) hold, controlled via (5.38) with a ISM controller $u_\sigma(t)$ as in (5.41) and sliding variable Σ as in (5.39)-(5.40), if $\|K_1\|_\infty > \beta_1$ (or $\|K_2\|_\infty > \beta_2$), then a sliding mode $\Sigma = 0$ is enforced $\forall\, t \geq 0$.

Proof. For the sake of simplicity, let $\sigma = 1$ (without loss of generality for the case of $\sigma = 2$). Consider system (5.21) expressed as

$$\dot{x}(t) = A_1 x(t) + B(u_{11} + \eta_1), \quad x(t_0) = x_0 \, . \tag{5.43}$$

Let us define the Lyapunov function candidate as

$$V = \frac{1}{2}\Sigma^\top \Sigma \, . \tag{5.44}$$

Then, let us compute its first time derivative

$$\begin{aligned}\Sigma^\top \dot{\Sigma} =& \Sigma^\top S(A_1 x + B(u_{11} + \eta_1) - A_1 x) \\ =& \Sigma^\top SB\,(u_{11} + \eta_1) \\ =& -\Sigma^\top \left(K_1 \mathrm{sgn}\,(\Sigma) - \eta_1\right) \\ <& -\left(\|K_1\|_\infty - \beta_1\right)\|\Sigma\| \end{aligned} \tag{5.45}$$

Hence, $\dot{V}$ is negative definite if and only if $\|K_1\|_\infty > \beta_1$, thus implying that there exists a reaching time $\bar{t}_\mathrm{r} > 0$ such that $\Sigma(t) = 0$, $\forall\, t \geq \bar{t}_\mathrm{r}$. Since, by construction in (5.39)-(5.40), $\Sigma(0) = 0$, then $\Sigma(t) = 0$, $\forall\, t \geq 0$, which concludes the proof. □

5.4.2 Perturbation Estimator and Chattering Alleviation

By virtue of Theorem 2, it is now possible to formulate the result assessing the perturbation estimation capability of the algorithm, which will be used to retrieve the necessary information to enforce the switching rule (5.37). Let us introduce the following:

Theorem 3. Given the error system (5.21) (or (5.31)) such that Assumption 1 (or 2) holds, controlled via control law (5.38) with (5.41) and sliding variable as in (5.39)-(5.40), if $\Sigma = 0$ is enforced $\forall\, t \geq 0$ solving u_{11} (or u_{12}) from $\dot{\Sigma} = 0$, then the equivalent control is such that $u_{11\mathrm{eq}} = -\eta_1$ (or $u_{12\mathrm{eq}} = -\eta_2$). Moreover, the equivalent system is Hurwitz and invariant with respect to the uncertainties.

Proof. For the sake of simplicity, let $\sigma = 1$ (with no loss of generality for $\sigma = 2$) and consider system (5.21) expressed as

$$\dot{x}(t) = A_1 x(t) + B(u_{11} + \eta_1), \quad x(t_0) = x_0 \, . \tag{5.46}$$

By virtue of Theorem 2, with $\|K_1\|_\infty > \beta_1$ an integral sliding mode is enforced since the initial time instant $t = 0$, i.e., $\Sigma(t) = 0$ and $\dot{\Sigma}(t) = 0$, $\forall\, t \geq 0$. The latter implies that

$$\begin{aligned}\dot{\Sigma} =& \dot{s} + \dot{\varphi} \\ =& S(A_1 x + B(u_{11} + \eta_1) - A_1 x) \\ 0 =& u_{11} + \eta_1 \ . \end{aligned} \tag{5.47}$$

By definition of equivalent control [37, 39], solving u_{11} from (5.47) allows one to obtain $u_{11\mathrm{eq}}$, i.e.,

$$u_{11\mathrm{eq}}(t) = -\eta_1(t), \quad \forall\, t \geq 0 \ . \tag{5.48}$$

Finally, by substituting the equivalent control $u_{11\mathrm{eq}}(t)$ in (5.21), one directly achieves system (5.25) which is Hurwitz and invariant with respect to the uncertainties, concluding the proof. □

This way, the ISM controller is able to estimate the uncertain and coupling terms if the equivalent control is available. In actuality, an approximation of the equivalent control can be obtained via a first order linear filter [51] with the real discontinuous control (5.41) as input signal, i.e.,

$$\tilde{u}_{1\sigma\mathrm{eq}}(t) = \frac{1}{\mu}\int_0^t \mathrm{e}^{-\frac{1}{\mu}(t-z)} u_{1\sigma}(z) dz \tag{5.49}$$

where μ is the time constant of the filter. It should be set such that the linear filter does not distort the slow component of the switching action. Furthermore, the integral term in (5.40) has to be redesigned as

$$\varphi(t) = -s(0) - \int_0^t S[x_2(z),\, u_\sigma(z) - u_{1\sigma}(z)]^\top dz \tag{5.50}$$

with initial condition $\varphi(0) = -s(0)$. By virtue of Theorem 3, one has that $\tilde{u}_{1\sigma\mathrm{eq}}(t)$ is a good estimate of η_σ. This quantity is then used to compute the metric $\mathcal{I}(\hat{\eta}_\sigma)$ in (5.37), which allows to realize the switching mechanism illustrated in Subsection 5.3.3. Moreover, due to the presence of the filter, the control law in (5.49) is continuous now a continuous signal provided to the plant, thus ensuring a chattering alleviation feature. Note that, having in mind a robotic application, in the following the equivalent control computed as in (5.49) with the integral term designed as (5.50) will be used.

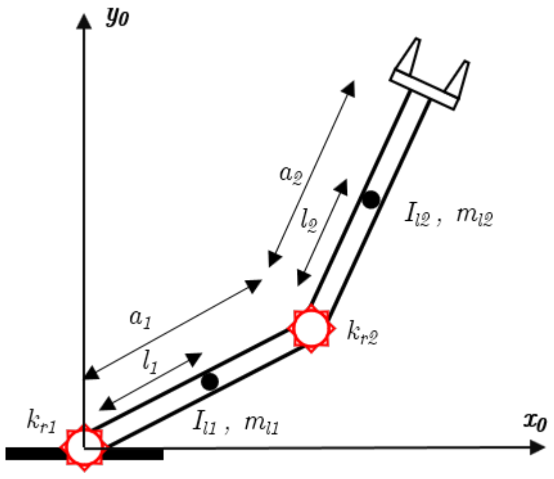


Figure 5.3: Planar manipulator with 2 revolute joints equipped with Variable Gears Actuators.

5.5 Case studies

In the following subsections, two different examples will be illustrated. The first one is an academic example (see [26]) where the case with a robot implementing Variable Gear Actuators (VGA) is presented and the proposed strategy is successfully applied. The second one is carried out on a robot Comau Smart3-S2, the model of which was realized on the basis of real data (Appendix A).

5.5.1 Case study 1: Variable Gear Actuators

Let us consider a 2-axis planar manipulator, represented in Figure 5.3 and featuring parameters summarized in Table 5.1 ([26, Chapter 7]). The example is intended to show the behavior of the strategy in presence of VGA at both joints, which are not identified by the inverse dynamics. Indeed, the feedback linearization takes into account the dynamics of the robot at a given constant nominal value of the motors reduction ratios. Therefore, the addition of VGA at the joints, in this case, aims to show the effects on the unmodeled dynamics caused by the variation of reduction ratios, and how the proposed strategy could handle such a situation.

The switched structure control scheme has been applied to track a reference q^* in the joint space equal to a constant signal with an amplitude of 10 rad. During operation, the reduction ratios at the joints evolve in time between 1 and their nominal value $k_{rn1,2} = 100$ as a sinusoidal function, starting at a value $k_{r1,2} = 50$.

Table 5.1: Planar 2 DoF Robot Parameters

Parameter	Value
a_1 $[m]$	1
a_2 $[m]$	1
l_1 $[m]$	0.5
l_2 $[m]$	0.5
m_{l_1} $[Kg]$	50
m_{l_2} $[Kg]$	50
I_{l_1} $[Kg \cdot m^2]$	10
I_{l_2} $[Kg \cdot m^2]$	10
k_{r_1}	[1; 100]
k_{r_2}	[1; 100]

In this case, the switching rule introduced in Subsection 5.3.3 is defined so that $P = 5$. A dwell time of $T_{\text{dw}} = 2$ s has been imposed, to avoid high frequency switching, and the simulation has a timestep duration $T = 5 \times 10^{-4}$ seconds. Both joints start with an initial position $q_{1,2}(0) = 0$ rad.

In Figure 5.4 the results for the tracking of the reference signals for joint 1 and joint 2 are reported. As can be observed, both joints manage to maintain the position stably, despite the changes in the reduction ratios and uncertainties acting on the system. By virtue of the ISM component, the coupling terms are rejected, making the two joints decoupled and behave consistently.

In Figure 5.5 the profile of the reduction ratio variation, valid for both joints of the manipulator, is reported alongside the metric $\mathcal{I}$ compared to the threshold P and the corresponding switching signal σ. As it can be observed in the evolution of the switching signal (Figure 5.5, bottom image), the control architecture is initialized as the centralized one ($\sigma = 2$) and after $T_{\text{dw}} = 2$ s, since the metric $\mathcal{I}_2$ is below P, it switches to the decentralized structure ($\sigma = 1$). After 5 seconds the index increases up to the threshold and, since the dwell-time has expired, σ switches again to 2 (i.e., centralized). Before the time instant 10 s, the index $\mathcal{I}_2$ is below P and σ becomes 1, but the index $\mathcal{I}_1$ again abruptly increases up to the threshold. Although the best solution would be the centralized one, the switching signal σ remains equal to 1

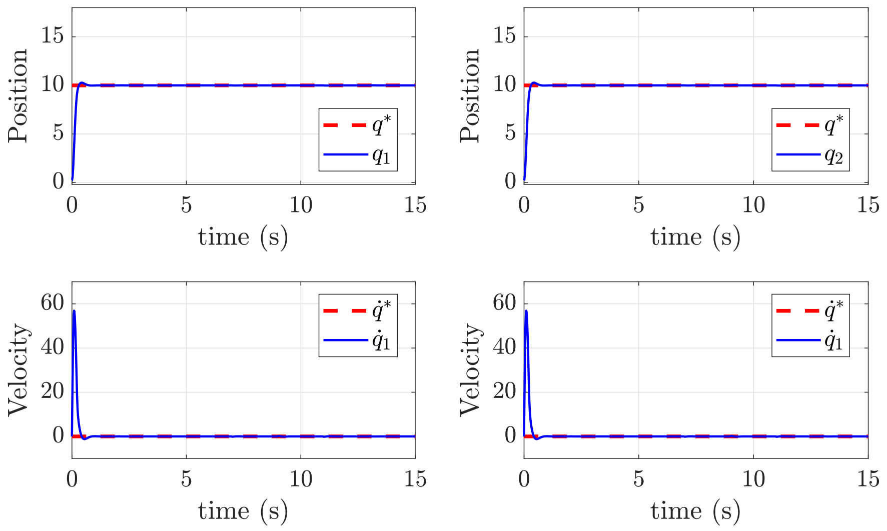


Figure 5.4: Position and velocity with references in the joint space

Table 5.2: Control Parameters for Case Study 1

Parameter	Dec. ($\sigma = 1$)	Cen. ($\sigma = 2$)
$K_{1\sigma}$	100	100
$K_{2\sigma}$	100	100
$K_{\mathrm{P}1\sigma}$	100	100
$K_{\mathrm{P}2\sigma}$	100	100
$K_{\mathrm{D}1\sigma}$	10	10
$K_{\mathrm{D}2\sigma}$	10	10

due to the dwell-time. After 2 seconds σ becomes 2 but the index decreases below P, causing another switch to the decentralized architecture, as expected.

The evolution of the sliding variables for both joints is reported in Figure 5.6. As it can be observed, sliding mode is always enforced, despite switches in the control structure. Due to the presence of a first order filter to generate the equivalent control, an overshoot is present at the beginning in the time interval when the system is still sensitive to the uncertainties and (5.49) has to converge. Control parameters used for the design of the ISM controller are summarised in Table 5.2.

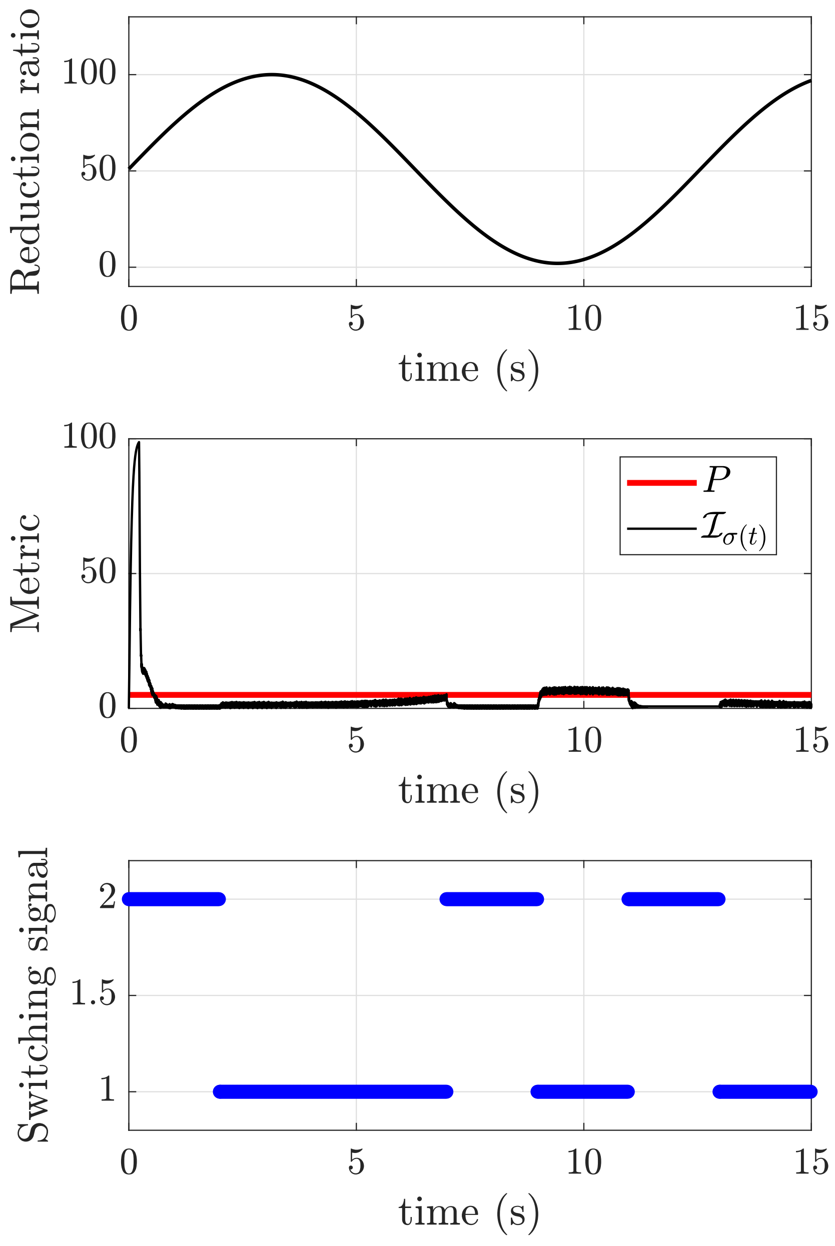


Figure 5.5: Variation of reduction ratios in time (top), for both joints; metric $\mathcal{I}(\hat{\eta}_\sigma)$ used for switching (middle); switching signal σ: 1 for the decentralized approach, 2 for centralized approach (bottom)

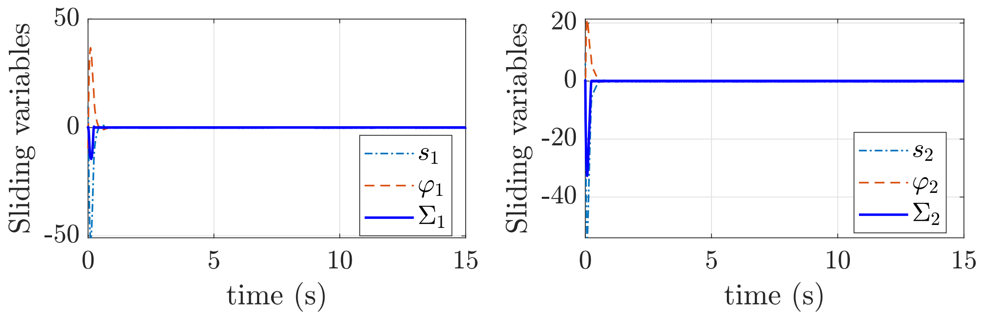


Figure 5.6: Time evolution of the auxiliary sliding variable Σ, the sliding variable s and the integral term φ for joint 1 and joint 2.

5.5.2 Case study 2: Industrial Robot Comau Smart3-S2

In the following, simulation results carried out relying on a realistic model of a robot manipulator Comau Smart3-S2, identified on the basis of real data [57, 58] (see Appendix A), are presented. For the sake of simplicity, only vertical planar motion of the robot has been enabled. The control structure has been applied to a motion control problem that requires the tracking of pick-and-place trajectories in the operative space. The reference signal for each joint are shaped as a sequence of Trapezoidal Velocity Profiles (TVP) trajectories. The simulation timestep has a duration of $T = 5 \times 10^{-5}$ s. It is assumed that the joint velocity $\dot{q}$ is measurable or estimated using a Levant's differentiator [44], as successfully shown in [59]. Two examples with two different trajectories are discussed in detail, before a comprehensive discussion on the results and a comparison with an Active Disturbance Rejection Control (ADRC) scheme.

Note that, since an estimation procedure was used to find the robot parameters [49], and this was based on the maximum likelihood approach applied to a suitable parametrized model, the switching mechanism in Subsection 5.3.3 was suitably adapted. For this reason two different thresholds P_τ and $P_{\ddot{q}}$ were specified, based on estimates of the uncertain terms acting on the torques for the decentralized case and on the acceleration for the centralized one, respectively. This way, when the metric overcomes P_τ, the centralized architecture is activated; when the metric $\mathcal{J}_{\sigma(t)}$ is instead below $P_{\ddot{q}}$, the decentralized architecture is activated. The value of the thresholds, which are related to the uncertain terms, have been suitably selected on the basis of the data collected from the real plant.

Table 5.3: Control Parameters for Case Study 2

Parameter	Dec. ($\sigma = 1$)	Cen. ($\sigma = 2$)
$K_{1\sigma}$	1000	2
$K_{2\sigma}$	500	2
$K_{3\sigma}$	500	2
$K_{\mathrm{P}1\sigma}$	2.5×10^4	200
$K_{\mathrm{P}2\sigma}$	2.5×10^4	200
$K_{\mathrm{P}3\sigma}$	2.5×10^4	200
$K_{\mathrm{D}1\sigma}$	5×10^3	100
$K_{\mathrm{D}2\sigma}$	5×10^3	100
$K_{\mathrm{D}3\sigma}$	5×10^3	100

The Proportional-Derivative parameters used for the nominal control in the regulators and the control gains for the sliding mode components are reported in Table 5.3. The metric $\mathcal{J}_\sigma$ (see (5.37)), it is recalled, has been chosen as the RMS value of the output of the perturbation estimator of the three considered joints, provided by the ISM controller. Thresholds for the switching logic have been set equal to $P_\tau = 150$ and $P_{\ddot{q}} = 0.08$. The dwell-time has been set equal to $T_{\mathrm{dw}} = 2$ s.

Example 1

The joint position reference signals fed to the plant, represented in Figure 5.7, are generated from a TVP with initial position $q_0^* = [0,\ 0,\ \pi/2]^\top$ and final position of $q^* = [\pi/2,\ \pi/2,\ 0]^\top$ up until $t = 13.125$ seconds (i.e., 7/8 of a total simulation time of 15 seconds). The trajectory is then inverted for the remaining portion of time. The robot's initial joint configuration is $q_0 = [\pi/15, 0,\ \pi/2]^\top$ (hence, the transient present at the first joint). In Figure 5.9, the corresponding reference signal p_e^* and the actual trajectory p_e of the end-effector in the operative space are represented. The torques exerted by each joint are also reported in the same picture in order to highlight their smoothness due to the chattering alleviation features provided by the ISM component. Note that, due to the large initial error in the position of the first joint, the computed control torque τ_1 presents large initial values, but quickly adjusts. On the right, the metric $\mathcal{J}_\sigma$ is illustrated. Specifically,

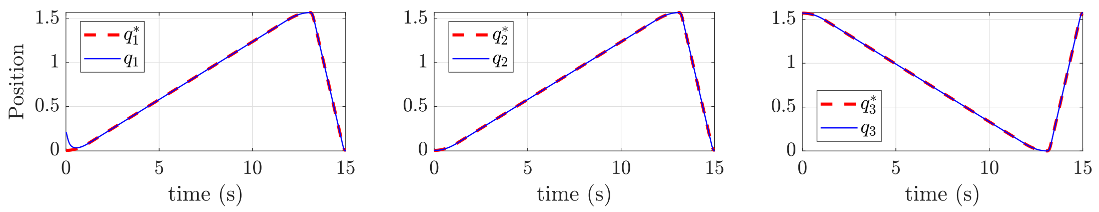


Figure 5.7: Trajectory profiles in the joint space. Joints must reach $\pi/2$, $\pi/2$ and 0 until 7/8 of the simulation duration, and then revert the motion

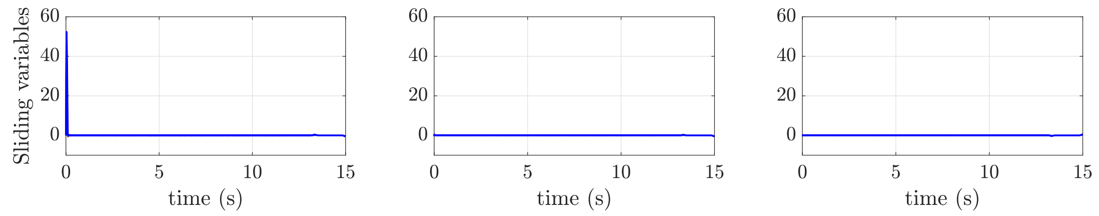


Figure 5.8: Time evolution of the sliding variables Σ_j, $j = 1, 2, 3$

the first part of the picture is a close-up of the metric when the decentralized architecture is used (switching signal $\sigma = 1$, as $\mathcal{J}_1 < 150$ and $\mathcal{J}_2 > 0.08$). As it can be noticed, the system runs on the decentralized loop while following a "slow" trajectory, and changes when a sudden variation occurs at 13.125 s, requiring higher velocity performances that can be guaranteed by the centralized approach (switching signal $\sigma = 2$, $\mathcal{J}_1 > 150$). The evolution of the auxiliary sliding variables Σ_j, $j = 1, 2, 3$ is shown in Figure 5.8, where it can be seen that a sliding mode is always ensured despite changes in the control scheme. Finally, joint velocity and acceleration tracking performances are reported in Figure 5.10.

Results of simulations using all three approaches (decentralized only, centralized only, and the proposed self-configuring switched approach) are reported in Table 5.4. The RMS values of the position error e_{RMS}, of the torque τ_{RMS}, and of the estimated uncertainties $\eta_{1\text{RMS}}$ and $\eta_{2\text{RMS}}$ have been computed for each joint. The RMS values obtained through the switching scheme are comparable with the other ones. The proposed approach is then suitable for motion operations and enables the robot to track a wider range of varying trajectories, which require both high and low performances in terms of velocity and acceleration. Nevertheless, it allows the use of lower control gains when possible, with benefits in terms of saturation or wear of the actuators.

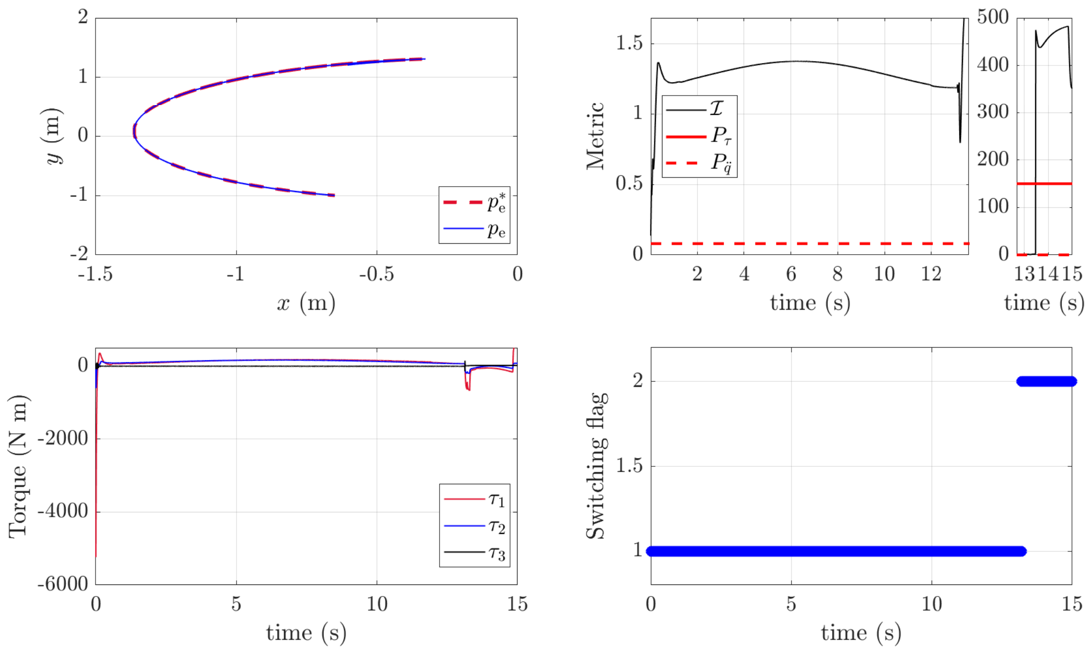


Figure 5.9: Trajectory tracking in the operative space (top left); metric $\mathcal{I}$ and switching thresholds P_τ and $P_{\ddot{q}}$ with a close-up when the switching occur at 13.125 s (top right); control torques τ for each joint (bottom left); switching signal σ: 1 for the decentralized approach, 2 for the centralized approach (bottom right). In this case, switching is required towards the end of the simulation due to a rapid variation of the reference trajectory.

Table 5.4: Result comparison with ISM control for Example 1

	Joint i	e_{RMS}	τ_{RMS}	$\eta_{1\text{RMS}}$	$\eta_{2\text{RMS}}$
Sw.	1	0.0177	206.30	141.65	0.2375
	2	0.0007	122.92	126.95	0.2615
	3	0.0006	9.2403	6.917	0.2346
Dec.	1	0.0176	208.031	146.22	–
	2	0.00036	123.00	120.47	–
	3	0.00003	9.2396	7.988	–
Cen.	1	0.0270	171.76	–	0.407
	2	0.00085	122.48	–	0.336
	3	0.00085	8.167	–	0.33

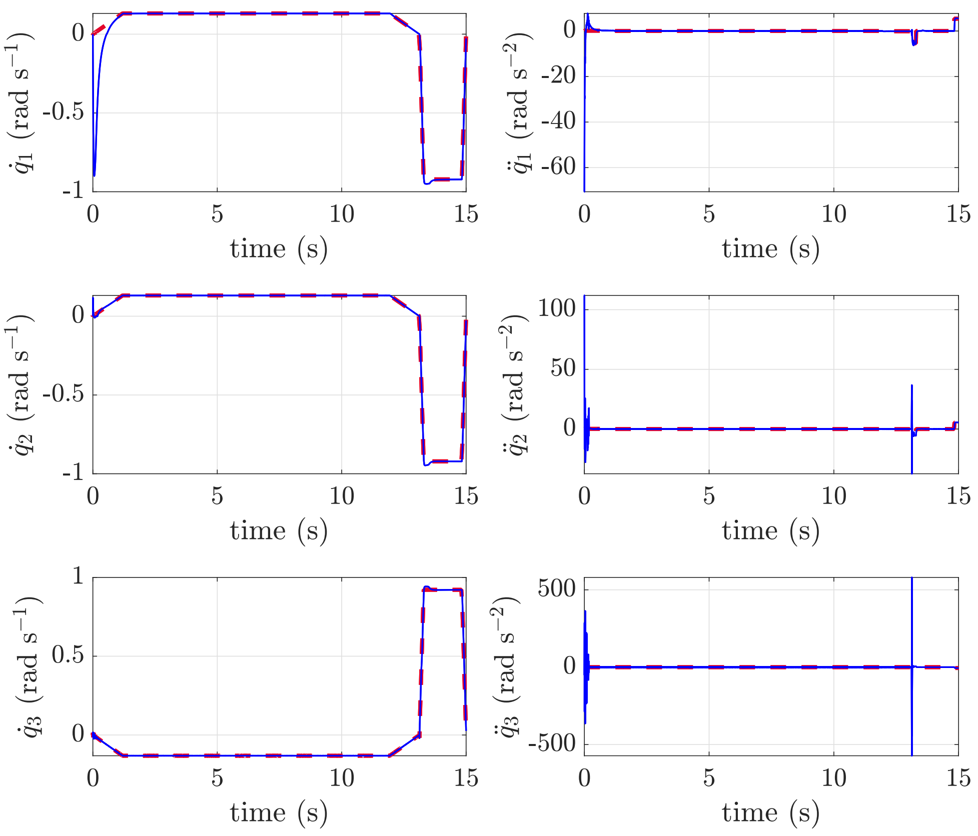


Figure 5.10: Profiles of velocity $\dot{q}$ and acceleration $\ddot{q}$ references (red) and measured signals (blue)

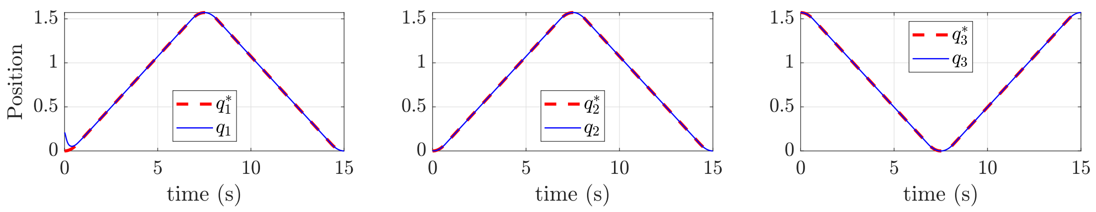


Figure 5.11: Trajectory profiles in the joint space for Example 2. Joints must reach $\pi/2$, $\pi/2$ and 0 until 1/2 of the simulation duration, and then revert the motion.

Example 2

Another example is given by the trajectory reported in Figure 5.11, where joint position reference signals fed to the plant are so that the joints must reach a final position of $q^* = [\pi/2, \pi/2, 0]^\top$ up until $t = 7.5$ seconds (i.e., 1/2 of a total simulation time of 15 seconds), before reverting motion back to their initial position in the last portion of time. In this case, as can be seen in Figure 5.12, the use of the centralized approach can be avoided entirely due to the low demanding nature of the trajectory given as reference which, contrary to the one reported in Example 1, does not require abrupt inversion of motion in a short time interval. The control is then maintained on the decentralized approach, as desirable. Again, auxiliary sliding variables Σ_j, $j = 1, 2, 3$ are shown in Figure 5.13.

5.5.3 Results

In order to obtain a validation of the proposed approach, ten different trajectories with different velocity profiles have been generated, and the benefits of the proposal have been measured according to the RMS value of the uncertain terms suitably estimated by ISM control. Table 5.5 reports the results achieved comparing the two approaches chosen 'a-priori' with the switched methodology. Indeed, the latter results more robust with respect to the decentralized approach and the centralized one, as shown by the tests showing an improvement in terms of disturbance reduction equal to 14.873% and 40.612%, respectively. A graphical rendering of the results is also reported in Figure 5.14.

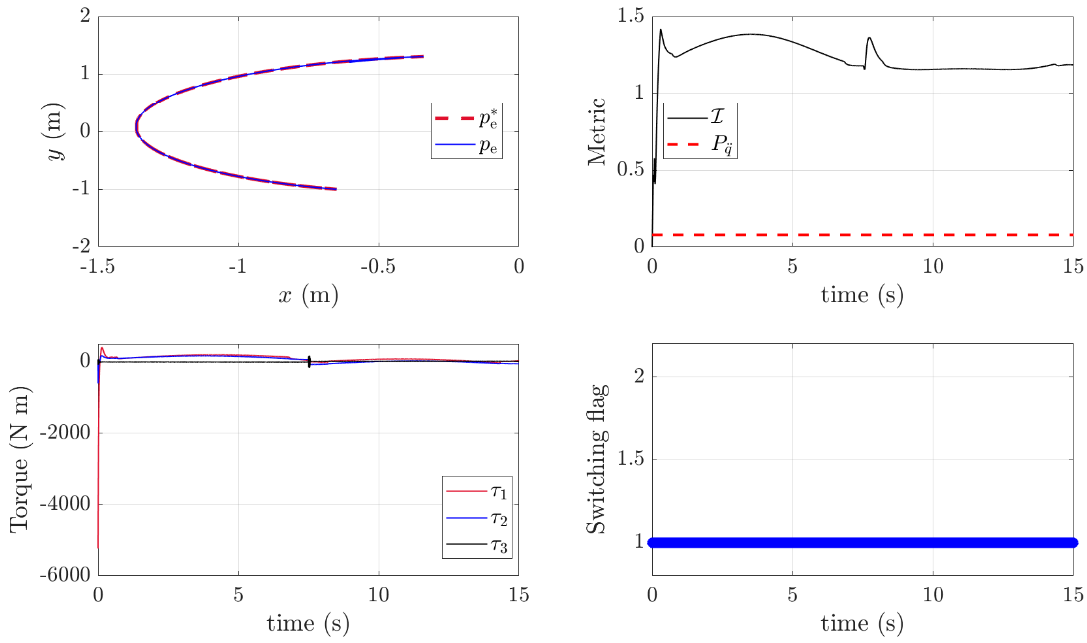


Figure 5.12: Trajectory tracking in the operative space (top left); metric $\mathcal{I}$ and switching threshold $P_{\ddot{q}}$ (top right); control torques τ for each joint (bottom left); switching signal σ: 1 for the decentralized approach, 2 for the centralized approach (bottom right). In this case, no switching is required.

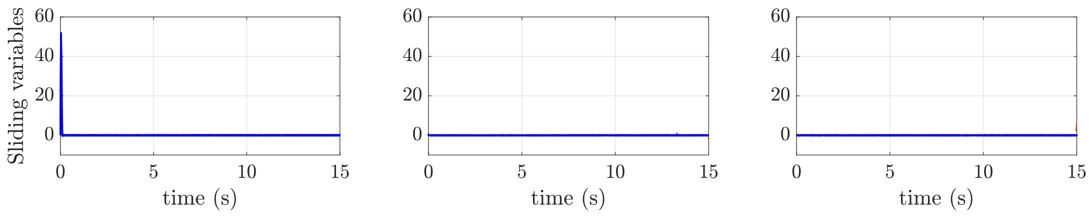


Figure 5.13: Time evolution of the sliding variables Σ_j, $j = 1, 2, 3$ for Example 2

Table 5.5: Percentage improvements obtained with the proposed switched approach for different velocity profiles

$\ddot{q}_{\max}$	$\eta_{1\text{RMS}}$ Dec.	$\eta_{1\text{RMS}}$ Sw.	$\Delta\eta_{1\text{RMS}\%}$	$\eta_{2\text{RMS}}$ Cen.	$\eta_{2\text{RMS}}$ Sw.	$\Delta\eta_{2\text{RMS}\%}$
0.0475	206.0	190.0	7.86	0.291	0.169	42.1
0.0845	236.0	218.0	7.88	0.339	0.168	50.5
0.19	286.0	264.0	7.85	0.42	0.167	60.4
0.601	84.3	64.7	23.2	0.206	0.157	23.7
0.76	133.0	92.6	30.5	0.357	0.166	53.4
1.35	101.0	79.5	21.7	0.374	0.146	61.0
3.04	134.0	121.0	9.98	0.871	0.787	9.6
5.41	142.0	113.0	20.4	0.849	0.819	3.48
12.2	128.0	117.0	9.07	0.532	0.399	24.9
48.7	211.0	190.0	10.4	0.744	0.17	77.1

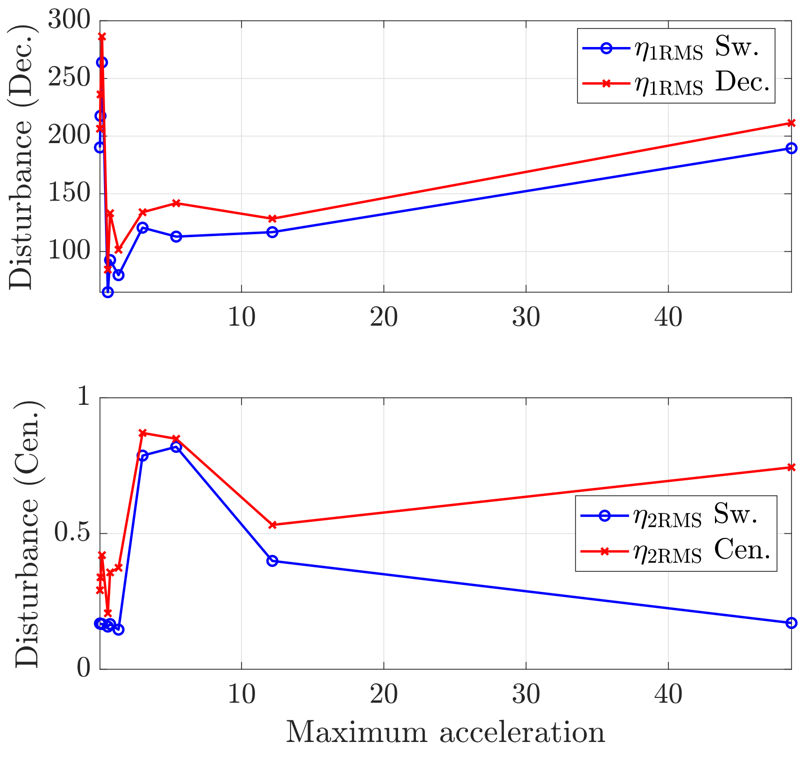


Figure 5.14: Graphical rendering of Table 5.5

Table 5.6: Result comparison with ADRC

	Joint i	e_{RMS}	τ_{RMS}	$\eta_{1\text{RMS}}$	$\eta_{2\text{RMS}}$
	1	0.0182	206.57	80.881	0.096
Sw.	2	0.0026	122.75	63.850	0.101
	3	0.0001	9.358	3.914	0.204
	1	0.0180	207.76	85.436	–
Dec.	2	0.0024	122.95	60.974	–
	3	0.00016	9.355	4.421	–
	1	0.0270	180.01	–	0.578
Cen.	2	0.0013	122.79	–	0.007
	3	0.0027	8.244	–	1.756

5.6 Comparison with ADRC

Considering Case Study 2, the proposed ISM control approach has been compared with an Active Disturbance Rejection Control (ADRC) method based on an External State Observer (ESO), as discussed in [60]. Results are summarized in Table 5.6 where it can be noticed that the performance is comparable, with the ISM control allowing to achieve a slightly better precision, as denoted by the RMS error. Yet, as discussed in this paper, the ISM control does not require the use of any ESO to achieve a satisfactory performance. This makes the proposal appealing for practical applications where the use of a low complexity but effective control scheme is required.

5.7 Conclusions

In this Chapter, a self-configuring approach for robot motion has been introduced. Specifically, by exploiting the perturbation estimation property of the ISM control law and suitably defining a switching metric, it is possible to decide whether a decentralized or a centralized approach is best for controlling the system at any given time of the operation and switch accordingly. This enables the system to operate under a broader range of conditions without the need for an a-priori choice. This can be an advantage in the case it is not possible to precisely estimate the inverse dynamics (for example, when in the presence of VGA) or when abrupt changes in the performance requirements arise. The proposed ISM based switching structure approach has been validated on an academic example of a robot with unmodeled dynamics and on the realistic model of a Comau Smart3-S2 anthropomorphic robot manipulator identified based on real data, showing improvements in the amount of disturbance acting on the plant under different motion requirements.

Chapter 6

DRL-based Switching Rule for Motion Control

This Chapter deals with the preliminary design of an intelligent self-configuring control scheme for robot manipulators, in which the switching rule regulating the self-configuration during motion is based on a policy obtained by training an agent with Deep Reinforcement Learning. Similarly to the self-configuring motion scheme introduced in Chapter 5, the proposed approach features two control structures: one of centralized type, implementing the inverse dynamics approach, and the other of decentralized type. Again, both control structures feature an Integral Sliding Mode controller, so that matched disturbances and uncertain terms caused by unmodelled dynamics or couplings effects are suitably compensated. The main contribution, with respect to the self-configuring approach based on fixed thresholds, relies on the design of an 'intelligent' switching module that has been trained on a variety of trajectories in order autonomously choose one of the two control structures present in the scheme, depending on the requested robot performances. The assessment of our proposal has been carried out relying on a model of the industrial robot manipulator Comau Smart3-S2, identified on the basis of real data.

6.1 Problem formulation

In this Section, let us recall for the convenience of the reader the main concepts behind the elements of the control scheme that will be described in the following.

Let us consider a MIMO nonlinear coupled model for the industrial manipulator used for the proposed approach. Given a n-joints robot, its dynamical model can be described as

$$B(q)\ddot{q} + n(q,\dot{q}) = \tau \tag{6.1}$$

$$n(q,\dot{q}) = C(q,\dot{q})\dot{q} + F_{\mathrm{v}}\dot{q} + F_{\mathrm{s}}\,\mathrm{sgn}(\dot{q}) + g(q). \tag{6.2}$$

For a detailed description of the elements, the reader is reminded to Section 5.2 of the previous Chapter. In the following, the time dependence of the control variables will be omitted for the sake of simplicity ($q = q(t)$ and $\dot{q} = \dot{q}(t)$).

Given the robot manipulator model in (6.1)-(6.2), assume that q^* and $\dot{q}^* \in \mathbb{R}^n$ are the reference signals for the joint variables and their first time derivative, specified a priori. It is assumed that the components of q^* are bounded and $\dot{q}^*$ is Lipschitz continuous. The tracking errors are defined as

$$e_1 = q^* - q \tag{6.3}$$

$$e_2 = \dot{q}^* - \dot{q} \tag{6.4}$$

so that $e = \begin{bmatrix} e_1 & \dot{e}_1 \end{bmatrix}^\top = \begin{bmatrix} e_1 & e_2 \end{bmatrix}^\top$. In the following, let $e_j = \begin{bmatrix} e_{1_j} & e_{2_j} \end{bmatrix}^\top$ be the position error and the velocity error of joint j, $j = 1, ..., n$, with n number of joints in the manipulator.

6.2 Self-configuring switched structure scheme

In this Section, the elements of the self-configuring scheme, represented in Figure 6.1 are briefly recalled and the DRL based switching block is introduced. In depth discussion on the decentralized and centralized control schemes can be found in Section 5.3 of the previous chapter.

6.2.1 Decentralized Control Scheme

Let us consider the robot as the composition of n linear and decouped SISO systems. The system can be represented by

$$M(q)\ddot{q}_{\mathrm{m}} = \tau_{\mathrm{m}} - d\ , \tag{6.5}$$

where

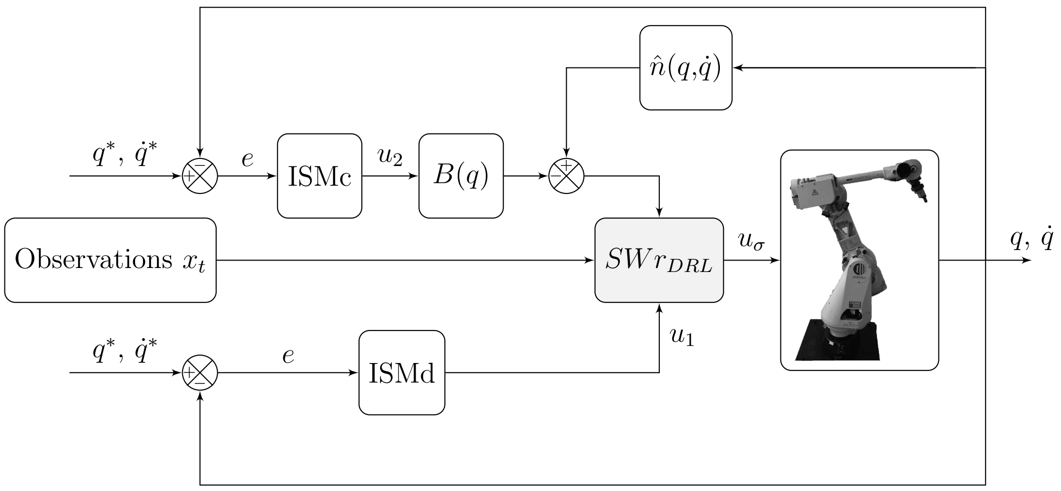


Figure 6.1: The proposed multi-loop switching ISM control scheme with a DRL based switching rule

$$M(q) = J_\mathrm{m} + K_\mathrm{r}^{-1}B(q)K_\mathrm{r}^{-1}$$

and

$$d = K_\mathrm{r}^{-1}C(q,\dot{q})K_\mathrm{r}^{-1}\dot{q}_\mathrm{m} - K_\mathrm{r}^{-1}F_\mathrm{v}K_\mathrm{r}^{-1}\dot{q}_\mathrm{m} - K_\mathrm{r}^{-1}g(q)$$

with $J_\mathrm{m} \in \mathbb{R}^{n\times n}$ and $K_\mathrm{r} \in \mathbb{R}^{n\times n}$ being the motor inerita matrix and the gear ratios. As can be noted, the larger are the reduction rations $k_{\mathrm{r}j}$, the smaller the impact of the disturbance term disturbance term d on the system. The control torque τ in (6.1) can be then computed as

$$\tau = K_\mathrm{r}M(q)K_\mathrm{r}u_1 \tag{6.6}$$

such that system (6.5) becomes

$$\ddot{q} = u_1 - K_\mathrm{r}^{-1}M^{-1}(q)d = u_1 - \eta_1 \tag{6.7}$$

where η_1 is an uncertainity that can be rejected by a control law u_1 designed as an ISM controller, in the form of

$$u_1 = \ddot{q}^* + K_{\mathrm{D}_1}e_2 + K_{\mathrm{P}_1}e_1 + u_{11} \ , \tag{6.8}$$

with

$$u_{11}(t) = -K_{11}\mathrm{sgn}\left(\Sigma(t)\right) \ , \tag{6.9}$$

as discussed in Chapter 5, Section 5.4.

6.2.2 Centralized Control Scheme

Let us assume again to precisely estimate the inertia matrix $B(q)$ and a quite accurate estimation of $n(q,\, \dot{q})$, such that $\hat{n}(q,\, \dot{q}) \neq n(q,\, \dot{q})$. Moreover, let u_2 be an auxiliary control vector such that the control torque is selected as

$$\tau = B(q)u_2 + \hat{n}(q, \dot{q}) \, . \tag{6.10}$$

Substituting (6.10) into model (6.1)-(6.2), one has

$$B(q)\ddot{q} + n(q, \dot{q}) = B(q)u_2 + \hat{n}(q, \dot{q}) \, , \tag{6.11}$$

which is a chain of n decoupled double integrator plants of the type

$$\ddot{q} = u_2 - \eta_2 \tag{6.12}$$

where the matched uncertain terms η_2 are rejected by a control law u_2 designed as an ISM controller, in the form of

$$u_2 = \ddot{q}^* + K_{\mathrm{D}_2} e_2 + K_{\mathrm{P}_2} e_1 + u_{12} \ , \tag{6.13}$$

with

$$u_{12}(t) = -K_{12} \mathrm{sgn} \left(\Sigma(t) \right) \ , \tag{6.14}$$

again as discussed in Chapter 5, Section 5.4.

6.2.3 DRL-based Switching Block

The decision to switch to the decentralized structure (Mode 1) or to the centralized one (Mode 2) is made by an agent trained with a Deep Reinforcement Learning algorithm. Specifically, for this work, the Normalized Advantage Function algorithm [18] was used for training, as it is suitable for continuous variables, both observed and controlled, and does not require discretization. Once trained, the deployed policy, given a set of observations x_t at step t of the simulation, is able to generate an action $u_t = \sigma$ with $\sigma = 1, 2$ for selecting either the decentralized controller or the decentralized one, as summarized in Figure 6.2. The framework used for training the agent will be detailed in the following Section 6.3.

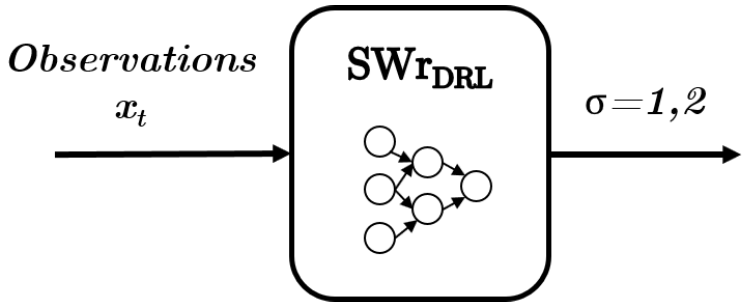


Figure 6.2: Scheme of the DRL based Switching Block.

6.3 RL Framework for Self-Configuring Motion Control

In this Section, the elements composing the reinforcement learning framework used for training will be detailed. Given any trajectory in the joint space, the learning process aims to find an optimal strategy for deciding which control structure present in the scheme is most suitable for meeting the demands in terms of tracking performances, while keeping into consideration that frequent variations of the control structure may cause excessive stress on the mechanical system.

6.3.1 State space

The state space $\mathcal{X}$ is defined as

$$\mathcal{X} \triangleq \{e,\, \dot{e},\, \ddot{e},\, \eta,\, \tau\} \ , \tag{6.15}$$

where the elements are defined as follows:

- $e \in \mathbb{R}^3$, $\dot{e} \in \mathbb{R}^3$ and $\ddot{e} \in \mathbb{R}^3$ are the joints position, velocity and accelerations errors retrieved during the robot's motion.
- $\eta \in \mathbb{R}^3$ are the value of the coupling effects on each joint, estimated by the ISM controller.
- τ are the torques exerted by each joint.

Each element of the state space is sampled at any time t of a given episode, having a finite duration T. Note that, although one could measure position and velocity errors through the use of encoders, the acceleration error is, in this case, computed numerically. This is done to feed the DRL framework with as much information as possible, which was believed to facilitate the learning process.

6.3.2 Action space

The action space $\mathcal{U}$ is defined as

$$\mathcal{U} \triangleq \{\sigma\}\,, \tag{6.16}$$

The action, in this case, is only a flag that determines on which control structure should be used at a given time instant; more specifically, the action equals 1 for the decentralized structure and 2 for the centralized one.

6.3.3 Reward

Since the decision procedure must be transparent with respect to the motion control problem that must be solved, in the sense that it should not have a negative impact on the robot performances, the reward function is defined in a way that encourages the least possible tracking error and a smooth transition between one scheme to the other, depending on the status of the robot. In this case, the reward is defined as:

$$r = -(c_1 r_{\text{torque}} + c_2 r_e + c_3 r_{\dot{e}} + c_4 r_{\ddot{e}} + cost) \tag{6.17}$$

where r_{torque}, r_e, $r_{\dot{e}}$ and $r_{\ddot{e}}$ are computed as the Euclidean norm of the normalized values of the torques, the position errors, the velocity errors and the acceleration errors of each joint, respectively. The terms $c_1 = 50$, $c_2 = 500$, $c_3 = 1000$ and $c_4 = 100$ are the weights associated to each component. The set of weights described has been defined via repeated experiments. As a matter of fact, the experiments showed that velocity errors are relatively more relevant than others for the overall performance, and are thus weighted more. The cost parameter is representative of the extra computational workload that the centralized scheme may entail, due to the presence of the inverse dynamics described in Section 6.2.2. Clearly, such cost may well depend on the specific application scenario considered; hence, the values described here are just exemplificative. In other words, in the proposed DRL framework, the computational cost is a *parameter* that can be adapted to the specific case study. It is worth noting that given the negativity of the overall

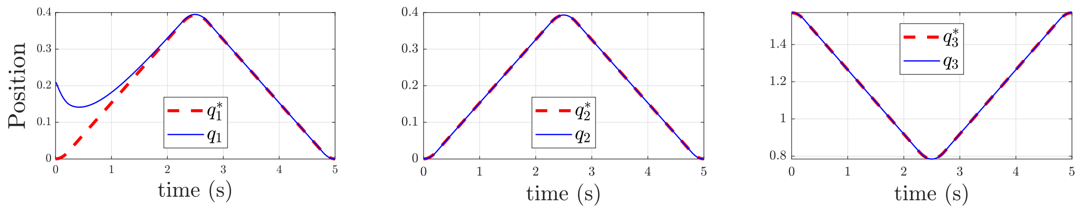


Figure 6.3: Example of trajectory followed by the robot joints during a training episode. In this case, $q_0^* = [0,\ 0,\ \pi/2]^\top$, $q_{t_m}^* = [\pi/8,\ \pi/8,\ 0]^\top$ and $t_m = 2.5$ s (i.e., half of the simulation duration). The robot's initial configuration is $q_0 = [\pi/15,\ 0,\ \pi/2]^\top$.

function, the maximization of the reward function in terms of reinforcement learning means, in this case, the minimization of the absolute value of the cumulative reward per episode (i.e., we want the values of all terms to be as low as possible), steering the function to zero.

6.4 Case study

In this section, the results obtained from training the switching block will be illustrated. The experiments were carried out on the simulated model of a Comau Smart-S2 identified on the basis of real data [57, 58] (Appendix A). During each episode, the robot has to track a pick-and-place trajectory in the operative space defined by a random Trapezoidal Velocity Profile (TVP), such as the one in Figure 6.3. Specifically, the TVP is such that, given a random initial position q_0^*, a random intermediate position $q_{t_m}^*$ and a random time $t_m \in [0,\ T_f]$, with $T_f = 5$ s being the total duration of each episode, the joint must reach $q_{t_m}^*$ by time t_m, after which the joint reverts its motion, back to the initial position q_0^*. Initial and intermediate positions are sampled from a uniform distribution within the motion range of the respective joints. The control structure selected at the initial time instant is chosen randomly as well. The policy derived from a learning session is then validated using a set of predefined trajectories characterized by different TVP profiles. Results refer to a policy obtained after 500 episodes, each consisting of 100 steps of 0.05 s each. Notice that, although the framework elements are retrieved every 0.05, the simulation runs with a timestep $T = 5 \times 10^{-5}$ s. In Figure 6.4, a polynomial interpolation of the cumulative rewards obtained for different cost parameters (i.e., the penalty for using the computationally expensive centralized control) are

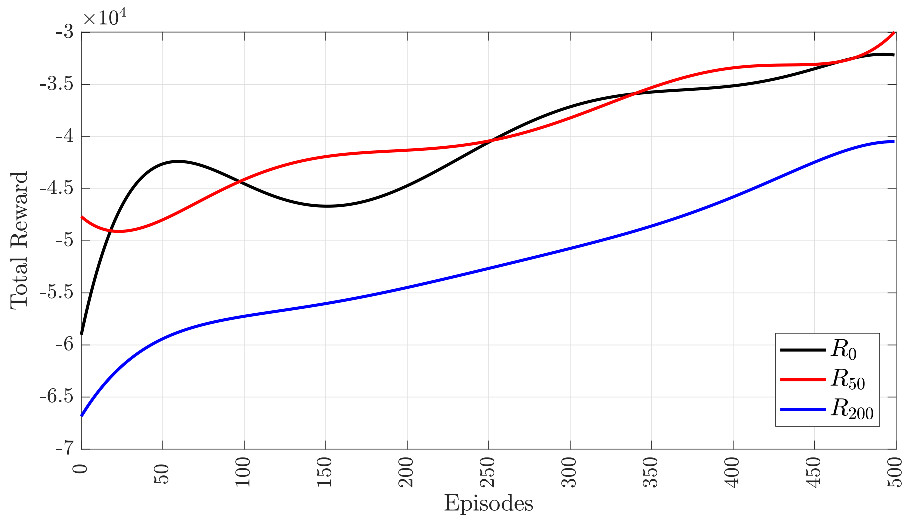


Figure 6.4: Polynomial interpolation of the cumulative rewards obtained when training the agent using a cost parameter in the reward function equal to 0, 50, and 200.

reported; as it can be observed, all rewards present a growing trend, indicating that the training is leading to an optimization of the selected reward. Due to internal memory management, training times become infeasibly long after about 500 episodes. For this reason, it is stopped at this point. Control parameters are the same as the ones employed in the threshold-based approach in Chapter 5.

6.4.1 System specifications

The experiments have been carried out in simulation using Simulink for MATLAB. The NAF algorithm is implemented in TensorFlow and suitably interfaced with a MATLAB 2017a script, controlling and sampling the Simulink model encapsulating the environment. The solver used for solving the model dynamics was chosen as ode1. The training sessions have been carried out on a machine mounting a 8x intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz with a 32GB RAM and a NVIDIA Quadro k5000 GPU with 4GB DRAM.

6.4.2 Results

Considering the trajectory previously introduced in Figure 6.3, deploying the policy obtained after training yelds to results reported in Figure 6.5.

As expected, when the cost parameter of the centralized scheme is set to zero, the policy obtained through DRL at the end of a training session tends to that of a fixed centralized approach, since such control scheme guarantees best performances in terms of velocity and acceleration, thus reducing the impact of terms $r_{\dot{e}}$ and $r_{\ddot{e}}$ on the episodic reward. With increasing values of the cost parameter for the centralized approach, the agent becomes more leaning towards a decentralized one, thus entailing more switching actions, as reported in Figure 6.5. With all non-zero values of the cost parameter, the actual amount of switching actions performed by the DRL agent can be controlled by acting on the c_1 in Eq. (6.17), since each switch between the two structures inevitably causes a peak in the torques exerted by the joints, as can be noted for example in Figure 6.6.

6.5 Conclusions

This Chapter presented preliminary results obtained for a DRL-enabled self-configuring motion control scheme for robot manipulators, based on the switched use of a decentralized control structure and a centralized inverse dynamics based control. The rule relies on the policy obtained by training and agent through the NAF algorithm for DRL. The proposal, which is then a combination of classical control concepts and machine learning elements, allows one to extend the operative velocity and acceleration range in which the robot manipulator can work. The proposed approach has been tested on a model of an industrial Comau Smart3-S2 anthropomorphic robot manipulator identified on the basis of real data.

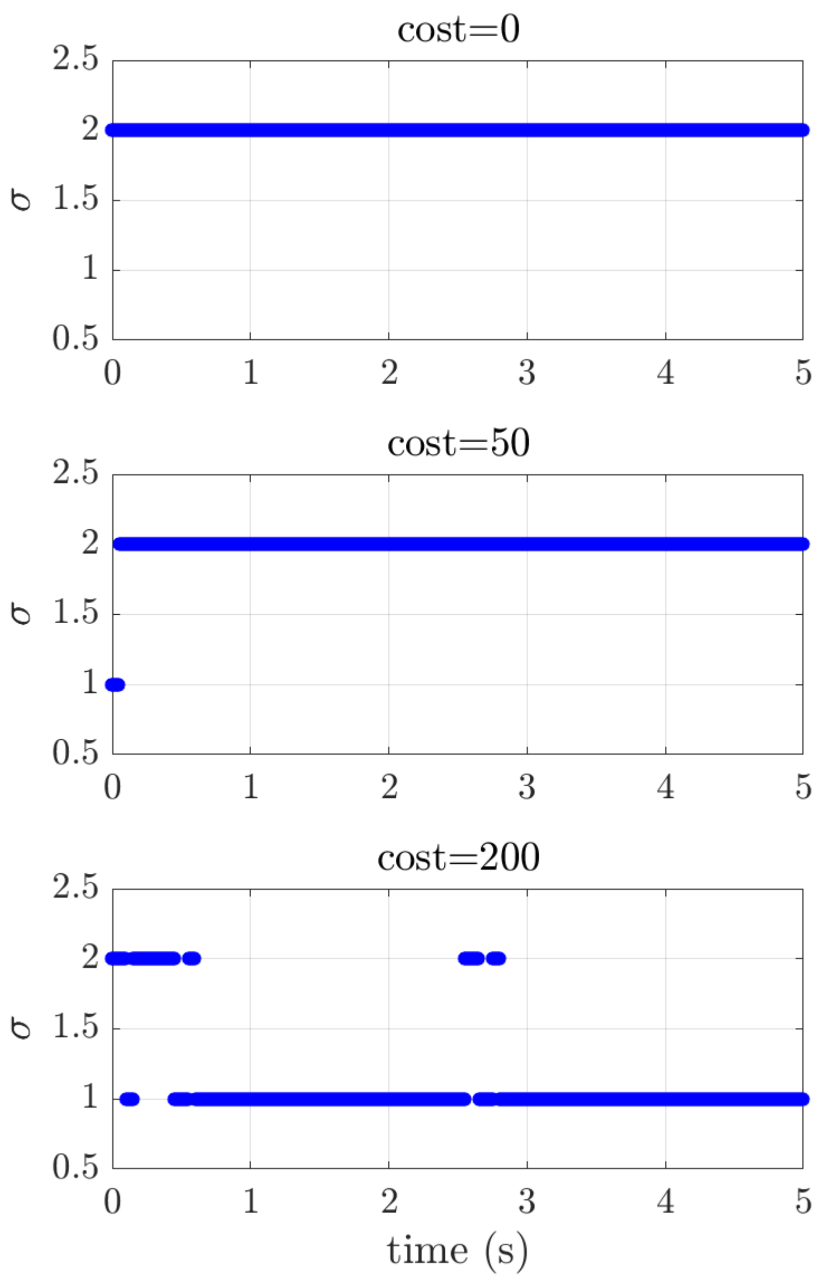


Figure 6.5: Policy on a validation trajectory, with cost parameters equal to 0, 50 and 200 (from top to bottom).

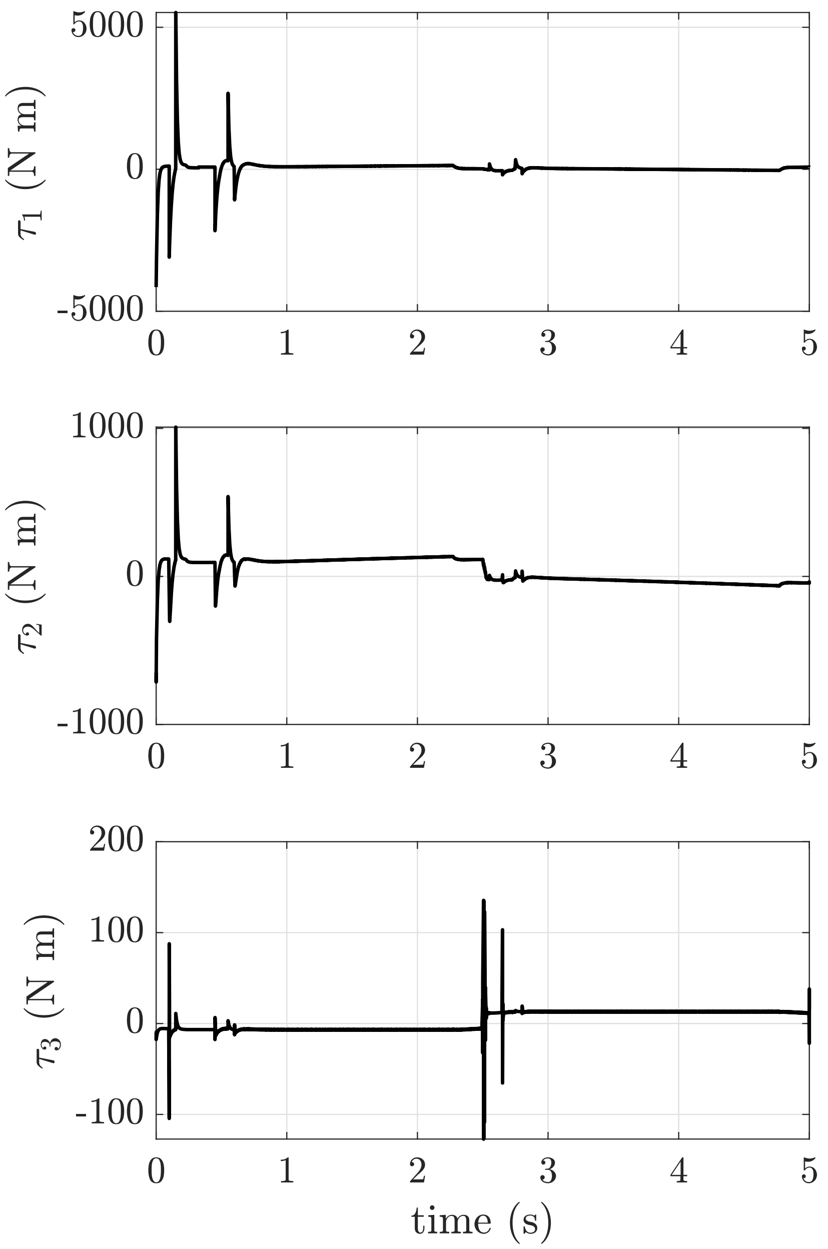


Figure 6.6: Torques exerted by the three joints during the execution of a validation trajectory using the policy with $cost = 200$

Chapter 7

Conclusions

This Part of the Thesis dealt with the design of a self-configuring approach for motion control of industrial manipulators. The considered robot is tasked to follow different types of trajectories in the joint space. In typical applications, whether to perform the control task using a decentralized approach or a centralized one is one of the design choices that need to be made a-priori. Nevertheless, the circumstances under which the robot operates may vary unexpectedly, resulting in degraded performance, should the adopted control structure not successfully handle them. To this end, a novel switched structure control system, implementing both a decentralized control structure and a centralized one based on the computation of the inverse dynamics of the robot, is proposed with a switching block that enables any of the two control strategies. The logic behind the switch is arbitrary, depending on the specific application, and for this work, the following approaches were explored:

1. The switching block operates on the basis of the unmodelled dynamics affecting the system at any given time, suitably estimated by the Integral Sliding Mode controller.

2. The switching block implements a policy obtained after training an agent through Deep Reinforcement Learning, intending to minimize tracking errors, which selects at any given time the most suitable control structure depending on the observed variables.

The first approach has been tested on a manipulator featuring Variable Gears Actuators and an industrial manipulator Comau Smart3-S2 identified based on real data. In both cases, tracking performances with the proposed approach are satisfactory, and a decrease in the effects of unmodelled dynamics are observed with respect to the stand-alone approaches. Finally, some preliminary results for the DRL enabled switching block are briefly discussed applied to the model of the Comau Smart3-S2, effectively showing that the agent can successfully learn to select the strategy that guarantees higher tracking performances. Nevertheless, by adding a measure of each control choice's computational cost, the agent also learns to take it into account when making decisions.

From a qualitative point of view, some considerations can be made about the choice of the switching rule in cases such as the ones presented in the previous chapter. Indeed, the capability of Deep Reinforcement Learning to obtain an (ideally) optimal policy starting from observed data would reduce the amount of hand-engineering dedicated to the selection of the suitable thresholds for the switching metrics: although the considered case only includes two control approaches, the proposal to switch among different controllers could well be extended to a higher number of methodologies. In this case, it might be desirable to let the decision-making process be handled by an 'intelligent' agent, suitably trained to select the best approach. On the other hand, the amount of time and data required to train such agents could, in some cases, overcome the advantages provided. Making reference to the presented work, it was observed that sometimes the differences in performance between the decentralized control scheme and the centralized one, over certain trajectories, was not substantial enough to clearly select a controller over the other: this could indeed lead to mixed results, as the agent is unable to classify the different methodologies. Nevertheless, the idea could be worthy of investigation for more sophisticated scenarios in future work.

For what concerns physical implementations of the presented framework, the deployment of the DRL agent is not believed to entail any specific problems, since it does not perform any end-to-end control that could lead to abrupt reactions. Nevertheless, from a practical point of view, the estimation of the disturbance requires filtering the discontinuous signal. Such a filter should be then carefully designed in order to take into account the time to converge, without losing any information on the perturbation shape provided by the high-frequency switching typical of Sliding Mode Controllers.

Part III

Self-configuring approaches for robot collision avoidance

Chapter 8

Motivation and state of the art

As anticipated in the Thesis introduction, since robots are required to perform increasingly demanding tasks in different and complex environments, significant focus is put in robotics research to ensure that such systems are able to move and interact with the surroundings without endangering equipment and, most importantly, humans [3]. For this reason, in the context of the so-called physical Human-Robot Interaction (pHRI), or in the case of robots operating alone in cluttered environments, obstacle detection and avoidance remains a core issue [61, 4]. A number of contributions document the interest in handling physical interaction between the robots and the environment, and especially on collision avoidance. Typically, collision avoidance methods consist of three parts: (i) perception of the environment; (ii) collision avoidance algorithms; and (iii) robot control [62]. These parts and their outputs can be explicated in seven phases (pre-collision, detection, isolation, identification, classification, reaction and post-collision) over a collision avoidance pipeline as reported in the survey [63]. An approach based on Artificial Potential Fields, introduced in [64], allows avoidance of obstacles in the proximity by generating artificial force fields that are then transformed into joint torques. This approach was later improved in [65, 66]. Another approach based instead on kinetostatic safety fields for robot operation is introduced in [67]. Real-time, adaptive motion planning is explored in [68, 69]. Again in [70], a trajectory generation approach based on the data provided by depth sensors is proposed in order to guarantee safety constraints. Many strategies then rely on Optimal Control Problems (OCPs), such as [71, 72, 73]. It is worth highlighting that, in order to implement potential fields methods, the dynamical model of the manipulator often must be known or accurately estimated. In contrast, planning methods require constant sampling and online planning of the trajectory in order to find an obstacle free path. Both requirements might be hard to satisfy in the presence of multiple moving obstacles or an unknown environment.

The main contribution illustrated in this Part of the Thesis is the introduction of an alternative approach based on a Deep Reinforcement Learning framework to perform end-to-end control for collision avoidance applied to industrial manipulators [74]. Then, on the topic of self-configuring systems, the proposed end-to-end strategy is applied in a hybrid control algorithm, which enables the robot to be controlled by both conventional motion approaches and end-to-end, DRL based ones [75]. Furthermore, the proposed framework is deployed for a case study involving a teleoperated cooperative robot (an UR5 by Universal Robot) and a case study with an industrial manipulator (an Epson VT6 by Epson).

8.1 Structure

This Part of the Thesis is structured as follows.

- **Chapter 9** introduces and discusses in detail the proposed DRL framework for collision avoidance of robot manipulators. The elements of the framework are described in detail, together with the setup used for the experiments. Preliminary results on preliminary case studies are reported and commented on.

- **Chapter 10** introduces the concept of transfer learning and discusses its effects on the considered scenarios in terms of its impact on the agent's performance and the possibility of reusing previous knowledge to obtain a policy for more complex environments.

- **Chapter 11** Chapter 3 discusses a novel hybrid approach for motion and collision avoidance of industrial manipulators, which allows the robot to be controlled by both conventional methods and end-to-end ones, conferring the system a self-configuring capability.

- **Chapter 12** illustrates the application of DRL for collision avoidance to a teleoperated robot, which has to track a reference originated from a human operator while avoiding obstacles present in its working space. The performance of DRL is compared to that of an MPC with the same settings. Experimental results are reported.

- **Chapter 13** discusses the deployment and reproduction of a trained DRL policy to an industrial manipulator. The interfacing between the components is described, and preliminary experimental results are reported.
- **Chapter 14** gathers some concluding remarks.

Chapter 9

Deep Reinforcement Learning for Collision Avoidance

This Chapter presents the design of a real-time collision avoidance approach for industrial manipulators exploiting machine learning techniques. Specifically, Deep Reinforcement Learning (DRL) methods are applied to robots whose working space is invaded by moving obstacles with unpredictable direction changes. Motivated by the system's complexity, featuring states and inputs defined in the continuous space, the Normalized Advantage Function (NAF) [18] algorithm is used to train the agent for end-to-end, model-free control. The proposal is assessed relying on the virtualized model of an anthropomorphic robot manipulator, a Comau Smart3-S2, interfaced with external tools for evaluation, control, and automatic training.

9.1 Problem definition

Let us consider a situation in which a robot manipulator is requested to operate in a populated industrial environment, possibly invaded by obstacles. During the motion, the robot could collide with other entities while reaching its target or executing a specific task. Furthermore, assume that the objects move randomly so that physical interaction between the environment and the robot can unexpectedly happen. To this end, let us introduce a simplification of the robot model and the definition for the collision avoidance control problem considered for this work; this is useful in order to recast the problem formulation into the reinforcement learning framework introduced in Chapter 3.

9.1.1 Robot model

Let us consider an industrial manipulator with an open kinematic chain, and let $q \in \mathbb{R}^n$ be the joint variables related to the motor positions, with n being the number of degrees of freedom. Given the end-effector's pose vector $x_\mathrm{e} = \begin{bmatrix} p_\mathrm{e} & \varphi_\mathrm{e} \end{bmatrix}^\top$, with p_e being the position, and φ_e the orientation in the operative space, the direct kinematics [26] is indicated as

$$x_\mathrm{e} = k(q), \tag{9.1}$$

with $k(q) \in \mathbb{R}^m$, where m is the dimension of the operative space, being a nonlinear function depending on the joint variables. Since, in practice, it is convenient to provide joint position and/or velocity references to the internal control loops, the dynamic model of the robot can be expressed as

$$\ddot{q} = f(q, \dot{q}) \ , \tag{9.2}$$

where $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ is a function of path coordinates and its time derivatives.

9.1.2 Collision avoidance problem

Let us now consider model (9.2) where $\dot{q}$ is assumed to be the input action, namely u in the following. Furthermore, the operative space strictly depends on the manipulator's geometry and the mechanical limits on the joints, so that input and state constraints are of the form $h(q, \dot{q}) \leq 0$ with $h : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^\ell$, and ℓ being the number of constraints. Finally, let $\mathcal{W}(q(t)) \subset \mathbb{R}^m$ be the space occupied by the robot at the instant t, and $\mathcal{O} \subset \mathbb{R}^m$ be the space occupied by the obstacles. Having in mind to execute a possible industrial task (e.g., spot welding, pick and place, or point-to-point motions), the goal is now to find a velocity control sequence $\dot{q}^*$ over the horizon T, which makes the robot end-effector move from the initial state $x_{\mathrm{e}0}$ to a target point x^*, (eventually following a predefined trajectory $x_\mathrm{e}^\star$), while avoiding the obstacles $\mathcal{O} \subset \mathbb{R}^m$ with minimum deviation from the target point and applying minimum control action. Therefore, let us define a finite-horizon Collision

Avoidance Optimal Control Problem (CAOCP) as

$$\begin{aligned} &\min_{u} \int_{0}^{T} c_1 R_{\mathrm{T}}(x_{\mathrm{e}}(q), x^*) + c_2 R_{\mathrm{A}}(u)\ \mathrm{d}t \\ \text{s.t.}\quad & \ddot{q} = f(q, \dot{q}), \quad \dot{q} = u, \quad x_{\mathrm{e}}(q) = k(q) \\ & h(q, \dot{q}) \leq 0 \\ & \mathcal{W}(q(t)) \cap \mathcal{O} \equiv \varnothing \\ & x_{\mathrm{e}}(0) = x_{\mathrm{e}0} \end{aligned} \tag{9.3}$$

where $R_{\mathrm{T}}(x_{\mathrm{e}}(q), x^*)$ and $R_{\mathrm{A}}(u)$ are suitably selected functions to account for the tracking error (i.e., the current end-effector position and the desired target point x^* in the operative space) and the system inputs (i.e., the joint velocities $\dot{q}$). Moreover, the constants c_1 and c_2 are suitably selected weights.

The key difficulty in solving (9.3) is given by the presence of the collision avoidance constraints $\mathcal{W}(q(t)) \cap \mathcal{O} \equiv \varnothing$, which are in general non-convex and non-differentiable.

9.2 RL Framework for Collision Avoidance

Making reference to the scenario introduced in Section 9.1, and the CAOCP in (9.3), let us recast the problem using the RL framework elements. Further, Let us assume that the dynamical parameters of the considered industrial manipulator are unknown, and are thus solely relying on observed information. In the same way, it is not possible to predict the motion of external elements invading the workspace, with the exception of the specific design parameters during the environment design phase, which will be detailed in Section 9.4.

9.2.1 State Space

The state space $\mathcal{X}$ is defined as

$$\mathcal{X} \triangleq \{q,\, \dot{q},\, p_{\mathrm{e}},\, p^*,\, p_{\mathrm{o}},\, \dot{p}_{\mathrm{o}}\}\ , \tag{9.4}$$

where the elements are defined as follows:

- $q \in \mathbb{R}^6$ and $\dot{q} \in \mathbb{R}^6$ are the vectors of each joint position and velocity, expressed in rad and rad/s, respectively.

- $p_\mathrm{e} \in \mathbb{R}^3$ is the robot's end-effector position $p_\mathrm{e} = \left[p_{\mathrm{e}_x}, p_{\mathrm{e}_y}, p_{\mathrm{e}_z}\right]$, expressed in m.
- $p^* \in \mathbb{R}^3$ is the target point's position $p^* = \left[p^*_x, p^*_y, p^*_z\right]$, expressed in m.
- $p_\mathrm{o} \in \mathbb{R}^3$ is the obstacle's position $p_\mathrm{o} = [p_{\mathrm{o}x}, p_{\mathrm{o}y}, p_{\mathrm{o}z}]$, expressed in m.
- $\dot{p}_\mathrm{o} \in \mathbb{R}$ is the obstacle's velocity in absolute value, expressed in m/s.

Each element of the state space is retrieved at each time t of the episode during training. Note that, in a real case, there should be suitable vision sensors that retrieve the full status of the environment at each step and then, after a series of image processing operations, extract the relevant information for the training. For instance, we assume to that the obstacle's position in space is detected by a perception system that allows for triangulation (e.g., stereo cameras). For the sake of simplicity, let us bypass this step for the time being and assume to be instantly provided with the relevant data about the scenario.

9.2.2 Action Space

The action space $\mathcal{U}$ is defined as

$$\mathcal{U} \triangleq \{\dot{q}^*\}, \tag{9.5}$$

where $\dot{q}^* \in \mathbb{R}^6$ is the vector of angular velocity references for each joint. An action issued at time t means that the robot must reach the requested angular velocities in the following time step $t+1$, by applying the maximum torque available. Tracking of the specified reference is handled by the integrated low-level velocity controller of the manipulator. Near-perfect tracking performance can be assumed, without loss of generality.

9.2.3 Reward Function

The reward function, as recalled in Chapter 3, is a scalar function used to define the desired behavior that one wants to achieve by training the agent. Thus, at each time instant t, it provides a feedback on "how well" the agent has performed given the observed state and the performed actions. For the collision avoidance problem introduced, the reward function is designed as

$$r = -(c_1 R_\mathrm{T} + c_2 R_\mathrm{A} + c_3 R_\mathrm{O}) \tag{9.6}$$

so that it is the weighted sum of three terms, accounting for different aspects of the training. Specifically:

1. The distance between the end-effector and the target point. The term R_T is computed using a *Huber-Loss function* and is defined as

$$R_\mathrm{T} = L_\delta(d) = \begin{cases} \frac{1}{2}d^2 & \text{for } |d| < \delta \\ \delta\left(|d| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{9.7}$$

   where d is the Euclidean distance between the tip and the target and δ is a parameter that determines its smoothness.

2. The magnitude of the actions performed by the manipulator, also known as *regularization term*, computed as

$$R_\mathrm{A} = \|u\|^2 = \|\dot{q}^*\|^2 , \tag{9.8}$$

   which has the purpose of encouraging smaller control actions, i.e., minimize the requested joint velocities.

3. The obstacle avoidance, computed as

$$R_\mathrm{O} = \left(\frac{d_\mathrm{ref}}{d_\mathrm{O} + d_\mathrm{ref}}\right)^p \tag{9.9}$$

   where d_O is the minimum distance between the robot's structure and the obstacle, d_ref is a constant parameter ensuring that $0 < R_\mathrm{O} < 1$, and p is a decay exponent. A limitation in (9.9), however, is that it penalizes to a lesser relative extent collisions occurring at a greater distance from the target point. To avoid this undesirable effect, R_O is actually computed as follows

$$R_\mathrm{O} = \max\left(k_r \frac{R_\mathrm{T}}{c_3}\,;\left(\frac{d_\mathrm{ref}}{d_\mathrm{O} + d_\mathrm{ref}}\right)^p\right) \tag{9.10}$$

   where k_r is a positive integer that ensures a minimum, fixed proportion between R_O and R_T.

The relative weights of the three terms above can be tuned via the constant parameters c_1, c_2, c_3 .

In order to better visualize the behaviour of the considered reward function, let us consider a planar cross section of the robot's working space. Such plane is parallel to the floor and at the same height of the target and the trajectory of the obstacle. The target point placed in (0.5, 0.5) and the obstacle centered in (0.1, 0). For the sake of simplicity, let us assume the robot is a point moving in the space and considered only the R_T and R_O terms in (9.7) and (9.9), resperctively. As it can be seen in Figure 9.1, the resulting function is concave with values ≤ 0, and it reaches its maximum when the tip of the robot is placed on the target point; moreover, the value diminishes as the robot moves in proximity of the obstacle. Furthermore, Figure 9.2 shows the 3D graph of the R_T and R_O components in the reward function in order to better show the behavior of the improved R_O term introduced in (9.10). As it can be observed, the reward component accounting for collisions with the obstacle in the proximity of the target point is computed as (9.9), with a penalty equal to 1. When collisions happen far from the target position, a steeper variation of R_T is used instead.

Given the negativity of the overall function, the maximization of the reward function in terms of reinforcement learning means, in this case, the minimization of the absolute value of the cumulative reward per episode (i.e., the values of all terms need to be as low as possible), steering the function to zero.

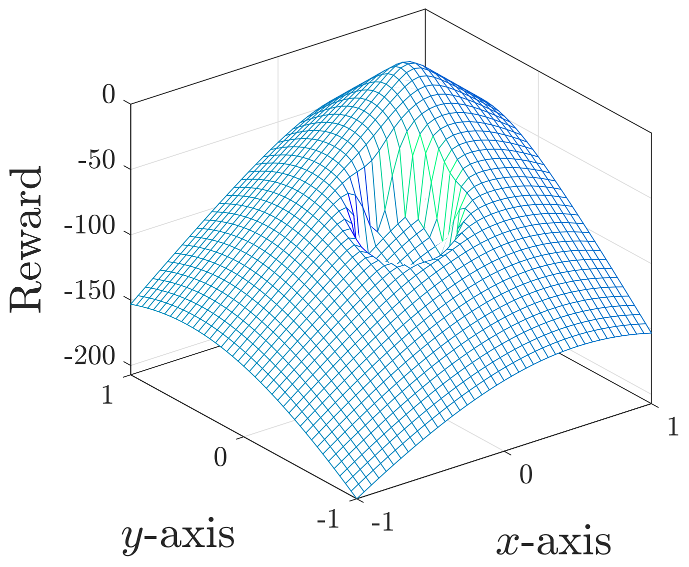


Figure 9.1: 3D rendering of the reward function on the planar section of the considered environment for target positions

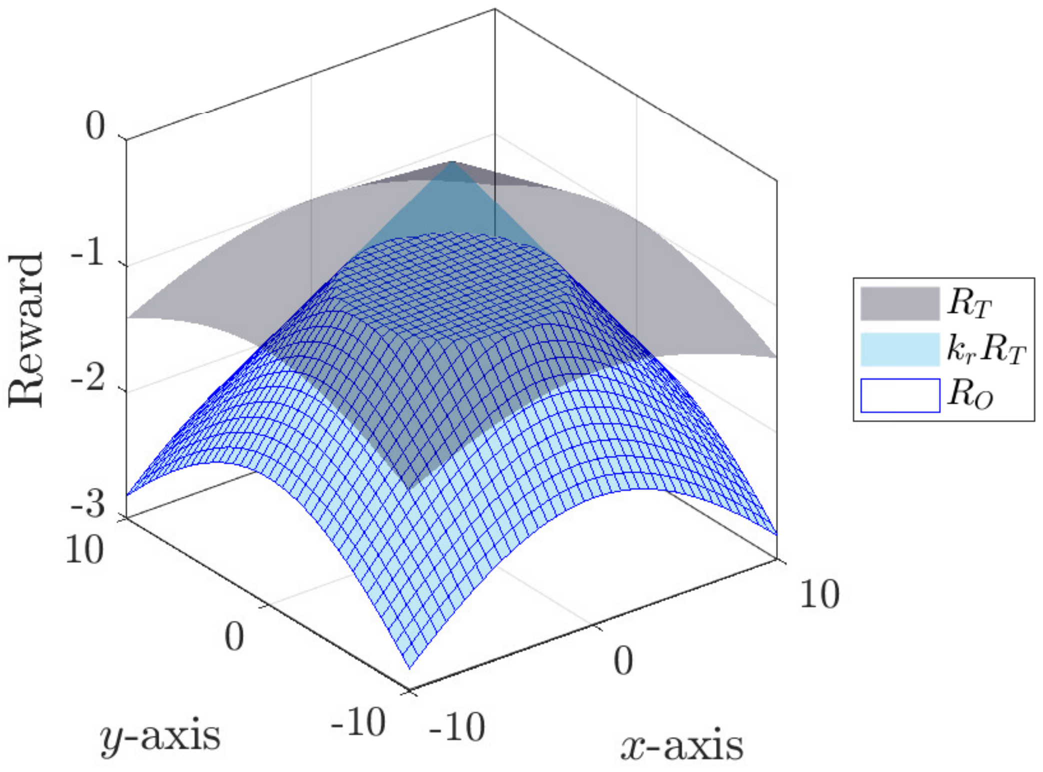


Figure 9.2: The R_T and R_O components of the reward function computed on a planar cross-section of the environment, where the target point is placed in $(0, 0)$. The surface in light gray corresponds to R_T while the surface in cyan corresponds to $k_r R_\mathrm{T}$. The net surface in blue corresponds to the maximal value of R_O, i.e., in case of collision with the obstacle, when $c_2 = 1$; farther from the target, the relative proportion $R_\mathrm{O}/R_\mathrm{T}$ is constant and equal to k_r.

9.2.4 Hyperparameters

Besides the hyperparameters needed to be set for the learning algorithms, the training using the proposed framework also requires the following:

- c_1, c_2, c_3: weights of the three reward function components.
- k_r: coefficient on the obstacle avoidance component compared to R_T.
- δ: discriminating parameter for the Huber-Loss function in (9.7).
- d_{ref}: default minimum distance between the obstacle and the body of the manipulator.
- p: exponential decay of the avoidance term when the distance from the obstacle increases.

9.3 Environment description and setup

The initial evaluation of the overall approach involving the application of the NAF algorithm to obstacle avoidance in the simulated environment was performed on the scenarios described in the following. The scenes predisposed for the training process, illustrated in Figure 9.3, all present the following elements:

- **Manipulator:** the virtual replica of the Comau Smart3-S2. It is dynamic and respondable with the external elements, and the motors are enabled in Force/Torque mode, with external position control loops disabled in order to allow joint velocity control.
- **Target:** the point in space the robot has to reach with its end-effector. It can either be fixed or randomly placed at the beginning of each episode during training.
- **Production line:** represented by a plane in front of the robot. The target is placed inside its area. The plane is 2.25 m wide and 0.4 m long. The plane is set at an height of 0.875 m, parallel to the floor. It is entirely contained in the robot's operative space, so that the target is always reachable.

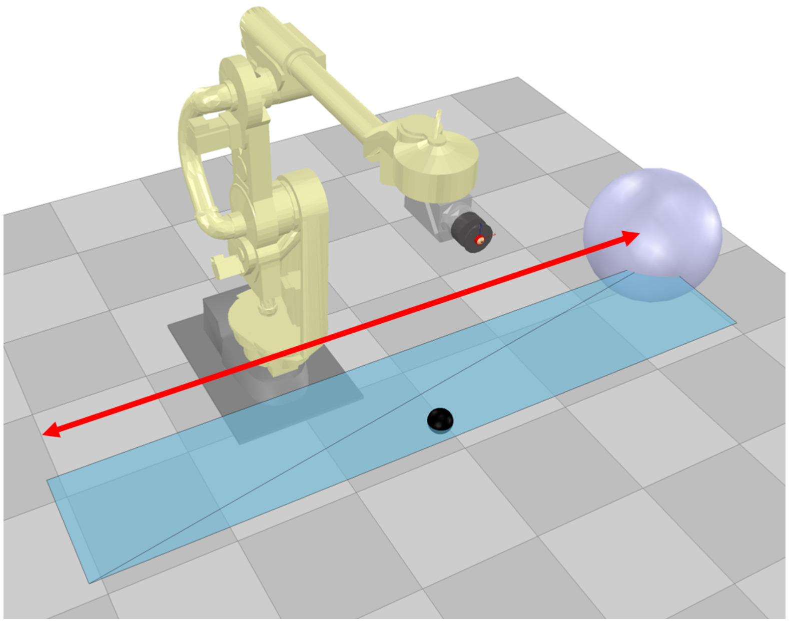


Figure 9.3: V-Rep scene with the virtual Comau Smart3-S2, a spherical obstacle moving along a linear path (red arrow) and target (black circle)

- **Obstacle:** represented by a sphere with ray 0.25 m that moves along a linear path. It can either move regularly back and forward or change its direction randomly, and it can also randomly stop. The shape is respondable, hence collidable by the manipulator (i.e., the physics engine will handle the collision reaction). The sphere moves at a velocity of 0.2 m/s.
- **Path:** the path where the obstacle moves. Its length is 2.25 m and is placed between the target and the manipulator.

All experiments share the same arrangement of state space, action space, and reward function, as well as the same hyperparameters discussed in Section 9.2, summarized in Table 9.1. In the following, it is assumed that, apart from the encoders fastened on the robot joints, a vision system is present to detect objects' motions with respect to a given reference frame.

9.3.1 System specifications

The experiments have been carried out in simulation using the general purpose robotics simulator V-Rep from Coppelia Robotics [76]. The NAF algorithm is implemented in TensorFlow and suitably interfaced with the simulator encapsulating

Table 9.1: Hyperparameter values for the experiments

Parameters	Value
number of time steps	360
time step	50 ms
c_1	1000
c_2	200
c_3	60
δ	0.1
p	8
d_{ref}	0.2
k_r	2
discount factor γ	0.99
update factor τ	0.001
learning rate η	0.001
noise type $\mathcal{D}$	*Ornstein–Uhlenbeck*
noise decay factor	0.01
noise scale	1

the environment via the PyRep [77] plugin. The dynamics of the environment are handled by the ODE Physics engine running with a frequency of 200 Hz. The training sessions have been carried out on a machine mounting a 8x intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz with a 32GB RAM and a NVIDIA Quadro k5000 GPU with 4GB DRAM.

9.4 Case studies

In this section, the results obtained from the training of the industrial manipulator will be illustrated. First, the description of the considered scenarios for training is provided in detail. Then, the results concerning the application of the NAF algorithm are presented and discussed. For all the experiments, the total reward and the average loss function per episode were traced. Moreover, data relative to the distance between the tip and the target, and the distance between obstacle and manipulator, are collected to better show the behavior of the robot after training; the results obtained from the trained agent refer to the values obtained over 550 episodes of 360 steps each, with a timestep duration of 50 ms.

Case 1: Fixed target, moving obstacle.

The position of the target point is the same in every episode, and the obstacle moves from one end to the other of the linear path, at constant velocity. The target position p^* to be reached is placed at the center of the production line, at an initial distance of 60 cm from the end-effector. Results regarding training and testing of the last episode are reported in Figure 9.4.

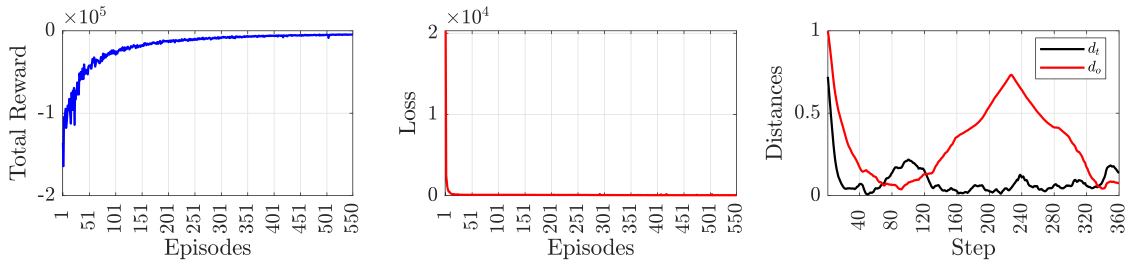


Figure 9.4: Results for Case 1

Case 2: Fixed target, randomly moving obstacle.

The position of the target point is the same for every episode, and the obstacle moves randomly along the path: the sphere can change direction at any time or stop for an interval of time. The target position p^* to be reached is placed at the center of the production line, at an initial distance of 60 cm from the end-effector. Results regarding training and testing of the last episode are reported in Figure 9.5.

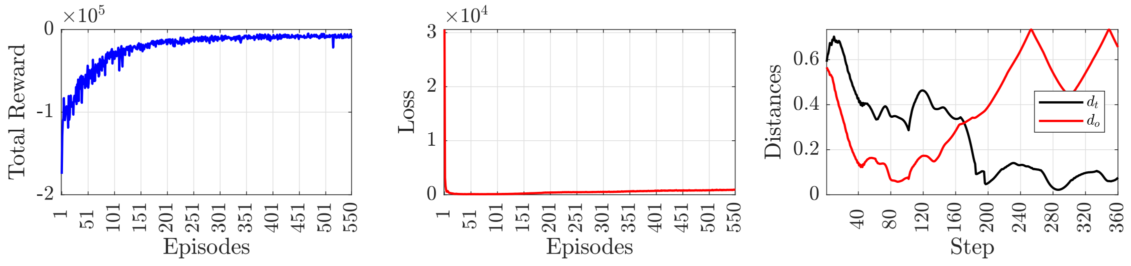


Figure 9.5: Results for Case 2

Case 3: Random target, moving obstacle.

The position of the target point is randomly initialized at the beginning of every episode, and the obstacle moves back and forth in a deterministic way. The position p^* of the target point to be reached is chosen with a uniform distribution spacing the dimensions of the production line. Results regarding training and testing of the last episode are reported in Figure 9.6.

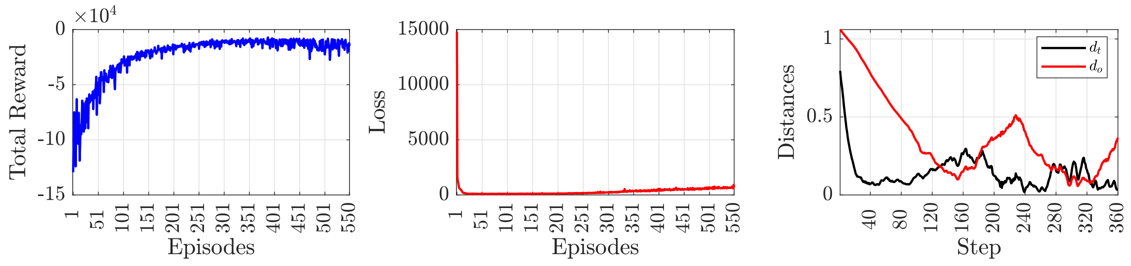


Figure 9.6: Results for Case 3

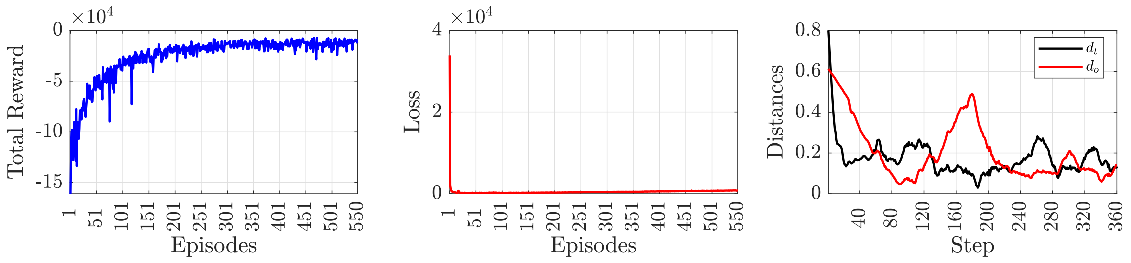


Figure 9.7: Results for Case 4

Case 4: Random target, randomly moving obstacle.

The position of the target point is randomly initialized at the beginning of every episode, and the obstacle moves randomly along the path. Results regarding training and testing of the last episode are reported in Figure 9.7.

9.4.1 Results

As it can be observed in Figures 9.4 - 9.7, in all the experiments the overall process does not present abrupt variations in the reward values and converges within acceptable time limits (around 24 hours per run), on the hardware and software configuration described above. This means that the agent has learned a way to complete the task as defined by the reward function efficiently. This also applies to the trend of the average loss function, which presents little variations and steers towards zero towards the end of the exploration phase, meaning the training process is effective. Furthermore, as the distance from the obstacle diminishes, the robot adjusts its position with respect to the target, moving away further until the obstacle is again at a safe distance. The distance between the tip and the target point diminishes as the robot approaches the target and maintains its position until the sphere becomes in close proximity to the body of the robot. Then, the robot backs off from the target until the obstacle starts moving further away from the manipulator. The same complementary behavior of the two distance graphs has been obtained in both the experiments with the deterministic moving obstacle and random moving obstacle (Figures 9.4 and 9.6), and in the scenarios with the randomly moving obstacle (Figures 9.5 and 9.7). Quantitative results on both the reward values of the different scenarios and the performance in terms of tracking error are reported in Table 9.2. Values for the reward R are computed as the mean

	R	RMS(d_t)
Case 1	-6.0566×10^3	0.1217
Case 2	-8.9794×10^3	0.1567
Case 3	-1.2454×10^4	0.1766
Case 4	-1.3873×10^4	0.1908

Table 9.2: Values for average converged reward function and tracking error.

of the reward function after convergence (i.e., after 300 episodes), while the tracking error is computed as the root mean square values of the distance d_t between the end-effector and the target point p^*. Except for Case 1, in which all elements in the scene maintain a deterministic behavior, values for other cases refer to the averaged results of 30 simulations of the trained agent with different target positions and/or obstacle movements. For the considered cases, no collision has occurred.

Remark. Seemingly high values of root mean square values of the distance from the target are justified by the fact that, during the episode, the robot moves away from the target every time the obstacles invade its operation space, thus increasing the distance between the end-effector and the target position.

It is also noteworthy to monitor the behavior of the agent during the training process. Making reference to Case 4, the training has been stopped every 10 episodes (i.e., the checkpoint saving frequency) in order to test the policy obtained up to that point. Both the RMS value of d_t and the collision count (i.e., the number of time steps at which the measured minimum distance d_o equals 0) are recorded and averaged for 30 simulations of 360 steps each, randomly initialized. Results are represented in Figure 9.8. It can be observed that both values decrease steadily throughout the training process, coherently to what has been observed for the cumulative reward function. Some screenshots of the simulations are reported in Figure 9.10, where it can be seen that the robot moves away from the incoming obstacle before placing its end-effector back in the proximity of the target point.

Random initial configuration

Training for Case 2 and 4 (i.e., those presenting the random motion of the obstacle) have been additionally carried out introducing the random initialization

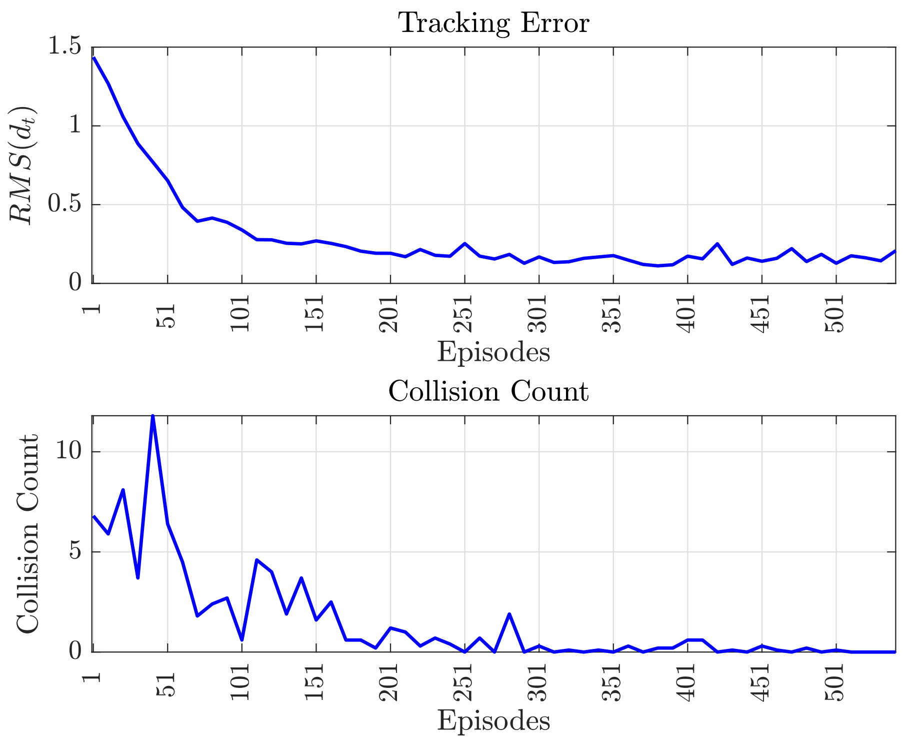
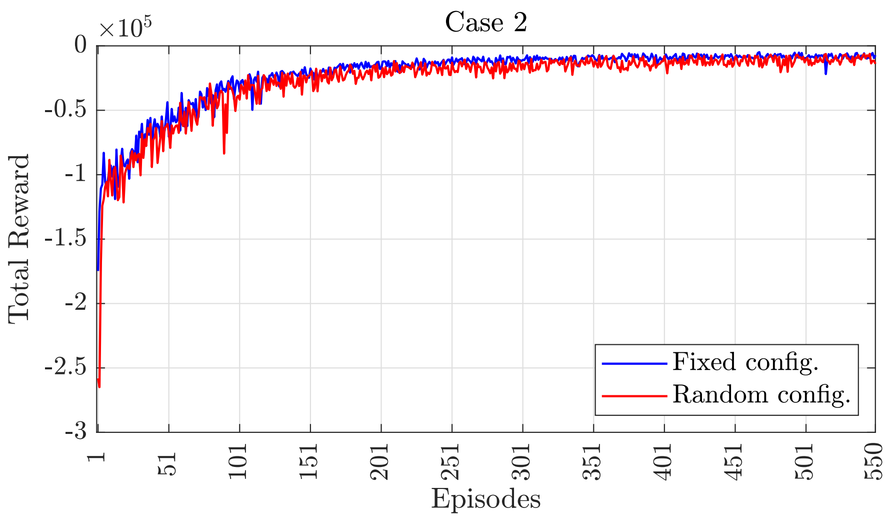


Figure 9.8: Evolution of performances in terms of tracking error (top) and collision count during training. Values are averaged over 30 test simulation every 10 episodes.

of the robot's configuration, contrary to previously described simulations in which the robot starts every episode in *home position* (i.e., each joint initialized at 0 rad). The initial joint position q_{i_0} is initialized by sampling with a uniform distribution over each joint's motion range, in order to try and explore as many configurations as possible.

Switching from a fixed initial configuration of the manipulator to a randomly perturbed one produces mixed results. In Figure 9.9 the blue line refers to trainings with fixed initial configuration, whereas red lines refer to randomly perturbed ones: as it can be seen, the growing trend remains satisfactory but the achieved level of effectiveness at convergence is not always superior to the former. In particular, for Case 2 the reward function using the randomly initiated robot is consistently lower throughout the training, with mean value at convergence equal to -1.244×10^4; for Case 4, perturbed configurations produce slightly better results at convergence, with mean value at -1.9788×10^4 but the overall process seems less stable and robust, with many negative peaks during the initial exploration phase. Nevertheless, starting the training with a randomized robot configuration helps increasing the



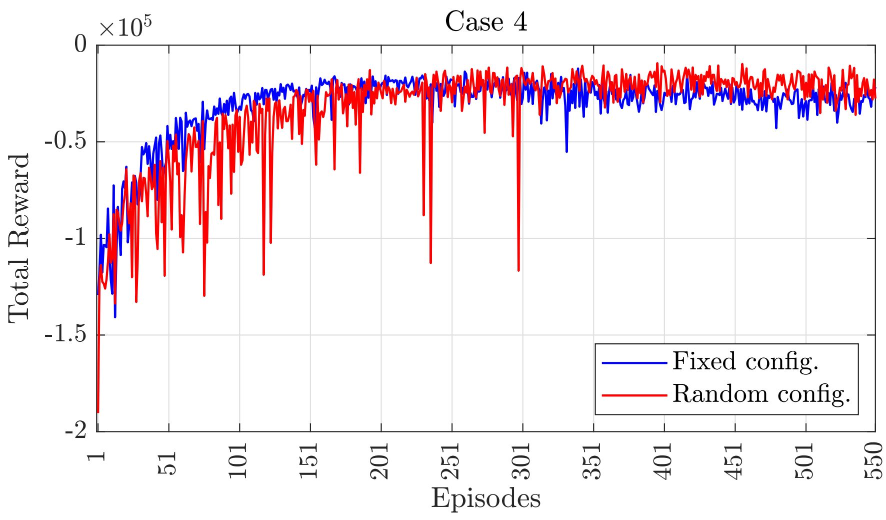


Figure 9.9: Reward functions comparison for Case 2 and Case 4 using for trainings with a fixed initial robot configuration and a randomized one.

generalization capabilities of the agent, exploring a higher variety of states.

9.5 Conclusions

In this Chapter, a Deep Reinforcement Learning approach has been proposed to solve collision avoidance problems in robotics. More specifically, the advantage-actor critic NAF algorithm has been used for training, due to its advantage of being a model-free and especially suitable for complex systems with continuous state and action spaces. The use of an *ad hoc* simulator has allowed to perform

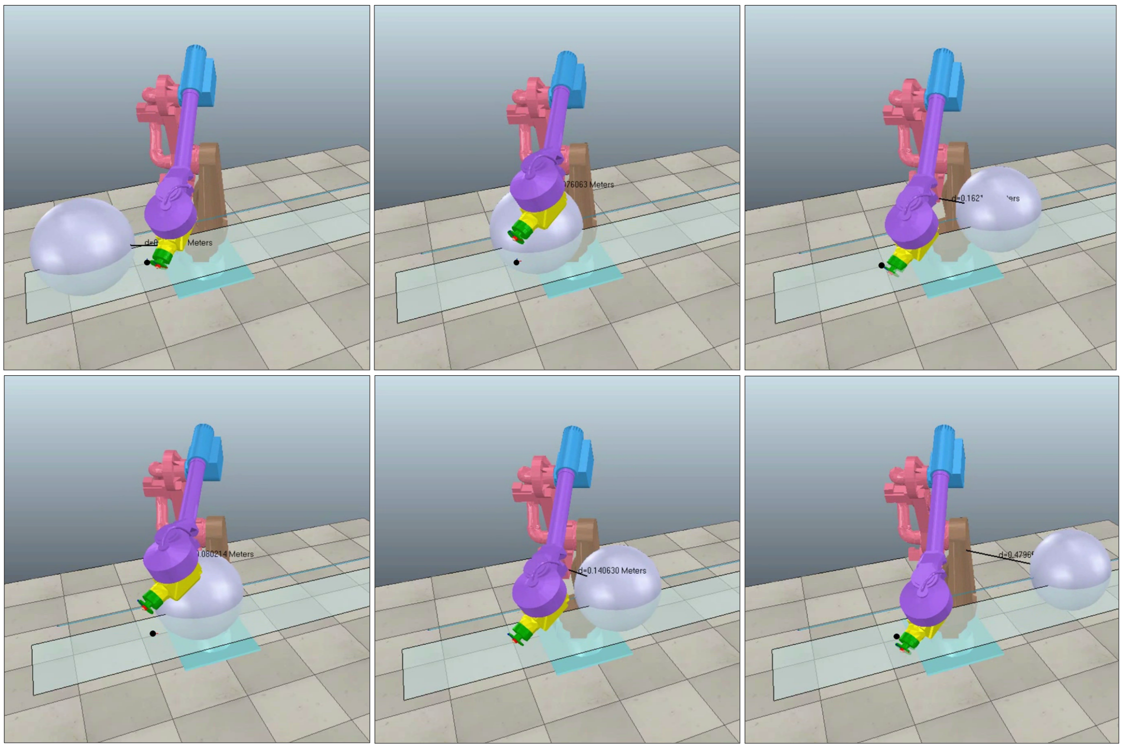


Figure 9.10: Screenshots from the simulation of the obtained policy in V-Rep.

long training sessions and evaluate the performance of the learning process under different conditions. Simulations performed on a realistic virtual environment show the effectiveness of the proposed DRL-based collision avoidance approach, proving that that the algorithm produces a successful learning process and is indeed suitable for systems with higher degrees of complexity.

Chapter 10

Transfer Learning for DRL-Based Collision Avoidance

The term *transfer learning* in Deep Reinforcement Learning refers to the possibility of reusing previously acquired knowledge in a particular training scenario for a different, possibly more complex, one. Making reference to the NAF algorithm employed for training the agent in the case studies illustrated in this Part, one can identify three different possibilities to perform transfer learning:

1. **Model transfer:** at the beginning of the training, the learned parameters θ^Q of the action-value function $\hat{Q}$, obtained at the end of a different training, are loaded as initializers for the action-value function in the subsequent training activity. Recalling Algorithm 2 in Chapter 3, this operation happens at the beginning of the training (line 1).

2. **Experience transfer**: the set of quadruplets $\{x_t, u_t, r_t, x_{t+1}\}$ collected during a previous training are used as initializers for the *replay buffer*, which collects all the samples produced by the training process throughout the episodes. Recalling Algorithm 2, this operation happens before starting the first iteration of the algorithm (line 3).

3. **Model and experience transfer**: both of the above procedures at the same time.

Indeed, being able to use the information obtained by training an agent in a simpler scenario as a base for facilitating training in scenarios with higher complexity would be beneficial to expand the applications of such end-to-end methodologies. In this Chapter, some considerations on this approach are made by comparing a set of trainings performed under different conditions. Specifically, trainings with transfer learning for the same scenario are compared with the results of the training conducted *ex-novo* (i.e., with no pre-loaded initializers) in terms of cumulative reward function and performance, measured with the RMS value of d_t, as introduced in the previous Chapter. Specifically, the percentage variation of both values is computed as:

$$\Delta R_{\%} = 100 \times \frac{R_{ex-novo} - R_{transfer}}{R_{ex-novo}} \tag{10.1}$$

where $R_{ex-novo}$ and $R_{transfer}$ refer to the mean values of the obtained cumulative function after convergence, i.e., from episode 250 to 550, and

$$\Delta d_{t_{\%}} = 100 \times \frac{d_{t_{ex-novo}} - d_{t_{transfer}}}{d_{t_{ex-novo}}} \tag{10.2}$$

where again $d_{t_{ex-novo}}$ and $d_{t_{transfer}}$ refer to the mean values of the averaged values of RMS(d_t) from episode 250 to 550. Note that, in this case, a positive variation means that, on average, the policy obtains better results (i.e., a lower value of RMS(d_t) is measured, meaning that the end-effector tends to stay closer to the target point).

10.1 Transfer learning for improved performances

Making reference to the case studies introduced in Section 9.4, in the following subsections the effects of different types of transfer learning on the performance of a training process, both in terms of cumulative reward and tracking precision, are presented and discussed. To this end, let us consider a scenario involving the obstacle moving with a fixed motion (specifically, Case 3) and the one involving, instead, the randomly moving obstacle (Case 4). Indeed, Case 3 presents a more predictable environment for the agent to train into, and it is of interest to show if an agent able to avoid an obstacle moving in an expected manner can improve the behavior of an agent that needs to ensure collision avoidance of obstacles moving unpredictably. Results are discussed for all cases and summarised in Table 10.1.

Model Transfer

The parameters of the DNN $\hat{Q}(x, u|\theta_{PRED})$, approximating the Q-function, are initialized as the ones obtained at the end of the training for Case 3, instead of randomly, before training the agent to perform the task in Case 4. The resulting reward function, in comparison with the one obtained with an *ex-novo* training, is reported in Figure 10.2 together with the evolution of the reaching performance and collision count.

As it can be observed, model transfer does not bring beneficial results, both in terms of cumulative reward R, in which an average variation $\Delta R_{\%}$ of -276.28% at convergence is obtained, and tracking error, where a variation $\Delta d_{t_{\%}}$ of -106.36% is obtained for the reaching. Such level of performance degradation is believed to be caused by an overfitting of the DNN weights for a scenario in which the observations about the obstacle position during an episode do not change throughout the training, thus making it difficult to generalize for a randomly moving obstacle, especially in cases where the interaction scenarios are rather basic. Nevertheless, the problem might be overcome with scenarios with substantially different behavior of the environment. Training times remain consistent.

Experience Transfer

The content of the Replay Buffer RB is initialized with the quadruplets obtained at the end of the training for Case 3, instead it being empty, before training the agent to perform the task in Case 4. The resulting reward function, in comparison with the one obtained with an *ex-novo* training, is reported in Figure 10.3 together with the evolution of the reaching performance and collision count.

As opposed to what was achieved with model transfer, it can be observed that the learning process using experience transfer improves its performances, obtaining higher cumulative reward per episode with an average $\Delta R_{\%}$ value increased by 37.81% at convergence, and faster learning of the reaching task, that presents a 13.32% improvement $\Delta d_{t_{\%}}$ on the average values. Differently from the results of model transfer, the content of the Replay Buffer does not generate overfitting problems in terms of DNN parameter, instead constituting a starting dataset of observations that are then used during the learning process. Training time doubles due to the increased size of the Replay Buffer that entails higher computational costs. Nevertheless, satisfactory performances can be achieved in fewer episodes when using experience transfer, thus reducing the need for extended training time.

Model and Experience Transfer

Both the content of the Replay Buffer RB and the parameters of the DNN $\hat{Q}(x, u|\theta_{\text{PRED}})$ are initialized with those retrieved from the training of Case 3. The resulting reward function, in comparison with the one obtained with an *ex-novo* training, is reported in Figure 10.4 together with the evolution of the reaching performance and collision count.

Using both model transfer and experience transfer brings a variation $\Delta R_{\%}$ of -12.69% and a variation $\Delta d_{t_{\%}}$ of -26.46%. Given the slightly degraded results and double training times, due to the initialization of the Replay Buffer, this method has been discarded in all the following experiments.

Table 10.1: Values of average cumulative reward and average RMS of the target distance for different types of training. Variations with respect to the ex-novo training are computed according to (10.1) and (10.2).

Training		R		$RMS(d_t)$
Ex-novo		-1.387×10^4		0.1659
Model Transfer		-5.22×10^4		0.3424
	$\Delta R_{\%}$	**-276.28%**	$\Delta d_{t_{\%}}$	**-106.36%**
Experience Transfer		-8.627×10^3		0.143
	$\Delta R_{\%}$	**37.81%**	$\Delta d_{t_{\%}}$	**13.32%**
Both		-1.5635×10^4		0.2098
	$\Delta R_{\%}$	**-12.69%**	$\Delta d_{t_{\%}}$	**-26.46%**

10.2 Transfer learning for scalability

The following subsections illustrate the use of transfer learning for training the agent to accomplish tasks in scenarios featuring environment behavior that was never explored in previous training. It is observed that reusing the knowledge

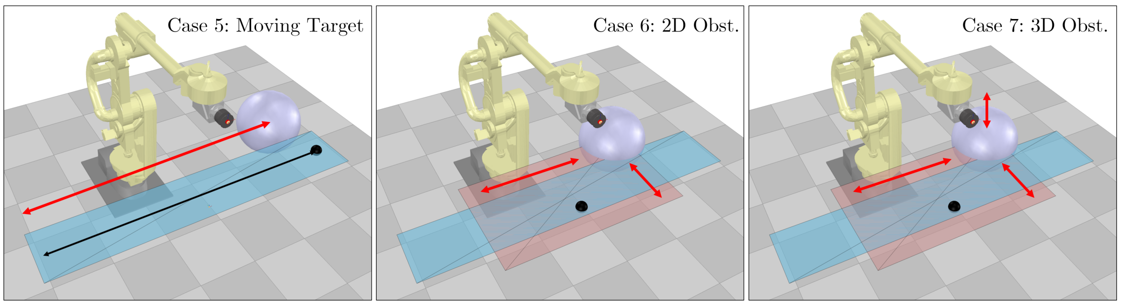


Figure 10.1: V-Rep scenarios for experiments on transfer learning. The black line in Case 5 represents the movement of the target. The red planes in Case 6 and case 7 represent the area in which the obstacle moves randomly.

acquired to complete simpler tasks facilitates and improves the learning of more complex, new tasks. Comparison is made in terms of cumulative reward functions over episodes between model transfer, experience transfer, and ex-novo training of the agent. Three case studies are introduced, represented in Figure 10.1, and detailed in the following. Again, average reward values for comparison are computed at convergence, i.e., between episode 250 and episode 550. Results are summarized in Table 10.2.

Case 5: Linear moving obstacle, moving target

Transfer learning is performed from Case 3 (random target, random 1D obstacle movement) to a scenario in which the target is moving at a fixed speed of 0.15 m/s along a linear path (i.e., as if it was positioned on a conveyor belt), while the obstacle moves randomly along a linear path (moving target, 1D movement). The evolution of the cumulative reward functions is reported in Figure 10.5.

Experience transfer is beneficial and produces a 29.35% improvement, while model transfer is apparently not beneficial at all, as it degrades the performances by 127.09%.

Case 6: Planar moving obstacle, random target

Transfer learning is performed from Case 2 (fixed target, random 1D obstacle movement) to a scenario in which the target is kept fixed in a randomly initiated position, but the obstacle now moves in two directions, on the same plane where the target is placed (random target, 2D movement). The area in which the obstacle

Table 10.2: Effects of transfer learning on cumulative reward functions in case of increased complexity in scenarios.

Scenario	Ex-novo	Model	Experience
Case 5	-1.24×10^4	-2.83×10^4	-8.85×10^3
	$\Delta R_{\%}$	-127.09%	29.35%
Case 6	-2.52×10^4	-1.65×10^4	-9.05×10^3
	$\Delta R_{\%}$	34.56%	64.13%
Case 7	-3.95×10^4	-5.05×10^4	-1.184×10^4
	$\Delta R_{\%}$	-27.691%	70.054%

moves is 1.3 m wide and 0.75 m long, overlapping the production line. The evolution of the cumulative reward functions is reported in Figure 10.6.

Differently from what was observed so far, both transfer learning modes are beneficial with respect to ex-novo training; experience transfer is substantially more effective since it yields an improvement ($\Delta R_{\%}$) of 64.13%, versus the 34.56% improvement achieved with model transfer.

Case 7: Spatial moving obstacle, fixed target

Transfer learning is performed from Case 2 (fixed target, random 1D obstacle movement) to a scenario in which the target is kept fixed in a randomly initiated position, but the obstacle moves in three directions, on the same plane where the target is fixed (random target, 3D movement). The evolution of the cumulative reward functions is reported in Figure 10.7.

Similarly to Case 5, again, experience transfer learning results in higher reward values at convergence, while model transfer degrades performance; experience transfer yields an improvement ($\Delta R_{\%}$) of 70.054%, while model transfer brings to a degradation of 27.691%.

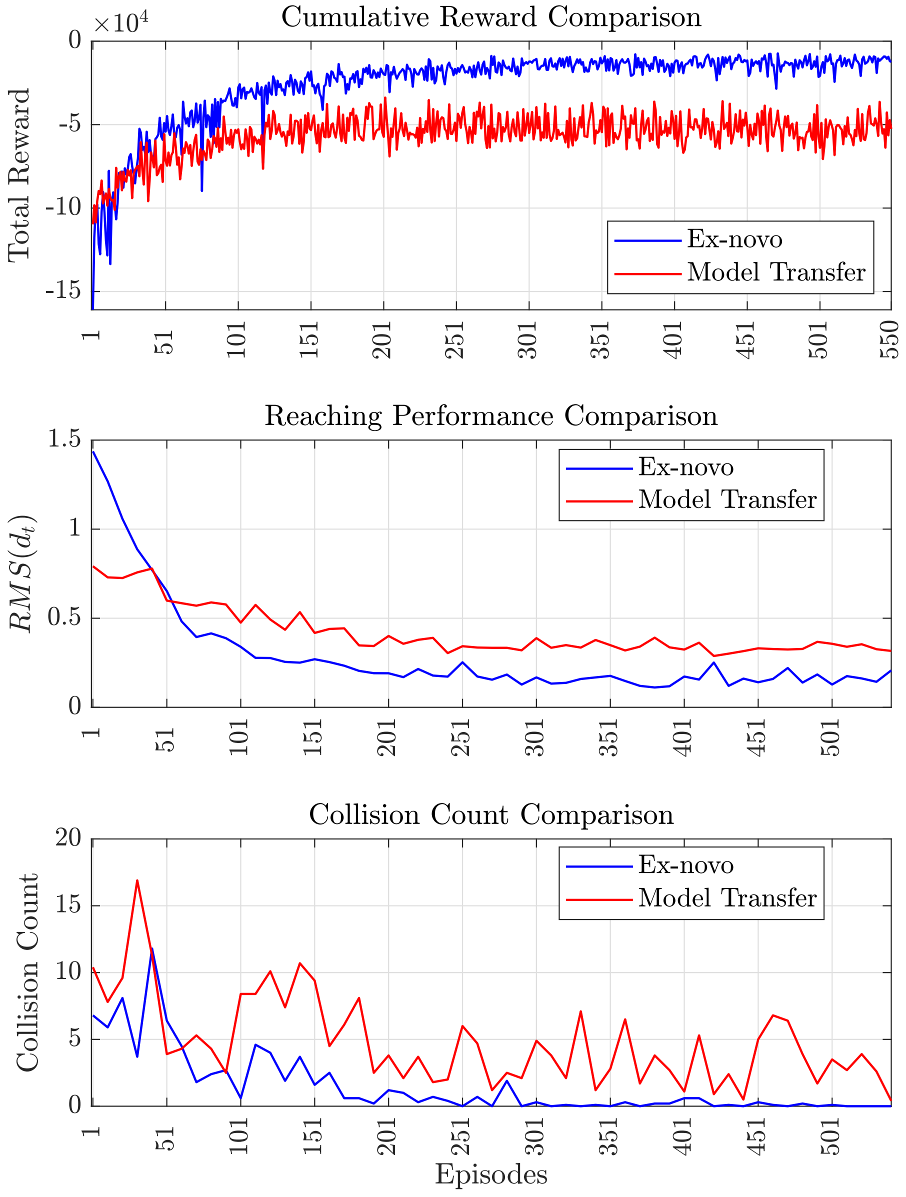


Figure 10.2: Comparison of reward functions, reaching performance, and collision count over episodes between ex-novo training and **model transfer** training.

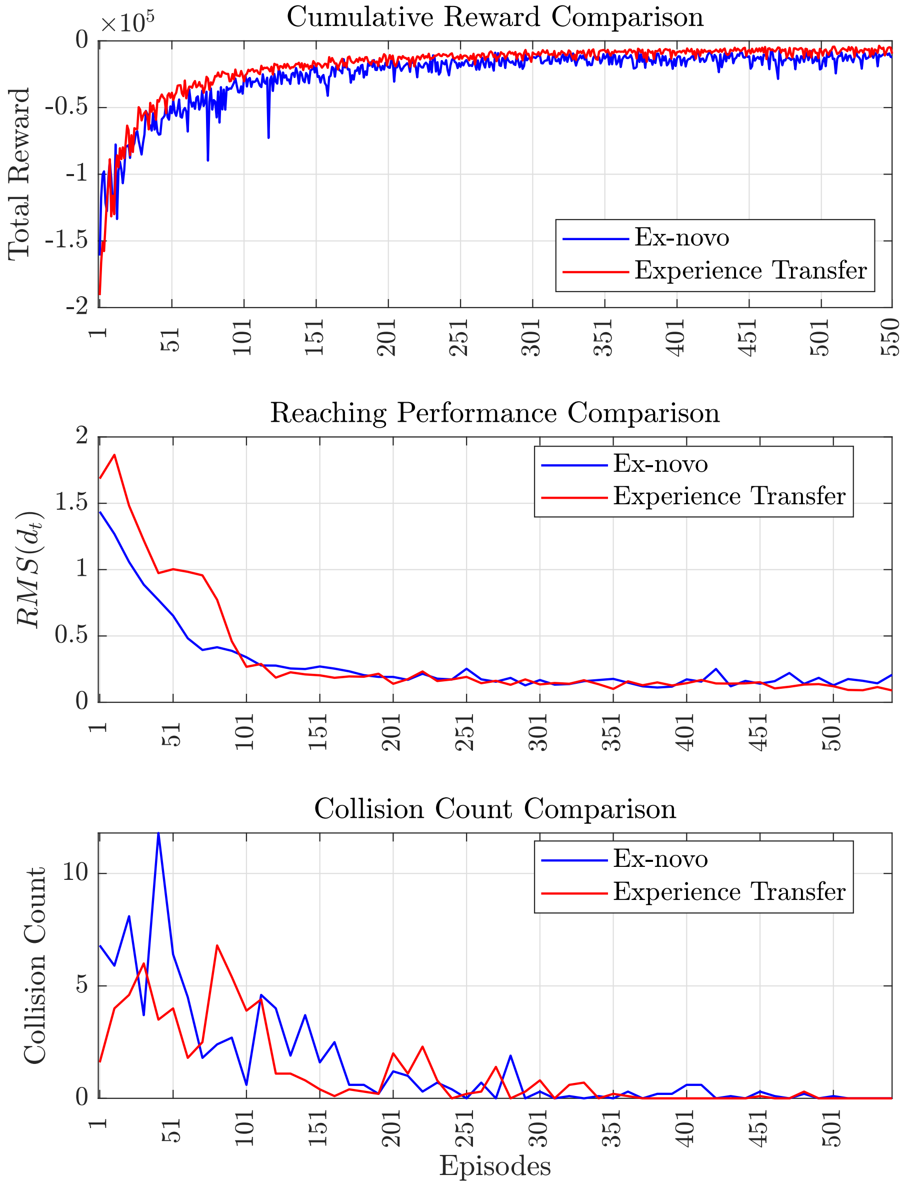


Figure 10.3: Comparison of reward functions, reaching performance, and collision count over episodes between ex-novo training and **experience transfer** training.

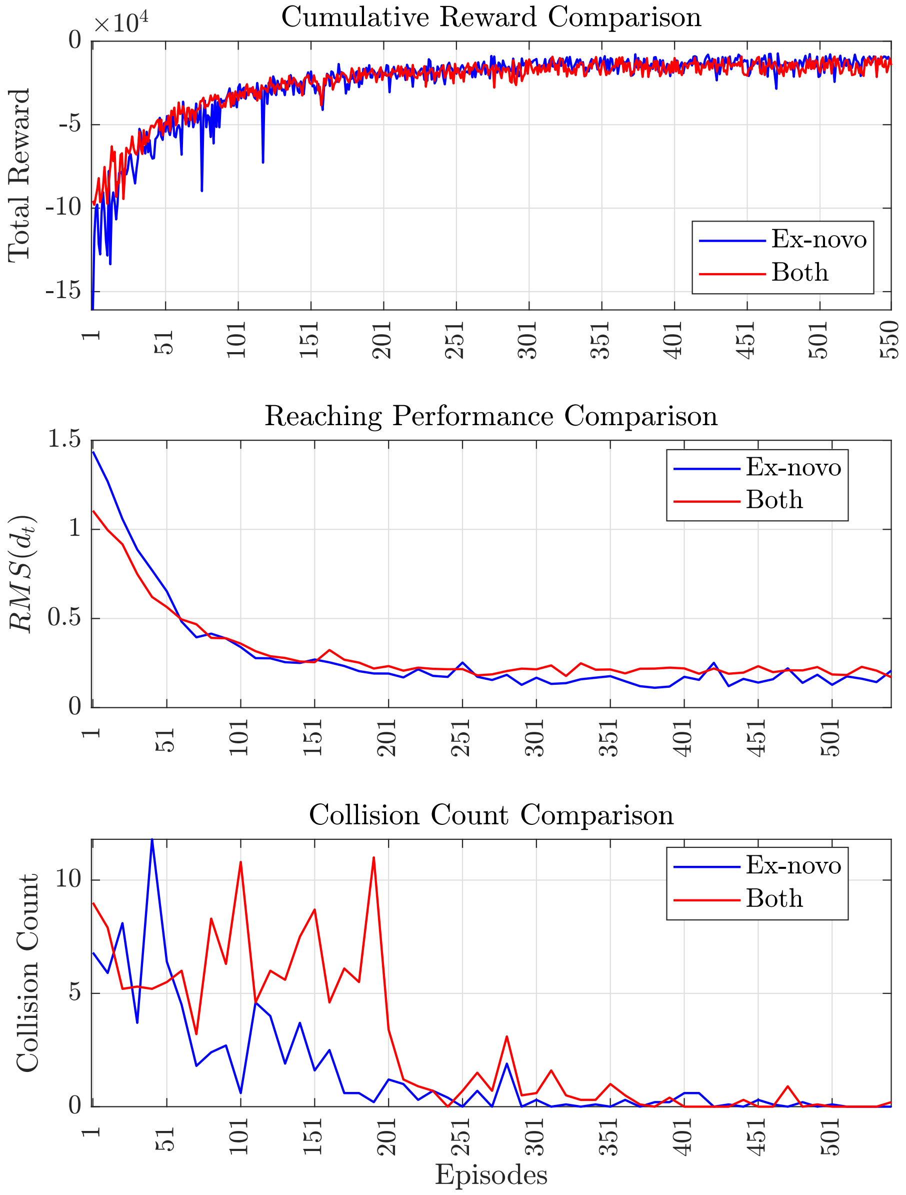


Figure 10.4: Comparison of reward functions, reaching performance, and collision count over episodes between ex-novo training and **both types of transfer** training.

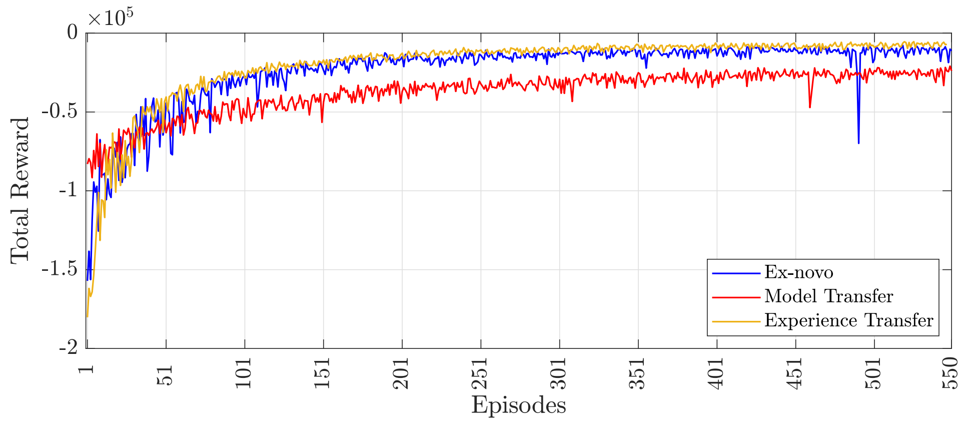


Figure 10.5: Comparison of reward functions between different types of training for Case 5

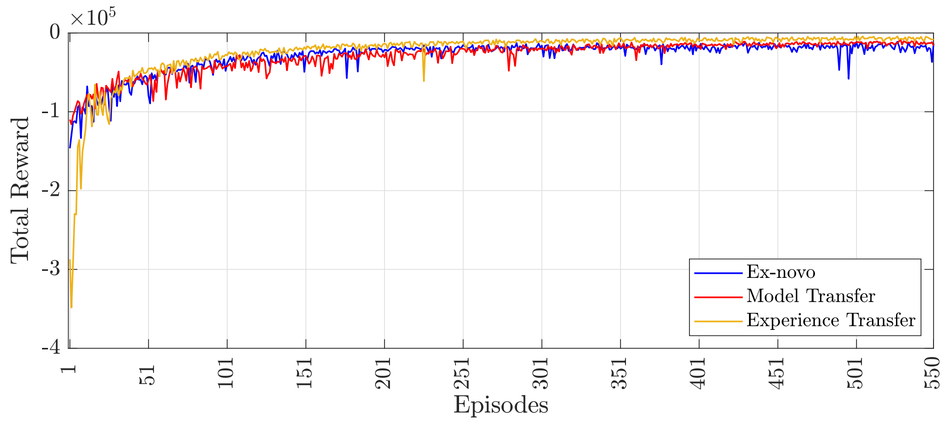


Figure 10.6: Comparison of reward functions between different types of training for Case 6

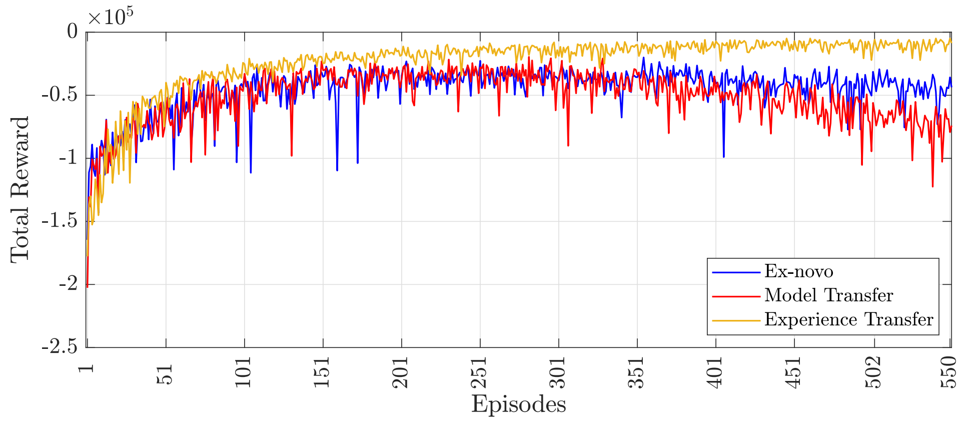


Figure 10.7: Comparison of reward functions between different types of training for Case 7

Chapter 11

Self-configuring Motion Planning and Obstacle Avoidance

After having introduced the Deep Reinforcement Learning framework for achieving full-body collision avoidance of robot manipulators (Chapter 9), as well as the possible ways to facilitate the training through transfer learning (Chapter 10), the goal of the present Chapter is to present a hybrid dual-mode architecture, which enables the combined use of motion planning and end-to-end control strategies. Indeed, it has been observed with the training sessions that, given enough episodes, the agent is able to find a policy that accomplishes the dual task of (i) reaching a point in space and (ii) avoiding an obstacle invading the robot's workspace. Nevertheless, although the avoidance procedure seems to produce very satisfactory results, the performance of the target reaching task is suboptimal with respect to conventional control methodologies, especially when the robot has to maintain its position for prolonged time: even with good performance indexes, the behavior of the robot is constantly influenced by the surrounding environment, resulting in a 'jittering' of the end-effector and overall unsteady motion. With the proposed methodology, the intention is to reduce the level of uncertainties that come with end-to-end approaches. To this end, a conventional motion planning algorithm is used to compute the reference trajectory to move the end-effector on the desired target point, without the burden of taking obstacles into account. On the other hand, using a policy obtained through DRL to perform the obstacle avoidance removes the need for model-based approaches to ensure that the robot's body does not collide with obstacles, which are hard to hand-engineer. This is convenient

Algorithm 3 Hybrid motion planning and obstacle avoidance algorithm

```
Input: threshold $\epsilon$, current joint positions $q$, target pose $x^*$
Output: a collision-free motion reference for (9.2)
 1: repeat
 2:     compute $x_\mathrm{e}$ as $x_\mathrm{e} = k(q)$
 3:     compute the metric $\mathrm{m}(d_\mathrm{o})$
 4:     if $\mathrm{m}(d_\mathrm{o}) < \epsilon$ then
 5:         use DRL and pose $\dot{q}^* = u_t$, as in (9.5)
 6:     else
 7:         let $x_{\mathrm{e}0} = x_\mathrm{e}$
 8:         use SBL starting from the initial condition $x_{\mathrm{e}0}$ to the final pose $x^*$
 9:         let $(q^*, \dot{q}^*) = (q^*, \dot{q}^*)^\diamond$
10:     end
11: until $x_\mathrm{e} \neq x^*$
12: return $(q^*, \dot{q}^*)$ (SBL) or $\dot{q}^*$ (DRL)
```

when obstacles are not known *a-priori* or we do not have an accurate knowledge of the system's dynamics. The use of either approach is determined by a given metric, which is evaluated at each time step and defines the condition of the switch.

11.1 Hybrid Dual-Mode Strategy

Let us consider the scenario initially introduced in Chapter 9, in which the robot (i.e., the Comau Smart3-S2) coexists with an obstacle invading its operative space. The main objective of the proposed strategy, presented in this section and summarized in Algorithm 3, aims to perform the following motion tasks:

T1) to reach a pre-specified target;

T2) to avoid unexpected obstacles invading the workspace.

To do so, starting from an initial configuration, the motion planner finds a trajectory that connects the initial pose $x_{\mathrm{e}0}$ of the end-effector with the pose of the target point x^*. During the execution of the motion solved by the motion planner, if the metric chosen to evaluate the risk of collision with the obstacles, namely $\mathrm{m}(d_\mathrm{o})$, is below a fixed threshold ϵ, suitably selected by the designer for the sake of safety, the system hands over the control to an appropriately selected DRL policy. Once this condition is no longer satisfied, i.e., the obstacles, which can have different shape and behavior, are considered at a safe distance, the motion planner is once again initialized with a starting point as the current end-effector's pose x_{e}. In this case, the obvious choice of the switching metric adopted is the measured minimum distance between the robot and the obstacle. In case of multiple obstacles, the minimum among all of the measured distances is selected. Furthermore, a short time transient is imposed between one switch and another to prevent high-frequency switching.

11.2 Motion Planning

Let us concentrate on a general tracking problem in which we assume that the robot moves in the free-motion operative space. The goal is to track a smooth reference trajectory x_{e}^*, starting from $x_{\mathrm{e}0}$ towards the target point at x^*. Relying on classical approaches, such as those in [26] and as recalled in Section 2.4, motion planning constitutes a crucial aspect, and, in order to find a collision-free trajectory, it can be performed in two ways: (i) in the configuration space, by defining motion independently for each joint from an initial configuration q_0 to a final one q^* or (ii) in the operative space, by determining x_{e}^*. Although planning trajectories in the joint space is simpler and computationally lighter, planning trajectories in the operative space provides a more natural task description and obstacles can be accounted in the design phase of the path. Nevertheless, if a collision can occur or the environment is especially cluttered, it entails high computational costs due to the online calculation of the inverse kinematics to take into account the obstacles position.

In the hybrid approach, motion planning from the initial pose of the end-effector to the target point in the operational space is done accordingly to the Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking (SBL) algorithm [78], i.e., a sample based planning algorithm that relies on the construction of explorable trees in the configuration space. SBL is computationally light and is prone to find the shortest path [79]. Making reference to the selected case study, let us assume that the obstacle is randomly placed in the space where the desired target to be reached also lies, and does not move (i.e., it is a static obstacle): in Figure 11.1, the computational time of the planning algorithm [78] is measured for cases in which the obstacle presence for trajectory generation is considered and cases where it is not. The time is computed as the average of 100 simulations with random obstacle and target positions, returning 0.424 s and 0.008 s, respectively.

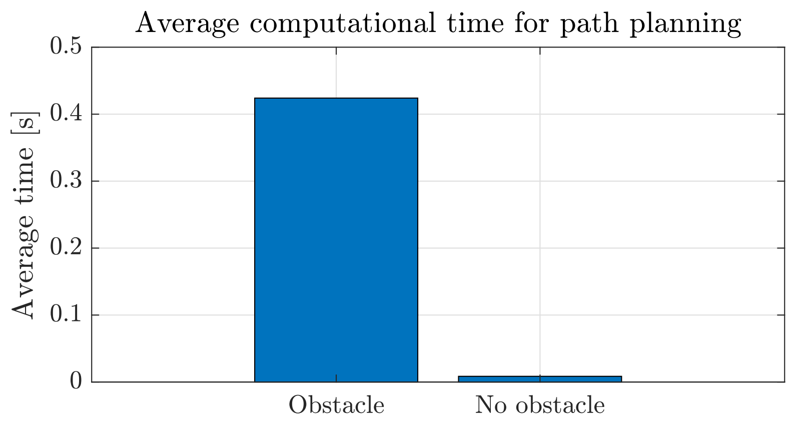


Figure 11.1: Comparison of computational times of the motion planning algorithm with and without an obstacle placed in the operative space

As expected, significant improvements in computational times are achieved if the presence of obstacles is not considered. For this reason, in the proposed approach, the SBL path $(q^*, \dot{q}^*)^\diamond$ is computed without accounting for the presence of obstacles in the operative space, significantly speeding up calculations and reducing cases where the planner does not find a feasible trajectory. The rationale behind this choice is that any interference of obstacles with the robot's motion would be handled directly by the end-to-end control strategy from the policy trained with DRL, thus removing the need to take the presence of obstacles into account a-priori. For the sake of simplicity and without loss of generality, it is assumed from here on that local joints controllers are capable to perfectly track the desired joint path, namely q^*, as well as its derivative $\dot{q}^*(t)$, that is $q(t) = q^*(t)$ and $\dot{q}(t) = \dot{q}^*(t)$, $\forall\, t \geq 0$.

Remark. Note that we assume that the internal control is ideal. However, since the proposed learning approach is exclusively based on data from sensors, independently of the inner control law, a certain degree of robustness is guaranteed in case of reasonable control errors.

11.3 End-to-end control

Collision avoidance is performed using the policy obtained by a suitably trained agent. The framework used for training the agent is the one illustrated in the previous Chapters 9 and 10. Hence, at each step, given the observed variables x_t as in (9.4), the policy produces the target velocities $\dot{q}^*$ for each joint in the next time step as actions u_t, according to (9.5), thus performing end-to-end control at the joints from the observed variables.

11.4 Case studies

The robot must complete task T1, with random initial conditions. Throughout its operation, the robot must perform also task T2, with randomly moving obstacles, with different shapes and behavior. Applications of the algorithm have been performed on the following scenarios:

A. Single obstacle, planar motion.

B. Single obstacle, spatial motion.

C. Multiple obstacles of different shape, planar motion.

The metric adopted for switching between the policy and planning with SBL is chosen as $\mathrm{m} = \min\{d_{\mathrm{o}_i}\}$, with threshold $\epsilon = 0.16$ chosen heuristically.

The proposed approach is validated by testing 500 different target positions, spanning the workspace of interest, with random initial robot configuration and random initial obstacle position. This procedure is repeated 30 times, and an average of the selected performance indexes is computed. More specifically, the two performance indexes consist of

- the *failure rate* i_f computed as the percentage of timesteps over an episode in which the distance d_o between an obstacle and the robot is measured equal to zero (that is, a collision event occurs), i.e.,

$$i_\mathrm{f} = 100 \times \frac{\sum n.of collisions}{T} \tag{11.1}$$

  where T is the number of timesteps in an episode.

- the *reaching performance* i_{t} in order to measure the precision of the executed tracking task, computed as

$$i_{\mathrm{t}} = \mathrm{RMS}(d_t) \tag{11.2}$$

11.4.1 System specifications

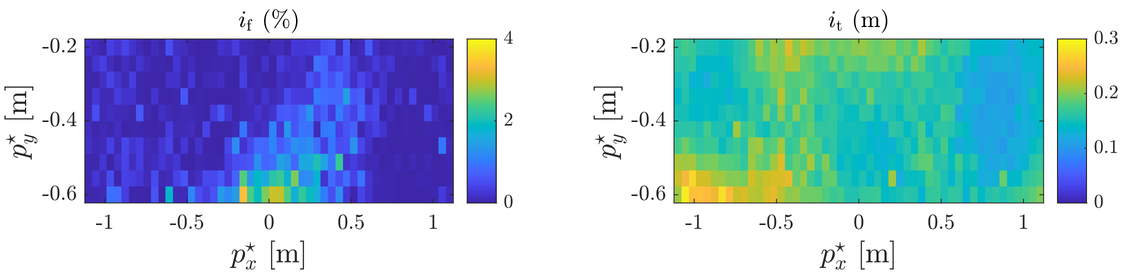
Experiments have been carried out in V-Rep [76], interfaced with TensorFlow, through the PyRep plugin [77]. The dynamics of the manipulator are handled by the ODE Physics engine running with a frequency of 200 Hz. The motion planning is handled by the Reflexxes Type II motion library, integrated within the simulator. Minimum distances measurements $d_{\mathrm{o}i}, i \in \mathbb{N}$ indicating the i-th obstacle, are also provided by the embedded calculation module of V-Rep. The simulation time step is equal to 50 ms.

11.4.2 Results

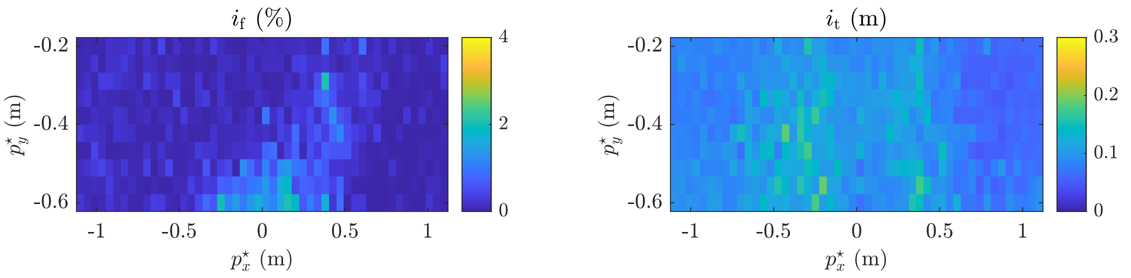
In the following, results from each case study are reported and discussed. A video showing the simulations of the hybrid approach is at the link: `http://bit.ly/CH11_HYBRID`.

Case A: Single obstacle, planar motion

Assuming a scenario in which a target is randomly placed, and the obstacle moves planarly, the selected policy to be deployed in the algorithm is the one achieved after performing experience transfer for Case 6, in Section 10.2. A graphic rendering of the indexes is reported in Figure 11.2 for both (a) full end-to-end approach (i.e., both reaching and avoidance are performed entirely using the DRL policy) and (b) the proposed hybrid strategy. In case of full end-to-end control, an overall average of 0.36% for the failure rate i_{f} is obtained, while as for i_{t} the average is 0.16 m. With the hybrid self-configuring approach, instead, it can be observed that the average failure rate i_{f} is 0.24%, while the average of i_{t} is 0.093 m, showing improvements with respect to the end-to-end approach.



(a) end-to-end approach



(b) hybrid approach

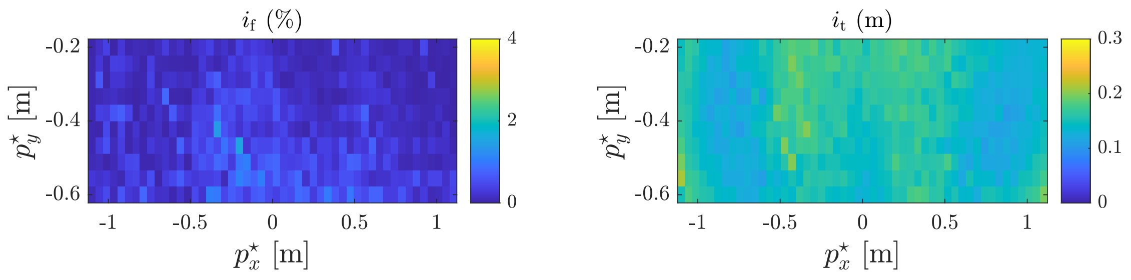
Figure 11.2: Indexes averaged for 30 simulations per target position of the trained policy with (a) end-to-end approach and (b) proposed hybrid approach for **Case A**

Case B: Single obstacle, spatial motion

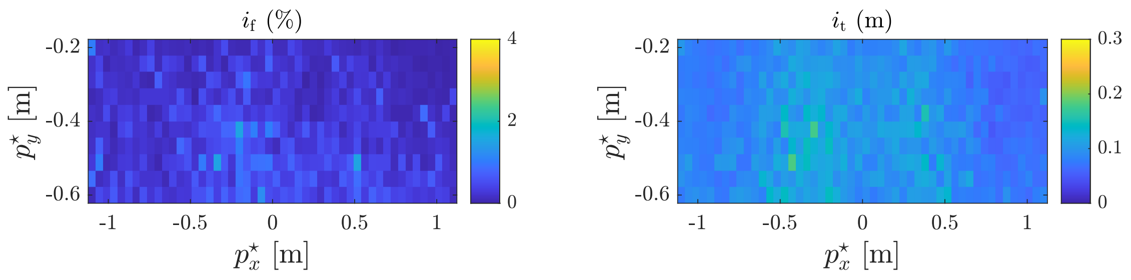
The target is randomly placed, and the obstacle moves in three directions (spatially) within the robot's workspace. The selected policy to be deployed in the algorithm is the one achieved after performing experience transfer for Case 7, in Section 10.2. Again, performance improves with respect to the end-to-end approach, which fares average values $i_\mathrm{f} = 0.28\%$ and $i_\mathrm{t} = 0.149$ m, while using the proposed approach presents average values $i_\mathrm{f} = 0.31\%$ and $i_\mathrm{t} = 0.091$ m, significantly improving the reaching precision while maintaining comparable values of collision avoidance.

Case C: Multiple obstacles, planar motion

The target is randomly placed, and two obstacles moving planarly at different heights are present in the robot's working space. The policy used for the end-to-end control has been obtained by training an agent ex-novo: because the state space has changed to accommodate a second obstacle, it has not been possible to reuse previously acquired knowledge. The possibility to enable transfer learning through



(a) end-to-end approach



(b) hybrid approach

Figure 11.3: Indexes averaged for 30 simulations per target position of the trained policy with (a) end-to-end approach and (b) proposed hybrid approach for **Case B**

different frameworks in this context will be the subject of future work. When using the end-to-end approach, performance indexes $i_\mathrm{f} = 1.13\%$ and $i_\mathrm{t} = 0.42$ m are obtained, compared with the hybrid approach that results in $i_\mathrm{f} = 1.35\%$ and $i_\mathrm{t} = 0.16$ m, showing significant improvements in terms of tracking, and again maintaining comparable results in terms of obstacle avoidance. This is believed to be due to the robot being less influenced by the obstacles' motion during the reaching phase, when the motion planner is activated, possibly causing occasional contacts before the system is able to switch control approach. Indeed, values for the safety threshold ϵ can be tuned accordingly to different cases, while in the presented cases, it was kept fixed for the sake of convenience. Nevertheless, the results are deemed satisfactory.

As an example, in Figure 11.5, the distance between the end-effector and the target point and distances between the robot and two obstacles are reported in case of genuine end-to-end DRL (a) and the proposal (b). As expected, when the adopted metric is below the threshold, the DRL policy is used; otherwise, the motion is solved by the SBL planner. As expected, differently from the end-to-end method, the hybrid approach ensures more stable movements during the reaching phase,

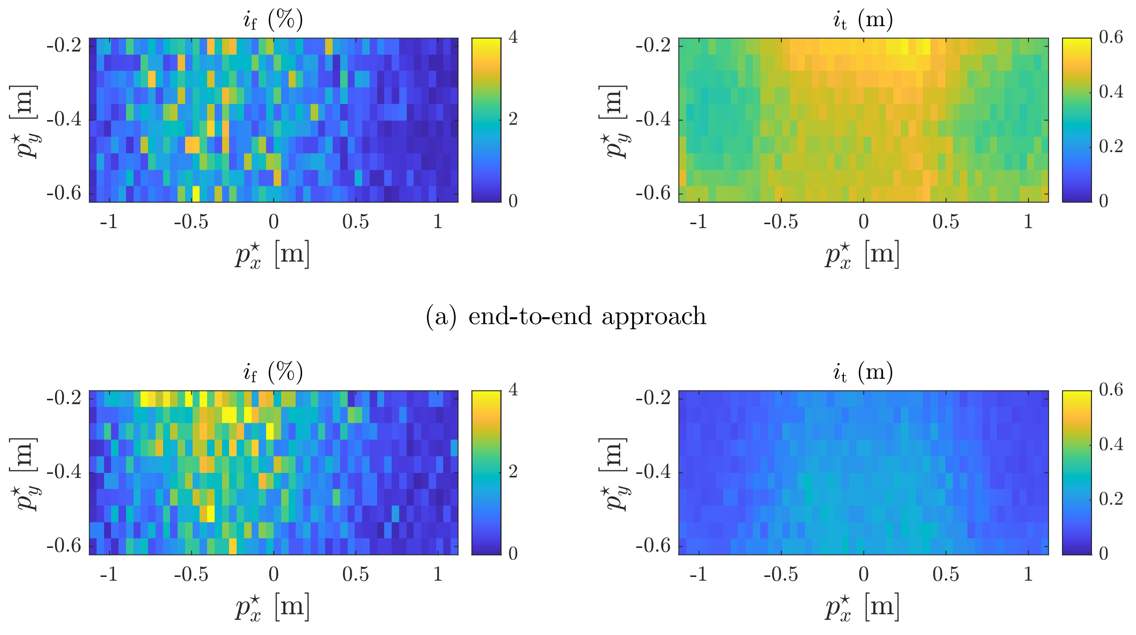


Figure 11.4: Indexes averaged for 30 simulations per target position of the trained policy with (a) end-to-end approach and (b) proposed hybrid approach for **Case C**

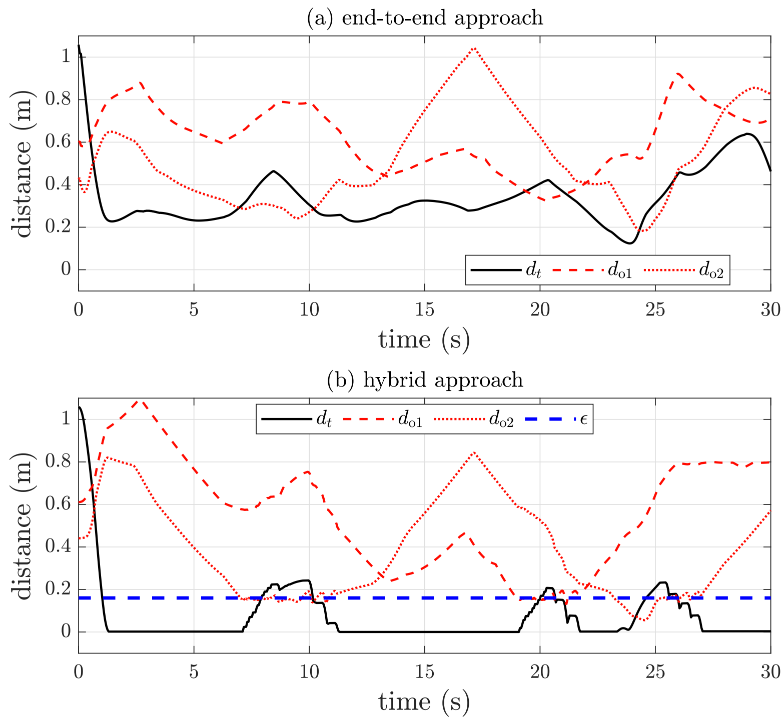


Figure 11.5: Distance from target d, distance from obstacles d_{oi}, $i = 1, 2$, and threshold ϵ, in case of end-to-end strategy (a), and of hybrid approach (b), with DRL used when $m = \min_i\{d_{oi}\} < \epsilon$)

maintaining the position of the end-effector precisely on the target and removing the ‘jittering’ effect that is present while maintaining position with end-to-end control.

Example application: spot welding

Finally, in order to show a possible application of the proposal in an industrial setting, the hybrid self-configuring algorithm has been tested on a spot welding task (Figure 11.6), in which the robot must follow a target gradually moving on a circular path (i.e., it advances by a portion of the path once the robot’s end-effector has reached it), and avoid collisions with two different obstacles of different shapes and behavior. Specifically, in Figure 11.7, the black line shows the motion of the robot’s end-effector, while the red and blue plots reproduce the obstacles’ activity.

11.4.3 Comparison with a model-based approach

The proposed hybrid algorithm has also been tested using the model-based Artificial Potential Fields (APF) approach in [64] for collision avoidance. Figure 11.8 shows that indeed, for the same scenario, both the DRL policy and the APF algorithm perform successful avoidance. Nevertheless, it is worth noticing that, while the model-based approach requires the knowledge of the dynamical model of the manipulator and intensive computation to perform the torque compensation derived from the artificial potential field, the proposal has the significant advantage of being model-free and virtually instantaneous output from the DRL strategy. Indeed, apart from the off-line training phase, it can be deployed in real-time, requiring only the available sensor measurements from the robot and the surrounding environment while providing velocity references to the robot joints, thus performing higher-level control with respect to joint torques.

11.5 Conclusions

In this Chapter, a dual-mode, hybrid control algorithm was proposed for robot manipulators to perform different tasks while avoiding collisions. The novel approach consists of a collision-unaware motion planner and a DRL policy trained to avoid obstacles by directly controlling the joint velocities. The most suitable operation

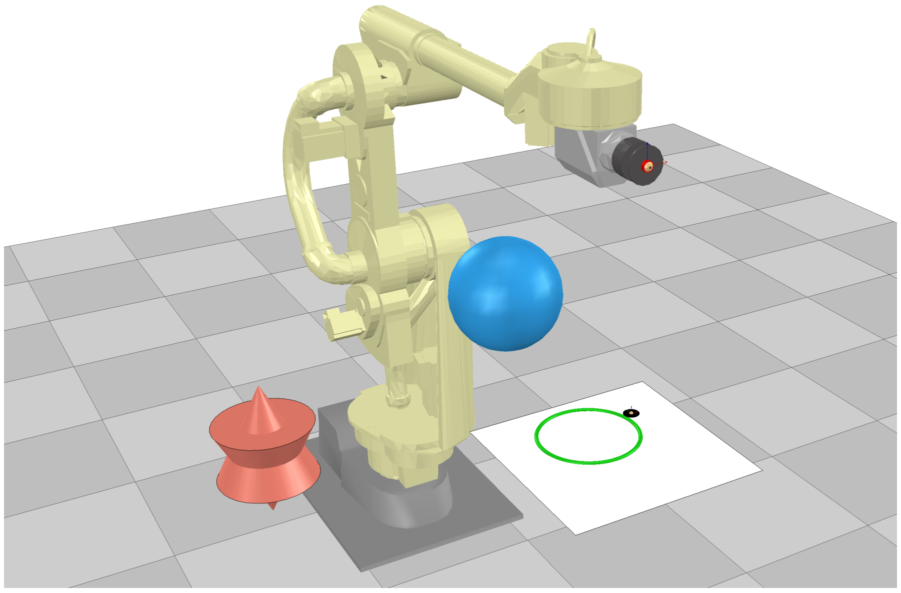


Figure 11.6: Virtual environment for the welding task

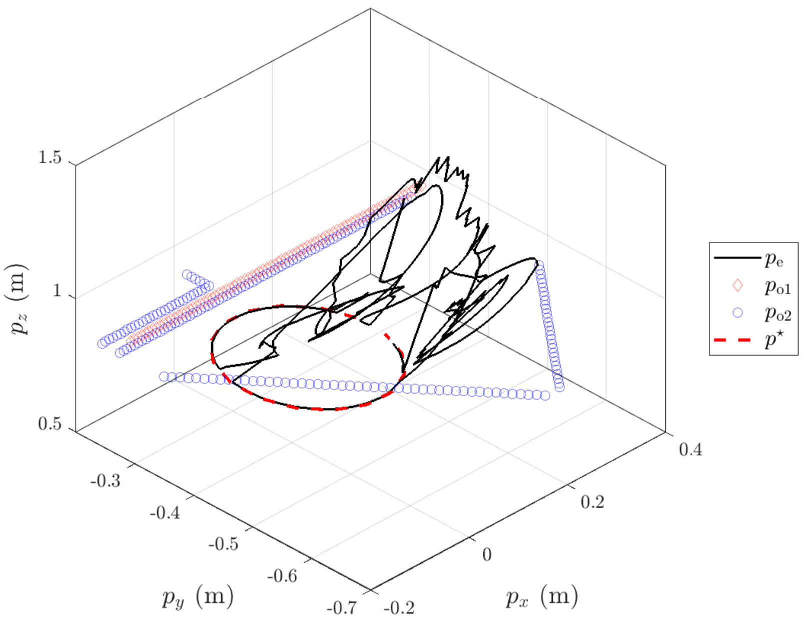
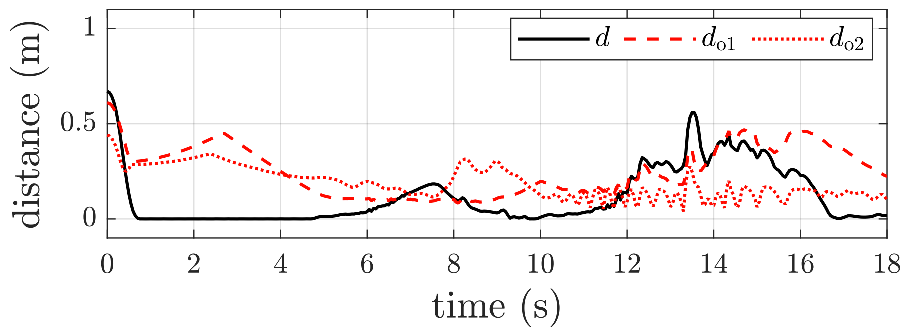
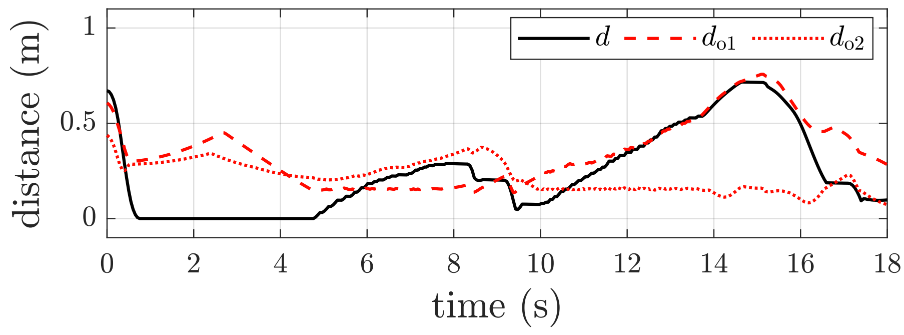


Figure 11.7: Hybrid approach deployed to perform a spot-welding task



(a) model-based strategy



(b) DRL-based strategy

Figure 11.8: Distance from target d, distance from obstacles d_{oi}, $i = 1, 2$, in case of (a) a model-based strategy (Artificial Potential Fields), and (b) a DRL-based strategy

mode is determined by a given metric, which confers to the whole system a self-configuring capability. The aim of this approach is to exploit the advantages of classical motion planning algorithms and those of end-to-end control based on DRL, thus limiting both the necessity of hand-engineering control algorithms and the risk of unexpected behaviors when using a trained policy. This concept could be indeed extended to solve different problems, where instead of relying entirely on end-to-end or model-based control, different modules implementing the most suitable approach can be assembled together.

Chapter 12

DRL for Teleoperated Robots

Robotic teleoperation [80, 81] constitutes an interaction paradigm between a human operator and a robot, with many potential applications [82, 83, 84]. The use of machine learning for the teleoperation of robot manipulators has been mainly focusing on imitation learning [85, 86, 87]. Deep Reinforcement Learning and teleoperation were combined in [88], in which complex dexterous manipulation tasks were learned by a robot manipulator by using human demonstration to accelerate the convergence rate of a DRL algorithm. In [89], an open-source framework is proposed for the control of robot manipulators based on state-of-the-art distributed reinforcement learning algorithms: among other features, the framework allowed the use of 3D motion devices to teleoperate the manipulators and collect human demonstrations. Furthermore, [90] proposed a DRL-based non-prehensile rearrangement strategy for rearranging objects on a tabletop surface, including obstacle avoidance.

This Chapter focuses on the application of the proposed DRL framework for end-to-end control of a teleoperated robot manipulator, in order to automatically avoid collisions with obstacles in the robot's operative space. The approach is compared with a model predictive control (MPC) approach proposed in [91], to solve the same problem. Results show (based on simulation and experimental data) that the DRL based end-to-end approach can provide faster computation times, together with improved performance.

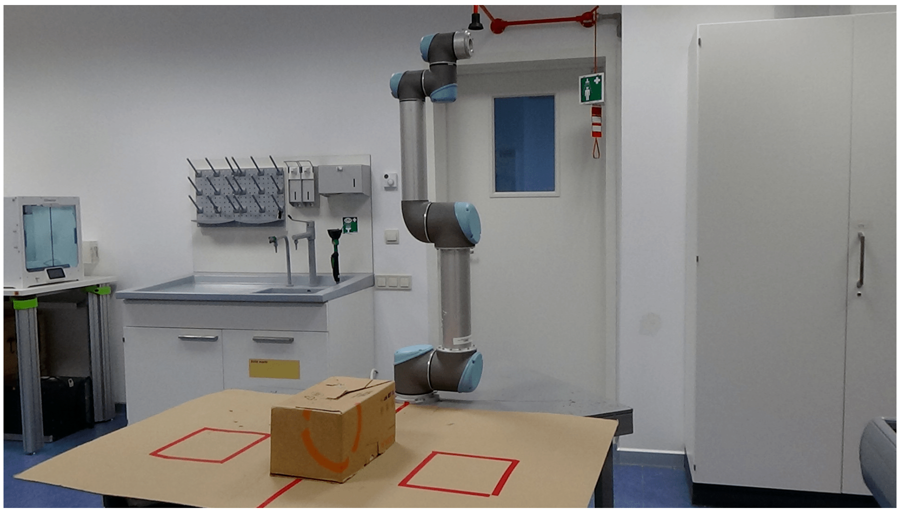


Figure 12.1: The experimental setup used in the case study, made of UR5 manipulator and obstacles

12.1 Problem definition

Let us consider a robot manipulator tasked with tracking, with its end-effector, a reference position generated online by an operator; while doing so, the robot has to avoid collisions with obstacles present in its operative space. Differently to what was proposed in the previous Chapters of this Part of the thesis, the reference provided to the robot is not fixed or pre-determined, but instead generated at run-time.

The operator's hand motion is converted to a suitable reference signal p^* for the robot via a custom-made 7-DOF human arm motion tracker system [92].

12.2 DRL Framework

For the proposed problem, some changes are applied to the previously introduced framework. The state space is defined as

$$\mathcal{X} \triangleq \{q,\, p_{\mathrm{e}},\, p^*,\, p_{\mathrm{o}}\} \ . \tag{12.1}$$

Differently to (9.4), the robot's joint velocities are no longer observed variables, as well as the obstacle's velocity. This is done to facilitate end-to-end control on the real robot, bypassing the problem of noisy velocity measurements. Again, the robot is controlled by providing joint velocity references, so the action space remains

$$\mathcal{U} \triangleq \{\dot{q}^*\} \ .$$

The reward function also is unchanged from the previously introduced framework in Eq. (9.6), here recalled for the sake of convenience:

$$r = -(c_1 R_{\mathrm{T}} + c_2 R_{\mathrm{A}} + c_3 R_{\mathrm{O}}) \ .$$

The agent is trained using the NAF algorithm introduced in [18] and recalled in Algorithm 2. The hyperparameters used during the training phase are summarized in Table 12.1.

Table 12.1: DRL hyperparameter values

Parameters	Value
number of time steps	360
time step	50 ms
c_1	1000
c_2	200
c_3	60
δ_T	0.1
g	8
$\delta_{\mathcal{O}}$	0.1
discount factor γ	0.99
update factor τ	0.001
learning rate	0.001
noise type $\mathcal{N}$	*Ornstein–Uhlenbeck*
noise decay factor	0.01
noise scale	1

12.2.1 System specifications

The training of the DRL agent is performed in simulation by interfacing the NAF algorithm, implemented in TensorFlow, with the V-Rep simulator encapsulating

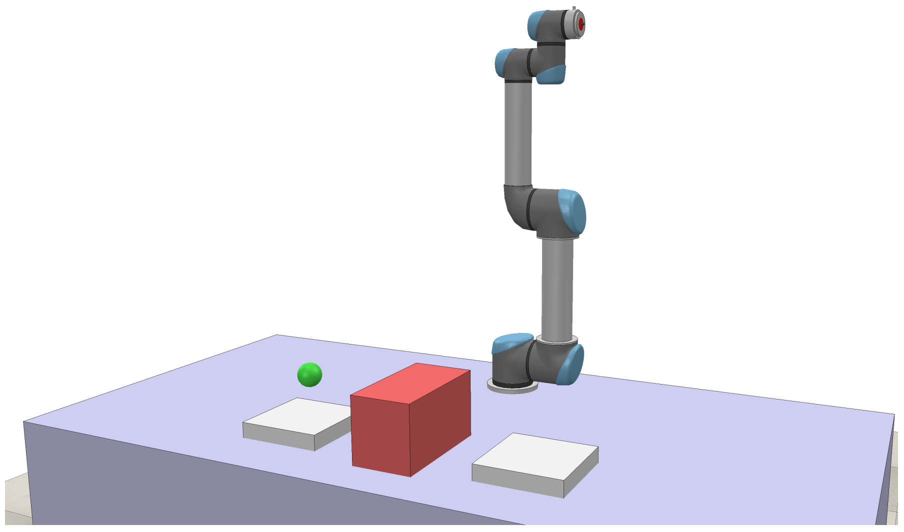


Figure 12.2: Virtualization of the experimental setup in V-Rep. The red box represents the obstacle placed on the table, while the green dot is a visual rendering of the moving point to be tracked. The white squares on the table only serve as a reference for the starting and finishing positions

the environment (Figure 12.2), via the PyRep [77] plugin. Synchronous simulations are run with a sampling interval $t_s = 50$. The dynamics of the simulation hare handled by the ODE physics engine running at a frequency of 200 Hz.

12.3 Case study

The considered scenario, represented in Figure 12.1, involves a UR5 6-axis robot manipulator, positioned on a table, that must track the provided trajectory. During operation, the robot must avoid collisions with a box placed on a table and the table itself. The trajectories used for training are sampled from the training set (consisting of a total of 447500 data points) at the beginning of each episode. The recorded motions consist of movements of the arm from left to right and vice-versa, while varying the height of the hand. Before starting, the reference idles for 5 s in order to allow the robot to reach the selected starting point of the randomly chosen trajectory segment. After each episode, both the trajectory and its starting point are reset, and the robot is re-initialized in its home configuration. The values of the reference position is such that $p^* \in [-0.57, +0.53] \times [-0.15, +0.60] \times [-0.23, +0.72]$ m, with respect to the robot's reference frame $O - xyz$. Note that all trajectories,

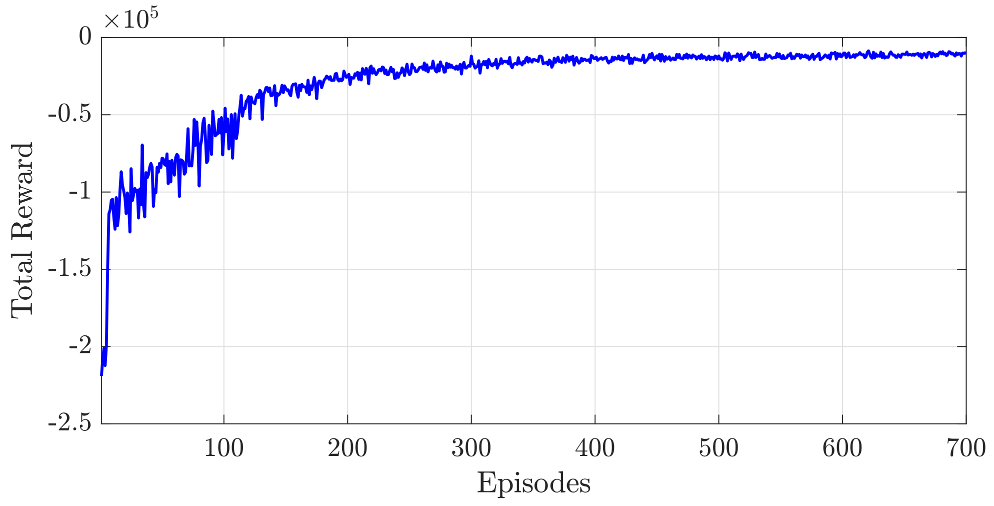


Figure 12.3: DRL reward curve for the UR5 case study

generated for either training or testing, are such that, if the robot had to perfectly follow the provided end-effector reference, a collision with the obstacles would happen at a certain point in time. It is important to notice that, although the trajectories are pre-recorded and rolled out during the execution, this is transparent to the agent, which only receives instantaneous observations from the environment. This way, since the trajectories are obtained through the actual motion sensor, it is possible to simulate the presence of a human operator generation the target's motion during training, so that the teleoperation nature of the problem is not lost.

The training has been executed for 700 episodes. The corresponding learning curve is reported in Figure 12.3, in which the total reward appears to converge to an average value of about $R = -10^4$.

12.3.1 Results

After completing the training phase, the DRL algorithm was validated in simulation on a test set consisting of 500 new reference trajectories, spanning the same range of values used during training, each of them with a total duration of 40 s (i.e., 800 steps). The performance index used for evaluating the performance of the proposed approach is the *episodic cumulative reward*, i.e.,

$$R_n \triangleq \sum_{k=0}^{n} r_{t+k+1} - R_{O_{t+k+1}} \tag{12.2}$$

the sum of the computed reward function at each step *without* accounting for the obstacle component, considering a sequence of n steps. As a result, an average value of $R_n = -1.69 \times 10^4$ was obtained, with a standard deviation of 2.70×10^3. Furthermore, no collisions were observed between the robot and the obstacles for this test set. In order to compare these results with those represented in the training curve, the same metrics have been evaluated in the first 18 s of the simulations in the test set, obtaining an average value of $R_n = -8.26 \times 10^3$, with a standard deviation of 1.10×10^3. The result is in line with the final value of $R_n = -10^4$ shown in the learning curve, considering the exclusion of R_O.

12.4 MPC as an alternative approach

This section summarizes a variation of the MPC approach described in [91] for comparing with the RL framework of Section 12.2. A finite-horizon optimal control problem (FHOCP) is solved online to directly determine a different deterministic policy $\hat{\mu}(x|\theta^{MPC})$, where θ^{MPC} is the set of tuning parameters of the MPC controller that will be detailed in the following. At any given time instant t, the reward function is defined as

$$\hat{r}(x_t, u_t) \triangleq -c_1 \hat{R}_T - c_2 R_A, \tag{12.3}$$

in which $\hat{R}_T$ is a purely quadratic function (needed to be able to efficiently determine $\hat{\mu}(x|\theta^{MPC})$ online), defined as $\hat{R}_T \triangleq \frac{1}{2} d^2$. Notice that the penalty for obstacle collisions is not inserted in the reward function; an explicit constraint on obstacle avoidance is instead directly inserted in the FHOCP. The cumulative reward for MPC is thus defined as

$$\hat{R}_t \triangleq \sum_{k=0}^{N} \hat{r}_{t+k+1|t}, \tag{12.4}$$

where $N \in \mathbb{R}_{>0}$ is the so-called *prediction horizon*, while $\hat{r}_{t+k+1|t}$ is the forecast of $\hat{r}$ at time $t+k+1$, predicted at time t. The FHOCP aims at determining the optimal realization of the control sequence $u_t \triangleq \{u_{t|t}, u_{t+1|t}, \dots, u_{t+N-1|t}\}$, namely u_t^*, based on a prediction of the evolution of the obstacle configuration $O_t \triangleq \left\{\mathcal{O}_{t+1|t}, \mathcal{O}_{t+2|t}, \dots, \mathcal{O}_{t+N|t}\right\}$ and of the reference $p_t^* \triangleq \left\{p^*_{t+1|t}, p^*_{t+2|t}, \dots, p^*_{t+N|t}\right\}$ from the human operator, both predefined at time t. On the other hand, the prediction of the robot configuration, namely $q_t \triangleq \left\{q_{t|t}, q_{t+1|t}, \dots, q_{t+N|t}\right\}$, is determined by the predicted realization of the control sequence. The FHOCP formulation is then as follows:

$$u_t^* = \underset{q_t, O_t, p_t^*, u_t}{\mathrm{argmax}} \ \hat{R}_t(q_t, p_t^*, u_t) \tag{12.5a}$$

$$\text{s.t. } q_{t|t} = q_t \tag{12.5b}$$

$$q_{t+k+1|t} = q_{t+k|t} + t_s u_{t+k|t}, \ k = 0, \dots, N-1, \tag{12.5c}$$

$$\hat{d}_O(q_{t+k|t}, \mathcal{O}_{t+k|t}) \geq 0, \ k = 1, \dots, N, \tag{12.5d}$$

where (12.5b) imposes that the sequence of predicted joint variables $q_{t+k|t}$ starts from the currently measured value q_t, (12.5c) determines a simplified system dynamics (by assuming that the desired joint velocity, instead of the actual joint velocity, directly influences the joint position) where t_s is the sampling interval, and (12.5d) imposes obstacle avoidance. More specifically, $\hat{d}_O(q, \mathcal{O})$ represents a conservative estimate of the distance between robot and obstacles $\mathcal{O}$ based on an over-approximation of their space occupancy. Indeed, contrary to the calculation of the distance d_O used in DRL, which can be done through a simulator, an explicit formula is needed to insert this distance inside the FHOCP. As an example, a number of *test points* can be defined on the robot frame and on the obstacles, each of them being at the center of a sphere with given radius; if the union of the spheres on the robot includes the whole robot frame, and analogously for the obstacle, then condition (12.5d) can be imposed by requesting that all distances between each test point on the robot and each test point on the obstacle are greater or equal than the sum of the corresponding sphere radii, as detailed in [91].

Once the FHOCP (12.5a)-(12.5d) is solved at time t, only the first element of the control sequence determines the MPC policy $\hat{\mu}(x|\theta^{MPC}) = u_{t|t}^*$, and the FHOCP is then solved at time $t+1$, after new measurements are available. The set of MPC parameters θ^{MPC} includes c_1 and c_2, the parameters of the method used to formulate (12.5d), and those of the method used to predict future obstacle and reference positions.

The MPC controller is designed using the same parameters c_1 and c_2 as in Table 12.1, and a prediction horizon $N = 20$, corresponding to a total prediction time of 1 s. The prediction of the end-effector position reference p^* is obtained by linearly extrapolating the current value of p^* based on an estimate of its current speed $\dot{p}^*$, as in [91]. Also, the avoidance of box and table is obtained similarly to the description provided in Section 12.4, by defining 7 test points on the manipulator, and the same number of spheres that cover the whole robot frame. The space occupied by the box is covered by an ellipsoid, and the condition for imposing absence of intersections between the spheres on the robot and the ellipsoid is imposed as described in [91, Sec. II.C]. Finally, the avoidance of the table is also imposed as described in [91, Sec. II.C], by requiring that all spheres on the robot remain entirely above the horizontal plane that defined the table surface.

12.5 Experimental Results

In this section, experimental results for the considered UR5 case study are presented and discussed. In order to compare the proposed approaches, 25 experiments, each with a duration of 40 s, were executed. In all tests, the same arm tracking data is provided to both DRL and MPC, by recording the time evolution of the reference position p^*, and then providing it to both control schemes. In none of the 25 experiments collisions were observed, as it was the case for the DRL simulations: this confirms the collision avoidance ability of both schemes.

12.5.1 System specifications

Both controllers are implemented in Robot Operating System (ROS) and run on an Acer laptop with a 2.6GHz Intel Core i7-9750H CPU with 16GB RAM. The UR5 robot provides, every 50 ms, joints and end-effector positions to the controllers via TCP/IP communication, and the controllers transmit the control inputs (i.e., the joint speed references for the internal control loops) over the same connection as URScript language commands. The optimization solver for the MPC controller is generated using the ACADO Toolkit [93], which includes a code generation tool. The FHOCP (12.5) is formulated via multiple shooting with a discretization interval coinciding with the sampling interval of 50 ms. A Gauss-Newton approximation is used to define the Hessian of the Lagrangian, and the problem is solved via sequential quadratic programming, with each (condensed) quadratic program solved using the dual active set method implemented by the qpOASES solver [94]. On the other hand, the DRL controller is written as a ROS integrated Python 3 program.

Table 12.2: Performance of DRL and MPC algorithms for 25 real-time experiments with a duration of 40 s: R_n (left) and $\hat{R}_n$ (right)

	R_n ($\times 10^4$)		$\hat{R}_n$ ($\times 10^4$)	
	mean	std. dev.	mean	std. dev.
DRL	-1.70	0.22	-2.24	0.32
MPC	-1.84	0.25	-2.48	0.32

Through ROS communication topics, the DRL controller receives both the UR5 joint positions and the references from the arm tracker system on the human operator.

12.5.2 Results

The two approaches are compared in terms of two episodic cumulative rewards, computed a-posteriori as Eq. (12.2) and as

$$\hat{R}_n \triangleq \sum_{t=1}^{n} \hat{r}_t, \tag{12.6}$$

where $\hat{r}_t$ is computed as in Eq. (12.3) (i.e., using a quadratic end-effector penalty instead of a Huber-Loss). As one can notice in Table 12.2, the DRL approach provides a performance improvement of about 7%; this result is rather consistent through the different experimental trials, as the standard deviation is only about 13% of the mean value of R_n for both experiments using DRL and MPC. Furthermore, the DRL performance is consistent with results obtained during the testing phase, as the mean value of R_n is approximately the same. Also in when using the metric $\hat{R}_n$, DRL consistently outperforms MPC of about 9.7%.

Additionally, two sets of experiments are conducted: in the first set, the DRL algorithm is executed in real-time on the experimental setup, and the reference signal recorded for this first experiment is used with the MPC algorithm for comparison. Conversely, in the second set of experiments, the MPC algorithm is executed with a real-time reference from the arm tracker, generated by a human operator, and the same signal is later provided as a reference for the DRL algorithm.

Set 1: Real-Time DRL

The overall performance of the two controllers for the first set of experiments is shown in Table 12.3. All the results are consistent with those of Table 12.2. The

Table 12.3: Performance of DRL and MPC algorithms for real-time experiments with a duration of 40 s and with reference mutually recorded (Set 1 and Set 2): R_n (left) and $\hat{R}_n$ (right)

		R_n ($\times 10^4$)	$\hat{R}_n$ ($\times 10^4$)
Set 1	DRL real-time	-1.91	-2.54
	MPC recorded	-2.13	-2.85
Set 2	DRL recorded	-3.91	-5.40
	MPC real-time	-4.16	-6.17

evolution of the reference joint speeds $\dot{q}^*$ for the two control schemes is reported in Figure 12.4, where it can be seen that the amplitude of the corresponding control inputs for DRL and MPC are consistent, showing that the tuning of the two controllers led to a similar trade-off between control energy and tracking error. The reference tracking for DRL and MPC is reported in Figure 12.5, in which both controllers also succeed to avoid the obstacles (table and box) in the workspace. The tracking for the x-component of $p^\star$ is similar for both controllers, while, for the y-component of $p^\star$, DRL shows better results. The contrary applies for the z-component, in which MPC showed better results.

Set 2: Real-Time MPC

In the second set of experiments, a reference trajectory is provided with variations of p^* in time that are faster, on average, than those present in the training set and in the test set. This is done to assess the ability of DRL to generalize on different data. The overall performance is shown in Table 12.3: even though the values of all costs are about twice those of the first set of experiments, DRL still shows better results. The corresponding evolution of the joint speeds, which are again in the same range for DRL and MPC, can be seen in Figure 12.6. Also, the reference tracking for both controllers is shown in Figure 12.7 to which, in spite of the faster trajectories, the same comments apply as for Figure 12.5.

12.6 Conclusions

The proposed DRL strategy is applied to a teleoperated robot, tasked to track the movements generated at run-time by a human operator. The proposed approach succeeds in avoiding obstacles while tracking the provided reference position with satisfactory results. The observed advantages with respect to MPC can be summarized as a drastic (and largely expected) reduction in terms of execution time, and a satisfactory improvement in terms of performance, measured via the cost functions defined for DRL and MPC. Nevertheless, as compared to DRL, the MPC approach presents the following main differences:

- MPC aims at tracking the reference while avoiding the obstacles based on an explicit model-based prediction of the robot motion on a limited time horizon, while DRL learns its behavior via trial and error (during training).
- MPC is based on a simplified system dynamics in order to limit the computational complexity of the FHOCP, while DRL can be trained on a sophisticated robot simulator, which replicates the dynamics of the manipulator and the behavior of its internal controllers.
- MPC uses a conservative evaluation of the distance with the obstacles, while DRL can rely on precise distances calculated by the employed simulator during training.
- MPC does not need training data, and is therefore much more robust towards managing unforeseen events, such as a reference motion never seen before; DRL might instead show unexpected behavior should this happen.

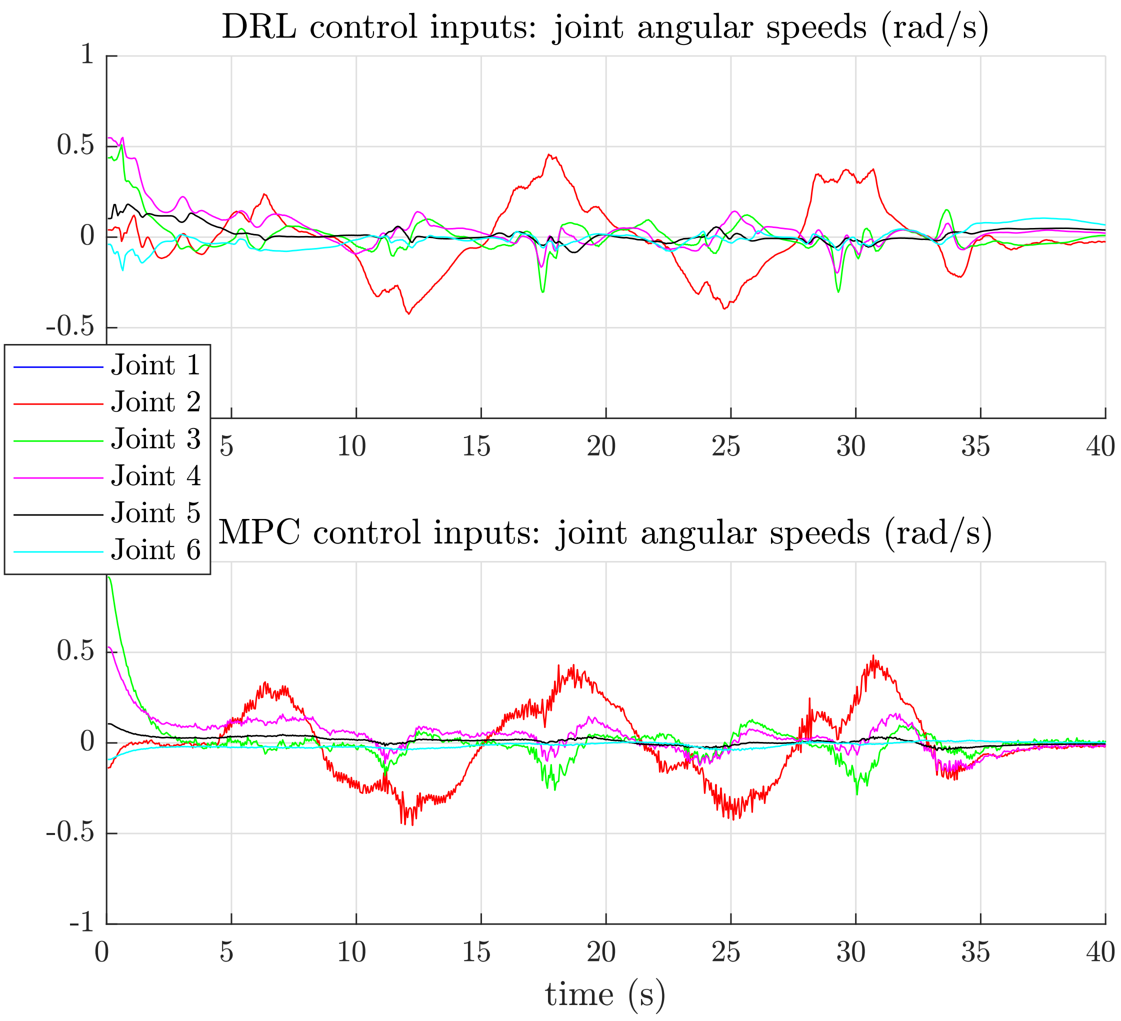


Figure 12.4: Time evolution of the components of joint angular speed reference vector $\dot{q}^*$ for the first set of experiments

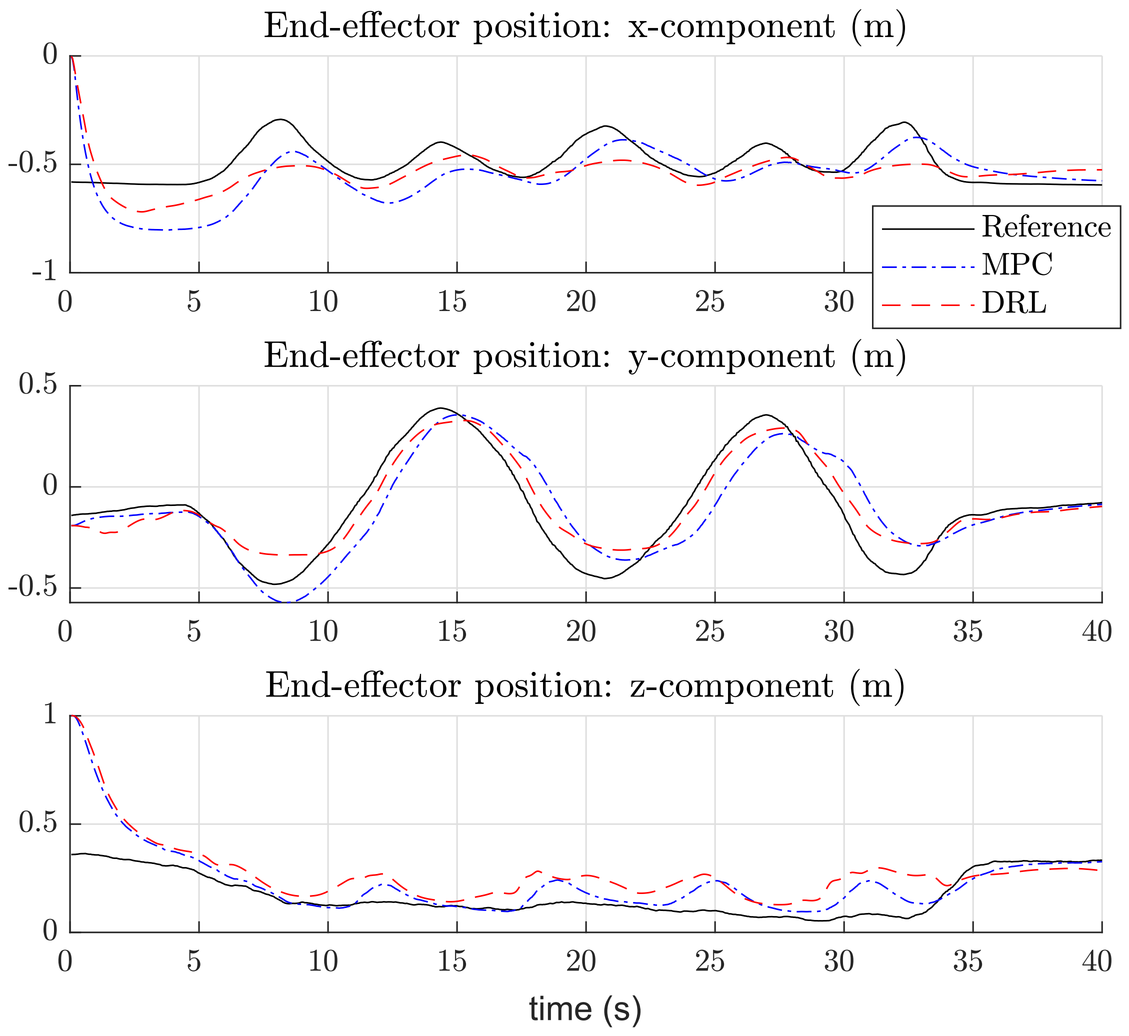


Figure 12.5: Time evolution of the xyz components of reference p^*, and corresponding values of p_e for DRL and MPC, in the first set of experiments

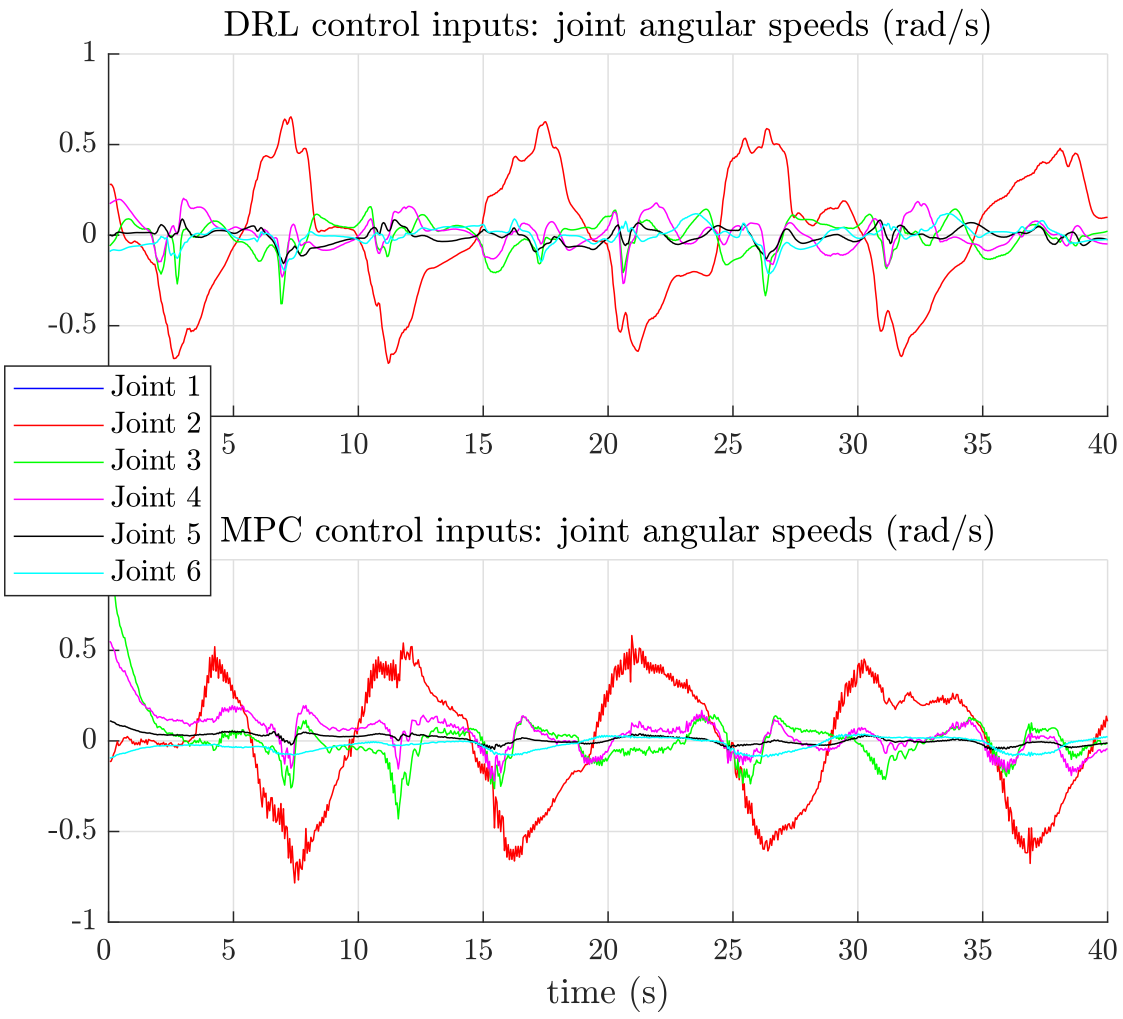


Figure 12.6: Time evolution of the components of joint angular speed reference vector $\dot{q}^*$ for the second set of experiments

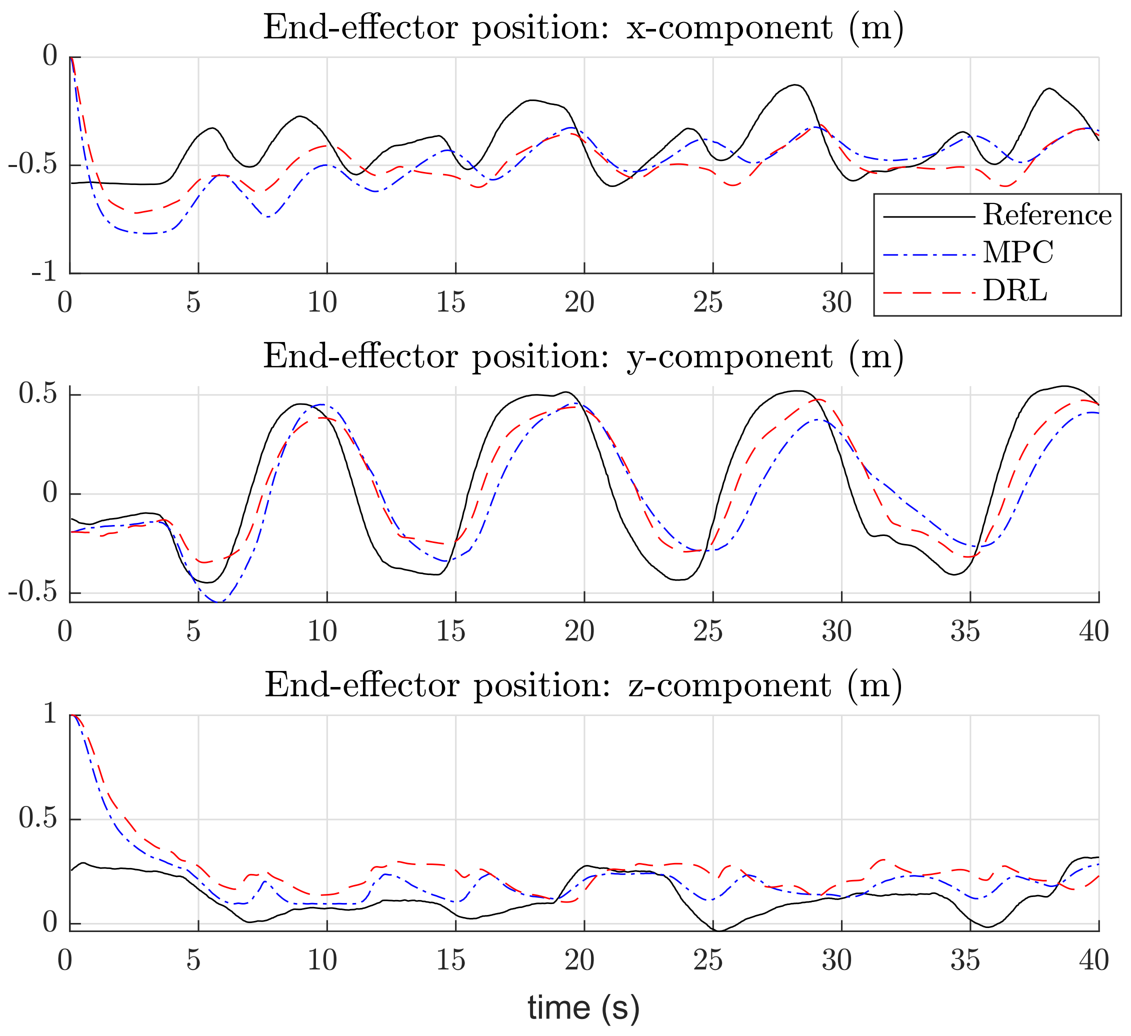


Figure 12.7: Time evolution of the xyz components of reference p^*, and corresponding values of p_e for DRL and MPC, in the second set of experiments

Chapter 13

Experiments on the Epson VT6 industrial manipulator

In this Chapter, preliminary work on the deployment of the proposed DRL framework for collision avoidance on a real manipulator is presented and discussed. The robot on which the experiments were performed is an Epson VT6[1] [95], a 6-axis industrial manipulator with built-in controllers. In order not to bypass the embedded security measures implemented within the proprietary industrial software (Epson RC+), it was decided to create a middleware interface between the learning environment and the industrial software, which communicates directly with the real robot, in order to replicate the movements of the virtualized robot. Furthermore, it allows the simulation of external elements (i.e., obstacles and other entities) without the actual physical obstacles and the necessary motion capture system.

13.1 System Setup

The system, property of the *Identification And Control of Dynamic Systems (ICDS)* Lab of the University of Pavia, is composed of three main elements and the interfaces between them, as summarized in Figure 13.1. Specifically:

1. **The Epson VT6 industrial manipulator:** the physical manipulator, anchored to a base on the concrete floor of a sensorized cage to which the safety

[1]The Epson VT6 that has been used to perform the experiments presented in this Chapter has been awarded to the University of Pavia for being one of the three winners of the Epson's ≪*Win-a-Robot*≫ contest, directed at EMEAR research institutes.

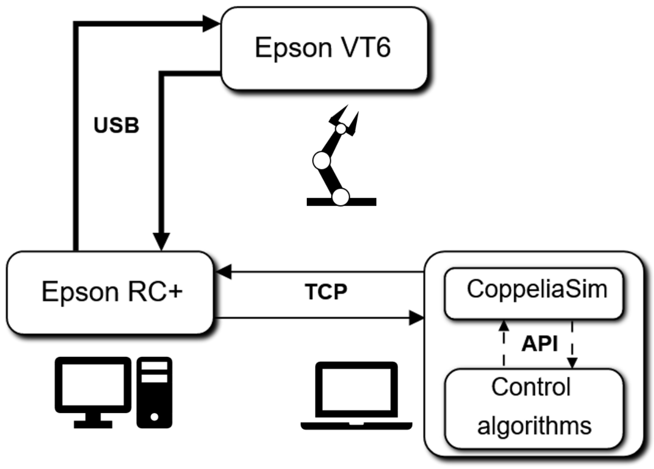


Figure 13.1: Interfacing between the robot and the control units

and emergency stop system are connected. It is directly interfaced with its proprietary software, from which it receives the low-level control commands and operation signals (turn on/off commands, emergency stops, etc.), and to which sends feedback information on the operation status.

2. **Epson RC+ proprietary control system:** the industrial software that manages exchanges of information from and to the physical system and regulates the robot's operation. The program runs on a dedicated computer, which stays connected via USB to the robot. On the other hand, the software communicates via TCP/IP with the virtualized environment, receiving the high-level control commands.

3. **The virtualized environment and control algorithm:** the virtual representation of the robot and the custom dynamic environment (reproduced with V-Rep), which simulates the robot together with other elements and reacts accordingly to the custom control algorithm (e.g., end-to-end DRL, MPC, etc.). The control algorithm and the virtualized environment are interconnected through a dedicated API.

13.1.1 Epson VT6 Industrial Manipulator

The Epson VT6 is a 6-axis manipulator from Epson. The robot, pictured in Figure 13.2, is designed for industrial applications and features built-in controllers,

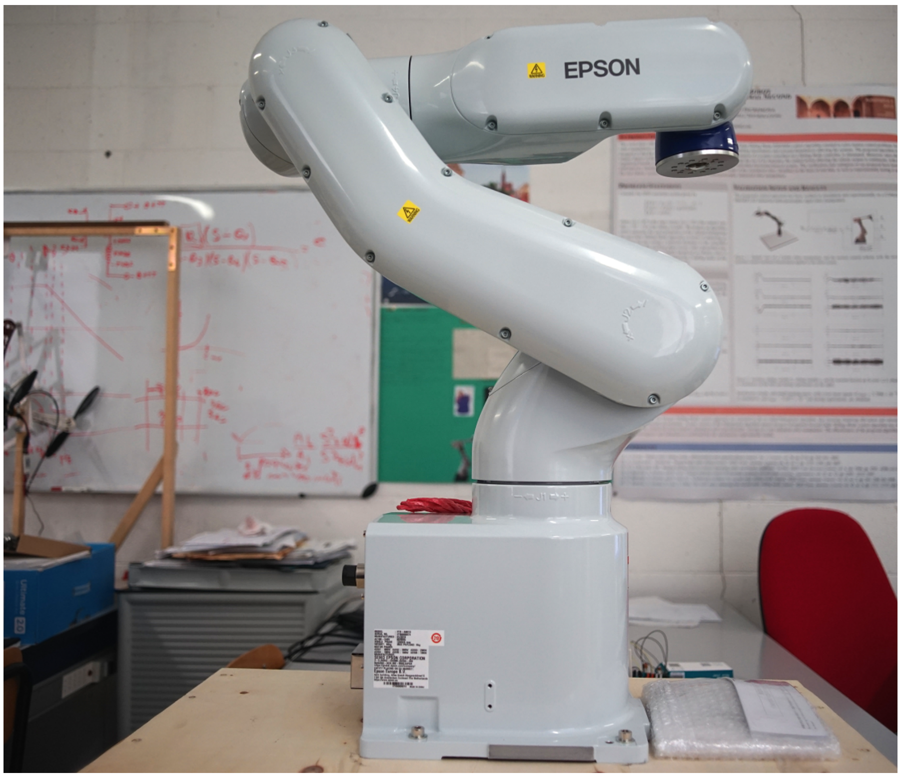


Figure 13.2: Photo of the Epson VT6 industrial manipulator

an operation range of 900 mm, and can support a payload of up to 6 kg. The joint specifications are reported in Table 13.1 and visualized in Figure 13.3. The motion has a repeatability of 0.1 mm for all joints.

Table 13.1: Epson VT6 joint specifications

	Position (deg)	**Velocity (deg/s)**	**Torque (Nm)**
Joint 1	[-170, 170]	±166.2	50
Joint 2	[-160, 65]	±122.2	50
Joint 3	[-51, 190]	±118.8	50
Joint 4	[-200, 200]	±271.4	12
Joint 5	[-125, 125]	±296.8	12
Joint 6	[-360, 360]	±293.2	7

Furthermore, the Epson VT6 supports three different operation modes:

- **Teach mode:** operates at low power and enables the operator to manually guide the robot to point data that must be reached sequentially, by means of a dedicated device.

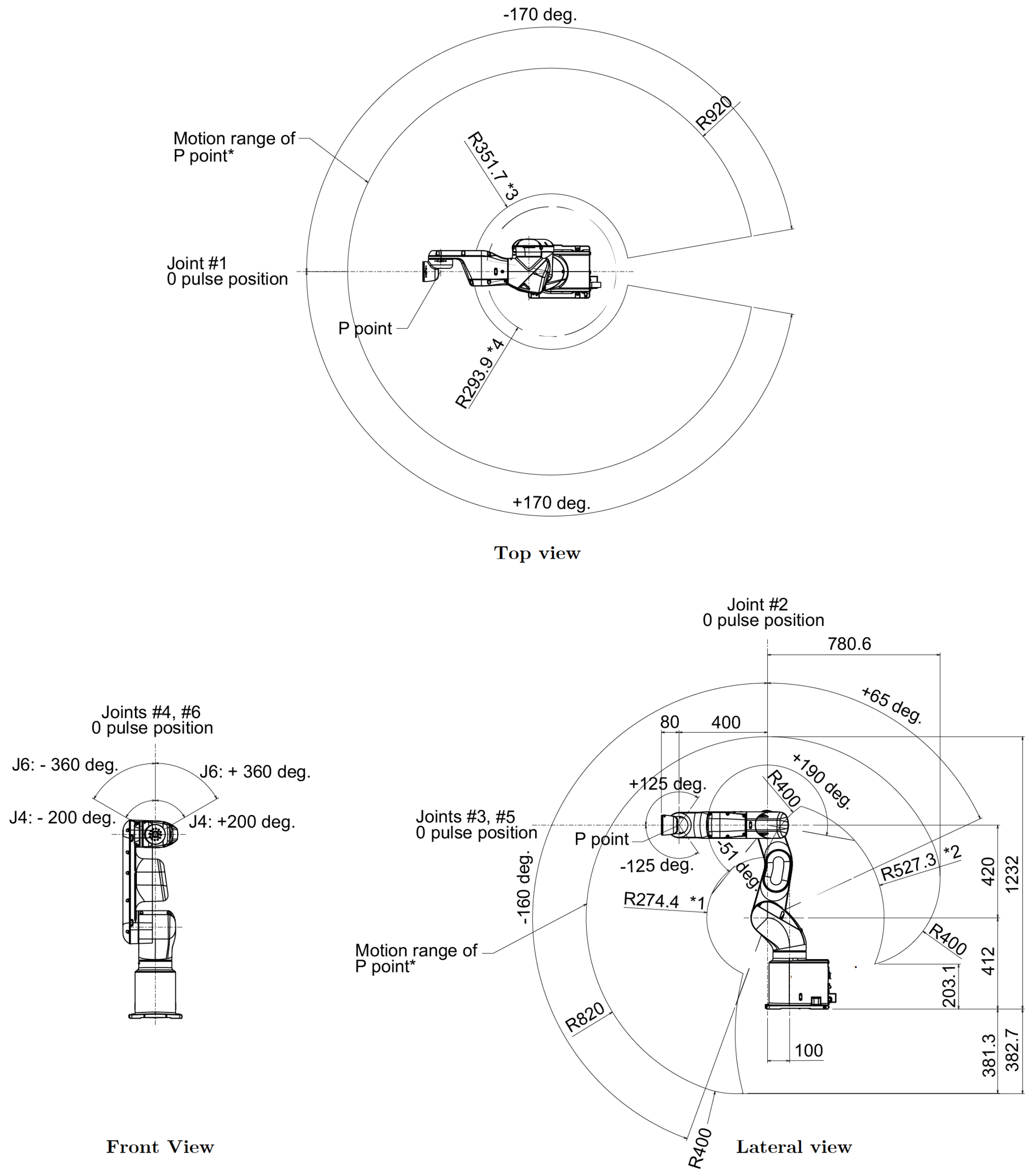


Figure 13.3: Epson VT6 motion ranges

- **Auto mode:** enables automatic execution of a preloaded program to deploy manufacturing operations.
- **Test mode:** enables program verification.

For the presented work, the robot always operates in *auto mode*, which in turn relies on external information to execute the routine. The robot is controlled by the *RC700* controller, which features a built-in motion system for the 6 AC Servo motors and manages I/O communication through the proprietary software *Epson RC+*, which will be detailed in the following subsection. Robot's motion, which can be done both in the joint space and the operative space, can be controlled in the following ways:

- **Point To Point (PTP):** moves directly the end-effector from its current position to a desired point.
- **Linear motion:** moves the end-effector from its current position to a (sequence of) desired point(s) *with a straight line.*
- **Curve motion:** moves the end-effector from its current position to a (sequence of) desired point(s) *with a predefined curve.*
- **Joint relative movement:** moves one joint up to a specified distance from the current position.

Although a native function to define joint reference positions all at once is not directly available, this can be easily achieved using a function that computes the forward kinematic given the desired joint configuration: motion can be then be performed in the operative space using the output of said function. More details on the motion functions are reported in Appendix B. The controller uses trapezoidal velocity profiles (TVP) to perform the motion. If a sequence of different movements must be performed, *continuous path* operation can be imposed, so that two consecutive velocity profiles are overlapped to avoid full deceleration and acceleration during motion, as exemplified in Figure 13.4. In all options, is possible to set the maximum velocity and acceleration, expressed in mm/s and mm/s^2.

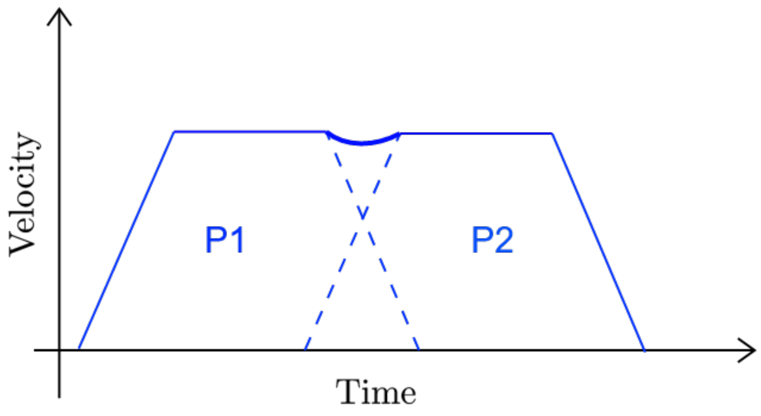


Figure 13.4: Trapezoidal velocity profile when imposing continuous motion between two points

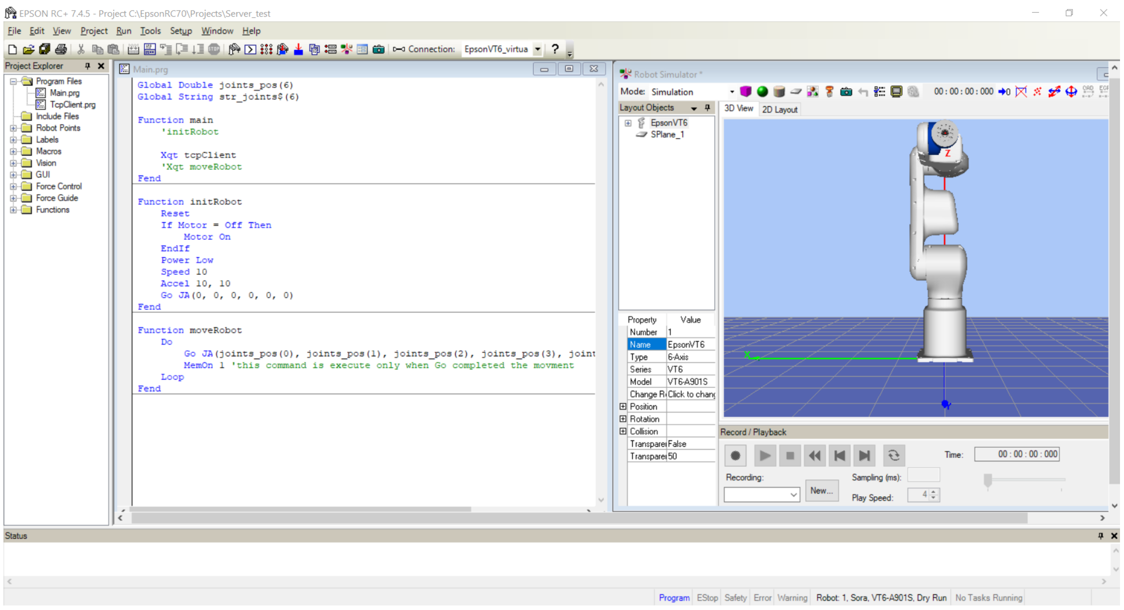
13.1.2 Proprietary interfacing software Epson RC+

Epson RC+ is Epson's proprietary software to directly interface a computer with the robot's controllers, via USB or Ethernet. Besides managing the proper setting and communications of the robotic application, the software features a proprietary IDE for high-level development of programs for the robot's controllers. As it is designed for industry applications, the language features intuitive, easy to interpret commands to define routines and robot motions, in order to facilitate operation by employees. The most relevant features are:

- **Developement environment:** enables the writing, editing, debugging, and testing of a robot routine. Programs are written in the *SPEL+* programming language, which is based on a robust library of pre-built, high-level functions to handle motion, multithreading, I/O control, sensor sampling, and actuation running in the RC700 controller.

- **Simulators:** allows the execution of the developed program on a built-in simulated environment before deploying on the physical system. The simulator presents an extensive library of robots, whose dynamic behaviors are reproduced with high accuracy, so that the deployment on the physical robot is transparent to the user.

- **External communication:** Epson RC+ can be easily interfaced with other software and applications using TCP/IP communication.



```
Global Double joints_pos(6)
Global String str_joints$(6)

Function main
    'initRobot

    Xqt tcpClient
    'Xqt moveRobot
Fend

Function initRobot
    Reset
    If Motor = Off Then
        Motor On
    EndIf
    Power Low
    Speed 10
    Accel 10, 10
    Go JA(0, 0, 0, 0, 0, 0)
Fend

Function moveRobot
    Do
        Go JA(joints_pos(0), joints_pos(1), joints_pos(2), joints_pos(3), joint
        MemOn 1 'this command is execute only when Go completed the movment
    Loop
Fend
```

Property	Value
Number	1
Name	EpsonVT6
Type	6-Axis
Series	VT6
Model	VT6-A901S
Change Ro	Click to chang
Position	
Rotation	
Collision	
Transparei	False
Transparei	50

Figure 13.5: Epson RC+ GUI showing the IDE with a program script to be executed on a simulated robot.

- **Safety checks and emergency management:** prevents dangerous operations from occurring by halting the controllers and activating the emergency system.

In Figure 13.5, the user interface showing the IDE and the built-in robot simulator is represented. The purpose of Epson RC+ is to provide a safe and compliant interface between any external module and the physical system: in fact, it needs to be ensured that all implemented safety procedures and native controllers are not bypassed or overridden by custom control algorithms. Epson RC+ is installed on a *Microsoft Windows* system running Windows 10.

13.1.3 Virtualized environment

The virtualized environment is the reproduction of the desired robotic scenario within a general-purpose robot simulator. As one can see in Figure 13.6, a replica of the Epson VT6 robot has been implemented in V-Rep, using the original CAD file and the available dynamical parameters provided in the datasheets. Together with the robot, a spherical obstacle was inserted in the environment, where it can move freely in two directions. Additionally, a visual rendering of the target point (the black dot) was added. The purpose of the virtualized environment is dual:

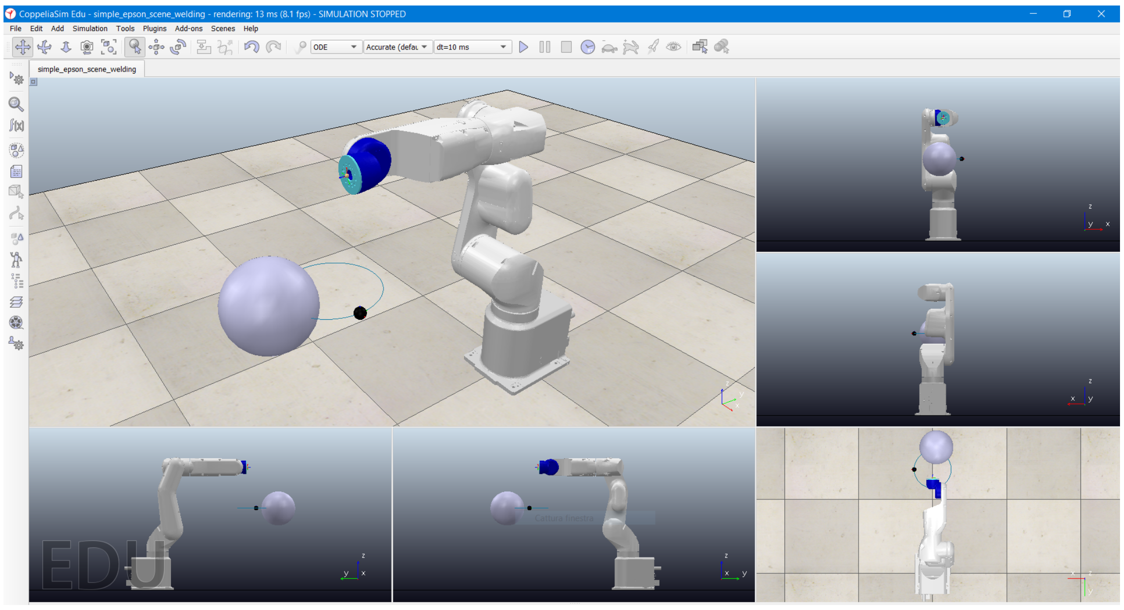

CoppeliaSim Edu - simple_epson_scene_welding - rendering: 13 ms (8.1 fps) - SIMULATION STOPPED
File
Edit
Add
Simulation
Tools
Plugins
Add-ons
Scenes
Help
ODE
Accurate (defau
dt=10 ms
simple_epson_scene_welding
EDU

Figure 13.6: Virtualization of the Epson VT6 robot with the V-Rep robotic simulator

first, it is used to recreate complex, dynamic scenarios (which would be impossible to reproduce in Epson RC+) that are needed for the deployment of the custom control algorithm running synchronously on the same system; then, it generates the high-level reference signals that are sent to the proprietary software, so that the actual robot can reproduce them. The virtualized environment, along with the controller, run on a *Linux Ubuntu* system running Ubuntu 18.04 LTS.

13.2 Case study

For the proposed experiments, the same scenario introduced in the previous chapters of this Part is considered. Specifically, the Epson VT6, placed in an environment together with a randomly moving obstacle, needs to perform the following two tasks:

T1) to track a desired trajectory in order to reach a pre-specified target;

T2) to avoid collisions with elements invading the workspace, while reaching the target.

To achieve this, the DRL-based hybrid approach for motion planning and obstacle avoidance introduced in Chapter 11, Algorithm 3 is used. The policy used for End-to-End control of collision avoidance is selected after 550 episodes of training

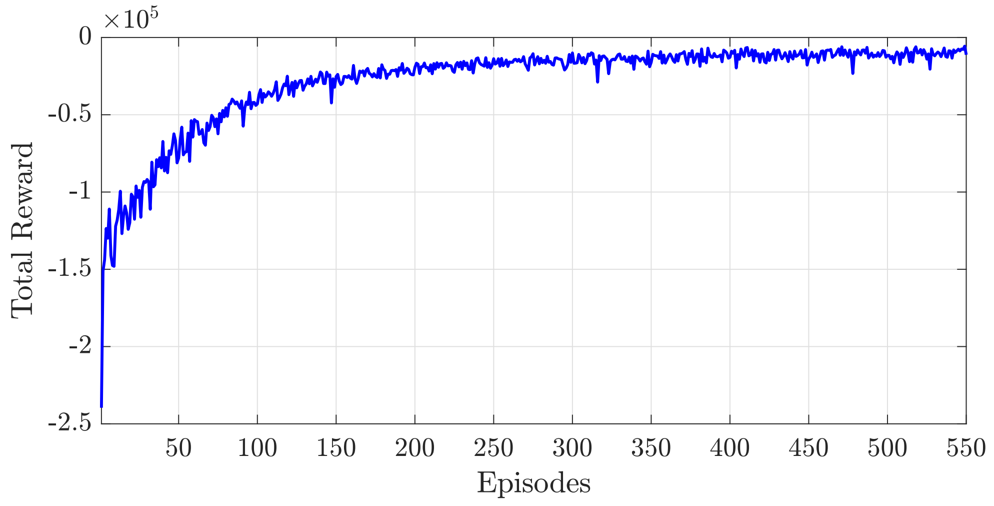


Figure 13.7: Cumulative reward function for training on the Epson VT6

on the Epson VT6 model. The cumulative reward function is represented in Figure 13.7. Furthermore, the scenario considered reproduces the spot welding example introduced in Section 11.4.2: the robot must track a target that moves along a path, on which it advances after being reached.

13.2.1 Interfacing between components

For the execution of the experiment, the elements in the framework communicate as detailed in the following. More specifically, two threads work in parallel to handle interfacing between components: one manages the execution of the control algorithm on the virtual environment, while the other deals with the communication between the virtual environment in V-Rep and Epson RC+.

1. First, the control algorithm is initialized and connected to the virtual environment in V-Rep.

2. At run-time, the algorithm reads the status of the simulation (i.e., sensor data, changes in the environment, collision detection, etc.) and elaborates the retrieved information from V-Rep to generate desired output, in this case $(q^\star, \dot{q}^\star)$ (SBL) or $\dot{q}^\star$ (DRL).

3. Once the algorithm determines the output, this is sent to V-Rep, where the physics engine executes the command and advances the simulation. Communication from and to V-Rep is handled by the PyRep [77] plugin, which

encapsulates the native V-Rep API.

4. Throughout execution, the virtual robot configuration q is sampled by a dedicated thread, running in an embedded script within V-Rep, which then sends the joint information, in the form of a string, to Epson RC+ through TCP/IP via Ethernet connection.

5. Every time new information is received in Epson RC+ from V-Rep, the running program decodes the received string into joint positions, and runs the motion command so that the configuration of the virtual robot is reproduced.

6. Epson RC+ then directly controls the Epson VT6 via USB to execute the desired motion in real-time.

13.2.2 Experimental results

In the following, the movements of the real Epson VT6 industrial manipulator are reported and compared to those of the virtualized version. Figure 13.8 shows that the end-effector's position of the real robot obtained through the presented interfacing architecture features a satisfactory level of fidelity with what is obtained in simulation, with a delay of about 0.5 s, allowing a reliable deployment of the proposed approach. In Figure 13.9 the distances from the target point and the distance between robot and obstacle are reported. A video of the experiment can be visualized at the link `http://bit.ly/CH13_EXP`, with some screenshots reported in Figure 13.10.

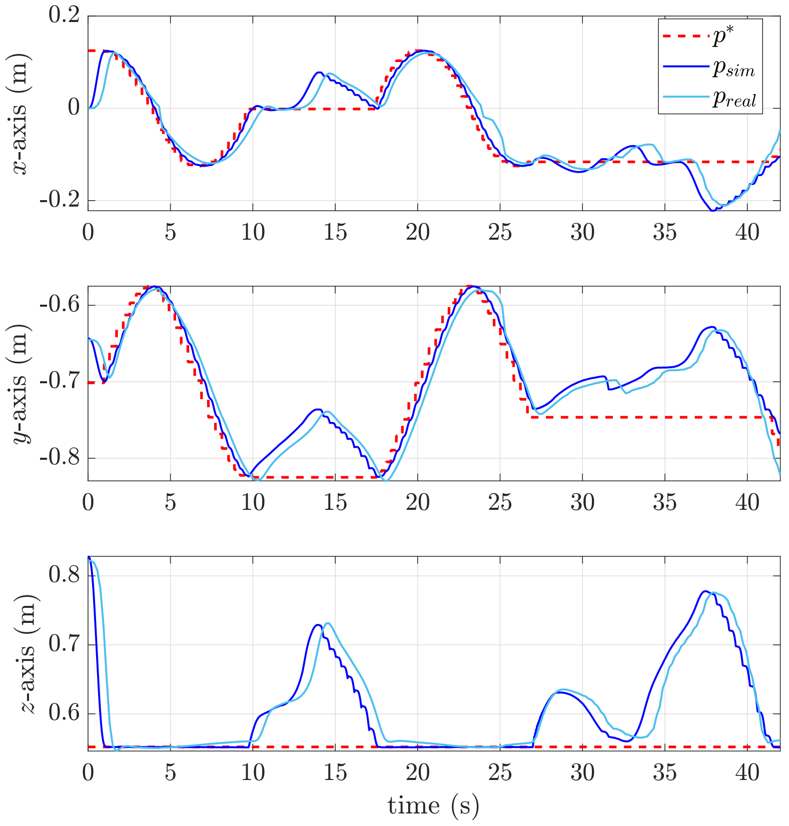


Figure 13.8: Position of the end-effector of the Epson VT6 industrial manipulator: virtual (blue) and real (cyan) with respect to the reference position (red)

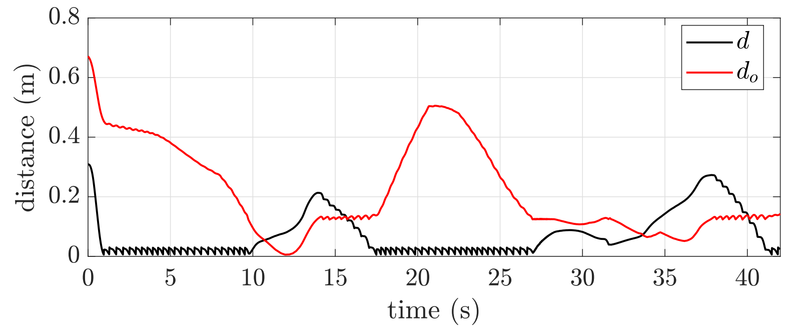


Figure 13.9: Distances between end-effector and target point (black) and robot and obstacle (red).

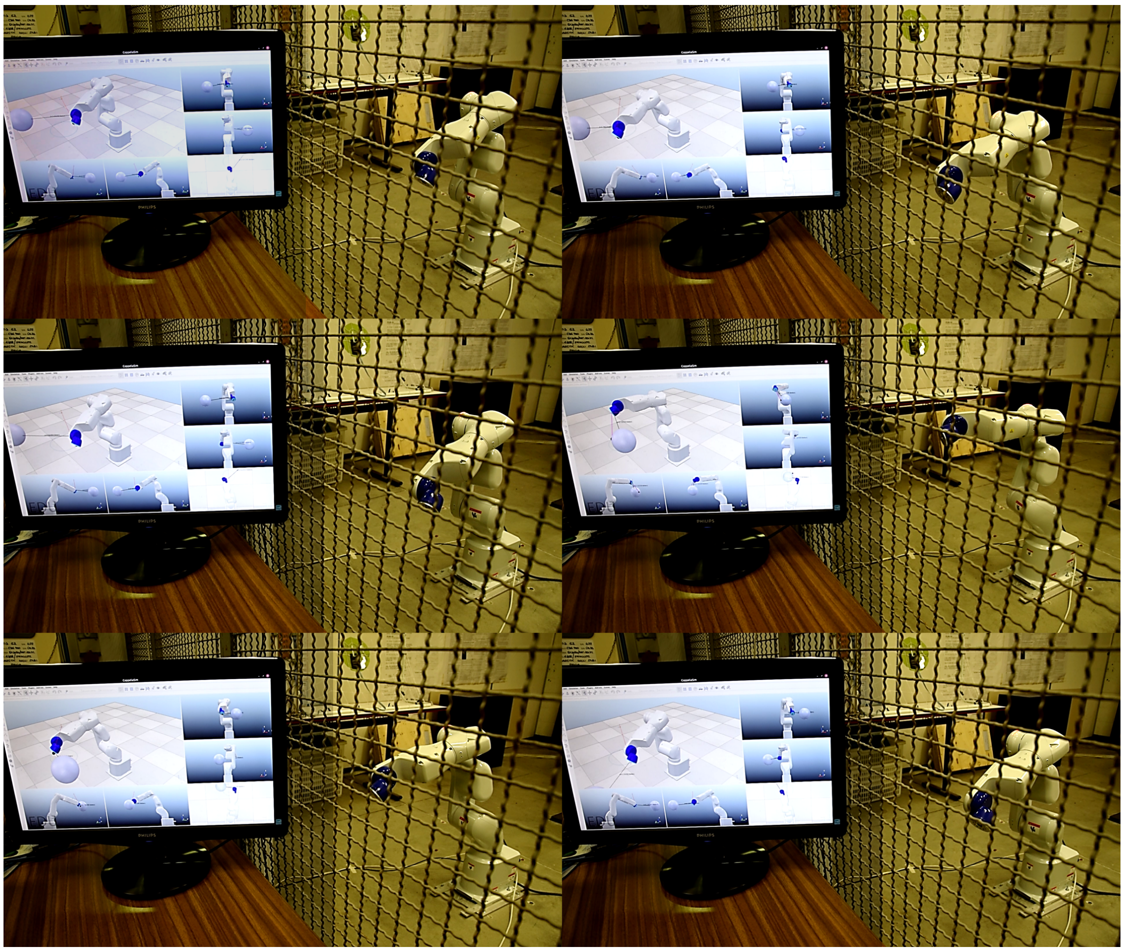


Figure 13.10: Some frames of the experiment described. On the monitor on the left, the virtual robot controlled by the algorithm in Chapter 11 is performing the task, while the real robot mimics it.

Chapter 14

Conclusions

This Part of the thesis dealt with the design and possible applications of a DRL-based, end-to-end control for full-body collision avoidance of industrial manipulators. In the considered scenario, robots are tasked with tracking reference positions with their end-effectors while avoiding collisions with obstacles invading their operative space. This is achieved by training a policy with a suitably selected framework used by a Deep Reinforcement Learning algorithm, i.e., Normalized Advantage Function [18], which enables model-free, end-to-end control of continuous systems. The proposed framework for full-body collision avoidance and target reaching has then been successfully applied to various scenarios, with varying degrees of complexity, featuring different robots, reference signals, and obstacle behaviors. Specifically, the following cases were considered and discussed:

1. Tracking of a fixed target, avoiding a single obstacle moving on a linear path.
2. Tracking fixed or linearly moving targets, with obstacles moving planarly or spatially. Training in more complex scenarios is facilitated using the so-called transfer learning, enabling the reuse of previously acquired knowledge.
3. Tracking a trajectory generated online by a human operator while avoiding fixed obstacles present in the operative space with a teleoperated robot.

In all cases, satisfactory performances were observed for both target reaching and obstacle avoidance, showing the validity of the proposed approach. Furthermore, a novel hybrid approach was introduced to combine end-to-end DRL-based control with conventional motion control: the robot then can achieve model-free collision avoidance while performing motion as generated by a suitable motion planning algorithm. The use of either approach is determined whether a given safety condition is met, thus conferring the system a self-configuring capability. As can be expected, the target reaching performance is improved using conventional methods, while collision avoidance maintained satisfactory performance. The proposed hybrid structure was then deployed on a physical robot, the Epson VT6, interfaced with a virtualized environment.

Part IV

Conclusion

Chapter 15

Concluding remarks

This Thesis aimed to present different examples of self-configuring systems involving robotic manipulators. The topics of motion control and collision avoidance for robots operating under varying degrees of uncertainty have been explored, and several case studies have been introduced showing the results obtained using the proposed approaches. In both cases, Deep Reinforcement Learning (DRL) has been implemented in the system into consideration, whether for decision-making or end-to-end control.

A novel switched-structure control scheme to achieve both centralized and decentralized control, using the perturbation estimation feature of the Integral Sliding Mode controller has been introduced; DRL has then been used to train the decision-maker that regulates the switch between either controller.

Then, the framework for end-to-end, model-free control of robotic manipulators to perform full-body collision avoidance with randomly moving obstacles has been introduced and validated on several scenarios, with varying complexity. The framework has then been used together with conventional motion planning algorithms in order to reduce the uncertainties given by the stochastic nature of the policy and improve performance in a novel algorithm that confers the system a self-configuring capability. Both the end-to-end approach and the hybrid algorithm have then been successfully deployed on real robots, and experimental data have been reported.

Encouraging results emerged, indicating that Deep Reinforcement Learning can indeed be successfully used in combination with conventional control methods to solve robotics applications.

15.1 Future work

Starting from the research conducted for this Thesis, future work in this direction could include the following:

- **Integrate a vision system in the training process:** in the presented research, sensor data was assumed known and retrieved from the simulator. Integrating the sensor models into the training, with image recognition and perception techniques, would facilitate deployment on the real system and improve the robustness of the proposed method.

- **Improve end-to-end control:** by training on a broader variety of scenarios, presenting specific industrial tasks, it would be possible to deploy the proposed framework for collision avoidance to solve more sophisticated tasks, such as interaction with other robots.

- **Scenario-based self-configuring control architecture:** given a set of control approaches, a decision-maker could be designed to select the most appropriate controller at any given time, for any given task, in order to achieve the best performance. A natural extension of the hybrid algorithm introduced in Chapter 11 would be to integrate other control strategies and different metrics, while a more sophisticated approach could use Deep Learning in order to pick the best choice, instead of relying of fixed metrics.

- **Improve policy training for decision making:** as shown by the preliminary results reported in Chapter 6, DRL has been used to train a policy for decision making. The approach could be extended to more controllers and a wider spectrum of applications.

- **Combine the proposed approaches:** an interesting case study could stem from the combination of the self-configuring control scheme for motion and the proposed DRL-based collision avoidance approach. Specifically, it could be of interest to observe the behavior of the switching strategy, employed as the low-level controller, in a scenario with unexpected changes caused by the reaction of the trained policy to incoming obstacles. Indeed, the effects of abrupt motion triggered by the avoidance strategy could be observed and handled by the ISM controller, perhaps improving performances.

Part V

Appendices

Appendix A

Comau Smart3-S2 model

In this Appendix, the model of the robot that has been used to produce the results presented in Chapters 5, 6 (Part II), and that has been virtually reproduced in V-Rep for the DRL experiments presented in Chapters 9, 10, 11 (Part III), is reported. The robot is an industrial anthropomorphic rigid manipulator *Comau Smart3-S2*, currently present at University of Pavia, and represented in Figure A.1. It consists of 6 links and 6 rotational joints driven by brushless electric motors.

In order to design the controllers introduced in Chapters 5 and 6, an estimation of dynamical parameters has been performed on the basis on real data, as extensively reported in the work of A. Calanca, M. Capisani, and A. Ferrara [57, 58], via a Maximum Likelihood estimation procedure. For this purpose, only vertical planar motions of the robot have been enabled, and a schematic representation of the considered model is reported in Figure A.2, where m_1, m_2, m_3, I_1, I_2, I_3 are the link's masses and inertias, l_{c_1}, l_{c_2}, l_{c_3} are the distances of the centers of mass with respect to the link's origin, and $l_1 = 0.65$ m, $l_2 = 0.6576$ m and $l_3 = 0.34$ m are the link lengths. For the sake of completeness, the results of the parameter estimation are reported in Table A.1 from the aforementioned papers, to which the reader is referred for in-depth explanation.

Considering the dynamical model of the robot expressed as in Eq. (2.19)

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s sign(\dot{q}) + g(q) = \tau,$$

for a 3-DoF rigid manipulator, elements composing the equation can be expressed using the estimated parameters as follows.

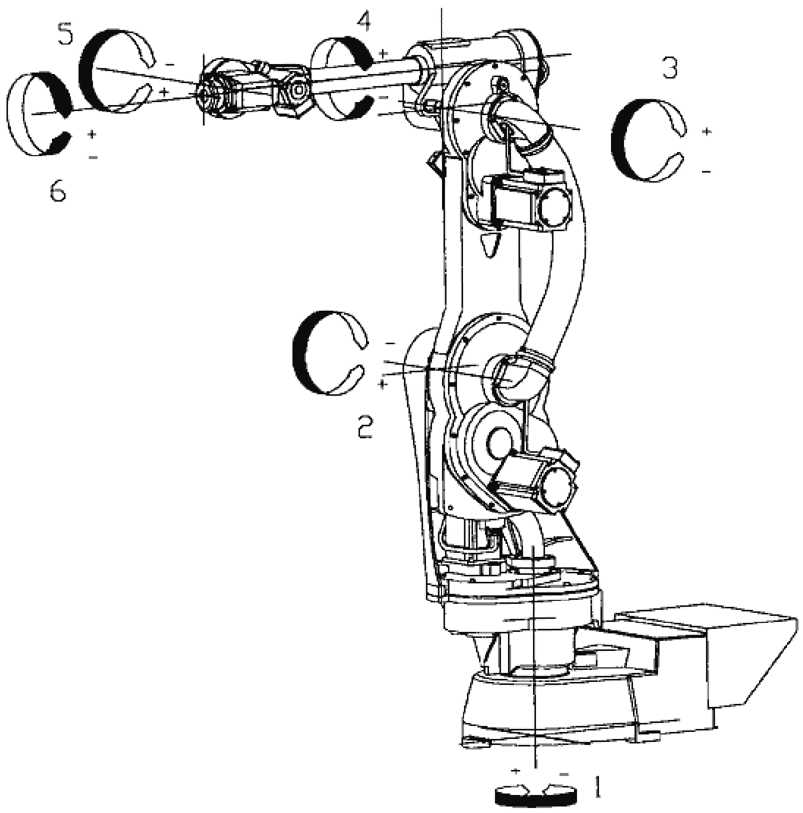


Figure A.1: Graphic representation of the Comau Smart3-S2 and its degrees of freedom.

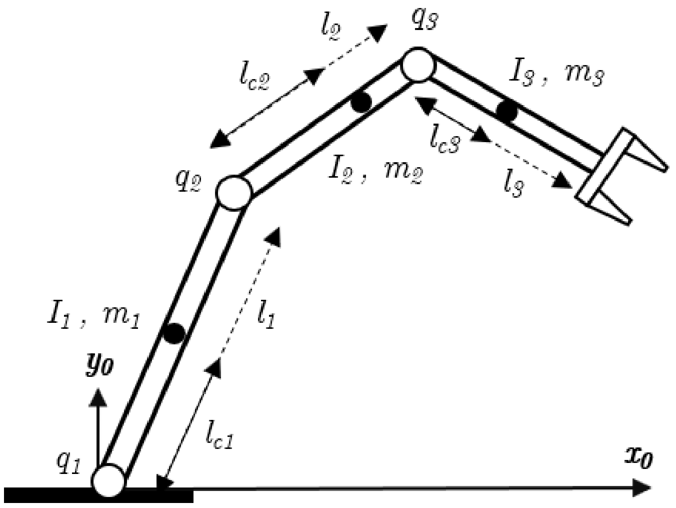


Figure A.2: Schematic representation of the planar model of the Comau Smart3-S2

Table A.1: Estimated values for dynamical parameters

Parameter	**Expression**	**Value**
γ_1	$m_3 l_{c_3}^2 + I_3$	0.2973
γ_2	$I_3 + m_3(l_1^2 + l_{c_3}^2) + I_2 + m_2 l_{c_2}^2$	10.066
γ_3	$I_3 + m_3(l_1^2 + l_2^2 + l_{c_3}^2) + I_2 + m_2(l_1^2 + l_{c_2}^2 + I_1 + m_1 l_{c_1}^2$	87.9151
γ_4	$m_1 l_{c_1} + m_2 l_{c_2} + m_3 l_1$	57.0347
γ_5	$m_2 l_{c_2} + m_3 l_2$	9.2148
γ_6	$m_3 l_{c_3}$	0.3163
γ_7	F_{s1}	190.4790
γ_8	F_{v1}	66.3430
γ_9	F_{s2}	20.9745
γ_{10}	F_{v2}	14.7050
γ_{11}	F_{s3}	4.6565
γ_{12}	F_{v3}	8.2911

Inertia Matrix

$$B(q) = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} \tag{A.1}$$

$$\begin{aligned}
B_{11} &= \gamma_3 + 2\gamma_5 l_1 cos(q_2) + 2\gamma_6 l_2 cos(q_3) + 2\gamma_6 l_1 cos(q_2 + q_3) \\
B_{12} &= \gamma_5 l_1 cos(q_2) + \gamma_2 + 2\gamma_6 l_2 cos(q_3) + \gamma_6 l_1 cos(q_2 + q_3) \\
B_{13} &= \gamma_6 l_1 cos(q_2 + q_3) + \gamma_6 l_2 cos(q_3) + \gamma_1 \\
B_{22} &= \gamma_2 + 2\gamma_6 l_2 cos(q3) \\
B_{23} &= \gamma_1 + 2\gamma_6 l_2 cos(q3) \\
B_{33} &= \gamma_1
\end{aligned} \tag{A.2}$$

Centripetal and Coriolis forces Matrix

$$C(q,\dot{q}) = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{32} \\ C_{31} & C_{32} & C_{33} \end{bmatrix}$$

$$\begin{aligned}
C_{11} &= -\dot{q}_2\gamma_5 l_1 sin(q_2) - (\dot{q}_2 + \dot{q}_3)\gamma_6 l_1 sin(q_2 + q_3) - \dot{q}_3\gamma_6 l_2 sin(q_3) \\
C_{12} &= -(\dot{q}_1 + \dot{q}_2)\gamma_5 l_1 sin(q_2)(\dot{q}_1 + \dot{q}_2 + \dot{q}_3)\gamma_6 l_1 sin(q_2 + q_3) - \dot{q}_3\gamma_6 l_2 sin(q_3) \\
C_{13} &= -(\dot{q}_1 + \dot{q}_2 + \dot{q}_3)\gamma_6(l_1 sin(q_2 + q_3) + l_2 sin(q_3)) \\
C_{21} &= \dot{q}_1\gamma_5 l_1 sin(q_2) + \dot{q}_1\gamma_6 l_1 sin(q_2 + q_3) - \dot{q}_3 l_2 sin(q_3) \\
C_{22} &= -\dot{q}_3\gamma_6 l_2 sin(q_3) \\
C_{23} &= -(\dot{q}_1 + \dot{q}_2 + \dot{q}_3)\gamma_6 l_2 sin(q_3) \\
C_{31} &= \gamma_6(\dot{q}_1 l_1 sin(q_2 + q_3) + \dot{q}_1 l_2 sin(q_3) + \dot{q}_2 l_2 sin(q_3)) \\
C_{32} &= (\dot{q}_1 + \dot{q}_2)\gamma_6 l_2 sin(q_3) \\
C_{33} &= 0
\end{aligned} \tag{A.3}$$

Friction coefficients

$$F_s = \begin{bmatrix} F_{s1} & 0 & 0 \\ 0 & F_{s2} & 0 \\ 0 & 0 & F_{s3} \end{bmatrix}, \quad F_v = \begin{bmatrix} F_{v1} & 0 & 0 \\ 0 & F_{v2} & 0 \\ 0 & 0 & F_{v3} \end{bmatrix}$$

$$\begin{aligned}
F_{s1} &= \gamma_7, \quad F_{v1} = \gamma_8 \\
F_{s2} &= \gamma_9, \quad F_{v2} = \gamma_1 0 \\
F_{s3} &= \gamma_{11} \quad F_{v3} = \gamma_1 2
\end{aligned} \tag{A.4}$$

Gravitational torques

$$g(q) = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix}$$

$$\begin{aligned}
g_1 &= \gamma_4 sin(q_1) + \gamma_5 gsin(q_1 + q_2) + \gamma_6 gsin(q_1 + q_2 + q_3) \\
g_2 &= \gamma_5 gsin(q_1 + q_2) + \gamma_6 gsin(q_1 + q_2 + q_3) \\
g_3 &= \gamma_6 gsin(q_1 + q_2 + q_3)
\end{aligned} \tag{A.5}$$

Appendix B

Epson RC+ motion commands

In this Appendix, a list of useful functions available for motion control of the Epson VT6 industrial manipulator (Chapter 13) is presented. The functions are defined in the SPEL+ programming language for the proprietary software Epson RC+ 7.0.

Motion in the operative space

Functions to define motion of the robot's end-effector in the cartesian space, through points expressed in x, y, z coordinates with respect to the base reference frame.

Go

`Go {Point}`

Performs Point-to-Point motion from the current position the the specified point `Point`. Each joint interpolates independently, and the resulting path is not predictable. Velocity and acceleration of the robot can be specified with the `Speed` and `Accel` instructions.

Jump

`Jump3 {PDepart, PApproach, PDestination}`

Performs a 3D gate motion from the current position to a desired point, through three segments connected as in Figure B.1, where `PDepart, PApproach, PDestination` are pre-specified points.

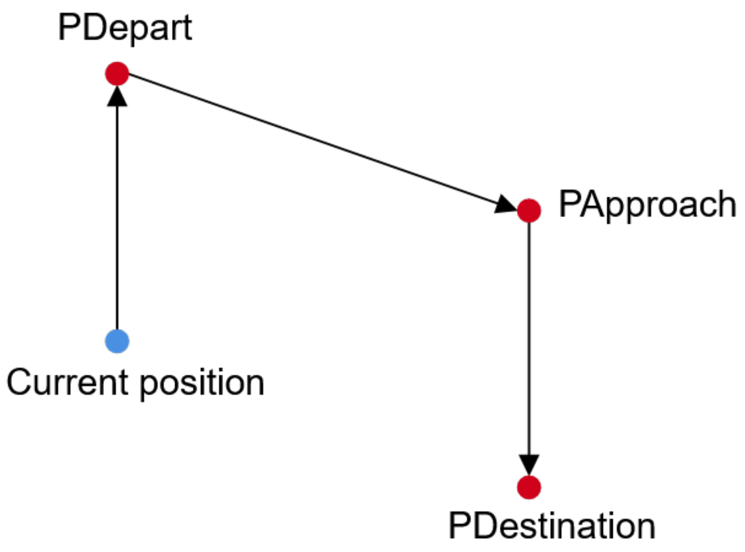


Figure B.1: 3D Gate motion

Pass

```
Pass {Point, [Point,...]}
```

Performs motion near (but not through) pre-specified points.

Move

```
Move {Point}
```

Perform motion from the current position to a pre-specified point using linear interpolation at constant velocity. Velocity and acceleration rates of the robot can be specified with the `SpeedS` and `AccelS` instructions.

Arc

```
Arc {PMiddle, PDestination}
```

Performs motion in a circular way from the current position to the desired point `PDestination`, passing through `PMiddle`. Velocity and acceleration rates of the robot can be specified with the `SpeedS` and `AccelS` instructions.

Continuous Path CP

```
CP {On | Off}
```

Enables continuous path. When activated, motion continues without deceleration between two consecutive motion commands.

Till

```
Till {Condition}
```

Specifies a condition that, if met, stops the robot's motion. It is applied directly after a motion statement.

Example

```
Till Condition = False;
Move Point Till;
```

Motion in the configuration space

Functions used to define motion of the robot directly through its joints. Target positions are expressed in degrees.

JTran

```
JTran {JointN, Dist}
```

Performs incremental motion of the selected joint to a pre-specified distance `Dist` from the current position.

JA

```
JA(Joint1, Joint2, Joint3,Joint4,Joint5,Joint6)
```

Returns the resulting point in the cartesian space for the defined joints positions. It computes the *forward kinematics* given a specific joint configuration. It can be used together with a function for motion in the operative space in order to perform motion in the configuration space, by directly providing joint references.

Example

```
Move JA(J1, J2, J3, J4, J5, J6)
```

Bibliography

[1] International Federation Of Robotics, “World robotics report 2020.” https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe.

[2] S. Haddadin, A. Albu-Schäffer, and G. Hirzinger, “Requirements for safe robots: Measurements, analysis and new insights,” *The International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1507–1527, 2009.

[3] A. Bicchi, M. A. Peshkin, and J. E. Colgate, *Safety for Physical Human–Robot Interaction*, pp. 1335–1348. Berlin, Heidelberg: Springer, 2008.

[4] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, “Safety in human-robot collaborative manufacturing environments: Metrics and control,” *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 882–893, 2016.

[5] L. Johannsmeier and S. Haddadin, “A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes,” *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 41–48, 2016.

[6] M. S. Branicky, V. S. Borkar, and S. K. Mitter, “A unified framework for hybrid control: Model and optimal control theory,” *IEEE Transactions on Automatic Control*, vol. 43, no. 1, pp. 31–45, 1998.

[7] J. P. Hespanha, D. Liberzon, and A. S. Morse, “Hysteresis-based switching algorithms for supervisory control of uncertain systems,” *Automatica*, vol. 39, no. 2, pp. 263–272, 2003.

[8] M. Zefran and J. W. Burdick, "Design of switching controllers for systems with changing dynamics," in *IEEE Conference on Decision and Control (CDC)*, vol. 2, pp. 2113–2118, 1998.

[9] D. S. Shah, J. P. Powers, L. G. Tilton, S. Kriegman, J. Bongard, and R. Kramer-Bottiglio, "A soft robot that adapts to environments through shape change," *Nature Machine Intelligence*, pp. 1–9, 2020.

[10] D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza, "The foldable drone: A morphing quadrotor that can squeeze and fly," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 209–216, 2018.

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

[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1–37, 2013.

[13] H. A. Pierson and M. S. Gashler, "Deep learning in robotics: a review of recent research," *Advanced Robotics*, vol. 31, no. 16, pp. 821–835, 2017.

[14] R. E. Bellman, *Dynamic Programming.* New York, NY, USA: Dover Publications, Inc., 2003.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

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

[17] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning (ICML)*, vol. 80, (Stockholmsmässan, Stockholm, Sweden), pp. 1582–1591, PMLR, 10–15 Jul 2018.

[18] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning (ICML)*, (New York, NY, USA), PMLR, June 2016.

[19] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4–5, pp. 421–436, 2017.

[20] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[21] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," in *Robotics: Science and Systems VII*, (Los Angeles, CL, USA), pp. 57–64, June 2011.

[22] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," 2017.

[23] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, (St. Paul, MN, USA), pp. 6244–6251, May 2018.

[24] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[25] B. Siciliano and O. Khatib, *Springer handbook of robotics.* Springer, 2016.

[26] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control.* Springer Science & Business Media, 2010.

[27] R. S. Hartenberg and J. Denavit, "A kinematic notation for lower pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 77, no. 2, pp. 215–221, 1955.

[28] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM Journal of Research and Development*, vol. 44, no. 1.2, pp. 206–226, 1959.

[29] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 2. Cambridge: MIT Press, 1998.

[30] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[31] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International Conference on Machine Learning (ICML)* (J. Dy and A. Krause, eds.), vol. 80, (Stockholmsmässan, Stockholm Sweden), pp. 1861–1870, PMLR, 10–15 Jul 2018.

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

[33] T. C. Hsia and L. S. Gao, “Robot manipulator control using decentralized linear time-invariant time-delayed joint controllers,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2070–2075, IEEE, 1990.

[34] T. Verstraten, R. Furnémont, P. López-García, D. Rodriguez-Cianca, B. Vanderborght, and D. Lefeber, “Kinematically redundant actuators, a solution for conflicting torque–speed requirements,” *The International Journal of Robotics Research*, pp. 612–629, 2018.

[35] A. Girard, *Fast and strong lightweight robots based on variable gear ratio actuators and control algorithms leveraging the natural dynamics.* PhD thesis, Massachusetts Institute of Technology, 2017.

[36] A. Girard and H. H. Asada, “Leveraging natural load dynamics with variable gear-ratio actuators,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 741–748, 2017.

[37] V. I. Utkin, *Sliding modes in control and optimization.* Springer Science & Business Media, 1992.

[38] V. Utkin, J. Guldner, and J. Shi, *Sliding mode control in electro-mechanical systems*, vol. 34. CRC press, 1999.

[39] C. Edwards and S. Spurgeon, *Sliding mode control: theory and applications.* CRC Press, 1998.

[40] J.-J. E. Slotine and S. S. Sastry, "Tracking control of non-linear systems using sliding surfaces, with application to robot manipulators," *International Journal of Control*, vol. 38, no. 2, pp. 465–492, 1983.

[41] J.-J. E. Slotine, "Sliding controller design for non-linear systems," *International Journal of Control*, vol. 40, no. 2, pp. 421–434, 1984.

[42] J.-J. E. Slotine and J. A. Coetsee, "Adaptive sliding controller synthesis for non-linear systems," *International Journal of Control*, vol. 43, no. 6, pp. 1631–1651, 1986.

[43] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control.* Prentice Hall, 1991.

[44] A. Levant, "Higher-order sliding modes, differentiation and output-feedback control," *International Journal of Control*, vol. 76, no. 9-10, pp. 924–941, 2003.

[45] G. Bartolini, A. Ferrara, E. Usai, and V. I. Utkin, "On multi-input chattering-free second-order sliding mode control," *IEEE Transactions on Automatic Control*, vol. 45, no. 9, pp. 1711–1717, 2000.

[46] G. Bartolini, A. Ferrara, and E. Usai, "Output tracking control of uncertain nonlinear second-order systems," *Automatica*, vol. 33, no. 12, pp. 2203–2212, 1997.

[47] F. Dinuzzo and A. Ferrara, "Higher order sliding mode controllers with optimal reaching," *IEEE Transactions on Automatic Control*, vol. 54, no. 9, pp. 2126–2136, 2009.

[48] A. Ferrara and L. Magnani, "Motion control of rigid robot manipulators via first and second order sliding modes," *Journal of Intelligent & Robotic Systems*, vol. 48, no. 1, pp. 23–36, 2007.

[49] L. M. Capisani, A. Ferrara, and L. Magnani, “Design and experimental validation of a second-order sliding-mode motion controller for robot manipulators,” *International Journal of Control*, vol. 82, no. 2, pp. 365–377, 2009.

[50] G. P. Incremona, M. Tanelli, M. Rubagotti, and A. Ferrara, “Switched third-order sliding mode control,” in *IEEE American Control Conference (ACC)*, (Boston, MA, USA), pp. 7189–7194, IEEE, July 2016.

[51] V. Utkin and J. Shi, “Integral sliding mode in systems operating under uncertainty conditions,” in *IEEE Conference on Decision and Control (CDC)*, vol. 4, (Kobe, Japan), pp. 4591–4596, IEEE, Dec. 1996.

[52] A. Ferrara, G. P. Incremona, and B. Sangiovanni, “Integral sliding mode based switched structure control scheme for robot manipulators,” in *International Workshop on Variable Structure Systems (VSS)*, (Graz, Austria), pp. 168–173, IEEE, July 2018.

[53] A. Ferrara, G. P. Incremona, and B. Sangiovanni, “Tracking control via switched integral sliding mode with application to robot manipulators,” *Control Engineering Practice*, vol. 90, pp. 257–266, 2019.

[54] B. Sangiovanni, G. P. Incremona, A. Ferrara, and M. Piastra, “Deep reinforcement learning based self-configuring integral sliding mode control scheme for robot manipulators,” in *IEEE Conference on Decision and Control (CDC)*, (Miami Beach, FL, USA), pp. 5969–5974, IEEE, Dec. 2018.

[55] A. Levant, “Chattering analysis,” *IEEE Transactions on Automatic Control*, vol. 55, pp. 1380–1389, June 2010.

[56] D. Liberzon, *Switching in Systems and Control.* Birkh auser Basel, 2003.

[57] A. Calanca, L. M. Capisani, A. Ferrara, and L. Magnani, “Mimo closed loop identification of an industrial robot,” *IEEE Transactions on Control Systems Technology*, vol. 19, no. 5, pp. 1214–1224, 2011.

[58] L. M. Capisani, A. Ferrara, and L. Magnani, “Mimo identification with optimal experiment design for rigid robot manipulators,” in *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pp. 1–6, 2007.

[59] G. P. Incremona, A. Saccon, A. Ferrara, and H. Nijmeijer, "Trajectory tracking of mechanical systems with unilateral constraints: Experimental results of a recently introduced hybrid PD feedback controller," in *IEEE Conference on Decision and Control (CDC)*, (Osaka, Japan), pp. 920–925, IEEE, Dec. 2015.

[60] J. Han, "From PID to active disturbance rejection control," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 3, pp. 900–906, 2009.

[61] A. De Luca and F. Flacco, "Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration," in *IEEE RAS & EMBS IInternational Conference on Biorobotics and Biomechanics*, (Rome, Italy), pp. 288–295, IEEE, June 2012.

[62] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *IEEE International Conference on Robotics and Automation (ICRA)*, (St Paul, MN, USA), pp. 338–345, May 2012.

[63] S. Haddadin, A. De Luca, and A. Albu-Schäffer, "Robot collisions: A survey on detection, isolation, and identification," *IEEE Transactions on Robotics*, vol. 33, no. 6, pp. 1292–1312, 2017.

[64] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[65] C. W. Warren, "Global path planning using artificial potential fields," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, (Scottsdale, AZ, USA), pp. 316–321, May 1989.

[66] P. Ogren, L. Petersson, M. Egerstedt, and X. Hu, "Reactive mobile manipulation using dynamic trajectory tracking: design and implementation," in *IEEE Conference on Decision and Control (CDC)*, vol. 3, (San Francisco, CA, USA), pp. 3001–3006, Dec. 2000.

[67] M. Parigi Polverini, A. M. Zanchettin, and P. Rocco, "Real-time collision avoidance in human-robot interaction based on kinetostatic safety field," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4136–4141, 2014.

[68] L. M. Capisani, T. Facchinetti, A. Ferrara, and A. Martinelli, "Obstacle modelling oriented to safe motion planning and control for planar rigid robot manipulators," *The Journal of Intelligent and Robotic Systems*, vol. 71, no. 2, pp. 159–178, 2013.

[69] L. Balan and G. M. Bone, "Real-time 3d collision avoidance method for safe human and robot coexistence," in *IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, (Beijing, PRC), pp. 276–282, Oct. 2006.

[70] M. Ragaglia, A. M. Zanchettin, and P. Rocco, "Trajectory generation algorithm for safe human-robot collaboration based on multiple depth sensor measurements," *Mechatronics*, vol. 55, pp. 267–281, 2018.

[71] L. Rozo, D. Bruno, S. Calinon, and D. G. Caldwell, "Learning optimal controllers in human-robot cooperative transportation tasks with position and force constraints," in *IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, (Hamburg, Germany), pp. 1024–1030, Sept. 2015.

[72] Y. Li, K. P. Tee, R. Yan, W. L. Chan, and Y. Wu, "A framework of human–robot coordination based on game theory and policy iteration," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1408–1418, 2016.

[73] Y. Wang, Y. Sheng, J. Wang, and W. Zhang, "Optimal collision-free robot trajectory generation based on time series prediction of human motion," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 226–233, 2018.

[74] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra, "Deep reinforcement learning for collision avoidance of robotic manipulators," in *European Control Conference*, (Lymassol, Cyprus), pp. 2063–2068, July 2018.

[75] B. Sangiovanni, G. P. Incremona, M. Piastra, and A. Ferrara, "Self-configuring robot path planning with obstacle avoidance via deep reinforcement learning," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 397–402, 2021.

[76] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: a versatile and scalable robot simulation framework," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, (Tokyo, Japan), pp. 1321–1326, Nov. 2013.

[77] S. James, M. Freese, and A. J. Davison, “Pyrep: Bringing V-REP to deep robot learning,” *arXiv preprint arXiv:1906.11176*, 2019.

[78] G. Sánchez and J.-C. Latombe, *A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking*, pp. 403–417. Berlin, Heidelberg: Springer, 2003.

[79] J. Meijer, Q. Lei, and M. Wisse, “An empirical study of single-query motion planning for grasp execution,” in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, (Munich, Germany), pp. 1234–1241, July 2017.

[80] P. F. Hokayem and M. W. Spong, “Bilateral teleoperation: An historical survey,” *Automatica*, vol. 42, no. 12, pp. 2035–2057, 2006.

[81] J. Luo, W. He, and C. Yang, “Combined perception, control, and learning for teleoperation: key technologies, applications, and challenges,” *Cognitive Computation and Systems*, vol. 2, no. 2, pp. 33–43, 2020.

[82] J. Vertut, *Teleoperation and robotics: applications and technology.* Springer Science & Business Media, 2013.

[83] A. Flores-Abad, O. Ma, K. Pham, and S. Ulrich, “A review of space robotics technologies for on-orbit servicing,” *Progress in Aerospace Sciences*, vol. 68, pp. 1–26, 2014.

[84] R. H. Taylor, A. Menciassi, G. Fichtinger, P. Fiorini, and P. Dario, “Medical robotics and computer-integrated surgery,” in *Springer handbook of robotics*, pp. 1657–1684, 2016.

[85] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, “Deep imitation learning for complex manipulation tasks from virtual reality teleoperation,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[86] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, “Survey of imitation learning for robotic manipulation,” *International Journal of Intelligent Robotics and Applications*, pp. 1–8, 2019.

[87] P. Owan, J. Garbini, and S. Devasia, “Faster confined space manufacturing teleoperation through dynamic autonomy with task dynamics imitation learning,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2357–2364, 2020.

[88] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,” in *Robotics: Science and Systems (RSS)*, (Pittsburgh, PA, USA), June 2018.

[89] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, “Surreal: Open-source reinforcement learning framework and robot manipulation benchmark,” in *Proc. Conference on Robot Learning*, pp. 767–782, 2018.

[90] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, “Rearrangement with nonprehensile manipulation using deep reinforcement learning,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[91] M. Rubagotti, T. Taunyazov, B. Omarali, and A. Shintemirov, “Semi-autonomous robot teleoperation with obstacle avoidance via model predictive control,” *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2746–2753, 2019.

[92] A. Shintemirov, T. Taunyazov, B. Omarali, A. Nurbayeva, A. Kim, A. Bukeyev, and M. Rubagotti, “An open-source 7-DOF wireless human arm motion-tracking system for use in robotics research,” *Sensors*, vol. 20, no. 11, p. 3082, 2020.

[93] B. Houska, H. J. Ferreau, and M. Diehl, “ACADO toolkit - an open-source framework for automatic control and dynamic optimization,” *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[94] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, “qpOASES: A parametric active-set algorithm for quadratic programming,” *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[95] Epson Robotics, “Epson VT6.” https://www.epson.eu/products/robot/epson-6-axis-vt6-a901s-with-built-in-controller.