Pontifícia Universidade Católica do Rio de Janeiro

Franklin Cardeñoso Fernández

Deep reinforcement learning for haptic shared control in unknown tasks

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Engenharia Elétrica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica.

Advisor: Prof. Wouter Caarls

Rio de Janeiro
April 2020

Pontifícia Universidade Católica
do Rio de Janeiro

Franklin Cardeñoso Fernández

Deep reinforcement learning for haptic shared control in unknown tasks

Dissertation presented to the Programa de Pós–graduação em Engenharia Elétrica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica. Approved by the Examination Committee:

Prof. Wouter Caarls
Advisor
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Antonio Candea Leite
NMBU

Prof. Helon Vicente Hultmann Ayala
Departamento de Engenharia Mecânica – PUC-Rio

Prof. Karla Tereza Figueiredo Leite
UERJ

Rio de Janeiro, April the 1st, 2020

Franklin Cardeñoso Fernández

Graduated in electronic engineering by the Universidad Nacional de San Antonio Abad del Cusco (Cusco, Perú) in 2016.

Bibliographic data

Cardeñoso Fernandez, Franklin

Deep reinforcement learning for haptic shared control in unknown tasks / Franklin Cardeñoso Fernández; advisor: Wouter Caarls. – 2020.

123 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2020.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Teleoperação;. 3. Controle compartilhado;. 4. Aprendizado por reforço.. I. Caarls, Wouter. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD: 621.3

To my parents, for being the most important part of my life.

Acknowledgments

To my advisor Prof. Wouter Caarls for his patience, guidance, and support, I will always truly grateful.

To all the professors in the postgraduate program, for their great advice in the courses and talks.

To my family for their moral support, and my girlfriend Paola, for all her love and encouragement to me.

To my colleagues from the Intelligent Control Laboratory and my friends for all the experiences and discussions.

To CAPES and PUC-Rio, for the aids granted, without which this work could not have been accomplished.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Cardeñoso Fernandez, Franklin; Caarls, Wouter (Advisor). **Deep reinforcement learning for haptic shared control in unknown tasks**. Rio de Janeiro, 2020. 123p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Recent years have shown a growing interest in using haptic shared control (HSC) in teleoperated systems. In HSC, the application of virtual guiding forces decreases the user's control effort and improves execution time in various tasks, presenting a good alternative in comparison with direct teleoperation. HSC, despite demonstrating good performance, opens a new gap: how to design the guiding forces. For this reason, the real challenge lies in developing controllers to provide the virtual guiding forces, able to deal with new situations that appear while a task is being performed. This work addresses this challenge by designing a controller based on the deep deterministic policy gradient (DDPG) algorithm to provide the assistance, and a convolutional neural network (CNN) to perform the task detection. The agent learns to minimize the time it takes the human to execute the desired task, while simultaneously minimizing their resistance to the provided feedback. This resistance thus provides the learning algorithm with information about which direction the human is trying to follow, in this case, the *pick-and-place* task. Diverse results demonstrate the successful application of the proposed approach by learning custom policies for each user who was asked to test the system. It exhibits stable convergence and aids the user in completing the task with the least amount of steps possible.

Keywords

Teleoperation;; Shared control;; Reinforcement learning..

Resumo

Cardeñoso Fernandez, Franklin; Caarls, Wouter. **Aprendizado por reforço profundo para controle háptico compartilhado em tarefas desconhecidas**. Rio de Janeiro, 2020. 123p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Os últimos anos mostraram um interesse crescente no uso do controle háptico compartilhado (HSC) em sistemas teleoperados. No HSC, a aplicação de forças orientadoras virtuais, diminui o esforço de controle do usuário e melhora o tempo de execução em várias tarefas, apresentando uma boa alternativa em comparação com a teleoperação direta. O HSC, apesar de demonstrar bom desempenho, abre uma nova lacuna: como disenhar as forças orientadoras. Por esse motivo, o verdadeiro desafio está no desenvolvimento de controladores para fornecer as forças orientadoras virtuais, capazes de lidar com novas situações que aparecem enquanto uma tarefa está sendo executada. Este trabalho aborda esse desafio, projetando um controlador baseado no algoritmo *Deep Deterministic Policy Gradient* (DDPG) para fornecer assistência, e uma rede neural convolucional (CNN) para executar a detecção da tarefa. O agente aprende a minimizar o tempo que o ser humano leva para executar a tarefa desejada, minimizando simultaneamente sua resistência ao *feedback* fornecido. Essa resistência fornece ao algoritmo de aprendizado informações sobre a direção que o humano está tentando seguir, neste caso na tarefa *pick-and-place*. Diversos resultados demonstram a aplicação bem-sucedida da abordagem proposta, aprendendo políticas personalizadas para cada usuário que foi solicitado a testar o sistema. Ele exibe convergência estável e ajuda o usuário a concluir a tarefa com o menor número possível de etapas.

Palavras-chave

Teleoperação;; Controle compartilhado;; Aprendizado por reforço..

Table of contents

List of figures

List of tables

List of Algorithms

List of Abreviations

ANN	Artificial Neural Network
BC	Behavioral Cloning
BCI	Brain-computer interfaces
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DOF	Degrees of freedom
DRL	Deep Reinforcement Learning
E2E	End-to-end learning
HRI	Human-Robot Interaction
HSC	Haptic Shared Control
IRL	Inverse Reinforcement Learning
LfD	Learning from Demonstration
MDP	Markov Decision Process
ML	Machine Learning
MTS	Manual task selection
PbD	Programming by Demonstration
RL	Reinforcement Learning
SC	Shared control
SSTD	Single-shot task detection
STD	Supervised task detection

List of symbols

Bold and lower case letters are used for vectors. Matrices are bold capitals. Hat symbol is used for approximations.

$\mathcal{D}$	Available dataset
ω	Scale factor
$\mathcal{S}$	Set of states
$\mathcal{A}$	Set of actions
$p(.)$	Probability distribution
R_t	Return
$r(\boldsymbol{s}, \boldsymbol{a})$	Reward function
π	Policy
π^*	Optimal policy
π^{E}	Expert policy
$\mathbb{E}[.]$	Expectation
$\boldsymbol{s}$	States vector
$\boldsymbol{a}$	Actions vector
γ	Discount factor
$V^{\pi}(\boldsymbol{s})$	State-value function
$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$	Action-value function
$V^*(\boldsymbol{s})$	Optimal state-value function
$Q^*(\boldsymbol{s}, \boldsymbol{a})$	Optimal action-value function
$\boldsymbol{\theta}$	Parameter vector of actor (policy)
$\boldsymbol{w}$	Parameter vector of critic (value function)
$\rho^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s})$	State distribution of policy π_θ
$J(.)$	Objective function

$\hat{\pi}(\boldsymbol{a}\vert\boldsymbol{s};\boldsymbol{\theta})$	Stochastic parameterized policy
$\hat{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{w})$	Parameterized action-value function
$\hat{Q}(.)$	Approximated action-value function
μ	Deterministic policy
$\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})$	Deterministic parametrized policy
τ	Update target parameter
p	AR-p order
α	AR-p paramete
$\boldsymbol{p}$	Position vector
$\dot{\boldsymbol{p}}$	Velocity vector
$\boldsymbol{p}_{\mathrm{T}}$	Haptic device position vector
$\boldsymbol{p}_{\mathrm{D}}$	Manipulator position vector
$\tilde{\boldsymbol{p}}_{\mathrm{T}}$	Scaled haptic device position vector
$\tilde{\boldsymbol{p}}_{\mathrm{D}}$	Scaled manipulator position vector
$\boldsymbol{f}$	Forces vector
$\mathcal{R}$	Replay buffer memory

Platero es pequeño, peludo, suave; tan blando por fuera, que se diría todo de algodón, que no lleva huesos. Sólo los espejos de azabache de sus ojos son duros cual dos escarabajos de cristal negro. Lo dejo suelto y se va al prado y acaricia tibiamente con su hocico, rozándolas apenas, las florecillas rosas, celestes y gualdas... Lo llamo dulcemente: ¿Platero?, y viene a mí con un trotecillo alegre, que parece que se ríe, en no sé qué cascabeleo ideal...

Julio Ramón Ribeyro, *Platero y yo.*

1
INTRODUCTION

1.1
Motivation

Robotic systems exist everywhere. Because of their wide diversity, robots can be used for all kinds of applications. As a result, the coexistence between humans and robots has been growing in recent years and consequently, the necessity to develop human-robot collaboration to *share control* to perform different tasks [1]. A common example of this collaboration are teleoperated systems, where users control robots placed in remote locations. Thereby, the shared control (SC) approach becomes a very useful tool in teleoperated systems combining the most powerful features of humans and robots for situations where humans cannot interact directly with the environment.

SC in teleoperation is not a new topic, earlier approaches demonstrated its performance and efficiency by applying optimal control and potential fields comparing direct and assistive control [2], learning how to modulate the control between the human and the robot semi autonomously as presented in [3] or learning how to produce forces to interact with unknown objects with different stiffness in virtual environments [4]. However, despite the fact that SC for teleoperation was widely addressed, it has not yet been *solved* in terms of optimizing the combined human-robot system's performance.

For many years this issue was addressed targeting complete transparency of the teleoperation system. However, despite all the research performed in this area, unfortunately, optimal transparency has not been achieved yet. In this context, later research has shown that user performance is improved by *decreasing* the transparency level and increasing the force feedback through the application of forces in the input device to guide the user movements. This decreases the user's control effort and improves the execution time to complete the task, obtaining in this way a new level of shared control known as Haptic Shared Control (HSC) [5], [6], [7]. This kind of control, despite demonstrating good performance in teleoperated systems providing haptic guidance [8], [9], opens a new gap: how to design the guiding forces.

Moreover, although HSC has been sucessfully tested in a wide diversity of tasks, such as: peg-in-hole [10], grasping objects [11], page turning task [12], home-service tasks [13], bolt-spanner task [5], [6] or achieve goals given a trajectory distribution [14], [15] and [16], [17]. It is possible to see that all of these controllers are designed to deal with fixed tasks or explicit information. However it is known that real-world tasks are not always presented with fixed goals or trajectories, for this reason, the real challenge lies in developing controllers to provide the guiding forces and at the same time able to deal with new situations that appear while a task is being performed [18], [19].

Although linear and nonlinear control-based controllers present stable functionality in HSC for teleoperation as is described in [8] where the controller allows avoiding obstacles in the slave side or [20] where trajectories for mobile robots are generated; it is necessary to have previous knowledge of the system model and dynamics. In the context of this thesis, the designed controllers should be able to handle unknown tasks as similar as humans do without much knowledge of the system. This kind of behavior is addressed by machine learning (ML) algorithms where performance is measured by the capability of abstraction and generalization.

In addition, to deal with situations where the task is unknown, it is desirable to implicitly give the controller some information about the task we are trying to do, so that the controller is able to compute the optimal behavior to complete the proposed task without explicit commands. Therefore, the algorithm only needs to be supplied with enough information about the task and with a learning function that reflects in a general way the user intention. This intention can be inferred from the user's actions, or implicitly observing the environment and user behavior visually.

Thereby, images can be passed to the learning controller giving implicit information about the task intention. That kind of information can be delivered to the controller directly or previously pre-processed with computer vision techniques. To achieve generalization in the image processing, Convolutional Neural Networks (CNN) are a good option, to provide the controller with the relevant features of the task.

In addition, given that the real challenge remains in designing a controller with enough adaptability to assist the user it is necessary to choose an approach able to provide this feature. Reinforcement Learning (RL) is an interesting option because it brings us the possibility to *teach* the controller what our intention is through trial and error interaction. The general learning rules are encoded in the *reward function* which is used as a metric to measure the performance of the controller during the training. Thus, learning the system model is not necessary (such as in control engineering techniques) and we only need to supply the algorithm with the observations composed of relevant information about the current states every time step the algorithm is executed. These observations about the task intention can be supplied in different ways, in some cases manually. Alternatively, that information can be provided using a camera, sending visual information which can be processed through the combination of RL with Deep Learning (DL) methods for computer vision applications termed as Deep Reinforcement Learning (DRL), becoming a powerful method taking as inputs information composed by measurements and images, and giving as outputs the needed assistance with the guide forces to complete the task (See Figure 1.1).

1.2 Objectives

1.2.1 General Objective

According all exposed, the main objective in this research is to develop a Haptic Shared Control (HSC) controller composed by a control-based section to perform the direct teleoperation and an RL-based section that learns the guiding forces according the user preferences and at the same time performs the task decoding from visual information.

1.2.2 Specific Objectives

The specific objectives of this work are the following:

1. Implement a master and slave side simulator to perform hyperparameter tuning and preliminary tests.

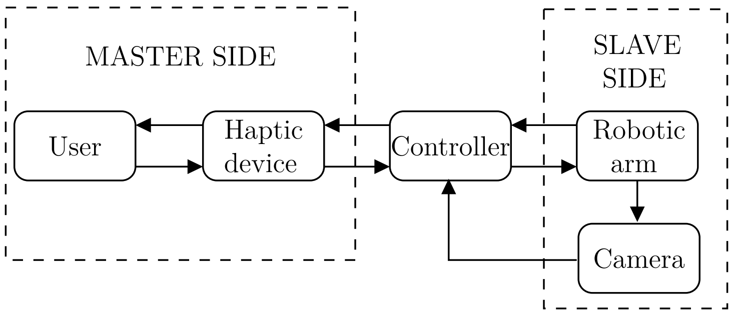


Figure 1.1: Overview of the system's architecture: A user on the master side interacts with a robotic arm placed on the slave side through a haptic device. On the slave side the information about the performed task is captured by a camera. Then, all the information is processed by a central controller.

2. Build an HSC controller that performs direct teleoperation from numerical information and task detection from visual information.
3. Design a reward function that influences the learning of the RL agent to provide the optimal guiding forces.
4. Analyze different approaches to select the best approach in terms of performance and stability.
5. Evaluate the performance of the developed controller in different subjects.

1.3 Contributions

As main contributions of this research we can consider:

– The design of a simulator for the teleoperated system from recorded data for preliminary tests.

– The design of a reward function based on user resistance to avoid dependency on predefined end-points or trajectories.

– A visual task detector that learns any visually distinguishable task.

1.4 Organization of the remaining parts of this thesis

The remainder of this document is as follows:

Chapter 2 presents all the theoretical foundations related to the field of teleoperated robotic systems, as well as the Shared Control approach and its variant known as Haptic Shared Control[5], [6], [7]. Next, are exposed some intelligent controllers used in robot learning and focusing on Reinforcement Learning and the policy gradient algorithm DDPG, as well as autocorrelated noise like ARP-noise and Deep Learning methods for computer vision applications. These concepts give the reader enough information to understand the implemented system.

Chapter 3 introduces an overall description of the HSC-controller describing the chosen states, the designed reward functions, and describing the proposed network architectures. In addition a simulator is presented, which serves as testing object to test the different parameter and hyperparameter configurations, as well as the variations in the reward function in order to improve performance.

Chapter 4 describes the experimental setup giving more details about the different parts of the system, physical restrictions and controller implementations. In addition, the different considerations used in order to perform the tests and experiments are explained.

Chapter 5 presents the preliminary tests performed with the simulator in order to validate the application of the HSC controller, before the final mounting of the system and its experimental evaluation. Moreover, the algorithms used, chosen parameters and the network architectures for the different variations are presented. Resulting plots for all the implementations are also shown followed by a brief discussion of the obtained results.

Chapter 6 details the experiments carried out in a real system validating the obtained results in simulations. Then, the different variations of the proposed method are introduced, combining the CNN as a task detector and the DDPG algorithm as central controller. Similar to chapter 5, resulting curves for all the implementations are shown followed by a brief discussion of the obtained results. Then, further experiments are presented to validate the effectiveness in different subjects for the proposed approach as well as a further discussion for all the combinations.

Finally, in chapter 7 the research is summarized, presenting the conclusions of this research and giving guidelines for further implementations in order to improve the proposed method.

2 BACKGROUND

In this chapter, the fundamental theory for a proper understanding of the proposed system is provided. We will start by describing the basis for teleoperation applied in robotics as well as its types. Then, the basic concept of haptic shared control and a brief description of the functioning of haptic devices will be presented. Next, different intelligent controllers for robot learning, focusing on the Reinforcement Learning approach will be described, including their functioning, types and methods. Next, the Deep Deterministic Policy Gradient algorithm and the Autorregressive process as exploration noise function are presented. Finally, we will explain the fundamentals of Deep Learning as well as transfer learning method used to improve pre-trained networks.

2.1 Telerobotics

Telerobotics is an area in robotics initially designed for scenarios where human-environment interaction is not directly possible. For instance hazardous environments, sub-sea, etc. Instead, robots are remotely controlled by operators to perform the desired task. A common implementation is presented in Figure 2.1, where an operator, located on the master side, sends control signals to a robot, placed on the slave-side. To do this, the operator uses input devices to send the orders and output devices to receive information about the actual state of the robot. In this way, the robot executes the received orders through its actuators and returns measured information from its sensors.

2.1.1 Robot control

When teleoperation is performed, the slave tries to mimic the master performing similar movements during the system operation. This is known as position control. According to the kinematics similarity of the input device and the controlled robot, the position control can be performed in the joint level or tip level [21].

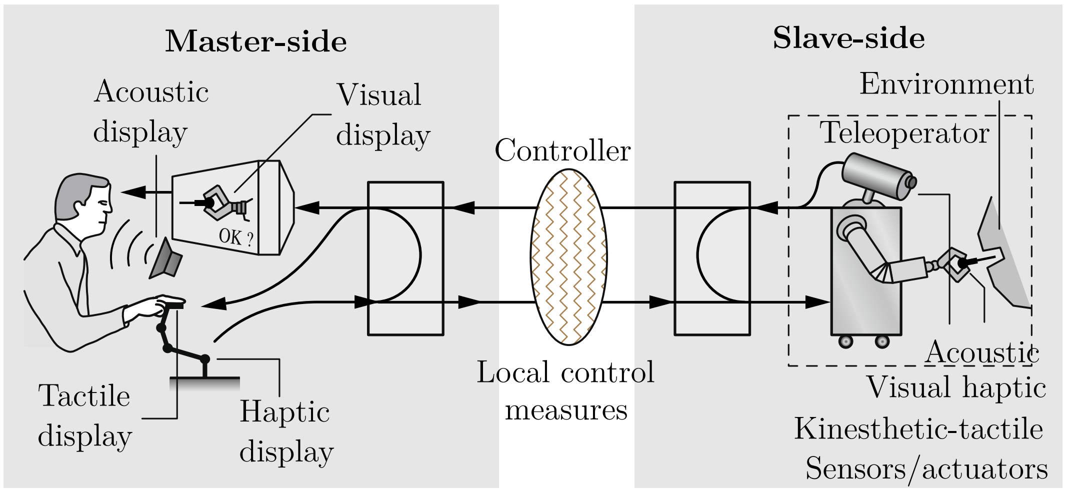


Figure 2.1: Robot control in a teleoperated scenery. Source: adapted from [21]

2.1.1.1 Position control in joint level

If the master and slave devices have a kinematically equivalent mechanism if not entirely identical (for example same number of joints, same configuration, same link lengths, etc) is said that both devices are kinematically similar [21]. In this case, the position control for both devices is performed on the joint level. Therefore, the slave desired position is given by the following equation:

$$\boldsymbol{q}_{\mathrm{Sd}} = \boldsymbol{q}_{\mathrm{M}} + \boldsymbol{q}_{\mathrm{offset}} \, , \tag{2-1}$$

where $\boldsymbol{q}$ denotes the joint coordinates vector, the sub-index S denotes the slave device, the sub-index M denotes the master device and the sub-index d denotes the desired value. The sub-index offset denotes the value to couple both devices.

Assuming that both devices are coupled, the position control is done maintaining the joint position error between the desired position and the actual position on the slave side as zero; that is:

$$\boldsymbol{q}_{\mathrm{Sd}} - \boldsymbol{q}_{\mathrm{S}} \to 0 \, . \tag{2-2}$$

2.1.1.2 Position control in tip level

On the other hand, when the mechanisms are kinematically dissimilar (for example different number of joints, different robot configuration, different link lengths, etc.), the position control is performed on the tip level [21]. The used equations are similar to the last case with the difference that it is necessary apply some modifications to perform the control:

$$\boldsymbol{p}_{\mathrm{Sd}} = \boldsymbol{p}_{\mathrm{M}} + \boldsymbol{p}_{\mathrm{offset}}\,, \tag{2-3}$$

where $\boldsymbol{p}$ denotes the end-effector coordinates vector, the sub-index S denotes the slave device, the sub-index M denotes the master device and the sub-index d denotes the desired value. The sub-index offset denotes the value to couple both devices.

In addition, because of the differences in their mechanisms, the workspace could be slightly different. Therefore, it is necessary map one workspace into another by applying a scaling factor ξ:

$$\boldsymbol{p}_{\mathrm{Sd}} = \xi\boldsymbol{p}_{\mathrm{M}} + \boldsymbol{p}_{\mathrm{offset}}\,. \tag{2-4}$$

To improve operator's performance, as recommends [21], the ξ parameter value should be selected to map both workspaces as close as possible or according the user comfort.

Similar to last section, if both devices are coupled, then position control is done maintaining the tip position error between the desired position and the actual position on the slave side as zero, that is:

$$\boldsymbol{p}_{\mathrm{Sd}} - \boldsymbol{p}_{\mathrm{S}} \to 0\,. \tag{2-5}$$

2.1.2 Control configuration

Another important topic in telerobotics is the kind of control configuration of the system. In this context, three kinds of control architectures can be recognized according the control type of the system in order of increasing autonomy [21]:

- Direct control.
- Shared control.
- Supervisory control.

2.1.2.1 Direct control

Direct control (also called direct teleoperation) is related to systems where there is no existence of intelligent control in the master and the slave side. As well as the nonexistence of transparency on the master side. Therefore, the user commands are directly sent to the robot without any kind of force feedback to help to the user control. The functionality of this control is limited as well as its applications, a well known example is the remote control of unmanned vehicles [22].

2.1.2.2
Shared control

Shared control in contrast, combines the human control with an intelligent controller or increasing the transparency level in order to assist the user in the task completion. In this way, the user's performance is improved through sensory feedback such as the application of virtual fixtures [23] or semi-autonomous functionality such as avoiding accidental drops in grasping tasks [24].

2.1.2.3
Supervisory control

Finally, supervisory control takes more advantage of autonomous controllers using the human only as supervisor. Thereby, the operator supervises the robot during its functioning providing directives to the robot when they are required in order to complete the given task such as telesensor programming in space applications [25].

2.1.3
Telerobotics applications

Telerobotics has been successfully applied with a wide range of input devices. For instance, from simple devices as computer keyboards [27], passing

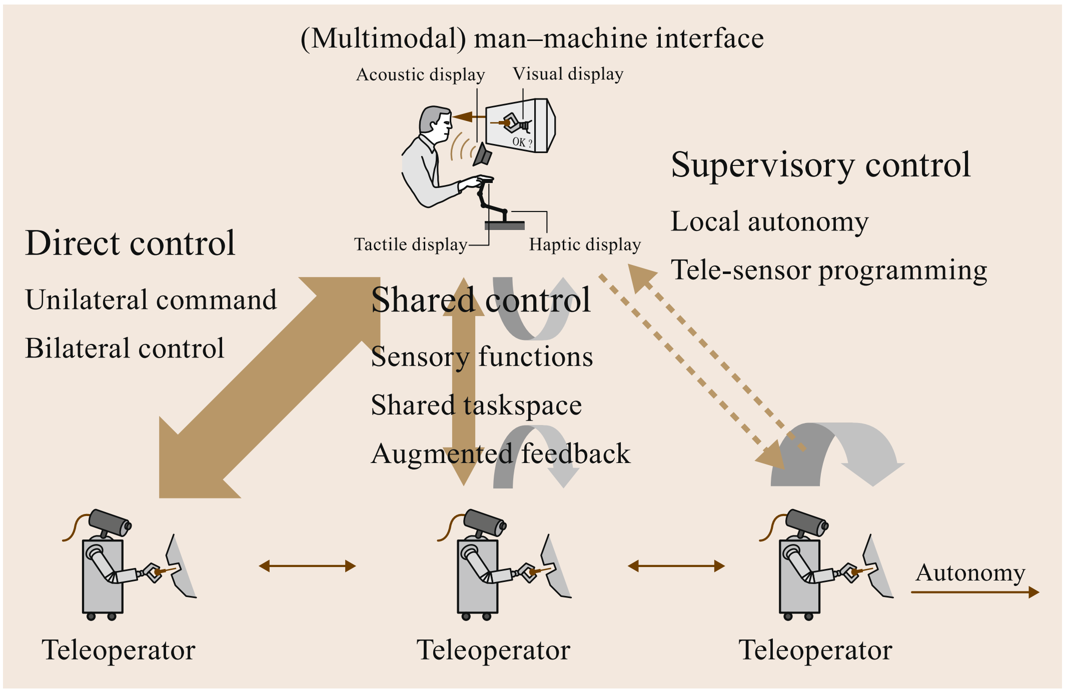


Figure 2.2: Control configuration classification. Source: taken from [60]

to smart devices, for instance mobile phones [13] or VR devices [10]; to more complex input devices as intracortical brain-computer interfaces (BCI) [28]. Although teleoperated systems can be controlled directly with only visual feedback as demonstrated in [29] and [12], performance can be improved by adding another kind of feedback. For instance, force feedback, which is often used to provide transparency or touch sensations to the operators. Taking advantage of this type of feedback, providing assistance by processing information about the remote environment is a natural next step. In this way, assistance can be performed using error potentials [2], through visual-audio signals illustrating restrictions [30], or using pure visual-force feedback [4], [3].

2.2 Haptic Shared Control

Haptics is a word believed to be derived from the Greek word *haptesthai*, which means *related to the sense of touch* [31]. In this context, Haptic Shared Control (HSC) is a sub-area in telerobotics where teleoperated systems use force feedback in the master side as the principal feedback by using haptic interfaces as input device. The main difference with other controllers that also use force feedback is that HSC does not try to increase the transparency level of the system. Instead, the controller takes advantage of the feedback decreasing the transparency level, in order to design *virtual guiding forces* in the input devices. These virtual forces allow decreasing the control effort of the operator to complete the task improving his performance as demonstrated in [5], [6] and [7].

2.2.1 Haptic devices

As was mentioned, the particularity in HSC is that the operator uses a haptic interface like the one shown in Figure 2.3 as input on the master side to perform the teleoperation. This kind of devices is able to provide touch sensations in human operators enhancing the touch experience by rendering virtual remote environments. In addition, haptic devices allow the operator to make desired motions because they are equipped with enough degrees of freedom to perform the different movements without difficulty.

The basic functionality of these devices is as follow: first, sense the operator commands with their input sensors, encoders for example. Next, this information is processed by a control program. And finally, the output is rendered with the actuators in the device to provide tactile sensations to the operator. Two broad classes of haptic devices exist: *admittance* or *impedance*

Figure 2.3: The Touch Haptic Device from 3D Systems. Source: taken from [32]

type. The difference between these classes lies in the input information that these are able to process. While the *admittance* devices sense the applied operator force, the *impedance* devices read the tip position.

Haptic devices have been tested in several implementations, providing haptic sensations to robots [33] and teleoperated systems improving performance as demonstrated in [34], or covering a wide range of applications assisting in task completion [8], [9], helping to generate trajectories [20], [35], or guiding to follow a reference point or trajectory [14], [15], [16], [17].

2.3 Robot learning

In the future, apart from repeatedly executing the same task thousands of times, robots will be faced with different tasks that rarely repeat in an ever changing environment. To achieve this behavior, robots will need to *learn* by themselves or with the help of humans. This problem is addressed by the field of Machine Learning (ML), which allows machines to gain a high level of abstraction and generalization. In this context, robot learning consists of a multitude of ML approaches used in robotics covering a wide range of topics such as: learning control and behavior generation, state abstraction, decision making, etc [60]. In this project, the robot learning approach was particularly focused on learning action generation and control as will be explained in later sections.

As is presented in Figure 2.4, the two main branches of robot learning for action generation and control use two key ingredients: the available data $\mathcal{D}$, generated from an expert or by interaction, and the learning approach that learns from the data. The data sets $\mathcal{D}$ are usually composed by samples $\mathcal{D}_i$ which contain information about states $\boldsymbol{s}_i$, actions $\boldsymbol{a}_i$, next states $\boldsymbol{s}'_i$ and depending on the approach, rewards r_i.

2.3.1 Model learning

Figure 2.4 presents the model learning approach as part of the main branch. Models are a very important part of robotics because they describe the main features of the system and allow to predict the behavior and the influence of the agent in the environment. Although classical methods like control engineering techniques have been applied to manually generate mathematical models, a large diversity of ML approaches are able to learn models from available datasets. Model learning is mainly used in robot control to design their control laws according the different situations. For instance, to perform accurate control without damaging the robot, to describe non-linear behavior of robotic components or to interact with unknown environments learning models online.

2.3.2 Reinforcement learning approaches

In contrast with model learning approach, reinforcement learning (RL) employs the available data directly to find the optimal policy π^* which guides the agent to take the action a that will maximize the expected future rewards given a certain state s. In contrast with other ML approaches, RL uses rewards

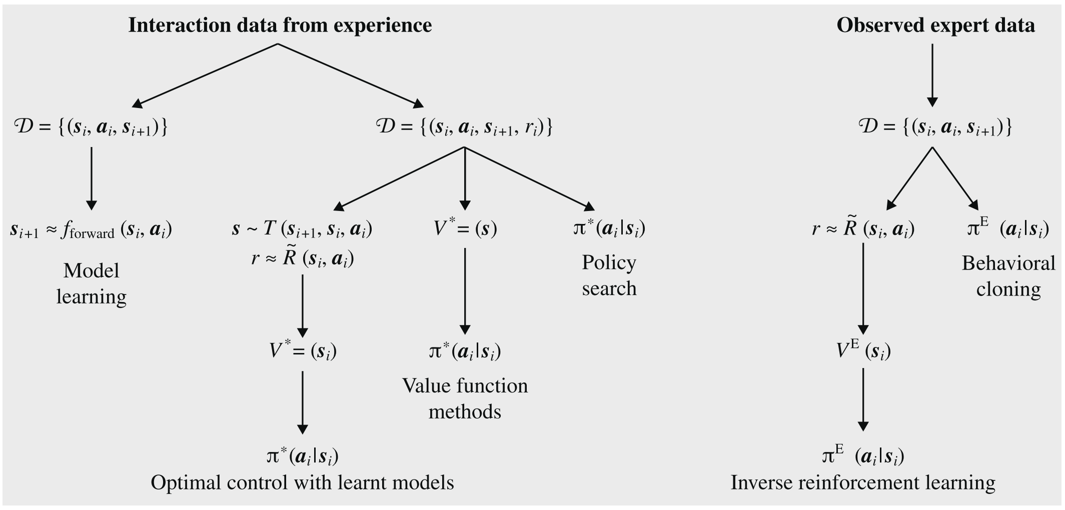

Interaction data from experience
Observed expert data
$\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}_{i+1})\}$
$\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}_{i+1}, r_i)\}$
$\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}_{i+1})\}$
$\boldsymbol{s}_{i+1} \approx f_{\text{forward}}(\boldsymbol{s}_i, \boldsymbol{a}_i)$
Model learning
$\boldsymbol{s} \sim T(\boldsymbol{s}_{i+1}, \boldsymbol{s}_i, \boldsymbol{a}_i)$
$r \approx \tilde{R}(\boldsymbol{s}_i, \boldsymbol{a}_i)$
$V^* = (\boldsymbol{s})$
$\pi^*(\boldsymbol{a}_i|\boldsymbol{s}_i)$
Policy search
$r \approx \tilde{R}(\boldsymbol{s}_i, \boldsymbol{a}_i)$
$\pi^E(\boldsymbol{a}_i|\boldsymbol{s}_i)$
Behavioral cloning
$V^* = (\boldsymbol{s}_i)$
$\pi^*(\boldsymbol{a}_i|\boldsymbol{s}_i)$
Value function methods
$V^E(\boldsymbol{s}_i)$
$\pi^*(\boldsymbol{a}_i|\boldsymbol{s}_i)$
Optimal control with learnt models
$\pi^E(\boldsymbol{a}_i|\boldsymbol{s}_i)$
Inverse reinforcement learning

Figure 2.4: Main branches of robot learning classification. Source: adapted from [60]

as feedback to obtain the optimal policy π^*. Optimal control with learnt models are model-based methods that learn a model from which they subsequently learn an optimal value function and policy. Instead, model-free methods learn a control policy without model knowledge either through a value function or by directly optimizing the policy with the available data.

2.3.3 Inverse Reinforcement Learning and Behavioral cloning

One of the most difficult parts, when RL controllers are designed, is to choose a reward function that is suitable for the given task. Inverse Reinforcement Learning (IRL) addresses this problem by reconstructing the reward function from expert's demonstrations in order to reproduce observed behavior π^{E} [60].

On the other hand, behavioral cloning (BC) is a method that can be treated as supervised learning because it tries to produce control rules that can clone the skills of an expert human operator given the respective dataset. Thereby, the system only tries to *replicate* the given demonstrations learning a policy that directly maps from the input to the action [36].

BC and IRL form two major classes of Learning from Demonstration (LfD) methods. LfD, also called Programming by Demonstration (PbD) or Imitation Learning (IL), is a set of paradigms that allow the robot to learn new tasks from *demonstrations*, provided by an expert, of how to perform the task [35].

2.3.4 Robot learning applications in telerobotics

In this context, the use of ML techniques in telerobotics has been growing in the last years as shown by the applications with different approaches, such as LfD in SC [3], [38], [10] and HSC [9] applications and IRL, as demonstrated in [17], or by using RL algorithms: in movements assistance [14], path planing in static and dynamic environments [15] and [16], dexterous telemanipulation in virtual environments [12], in semiautonomous control playing video-games or flying a real quadrotor [27].

2.4 Reinforcement Learning

Reinforcement Learning (RL), as illustrated in Figure 2.5, is an ML method that bases its operation on the maximization of rewards received by an agent that interacts with its environment following a certain control policy π(which can be deterministic or probabilistic) that defines the learning agent's way of behaving at a given time [39]. RL problems are mathematically defined as Markov decision processes (MDPs), which describe the problem of learning from interaction to achieve a goal. MDPs are composed of: a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a probability distribution $p(\boldsymbol{s}'|\boldsymbol{s},\ \boldsymbol{a})$ where p is the probability of reaching a state $\boldsymbol{s}' \in \mathcal{S}$ from the state $\boldsymbol{s} \in \mathcal{S}$ taking an action $\boldsymbol{a} \in \mathcal{A}$, and a reward function $r(\boldsymbol{s}, \boldsymbol{a})$. Using this information MDPs are able to model the dynamics of the environment.

In robotics, RL is a widely used framework because it allows flexibility in the learning of complex behaviors. Thereby, it provides the robot with the capability to discover optimal behaviors within the environment through trial and error interactions.

RL is an interesting approach because there are no explicit teaching rules in the algorithm, but rather implicit rules are given to the robot through the reward function $r(\boldsymbol{s}, \boldsymbol{a})$. This is a scalar function that specifies how good it is to take action $\boldsymbol{a}$ in state $\boldsymbol{s}$ in an immediate sense. The main objective of RL is to find the optimal policy π^* (often called controller or control policy) that maximizes the expected return $\mathbb{E}[R_t]$, defined as the future accumulated rewards [40]. To find π^*, a given policy π is evaluated and updated during the training of the agent as the result of its interaction with the environment. This evaluation may be performed through a *value function* ($V^{\pi}(\boldsymbol{s})$) that uses the relevant information collected by the robot every time step. This function is the expected return ($\mathbb{E}[R_t]$) starting in a state $\boldsymbol{s}$ following policy π thereafter.

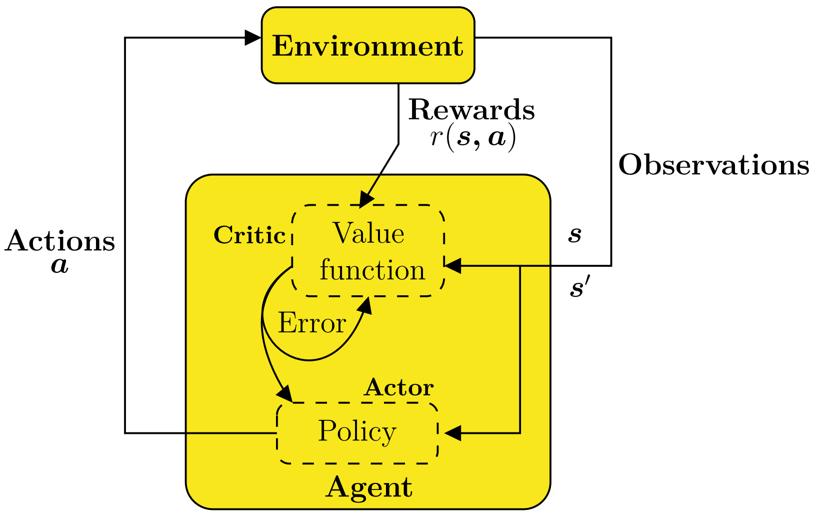


Figure 2.5: Reinforcement Learning scheme in the agent-environment interaction. Source: adapted from [39]

2.4.1 Reinforcement Learning classification

The different methods that are found in RL can be classified in: *model-based* and *model-free*. Model-based RL performs *planning* after attempting to learn the system model with the collected information to mimic or simulate experience. Planning is a way to optimize trajectories considering possible future situations before they are actually experienced. In contrast, model-free methods performs *learning* without knowledge of the system model through direct trial-and-error.

On the other hand, on-policy and off-policy methods, are related in the way how the policy is trained. Whereas on-policy methods learn and follow a determined policy evaluating and updating this policy during the training, off-policy methods learn a target policy from information generated by a behavior policy. In general, off-policy algorithms are more sample-efficient because they can re-use experience gathered in previous episodes [41], while on-policy algorithms are more stable.

Finally, according the action selection an extra classification appears. Thus, we can classify RL algorithms in *value-base*d or *policy-based* algorithms. In value-based approach, the action is taken through using the value function and an exploration strategy (ε-greedy or *softmax* for instance) [39]. So that, the policy estimation is derived from the value-function. Unlike value-based methods, policy-based algorithms estimate the policy directly, adjusting the policy parameters following an optimization method. The use of a direct policy parameterization allows using continuous actions.

2.4.2 Value-functions and optimal value-functions

Most RL algorithms base their operation in the computation of *value functions*. This function give us an idea of how good, in terms of *expected discounted return*, a given policy π will be when executed starting in an initial state $\boldsymbol{s}$. In this context, the discounted return (R_t) is the sum of discounted

Table 2.1: Reinforcement learning algorithms classification.

RL algorithms classification	
According the use of a model	• Model-based • Model-free
According the policy training	• On-policy • Off-policy
According the action selection	• Value-based • Policy-based

future rewards, and is mathematically defined by:

$$\begin{aligned} R_t =& r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \\ R_t =& \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \,, \end{aligned} \tag{2-6}$$

where $r_{\{.\}} \in \mathbb{R}$ are the future rewards and $\gamma \in [0,1]$ is known as the *discount factor*. The selection of this parameter determines the present value of future rewards [39].

There are two kind of value functions: the *state-value function* and the *action-value function*:

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}[R_t \mid \boldsymbol{s}_t = \boldsymbol{s}] \,, \tag{2-7}$$

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi}[R_t \mid \boldsymbol{s}_t = \boldsymbol{s},\ \boldsymbol{a}_t = \boldsymbol{a}] \,. \tag{2-8}$$

The *state-value function*(2-7), denoted by $V^{\pi}(\boldsymbol{s})$, is the expected return given a certain state $\boldsymbol{s}$ and following the policy π. This function can be defined in its recursive definition in form of the Bellman equation by:

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(\boldsymbol{s}_{t+1}) | \boldsymbol{s}_t = \boldsymbol{s}] \,. \tag{2-9}$$

Similarly, the *action-value function*(2-8), denoted by $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$, is the expected return given a certain state $\boldsymbol{s}$, taking the action $\boldsymbol{a}$, and following the policy π. The *action-value function* can be defined in form of the Bellman equation by:

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi}[r_{t+1} + \gamma Q^{\pi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) | \boldsymbol{s}_t = \boldsymbol{s},\ \boldsymbol{a}_t = \boldsymbol{a}] \,. \tag{2-10}$$

When an optimal policy (denoted by π^*) is found, the expected return is better than or equal to any other policy. In this case, the value-functions are *optimal.*

$$\begin{aligned} Q^*(\boldsymbol{s}, \boldsymbol{a}) &= \max_{\pi}\ Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \\ &= \mathbb{E}[r_{t+1} + \gamma \max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}_{t+1}, \boldsymbol{a}') | \boldsymbol{s}_t = \boldsymbol{s},\ \boldsymbol{a}_t = \boldsymbol{a}] \,. \end{aligned} \tag{2-11}$$

2.4.3 Value function approximation and policy gradient methods

When actions and states are discrete and low-dimensional, tabular methods to value-function calculation are highly recommended. However, for continuous states and actions, complexity increases immensely and the use of tabular methods is not suitable. Therefore, it is necessary to find a lower dimensional representation of the system. This is achieved using *value function approximation*. Real-world RL problems, especially in robotics, are high-dimensional and present continuous states and actions. Policy gradient algorithms are probably the most common approach used to address this kind of problem. They use a parametric policy representation ($\pi_{\boldsymbol{\theta}}$) through a parameter vector $\boldsymbol{\theta}$. Then, in order to maximize the cumulative discounted reward, it is possible to define a performance objective function $J(\pi_{\boldsymbol{\theta}})$ that integrates the performance over the entire state space. This objective function can be written as:

$$\begin{aligned} J(\pi_{\boldsymbol{\theta}}) =& \int_{\mathcal{S}} \rho^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}) \int_{\mathcal{A}} \hat{\pi}(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta}) r(\boldsymbol{s},\boldsymbol{a}) \mathrm{d}\boldsymbol{a}\mathrm{d}\boldsymbol{s}\,, \\ =& \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[r(\boldsymbol{s},\boldsymbol{a})]\,, \end{aligned} \tag{2-12}$$

where $\rho^{\pi_{\boldsymbol{\theta}}}(s)$ is the state distribution. Thereby, the parameter vector $\boldsymbol{\theta}$ may be optimized to maximize the objective function in direction of the performance gradient denoted by $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ [42], resulting in the well known *policy gradient theorem* [39] for stochastic policies:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_{\mathcal{S}} \rho^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \hat{\pi}(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta}) Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) \mathrm{d}\boldsymbol{a}\mathrm{d}\boldsymbol{s}\,, \tag{2-13}$$

which can be rewritten in terms of an expectation:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \hat{\pi}(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta}) Q^{\pi}(\boldsymbol{s},\boldsymbol{a})]\,. \tag{2-14}$$

Based on policy gradient theorem, a widely used architecture is *actor-critic*. In this architecture the *actor* adjusts the parameter vector $\boldsymbol{\theta}$ of the policy through (2-14), and the action-value function is approximated through a parameter vector $\boldsymbol{w}$ that is evaluated and updated by the *critic*.

$$Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) \approx \hat{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{w})\,, \tag{2-15}$$

thus, the policy gradient becomes:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \hat{\pi}(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta}) \hat{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{w})]\,. \tag{2-16}$$

2.4.4 Deterministic policy gradient theorem

Similar to the stochastic case, the policy gradient theorem presented in (2-14) can be extended to the deterministic case [42]. Given a deterministic policy $\mu_{\boldsymbol{\theta}}$ with parameter vector $\boldsymbol{\theta}$ denoted by $\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})$, the objective function can be defined as:

$$\begin{aligned} J(\mu_{\boldsymbol{\theta}}) =&\mathbb{E}_{\mu_{\boldsymbol{\theta}}}[R_t|\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})] \\ =&\mathbb{E}_{\mu_{\boldsymbol{\theta}}}[r(\boldsymbol{s},\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta}))] \\ =&\int_{\mathcal{S}}\rho^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{s})r(\boldsymbol{s},\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta}))\mathrm{d}\boldsymbol{s}\,. \end{aligned} \tag{2-17}$$

Then, the deterministic policy gradient theorem can be calculated as:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}}J(\mu_{\boldsymbol{\theta}}) =& \int_{\mathcal{S}}\rho^{\mu_{\boldsymbol{\theta}}}(s)\nabla_{\boldsymbol{\theta}}\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})\nabla_{\boldsymbol{a}}Q^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{s},\boldsymbol{a})|_{a=\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})}\mathrm{d}\boldsymbol{s} \\ =&\mathbb{E}_{\mu_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}}\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})\nabla_{\boldsymbol{a}}Q^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{s},\boldsymbol{a})|_{\boldsymbol{a}=\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})}]\,. \end{aligned} \tag{2-18}$$

2.4.5
Deep Deterministic Policy Gradient algorithm (DDPG)

Since we have a real system with continuous actions which is difficult to model because of the presence of the human in the control loop, it is necessary to choose an RL algorithm able to deal with these specifications that also allows faster convergence. Therefore, the designed controller for this project is based on the Deep Deterministic Policy Gradient (DDPG) algorithm. This is a model-free off-policy actor-critic method that uses an experience replay buffer $\mathcal{R}$. This buffer consists in a series of tuples containing relevant information of the interaction of the agent with the environment as states (initial and final), rewards and actions: $(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')$. This information is then sampled in mini batches and used to update the parameters of the function approximators in order to achieve the optimal policy. DDPG takes advantage of the two innovations presented in [43] to learn stable and robust policies: off-policy training with random samples from a replay buffer to minimize correlation between samples and train the main function with a target Q-function to obtain consistent targets during temporal difference backups. Unlike [43], the target network is composed of a copy of the actor and critic networks and it is softly updated through an update parameter τ. Thereby, the weights of the target networks are slowly updated from the learned networks rather than directly copy the weights. This modification results in a stable architecture able to learn competitive polices using low-dimensional observations, same hyper-parameters and same structure [44].

DDPG was the state-of-the-art in continuous control tasks at its introduction in 2015 presenting good performance in simulated and real implementations [41].

2.4.5.1 DDPG functioning

Given a deterministic policy μ in an actor-critic configuration, the actor can be parameterized with $\boldsymbol{\theta}$ and the critic with $\boldsymbol{w}$ respectively. Thereby, both are parametric functions denoted by:

$$\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})\,, \tag{2-19}$$

$$\hat{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{w})\,, \tag{2-20}$$

where (2-19) denotes the deterministic parameterized actor (policy) and (2-20) denotes the deterministic parameterized critic (action-value function). Then, the deterministic target parameterized action-value function can be defined as:

$$\hat{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{w}^{\dagger}) = \mathbb{E}_{\mu_{\boldsymbol{\theta}}}[r(\boldsymbol{s},\boldsymbol{a}) + \gamma\hat{Q}(\boldsymbol{s}',\hat{\mu}(\boldsymbol{s}';\boldsymbol{\theta}^{\dagger});\boldsymbol{w}^{\dagger})]\,, \tag{2-21}$$

where γ is the discount rate, $\boldsymbol{s}'$ denotes a next state, $\boldsymbol{\theta}^{\dagger}$ is the target parameter vector for the actor and $\boldsymbol{w}^{\dagger}$ is the target parameter vector for the critic. On the other hand, we can consider parameterized approximated action-value functions that are optimized by minimizing the loss function:

$$L(\boldsymbol{w}) = \mathbb{E}_{\mu_{\boldsymbol{\theta}}}[\hat{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{w}) - y]\,, \tag{2-22}$$

with

$$y = r(\boldsymbol{s},\boldsymbol{a}) + \gamma\hat{Q}(\boldsymbol{s}',\hat{\mu}(\boldsymbol{s}';\boldsymbol{\theta}^{\dagger});\boldsymbol{w}^{\dagger})\,, \tag{2-23}$$

where y is calculated in a separate target representation in the same way as in [43]. Thereby, using (2-22) and (2-23) and applying the concept of batch learning for stability, the parameter vector $\boldsymbol{w}$ can be optimized defining the next loss function for a mean of N samples per mini-batch:

$$L(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}[(\hat{Q}(\boldsymbol{s}_i,\boldsymbol{a}_i;\boldsymbol{w}) - y_i)^2]\,. \tag{2-24}$$

Then, the actor can be updated applying the deterministic version of the *policy gradient theorem*, presented in (2-18) for the mini-batch:

$$\frac{1}{N}\sum_{i=1}^{N}[\nabla_{\boldsymbol{a}}\hat{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{w})|_{\boldsymbol{s}=\boldsymbol{s}_i,\boldsymbol{a}=\hat{\mu}(\boldsymbol{s}_i;\boldsymbol{\theta})}\nabla_{\boldsymbol{\theta}}\hat{\mu}(\boldsymbol{s};\boldsymbol{\theta})|_{\boldsymbol{s}=\boldsymbol{s}_i}]\,. \tag{2-25}$$

The samples used to train the actor and critic functions come from the replay buffer $\mathcal{R}$ where are stored all the past transitions $(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')$ sampled from the environment using the exploration policy. In addition, to build the exploration policy, noise sampled from a suitable noise process $\mathcal{N}$ is added to the actor policy. Finally, to avoid instabilities, when the network is being trained , the parameter vectors in the target network are softly updated through:

$$\begin{aligned} \boldsymbol{\theta}^{\dagger} &\leftarrow \boldsymbol{\theta}^{\dagger} + \tau(\boldsymbol{\theta} - \boldsymbol{\theta}^{\dagger}) \,, \\ \boldsymbol{w}^{\dagger} &\leftarrow \boldsymbol{w}^{\dagger} + \tau(\boldsymbol{w} - \boldsymbol{w}^{\dagger}) \,. \end{aligned} \tag{2-26}$$

where $\boldsymbol{\theta}^{\dagger}$ and $\boldsymbol{w}^{\dagger}$ denote the parameter vector of the target network, with an update target parameter $\tau << 1$. The complete algorithm is presented in Algorithm 1. For additional reading, the complete mathematical foundation of policy gradient methods as well as the complete analysis of the DDPG algorithm can be found in [42] and [44] respectively.

Algorithm 1 DDPG algorithm

```
function DDPG($\gamma, \tau$)
    Initialize $\boldsymbol{w}, \boldsymbol{\theta}, \boldsymbol{w}^{\dagger}, \boldsymbol{\theta}^{\dagger}$ with random values close to zero
    $t \leftarrow 0$
    while True do
        $\boldsymbol{s} \leftarrow$ Start
        repeat
            Select action $\boldsymbol{a}_t$ from $\hat{\mu}(\boldsymbol{s}_t; \boldsymbol{\theta}) + \mathcal{N}_t$
            Execute action $\boldsymbol{a}_t$, observe reward $r$ and observe new state $\boldsymbol{s}'_t$
            Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)$ in $\mathcal{R}$
            TRAIN-MINIBATCH($\mathcal{B}, \gamma, \tau, \boldsymbol{\theta}, \boldsymbol{\theta}^{\dagger}, \boldsymbol{w}, \boldsymbol{w}^{\dagger}$)
            $\boldsymbol{\theta}^{\dagger} \leftarrow \boldsymbol{\theta}^{\dagger} + \tau(\boldsymbol{\theta} - \boldsymbol{\theta}^{\dagger})$
            $\boldsymbol{w}^{\dagger} \leftarrow \boldsymbol{w}^{\dagger} + \tau(\boldsymbol{w} - \boldsymbol{w}^{\dagger})$
            $t \leftarrow t + 1$
        until episode ends

function TRAIN-MINIBATCH($\mathcal{B}, \gamma, \tau, \boldsymbol{\theta}, \boldsymbol{\theta}^{\dagger}, \boldsymbol{w}, \boldsymbol{w}^{\dagger}$)
    Sample a random minibatch $\mathcal{B} \subset \mathcal{R}$ with $N$ samples
    for each $b_i : (\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_i^{'}) \in \mathcal{B}$ do
        $y_i = r_i + \gamma \hat{Q}(\boldsymbol{s}'_i, \hat{\mu}(\boldsymbol{s}'_i; \boldsymbol{\theta}^{\dagger}); \boldsymbol{w}^{\dagger})$
    Train $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w})$ on all samples $N$
    Move $\hat{\mu}(\boldsymbol{s}; \boldsymbol{\theta})$ according to the sampled deterministic policy gradient
    $\frac{1}{N} \sum_{i=1}^{N} [\nabla_{\boldsymbol{a}} \hat{Q}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w})|_{\boldsymbol{s}=\boldsymbol{s}_i, \boldsymbol{a}=\hat{\mu}(\boldsymbol{s}_i; \boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \hat{\mu}(\boldsymbol{s}; \boldsymbol{\theta})|_{\boldsymbol{s}=\boldsymbol{s}_i}]$
```

2.4.6 Autorregressive (AR) processes

To encourage exploration, noise is added to the policy in the action selection when the RL agent is being trained. Usually, Gaussian noise is used to achieve this effect of exploration showing high effectiveness in simulated systems. However, for real implementations, changing the control signal very quickly can result in damages to the system. This is due to the large amount of training that is required until convergence. For this reason, noise signals with correlated terms need to be chosen to avoid the chattering effect, changing smoothly in relation to the previous term. Ohrnstein-Uhlenbeck (OU) process is one kind of noise with smooth changes used in previous work for continuous cases [44]. In a discrete-time form, it can be shown that OU process is a first-order Gaussian autorregressive (AR) process [45]. Thereby, the AR process (termed as AR-p for now on), used in this work, generalizes the OU processes and provide a wider space of possible exploration trajectories. A stationary AR-p $\mathcal{X}_t$ of order p, with zero mean and finite variance is defined by:

$$\begin{aligned} \mathcal{X}_t &= \sum_{i=1}^{p} \tilde{\phi}_i \mathcal{X}_{t-i} + Z_t \,, \\ \tilde{\phi}_i &= (-1)^{i+1} \binom{p}{i} \alpha^i \,, \\ Z_t &\sim \mathcal{N}(0,\ \tilde{\sigma}_Z^2) \,. \end{aligned} \tag{2-27}$$

where $\tilde{\phi}_i \in \mathbb{R}$, $i = 1, ..., p$ are real coefficients, $\binom{p}{i} = \frac{p!}{i!(p-i)!}$, the AR-$p$ parameter $\alpha \in [0,\ 1)$, and Z_t is a white noise $Z_t \sim \mathcal{N}(0,\ \tilde{\sigma}_Z^2)$ with $\tilde{\sigma}_Z^2 < \infty$. $\tilde{\sigma}_Z^2$ is a solution of the system of Yuve-Walker equations described in [45] with $\{\phi_i = \tilde{\phi}_i\}$. For continuous control, actions taken given the parameterized polices are defined by:

$$\boldsymbol{a}_t = \hat{\mu}(\boldsymbol{s}_t; \boldsymbol{\theta}) + \mathcal{N}_t \,, \tag{2-28}$$

where $\mathcal{N}_t$ is white Gaussian noise. In order to use AR-p process in the action selection, Gaussian noise $\mathcal{N}_t$ in (2-28) is replaced by an $\mathcal{X}_t$ term with order $p \in \mathbb{N}$ and $\alpha \in [0,\ 1)$ parameters defined in (2-27), obtaining:

$$\boldsymbol{a}_t = \hat{\mu}(\boldsymbol{s}_t; \boldsymbol{\theta}) + \mathcal{X}_t \,. \tag{2-29}$$

So, the noise added in the action selected is time-correlated with the previous terms obtaining in this way a smoothed noise that prevents hardware damage.

2.5 Deep Learning (DL)

Deep learning (DL), also known as *deep networks*, is a machine learning subset that uses artificial neural networks (ANN) at its core. Deep Learning models have gained attention in the last years by outperforming classical machine learning methods in different competitions and challenges. DL works as any ANN representation, where a function $y = \hat{f}(x; \boldsymbol{\theta})$ is trained to match the real representation $f^*(x)$ which give us a desired output y. Thereby, y trains the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation for every input x [46].

In this context, as every layer can be considered as a function approximator, the stack of layers is denoted as: $f^N \circ ... \circ f^2 \circ f^1(x) = f^{(N)}(...f^{(2)}(f^{(1)}(x))...)$, where $f^{(1)}$ is called the input layer and the overall length N is the depth of the model.

The different architectures used for DL models depend on the application and the available dataset. If a labeled dataset is used, the method is known as *supervised learning*, and every input x is accompanied by a label (or desired output) $y \approx \hat{f}(x)$. The objective of this approach is to minimize the error between the network output and the given label. In this way, the network is trained and updated until achieving a minimal error or reach the desired performance.

On the other hand, when a no-labelled dataset is given, the method is known as *unsupervised learning*. In this approach, the error is calculated according to a series of features that the network learns by itself along with the training.

The choice of the number of units, layers, activation functions and parameters of the network are part of the design and the application type in which it will be used. In recent years, deep learning has become an important tool to address different tasks such as signal processing, object recognition, natural language processing, etc [47].

2.5.1 Convolutional Neural Networks (CNN)

Convolutional neural networks (CNN), as indicated by their name, are a kind of ANN that uses the mathematical operation of *convolution* in at least one of their layers [46]. Their popularity and success lie in the fact that they

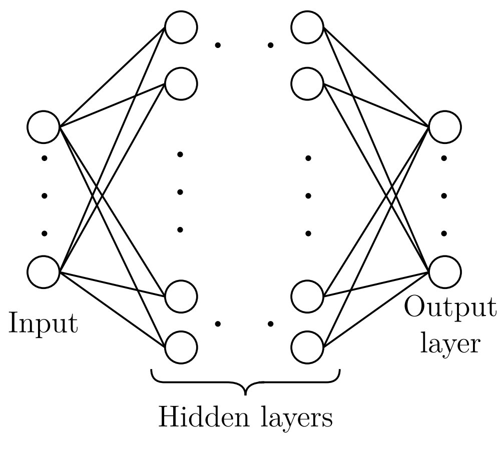


Figure 2.6: Basic architecture for an ANN representation.

can be applied in a wide range of applications with arrays of any dimension as the particular input type. Thereby, image data can be easily treated with CNN. In recent years, CNN have been widely adopted as the main processing tool for computer vision applications.

2.5.1.1 Convolution principle

Remembering, the convolution is a simple multiplication between two signals (continuous or discrete), where one of them is flipped and displaced along the time axis. Mathematically it is denoted with:

$$y(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)h(t-\tau)\,, \tag{2-30}$$

for the discrete case. Where x and h are causal signals, τ is the lag time and y is the signal output. This operation is typically denoted with an asterisk in the following way:

$$y(t) = (x * h)(t)\,. \tag{2-31}$$

This approach can be extended to the ANN approach where x represents a multidimensional input array with shape: width $\times$ height $\times$ channels. h is represented as a multidimensional kernel with shape: k $\times$ k $\times$ N, where N represents the number of features. And finally the output y is the resulting feature map whose shape is defined by the next equation:

$$\text{Output shape} = \left\lfloor \frac{\text{width} + 2\text{p} - \text{k}}{\text{s}} + 1 \right\rfloor \times \left\lfloor \frac{\text{height} + 2\text{p} - \text{k}}{\text{s}} + 1 \right\rfloor \times \text{N}\,. \tag{2-32}$$

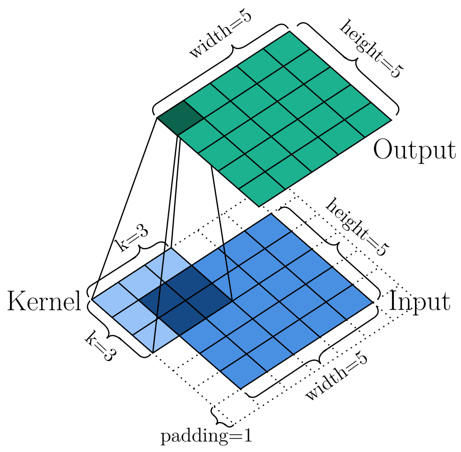
where p is the padding of the image (usually zero) and s is the stride used for the kernel to move it along all the image.

For example (See Figure 2.7), given an input image with shape $5 \times 5 \times 1$, and a kernel of shape $3 \times 3 \times 1$. Using a padding p $= 1$, and a stride s $= 1$, applying (2-32), the resulting output shape will be: $5 \times 5 \times 1$.

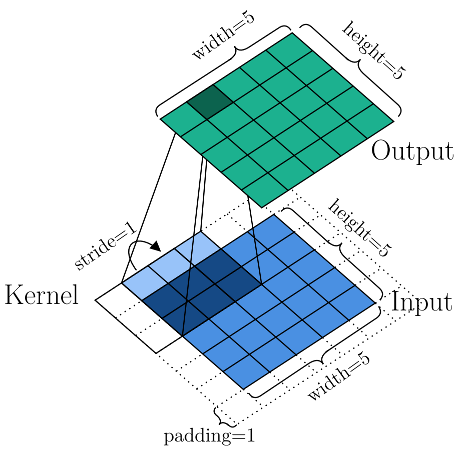
2.5.1.2 CNN architecture

In a common configuration of CNN, three types of layers can be found:

1. Convolutional
2. Pooling
3. Fully connected



(a) Convolution at start.



(b) Convolution in the next step.

Figure 2.7: Convolution process. Source: adapted from [48]

The *convolutional* layers are applied to extract relevant features from the image input, obtaining in this way the relevant information. One problem, especially when convolutional layers are stacked, is that the number of parameters grows considerably. Training many parameters can result in a time-consuming process. To avoid this effect, a *pooling* layer is used after the set of convolutional layers performing the pooling operation. This operation replaces the output at a certain location with a summary statistic of the nearby outputs. Thereby, the spatial size of the feature map is reduced and consequently, the number of parameters and the computation time are decreased. The most common pooling operation is the *max-pooling*, which takes the maximum value, discarding the other elements in the pooling kernel's input field. Finally, the *fully connected* layers are a common ANN architecture where the information obtained of the input image in the previous layers is processed, computing the desired outputs.

2.5.2 VGG16 network

VGG16 is a CNN architecture proposed by Simonyan and Zisserman that in 2014 won the *imagenet* large scale visual recognition (ILSVRC) competition. This is an annual competition for localization and classification tasks with 1000 classes, where VGG obtained the 1st and 2nd place respectively [49]. The novelty with respect to its predecessors was the analysis and inclusion of very small convolutional kernels ($k = 3$) and the increase of the network depth. This change reduced the number of parameters in the proposed architecture and increased non-linearity. In this context, it was demonstrated that a better image representation could be achieved using a deeper network [50]. The architecture representation is presented in Figure 2.8 and is composed of a series of convolution blocks, each followed by a *max-pooling* layer. And finally, three fully connected layers are used whose size depends on the number of classes of the proposed task.

Due to its high performance, the VGG network can be found as a pre-trained model in most known frameworks for DL, such as tensorflow [52], keras [53], pytorch [54], caffe [55] and so on, to load it and use it in different applications, for instance in a transfer learning approach.

2.5.3 Transfer learning

The transfer learning approach is a tool of ML that addresses the problem of insufficient training data. Here, the objective is to transfer knowledge from a source domain to the target domain [56]. In this context, the technique takes advantage of pre-trained networks to *transfer* learned knowledge to new neural networks. Thereby, there is no necessity to train all the parameters from scratch. Besides, it is not necessary to have a large-scale dataset to perform the training and the computation time is also decreased.

To apply transfer learning the following methods can be used:

- Frozen features
- Fine-tuning features

When frozen features method is used, the basic functioning is as follows: the first part of the source network (called *base*), previously trained with a large-scale dataset, is copied to a target network and then is frozen. Next, in the last part of the target network, new layers are added with non-trained weights. Finally, the target network is trained according to the objective task with a new small dataset until the desired performance is achieved. In this

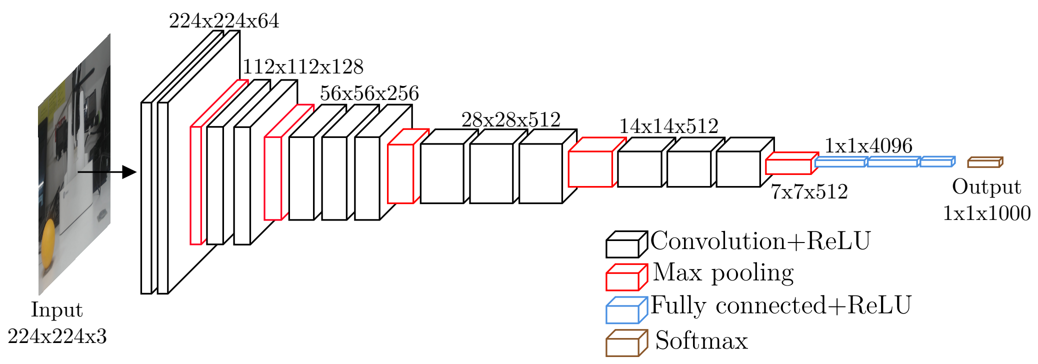


Figure 2.8: VGG16 network architecture. Source: adapted from [51].

way, the target network is composed of a *base* that contains general pre-learned knowledge and new layers that contain specific knowledge for the task.

On the other hand, the fine-tuning feature method has practically the same basic functionality as the first technique. However, the main difference lies in the fact that the *base* is not frozen. In this way, all the weights are trained to perform a *fine-tune* of the parameters according to the new dataset. Previous research shows that in the target network the features learned previously still persist after use the fine-tuning technique with a new dataset [57].

In addition, an extra method, combines the power of both variants, performing first a feature extraction with the frozen features technique and then, performing a fine-tuning unfreezing gradually the frozen layers performing different training in order to achieve a higher performance in comparison with the initial approaches [58].

3
Haptic Shared Control controller design

In this chapter, the design of the proposed system to perform haptic shared control is described. The necessity to achieve complex behaviors with flexible algorithms gave rise to robot learning approach, where the use of RL-based controllers is becoming popular. Thereby, a desired behavior can be learned with few training episodes. We will describe the proposed method based on the DDPG algorithm including a *task detector* section which is responsible for decoding the user intention when performing the task. We will also describe the different approaches used in order to improve performance in the task completion as well as the implementation of a simulator that serves to test the different proposed modifications.

3.1
System Description

As was discussed in the previous chapter, HSC has shown promising results improving the operator's performance in teleoperation [5] [6] [7]. On the other hand, the use of ML methods to achieve complex behaviors has grown in recent years. Previous applications presented in Chapter 2 using ML showed its successful application in telerobotics by applying different control algorithms. However, practically all applications learn at their core of a set of trajectory distributions [3] [15] [16]. Although it is well known the importance of inferring user's goal from his actions in shared autonomy, recent research still uses fixed goals [59]. Therefore, despite the good performance achieved with these algorithms, we can not consider them if the intention is to help the user to perform tasks that depend on implicit information, for example current images of the performed task. We propose to address the problem of implicitly encoding task intention using two types of data: visual and numerical. In this way, it is possible to perform any task that is able to be encoded through the state vector, which will be described in the next sections.

Thereby, the architecture presented in Figure 1.1 can be extended to an HSC approach applying an RL-based controller as central core. In this way, as input/output devices can be used a haptic device in the master side and a robotic manipulator on the slave side. Considering that both devices are kinematically dissimilar, the position control will be performed on the tip level.

As shown in Figure 3.1, the proposed system bases its operation in the central controller which is composed of two separate sub-controllers: the *teleoperation controller* and the *RL-controller*. The teleoperation controller is responsible for *replicating* the master movements on the slave side, while the RL-controller is responsible for processing the information and sending the commands to the haptic device in order to assist the operator. The basic functioning of the two controllers can be explained by the flowcharts presented in Figure 3.2 and Figure 3.3 respectively.

The chosen task will be the *pick-and-place* task due to its simple operation. This task consists of picking an object from one place, named *initial point*, and dropping it in an objective position, named *goal*. For this implementation, the initial point and the goal are selected randomly into the

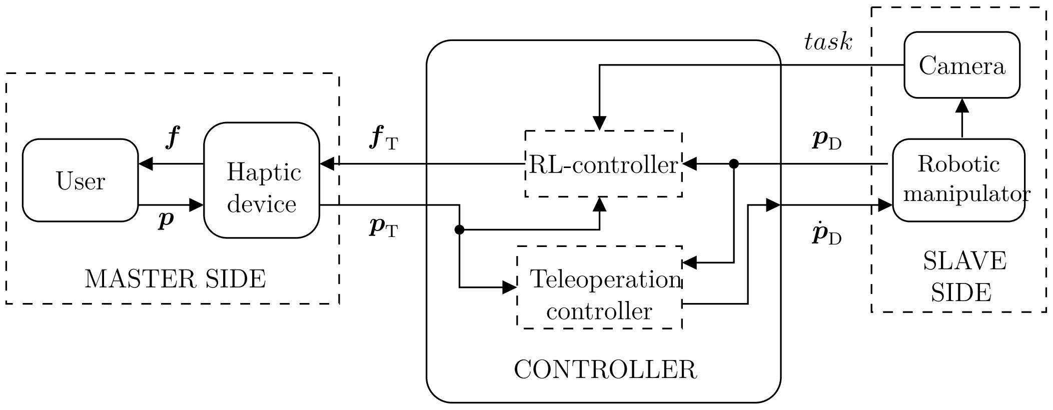
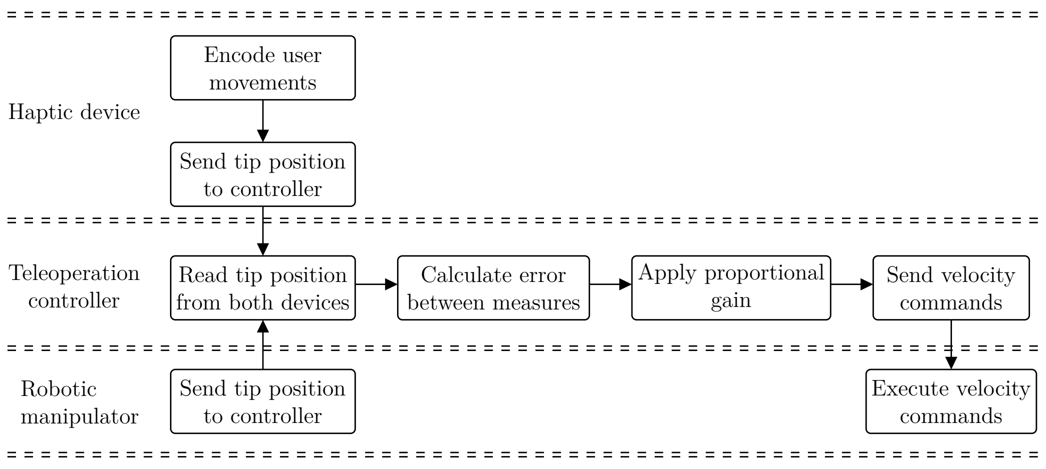


Figure 3.1: Overall functionality of the system: The teleoperation controller performs the *imitation* and the RL-controller performs the *assistance*.



Haptic device
Encode user movements
Send tip position to controller
Teleoperation controller
Read tip position from both devices
Calculate error between measures
Apply proportional gain
Send velocity commands
Robotic manipulator
Send tip position to controller
Execute velocity commands

Figure 3.2: Teleoperation controller flowchart.

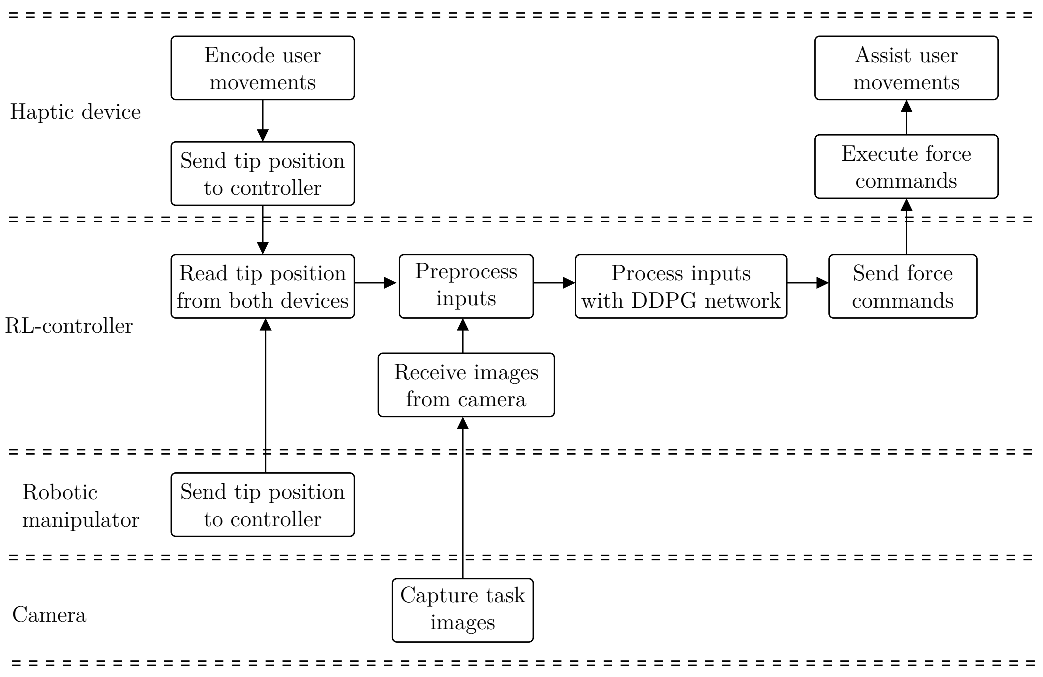


Figure 3.3: RL-controller flowchart.

manipulator end-effector workspace to avoid physical restrictions. To increase complexity in performing the task, an obstacle will be placed between the initial and the final points. In addition, the pick-and-place task will be performed in a bidirectional way; that is, take the ball from the first position to the goal, and then, in the next half, the point positions are exchanged as illustrated in Figure 3.4; this modification was added to demonstrate that the proposed system is able to learn more than one task. So, a complete episode is considered to following the next trajectory: *initial point-goal-initial point*.

3.2 HSC controller

As was mentioned in Section 3.1, the HSC controller is composed of two sub-controllers: the teleoperation controller and the RL-controller. It is worth to mention that both controllers work in an independent way; that is, despite both controllers share some information (position and velocity), the functioning of one does not affect directly the functioning of the remaining controller. Therefore, any influence one has on the other is mediated through the user who is performing the task. This can be observed in Figures 3.2 and 3.3, where the flowcharts describe the general functionality of each controller (See Figure 3.1).

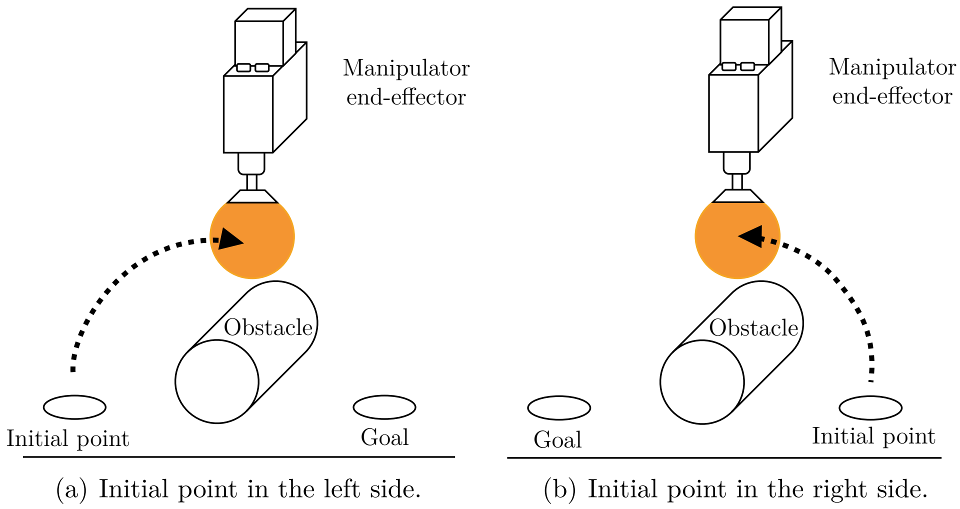


Figure 3.4: Task definition.

3.2.1 Teleoperation controller

The *teleoperation controller* is responsible to replicate the user movements with the Haptic device in the manipulator. This is performed through direct teleoperation using position control in the tip level using the position coordinates of the haptic device $\boldsymbol{p}_{\mathrm{T}}$ and the manipulator $\boldsymbol{p}_{\mathrm{D}}$ respectively. The basic implementation for this controller is presented in Figure 3.5

The direct teleoperation is achieved in the next way: Let $\boldsymbol{p}_{\mathrm{T}} = [x_{\mathrm{T}},\ y_{\mathrm{T}},\ z_{\mathrm{T}}]$ and $\boldsymbol{p}_{\mathrm{D}} = [x_{\mathrm{D}},\ y_{\mathrm{D}},\ z_{\mathrm{D}}]$ be the current position of the end-effector for the haptic device and the manipulator w.r.t their base link in Cartesian coordinates respectively. Then, the desired manipulator position is obtained using(2-4). The error between the desired and the current position is calculated by:

$$\boldsymbol{e} = \boldsymbol{p}_{\mathrm{Dd}} - \boldsymbol{p}_{\mathrm{D}}\,, \tag{3-1}$$

$$\boldsymbol{p}_{\mathrm{Dd}} = \xi \boldsymbol{p}_{\mathrm{T}} + \boldsymbol{p}_{\mathrm{offset}}\,. \tag{3-2}$$

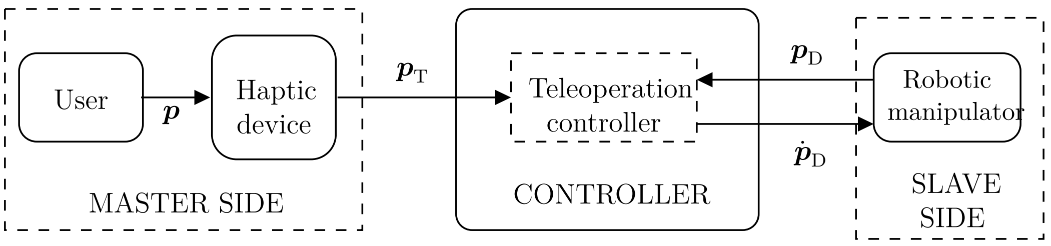


Figure 3.5: Basic teleoperation setup: The teleoperation controller takes $\boldsymbol{p}_{\mathrm{T}}$ and $\boldsymbol{p}_{\mathrm{D}}$ as inputs and sends velocity commands $\dot{\boldsymbol{p}}_{\mathrm{D}}$ to the slave side.

where $\boldsymbol{e} = [\mathrm{e}_x,\ \mathrm{e}_y,\ \mathrm{e}_z]$ are the calculated errors for axis x, y and z respectively. The manipulator velocity to track the haptic device tip is calculated applying a proportional positive gain k_D to the error vector calculated in (3-2):

$$\dot{\boldsymbol{p}}_{\mathrm{D}} = k_D \boldsymbol{e}\,. \tag{3-3}$$

where $\dot{\boldsymbol{p}}_{\mathrm{D}} = [\dot{x}_{\mathrm{D}},\ \dot{y}_{\mathrm{D}},\ \dot{z}_{\mathrm{D}}]$ are the calculated velocities for the manipulator end-effector. Thereby, the manipulator replicates the haptic device movements with a velocity proportional to the error between both devices.

3.2.2 RL-controller

The RL-controller is responsible to perform the *assistance* to the teleoperator. This controller bases its operation in the DDPG algorithm explained in Section 2.4.5. To perform the assistance, the DDPG network is trained with relevant information as inputs about the positions to provide the guiding forces as outputs.

3.2.2.1 DDPG state vector

To handle the information coming from both devices and the task detector, the information is encapsulated in the form of a state vector. This state vector is used in the DDPG network and stored in the replay buffer with other samples to train the agent. The proposed state vector for the DDPG network consists in the tip positions for the manipulator $\boldsymbol{p}_{\mathrm{D}}$ and the haptic device $\boldsymbol{p}_{\mathrm{T}}$ respectively. In addition, an extra term called *task* is appended which encodes the task direction intention:

$$\boldsymbol{s} = [\boldsymbol{p}_{\mathrm{D}},\ \boldsymbol{p}_{\mathrm{T}},\ \mathit{task}]\,, \tag{3-4}$$

where the position points are in Cartesian coordinates; that is, $\boldsymbol{p}_{\mathrm{D}} = [x_{\mathrm{D}},\ y_{\mathrm{D}},\ z_{\mathrm{D}}]$ and $\boldsymbol{p}_{\mathrm{T}} = [x_{\mathrm{T}},\ y_{\mathrm{T}},\ z_{\mathrm{T}}]$. Moreover, the *task* term is set according to one of the architectures that will be described in the next sections. An extra condition that needs to be considered is the network input. It is common that the input vector that ANN handles is in the range of $[-1,\ 1]$, therefore, the position vectors need to be scaled in an accepted range before being processed by the DDPG network in the RL-controller. This pre-processing is done using a scale factor ω which divides the position vector in order to be in the determined range.

To re-scale the states vector, this value was chosen as the maximum value in meters between the largest possible value that both devices can reach in their respective workspaces as shown in the following:

$$\omega = \max(\max(\boldsymbol{p}_{\mathrm{D}}), \max(\boldsymbol{p}_{\mathrm{T}}))\,. \tag{3-5}$$

In this way, the result of applying a scaling in the state vector given in (3-4) is:

$$\boldsymbol{s} = [\tilde{\boldsymbol{p}}_{\mathrm{D}},\ \tilde{\boldsymbol{p}}_{\mathrm{T}},\ \mathit{task}]\,, \tag{3-6}$$

where $\tilde{\boldsymbol{p}}_{\mathrm{D}} = \frac{\boldsymbol{p}_{\mathrm{D}}}{\omega}$ and $\tilde{\boldsymbol{p}}_{\mathrm{T}} = \frac{\boldsymbol{p}_{\mathrm{T}}}{\omega}$. Resulting in $\tilde{\boldsymbol{p}}_{\mathrm{D}} = [\tilde{x}_{\mathrm{D}},\ \tilde{y}_{\mathrm{D}},\ \tilde{z}_{\mathrm{D}}]$ and $\tilde{\boldsymbol{p}}_{\mathrm{T}} = [\tilde{x}_{\mathrm{T}},\ \tilde{y}_{\mathrm{T}},\ \tilde{z}_{\mathrm{T}}]$ respectively. The resulting state vector is:

$$\boldsymbol{s} = [\tilde{x}_{\mathrm{D}},\ \tilde{y}_{\mathrm{D}},\ \tilde{z}_{\mathrm{D}},\ \tilde{x}_{\mathrm{T}},\ \tilde{y}_{\mathrm{T}},\ \tilde{z}_{\mathrm{T}},\ \mathit{task}]\,. \tag{3-7}$$

3.2.2.2 DDPG network architecture

Here, it is worth noticing that (3-4) in Section 3.2.2.1 presents an extra term in the state vector that is appended to encode the task information. To perform this *task coding* in the DDPG network is considered a new functionality in the proposed system which we name as *task detector*. A first look of the task detector was shown in Figure 1.1, where task information comes from a camera placed on the slave side.

As part of the implementation of the task detector functionality, two implementations are proposed: manual task detection and autonomous task detection. Because the addressed task is a simple pick-and-place bidirectional movement, then, it is possible to encode the direction as a single value which determines which is the direction that the user wants to perform the task. Taking advantage of this condition, the first approach encodes the task direction as a numeric character passed to the DDPG network through the terminal. In this way, the operator provides the information about the task intention before performing the task. The basic implementation of this approach is presented in Figure 3.6.

On the other hand, in the second approach, the task is encoded in a dynamical way through images that are then decoded using an image processing technique. Thereby, the operator does not have to provide any extra information while is performing the task. The basic implementation is shown in Figure 3.7.

The first approach is implemented to demonstrate that the DDPG network can *learn* to assist the user in more than one task. In contrast, the second approach shows that the system can detect the task autonomously, as well as learn the task detection on-the-fly. Figure 3.8 summarizes the proposed implementation.

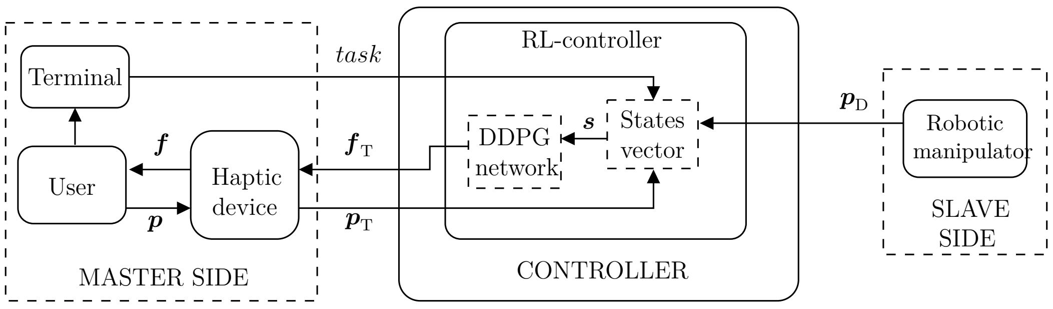


Figure 3.6: RL-controller with manual task selection: This approach takes the state vector $\boldsymbol{s}$, where the *task* information is given by the user, as input and provides the force commands $\boldsymbol{f}_\mathrm{T}$ to the haptic device to provide the assistance.

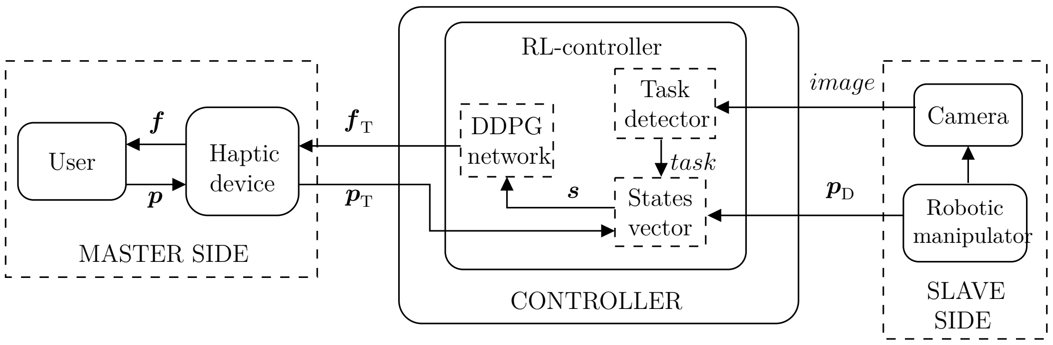


Figure 3.7: RL-controller with autonomous task detection: This approach takes the state vector $\boldsymbol{s}$, where the *task* information is given by a camera, as input and provides the force commands $\boldsymbol{f}_\mathrm{T}$ to the haptic device to provide the assistance.

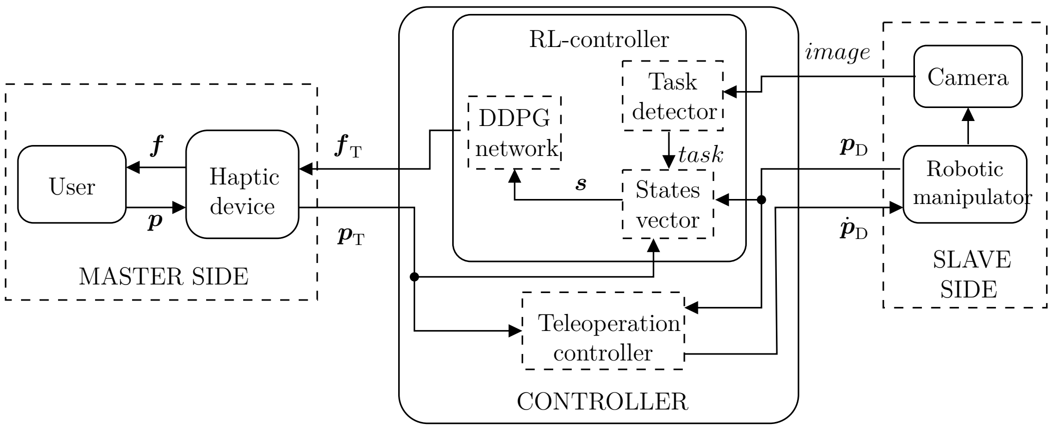


Figure 3.8: HSC controller with camera input functionality.

3.2.3
Reward function design

As explained in [60], one of the underestimated problems in RL is the goal specification, which is achieved by designing a proper reward function. This function must capture the general learning rules allowing the RL algorithm to converge to the desired behavior. In practice, designing a proper reward function is very difficult. Therefore, tuning becomes an important part of the path to find a good reward function. For that reason, for the HSC controller proposed in this thesis, two reward functions are designed.

3.2.3.1
Angle-based reward function

The first reward function is an angle-based function which takes as arguments the force vector and the haptic device velocities vector as illustrated in Figure 3.9. Thereby, the desired behavior that the reward function pretends to learn is to assist the user with the haptic device in the direction of the user movements. In addition, to boost learning, an extra reward is considered at the end of the episode, based on whether the episode was successful or not.

The functioning of the angle-based reward function is described in the following:

$$r(\boldsymbol{s},\boldsymbol{a}) = \begin{cases} r_{\mathrm{A}} & \text{, if } \mathbf{s} \text{ is an absorbing state} \\ -\beta & \text{, otherwise} \end{cases}$$

where

$$r_{\mathrm{A}} = \begin{cases} +10 & \text{, if goal position was reached} \\ -10 & \text{, if goal position was not reached} \end{cases} \tag{3-8}$$

The sub-index A in the reward term indicates that it is an absorbing state and β is the angle between the assistive force and the velocity vector. This angle

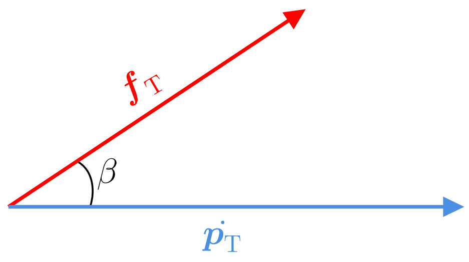

$\boldsymbol{f}_{\mathrm{T}}$
β
$\dot{\boldsymbol{p}}_{\mathrm{T}}$

Figure 3.9: First reward function: calculates the rewards based in the angle difference between the $\boldsymbol{f}_{\mathrm{T}}$ and $\dot{\boldsymbol{p}}_{\mathrm{T}}$ vectors.

can be calculated through:

$$\beta = \arccos\left(\frac{\boldsymbol{f}_{\mathrm{T}} \cdot \dot{\boldsymbol{p}}_{\mathrm{T}}}{\|\boldsymbol{f}_{\mathrm{T}}\|\|\dot{\boldsymbol{p}}_{\mathrm{T}}\|}\right). \tag{3-9}$$

The functioning of this function is as follows: the reward will be maximized when the direction of the assistive force $\boldsymbol{f}_{\mathrm{T}}$ is equal to the haptic device velocity $\dot{\boldsymbol{p}}_{\mathrm{T}}$. The negative signal ensures that the agent will try to minimize the angle between both vectors when maximizing the expected reward along all the training.

3.2.3.2 Fuzzy-based reward function

Although the angle-based reward function is designed to learn assistive forces in the direction of the user movements, it does not take into account the force or velocity magnitude, which could result in an undesired behavior.

Therefore, a second reward function is proposed as a fuzzy-based reward function. This approach bases its operation on the difference of magnitude between the assistive force and the user velocity presented in Figure 3.10. Thereby, this function can be designed to encourage the agent to influence the user to execute the task quickly and give him more control in the task execution; to achieve that, fuzzy rules condition the amount of reward that the agent receives according the velocity of the user.

The functioning of the fuzzy-based reward function is described in the

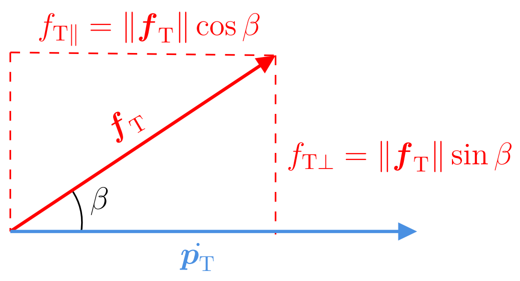

$f_{\mathrm{T}\parallel} = \|\boldsymbol{f}_{\mathrm{T}}\| \cos\beta$
$\boldsymbol{f}_{\mathrm{T}}$
β
$f_{\mathrm{T}\perp} = \|\boldsymbol{f}_{\mathrm{T}}\| \sin\beta$
$\dot{\boldsymbol{p}}_{\mathrm{T}}$

Figure 3.10: Second reward function diagram: calculates the rewards based in the magnitude of the $\boldsymbol{f}_{\mathrm{T}}$ and $\dot{\boldsymbol{p}}_{\mathrm{T}}$ vectors.

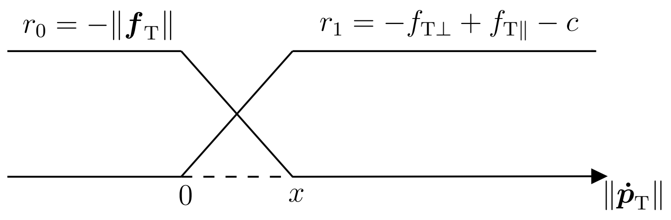

$r_0 = -\|\boldsymbol{f}_{\mathrm{T}}\|$
$r_1 = -f_{\mathrm{T}\perp} + f_{\mathrm{T}\parallel} - c$
0
x
$\|\dot{\boldsymbol{p}}_{\mathrm{T}}\|$

Figure 3.11: Second reward function: conditional rules.

following:

$$r(\boldsymbol{s},\boldsymbol{a}) = \begin{cases} r_{\mathrm{A}} & \text{, if } \mathbf{s} \text{ is an absorbing state} \\ r_{\mathrm{F}} & \text{, otherwise} \end{cases}$$

where

$$r_{\mathrm{A}} = \begin{cases} +10 & \text{, if goal position was reached} \\ -10 & \text{, if goal position was not reached} \end{cases} \tag{3-10}$$

where the sub-index A indicates that it is an absorbing state and the sub-index F indicates that it is a *fuzzified* reward. This reward term is defined as:

$$\begin{aligned} r_{\mathrm{F}} &= \varphi r_1 + (1-\varphi) r_0 \,, \\ r_0 &= -\|\boldsymbol{f}_{\mathrm{T}}\| \,, \\ r_1 &= -f_{\mathrm{T}\perp} + f_{\mathrm{T}\|} - c \,. \end{aligned} \tag{3-11}$$

The predicted force vector components are defined by $f_{\mathrm{T}\|} = \|\boldsymbol{f}_{\mathrm{T}}\| \sin\beta$ and $f_{\mathrm{T}\perp} = \boldsymbol{f}_{\mathrm{T}} \cos\beta$ respectively. The term c is defined by: $c = \|\max(\boldsymbol{f}_{\mathrm{T}})\|$ and the φ term is defined as:

$$\varphi = \begin{cases} 0 & \text{, if } \|\dot{\boldsymbol{p}}_{\mathrm{T}}\| = 0 \\ 1 & \text{, if } \|\dot{\boldsymbol{p}}_{\mathrm{T}}\| > x \\ \frac{\|\dot{\boldsymbol{p}}_{\mathrm{T}}\|}{x} & \text{, otherwise} \end{cases} \tag{3-12}$$

The scalar positive x term is defined as the maximum user velocity when random forces are applied. This φ term gives the reward function a particular behavior. When the user is performing the task and his current velocity is greater than x, then the term r_0 is eliminated from the reward function. In this case, the rewards are maximum when assistance forces go in the same direction as user velocity encouraging high assistive forces.

In contrast, when the user does not perform movements and the velocity is zero, then the r_1 term is eliminated from the reward function. Thereby, the assistance forces are not required and reward values are maximum when the forces go to zero. In this context, when the user decreases the velocity less than x, then, the reward function encourages the DDPG agent to decrease the assistance. In this way, the force magnitude is decreased in a smooth way to provide the user more control in the final part of the execution in order to reach the desired position. In addition, using the constant c in r_1, is ensured that the reward function maintains the rewards below to zero without achieve positive values, and continues implicitly minimizing the number of required steps.

3.3 Behavioral cloning

As part of the preliminary test, the use of behavioral cloning (BC) approach is proposed. This approach is commonly used as initialization in order to boost agent learning endowing some initial performance [26]. The proposed approach consists of using a simple FC-ANN as part of the implementation of the BC method. Thereby, the network can be trained with input data as the state vector $\boldsymbol{s}$ and some output vector that provides the basic intention of the task. For this case, a velocity vector $\dot{\boldsymbol{p}}$ can be used as output of the network and then used to approximate the assistive forces to guide the user in performing the task. In this way, it is possible to calculate the velocity information from the haptic device or the manipulator from different demonstrations and then use it to train the network. To build the dataset, a set of successful demonstrations is recorded with all the relevant information of the system; in this case, position and velocities for both devices executing the task. Then, two networks to be tested are proposed, the first one is trained using the proposed state vector $\boldsymbol{s}$ as input, and the estimated velocities of the haptic device $\hat{\dot{\boldsymbol{p}}}_{\mathrm{T}}$ as output, and the second one, which is trained using the proposed state vector $\boldsymbol{s}$ as input and the estimated velocities of the robot manipulator $\hat{\dot{\boldsymbol{p}}}_{\mathrm{D}}$ as output as is presented in Figure 3.12 where both networks are presented respectively. Because of the delay presented between both devices when the teleoperation is being performed, it is intended to test both approaches in the output network ($\hat{\dot{\boldsymbol{p}}}_{\mathrm{T}}$ or $\hat{\dot{\boldsymbol{p}}}_{\mathrm{D}}$). Thereby with $\hat{\dot{\boldsymbol{p}}}_{\mathrm{D}}$ it is expected a higher delay than $\hat{\dot{\boldsymbol{p}}}_{\mathrm{T}}$ in the beginning of the episode.

Finally, to approximate the assistive forces, the outputs of the FC-ANNs are multiplied with a positive proportional gain k_{BC} for every approach as following the next equations:

$$\begin{aligned} \boldsymbol{f}_{\mathrm{T}} &= k_{\mathrm{BC}}\hat{\dot{\boldsymbol{p}}}_{\mathrm{D}} \\ \boldsymbol{f}_{\mathrm{T}} &= k_{\mathrm{BC}}\hat{\dot{\boldsymbol{p}}}_{\mathrm{T}} \end{aligned} \tag{3-13}$$

$\boldsymbol{s}$
FC-ANN
$\hat{\dot{\boldsymbol{p}}}_{\mathrm{T}}$

(a) ANN with $\hat{\dot{\boldsymbol{p}}}_{\mathrm{T}}$ as output.

$\boldsymbol{s}$
FC-ANN
$\hat{\dot{\boldsymbol{p}}}_{\mathrm{D}}$

(b) ANN with $\hat{\dot{\boldsymbol{p}}}_{\mathrm{D}}$ as output.

Figure 3.12: ANN trained for the behavioral cloning approach.

3.4 Simulator

Following the implementation of the proposed system, the programming of a simulator in an open-source platform is also proposed. The objective of this functionality is to *simulate* the behavior of the master and the slave sides given the different configurations. Thereby, it is possible to test different network architectures of the proposed HSC-controller, as well as perform a hyper-parameter tuning, or test the different proposed reward functions. The intention to implement this simulator is not to replace the user in the different task executions in a virtual way. As was mentioned, the final objective is to have an approximation that can be used to find the architecture in which the system achieves better performance. The functioning of the simulator is explained in the next paragraph.

3.4.1 Simulator functioning

Similar to the real system, the simulator is composed of a master side and a slave side. The master side is responsible for providing the virtual trajectory simulating the user behavior in terms of user position in every time step. To do this, the master-side simulator uses a pre-recorded dataset of N successful trajectories which are then used to simulate the current position. Then, using a proportional function, the virtual user's forces are simulated together with the haptic device forces that come from the RL-controller. Finally, combining the haptic device and user forces, the user velocity is obtained and used to simulate the next virtual user position. In contrast, the slave-side simulator is used to provide the virtual manipulator position using the velocity commands coming from the teleoperation controller.

3.4.1.1 Master-side simulator

Consider a set of N successfully executed demonstrated trajectories:

$$\mathcal{D} = \{\{(x_0^i, y_0^i, z_0^i), (x_1^i, y_1^i, z_1^i), ..., (x_{M_i}^i, y_{M_i}^i, z_{M_i}^i)\}_{i=1}^N\},\ M_i > 0\,. \tag{3-14}$$

A single trajectory $\mathcal{D}^i$, $i \in [1, N]$ can be randomly chosen to simulate a single episode for task execution in the following way: the mean of the initial points $(\bar{x_0}, \bar{y_0}, \bar{z_0})$ of the sampled trajectories is taken as the starting point. On the other hand, the mean of the final points $(\bar{x}_{M_i}, \bar{y}_{M_i}, \bar{z}_{M_i})$ is taken as the *goal* point. In this context, at time t, the current index I_t^i of the i-th trajectory is calculated as the nearest point between the current simulated position $\check{\boldsymbol{p}}_{\mathrm{T}}$ and the sampled trajectory $\mathcal{D}^i$.

$$I_t^i = \arg\min(\mathcal{D}^i, \check{\boldsymbol{p}}_{\mathrm{T}})\,. \tag{3-15}$$

Then, the index of the next desired point is calculated as the current index added with a *step* constant.

$$I_{t+1}^i = I_t^i + \mathit{step}\,. \tag{3-16}$$

So that, given the current simulated position and the next desired sampled position, the simulated user force vector ($\check{\boldsymbol{f}}_{\mathrm{U}}$) is approximated as the difference between the two points multiplied with a proportional gain k_u.

$$\check{\boldsymbol{f}}_{\mathrm{U}} = k_u(\mathcal{D}^i_{I^i_{t+1}} - \check{\boldsymbol{p}}_{\mathrm{T}})\,. \tag{3-17}$$

Moreover, because we are modeling the human behavior, which translates felt force to a velocity, not an acceleration, the velocity vector is approximated as the addition between the user force vector $\check{\boldsymbol{f}}_{\mathrm{U}}$ with the assistance forces $\boldsymbol{f}_{\mathrm{T}}$ received from the HSC controller, multiplied with a scaling gain k_s.

$$\dot{\check{\boldsymbol{p}}}_{\mathrm{T}} = k_s(\check{\boldsymbol{f}}_{\mathrm{U}} + \boldsymbol{f}_{\mathrm{T}}) \tag{3-18}$$

This equation is used to simulate the effect of the application of the combined forces in the system. Finally, this velocity vector is multiplied with the delta time and added to the current simulated point to get the next simulated point. This process is repeated every time step simulating the virtual trajectory.

$$\check{\boldsymbol{p}}_{\mathrm{T}}(t+\Delta t) = \check{\boldsymbol{p}}_{\mathrm{T}} + \dot{\check{\boldsymbol{p}}}_{\mathrm{T}}\Delta t\,. \tag{3-19}$$

Figure 3.13 illustrates the basic functioning of the master-side simulator. In this way, the Master-side simulator provides a position vector $\check{\boldsymbol{p}}_{\mathrm{T}}$ and receives the assistance forces $\boldsymbol{f}_{\mathrm{T}}$ from the controller.

3.4.1.2
Slave-side simulator

On the other hand, the functioning of the Slave-side simulator is more simple. Since the slave side *replicates* the movements of the master side, only the velocity commands and the initial and final points are necessary. Similarly to the Master-side simulator, the mean of the samples' initial points $(\bar{x_0}, \bar{y_0}, \bar{z_0})$ and the mean of the samples' final points $(\bar{x_j}, \bar{y_j}, \bar{z_j})$ can be calculated from a set of demonstrated trajectories for the manipulator case becoming in the initial position and the *goal* position.

Then, at time t, given a certain position for the manipulator $\check{\boldsymbol{p}}_{\mathrm{D}}$, the

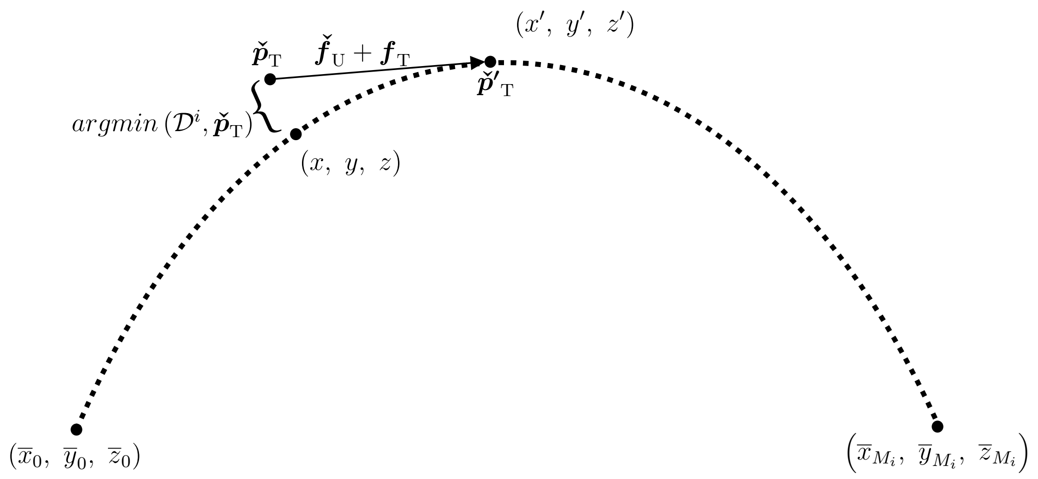

(x', y', z')
$\check{\boldsymbol{p}}_\mathrm{T}$
$\check{\boldsymbol{f}}_\mathrm{U} + \boldsymbol{f}_\mathrm{T}$
$\check{\boldsymbol{p}}'_\mathrm{T}$
$argmin\,(\mathcal{D}^i, \check{\boldsymbol{p}}_\mathrm{T})$
(x, y, z)
$(\overline{x}_0, \overline{y}_0, \overline{z}_0)$
$(\overline{x}_{M_i}, \overline{y}_{M_i}, \overline{z}_{M_i})$

Figure 3.13: Master-side simulator: Once the reference point w.r.t the sampled trajectory is calculated with the *argmin* function, the virtual user force is obtained through a proportional difference between a desired and the current position.

next position can be calculated simply by using the Euler integration method, similar to (3-19). Thereby, the virtual manipulator position $\check{\boldsymbol{p}}_\mathrm{D}$ is sent to the HSC controller. Similar with the master-side simulator, this process is repeated every time step until the current manipulator position reaches the *goal* point, where the episode ends and the trajectory is restarted.

$$\check{\boldsymbol{p}}_\mathrm{D}(t + \Delta t) = \check{\boldsymbol{p}}_\mathrm{D} + \dot{\check{\boldsymbol{p}}}_\mathrm{D}\Delta t\,. \tag{3-20}$$

A complete architecture for the simulator basic functionalities is shown in Figure 3.14.

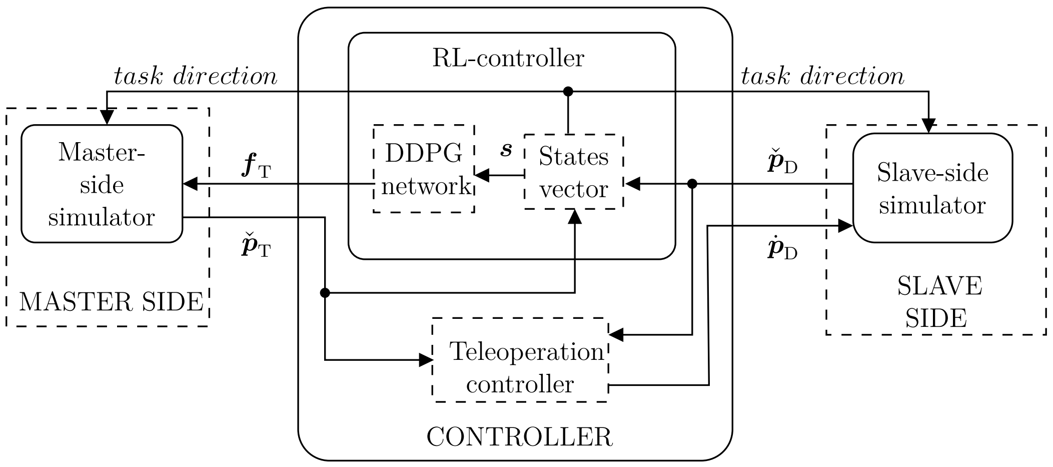


Figure 3.14: Simulator basic functionality: Based on the real system, the simulator replicates the master and the slave side functioning providing the information to the agent for the training.

4
EXPERIMENTAL SETUP

This chapter presents the experimental implementation of the proposed *HSC-controller* presented in the last chapter. Technical specifications of software and hardware are first presented followed by the explanation for the basic implementation of the teleoperation and the RL-controllers in the simulator and real system respectively.

4.1
Hardware and software specifications

The implemented system is composed of three components: the master side, the central controller and the slave side. For the real system, the master side is composed of the 6 DOF Touch 3D haptic device from 3D Systems (Figure 4.1(a)); in the slave side is used the 4 DOF robotic manipulator Dobot Magician from Dobot (Figure 4.1(b)). On the other hand, for the simulation, the master and the slave side were replaced by two programs implemented following the specifications given in Section 3.4. For both implementations, the real system and the simulator, a computer was used as a central controller. This computer is a desktop workstation with a 4-core Intel® Core™ i7-7700 CPU @ 3.60GHz as the processor, 16 GB of RAM, GeForce GTX 1080 as graphic card and Ubuntu 18.04.3 LTS as operating system. The Robot Operating System (ROS) Melodic was the middleware used for communication between the various software components in the system. The main programming language used to implement the controller codes was Python 2.7 with the exception of the Touch device whose controller was written in C++. The ROS controller for the Touch device and the Dobot magician were adapted from [61] and [62] respectively. The artificial neural networks used in this research were implemented using the Keras libraries and trained using the Tensorflow backend. Finally, for the visual tests an HD USB webcam from Logitech was used[63].

Although both devices are kinematically similar, differences are found in the workspace level and the axis configuration (both axes are displaced and rotated in relation to each other). For this reason, both devices are treated as dissimilar and the position control is performed at tip level following the

(a)

(b)

Figure 4.1: Input/output devices. (a): Touch Haptic Device from 3D Systems used in the master side, taken from [32] and (b): Dobot Magician robotic arm from Dobot.cc used in the slave-side, taken from [64].

concepts presented in Section 2.1.1.2. First of all, axis translation and rotation were performed on the master side with respect to the slave side as shown in Figure 4.2. Thereby, both devices have the same axis reference located in their *base link* (same origin). In addition, both workspaces were mapped using a scaling factor ξ as given in (2-4) which will be repeated here for the reader convenience: $\boldsymbol{p}_{\mathrm{Sd}} = \xi \boldsymbol{p}_{\mathrm{M}} + \boldsymbol{p}_{\mathrm{offset}}$. This value was selected following the recommendation to provide comfort in user control [21]. In what follows we describe all the needed operations in order to have both devices coupled in the same orientation and scale.

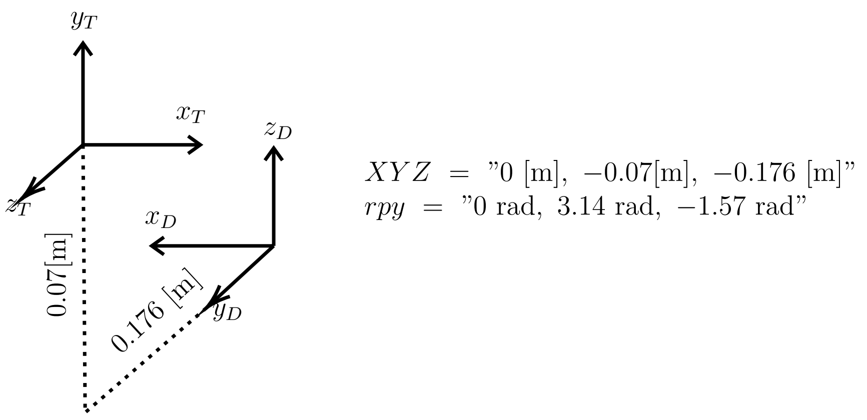
$$\begin{bmatrix} x_{\mathrm{Dd}} \\ y_{\mathrm{Dd}} \\ z_{\mathrm{Dd}} \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & \xi & 0.2464 \\ 0 & 1 & 0 & 0.098 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{T}} \\ y_{\mathrm{T}} \\ z_{\mathrm{T}} \\ 1 \end{bmatrix} \tag{4-1}$$

with $\xi = 1.4$.

4.1.1 Teleoperation controller implementation

To perform direct teleoperation, the *teleoperation controller* code was implemented following the specifications given in Section 3.2.1. This controller allows the slave side to track the reference point given by the current position of the master side using position control. This functioning is valid for the real system and simulator equally.

The teleoperation-controller uses a proportional linear control where the robot velocities $\dot{\boldsymbol{p}}_{\mathrm{D}}$ are calculated based on the tracking error vector $\boldsymbol{e}$



y_T
x_T
z_T
z_D
x_D
y_D
0.07[m]
0.176 [m]
XYZ = "0 [m], −0.07[m], −0.176 [m]"
rpy = "0 rad, 3.14 rad, −1.57 rad"

Figure 4.2: Axis displacement and rotation presented between the Touch haptic device and the robotic manipulator Dobot magician.

multiplied with a positive proportional gain $k_D = 3$ following the control law given in (3-3).

Summarizing, the basic functioning of the teleoperation controller is as follows, the tip positions for both sides (master and slave) are mapped every time-step in Cartesian coordinates, while the controller sends velocity commands proportional with the calculated error between both positions to the slave side (Algorithm 2). In addition, for the real system, one of the user buttons located in the pen of the touch was used to activate the suction motor to grip the ball on the remote side. Thereby, the suction motor is only activated while the white button is pressed on the master side, otherwise, the suction motor is always inactive.

Algorithm 2 Teleoperation-controller

```
function TELEOPERATION-CONTROLLER()
    while True do
        Read $\boldsymbol{p}_\mathrm{T}$, $\boldsymbol{p}_\mathrm{D}$
        $x_\mathrm{Dd} \leftarrow -x_\mathrm{T}$
        $y_\mathrm{Dd} \leftarrow 1.4 z_\mathrm{T} + 0.2464$
        $z_\mathrm{Dd} \leftarrow y_\mathrm{T} + 7$
        $\boldsymbol{p}_\mathrm{Dd} \leftarrow [x_\mathrm{Dd},\ y_\mathrm{Dd},\ z_\mathrm{Dd}]$
        $\boldsymbol{e} = \boldsymbol{p}_\mathrm{Dd} - \boldsymbol{p}_\mathrm{D}$
        $\dot{\boldsymbol{p}}_\mathrm{D} = k_D \boldsymbol{e}$
        Send $\dot{\boldsymbol{p}}_\mathrm{D}$ to Dobot
```

4.1.2 RL-controller implementation

Chapter 3 presented the RL-controller. This controller is responsible for providing the assistive forces exerted on the master side in order to help the user in the task completion. This controller is based on the DDPG algorithm presented in Chapter 2 and the implementation was adapted from the configuration described in the supplementary information section of [44]. For the preliminary tests, the network architecture is the same as the example found in the original paper.

Figure 4.3 presents the network architecture for the implemented actor-critic DDPG algorithm. As we can see, the actor network consists of an ANN with three layers of 400, 300 and 3 units with *ReLU* activation for the first two layers and *tanh* activation in the last layer respectively. Moreover, the input size is a batch of state vectors $\boldsymbol{s}$ described in Section 3.2.2.1 and the output assistive force vectors $\boldsymbol{f}_{\mathrm{T}}$.

In contrast, the critic network is composed of three layers of 400, 300 and 1 units with *ReLU* activation for the first two layers and *linear* activation in the last layer respectively. The input size is similar to the actor network with the difference that the action vector $\boldsymbol{a}$ is concatenated in the second hidden layer. The critic network output is the set of q-values for the given input state vectors.

On the other hand, the replay buffer $\mathcal{R}$ is a set of tuples, where each tuple is composed of previous observation $\boldsymbol{s}$, the action $\boldsymbol{a}$, the reward r, the current observation $\boldsymbol{s}'$ and an extra element termed A which indicates if the state was terminal or not:

$$\mathcal{R} \leftarrow \mathcal{R} \cup \{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', \mathrm{A})\} \tag{4-2}$$

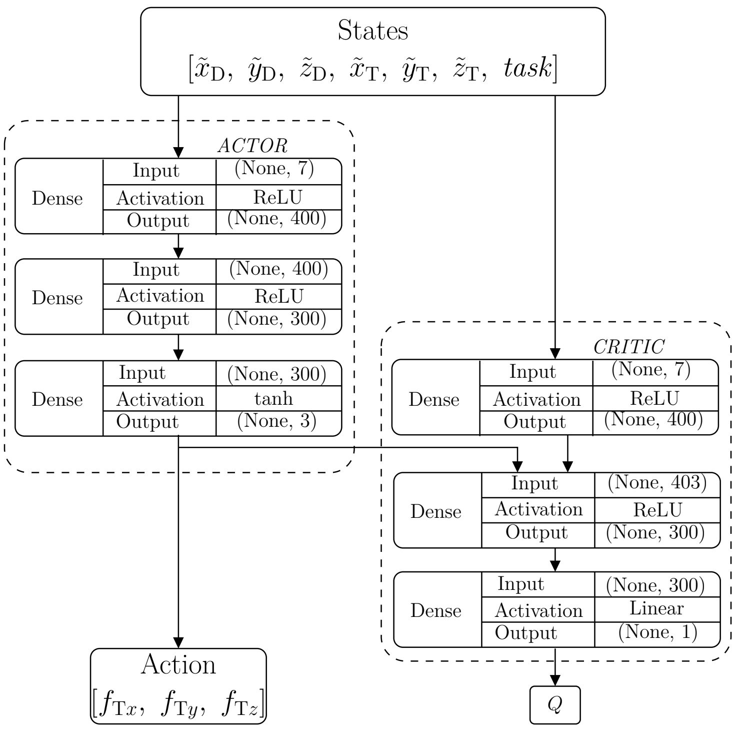

States
$[\tilde{x}_{\mathrm{D}},\ \tilde{y}_{\mathrm{D}},\ \tilde{z}_{\mathrm{D}},\ \tilde{x}_{\mathrm{T}},\ \tilde{y}_{\mathrm{T}},\ \tilde{z}_{\mathrm{T}},\ task]$
ACTOR
Dense | Input (None, 7) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation ReLU | Output (None, 300)
Dense | Input (None, 300) | Activation tanh | Output (None, 3)
CRITIC
Dense | Input (None, 7) | Activation ReLU | Output (None, 400)
Dense | Input (None, 403) | Activation ReLU | Output (None, 300)
Dense | Input (None, 300) | Activation Linear | Output (None, 1)
Action
$[f_{\mathrm{T}x},\ f_{\mathrm{T}y},\ f_{\mathrm{T}z}]$
Q

Figure 4.3: DDPG original architecture implemented.

Finally, according to the Dobot Magician manufacturer [65], the maximum exerted forces are 3.3 N. For this reason, to prevent possible damage to the device, the output forces were bounded following two different approaches: using normalized forces or bounded variable forces. For the first approach, the guiding force vector was normalized according to (4-3) such that every time-step the force vector has a constant magnitude of 1 N.

$$\hat{\boldsymbol{f}_{\mathrm{T}}} = \frac{\boldsymbol{f}_{\mathrm{T}}}{\|\boldsymbol{f}_{\mathrm{T}}\|} \,. \tag{4-3}$$

For the second approach, variable forces clipped in a safe range of [-1 N, 1 N] were used as following:

$$\boldsymbol{f}_{\mathrm{T_{min}}} \leq \boldsymbol{f}_{\mathrm{T}} \leq \boldsymbol{f}_{\mathrm{T_{max}}} \,. \tag{4-4}$$

with $\boldsymbol{f}_{\mathrm{T_{min}}} = -1$ N and $\boldsymbol{f}_{\mathrm{T_{max}}}$ =1 N.

Figure 4.4 shows the mounted system with the haptic device, the robot manipulator and the computer.

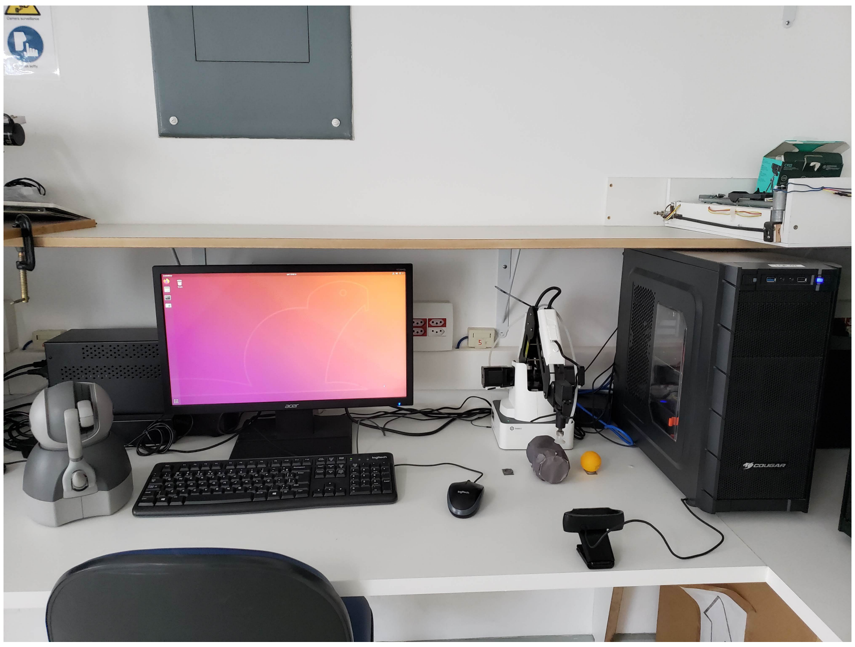


Figure 4.4: Mounted system.

5 SIMULATION ANALYSIS

This chapter presents the various experiments carried out using the simulator. Starting from the preliminary tests, as well as the different modifications used. The experiments are presented with the description of the network and the set of parameters used, followed by the obtained results and a brief discussion of them. Table 5.1 summarizes the simulations performed in this Chapter.

Table 5.1: Simulations summary.

Experiment	**Description**
Preliminary tests	Hyperparameter search and first tests performed to demonstrate the performance and functioning of the proposed approach. (See Section 5.1)
Behavior cloning approach as initialization	Simulations using each FC-ANNs described in Chapter 3 as initialization using the BC approach to analyze the performance effects of its application. (See Section 5.2)

5.1 Preliminary tests

As preliminary tests, the simulator was first applied in order to perform the parameter search and choosing the network architecture to be used in the real system implementation. Following the specifications given in previous chapters, different experiments were performed with the simulator varying the different hyperparameters for the DDPG network: batch size, discount rate γ, and the update target parameter τ. In the same way, the parameters for the AR-p process were also varied: the order p and the parameter α. The complete set of parameters and the variations are summarized in Table 5.3. Table 5.2 shows the parameters used for the master side simulator. As task detector, the manual task selection approach was modified. Thereby, simulations were performed selecting the task automatically at the beginning of the episodes, this modification is presented in Algorithm 3. In addition, as every episode of the simulation is composed of the task execution in one direction, therefore, since the task is being performed in both directions, we can consider two episodes as an epoch. An extra consideration was the addition of a testing epoch every 10 training epochs and the condition that the simulator always finished successfully. This condition remains in the fact that the master side simulator bases its trajectory in a set of successful demonstrations. For this reason, in the final part of the execution of the simulated trajectory, its final position is always proximal to the goal position, which leads to a successful execution.

5.1.1 Results

Figure 5.1 shows the results obtained for preliminary tests. These graphs are a mean of a set of five simulations. The solid line represents the mean value and the shaded area represents the standard error for rewards and steps graphs respectively.

5.1.2 Discussion

As show the results of the preliminary tests simulated for the proposed system, the implemented algorithm achieves convergence in a limited amount of episodes. Practically, after 15 epochs approximately the DDPG agent is able to learn an acceptable policy presenting stable functioning in training

Table 5.2: Parameters used for master-side simulator. These values were obtained using a similar technique as in Table 5.3

Parameter	Value
Step	12
User gain (k_u)	5
Force gain (k_s)	5
Sampling time (Δt)	0.01

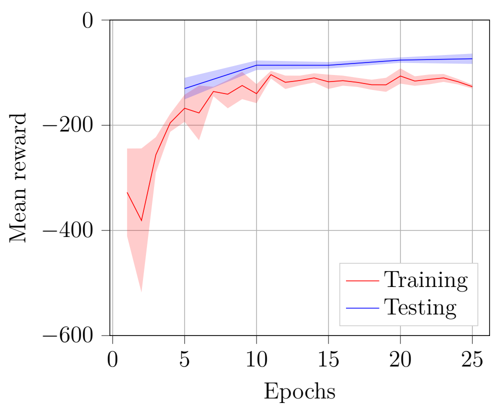
Table 5.3: Parameters used for DDPG simulation.

Parameter	Min	Max	Best value
Batch size (N)	32	128	64
Discount rate (γ)	0.96	0.99	0.99
Update target parameter (τ)	0.001	0.01	0.01
Assistance forces	-	-	Normalized
Actor optimizer	-	-	Adam
Actor learning rate	0.0001	0.001	0.001
Critic optimizer	-	-	Adam
Critic learning rate	0.0001	0.001	0.0001
Epochs	15	100	25
AR-p order (p)	2	3	3
AR-p parameter (α)	0.8	0.9	0.8

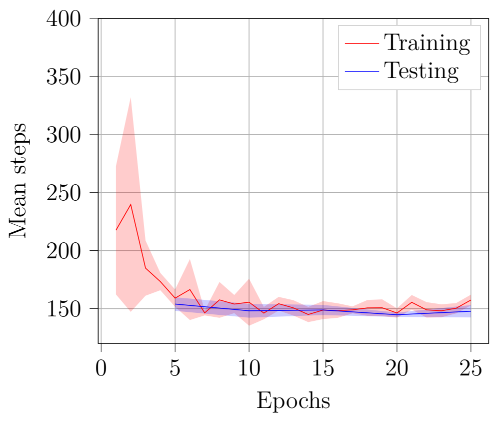
Algorithm 3 DDPG training in simulation

```
function DDPG($\gamma, \tau, M, N$)
    Initialize $\boldsymbol{w}, \boldsymbol{\theta}, \boldsymbol{w}^\dagger, \boldsymbol{\theta}^\dagger$ with random values close to zero
    $n \leftarrow 0,\ t \leftarrow 0,\ task \leftarrow 0$
    for episode=1,M do
        $\boldsymbol{s} \leftarrow$ Start for $task$
        while $\boldsymbol{s}$ is not $\boldsymbol{s}_n$ do
            Select action $\boldsymbol{a}_t$ from $\hat{\mu}(\boldsymbol{s}_t; \boldsymbol{\theta}) + \mathcal{X}_t$
            Execute action $\boldsymbol{a}_t$, observe reward $r$ and observe new state $\boldsymbol{s}'_t$
            Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t, 0)$ in $\mathcal{R}$
            $\boldsymbol{s}_t \leftarrow \boldsymbol{s}'_t$
            $n \leftarrow n + 1$
            $t \leftarrow t + 1$
        $r_n \leftarrow r_n + 10$
        Store transition $(\boldsymbol{s}_n, \boldsymbol{a}_n, r_n, \boldsymbol{s}'_n, 1)$ in $\mathcal{R}$
        for steps=1,n do
            TRAIN-MINIBATCH($\mathcal{R}, \gamma, \tau, \boldsymbol{\theta}, \boldsymbol{\theta}^\dagger, \boldsymbol{w}, \boldsymbol{w}^\dagger, N$)
        $task \leftarrow$ not $task$

function TRAIN-MINIBATCH($\mathcal{R}, \gamma, \tau, \boldsymbol{\theta}, \boldsymbol{\theta}^\dagger, \boldsymbol{w}, \boldsymbol{w}^\dagger, N$)
    Sample a random minibatch $\mathcal{B} \subset \mathcal{R}$ of size $N$
    for each $b_i : (\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i, \mathrm{A}_i) \in \mathcal{B}$ do
        if $\mathrm{A}_i = 0$ then
            $y_i = r_i + \gamma \hat{Q}(\boldsymbol{s}'_i, \hat{\mu}(\boldsymbol{s}'_i; \boldsymbol{\theta}^\dagger); \boldsymbol{w}^\dagger)$
        else
            $y_i = r_i$
    Train $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w})$ on all samples $N$
    Move $\hat{\mu}(\boldsymbol{s}; \boldsymbol{\theta})$ according to the sampled deterministic policy gradient
    $\frac{1}{N} \sum_{i=1}^{N} [\nabla_{\boldsymbol{a}} \hat{Q}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w})|_{\boldsymbol{s}=\boldsymbol{s}_i, \boldsymbol{a}=\hat{\mu}(\boldsymbol{s}_i; \boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \hat{\mu}(\boldsymbol{s}; \boldsymbol{\theta})|_{\boldsymbol{s}=\boldsymbol{s}_i}]$
    $\boldsymbol{\theta}^\dagger \leftarrow \boldsymbol{\theta}^\dagger + \tau(\boldsymbol{\theta} - \boldsymbol{\theta}^\dagger)$
    $\boldsymbol{w}^\dagger \leftarrow \boldsymbol{w}^\dagger + \tau(\boldsymbol{w} - \boldsymbol{w}^\dagger)$
```



(a) Epoch rewards.



(b) Steps per epoch.

Figure 5.1: Learning curves obtained for DDPG preliminary test.

and testing episodes (Figure 5.1(a)). This was confirmed using the necessary amount of steps required to perform the task shown in Figure 5.1(b), where it can be observed that the steps decrease along the training w.r.t the first episodes.

5.2 Behavior cloning (BC) as initialization

For the presented system, BC as initialization was tested by training two ANN named BC-networks: using the DDPG states vector $\boldsymbol{s}$ as inputs and as outputs the Dobot velocities ($\dot{\boldsymbol{p}}_{\mathrm{D}}$) or the touch velocities ($\dot{\boldsymbol{p}}_{\mathrm{T}}$) respectively. Previously, a set of $N = 14$ demonstrations for every task direction were recorded to build the needed dataset $\mathcal{D}$ used to train the networks. The dataset was composed of information of the scaled dobot tip position $\tilde{\boldsymbol{p}}_{\mathrm{D}} = [\tilde{x}_{\mathrm{D}},\ \tilde{y}_{\mathrm{D}},\ \tilde{z}_{\mathrm{D}}]$, the scaled touch tip position $\tilde{\boldsymbol{p}}_{\mathrm{T}} = [\tilde{x}_{\mathrm{T}},\ \tilde{y}_{\mathrm{T}},\ \tilde{z}_{\mathrm{T}}]$ and the task direction *task*. Finally, to encode the task, a signed integer was used for both directions: *left* $= 1$ and *right* $= -1$. Thereby:

$$\mathcal{D} = \{\{(\tilde{\boldsymbol{p}}_{\mathrm{D}}, \tilde{\boldsymbol{p}}_{\mathrm{T}}, task)_0^i,\ ..,\ (\tilde{\boldsymbol{p}}_{\mathrm{D}}, \tilde{\boldsymbol{p}}_{\mathrm{T}}, task)_{M_i}^i\}\}_{i=1}^N,\ M_i > 0\,. \tag{5-1}$$

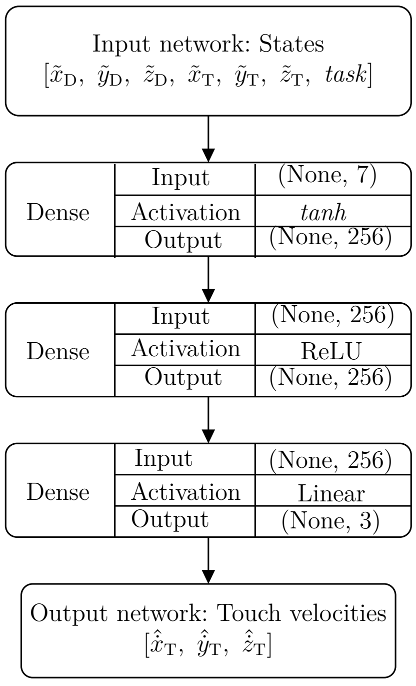
$$task = \begin{cases} 1 & \text{, if task follows left direction} \\ -1 & \text{, if task follows right direction} \end{cases} \tag{5-2}$$

The architecture used for the ANN consisted of a fully connected network of two hidden layers of 256 units with *tanh* and ReLU activation respectively, and an output layer of 3 units with linear activation as presents Figure 5.2. Moreover, Table 5.4 summarizes the parameters used to train the networks.

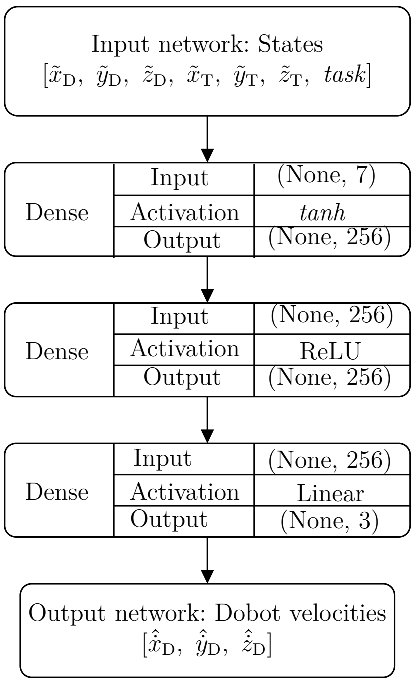
For the experiments, the BC networks were used in the first four episodes. Some modifications with respect to Algorithm 3 were performed as presents Algorithm 4. The same considerations as in the preliminary tests were used in this experiment (set of five simulations, episode composition and training/testing episodes, parameters).

Table 5.4: Parameters used for BC network training. These values were obtained using a similar technique as in Table 5.3

Parameter	**Value**
Optimizer	Adam
Learning rate	0.001
Loss	Mean squared error
Batch size	32
ANN epochs	100



(a) ANN with $\hat{\dot{\boldsymbol{p}}}_T$ as output.



(b) ANN with $\hat{\dot{\boldsymbol{p}}}_D$ as output.

Figure 5.2: ANN architecture implemented for BC approach.

Algorithm 4 DDPG simulator with initialization

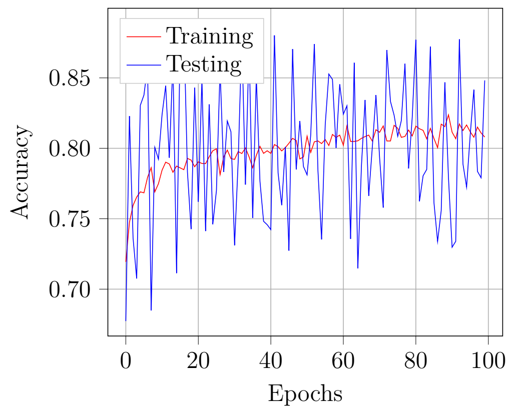
```
function DDPG($\gamma, \tau, M, N$)
    Initialize $\boldsymbol{w}, \boldsymbol{\theta}, \boldsymbol{w}^\dagger, \boldsymbol{\theta}^\dagger$ with random values close to zero
    $n \leftarrow 0$, $t \leftarrow 0$, $task \leftarrow 0$
    for episode=1,M do
        $\boldsymbol{s}_t \leftarrow$ Start for $task$
        while $\boldsymbol{s}$ is not $\boldsymbol{s}_n$ do
            if episode $< 4$ then
                Select action $\boldsymbol{a}_t$ from BC-network
            else
                Select action $\boldsymbol{a}_t$ from $\hat{\mu}(\boldsymbol{s}_t; \boldsymbol{\theta}) + \mathcal{X}_t$
            Execute action $\boldsymbol{a}_t$, observe reward $r$ and observe new state $\boldsymbol{s}'_t$
            Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t, 0)$ in $\mathcal{R}$
            $\boldsymbol{s}_t \leftarrow \boldsymbol{s}'_t$
            $n \leftarrow n + 1$
            $t \leftarrow t + 1$
        $r_n \leftarrow r_n + 10$
        Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t, 1)$ in $\mathcal{R}$
        for steps=1,n do
            TRAIN-MINIBATCH($\mathcal{R}, \gamma, \tau, \boldsymbol{\theta}, \boldsymbol{\theta}^\dagger, \boldsymbol{w}, \boldsymbol{w}^\dagger, N$)
        $task \leftarrow$ not $task$
```

5.2.1 Results

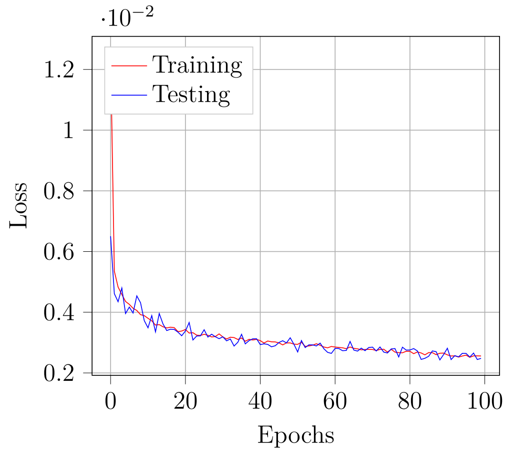
Figures 5.3 and 5.4 present the training results for accuracy and loss metrics for both BC networks used. Besides, Figures 5.5 and 5.6 present the resulting learning curve for both BC-networks showing the mean (solid line) and the standard error (shaded area) in training and testing for rewards and steps respectively.

5.2.2 Discussion

The obtained results using BC-networks as initialization present a stable functioning after approximately 15 epochs as in the previous case (without

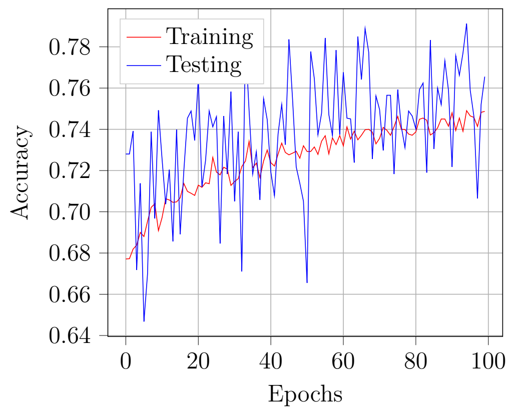


(a) Accuracy.

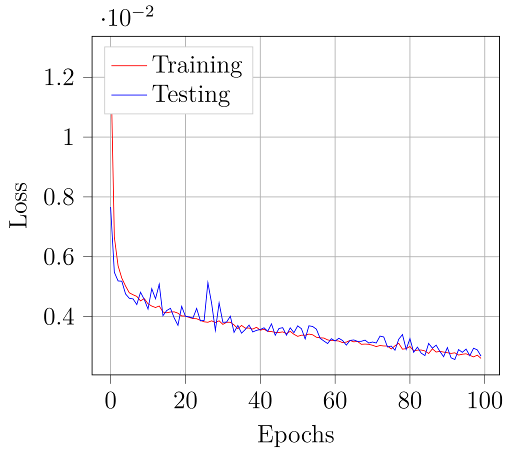


(b) Loss.

Figure 5.3: Learning curves obtained for BC network with $\hat{\boldsymbol{p}}_{\mathrm{D}}$ as outputs.

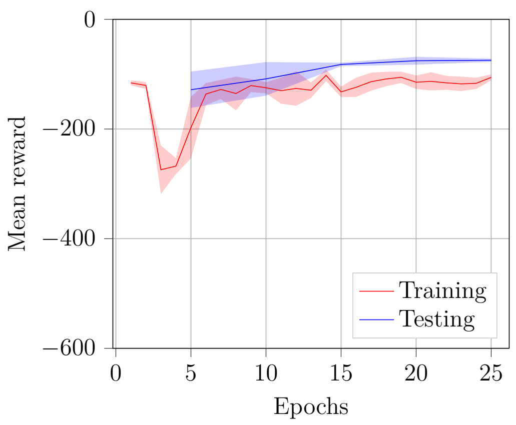


(a) Accuracy.

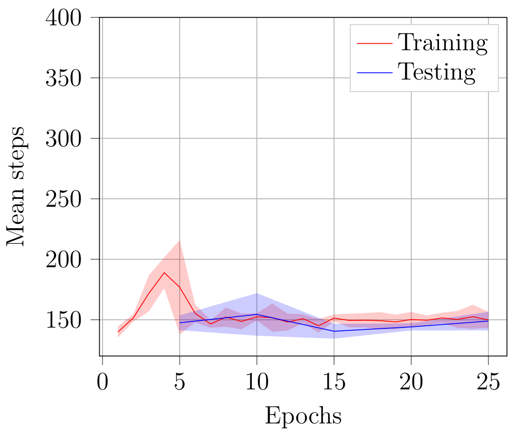


(b) Loss.

Figure 5.4: Learning curves obtained for BC network with $\hat{\boldsymbol{p}}_{\mathrm{T}}$ as outputs.

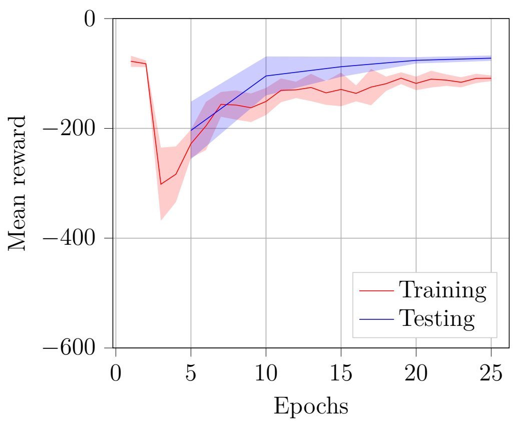


(a) Epoch rewards.

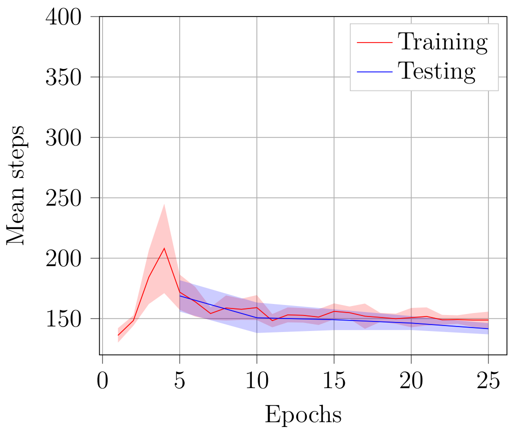


(b) Steps per epoch.

Figure 5.5: Learning curves obtained for DDPG using BC with $\hat{\boldsymbol{p}}_{\mathrm{D}}$ as outputs.



(a) Epoch rewards.



(b) Steps per epoch.

Figure 5.6: Learning curves obtained for DDPG using BC with $\hat{\boldsymbol{p}}_{\mathrm{T}}$ as outputs.

initialization tests). Despite the initialization, the obtained rewards in the final episodes are not better than the first tests of section 5.1 and the obtained behavior is similar as in the previous section.

5.3
General discussion

Table 5.5 summarizes the obtained final results for all the experiments described in this chapter. The presented values are the results (mean and standard error) in the last testing epoch for the rewards and number of steps per episode respectively. As can be observed, the mean value is almost the same but the standard error is a bit different, particularly with respect to rewards. So, it is possible to say that results for both approaches (with and without initialization) present statistically the same results. In addition, the learning curves in all the presented graphs illustrate stable functioning for the agent in the final episodes of the training. As was mentioned in the last paragraph, despite the use of BC as initialization, the agent was not able to achieve better performance than an agent that was trained without initialization, nor does it train faster. This affirmation can be confirmed by observing Figures 5.1, 5.5 and 5.6 and the results in Table 5.5. Following these affirmations, it is possible to say that using BC as initialization does not present a high influence in the proposed system. Finally, it can be concluded that the use of the proposed system presents a stable convergence in the learning of the virtual guiding forces to assist users in the *pick-and-place* task realization.

Table 5.5: Comparison of the results obtained in the last testing epoch for the different DDPG approaches implemented

	Rewards		**Steps**	
	Mean	**Std. Error**	**Mean**	**Std. Error**
Without initialization	-73.607	8.608	147.7	4.676
Using $\dot{\boldsymbol{p}}_{\mathrm{D}}$ as initialization	-74.642	2.586	148.9	6.752
Using $\dot{\boldsymbol{p}}_{\mathrm{T}}$ as initialization	-72.141	4.27	141.6	4.049

6 EXPERIMENTAL ANALYSIS

Chapter 5 showed preliminary results demonstrating that the learned policy of the DDPG algorithm in the teleoperated simulated system presents stable convergence. Those results serve as a basis for the application of the proposed method in a real system. This chapter presents the implementation and results in the real system described in Chapter 4 validating the obtained results. Next, different stages of improvement are described until the final implementation described in Chapter 3 is reached (See Table 6.1). Finally, a set of experiments using different subjects was carried out in order to validate the implemented controller.

6.1 Preliminary considerations

For the real system setup, some considerations were added in contrast with the simulator. First of all, the main difference lies in the fact that the simulations always finish in a successful episode, however, in the real system, this condition does not always hold. For that reason, it was necessary to implement a conditional function that described the episode behavior. Three conditions were considered: successful task completion, unsuccessful task completion and finally, task fail and repeat episode. *Successful task completion* refers to when the user completes the task placing the object in the goal position. *Unsuccessful task completion* refers to when the user completes the task but places the object in a wrong position. Finally, if the ball is dropped or an unexpected event occurs, then the task fails and the episode needs to be restarted (See Figure 6.1).

Thereby, after performing every task execution, the operator was asked to select the option that best described his performance in that execution. In this context, conditions presented in Equation 3-8 were added in the absorbing state assigning different rewards according to the described conditions; that is, if the user chooses the first option, an extra high reward ($+10$) is assigned for the absorbing state. In contrast, if it was an unsuccessful episode and the user chooses the second option, a high penalty reward (-10) is assigned to the absorbing state in that episode. Finally, if the third option is selected, then

Table 6.1: Experiments summary.

Approach	Description
Manual task selection (MTS)	Experiments performed to validate the results obtained with the simulator in the real system. The task is selected manually through the terminal and the policy is trained using a DDPG network. (See Section 6.2)
Supervised task detection (STD)	Experiments performed replacing the manual task selection with a visual task detector using a CNN previously trained with a small dataset of the selected task. (See Section 6.3)
End-to-End learning (E2E)	Experiments performed enhancing the STD approach appending the CNN network to the DDPG network in order to detect the task autonomously. (See Section 6.4)
DDPG: Single-shot task detection (SSTD)	Experiments performed modifying the E2E to decrease the number of trainable parameters and the inference time, using for that, only the first frame of the task and a modified architecture of the network. (See Section 6.5)
SSTD with variable forces	Experiments performed using variable forces to decrease the oscillations appeared in previous approaches in the task execution. (See Section 6.6)
SSTD with variable forces and fuzzy-based reward function	Experiments performed using all the improvements achieved in previous approaches and a new reward function based on fuzzy rules to eliminate completely undesired behaviors in the task execution. (See Section 6.7)

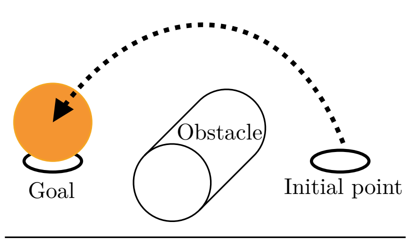
the episode is repeated from the last start position and the collected samples in the current episode are discarded from the replay buffer memory.

As a final consideration, similar to the simulator, an epoch was considered as a set of two episodes, a different task direction for each one.

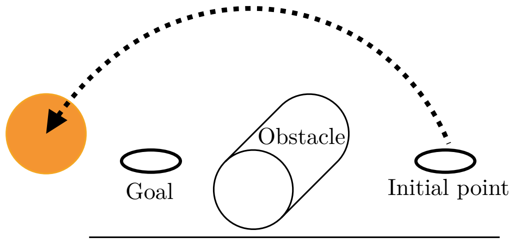
In addition, Figure 6.2 shows a successful execution of the task following the way from left to right and using direct teleoperation without force feedback. One observable feature that becomes apparent in this figure is the delay presented in the trajectory described by the robotic arm with respect to the haptic device.

6.2 Manual task selection (MTS) approach

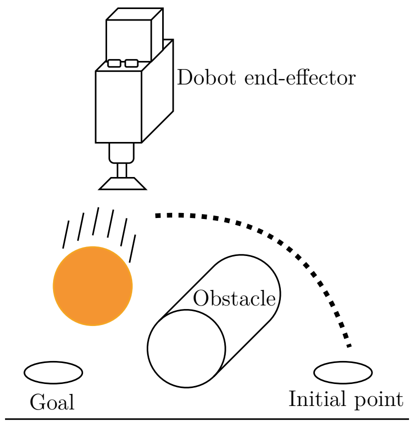
To validate the results presented in Chapter 5, only one set of experiments was carried out performing manual task selection (MTS). In simulations, task directions were automatically encoded at the start of the episodes and



(a) Successful task completion.



(b) Unsuccessful task completion.



(c) Task fail and repeat task episode.

Figure 6.1: Episode events conditions.

sent to the different parts of the implementation. For the real system, the task direction can be manually selected using the terminal through a prompt input. Thereby, the user can write the value he wants to encode the task direction he is going to perform. For this implementation, Algorithm 3 was slightly modified to add the manual task selection method as presented in Algorithm 5. In addition, according the hyperparameters found in the previous chapter, Table 6.2 summarizes the hyperparameters used in this implementation which are practically the same as the presented in Table 5.3. Unlike the simulator, only one set of experiments was performed using this approach to validate the use of the proposed controller in the real system. The remaining considerations (epochs composition and training/testing episodes) were maintained. Moreover, since the BC approach does not provide better performance than without initialization, as shown in Table 5.5, its use was dismissed for the presented approaches in this Chapter.

6.2.1 Results

Figure 6.3 shows the learning curve for the rewards and the number of steps for the MTS approach. Besides presented graphs, numerical results are

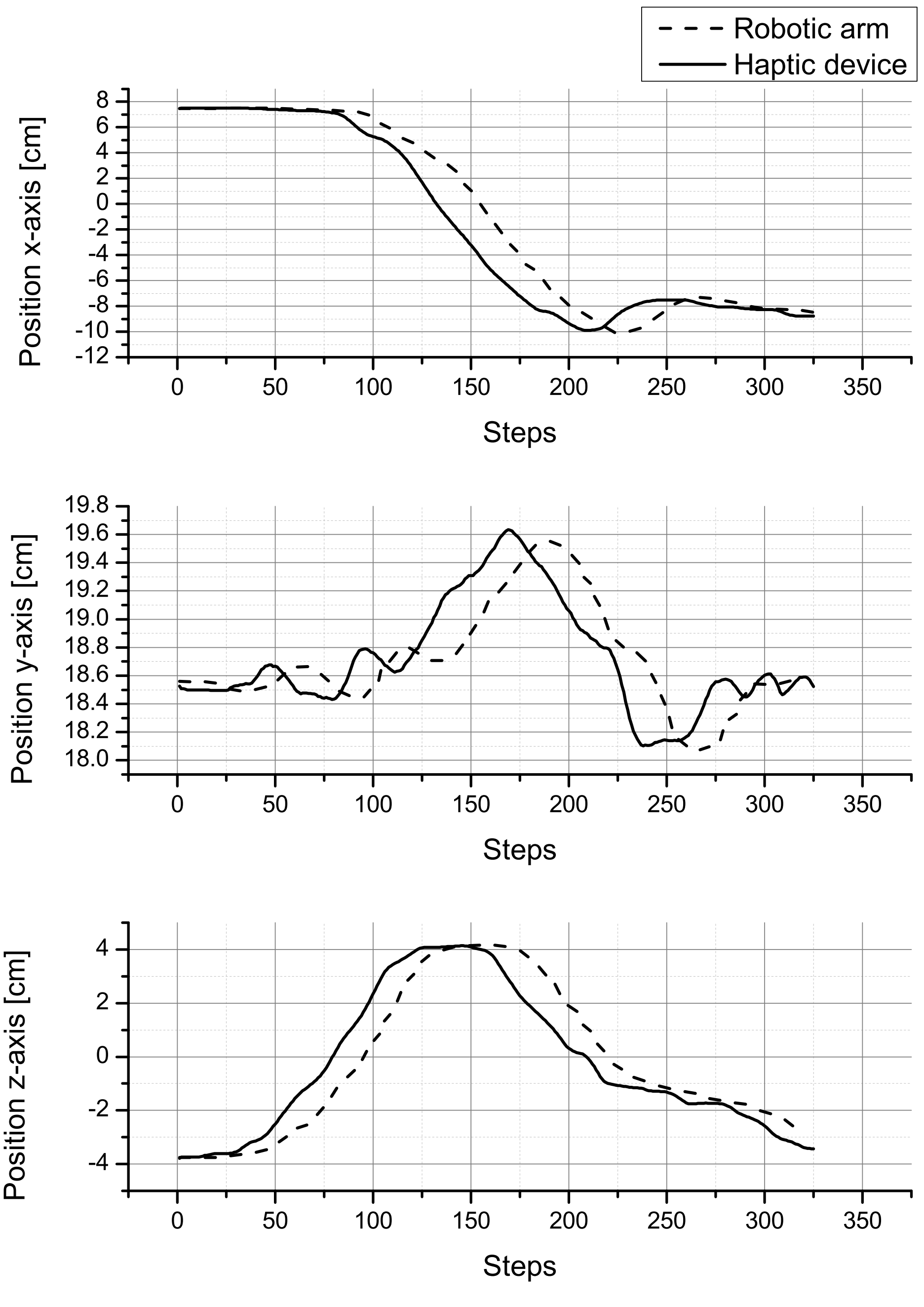


Figure 6.2: Direct teleoperation without force feedback. Notice the delay presented in the trajectory described of both devices.

Table 6.2: Parameters used for MTS approach.

Parameter	**Value**
Batch size (N)	64
Discount rate (γ)	0.99
Update target parameter (τ)	0.01
Assistance forces	Normalized
Actor optimizer	Adam
Actor learning rate	0.001
Critic optimizer	Adam
Critic learning rate	0.0001
Epochs	25
AR-p order (p)	3
AR-p parameter (α)	0.9

Algorithm 5 DDPG used with MTS

```
function DDPG($\gamma, \tau, M, N$)
    Initialize $\boldsymbol{w}, \boldsymbol{\theta}, \boldsymbol{w}^\dagger, \boldsymbol{\theta}^\dagger$ with random values close to zero
    $n \leftarrow 0$, $t \leftarrow 0$, $task \leftarrow 0$
    for episode=1,M do
        Get $task$ from user
        $\boldsymbol{s} \leftarrow$ Start for $task$
        while $\boldsymbol{s}$ is not $\boldsymbol{s}_n$ do
            Select action $\boldsymbol{a}_t$ from $\hat{\mu}(\boldsymbol{s}_t; \boldsymbol{\theta}) + \mathcal{X}_t$
            Execute action $\boldsymbol{a}_t$, observe reward $r$ and observe new state $\boldsymbol{s}'_t$
            Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t, 0)$ in $\mathcal{R}$
            $\boldsymbol{s}_t \leftarrow \boldsymbol{s}'_t$
            $n \leftarrow n + 1$
            $t \leftarrow t + 1$
        DEFINE BEHAVIOR()
        Store transition $(\boldsymbol{s}_n, \boldsymbol{a}_n, r_n, \boldsymbol{s}'_n, 1)$ in $\mathcal{R}$
        for steps=1,n do
            TRAIN-MINIBATCH($\mathcal{B}, \gamma, \tau, \boldsymbol{\theta}, \boldsymbol{\theta}^\dagger, \boldsymbol{w}, \boldsymbol{w}^\dagger, N$)

function DEFINE BEHAVIOR()
    if episode was successful then
        $r_n \leftarrow r_n + 10$
    else if episode was unsuccessful then
        $r_n \leftarrow r_n - 10$
    else
        Discard samples from current episode
        episode $\leftarrow$ episode $-1$
```

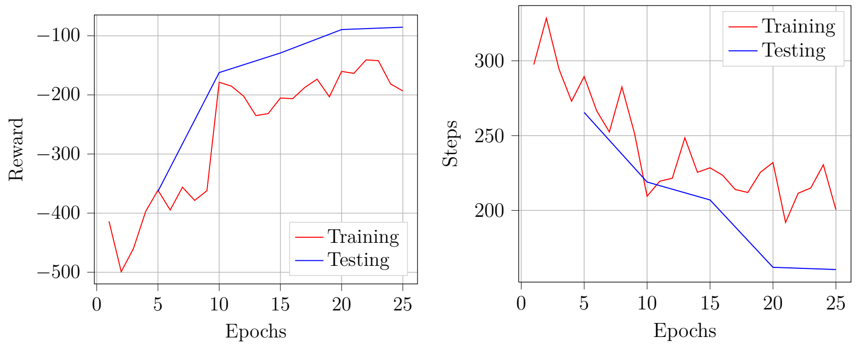
summarized for the last testing epoch for the current approach in Table 6.3.

6.2.2 Discussion

According to the presented results in this section, various conclusions can be drawn. First of all, for the learning curve case, obtained results in the real system (Figure 6.3) show a similar trend to obtained results with the simulator (Figures 5.1, 5.5 and 5.6) in both training and testing phases, presenting proximal values for the rewards in both implementations. In this context, with the presented results (Table 6.3), it is possible to validate the use of an RL-controller for the learning of assistance guiding forces in a real system for the *pick-and-place* task.

6.3 Supervised task detection (STD) approach

Placing a camera on the slave side (as the presented in Chapter 4), the *manual task selection* method can be enhanced to an autonomous and *supervised task detection* method. This can be done by capturing different images of the task intention and subsequently performing a labelling to build a dataset. Then, this dataset can be used to train a CNN using supervised learning being applied as task detector. In this way, the *task* element in the DDPG state vector $\boldsymbol{s}$ is provided by the CNN which performs the task detection automatically.



(a) Epoch rewards. (b) Steps per epoch.

Figure 6.3: Learning curves obtained for the MTS approach.

Table 6.3: Results obtained in the last testing epoch for the MTS approach.

	Forces	Reward function	Rewards	Steps
MTS	normalized	angle	−85.431	160.5

The supervised task detection method was implemented applying the *transfer learning* approach described in Chapter 2 using the pre-trained network VGG16 with input size 160x160x3 as base network, removing the last dense layers and placing instead a GlobalAveragePolling2D layer with 512 units and finally a dense layer with *sigmoid* activation with a single unit to obtain the task direction (Figure 6.4). The method selected to train the built network was the frozen features method. To practical effects, the trained network will be represented as presented in Figure 6.5 in the remain network illustrations. To build the dataset, different images were taken placing the object in different positions of the region of interest (ROI) for both directions (Figures 6.6 and 6.7). Given the task definition described in Section 3.1 the taken pictures were labelled for each task direction using a binary classification as presented in Table 6.4.

To have diversity in the dataset, a *data augmentation* image preprocessing function was used varying the shift, rotation, brightness and zoom of the images within a given range (Table 6.5). The trained model for the task detector has a binary output were each value is related to a task direction. Table 6.6 summarizes the parameters used to train the task detection model. Besides, the amount of images used for training and validation datasets are also shown.

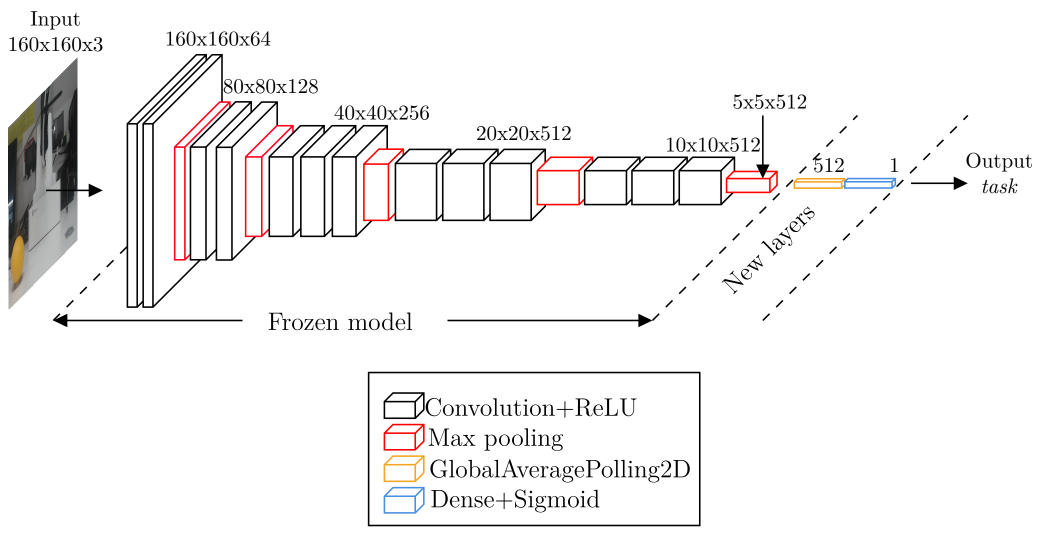


Figure 6.4: VGG16 modified model. Notice the substitution of the last FC original layers for a new ones to detect the task direction.

Table 6.4: Task direction labelling.

	Label
Right direction	0
Left direction	1

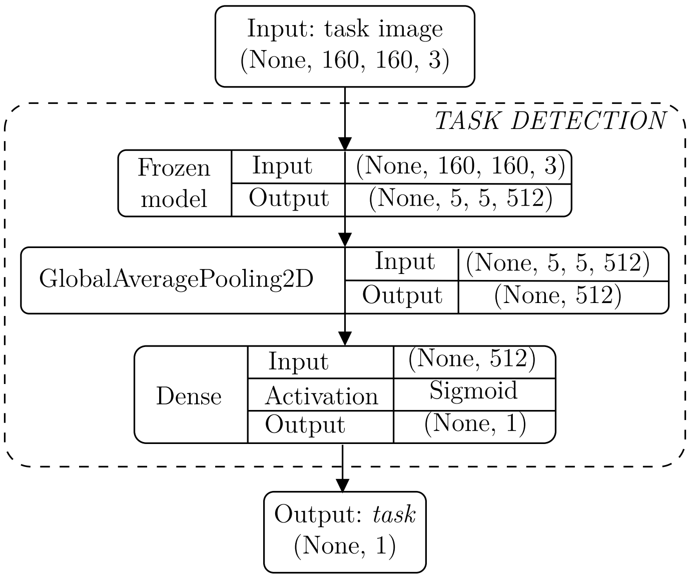


Figure 6.5: VGG16 task detection network implemented.

Table 6.5: Parameters used for the data augmentation function.

Parameter	**Value**
Height shift range	$10px$
Width shift range	$10px$
Rotation range	$\pm5°$
Brightness range	$[0.5,\ 2]$
Zoom range	0.1

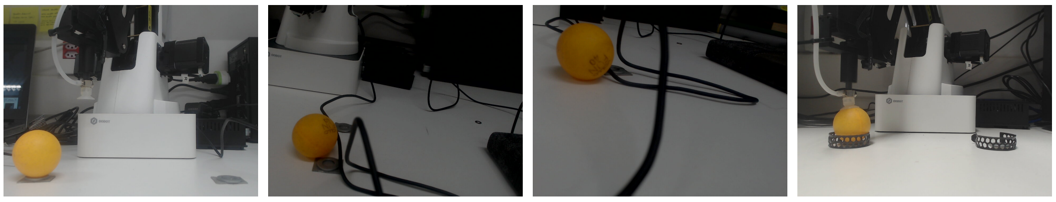


Figure 6.6: Sample images used for right direction.

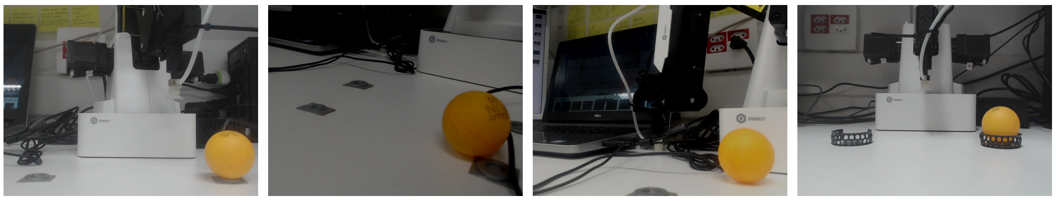
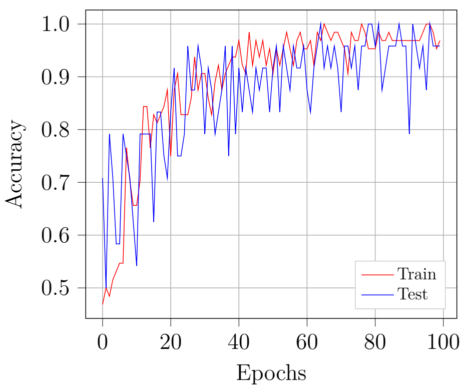


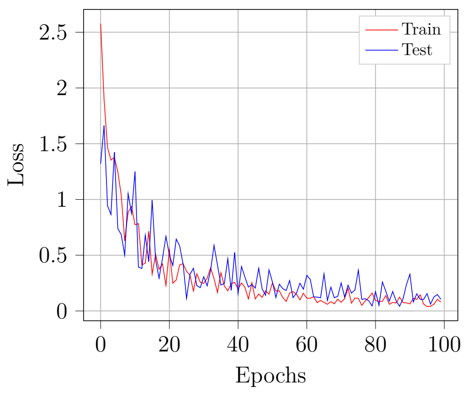
Figure 6.7: Sample images used for left direction.

Table 6.6: Parameters used for transfer learning training.

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Loss	Binary cross-entropy
Batch size	8
ANN Epochs	100
Training dataset size	136
Testing dataset size	28



(a) Accuracy.



(b) Loss.

Figure 6.8: Learning curves obtained for the VGG16 network training.

Table 6.7: Results obtained for the VGG16 network training

	Accuracy	**Loss**
Training	0.9688	0.08
Validation	0.9583	0.104

The algorithm presented in Algorithm 5 is also valid for the STD approach, due to there are no significant differences in comparison with the previous MTS approach. The unique difference lies in the addition of the *VGG16* function which uses the CNN trained network to provide the task direction value. Table 6.8 shows the parameters which were modified for this experiment w.r.t Table 6.2.

6.3.1 Results

Accuracy and loss curves for the CNN training and testing are presented in Figure 6.8 as well as the numerical values for both metrics achieved after 100 epochs (Table 6.7). In contrast with the previous approach, the obtained graphs correspond to a set of five experiments applying the STD approach,

Table 6.8: Parameters used for STD training.

Parameter	**Value**
Discount rate (γ)	0.97
Epochs	50

which are illustrated in Figure 6.9 [1]. In addition, a comparison between STD and MTS approaches is shown in Table 6.9, where the results for the STD approach were calculated as the mean of the last testing epoch for the rewards and the number of steps per epoch respectively.

6.3.2 Discussion

Again, the obtained results for the STD approach validated the use of the RL-controller (See Table 6.9). The results of the CNN network trained as task detector present an acceptable accuracy. This means that despite the limited dataset, the trained network was able to extract enough features to predict the correct label output.

In overall terms, the present set of experiments demonstrated that it is possible to perform the task detection autonomously using a pre-trained

[1]Results presented in this section were obtained with a modified DDPG network. However, because previous simulations presented statistically the same results for original and modified DDPG, these results were considered valid. DDPG modifications are presented in appendix section.

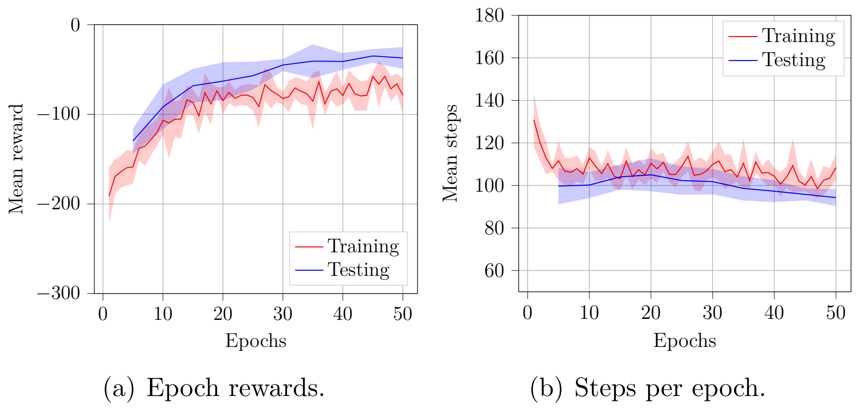


Figure 6.9: Learning curves obtained for STD approach.

Table 6.9: Results obtained in the last testing epoch for the MTS and STD approaches.

	Forces	**Reward function**	**Rewards**	**Steps**
MTS	normalized	angle	-85.431	160.5
STD	normalized	angle	-37.129	94.3

CNN network. An extra observable feature is the reduction of the number of steps required and the higher reward value obtained with this approach. However, to use this method, the user should have previous knowledge about which task they are going to perform in order to build the dataset. This is not useful if the intention is to use the system for new tasks because the training process becomes tedious work. This inconvenience is addressed with next implementation. To analyze the impact of the use of the VGG16 network in the training and testing epochs, an extra calculation was performed, the required time to perform the task was measured. Thereby, the number of steps per epoch was divided by this value being obtained the working frequency used for this approach, this impact will be discussed later. In addition, the inference time is around of $\sim$ 31ms, which can become a serious problem if it is necessary to make decisions with high velocity.

6.4 End-to-End learning (E2E) approach

As was demonstrated in the last section, images can be used to build a dataset and train a CNN network in order to perform the task detection. However, this approach can be difficult to use when the task is unknown and when there is not a fixed number of task intentions (unknown labels). In addition, building a dataset for this kind of situations can be tedious and time consuming. For this reason, an approach was implemented taking advantage of the DRL principle, where the agent can learn directly from pixels. Thereby, it is not necessary to train previously a CNN network to encode the task intention; the CNN layers can be included in the RL-network to train the agent. As such, the images of the task execution are captured by the camera and then stored in the DDPG replay buffer $\mathcal{R}$ every time-step (Figure 6.10). However, despite that images can be used to map the current state of the task, it is necessary to remember that the single output of the CNN network is used as part of the state vector $\boldsymbol{s}$ for the actor and critic respectively (See Figure 6.11). This is due to the fact that, as was mentioned in Chapter 3, the direction of the addressed task can be represented as a single value. Therefore, although we can denote the task images as part of the replay buffer $\mathcal{R}$, they are still part of the state vector.

The complete architecture for this implementation is shown in Figure 6.11. In addition, as was mentioned previously, the replay buffer is now composed of numerical and visual information as shown in Relation 6-1. To perform the training, the hyperparameters used in this implementation are presented in Table 6.10.

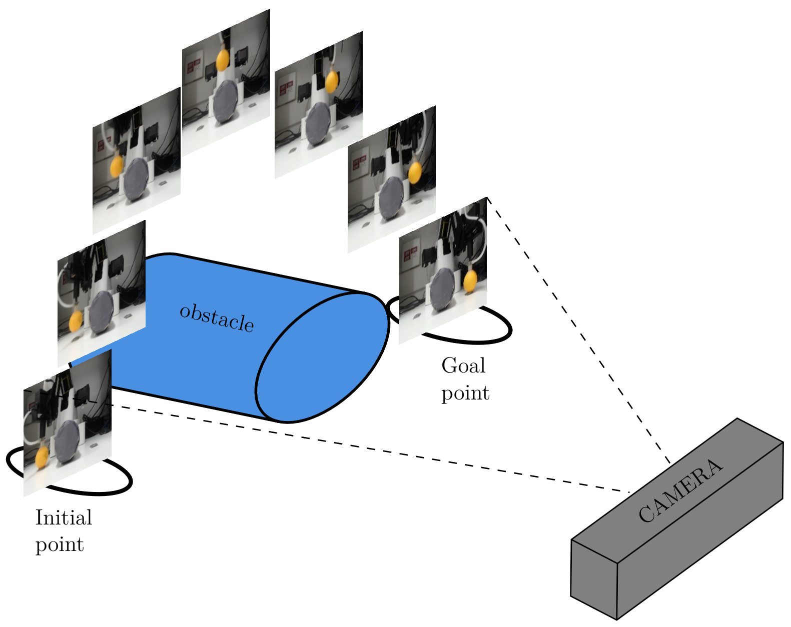


Figure 6.10: Basic functioning of end-to-end learning: In every time-step an image of the performed task is obtained with the camera and sent it to the HSC-controller.

$$\mathcal{R} \leftarrow \mathcal{R} \cup \{\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', \mathrm{A}, \textit{task image}\} \tag{6-1}$$

6.4.1 Results

The graphs for the reward and the number of steps per epoch are presented in Figure 6.12. The solid line represents the mean value and the shaded area represents the standard error obtained. Additionally, Figure 6.13 illustrates the behavior obtained[2].

6.4.2 Discussion

Results presented in Figure 6.12 validates the use of images directly to train the RL agent, presenting similar behavior with respect to previous implementations. In both sub-graphs of Figure 6.12, two points can be observed: the higher variation of the amount of steps and the stable convergence of the

[2]Similar to the previous section, the results presented for this approach were obtained with a modified DDPG network.

Table 6.10: Parameters used for E2E training.

Parameter	Value
Discount rate (γ)	0.97
Epochs	50

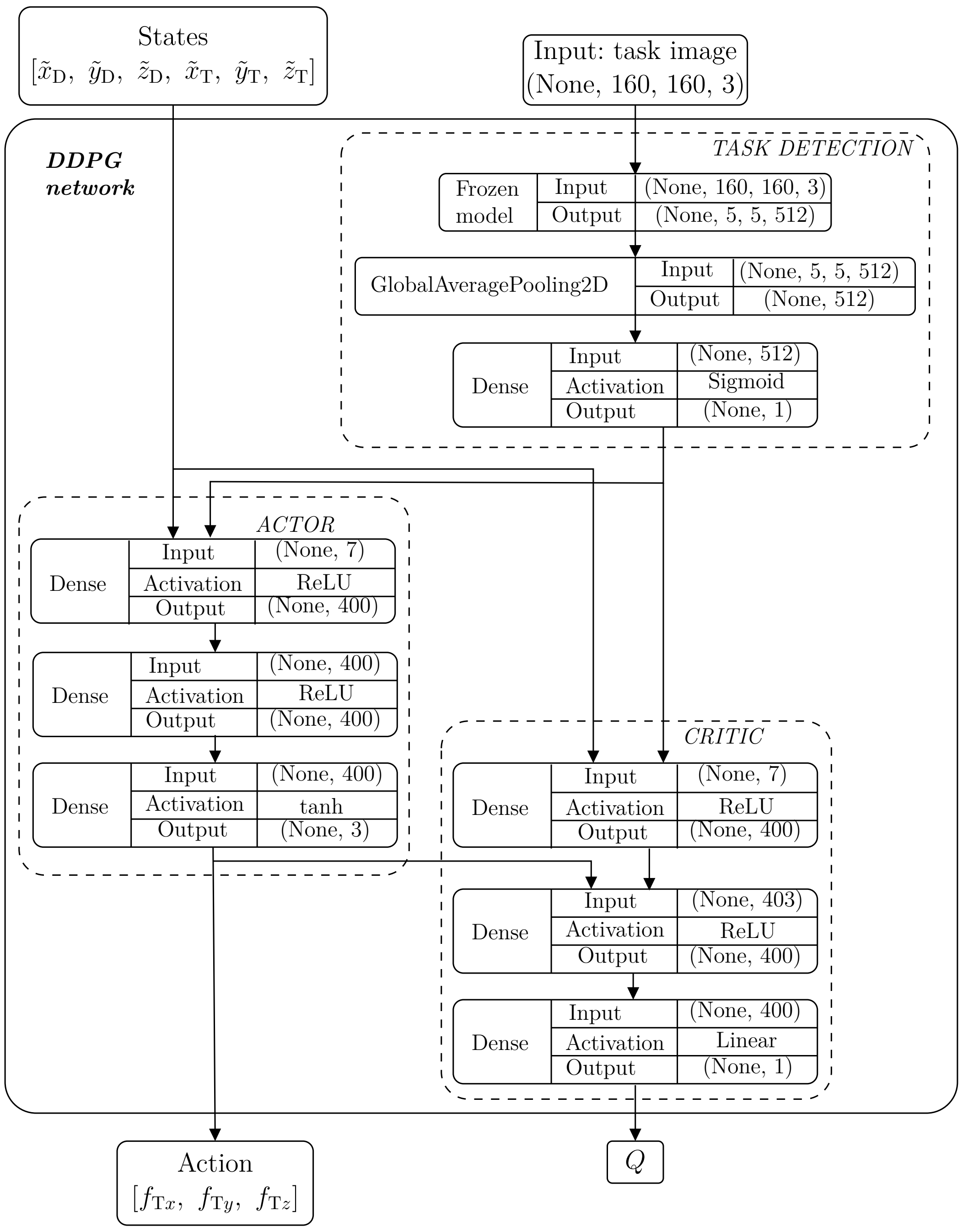
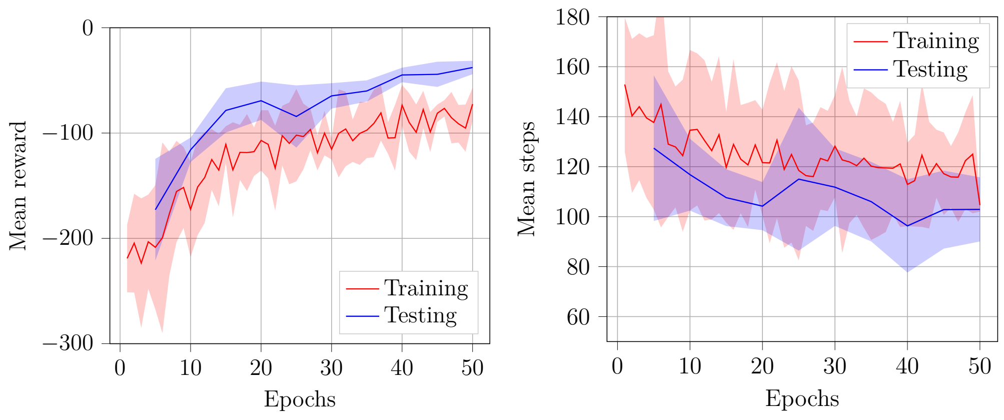

States
$[\tilde{x}_D,\ \tilde{y}_D,\ \tilde{z}_D,\ \tilde{x}_T,\ \tilde{y}_T,\ \tilde{z}_T]$
Input: task image
(None, 160, 160, 3)
DDPG network
TASK DETECTION
Frozen model | Input | (None, 160, 160, 3) | Output | (None, 5, 5, 512)
GlobalAveragePooling2D | Input | (None, 5, 5, 512) | Output | (None, 512)
Dense | Input | (None, 512) | Activation | Sigmoid | Output | (None, 1)
ACTOR
Dense | Input | (None, 7) | Activation | ReLU | Output | (None, 400)
Dense | Input | (None, 400) | Activation | ReLU | Output | (None, 400)
Dense | Input | (None, 400) | Activation | tanh | Output | (None, 3)
CRITIC
Dense | Input | (None, 7) | Activation | ReLU | Output | (None, 400)
Dense | Input | (None, 403) | Activation | ReLU | Output | (None, 400)
Dense | Input | (None, 400) | Activation | Linear | Output | (None, 1)
Action
$[f_{Tx},\ f_{Ty},\ f_{Tz}]$
Q

Figure 6.11: E2E learning architecture implemented.

Table 6.11: Results obtained in the last testing epoch for the MTS and STD approaches.

	Forces	**Reward function**	**Rewards**	**Steps**
MTS	normalized	angle	−85.431	160.5
STD	normalized	angle	−37.129	94.3
E2E	normalized	angle	−37.758	102.9



(a) Epoch rewards. (b) Steps per epoch.

Figure 6.12: Curves obtained for E2E approach.

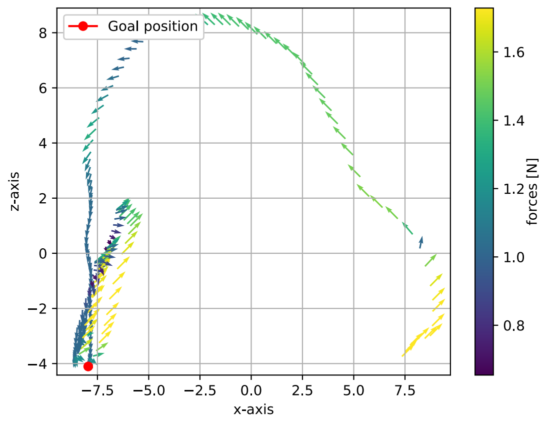


Figure 6.13: Behavior obtained using the E2E approach. Notice the larger oscillations in the end of the episode.

learning despite the noisier performance in the training phase. As such, it can be seen that training both networks at the same time (DDPG and VGG16 output) can achieve good performance. However, despite the acceptable results, this implementation presents one inconvenience: the higher the network complexity, the more training time required; that is, according the increased amount of parameters, the time required to train all the network is increased as well (around $\sim$ 1 min. per episode). In this way, the training process of this approach can become time consuming. Moreover, similar to the STD approach, the inference time used by the complete network takes around 30ms. On the other hand, an atypical behavior was observed in its execution, as presents the Figure 6.13, oscillations appear when the goal position is reached.

6.5 DDPG: Single-shot task detection (SSTD) approach

In order to decrease the training time, a set of modifications was made. First of all, the use of RGB images in the replay buffer ($\mathcal{R}$) can lead to a memory saturation of the computer. In addition, the training time used increases due to the amount of parameters of the complete network. To deal with this situation, the next modification was made: the CNN frozen model until the *GlobalAveragePooling2D* layer was separated maintaining only the last dense layer into the DDPG network . In this way, the separated part of the CNN network is used to provide the resulting features of the convolutions from the passed frames. In addition, to decrease the inference time, the first frame is passed through the CNN network and the resulting output values are maintained along the entire episode. Using this modification the number of parameters to be trained and the inference time were decreased which leads to less time to train the HSC controller.

Using the current image of the task, as in the E2E approach, it is not possible to determine the task direction intention when low velocities are applied, leading to the oscillations seen in Figure 6.13. For this reason, the SSTD approach uses the information of the first image to decode the task direction.

Note that this approach will only be effective if the addressed task can be determined with visual information of the initial position, for example the pick-and-place task in this case. However, this information is not necessarily the case, for example peg-in-hole task. If the intention is to address any task, it is recommendable to use the visual information of both the first and the current image, thereby, any task intention could be completely decoded.

Figure 6.15 presents the mounted network with the proposed modifications. It can be noted that the VGG16 network is separated and is not trained within the DDPG network (Figure 6.14). In this way, only the output of the dense layer is concatenated with the position vectors. To validate this approach, only one experiment was performed since DDPG convergence was demonstrated in previous implementations. Algorithm 6 was used in this implementation and the complete set of parameters is summarized in Table 6.12.

6.5.1 Results

Results for the use of this approach are shown in Figure 6.16 showing the reward per epoch and the number of steps per epoch. In contrast with previous results, for this approach was performed only one experiment. Besides

Algorithm 6 DDPG used with SSTD

```
function DDPG($\gamma, \tau, M, N$)
    $\boldsymbol{w} \leftarrow 0,\ \boldsymbol{\theta} \leftarrow 0,\ \boldsymbol{w}^{\dagger} \leftarrow 0,\ \boldsymbol{\theta}^{\dagger} \leftarrow 0,\ n \leftarrow 0$
    for episode=1,M do
        $\boldsymbol{s} \leftarrow$ Start for *task*
        while $\boldsymbol{s}$ is not $\boldsymbol{s}_n$ do
            if steps=1 then
                *convolutions* ←VGG16(*task image*)
            Select action $\boldsymbol{a}_t$ from $\hat{\mu}(\boldsymbol{s}_t;\boldsymbol{\theta}) + \mathcal{X}_t$
            Execute action $\boldsymbol{a}_t$, observe reward $r$ and observe new state $\boldsymbol{s}_t^{'}$
            Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_t', 0,$ *convolutions*$)$ in $\mathcal{R}$
            $\boldsymbol{s}_t \leftarrow \boldsymbol{s}_t^{'}$
            $n \leftarrow n + 1$
            $t \leftarrow t + 1$
        DEFINE BEHAVIOR()
        Store transition $(\boldsymbol{s}_n, \boldsymbol{a}_n, r_n, \boldsymbol{s}_n', 1,$ *convolutions*$)$ in $\mathcal{R}$
        for steps=1,n do
            TRAIN-MINIBATCH($\mathcal{B}, \gamma, \tau, \boldsymbol{\theta}, \boldsymbol{\theta}^{\dagger}, \boldsymbol{w}, \boldsymbol{w}^{\dagger}, N$)
```

Table 6.12: Parameters used for SSTD implementation.

Parameter	Value
Batch size (N)	64
Discount rate (γ)	0.99
Update target parameter (τ)	0.01
Assistance forces	Normalized
Reward function	Angle-based
Episodes	25
AR-p order (p)	3
AR-p parameter (α)	0.9

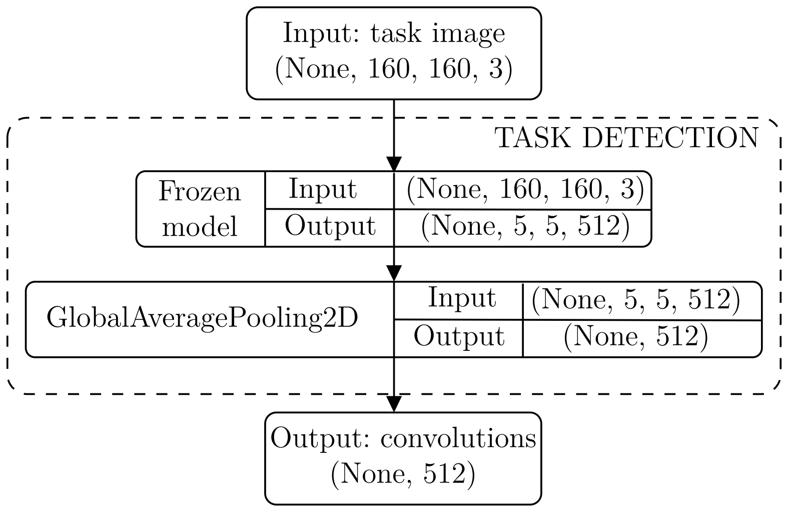


Figure 6.14: Task detector CNN network architecture used.

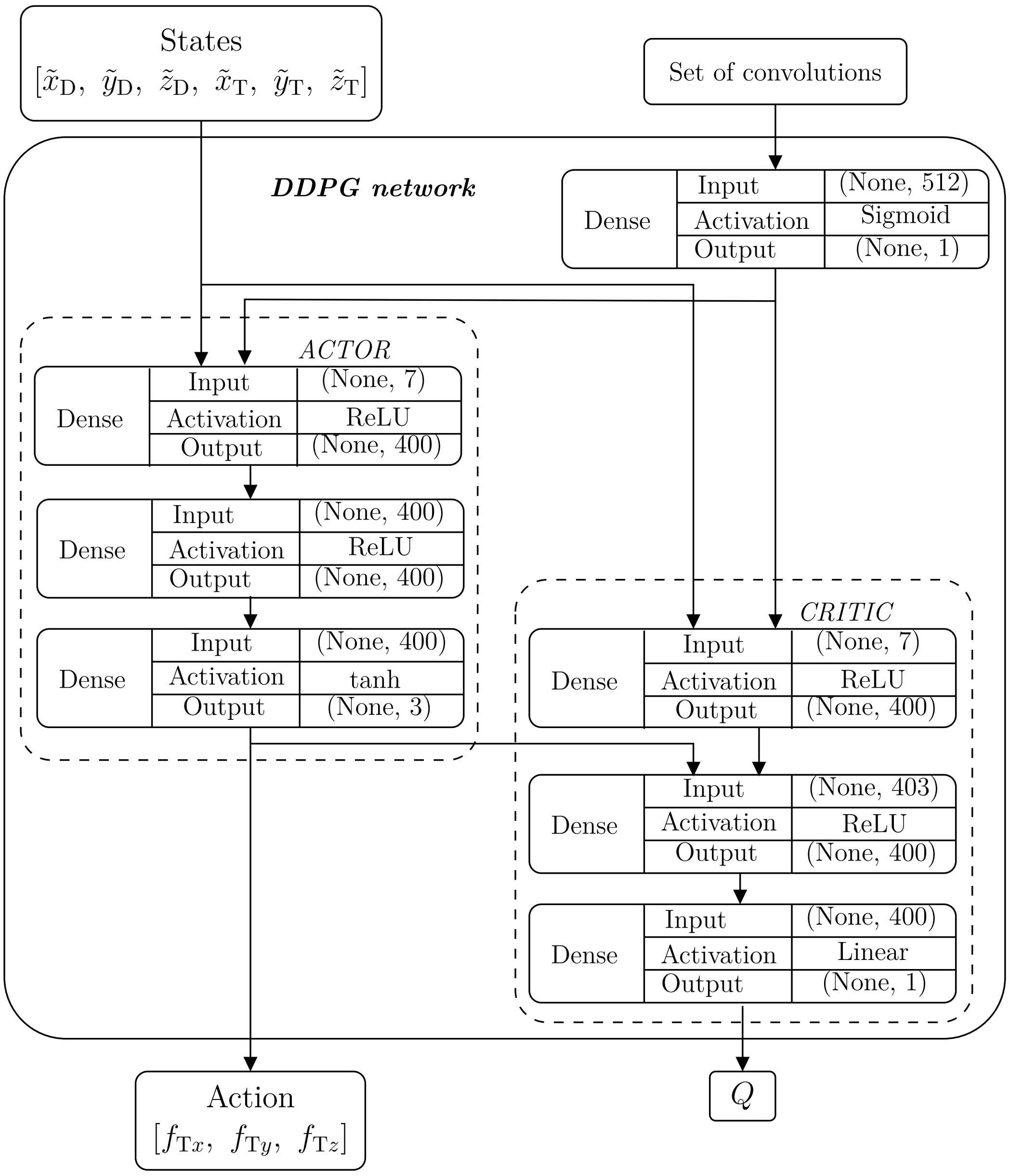
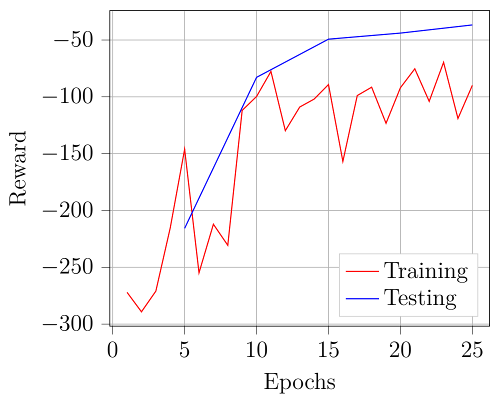

States
$[\tilde{x}_\mathrm{D},\ \tilde{y}_\mathrm{D},\ \tilde{z}_\mathrm{D},\ \tilde{x}_\mathrm{T},\ \tilde{y}_\mathrm{T},\ \tilde{z}_\mathrm{T}]$
Set of convolutions
DDPG network
Dense | Input (None, 512) | Activation Sigmoid | Output (None, 1)
ACTOR
Dense | Input (None, 7) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation tanh | Output (None, 3)
CRITIC
Dense | Input (None, 7) | Activation ReLU | Output (None, 400)
Dense | Input (None, 403) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation Linear | Output (None, 1)
Action
$[f_{\mathrm{T}x},\ f_{\mathrm{T}y},\ f_{\mathrm{T}z}]$
Q

Figure 6.15: SSTD architecture implemented.

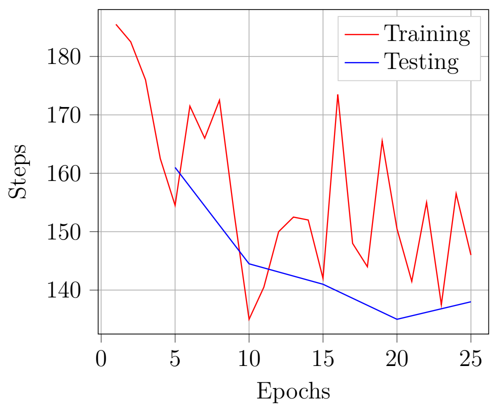
the stable convergence of the DDPG agent, the amount of steps required to perform the task decreases along the RL-controller reaches the convergence.

6.5.2 Discussion

The main features of this modification are reduced amount of parameters(which leads to less processing time) and reduced amount of memory required for the replay buffer. Thereby, similar behavior can be achieved without decrease performance in the assistance guiding forces. The main advantage remains in the fact that images are only used to encode the user intention in the beginning of the episode. Table 6.13 summarizes the numeric results obtained in the experiments performed in real system where interesting results can be observed. First of all, an observable result is the amount of steps for the STD and E2E approaches, which are lower than the remain implementations. However, despite the promising results, it is not possible to say that both approaches present better performance w.r.t the other ones. As was explained before, this is due to the inference time that is required by the CNN network ($\sim$ 30ms for both cases), which influences their working frequency ($\sim$ 28Hz) in comparison to remaining implementations which working frequency is unaltered ($\sim$ 50 Hz). For this reason, it is possible to say that although both implementations work, their performance is not the best. This can be confirmed with normalized values presented in the bottom side of Table 6.13 where the results for STD and E2E approaches were normalized to a frequency rate of 50Hz to establish a fair comparison with the previous results, showing similar performance in terms of steps with MTS results. Unlike previous approaches, the SSTD approach presents better results in terms of rewards and steps. In addition, due to the VGG16 network is only used twice every episode (both



(a) Epoch rewards.



(b) Steps per epoch.

Figure 6.16: Learning curves obtained for SSTD with normalized forces.

directions), the amount of steps is not as affected as in STD and E2E approaches. In the other hand, the training time ($\sim$ 4s) per episode is much less than the E2E approach ($\sim$ 56s).

All approaches presented until now, present one problem that could be observed by testing the learned policy: the appearance of assistive forces that produce oscillations in the user movements when the goal position is reached (Figure 6.13). This behavior is obtained due to the use of normalized forces described in the last part of Section 4.1.2. Due to the experience replay buffer being built with normalized forces, there is no way for the agent to learn how to modulate the assistance forces along the task execution. In this way, as result of the combination of the explained situations, reaching the goal position by the user becomes a difficult part of the task. To deal with this situation, the use of clipped variable forces was applied as presented in the next section.

6.6 SSTD with variable forces

Despite the acceptable performance achieved in previous experiments, all the described implementations presented a set of oscillations that appeared in the last part of the trajectories in the task execution. These oscillations appeared because the addition of different situations: first, the force magnitude is the same along all the steps while the user is performing the task. Second, exists a position delay between both devices when the task is being executed, resulted in large oscillations at the end of the trajectory. Thereby, the learned policy did not allow the user to place the object accurately in the goal without significant effort. To avoid this kind of behavior, a new modification was implemented: the use of variable forces along the training, storing directly the forces in the replay buffer without performing normalization. For this approach, only one experiment was performed and the algorithm used was not modified. In addition, the set of parameters is also maintained.

Table 6.13: Comparison of the results obtained in the last testing epoch for the different DDPG implemented. The first values for STD and E2E are valid for 28Hz, the normalized values correspond for 50Hz

	Forces	**Reward function**	**Rewards**	**Steps**
MTS	normalized	angle	−85.431	160.5
STD	normalized	angle	−37.129	94.3
E2E	normalized	angle	−37.758	102.9
Normalized STD	normalized	angle	−66.302	168.4
Normalized E2E	normalized	angle	−67.425	183.75

6.6.1 Results

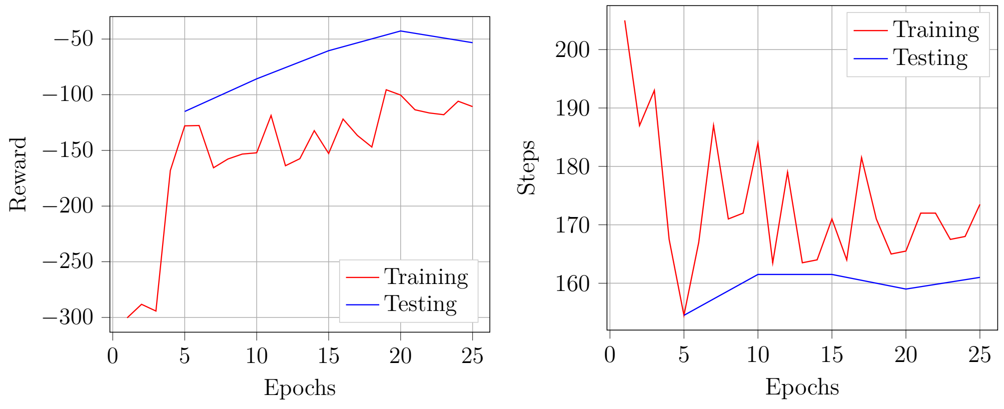
Results for the use of this approach are shown in Figure 6.17 showing the learning curve and the amount of the steps. Similar to previous implementations, besides the stable convergence of the DDPG agent, the amount of steps required to perform the task decreases as the RL-controller reaches convergence.

6.6.2 Discussion

Despite the good performance achieved as shown in Figure 6.17, the trajectory illustrated in Figure 6.18 still presents oscillations. In this figure it can be observed that despite the oscillations force is being attenuated using variable forces they are still present. This is because, despite the use of variable forces and the visual information about the initial position of the task, the reward function is independent of force and velocity magnitudes. Therefore, it is not able to transfer enough knowledge to the agent about the correct behavior when the user is reaching the goal position. In order to provide the correct assistance forces a composite reward function was proposed as described in Section 3.2.3.2.

6.7 SSTD with variable forces and fuzzy-based reward function

As was explained in the last section a composite reward function, named *fuzzy-based* reward function, was implemented to deal with the oscillation problem presented in the previous implementations. This function was designed based on decisions along the different stages in the task trajectory (Sec-



(a) Epoch rewards. (b) Steps per epoch.

Figure 6.17: Learning curves obtained for SSTD with variable forces.

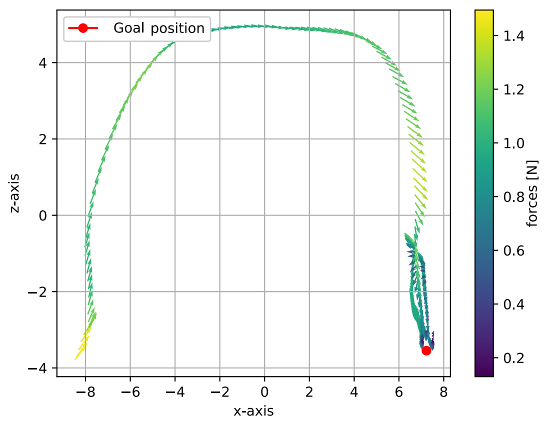


Figure 6.18: Trajectory obtained using the SSTD with variable forces. Notice that in the final of the execution the oscillations are still present.

tion 3.2.3.2). Thereby, the function is able to influence the agent learning when small velocities are required, providing the user accurate control when reaching the goal position. The final implementation of this approach includes the main changes presented in the previous implementations: variable forces in the SSTD approach. Table 6.14 summarizes the parameters used in this experiment .

6.7.1 Results

Results for the use of this approach are shown in Figure 6.16 showing the learning curve and the amount of the steps for a set of three experiments. The solid line represents the mean value and the shaded area represents the standard error obtained. Moreover, graphs showing the trajectory described on a successful episode and the forces received by the operator using the learned policy are presented in Figures 6.20 and 6.21 respectively. Besides the graphs, Table 6.15 summarizes the results obtained for all the implemented approaches

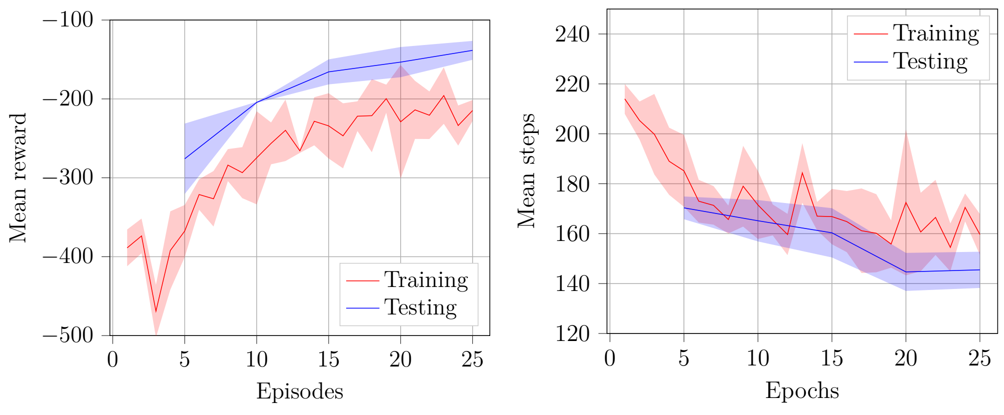
Table 6.14: Parameters used for SSTD with variable forces and fuzzy-based reward function.

Parameter	Value
Batch size (N)	64
Discount rate (γ)	0.99
Update target parameter (τ)	0.01
Assistance forces	Variable
Reward function	Fuzzy-based
Episodes	25
AR-p order (p)	3
AR-p parameter (α)	0.9

performed in this research.

6.7.2 Discussion

As is observed in the learning curves, the set of experiments performed for the SSTD second variant present similar results with the previous SSTD implementations in terms of steps per epoch. The large difference in terms of rewards presented in Table 6.15 for the SSTD second variant is due to the use of the fuzzy-based reward function with respect to the remaining variants that use the angle-based reward function which evaluates the taken action with a different argument. Another observable difference is presented in Figure 6.22 where it can be seen that the forces decrease when goal position is being reached by the user. Thereby, residual oscillations in the final part of the execution were eliminated using this modification. As expected, similar to previous experiments, the DDPG agent presents a stable convergence in the



(a) Epoch rewards. (b) Steps per epoch.

Figure 6.19: Learning curves obtained for SSTD with fuzzy-based reward function.

Table 6.15: Comparison of the results obtained in the last testing epoch for the different SSTD implemented.

	Forces	**Reward function**	**Rewards**	**Steps**
MTS	normalized	angle	−85.431	160.5
STD	normalized	angle	−37.129	94.3
E2E	normalized	angle	−37.758	102.9
Normalized STD	normalized	angle	−66.302	168.4
Normalized E2E	normalized	angle	−67.425	183.75
SSTD base	normalized	angle	−36.729	138
SSTD first variant	variable	angle	−53.268	161
SSTD second variant	variable	fuzzy	−138.439	145.5

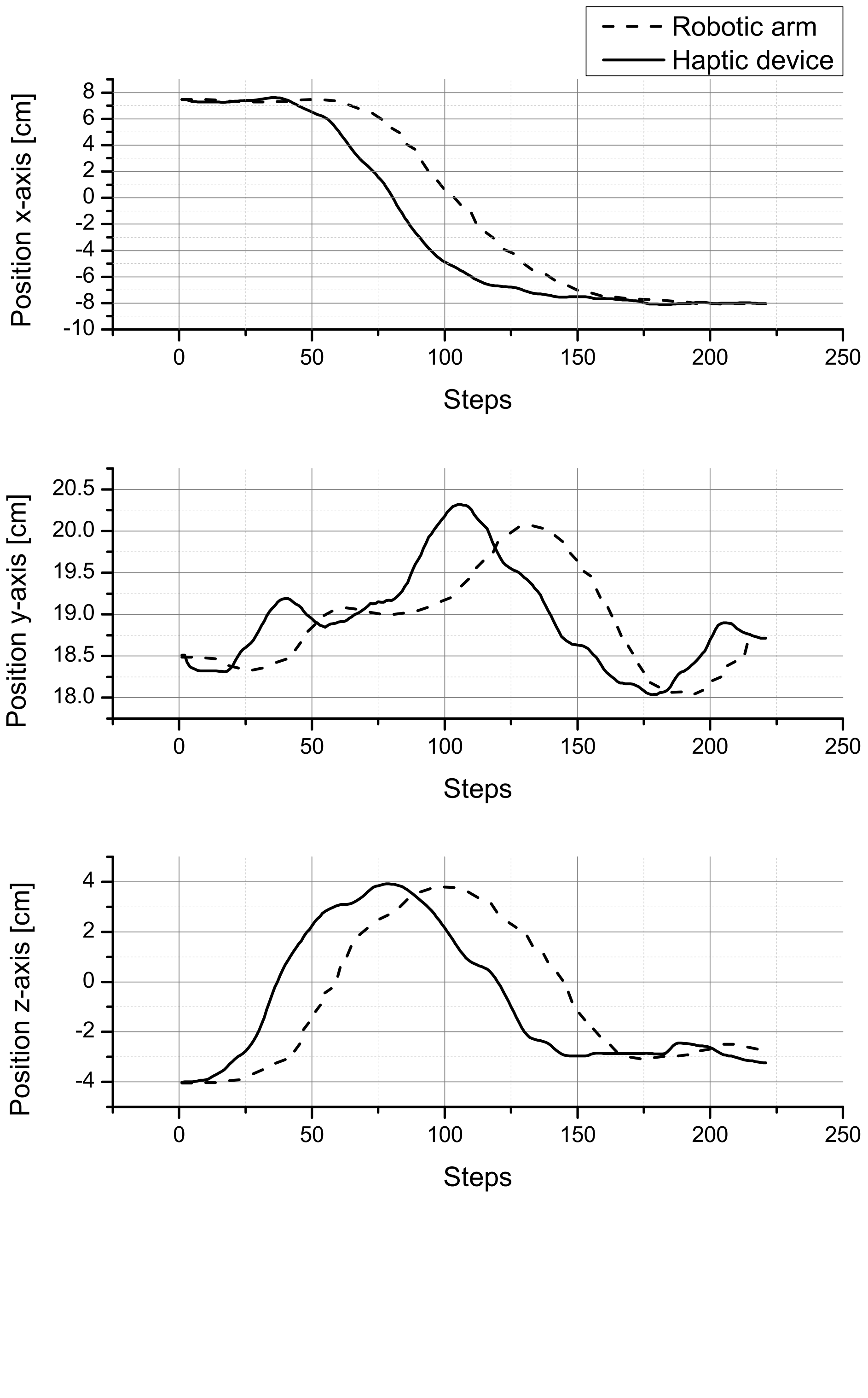


Figure 6.20: Trajectory described by both devices using the SSTD approach with variable forces and the fuzzy-based reward function.

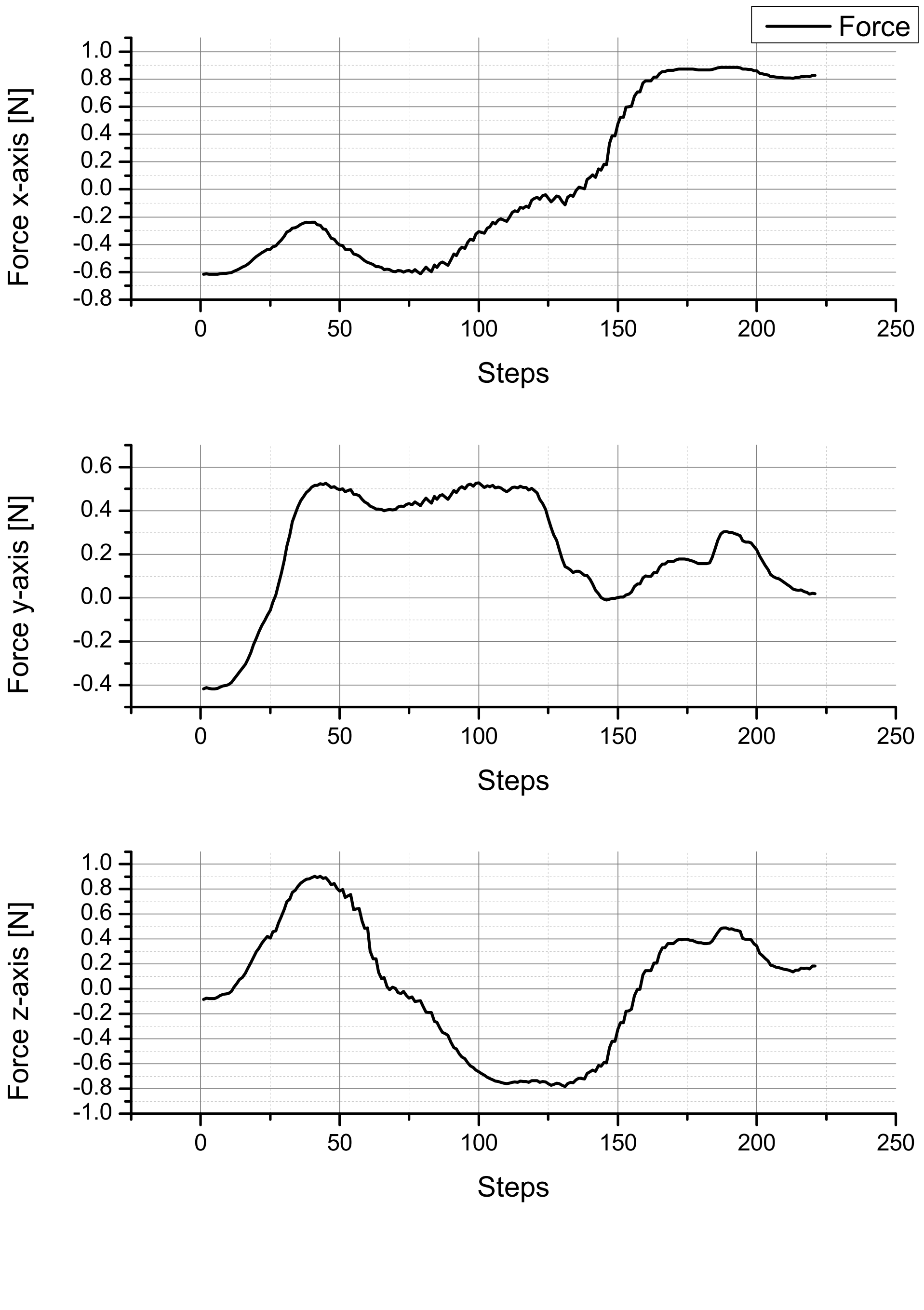


Figure 6.21: Forces received by the operator through the haptic device using the SSTD approach with variable forces and the fuzzy-based reward function.

final part of the training in terms of rewards and steps, which leads the next conclusion: the SSTD approach with variable forces and fuzzy-based reward function can be successfully applied in teleoperated systems providing helping to users in the task completion. Finally, it is worth mentioning that all the experiments were performed by the student in charge of this research, for future references named as subject 0.

6.8 Validation with different subject

To investigate the functioning of the developed HSC controller, two different subjects were asked to perform a series of tests. These tests consisted in four stages: familiarity with the system, pre-training tests, policy training and post-training tests. In the first stage, the basic functioning of the implemented system was explained while the testing subject was manipulating the devices using the teleoperation controller. Next, a set of episodes were tested with and without providing assistance with a trained policy by subject zero (named as *policy-zero*) while the testing subject was trying to perform the task. Then, the testing subject was asked to train a new policy with the RL-controller performing the designed task. Finally, a new set of tests with the same structure as the pre-training stage was performed with the trained policy.

6.8.1 Results

Results comparing the number of steps per epochs for both subjects before and after performing the training are presented in Table 6.16. In addition, results using the same criterion comparing the velocity are presented

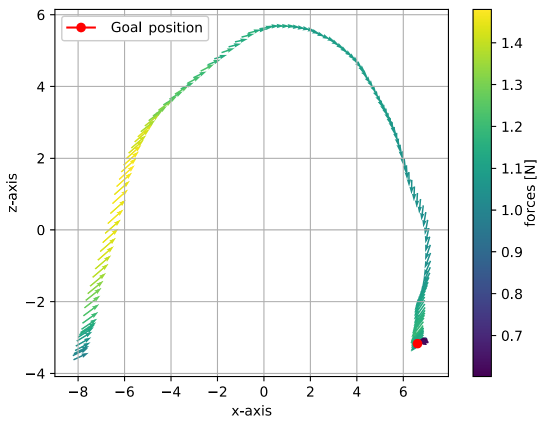


Figure 6.22: Trajectory obtained with last SSTD variant: Notice the complete elimination of the oscillations when the goal position is reached.

in Table 6.17. Finally, force results receiving assistance before and after the training for both directions are presented in Tables 6.18.

6.8.2 Discussion

Various conclusions can be drawn with the results presented in this section. As was mentioned in the introduction part: *every subject is unique*, thus, each one performs the task in a different way, this conclusion is reflected with the trajectories illustrated in Appendix D and E for subjects 1 and 2.

First of all, despite the RL-agent learns an assistance policy, this policy is unique for each subject and not universal for all users. This conclusion can be explained using Figure 6.23 where is presented the basic scheme of the implemented system. In a common RL-implementation, the observations and rewards received by the RL agent are result of his direct interaction with the environment through the taken actions. However, in RL-implementations with a human-in-the-loop (See Figure 6.23), the interaction between the RL agent and the environment is through the user. As result, the RL agent optimizes the combined user-environment system. Therefore, if the user behaves differently, a different policy will be optimal. Thereby the policy becomes particular for every user. Additional data presented in Tables 6.17 and 6.18 reinforce this conclusion, where is shown the particular behavior for the different subjects.

In this context, another observable result that confirms the unique be-

Table 6.16: Number of steps per epoch for test subjects. The bold values represents the less amount of steps per epoch.

Number of steps per epoch			
	Mean	**Std. dev**	**Assistance**
Subject 0			
After training	230.25	63.86	-
	200.75	**19.397**	✓
Subject 1			
Before training	496.25	107.8	-
	487.5	**54.51**	✓[3]
After training	337.25	79.65	-
	315.5	**23.01**	✓
Subject 2			
Before training	300.25	79.65	-
	275.0	**55.76**	✓[3]
After training	208.5	39.04	-
	166.25	**44.31**	✓

[3] Using *policy-zero*

Table 6.17: Velocity results for test subjects. The bold values represents the highest value for all cases.

	Velocity [cm/s]			
	Max	**Mean**	**Direction**	**Assistance**
Subject 0				
After training	14.073	6.229	**Left**	-
	22.38	**8.988**		✓
	14.274	5.949	**Right**	-
	17.688	**6.73**		✓
Subject 1				
Before training	8.02	3.166	**Left**	-
	10.987	**3.779**		✓[4]
	9.346	**3.376**	**Right**	-
	8.973	2.836		✓[4]
After training	10.751	4.861	**Left**	-
	11.328	**4.848**		✓
	9.558	4.076	**Right**	-
	13.175	**4.578**		✓
Subject 2				
Before training	**52.579**	**10.372**	**Left**	-
	44.915	9.542		✓[4]
	27.692	6.909	**Right**	-
	45.525	**10.567**		✓[4]
After training	14.849	6.254	**Left**	-
	23.286	**7.784**		✓
	23.75	6.726	**Right**	-
	32.196	**10.884**		✓

[4] Using *policy-zero*

Table 6.18: Force results for test subjects. The bold values represents the highest value for all cases.

	Forces [N]		
	Max	**Mean**	**Direction**
Subject 0			
After training	1.549	1.134	**Left**
	1.086	**0.825**	**Right**
Subject 1			
Before training	**1.374**	**0.842**	**Left**
	0.992	0.839	**Right**
After training	1.12	0.599	**Left**
	1.248	**0.46**	**Right**
Subject 2			
Before training	**1.44**	**0.902**	**Left**
	1.418	0.848	**Right**
After training	1.52	1.262	**Left**
	1.728	**0.864**	**Right**

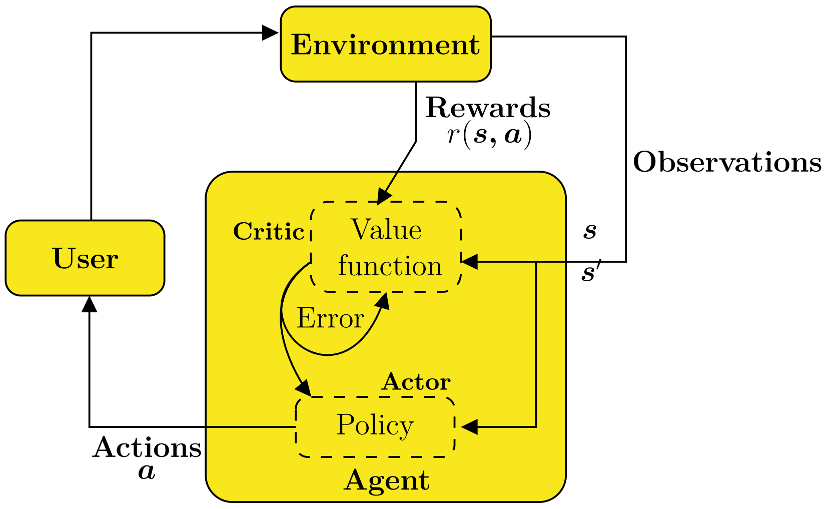


Figure 6.23: Reinforcement Learning scheme in the agent-environment interaction in a human-in-the loop approach.

havior for each user is when velocity and steps are compared, for example, clearly, Subject 2 presents the highest velocity, which results in less number of steps for all cases: before and after perform training and with and without receive assistance (Tables 6.16 and 6.17). On the other hand, opposite performance is achieved by Subject 1, who presented the lowest velocity and more amount of steps. Finally, we can see that Subject 0 presents an intermediate behavior compared with the other subjects.

Next, the amount of assistance depends on the policy learned by the RL agent for each user. As was presented before, the policy is highly influenced by the user execution and the reward function. Therefore, for quicker subjects the assistance forces will be greater than the other, as well as in the opposite case (See Table 6.18). Despite the reward function captures most of the intention of the different subjects, threshold x was defined according the Subject 0 behavior. However, this value could be not optimal for the other subjects. Then, the fuzzy-based reward function could be improved by changing the x value according the subject preference. An additional test was made lowering the value of the threshold to $x = 3$ for the Subject 1, the results of this test are presented in Appendix F.

Another interesting result observed in Table 6.16 is that the number of steps required to complete the task for each subject decreases in all cases comparing before and after performing the training, which can lead to say that the subjects are able to learn the task independent of the received assistance. However, the difference between without assistance and with assistance also increases. This may be attributed to the user being more in tune with his own trained policy than with the policy trained by subject 0.

Finally, from Table 6.16, it is possible to observe that when the test subjects use the implemented system for first time, they present some difficulties to adapt to the functioning. This is reflected in the amount of steps per epoch required before perform the training, which decreases according the subjects become familiar with the implemented system after the trained was performed. This values demonstrate the capability for humans to adapt to policies that were not training for them, even for sub-optimal polices. However, it is observed that the performance achieved for each subject is better for all the cases after perform a training for the *pick-and-place* task.

7 CONCLUSIONS AND FUTURE REMARKS

In this work, a novel HSC controller was developed to be used in a teleoperated system composed by a robotic arm and a haptic device. The proposed controller is composed by an RL section and a direct teleoperation section. Taking advantage of the policy gradient methods that present stable functioning on continuous control systems, the RL-controller was implemented using as core the DDPG algorithm. On the other side, a proportional controller was used as the base of the teleoperation controller.

The resulting HSC controller was able to learn custom guiding forces for every subject who tested the system (Section 6.8). Thereby, the learned policy becomes a personal policy which contains enough guidelines to assist the particular behavior of the user it was trained for. Besides, task direction was learned dynamically using the VGG16 network without defining which is the task to be performed. Therefore, the RL-controller learns on-the-fly the guiding forces with a limited amount of training episodes. Finally, the learned policy presented acceptable behavior in terms of convergence and performance, being able to assist successfully the subjects to execute the task faster and in a personal way.

During the HSC controller development, diverse variants were tested before reaching the final implementation. Those approaches included the use of a CNN (STD and E2E approaches) to perform the task detection. However, it was demonstrated that the processing time increases greatly when there is a large number of parameters to use. Because of this, we chose use the VGG16 network in the beginning of the episode (SSTD approach), which decreased enormously the training and testing time.

Despite of the promising results obtained with the use of the SSTD approach, irregular behavior appeared using the normalized forces approach. The policy became sub-optimal in terms of controllability in the final part of the task execution. One of the solutions proposed was the use of a variable forces approach. However, despite the improvements obtained, this problem was not completely solved.

For this reason, it was necessary to design a new reward function that helped the learning of the variation of the force magnitudes and encouraged high assistive forces. These specifications were condensed in the fuzzy-based reward function proposed, which showed the complete elimination of the atypical oscillations described previously. Extra features were the learning of custom policies and the performance improvements in terms of the amount of steps that users require to complete the task.

Finally, it is worth noting that, despite good results obtained with normalized forces and angle-based reward function for SSTD base presented in Table 6.15, these results are only valid with an experienced user. Novel users present difficulties when trying to execute the task, especially to place the object in the goal position. This was the main reason to deprecate the policy and consider it as sub-optimal.

Future implementations

Despite the good results obtained with the implemented simulator, it was not possible to completely mimic the human behavior. In this context, modelling techniques as cybernetics modelling or system identification methods could be used to improve the simulator functioning for future training.

In addition, different RL algorithms (for instance: PPO [66], ACER [67], TD3 [68], SAC [69], etc) could be tested in simulations in order to observe resulting behavior in the implemented system and perform a comparison between them in terms of performance and required number of training episodes.

In the real system, most of the basic parameters were set empirically. For instance: proportional gains in algorithms. It is necessary to perform new analysis in these sections in order to improve performance in the implemented system.

In the design of the system it was intended that the slave side follows all the orders received by the master side. So that, any movement in the slave side is the result of tracking the interaction between the user and the RL-controller. Thereby, the proportional control used in the teleoperation controller could be improved using a proportional-integral controller or maybe an advanced control technique, as adaptive or robust control for instance, to avoid or handle the singularities appeared in the robotic manipulator when the task is being performed.

In this context, chosen value for x term in fuzzy-based reward function worked for all subjects. However, an additional test was performed for subject one decreasing this threshold. This test showed better performance for this subject with threshold $x = 3$ instead of $x = 6$[1]. Therefore, it is necessary to design a method to find the value that is more suitable for each user.

Finally, the use of the image of the initial state allow the users to perform tasks that depends on this kind of information, the pick-and-place task for example. However, to perform any kind of task, for instance the peg-in-hole task, it is desirable to have information about the initial and the current state. Therefore, it could be advisable this modification as part of the future implementations as well as more tests per approach.

[1]Results presented in Appendix F

Bibliography

[1] FLEMISCH, F.; ABBINK, D.; ITOH, M.; PACAUX-LEMOINE, M.-P. ; WESSEL, G.. **Shared control is the sharp end of cooperation: Towards a common framework of joint action, shared control and human machine cooperation**. IFAC-PapersOnLine, 49(19):72 – 77, 2016. 13th IFAC Symposium on Analysis, Design, and Evaluation ofHuman-Machine Systems HMS 2016.

[2] CRANDALL, J. W.; GOODRICH, M. A.. **Characterizing efficiency of human robot interaction: a case study of shared-control tele-operation**. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, volumen 2, p. 1290–1295 vol.2, Sep. 2002.

[3] ABI-FARRAJ, F.; OSA, T.; PETERS, N. P. J.; NEUMANN, G. ; GIORDANO, P. R.. **A learning-based shared control architecture for interactive task execution**. In: 2017 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), p. 329–335, May 2017.

[4] PETERNEL, L.; OZTOP, E. ; BABIČ, J.. **A shared control method for online human-in-the-loop robot learning based on locally weighted regression**. In: 2016 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), p. 3900–3906, Oct 2016.

[5] BOESSENKOOL, H.; ABBINK, D. A.; HEEMSKERK, C. J. M. ; VAN DER HELM, F. C. T.. **Haptic shared control improves tele-operated task performance towards performance in direct control**. In: 2011 IEEE WORLD HAPTICS CONFERENCE, p. 433–438, June 2011.

[6] BOESSENKOOL, H.; ABBINK, D. A.; HEEMSKERK, C. J. M.; VAN DER HELM, F. C. T. ; WILDENBEEST, J. G. W.. **A task-specific analysis of the benefit of haptic shared control during telemanipulation**. IEEE Transactions on Haptics, 6(1):2–12, First 2013.

[7] V. OOSTERHOUT, J.; WILDENBEEST, J. G. W.; BOESSENKOOL, H.; HEEMSKERK, C. J. M.; D. BAAR, M. R.; V. D. HELM, F. C. T. ; ABBINK,

D. A.. **Haptic shared control in tele-manipulation: Effects of inaccuracies in guidance on task execution.** IEEE Transactions on Haptics, 8(2):164–175, April 2015.

[8] SELVAGGIO, M.; ROBUFFO GIORDANO, P.; FICUCIELLOL, F. ; SICILIANO, B.. **Passive task-prioritized shared-control teleoperation with haptic guidance.** In: 2019 INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), p. 430–436, May 2019.

[9] PÉREZ-DEL-PULGAR, C. J.; SMISEK, J.; MUÑOZ, V. F. ; SCHIELE, A.. **Using learning from demonstration to generate real-time guidance for haptic shared control.** In: 2016 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS (SMC), p. 003205–003210, Oct 2016.

[10] XI, B.; WANG, S.; YE, X.; CAI, Y.; LU, T. ; WANG, R.. **A robotic shared control teleoperation method based on learning from demonstrations.** International Journal of Advanced Robotic Systems, 16(4):1729881419857428, 2019.

[11] WANG, X.; YANG, C.; MA, H. ; CHENG, L.. **Shared control for teleoperation enhanced by autonomous obstacle avoidance of robot manipulator.** In: 2015 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), p. 4575–4580, Sep. 2015.

[12] MATSUBARA, T.; HASEGAWA, T. ; SUGIMOTO, K.. **Reinforcement learning of shared control for dexterous telemanipulation: Application to a page turning skill.** In: 2015 24TH IEEE INTERNATIONAL SYMPOSIUM ON ROBOT AND HUMAN INTERACTIVE COMMUNICATION (RO-MAN), p. 343–348, Aug 2015.

[13] SONG, K.; JIANG, S. ; LIN, M.. **Interactive teleoperation of a mobile manipulator using a shared-control approach.** IEEE Transactions on Human-Machine Systems, 46(6):834–845, Dec 2016.

[14] EWERTON, M.; ROTHER, D.; WEIMAR, J.; KOLLEGGER, G.; WIEMEYER, J.; PETERS, J. ; MAEDA, G.. **Assisting movement training and execution with visual and haptic feedback.** Frontiers in Neurorobotics, 12, May 2018.

[15] EWERTON, M.; ARENZ, O.; MAEDA, G.; KOERT, D.; KOLEV, Z.; TAKAHASHI, M. ; PETERS, J.. **Learning trajectory distributions for assisted teleoperation and path planning.** Frontiers in Robotics and AI, 6:89, 2019.

[16] EWERTON, M.; MAEDA, G.; KOERT, D.; KOLEV, Z.; TAKAHASHI, M. ; PETERS, J.. **Reinforcement learning of trajectory distributions: Applications in assisted teleoperation and motion planning**. In: PROCEEDINGS OF THE IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2019.

[17] SCOBEE, D. R. R.; RUBIES ROYO, V.; TOMLIN, C. J. ; SASTRY, S. S.. **Haptic assistance via inverse reinforcement learning**. In: 2018 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS (SMC), p. 1510–1517, Oct 2018.

[18] DE JONGE, A. W.; WILDENBEEST, J. G. W.; BOESSENKOOL, H. ; ABBINK, D. A.. **The effect of trial-by-trial adaptation on conflicts in haptic shared control for free-air teleoperation tasks**. IEEE Transactions on Haptics, 9(1):111–120, Jan 2016.

[19] PASSENBERG, C.; GLASER, A. ; PEER, A.. **Exploring the design space of haptic assistants: The assistance policy module**. IEEE Transactions on Haptics, 6(4):440–452, Oct 2013.

[20] MASONE, C.; GIORDANO, P. R.; BÜLTHOFF, H. H. ; FRANCHI, A.. **Semi-autonomous trajectory generation for mobile robots with integral haptic shared control**. In: 2014 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), p. 6468–6475, May 2014.

[21] NIEMEYER, G.; PREUSCHE, C.; STRAMIGIOLI, S. ; LEE, D.. **Telerobotics**. In: SPRINGER HANDBOOK OF ROBOTICS, p. 1085–1108. 2016.

[22] **Unmanned vehicle**. `https://en.wikipedia.org/wiki/Unmanned_vehicle`. Accessed: 2019-03-17.

[23] ROSENBERG, L. B.. **Virtual fixtures: Perceptual tools for telerobotic manipulation**. In: PROCEEDINGS OF IEEE VIRTUAL REALITY ANNUAL INTERNATIONAL SYMPOSIUM, p. 76–82, Sep. 1993.

[24] ORTMAIER, T.; GROGER, M.; BOEHM, D. H.; FALK, V. ; HIRZINGER, G.. **Motion estimation in beating heart surgery**. IEEE Transactions on Biomedical Engineering, 52(10):1729–1740, Oct 2005.

[25] BRUNNER, B.; ARBTER, K. ; HIRZINGER, G.. **Task directed programming of sensor based robots**. In: PROCEEDINGS OF IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS'94), volumen 2, p. 1080–1087 vol.2, Sep. 1994.

[26] PETERS, J.; LEE, D. D.; KOBER, J.; NGUYEN-TUONG, D.; BAGNELL, J. A. ; SCHAAL, S.. **Robot Learning**, p. 357–398. Springer International Publishing, Cham, 2016.

[27] REDDY, S.; LEVINE, S. ; DRAGAN, A. D.. **Shared autonomy via deep reinforcement learning**. CoRR, abs/1802.01744, 2018.

[28] MUELLING, K.; VENKATRAMAN, A.; VALOIS, J.-S.; DOWNEY, J. E.; WEISS, J.; JAVDANI, S.; HEBERT, M.; SCHWARTZ, A. B.; COLLINGER, J. L. ; BAGNELL, J. A.. **Autonomy infused teleoperation with application to brain computer interface controlled manipulation**. Autonomous Robots, 41(6):1401–1422, Aug 2017.

[29] DRAGAN, A.; SRINIVASA, S.. **A policy blending formalism for shared control**. International Journal of Robotics Research, May 2013.

[30] XU, X.; SONG, A.; NI, D.; LI, H.; XIONG, P. ; ZHU, C.. **Visual-haptic aid teleoperation based on 3-d environment modeling and updating**. IEEE Transactions on Industrial Electronics, 63(10):6419–6428, Oct 2016.

[31] HANNAFORD, B.; OKAMURA, A. M.. **Haptics**. In: SPRINGER HANDBOOK OF ROBOTICS, p. 1063–1084. 2016.

[32] **Touch**. `https://uk.3dsystems.com/haptics-devices/touch?_ga=2.102676423.751424038.1577034862-195074328.1577034862`. Accessed: 2019-12-22.

[33] SUN, H.; MARTIUS, G.. **Machine learning for haptics: Inferring multi-contact stimulation from sparse sensor configuration**. Frontiers in Neurorobotics, 13:51, 2019.

[34] NITSCH, V.; FAERBER, B.. **A meta-analysis of the effects of haptic interfaces on task performance with teleoperation systems**. IEEE Transactions on Haptics, 6:387–398, 08 2013.

[35] YU, N.; WANG, K.; LI, Y.; XU, C. ; LIU, J.. **A haptic shared control approach to teleoperation of mobile robots**. In: 2015 IEEE INTERNATIONAL CONFERENCE ON CYBER TECHNOLOGY IN AUTOMATION, CONTROL, AND INTELLIGENT SYSTEMS (CYBER), p. 31–35, June 2015.

[35] BILLARD, A. G.; CALINON, S. ; DILLMANN, R.. **Learning from humans**. In: SPRINGER HANDBOOK OF ROBOTICS, p. 1995–2014. 2016.

[36] OSA, T.; PAJARINEN, J.; NEUMANN, G.; BAGNELL, J.; ABBEEL, P. ; PETERS, J.. **An algorithmic perspective on imitation learning**. Foundations and Trends in Robotics, 7(1-2):1–179, Mar. 2018.

[38] ZEESTRATEN, M. J. A.; HAVOUTIS, I. ; CALINON, S.. **Programming by demonstration for shared control with an application in teleoperation**. IEEE Robotics and Automation Letters, 3(3):1848–1855, July 2018.

[39] SUTTON, R. S.; BARTO, A. G.. **Introduction to Reinforcement Learning**. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[40] KOBER, J.; BAGNELL, J. A. ; PETERS, J.. **Reinforcement learning in robotics: A survey**. Int. J. Rob. Res., 32(11):1238–1274, Sept. 2013.

[41] DE BRUIN, T.; KOBER, J.; TUYLS, K. ; BABUŠKA, R.. **The importance of experience replay database composition in deep reinforcement learning**. In: DEEP REINFORCEMENT LEARNING WORKSHOP, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NIPS), 2015.

[42] SILVER, D.; LEVER, G.; HEESS, N.; DEGRIS, T.; WIERSTRA, D. ; RIEDMILLER, M.. **Deterministic policy gradient algorithms**. In: PROCEEDINGS OF THE 31ST INTERNATIONAL CONFERENCE ON INTERNATIONAL CONFERENCE ON MACHINE LEARNING - VOLUME 32, ICML'14, p. I–387–I–395. JMLR.org, 2014.

[43] MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D. ; RIEDMILLER, M.. **Playing atari with deep reinforcement learning**. 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[44] LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D. ; WIERSTRA, D.. **Continuous control with deep reinforcement learning**. In: Bengio, Y.; LeCun, Y., editors, ICLR, 2016.

[45] KORENKEVYCH, D.; MAHMOOD, A. R.; VASAN, G. ; BERGSTRA, J.. **Autoregressive policies for continuous control deep reinforcement learning**. In: PROCEEDINGS OF THE 28TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI'19, p. 2754–2762. AAAI Press, 2019.

[46] GOODFELLOW, I.; BENGIO, Y. ; COURVILLE, A.. **Deep Learning**. MIT Press, 2016. `http://www.deeplearningbook.org`.

[47] BENGIO, Y.; COURVILLE, A. ; VINCENT, P.. **Representation learning: A review and new perspectives**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8):1798–1828, Aug 2013.

[48] DUMOULIN, V.; VISIN, F.. **A guide to convolution arithmetic for deep learning**. ArXiv e-prints, mar 2016.

[49] SIMONYAN, K.; ZISSERMAN, A.. **Very deep convolutional networks for large-scale image recognition**. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 2015.

[50] **Convolutional networks - vgg16**. `https://www.jasonosajima.com/convnets_vgg.html`. Accessed: 2019-12-16.

[51] `https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/`. Accessed: 2019-12-26.

[52] **tf.keras.applications.vgg16**. `https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/keras/applications/VGG16`. Accessed: 2019-03-09.

[53] **Applications**. `https://keras.io/applications/`. Accessed: 2019-03-09.

[54] **torchvision.models**. `https://pytorch.org/docs/stable/torchvision/models.html#torchvision-models`. Accessed: 2019-03-09.

[55] **Model zoo**. `https://github.com/BVLC/caffe/wiki/Model-Zoo`. Accessed: 2019-03-09.

[56] TAN, C.; SUN, F.; KONG, T.; ZHANG, W.; YANG, C. ; LIU, C.. **A survey on deep transfer learning**. CoRR, abs/1808.01974, 2018.

[57] YOSINSKI, J.; CLUNE, J.; BENGIO, Y. ; LIPSON, H.. **How transferable are features in deep neural networks?** In: PROCEEDINGS OF THE 27TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS - VOLUME 2, NIPS'14, p. 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[58] **Transfer learning with a pretrained convnet.** `https://www.tensorflow.org/tutorials/images/transfer_learning`. Accessed: 2019-12-26.

[59] JAVDANI, S.; ADMONI, H.; PELLEGRINELLI, S.; SRINIVASA, S. S. ; BAGNELL, J. A.. **Shared autonomy via hindsight optimization for teleoperation and teaming.** The International Journal of Robotics Research, 37(7):717–742, 2018.

[60] PETERS, J.; LEE, D. D.; KOBER, J.; NGUYEN-TUONG, D.; BAGNELL, J. A. ; SCHAAL, S.. **Robot learning.** In: SPRINGER HANDBOOK OF ROBOTICS, p. 357–398. 2016.

[61] **3d systems geomagic touch ros driver.** `https://github.com/bharatm11/Geomagic_Touch_ROS_Drivers`. Accessed: 2019-03-04.

[62] **Dobot magician ros stack.** `https://github.com/wcaarls/dobot`. Accessed: 2019-03-11.

[63] **Logitech.** `https://www.logitech.com/es-roam/product/hd-pro-webcam-c920`. Accessed: 2020-03-04.

[64] **Dobot magician.** `https://www.dobot.cc/dobot-magician/product-overview.html`. Accessed: 2020-02-22.

[65] **3d systems touch haptic device user guide.** `https://www.3dsystems.com/sites/default/files/2017-12/3DSystems-Touch-UserGuide.pdf`. Accessed: 2019-12-26.

[66] SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A. ; KLIMOV, O.. **Proximal policy optimization algorithms.** CoRR, abs/1707.06347, 2017.

[67] WANG, Z.; BAPST, V.; HEESS, N.; MNIH, V.; MUNOS, R.; KAVUKCUOGLU, K. ; DE FREITAS, N.. **Sample efficient actor-critic with experience replay.** CoRR, abs/1611.01224, 2016.

[68] FUJIMOTO, S.; VAN HOOF, H. ; MEGER, D.. **Addressing function approximation error in actor-critic methods.** CoRR, abs/1802.09477, 2018.

[69] HAARNOJA, T.; ZHOU, A.; ABBEEL, P. ; LEVINE, S.. **Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.** CoRR, abs/1801.01290, 2018.

A
Simulator ROS nodes

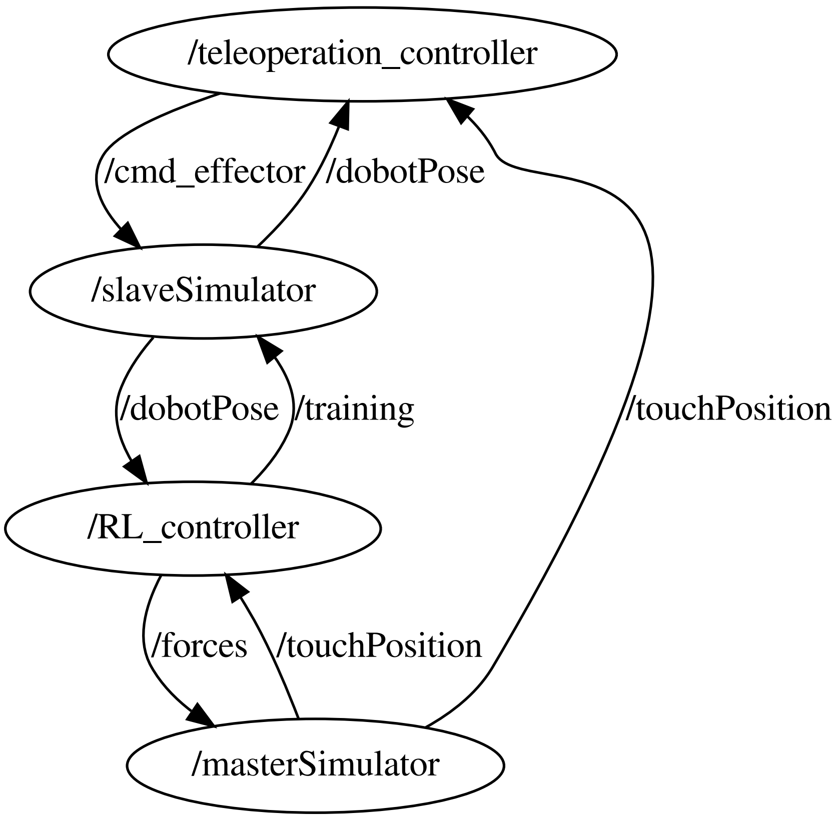


Figure A.1: ROS nodes from simulator

B
Real system ROS nodes

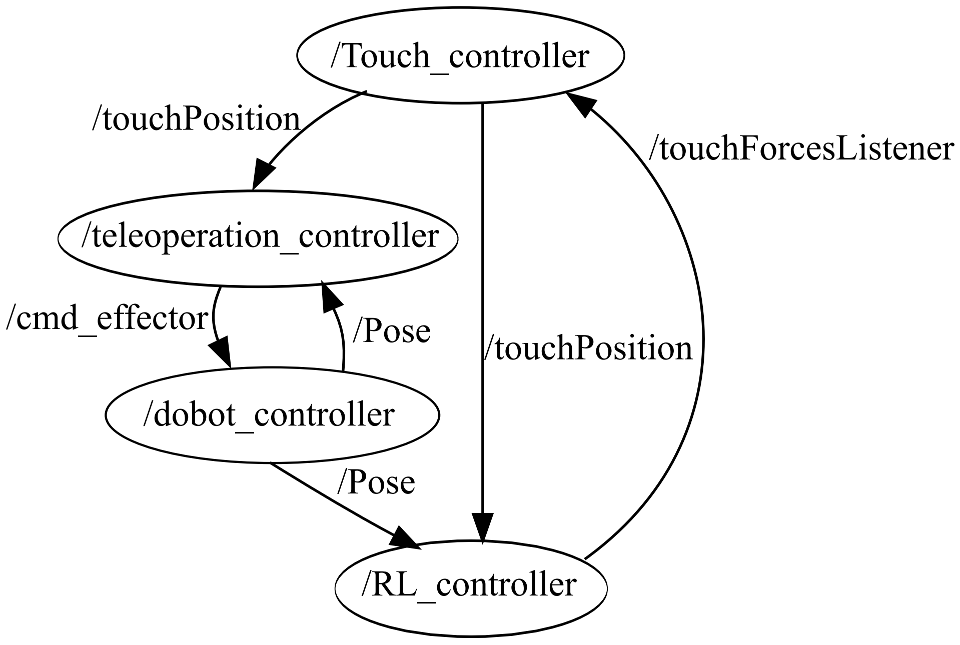
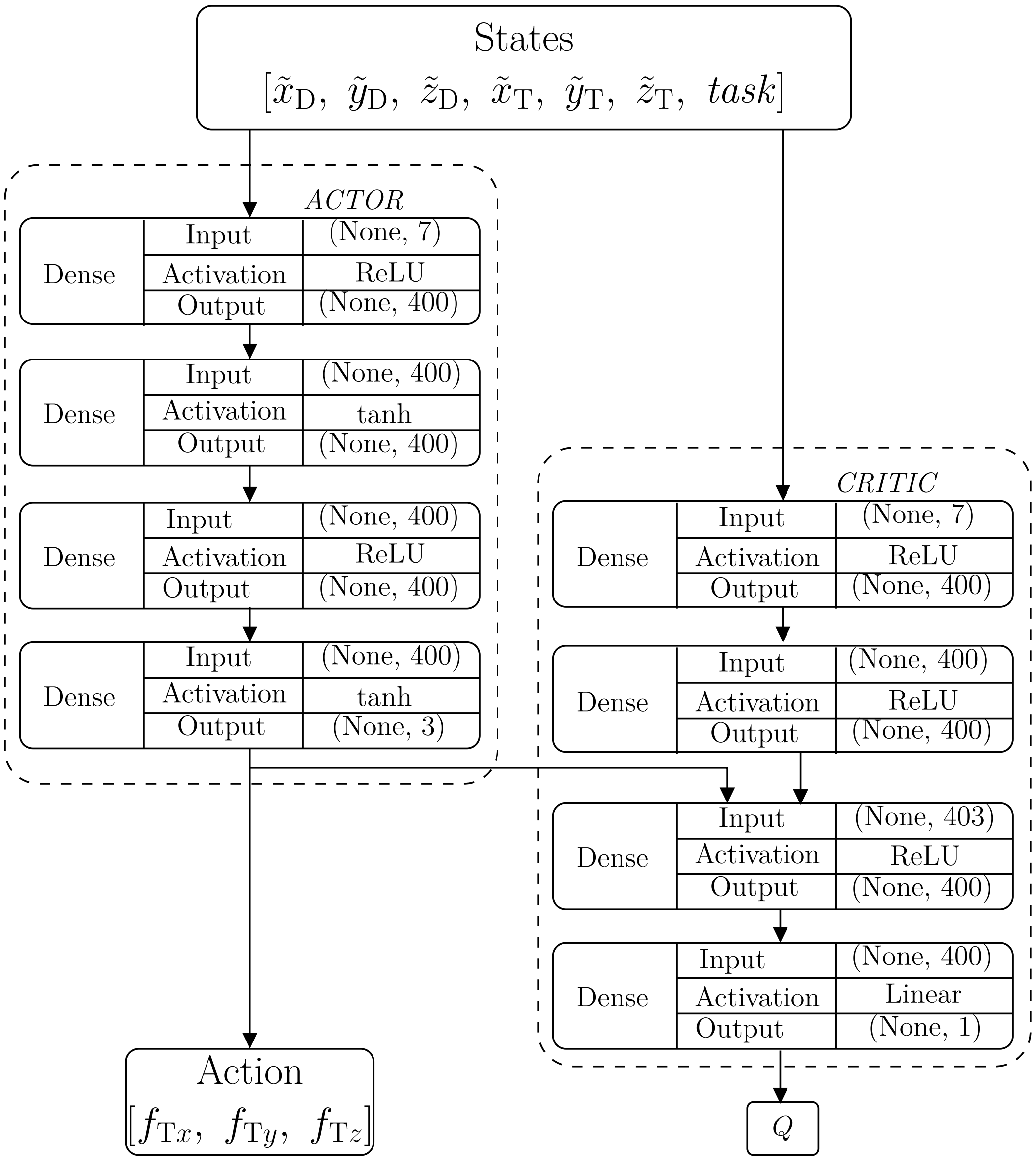


Figure B.1: ROS nodes from real system

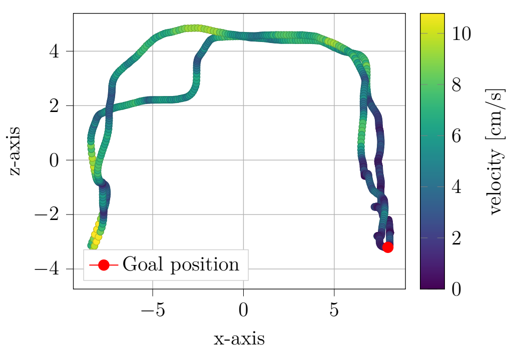
C
DDPG modified network architecture



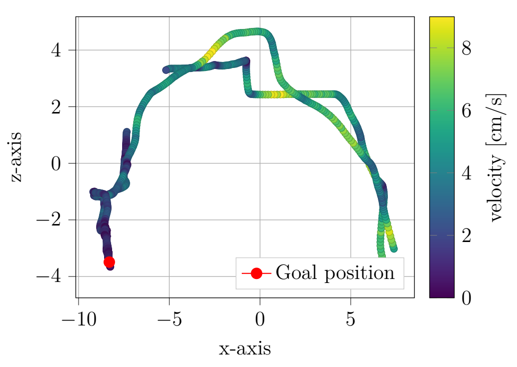
States
$[\tilde{x}_\mathrm{D},\ \tilde{y}_\mathrm{D},\ \tilde{z}_\mathrm{D},\ \tilde{x}_\mathrm{T},\ \tilde{y}_\mathrm{T},\ \tilde{z}_\mathrm{T},\ task]$
ACTOR
Dense | Input (None, 7) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation tanh | Output (None, 400)
Dense | Input (None, 400) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation tanh | Output (None, 3)
CRITIC
Dense | Input (None, 7) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation ReLU | Output (None, 400)
Dense | Input (None, 403) | Activation ReLU | Output (None, 400)
Dense | Input (None, 400) | Activation Linear | Output (None, 1)
Action
$[f_{\mathrm{T}x},\ f_{\mathrm{T}y},\ f_{\mathrm{T}z}]$
Q

Figure C.1: DDPG modified architecture implemented.

D
Behavior obtained before perform training

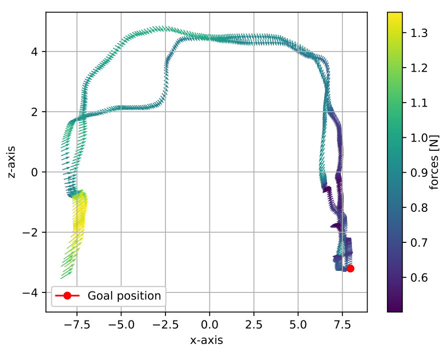


(a) Left direction.

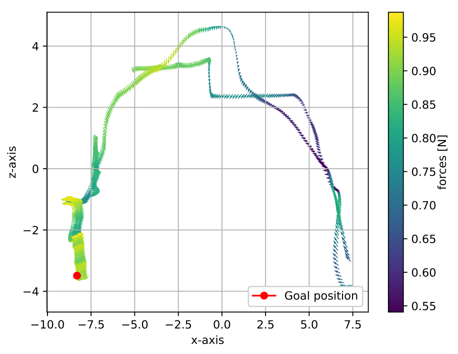


(b) Right direction.

Figure D.1: Trajectory obtained for Subject 1 in the slave side with assistance forces. Notice the difficult to perform the task when the policy is not suitable for the user.

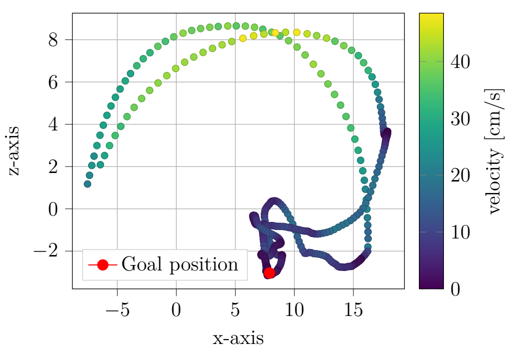


(a) Left direction.

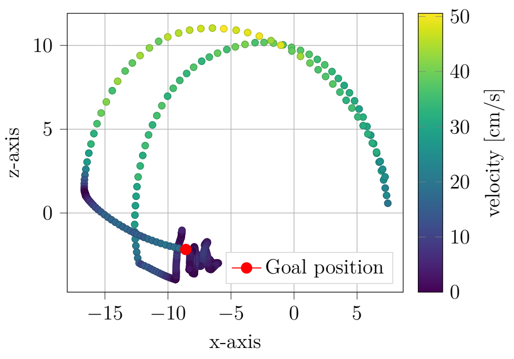


(b) Right direction

Figure D.2: Assistance forces for Subject 1 in the slave side. Despite the forces assist the user, the behavior obtained is not optimal

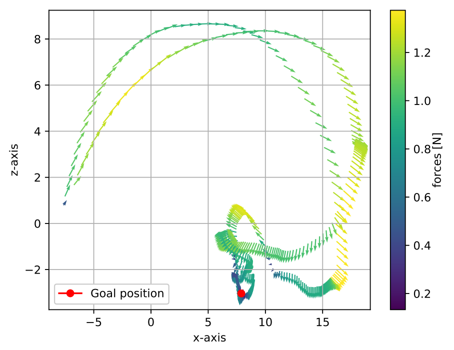


(a) Left direction.

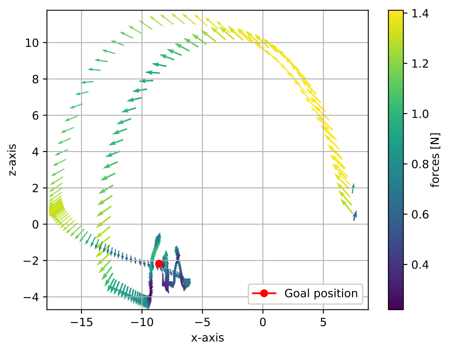


(b) Right direction.

Figure D.3: Trajectory obtained for Subject 2 in the slave side with assistance forces. In contrast with Subject 1, Subject 2 behavior is completely different.



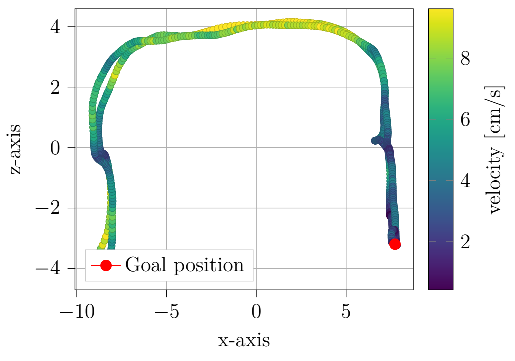
(a) Left direction.



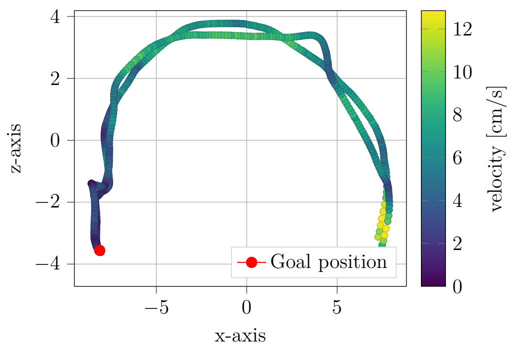
(b) Right direction.

Figure D.4: Assistance forces for Subject 2 in the slave side. Due to the different behavior, the assistive forces magnitude is different for Subject 2.

E
Behavior obtained after perform training

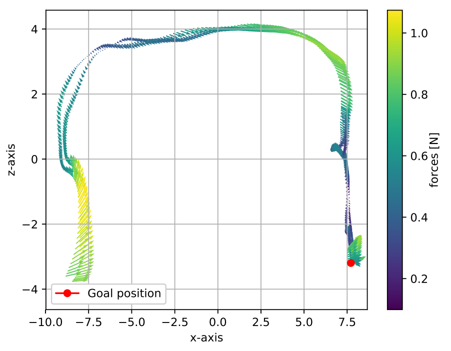


(a) Left direction.

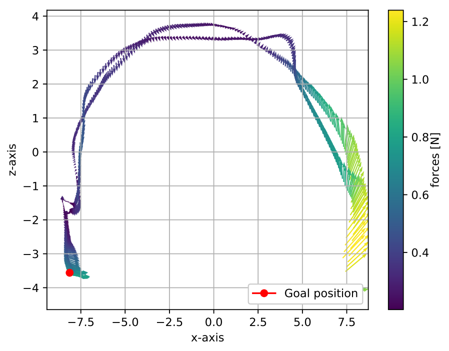


(b) Right direction.

Figure E.1: Trajectory obtained for Subject 1 in the slave side with assistance forces. As expected, the trajectory obtained for his own policy is smoother.

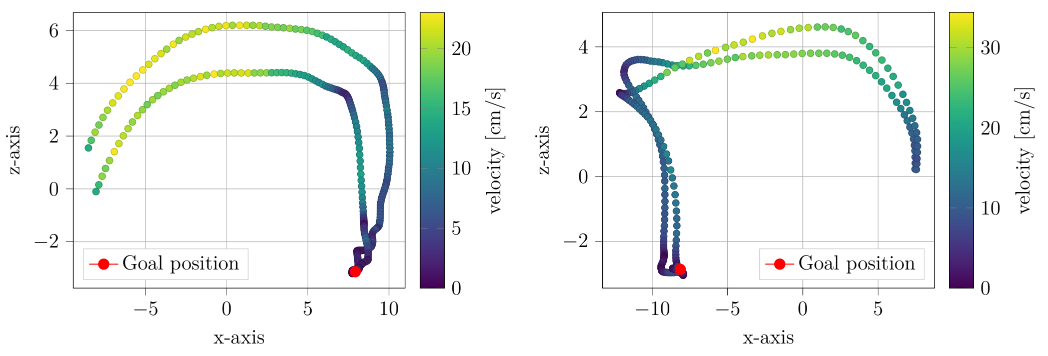


(a) Left direction.



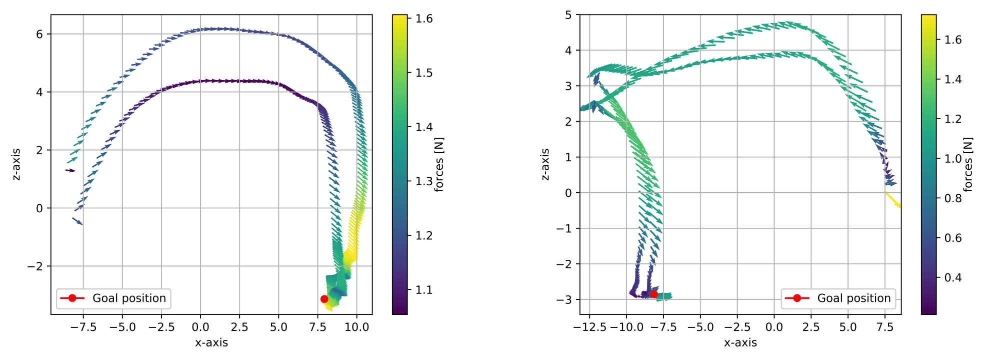
(b) Right direction.

Figure E.2: Assistance forces for Subject 1 in the slave side. Notice that the forces are more suitable when the subject trains his own policy.



(a) Left direction. (b) Right direction.

Figure E.3: Trajectory obtained for Subject 2 in the slave side with assistance forces. Similar to previous subject, Subject 2 presents a better behavior when uses its own policy.



(a) Left direction. (b) Right direction.

Figure E.4: Assistance forces for Subject 1 in the slave side. Notice the difference of the assistance for each direction.

F
Results for Subject 1 with decreased threshold x

As expected, the performance achieved by Subject 1 was influenced by varying the value of the threshold x. As result, the number of steps required to reach the goal position decreased as shown in Table F.1. In addition, velocity is also influenced by the effect of receiving better assistance through the guiding forces (See Table F.2).

This influence can be observed when the obtained results for the different threshold values are compared as presented in Tables F.4, F.5 and F.6. Where Subject 1 presented better performance using the decreased threshold in comparison with original value. Then, the resulting performance can be summarized as less number of steps required, higher velocity and better assistance received.

Moreover, by taking into account the mean values of the force received and the velocity applied for every direction, it can be inferred that the user executes the task in a different way in every direction. These results demonstrate the particular behavior for every user to execute any movements. Despite that the same person is performing the task with the same hand, it can be noticed a difference in the expertise between performing the left and right movements.

Finally, an extra conclusion from the results in Table F.4 lies in the number of steps required to perform the task when all the stages are compared. Again, it is observable that the steps required to perform the task decreases for both cases: with and without receive assistance. This could mean that according ot the user trains the agent, the assistive forces help the user to increase his ability to perform the task, independent of whether he receives assistance or not.

Table F.1: Number of steps per epoch for decreased threshold ($x = 3$). The bold values represents the less amount the steps

Number of epochs per epoch		
Mean	**Std dev**	**Assistance**
236.75	28.9	-
192.25	**44.31**	✓

Table F.2: Velocity results for decreased threshold ($x = 3$). The bold values represents the highest value for all cases.

Velocity [cm/s]			
Max	**Mean**	**Direction**	**Assistance**
14.894	5.577	**Left**	-
18.622	**7.112**		✓
18.262	6.337	**Right**	-
18.539	**7.667**		✓

Table F.3: Force results for decreased threshold ($x = 3$). The bold values represents the highest value for all cases.

Forces[N]		
Max	**Mean**	**Direction**
1.307	0.888	**Left**
1.492	1.123	**Right**

Table F.4: Number of steps per epoch comparing the different tests. The bold values represents the less amount the steps.

Number of steps per epoch			
	Mean	**Std. dev**	**Assistance**
Before training			
	496.25	107.8	-
	487.5	**54.51**	✓[1]
After training			
Threshold 6	337.25	79.65	-
	315.5	**23.01**	✓
Threshold 3	236.75	28.65	-
	192.25	**28.9**	✓

[1]Using *policy-zero*

Table F.5: Comparison of velocity results for original and decreased thresholds. The bold values represents the highest value for all cases.

Velocity [cm/s]			
	Max	**Mean**	**Assistance**
Left direction			
Threshold 6	10.751	4.861	-
	11.328	**4.848**	✓
Threshold 3	14.894	5.577	-
	18.622	**7.112**	✓
Right direction			
Threshold 6	9.558	4.076	-
	13.175	**4.578**	✓
Threshold 3	18.262	6.337	-
	18.539	**7.667**	✓

Table F.6: Comparison of force results for original and decreased thresholds. The bold values represents the highest value for all cases.

Forces [N]		
	Max	**Mean**
Left direction		
Threshold 6	1.12	0.599
Threshold 3	**1.307**	**0.888**
Right direction		
Threshold 6	1.248	0.46
Threshold 3	**1.492**	**1.123**