Università
degli Studi
della Campania
Luigi Vanvitelli

Dipartimento di
Ingegneria

Corso di Laurea Magistrale in
Ingegneria Informatica

Tesi di Laurea in Robotica e Automazione

Collaborative robotized polishing of plastic optical fibers in sensors for SARS-CoV-2 detection

Candidato
Luigi Vetrella
Matr. A18/288

Relatore
Ch.mo Prof.
Ciro Natale

Correlatore
Prof.
Nunzio Cennamo

A.A. 2019/2020

A mia madre

Abstract

In a period of global health emergency such as that derived from the COVID-19 pandemic rapid, low-cost and efficient instruments are needed in order to allow the detection of pathogenic agents as well as automatic procedures are required for a mass production in a perspective of large scale use of these tools.

This thesis introduces a novel approach to automate the polishing task of innovative plastic optical fibers in sensors for SARS-CoV-2 detection by allowing to relieve human operator from an exhausting and alienating work and to increase the amount of produced sensors at the same time. The proposed approach is based on the use of a collaborative robot with the implementation of a force control algorithm to allow the correct execution of the polishing task which considers the motion along a Lissajous figure while a desired force value is exerted on the sensor during the movement. The thesis starts with a detailed description of the problem and the objectives, then it reports the experimental setup based on a commercial cobot, the robot control algorithm and the artificial intelligence approach to operate the production system. To validate the proposed approach, simulations and experimental results are finally reported.

Sommario

In un periodo di emergenza sanitaria globale come quello derivato dalla pandemia da COVID-19 sono necessari strumenti rapidi, economici ed efficienti per consentire la rilevazione degli agenti patogeni così come procedure automatiche sono necessarie per una produzione di massa in un'ottica di uso su larga scala di questi strumenti.

Questa tesi presenta un nuovo approccio per automatizzare l'attività di lucidatura di fibre ottiche plastiche innovative in sensori per il rilevamento del SARS-CoV-2 consentendo di sollevare l'operatore umano da un lavoro estenuante ed alienante e di aumentare allo stesso tempo la quantità di sensori prodotti. L'approccio proposto si basa sull'utilizzo di un robot collaborativo con l'implementazione di un algoritmo di controllo in forza per consentire la corretta esecuzione del compito di lucidatura che prevede il movimento lungo una figura di Lissajous mentre un valore di forza desiderato viene esercitato lungo la perpendicolare sul sensore durante il moto. La tesi parte con una descrizione dettagliata del problema e degli obiettivi, in seguito riporta il setup sperimentale basato su un cobot commerciale, l'algoritmo di controllo del robot e l'approccio basato sull'intelligenza artificiale per gestire il sistema di produzione. Per convalidare l'approccio proposto, vengono infine riportati simulazioni e risultati sperimentali.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work tackles polishing problem of plastic optical fibers into sensors which can be used for SARS-CoV-2 detection. The polishing of these sensors, designed by the Optoelectronics research group of University of Campania "Luigi Vanvitelli", is typically carried out manually whit a huge waste in terms of human work and time as single sensor polishing requests some minutes. Consequently, nowadays, there is not the possibility to execute the task continuously unless the operator is subject to enslaving and tiring work.

In order to preserve human work and ensure sensor production in large-scale use, this thesis proposes an automated approach in which a collaborative robot carries out the task of multi-sensor polishing. In particular, this work proposes a force control algorithm which can allow the robot to interact with the surrounding environment safely, by sharing workspace with operators, who can perform the other operations necessary to sensor production, and executing a correct polishing task at the same time.

1.1 COVID-19 pandemic

COVID-19 pandemic has upset our lifestyles from all points of view: from social interactions to work, from family relationships to the way we enjoy our hobbies: singing, dancing, going to the gym or playing football. Social distance, face mask and personal hygiene have become the three keywords of our lives.

This is due to the rapid spread of the COVID-19 pandemic which has generated a huge international public health emergency. The pathogen responsible for COVID-19, initially called 2019-nCoV and then SARS-CoV-2, which originally occurred in China in December 2019, is a virus belonging to the betacoronavirus genus (Coronaviridae family) and is closely related to other coronaviruses responsible for similar *severe acute respiratory syndrome* (SARS) contagions in 2002-2003 [9].

The speed of transmission of this emerging coronavirus has made its

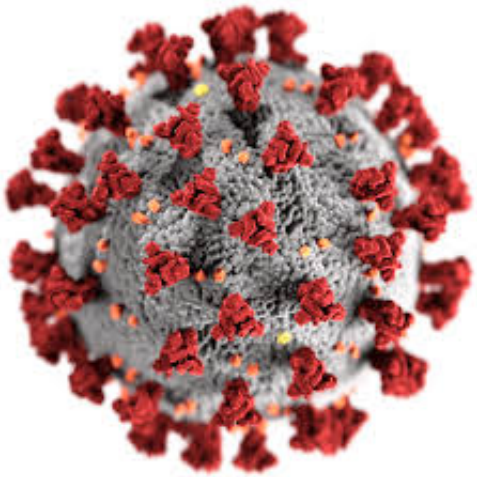


Figure 1.1: SARS-CoV-2 virus

containment extremely difficult and has forced national healthcare systems to reduce their daily activities to a minimum [8]. Diagnostic testing becomes a particularly important tool, in the absence of an effective therapy or a vaccine, to improve patient management and potentially save lives by limiting the contagion. Currently the presence of the virus in patients is routinely determined by molecular techniques which identify viral RNA through nucleic acid reverse transcription and amplification, RT-PCR [22]. However this technique is not readily deployable due to the high cost of real-time PCR machines and the expertise needed to carry out the analysis.

Furthermore, the huge demand for testing has caused shortages of reagents, materials and serious difficulties in the service provision of laboratories. Real-time PCR based diagnosis takes at least 3 hours including sample preparation. Moreover, collected samples (oral or nasopharyngeal swabs, sputum, bronchoalveolar lavage fluid etc.) are often stored, after sampling, up to 24-48 hours prior to analysis. Consequently, the SARS-CoV-2 pandemic highlighted the need for more suitable and effective tools for the detection and prevention of outbreaks [19].

After this SARS-CoV-2 pandemic, in order to obtain a low-cost, simple to use and small-size sensing platform to detect the virus in the population, novel sensor systems will be required to analyse biological fluids in real-time and transmit results via the Internet. Thus, the realization of simple, effective, low-cost and rapid diagnostics has become a global priority.

1.2 Optical Fibers

Optical fiber is currently the most widely used means of communication for broadband cabling of computer networks and network infrastructures.

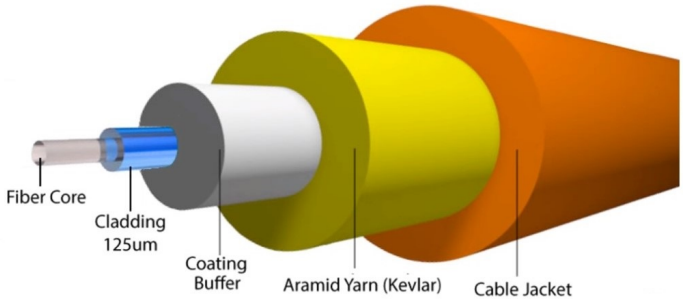


Figure 1.2: Optical fiber structure

It represented the main response to the need for transmission bandwidth, which arose in the 1980s following the spread of the Internet. This dielectric structure, intended for the guided propagation of light, is widely used for applications in the field of telecommunications, medical diagnostics and structural monitoring.

Optical fibers are guiding structures made from filaments of glassy or polymeric materials, inside which an electromagnetic wave such as light propagates. Optical fibers are flexible, resistant to electromagnetic interference, not very sensitive to the most extreme atmospheric conditions and are very light. They allow to convey and guide inside them an electromagnetic field of sufficiently high frequency with extremely limited losses. The optical fiber consists of an internal part called *core*, covered with a concentric coating called *cladding*; core and cladding, in turn, are covered with a coating of plastic material, the *buffer*, and finally on the outside of the fiber there is a protective polymer sheath called *jacket*. A generic optical fiber structure can be seen in figure 1.2.

Fiber core is the nucleus within which propagation takes place and has a diameter of a few tens of μm; cladding is a very thin coating (with a diameter of about $125\,\mu m$) and has a lower refractive index than the core in order to guarantee, under certain conditions, the propagation of light in the fiber; buffer is made of plastic material which performs the function of capturing light not reflected by the cladding; jacket is the outermost coating and has a diameter of about $250\,\mu m$: its main purpose is to give mechanical strength and protect the fiber from abrasion or mechanical scratches. Depending on the applications, a multiple protective coating can be obtained due to external agents: a second protective coating and an additional outer sheath with the aim of protecting the fiber from atmospheric factors (humidity, temperature) or mechanical resistance (installations details, curvatures).

1.3 Surface Plasmon Resonance (SPR)

In order to face COVID-19 pandemic and try to resolve the problems mentioned in section 1.1 many efforts have been done in realizing more rapid diagnostic tools, mostly based on the determination of antibodies in blood and few concerning sensing devices for the detection of the virus in other biological samples.

Some researchers realized biosensor [1] for detecting SARS-CoV-2 in clinical specimens by using an antibody as the recognition element. This kind of sensors are interesting because they represent a general-purpose sensing platform, able to monitor a reprogrammable Molecular Recognition Element (MRE), and could be highly desirable to face the next crisis. Alternatively, surface plasmon resonance sensing platform is technologically suitable to monitor specific receptors (natural or synthetic). In fact, *surface plasmon resonance* (SPR) is widely exploited as an optical detection method for monitoring interactions between an analyte in solution and an MRE immobilized on the SPR sensor.

Surface Plasmon Resonance is known to be a very sensitive technique used to determine the refractive index variations of the interface between a metal layer and a dielectric medium. SPR is a widely used technique as a sensing principle for many sensors operating in various fields of application, such as chemical sensors and biosensors; it is based on the interaction of light and free electrons in the semi-transparent noble metal layer (usually gold) placed on a dielectric substrate. When a biological or chemical receptor (ligand) is bonded on the metallic layer surface, its interaction with the target molecules changes the refractive index at the outer interface, and this variation is detected by optical interrogation. The sensitivity of the plasmonic phenomenon exponentially decreases with the distance from the metal-dielectric interface as the effective interaction length is usually no larger than a few hundred nanometers. So, in SPR detection, the MRE coupled with a metal surface (usually gold), selectively recognizes and captures the substance in the sample, producing a local variation of the refractive index. The extent of the change in the refractive index depends on the thickness of the selective layer (for example molecularly imprinted polymers, see section 1.5) and on the structure and size of the target element (virus, molecule, etc.).

[1] A biosensor is a device containing a biological system in contact with an optoelectronic transducer capable of providing a signal proportional to the concentration of an analyte or group of analytes. The biological system can consist of proteins, cells, enzymes, DNA, antibodies/antigens. Biological molecules interact with some molecules in a specific and reversible way, changing some physics-chemical parameters associated with the interactions. What makes a biosensor highly selective is the ability of some substances to react only with a certain analyte. The optoelectronic transducer usually measures changes in refractive index or fluorescence emission.

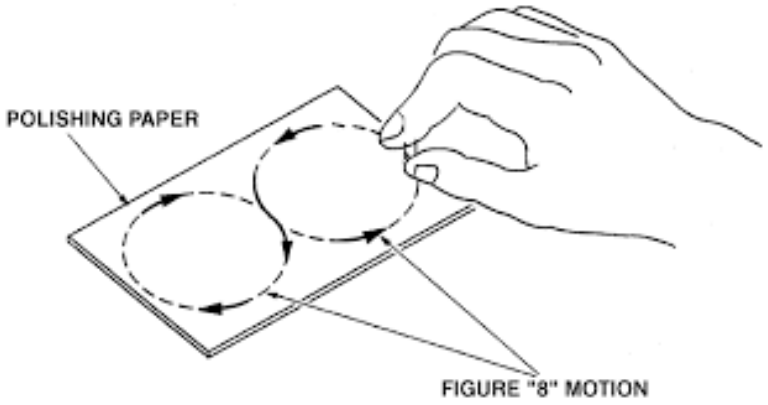


Figure 1.3: Polishing procedure

1.4 Plastic Optical Fiber (POF)

A POF SPR optical sensor is made by removing the coating of a plastic optical fiber along the semicircle, spin coating the exposed core with a Microposit S1813 photoresist buffer, and finally sputtering a thin film of gold using a sputter coater [4].

The plastic optical fiber has a PMMA core of $980\,\mu m$ and a fluorinated polymer cladding of $20\,\mu m$. The refractive index is approximately 1.49 for PMMA, 1.41 for fluorinated polymers and 1.61 for the Microposit S1813 photoresist. The initial sample consists of a plastic optical fiber without a protective sheath (jacket) inserted in a resin block, with the aim of facilitating the subsequent polishing process. A special pliers for plastic optical fibers are used to remove the jacket but also to cleanly and neatly cut fiber segments.

The polishing process is carried out with a $5\,\mu m$ sandpaper to remove the cladding and part of the core. After placing the sheets on a glass base, the end part of the fiber is rubbed on them with a circular motion as if to form an "8" (see figure 1.3). Subsequently, the described operations are repeated with $1\,\mu m$ sandpaper. The sensitive region created is about $10\,mm$ in length.

The Microposit S1813 photoresist buffer is made by rotational coating. The photoresist is deposited (about $0.1\,ml$) in the center of the substrate; the sample is then rotated at $6000\,rpm$ for $60\,s$ in order to obtain a final thickness of about $1.5\,\mu m$. Finally, a thin film of gold is deposited by sputtering using a sputter coater. The sputtering process is repeated three times with a current of $60\,mA$ for a time of $35\,s$ ($20\,nm$ at a time).

The first phase consists in inserting the gold in a vacuum chamber, which is evacuated at low pressure and then maintained at a dynamic pressure of $10^{-2} \div 10^{-4}\,mbar$ by means of argon. The second phase begins when this pressure is reached: at this point the cathode is fed at a high voltage and in these conditions a plasma is formed; the positive ions of the gas (argon) bombard the material to be deposited (target) with a process that can be

defined "atom by atom" allowing the deposition of gold on the substrate considered. The gold film obtained has a thickness of $60\,nm$ and has good adhesion to the substrate, verified by resistance to rinsing in deionized water.

Several SPR sensors have been realized to date, from the classic prism-based configurations to the latest fiber-optic-based ones. Among the latter, SPR sensors based on plastic optical fibers (POFs) are particularly suitable for the development of very low-cost, simple, and small-size sensor devices due to some advantageous properties of the POFs, such as easy manufacturing and high flexibility. Nowadays, POF-based SPR sensors are developed in different configurations, coupled with either biological (antibodies, aptamers, etc.) [5] or chemical receptors such as molecularly imprinted polymers (MIPs, see section 1.5) [3] for environmental [2], industrial and medical applications [25]. In particular, to obtain a general-purpose plasmonic sensor, suitable for different kinds of receptor layers (i.e., bio-receptors or MIPs), several years ago an SPR sensor in D-shaped POFs was designed and realized [4].

1.5 Molecularly Imprinted Polymer (MIP)

Molecularly Imprinted Polymers (MIPs) are synthetic receptors, obtained with the molecular *imprinting* method, which have a number of favourable aspects compared to bio-receptors (for example antibodies) for the detection of specific substances, including better stability, low cost and good reproducibility.

The molecular template technique allows the formation of specific recognition sites in macromolecules. In this process [6], functional monomers and cross-linkers are co-polymerized in the presence of a target analyte. Functional monomers self-organize around the template molecule and through polymerization are held in place in a highly cross-linked structure. After the first phase of polymerization it will be necessary to remove the template molecule from the site; for this purpose, a suitable solvent is used that can only bring the analyte back into solution without changing the structure of the polymer. The extraction of the template creates specific cavities in the sites where the analyte has previously been positioned. The recognition of the analyte by the extracted polymer is of the "key-lock" type: the MIPs have excellent absorption properties for the analyte, due to the stereoscopic shape of the cavities and specific interactions. Furthermore, they are reusable and resistant under various chemical conditions. Figure 1.4 on the following page allows a schematic and simplified view of the manufacturing principle of MIPs.

Figure 1.5 on the next page shows a schematic representation of the SPR-POF-MIP platform production process including the deposition of the specific MIP for the application with the instruments used for measurements (a white light source and a spectrometer) while figure 1.6 on page 8 shows the platform (so the sensor to be polished).

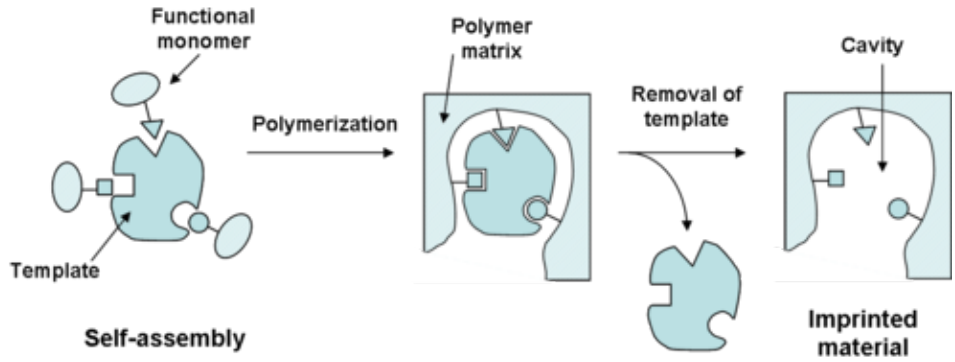


Figure 1.4: Schematic representation of the operating principle of a MIP

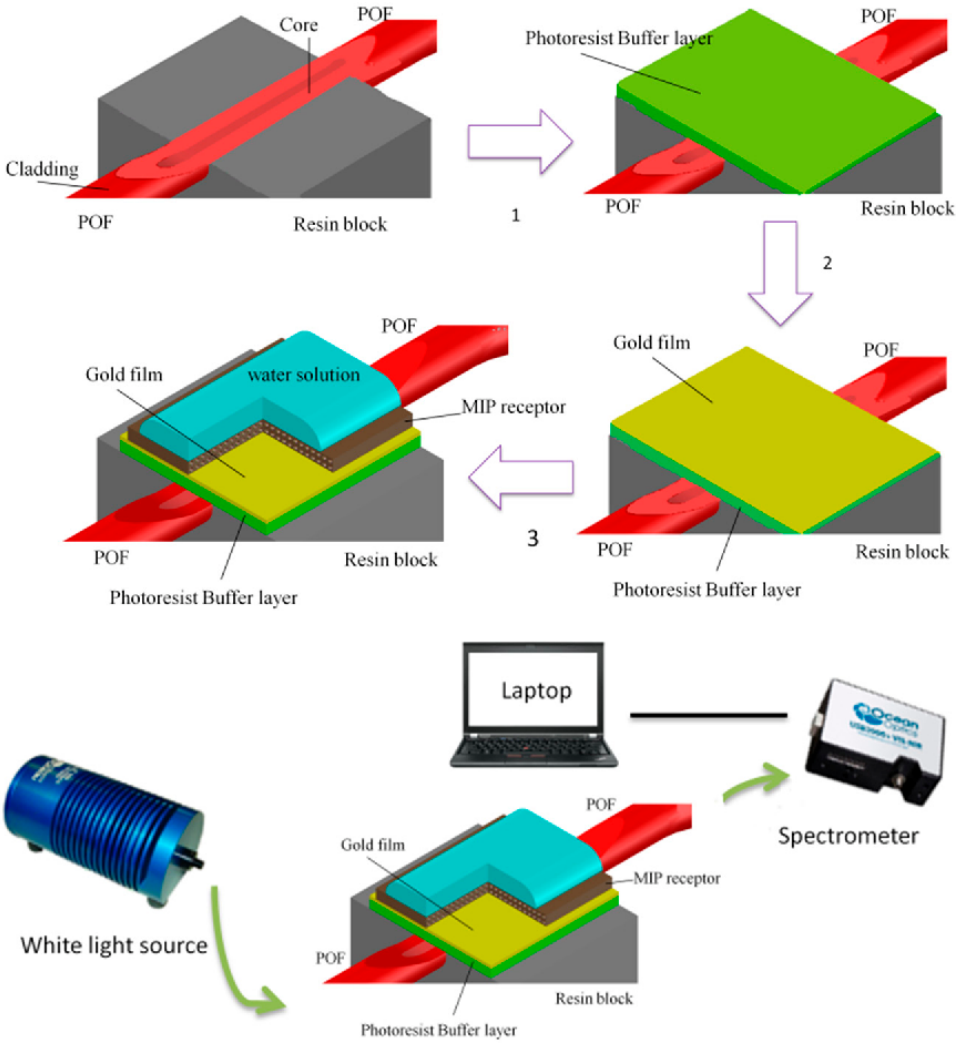


Figure 1.5: Schematic representation of the SPR-POF-MIP platform production process

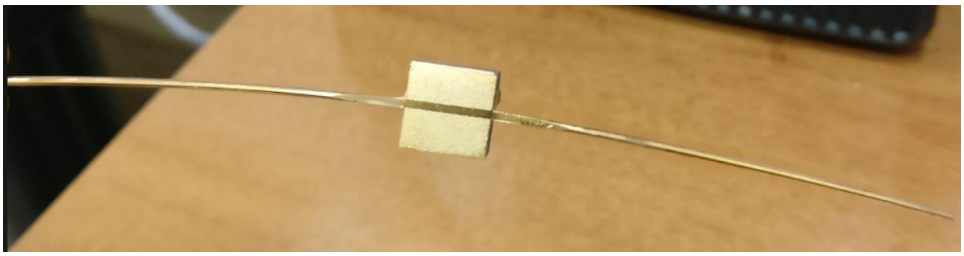


Figure 1.6: Detail of the platform

1.6 Effects of the constructional parameters upon performances

Some researchers investigated the impact of different constructional parameters on the performance of SPR optical fiber sensors based on low-cost multimode POFs [11].

The obtained results can be useful for optimization of the D-shaped and U-bent sensing probes. Findings concerning the D-shaped SPR sensors did not confirm the belief spread in the literature that the optimum polishing depth h should be close to 50% of the fiber core diameter. In contrast, it was demonstrated that a polishing depth equal to 10% of the fiber diameter is enough to obtain resonance of high quality (17% deep for $35\,mm$ long sensing probe), which cannot be improved by further polishing. Moreover, a lower value of the h parameter results in *greater* sensitivity of the SPR sensor based on multimode fiber at the expense of only a small increase in the FWHM (Full Width at Half Minimum) compared to the sensors with h equal to 50% of the core diameter. It was also demonstrated that significant enhancement of the SPR can be achieved by bending the multimode fiber covered with a metal layer. The resonance in a U-bent probe made of stripped fiber with bending radius $R = 10\,mm$ and length $L = 5\,mm$ is stronger than in the sensors based on straight fiber with a sensing part seven times longer, $L = 35\,mm$. So strong amplification of the resonance can potentially be exploited for miniaturization of the SPR sensing probes based on multimode fibers. Finally, it is shown that laborious polishing of the D-shaped probes from the top of bending does not improve their performance compared to probes made of U-bent stripped fiber.

1.7 Proposed approach

As shown in section 1.4, polishing is only one of the steps needed to produce the sensor. In particular, during these steps many equipments are used by the operator in order to obtain the finished product such as a pliers, a machinery for the spin coating and a sputter coater.

Figure 1.7: Human interaction with a cobot

Our approach introduces the use of a collaborative robot (called *cobot*) in order to automate the polishing procedure in a *shared workspace* in which humans and the machine can work in a cooperative way.

According to the human-robot collaboration paradigm the robot no longer needs to be isolated in a work cell but a collaborative workspace, in which human operators and robots interact physically, can be acceptable by proper designing of robots using advanced torque sensors which ensure human safety.

In this way, with a view to a mass production of sensors for large-scale use, it is possible to maximize the production time and relieve humans from the exhausting polishing procedure.

1.8 Thesis outline

This thesis is structured as follows:

- **Chapter 2 - Hardware and Software description**
  This chapter describes hardware and software tools used during the thesis work.
- **Chapter 3 - Panda Robot Setup**
  It describes all setup details for software side robot installation.
- **Chapter 4 - Trajectory Planning of a Lissajous Figure**
  It illustrates the way through which we have planned an operational space trajectory for the polishing by following a path based on a Lissajous figure.

- **Chapter 5 - Force/Compliance Control Algorithm**
  This chapter shows the implemented algorithm on the robot which allows to accomplish the polishing task.

- **Chapter 6 - Task learning from Human Demonstration**
  It shows how we learn the applied force from the operator during the polishing by using an AI framework.

- **Chapter 7 - Experimental results**
  This chapter shows the results of the procedure explained in the previous chapters.

- **Chapter 8 - Conclusions and Future developments**
  The last chapter contains the conclusions and the future developments of this work.

Chapter 2

Hardware and Software description

This chapter illustrates the tools used during this work by making a distinction between hardware and software ones as is usual in computer science engineering. The aim is to highlight their most important features, providing a quick overview of the tools adopted.

2.1 Hardware description

The following section is an overview of the hardware devices, in particular: the manipulator used for the automated polishing, the gripper mounted as robot end-effector for the grasping of the sensors and the force/torque sensor used for the task learning.

2.1.1 Panda Robot - Franka Emika

Panda robot, produced by Franka Emika GmbH (a deep-tech company from Munich, Germany), is a new generation *lightweight* industrial robot which is equipped with 7 revolute joints which make it an *intrinsically redundant* robot. It is characterized by a total weight of $\sim 18\,kg$, having the possibility to handle payloads up to $3\,kg$, and by a workspace of $855\,mm$ radius as shown in figure 2.2 on the next page.

Each joint mounts a torque sensor that provides inherent sense of touch and mechanical adaptiveness; for this reason, like humans, Panda can measure forces to dynamically sense the surrounding environment. Thanks to this feature, force sensitivity can be used to e.g., detect collisions and identify objects.

Panda's equipment consists of:

- Panda Arm (so the manipulator itself);

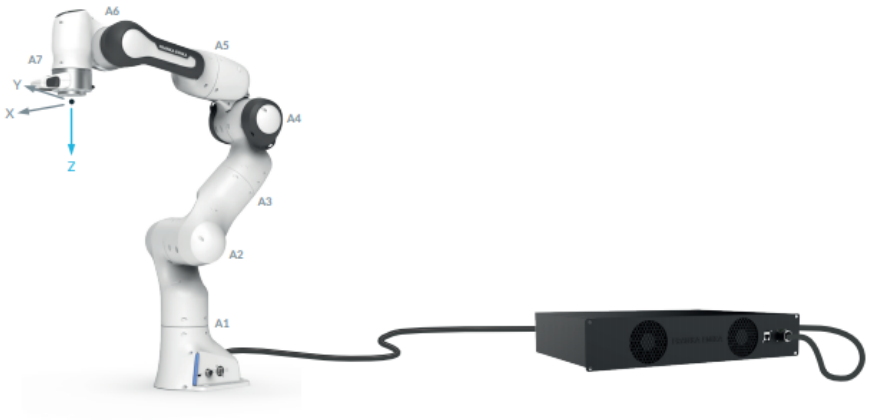
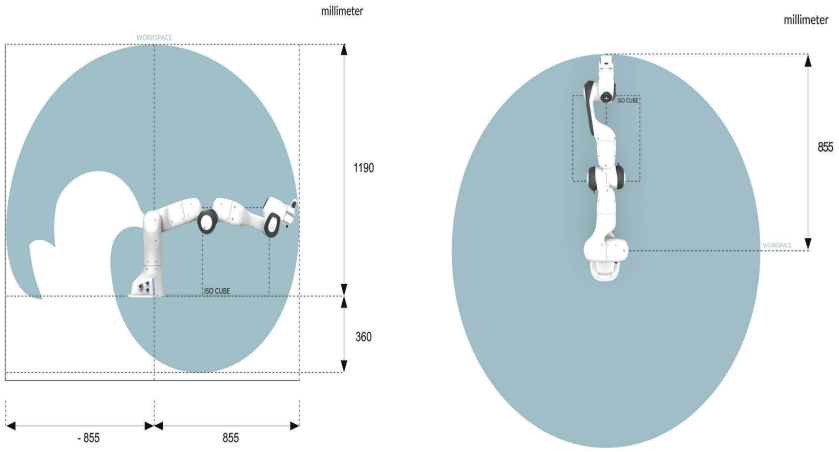


Figure 2.1: Panda robot



(a) *Side-view robot workspace* **(b)** *Top-view robot workspace*

Figure 2.2: Robot workspace

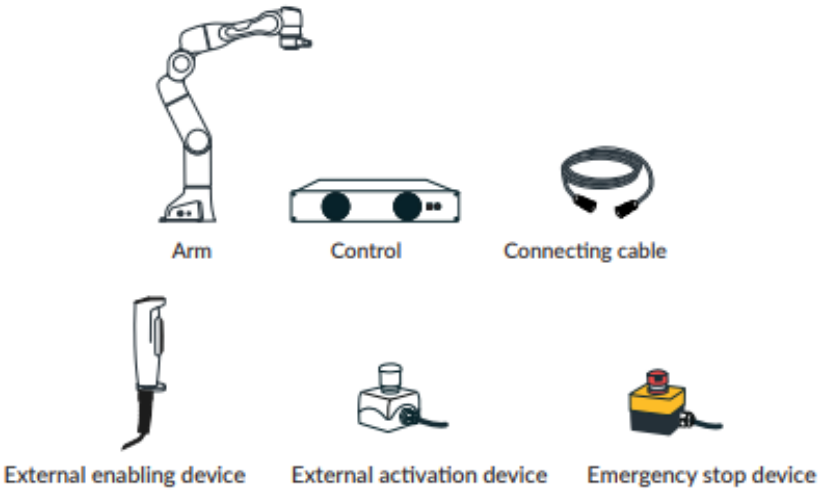


Figure 2.3: Panda equipment

- Panda Control;
- Connecting cable (to connect the arm to the control);
- External enabling device (deadman switch to enable movements);
- External activation device (alternative to the deadman switch but with the same functionality);
- Emergency stop device (to stop the robot in case of emergencies).

Moreover on the robot itself there is a user interface for direct piloting (figure 2.4 on the following page), with which we can control the grippers, move the arm freely or perform other customized tasks.

Figure 2.5 on the next page displays Modified Denavit-Hartenberg parameters of Panda's kinematic chain which are listed in table 2.1 on page 15.

Finally, table 2.2 on page 16 shows the kinematic and dynamic limits in the joint space of the manipulator where the necessary conditions are:

- $q_{min} < q < q_{max}$
- $-\dot{q}_{max} < \dot{q} < \dot{q}_{max}$
- $-\ddot{q}_{max} < \ddot{q} < \ddot{q}_{max}$
- $-\dddot{q}_{max} < \dddot{q} < \dddot{q}_{max}$

and the recommanded conditions are:

Figure 2.4: Piloting interface

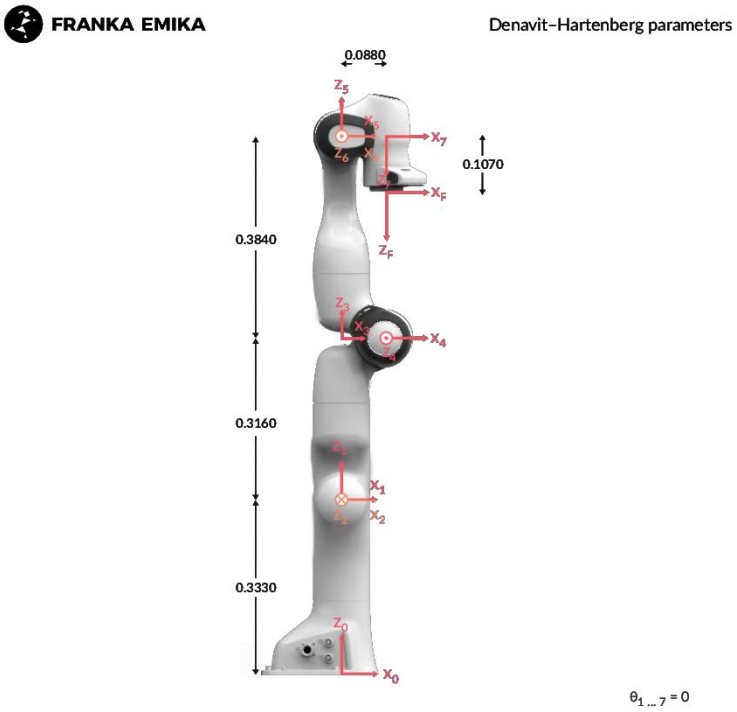


Figure 2.5: Modified Denavit-Hartenberg parameters of Panda's robot

Table 2.1: Modified Denavit-Hartenberg parameters table

Joint	$a\,(m)$	$d\,(m)$	$\alpha\,(rad)$	$\theta\,(rad)$
Joint 1	0	0.333	0	θ_1
Joint 2	0	0	$-\frac{\pi}{2}$	θ_2
Joint 3	0	0.316	$\frac{\pi}{2}$	θ_3
Joint 4	0.0825	0	$\frac{\pi}{2}$	θ_4
Joint 5	-0.0825	0.384	$-\frac{\pi}{2}$	θ_5
Joint 6	0	0	$\frac{\pi}{2}$	θ_6
Joint 7	0.088	0	$\frac{\pi}{2}$	θ_7
Flange	0	0.107	0	

- $-\tau_{j_{max}} < \tau_j < \tau_{j_{max}}$
- $-\dot{\tau}_{j_{max}} < \dot{\tau}_j < \dot{\tau}_{j_{max}}$

2.1.2 Franka Hand

Franka Hand is an electrical two-finger parallel gripper produced by Franka Emika. Hand communicates directly via the connection in the Arm and is also supplied with power from the Arm.

The fingertips can easily be changed and adapted to the objects to be grasped (e.g., using 3D-printed fingertips) while the fingers can be simply mounted differently in order to increase the span length of the gripper (see figure 2.6 on page 17).

Franka Hand has the following characteristics:

- weight of Hand: $0.73\,kg$;
- center of mass of Hand to end-effector flange $[m]$: $F_{x_{Cee}} = [\,-0.01\ 0\ 0.03\,]$;
- inertia tensor $[kg * m^2]$: $I_{ee} = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.0025 & 0 \\ 0 & 0 & 0.0017 \end{bmatrix}$;
- transformation matrix of end-effector flange to Hand (center point of finger tips when closed): $T_{EE}^{F} = \begin{bmatrix} 0.707 & 0.707 & 0 & 0 \\ -0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 1 & 0.1034 \\ 0 & 0 & 0 & 1 \end{bmatrix}$;
- grasping continuous force: $70\,N$;
- maximum grasping force: $140\,N$;

Table 2.2: Joints space limits

Name	Joint1	Joint2	Joint3	Joint4	Joint5	Joint6	Joint7	Unit
q_{max}	2.8973	1.7628	2.8973	−0.0698	2.8973	3.7525	2.8973	rad
q_{min}	−2.8973	−1.7628	−2.8973	−3.0718	−2.8973	−0.0175	−2.8973	rad
$\dot{q}_{max}$	2.1750	2.1750	2.1750	2.1750	2.6100	2.6100	2.6100	$\frac{rad}{s}$
$\ddot{q}_{max}$	15	7.5	10	12.5	15	20	20	$\frac{rad}{s^2}$
$\dddot{q}_{max}$	7500	3750	5000	6250	7500	10000	10000	$\frac{rad}{s^2}$
$\tau_{j_{max}}$	87	87	87	87	12	12	12	Nm
$\dot{\tau}_{j_{max}}$	1000	1000	1000	1000	1000	1000	1000	$\frac{Nm}{s}$

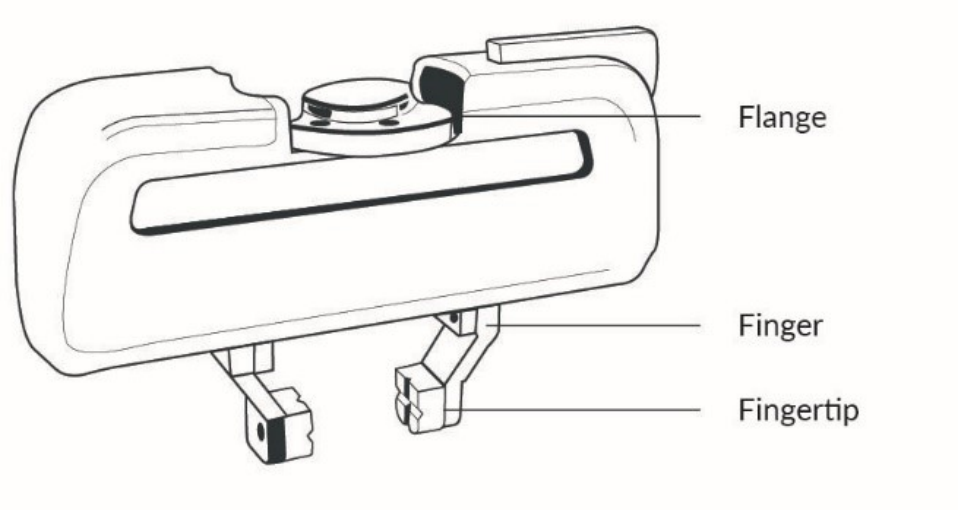


Figure 2.6: Franka Hand

- travel span: $80\,mm$;
- travel speed (per finger): $50\,\frac{mm}{s}$.

2.1.3 Robotous Force-Torque Sensor

The Robotous RFT40-SA01 sensor -used in this work- is an innovative 6-AXIS force torque sensor (see figure 2.7 on the next page).

In particular, RFT Series are capacitive F/T sensors that can measure the six components of force and torque simultaneously. The sensor has an innovative structure design and a robust calibration algorithm to achieve both cost competitiveness and reliable performance. It provides users with various interface options to support a wide range of automation and research applications. It also has an internal overload protection to avoid damage from external impact or excessive load.

Sensor specifications are shown in table 2.3 on the following page.

2.2 Software description

This section is an overview of the software tools, in particular: MATLAB&Simulink used for simulations, App Desk (a web interface to manage the Panda robot), libfranka used for robot programming and ROS used for data collection from the Robotous sensor.

Figure 2.7: Robotous RFT40-SA01

Table 2.3: Specification table Robotous sensor

Item		Unit	RFT40-SA01
Dimension	Diameter	mm	40
	Height	mm	18.5
Data Rate	(maximum)	Hz	200
Load Capacity	F_x, F_y	N	100
	F_z	N	150
	τ_x, τ_y, τ_z	Nm	2.5
Resolution	F_x, F_y	mN	200
	F_z	mN	200
	τ_x, τ_y, τ_z	mNm	8
Overload Capacity	F_x, F_y	%	150
	F_z+	%	150
	F_z-	%	300
	τ_x, τ_y, τ_z	%	150
Hysteresis	F_x, F_y	$\%FS$	2
	F_z	$\%FS$	0.5
	τ_x, τ_y, τ_z	$\%FS$	1
Cross Talk	F_x, F_y	$\%FS$	3
	F_z	$\%FS$	3
	τ_x, τ_y, τ_z	$\%FS$	3
Weight	(w/o cable)	g	60

Figure 2.8: MATLAB&Simulink logo

2.2.1 MATLAB&Simulink

MATLAB is a scientific computing environment that can be used on multiple levels, from the pocket calculator to the simulation and analysis of complex systems, and is developed by *MathWorks* [17].

MATLAB is supported by operating systems such as Windows, Mac OS, GNU/Linux and Unix, and is optimized to solve scientific and design problems. Engineers and scientists from all over the world use this software to analyse and design the systems and products that transform our world: MATLAB is in fact active safety systems in automobiles, in interplanetary spacecraft, in health monitoring devices, in smart power grids and in LTE cellular networks. MATLAB is used for machine learning, signal processing, image processing, machine vision, communications, computational finance, control design, robotics and much more. The name MATLAB is an abbreviation of *Matrix Laboratory*: in fact, its programming language is based on the matrix (a scalar is a 1x1 matrix), which represents the most natural way in the world to express computational mathematics. The integrated graphics, combined with a large library of toolboxes, simplifies visualization and offers a detailed understanding of the data.

Simulink [18] is an environment for modelling, analysing, and simulating dynamic systems, and such as MATLAB is developed by the American company *MathWorks*. Simulink supports the simulation of linear, and/or non-linear systems, which operate with continuous and/or discrete signals of a continuous and/or discrete time type. Simulink provides a graphical interface that allows to define and build the model through its block diagram. This environment includes a large number of libraries that contain many predefined blocks that can perform different signal operations. Simulink is closely integrated with MATLAB. All blocks and parameters used in this works can be seen in the following sections.

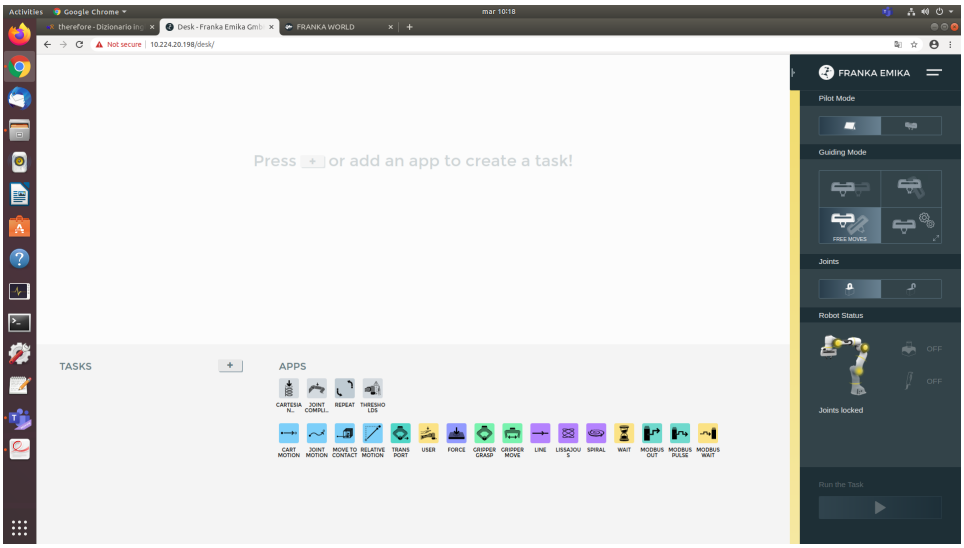


Figure 2.9: App Desk

2.2.2 App Desk

An administration and programming web interface called App Desk is available, accessible at the IP address of the controller (see 3.4.1 for the setup), and with which we connect to the robot before starting to program, with the possibility of unlocking the brakes of each joint. Figure 2.9 shows the main Desk screen.

From this interface, tasks can also be programmed using an intuitive block programming language.[1] In the case of the activities of this thesis, Desk was used only for starting and stopping the robot, locking and unlocking joints, and for managing the manipulator settings. In the side bar we can see which guiding mode Panda is in. In other words, whether, for example, translational or rotational motion is enabled in guiding Panda. Thereunder are situated important notices on Panda's status, such as, for example, whether the external activation switch is on or off, or whether there is an error (red light).

In the Desk settings screen, accessible by clicking on the button in the upper right corner, IP addresses of the controller and the robot are editable in the NETWORK section as shown in figure 2.10 on the following page. Instead in the DASHBOARD section, shown in figure 2.11 on the next page, there is an overview of the connection status to the robot.

In the END-EFFECTOR section, shown in figure 2.12 on page 22, we can modify the mechanical data of the gripper used (mass, position of the center of mass, inertial matrix, etc.) and perform its initialization (homing).

[1] Tasks are program sequences and consist of a chronological sequence of apps. Apps are the building blocks of a task and describe the basic capabilities of Panda, such as "grip", "put down", "push button".

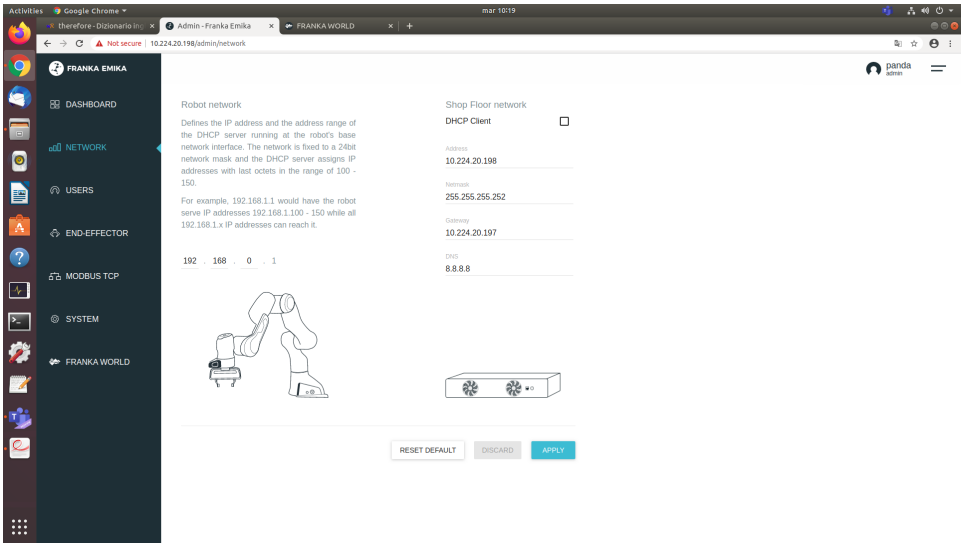


Figure 2.10: NETWORK section

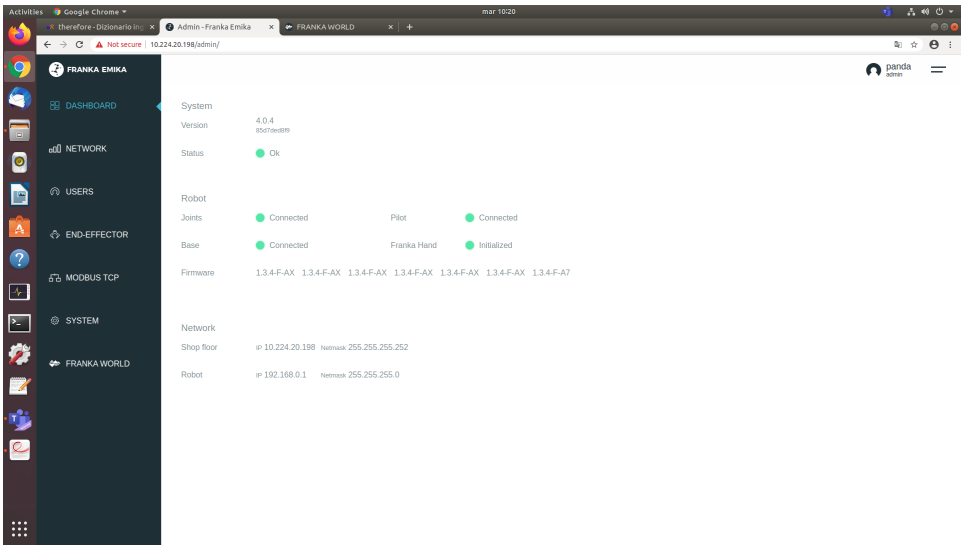


Figure 2.11: DASHBOARD section

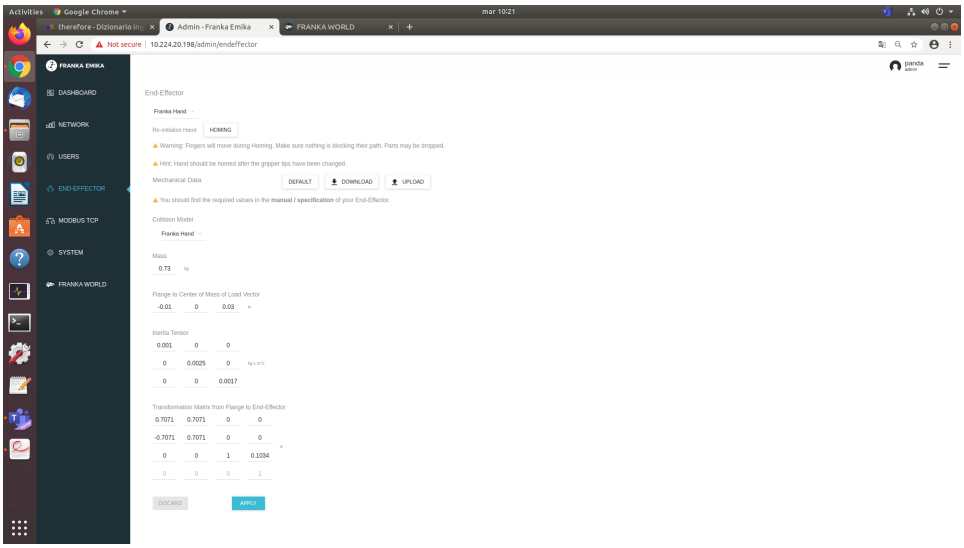


Figure 2.12: END-EFFECTOR section

Finally, Franka World[2] section can be used to automatically update robot (after connecting it properly to the Internet, see subsection 3.4.2). This section is shown in figure 2.13 on the next page.

2.2.3 FCI and libfranka

The Franka Control Interface (FCI) [14] allows a fast and direct low-level bidirectional connection to the Arm and Hand. It provides the current status of the robot and enables its direct control with an external workstation PC connected via Ethernet. By using libfranka, open source C++ interface, we can send real-time control values at $1\,kHz$ with 5 different interfaces:

- Gravity&friction compensated joint level torque commands;
- Joint position command;
- Joint velocity command;
- Cartesian pose command;
- Cartesian velocity commands.

[2]Franka World - `https://world.franka.de/` - is an online platform that interconnects Franka Emika's customers, partners, developers and robots. By bringing Pandas into the cloud, all parties can mutually benefit from each other's interaction, and gain integrated access to the products and services provided by Franka Emika and our network of qualified partners. Franka World provides customers with centralized and remote management of their fleets of Franka Emika robots, and the possibility to access the Store, to browse a continuously growing portfolio of accredited software and hardware extensions.

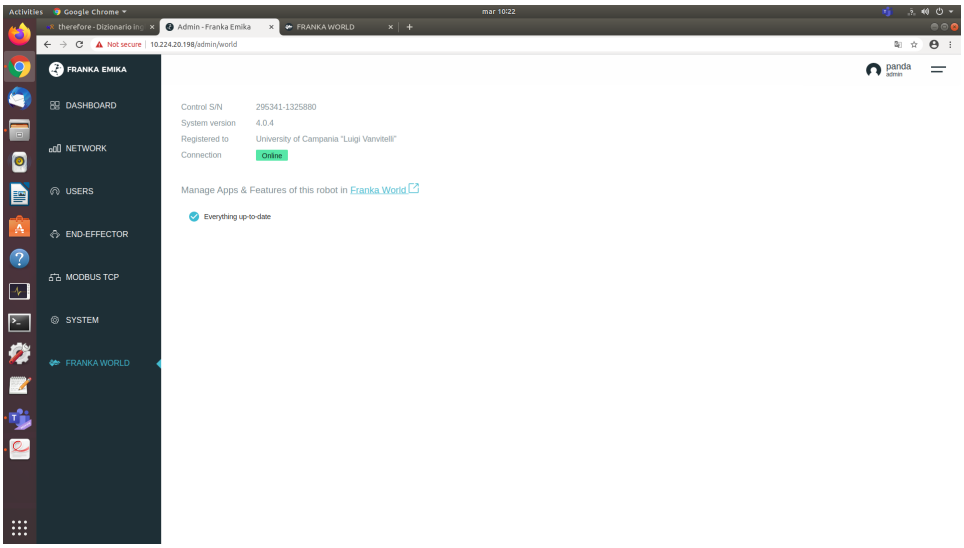


Figure 2.13: FRANKA WORLD section

At the same time, we get access to $1\,kHz$ measurements of:

- measured joint data, such as the position, velocity and link side torque sensor signals;
- estimation of externally applied torques and wrenches;
- various collision and contact information.

It allows access robot model library which provides:

- forward kinematics of all robot joints;
- Jacobian matrix of all robot joints;
- dynamics: inertia matrix, Coriolis and centrifugal vector and gravity vector.

Therefore, libfranka is the C++ implementation of the client side of the FCI. It handles the network communication with Control and provides interfaces to easily:

- execute non-realtime commands to control the Hand and configure Arm parameters;
- execute realtime commands to run our own $1\,kHz$ control loops;
- read the robot state to get sensor data at $1\,kHz$;
- access the model library to compute our desired kinematic and dynamic parameters.

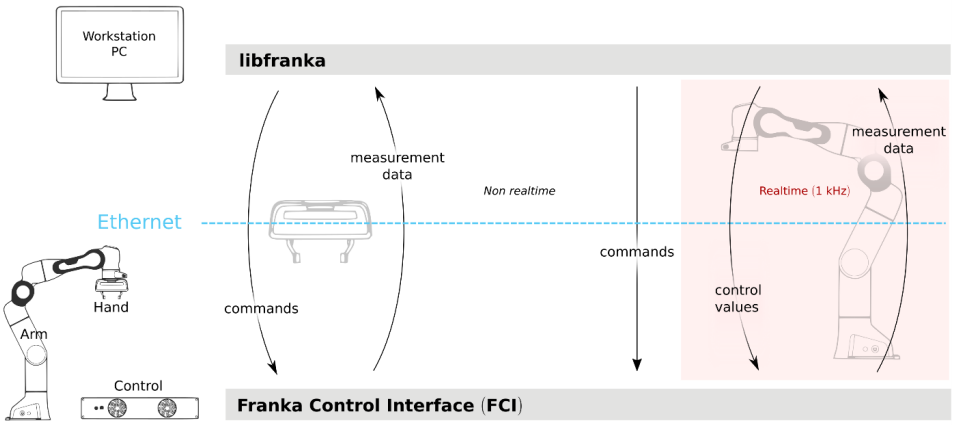


Figure 2.14: Schematic overview

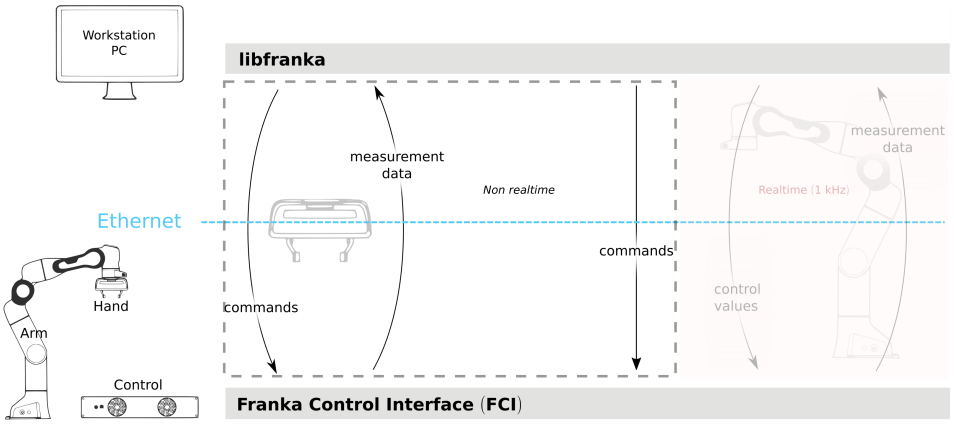


Figure 2.15: Non-realtime commands for both Arm and Hand

Non-realtime commands

Non-realtime commands are blocking, TCP/IP-based and always executed outside of any realtime control loop. They encompass all of the Hand commands and some configuration-related commands for the Arm.

The most relevant ones for the Hand are:

- *homing* which calibrates the maximum grasping width of the Hand;
- *move, grasp* and *stop*, to move or grasp with the Hand;
- *readOnce*, which reads the Hand state.

Concerning the Arm, some useful non-realtime commands are:

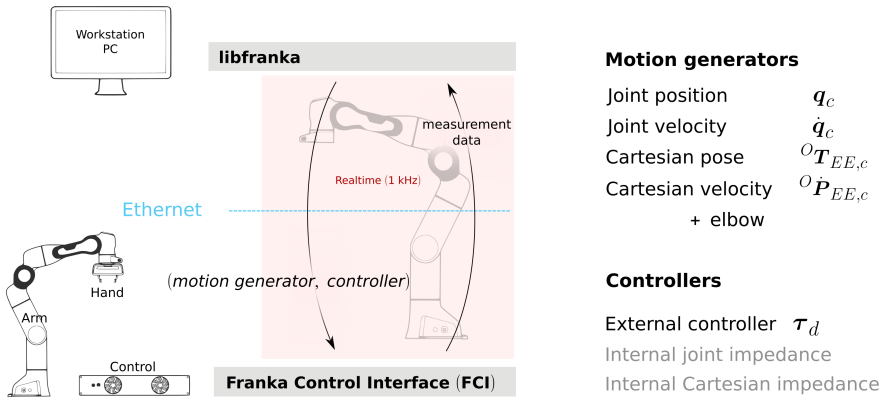


Figure 2.16: Realtime interfaces: motion generators and controllers

- *setCollisionBehavior* which sets the contact and collision detection thresholds;
- *setCartesianImpedance* and *setJointImpedance* which set the impedance parameters for the internal Cartesian impedance and internal joint impendace controllers;
- *setEE, setK* and *setLoad* which set end effector and load parameters;
- *automaticErrorRecovery* that clears any command or control exception that previously happened in the robot.

For a complete and fully specified list check the API documentation for the Arm [13] or the Hand [15]. All operations (non-realtime or realtime) on the Arm or the Hand are performed through the *franka::Robot* and *franka::Gripper* objects respectively.

Realtime commands

Realtime commands are UDP based and require a $1\,kHz$ connection to Control. There are two types of realtime interfaces:

- motion generators, which define a robot motion in joint or Cartesian space;
- controllers, which define the torques to be sent to the robot joints.

There are four different types of external motion generators and 3 different types of controllers (one external and two internal) as depicted in the figure 2.16.

We can either use a single interface or combine two different types. Specifically, we can command:

Figure 2.17: ROS logo

- only a motion generator and therefore use one of the two internal controllers to follow the commanded motion;
- only an external controller and ignore any motion generator signals, i.e., torque control only;
- a motion generator and an external controller to use the inverse kinematics of Control in our external controller.

All realtime loops (motion generator or controller) are defined by a callback function that receives the robot state and the duration of the last cycle ($1\,ms$ unless packet losses occur) and returns the specific type of the interface. The control method of the *franka::Robot* class will then run the control loop by executing the callback function at a $1\,kHz$ frequency. All control loops are finished once the *motion_finished* flag of a realtime command is set to *true*.

For more details about libfranka it is possible to consult the documentation site [16].

2.2.4 ROS

ROS [10] is an open-source *robot operating system* which includes a set of software libraries and tools that help building robot applications that work across a wide variety of robotic platforms. The operating system side provides standard operating system services such as hardware abstraction, low-level device control, message-passing between processes and package management. The ROS runtime "graph" is a peer-to-peer network of processes: it consist of numerous small computer programs which connect to each other and continuously exchange messages. ROS is tools-based since there are many small, generic programs that perform tasks such as visualization, logging, plotting data streams, etc. Another ROS property is the multi-languages in fact the software modules can be written in any language for which a client library has been written: currently client libraries exist for C++, Python, etc.

ROS is based on concepts as nodes, topics, messages and services. The nodes are single-purposed executable programs e.g., sensor drivers, actuator drivers, mapper and planner that are individually compiled, executed, and managed. Nodes are written using a ROS client library (roscpp, rospy), they

can publish or subscribe to a topic and also provide or use a service. Nodes communicate with each other by publishing messages to topics, so the topics represent a stream of messages with a defined type.

Chapter 3

Panda Robot Setup

This chapter shows how we setup Panda robot in laboratory.

After a briefly setup overview, a real-time kernel was installed upon the workstation and network settings were configured properly in order to control the robot through libfranka.

3.1 Setup Overview

In order to setup the robot in the right way, we faced two aspects:

- Workstation PC: Linux with PREEMPT_RT patched kernel or Windows 10 (experimental) is required with a 100BASE-TX network card (Linux system was used for this work);
- Network settings.

Since the robot sends data at $1\,kHz$ frequency, it is important that the workstation PC is configured to minimize latencies. In fact, CPUs are often configured to use a lower operating frequency when under a light load in order to conserve power. Unfortunately, this feature can lead to higher latencies when using libfranka.

To check and modify the power saving mode, *cpufrequtils* package was installed with the following command:

```
sudo apt install cpufrequtils
```

By running *cpufreq-info* it is possible to see available "governors" and the current CPU frequency. Since our CPU was in *powersave* policy it was needed to modify this setting to *performance* by running in the terminal the following commands:

```
sudo systemctl disable ondemand
sudo systemctl enable cpufrequtils
sudo sh -c 'echo "GOVERNOR=performance" > /etc/default/
    cpufrequtils'
```

```
sudo systemctl daemon-reload && sudo systemctl restart
    cpufrequtils
```

which disable the *on-demand* CPU scaling daemon, create a */etc/default-/cpufrequtils* configuration file, and then restart the *cpufrequtils* service.

After we enabled the performance governor, the *cpufreq-info* results in our workstation are:

```
$ cpufreq-info
...
analyzing CPU 0:
  driver: acpi-cpufreq
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated
   by software: 0
  maximum transition latency: 4.0 us.
  hardware limits: 800 MHz - 3.50 GHz
  available frequency steps: 3.50 GHz, 2.80 GHz,
                             2.20 GHz, 800 MHz
  available cpufreq governors: conservative, ondemand,
                               userspace, powersave,
                               performance, schedutil
  current policy: frequency should be within 800 MHz
                  and 3.50 GHz.
                  The governor performance may decide
                  which speed to use within this range.
  current CPU frequency is 3.50 GHz.
...
```

This output shows a CPU frequency of 3.50 GHz which is equal to the maximum one. It is also possible directly verify the current governor using the *cpufreq-info -p* command.

The workstation PC which commands the robot using the FCI is always connected to the LAN port of Control (shop floor network) and not to the LAN port of the Arm (robot network). In fact, having relays in between could lead to delay, jitter or packet loss. This decreases the performance of the controller or makes it unusable.

The best performance can be achieved when connecting directly to the LAN port of Control. This requires setting up a static IP for the shop floor network in the administrator's interface beforehand (see subsection 3.4.1).

Due to $1\,Khz$ work frequency, if the $< 1\,ms$ constraint is violated for a cycle, the received packet is dropped by FCI. After 20 consecutively dropped packets, the robot will stop with the *communication_constraints_violation* error. Current measure of communication quality can be read from the *RobotState::control_command_success_rate* field.

If a controller command packet is dropped, FCI will reuse the torques of the last successful received packet. If more than 20 consecutive lost or dropped packets will cause the robot to stop.

3.2 Setting up real-time kernel

In order to control the robot using libfranka, the controller program on the workstation PC must run with real-time priority under a *PREEMPT_RT* kernel.

For this reason, at first we installed the necessary dependencies:

```
apt-get install build-essential bc curl ca-certificates
    fakeroot gnupg2 libssl-dev lsb-release libelf-dev
    bison flex
```

Then it is necessary to choose the right kernel version to install by using *uname -r* command. Real-time patches are only available for select kernel versions (`https://www.kernel.org/pub/linux/kernel/projects/rt`). In particular we chose the version 4.19.103-rt42 while the one previously used was 5.4.0-51-generic.

Having decided the right version, we used curl to download the source files:

```
curl -SLO https://www.kernel.org/pub/linux/kernel/v4.x/
    linux-4.19.103.tar.xz
curl -SLO https://www.kernel.org/pub/linux/kernel/v4.x/
    linux-4.19.103.tar.sign
curl -SLO https://www.kernel.org/pub/linux/kernel/
    projects/rt/4.14/older/patch-4.19.103-rt42.patch.xz
curl -SLO https://www.kernel.org/pub/linux/kernel/
    projects/rt/4.14/older/patch-4.19.103-rt42.patch.sign
```

and decompressed them with:

```
xz -d linux-4.19.103.tar.xz
xz -d patch-4.19.103-rt42.patch.xz
```

At this point we extracted the source code and applied the patch:

```
tar xf linux-4.19.103.tar
cd linux-4.19.103
patch -p1 < ../patch-4.19.130-rt42.patch
```

The next step was to configure the kernel:

```
make oldconfig
```

This opened a text-based configurations menu and when asked for the Preemption Model, we chose the Fully Preemptible Kernel:

```
1. No Forced Preemption (Server) (PREEMPT_NONE)
2. Voluntary Kernel Preemption (Desktop) (
    PREEMPT_VOLUNTARY)
3. Preemptible Kernel (Low-Latency Desktop) (PREEMPT__LL)
     (NEW)
```

```
4. Preemptible Kernel (Basic RT) (PREEMPT_RTB) (NEW)
> 5. Fully Preemptible Kernel (RT) (PREEMPT_RT_FULL) (NEW
    )
```

Afterwards, we compiled the kernel. As this is a lengthy process, we set the multithreading option *-j* to the number of our CPU cores:

```
fakeroot make -j4 deb-pkg
```

Finally, we installed the newly created package by using *sudo dpkg -i* command. Restarting the system the Grub boot menu now allows us to choose our newly installed kernel. By seeing the output of the *uname -a* command we see it contains the string *PREEMPT_RT* and the version number we chose. Additionally, */sys/kernel/realtime* exists and contains the number 1.

After the *PREEMPT_RT* kernel was installed and running, we added a group named "realtime" and add the user controlling our robot to this group:

```
sudo addgroup realtime
sudo usermod -a -G realtime panda
```

Afterwards, we added the following limits to the realtime group in */etc/security/limits.conf*:

```
@realtime soft rtprio 99
@realtime soft priority 99
@realtime soft memlock 102400
@realtime hard rtprio 99
@realtime hard priority 99
@realtime hard memlock 102400
```

The limits were applied after we logged out and in again.

3.3 Building libfranka

To build libfranka, we installed the following dependencies from Ubuntu's package manager:

```
sudo apt install build-essential cmake git libpoco-dev
    libeigen3-dev
```

Then, we downloaded the source code by cloning libfranka from GitHub:

```
git clone --recursive https://github.com/frankaemika/
    libfranka
cd libfranka
```

which install by default the newest release of libfranka.

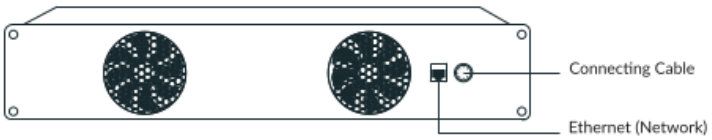


Figure 3.1: Control Ethernet port

3.4 Setting up the network

Good network performance is *crucial* when controlling the robot using FCI. This section describes how to configure the network.

In particular, our workstation was equipped with a double network interface: a first interface was used to connect the workstation to the laboratory wall socket through an Ethernet cable to allow access to the Internet while a second interface was used to directly connect the PC to the robot controller through another Ethernet cable.

On the second interface we configured the control and our workstation to appear on the same network. Simplest way to achieve that is to use static IP addresses (any two addresses on the same network works). With this network configuration, Desk can be accessed via the static IP address chosen (<*fci-ip*>), although the browser shows a certificate warning.

The configuration process consists of two steps:

- configuring control's network settings;
- configuring your workstation's network settings.

3.4.1 Control network configuration

The Control's network can be configured in Network section of the administrator's interface (see figure 2.10 on page 21). For the duration of this step we connected to the robot through the port in the robot's base. We pressed apply and after the settings were successfully applied, we connected our workstation's LAN port to the robot's control unit.

3.4.2 Linux workstation network configuration

To setup a static IP address on our Ubuntu system, we went to Network section in Setting widget using the GUI and clicked upon the gearwheel icon of the wired connection used to control the robot (second interface, enp3s0).

Next we clicked on the IPv4 settings tab, set the method to manual and entered: a static IP address (on the same subnet of the controller), a netmask, the gateway address and the DNS server address (for example 8.8.8.8, Google DNS). This step disables DHCP, which means it is no longer

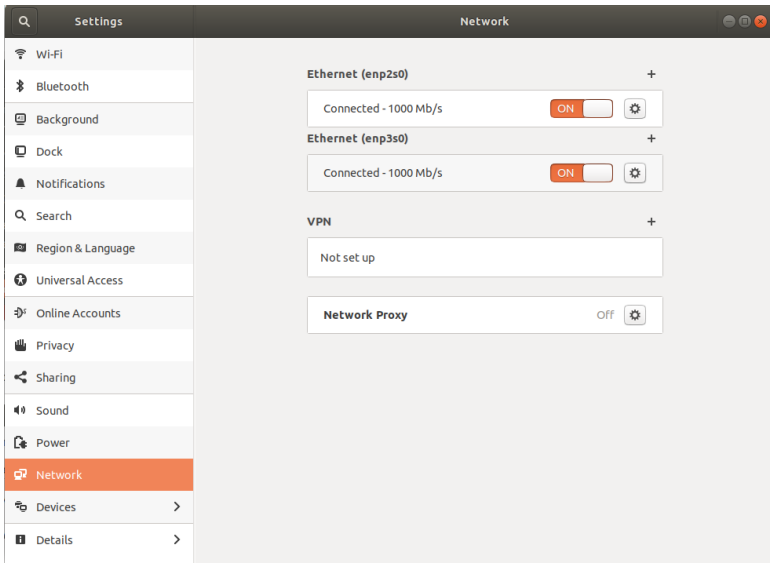


Figure 3.2: Edit the connection in the Network section

possible to obtain an address when connecting to a DHCP server, like the one in Arm's base. When no longer using FCI, we can change the Method back to Automatic (DHCP).

We saved the changes and closed the Network Connection window so that it was possible to connect to the robot from our workstation by clicking on the connection name from the drop down menu.

In order to allow th robot to access the Internet, for example to connecting to Franka World for automatic updates (see subsection 2.2.2), we decided to configure the first network interface to the same subnet as the second and the controller even though this is not necessary to control the robot. To do this, we setup routing on our workstation to allow the robot to access the internet through the first network interface.

First, packet forwarding needed to be enabled in Ubuntu's default firewall. So, in */etc/default/ufw* we changed the DEFAULT_FORWARD_POLICY to "ACCEPT":

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Then we edited */etc/ufw/sysctl.conf* and uncommented/added:

```
net/ipv4/ip_forward=1
net/ipv4/conf/all/forwarding=1
```

Moreover the robot had to be configured to use the IP address of our workstation (second net interface) as "gateway" and netmask, both on robot interface and workstation interface, had to be set as 255.255.255.252 since a more specific route to the robot is needed because we had the same network on both interfaces.

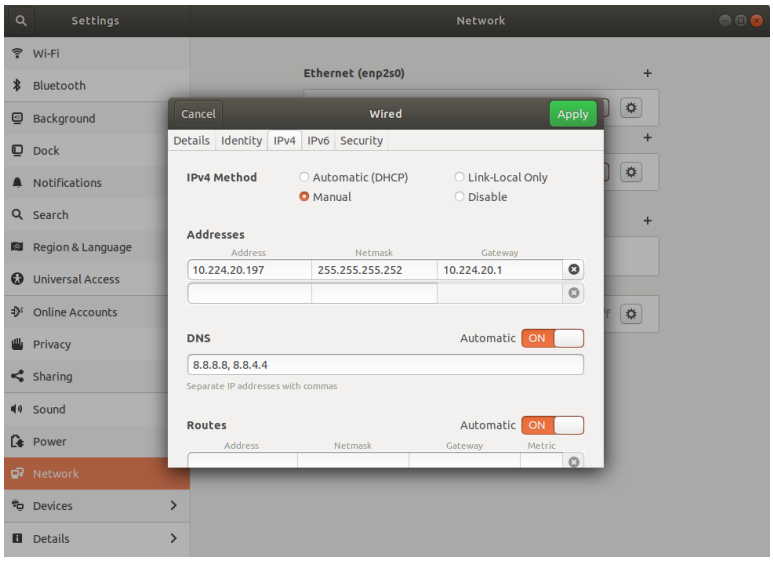


Figure 3.3: Setting a static IP for the Workstation PC

Finally, we used ARP proxying so that all the network knew about our routing setting. So we used the following command:

```
sudo su echo 1 >/proc/sys/net/ipv4/conf/enp2s0/proxy_arp
```

3.4.3 Verifying the connection

In order to verify that everything was correctly set up, we ran the *echo_robot_state* example from libfranka. We changed to the build directory of libfranka and executed the example:

```
examples/echo_robot_state <fci-ip>
```

The program prints the current state of the robot to the console and terminates after a few iterations.

To verify the connection to Franka World, instead, just go to Franka World section via App Desk and check that status is "online".

Chapter 4

Trajectory Planning of a Lissajous Figure

This chapter presents the planned trajectory so that the robot can polish the POF following a "8-shaped" path. In particular, the next sections shows the Lissajous figure considered for the path, the operational space trajectory planned and some simulation results.

4.1 Lissajous Figures

In order to polish the fiber as described in section 1.4, we considered a particular Lissajous figure.

Named after Jules Antoine Lissajous (French physicist, $1822 - 1880$), a *Lissajous Figure* is the graph of parametric equations

$$\begin{cases} x(t) = A_x \sin(\omega_x t + \phi_x) \\ y(t) = A_y \sin(\omega_y t + \phi_y) \end{cases} , \quad t \in [0, 2\pi]$$

The appearance of the figure is highly sensitive to the ratio ω_x/ω_y. For a ratio of 1, the figure is an ellipse, with special cases including circles $\big(A_x = A_y$, $\phi_x = -\pi/2$, $\phi_y = 0\big)$ and lines $\big(\phi_x = 0$, $\phi_y = 0\big)$. Another simple Lissajous figure is the parabola $\big(\omega_x/\omega_y = 2$ and $\phi_x = 0, \phi_y = 0\big)$. Other ratios produce more complicated curves, which are closed only if ω_x/ω_y is rational. The visual form of these curves is often suggestive of a three-dimensional knot, and indeed many kinds of knots, including those known as Lissajous knots, project to the plane as Lissajous figures.

Visually, the ratio ω_x/ω_y determines the number of "lobes" of the figure. For example, a ratio of 3/1 or 1/3 produces a figure with three major lobes. Similarly, a ratio of 5/4 produces a figure with five horizontal lobes and four vertical lobes. Rational ratios produce closed (connected) or "still" figures, while irrational ratios produce figures that appear to rotate.

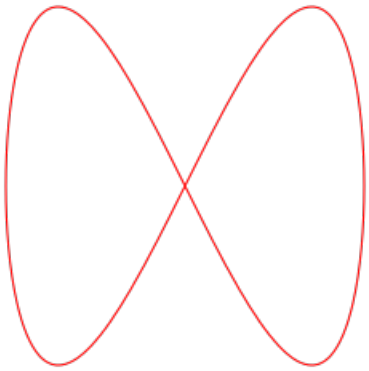


Figure 4.1: Lissajous figure considered in this thesis

The ratio A_x/A_y determines the relative width-to-height ratio of the curve. For example, a ratio of 2/1 produces a figure that is twice as wide as it is high. Finally, the difference $\phi_x - \phi_y$ determines the apparent "rotation" angle of the figure, viewed as if it were actually a three-dimensional curve. For example, $\phi_x - \phi_y = 0$ produces x and y components that are exactly in phase, so the resulting figure appears as an apparent three-dimensional figure viewed from straight on 0°. In contrast, any non-zero difference produces a figure that appears to be rotated, either as a left–right or an up–down rotation (depending on the ratio A_x/A_y), for example $\phi_x - \phi_y = -\pi/2$ produces oscillating motions between them in quadrature.

In particular, the Lissajous figure that best suits the polishing task and that we considered in this work, is the figure obtained with $\omega_x/\omega_y = 1/2$ and $\phi_x = \pi/2$, $\phi_y = 0$ (see figure 4.1). A_x and A_y can be chosen arbitrarily according to the dimensions of the figure that we want to get.

4.2 Operational Space Trajectory

In Robotics there are several methods to plan a trajectory, both in joint space and in operational space [23]. Since we wanted the robot end-effector to follow a path which is described by well-known parametric equations, we decided to plan the trajectory in the operational space by using a *path primitive*.

A parametric representation of a path Γ in space is

$$\boldsymbol{p} = \boldsymbol{f}(s)$$

The scalar $s \in [s_i, s_f]$ is called *curvilinear abscissa*, or simply *parameter* and represents the *arc length*. As s increases, the point $\boldsymbol{p}$ moves on the path Γ in a given direction. This direction is said to be the direction induced on Γ by the parametric representation.

Except for special cases, $\boldsymbol{p}$ allows the definition of three unit vectors characterizing the path. The orientation of such vectors depends exclusively on

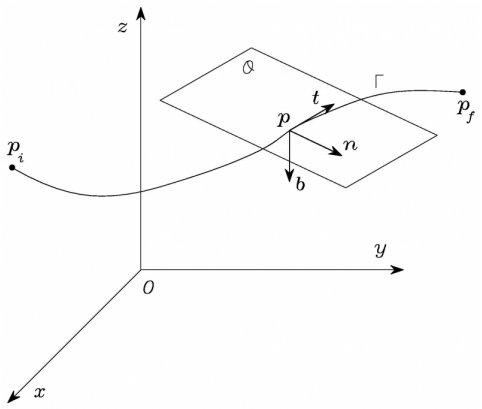


Figure 4.2: Parametric representation of a path in space

the path geometry, while their direction depends on the direction introduced by the parametric representation on the path.

The first of such unit vectors is the *tangent unit vector*

$$\boldsymbol{t} = \frac{d\boldsymbol{p}}{ds}$$

which is oriented along the direction induced on the path by s.

The second unit vector is the *normal unit vector*

$$\boldsymbol{n} = \frac{1}{\left\| \frac{d^2\boldsymbol{p}}{ds^2} \right\|} \frac{d^2\boldsymbol{p}}{ds^2}$$

which is oriented along the line intersecting $\boldsymbol{p}$ at a right angle with $\boldsymbol{t}$ and lies in the so-called *osculating plane* ϑ (see figure 4.2). This plane is the limit position of the plane containing the unit vector $\boldsymbol{t}$ and a point $\boldsymbol{p}' \in \Gamma$ when $\boldsymbol{p}'$ tends to $\boldsymbol{p}$ along the path. The direction of $\boldsymbol{n}$ is so that the path Γ, in the neighbourhood of $\boldsymbol{p}$ with respect to the plain containing $\boldsymbol{t}$ and normal to $\boldsymbol{n}$, lies on the same side of $\boldsymbol{n}$.

The third unit vector is the *binormal unit vector*

$$\boldsymbol{b} = \boldsymbol{t} \times \boldsymbol{n}$$

This vector is so that the frame ($\boldsymbol{t}$, $\boldsymbol{n}$, $\boldsymbol{b}$) is right handed (figure 4.2). Notice that it is not always possible to define uniquely such a frame.

In order to obtain a trajectory for the end-effector position, it is necessary to link a *time law* to the path described by the Lissajous figure. Therefore we have defined an interpolating polynomial $s(t)$ (since s is the independent variable of the path) which in $[0, t_f]$ goes from zero to the path length:

$s(t)$ interpolating polynomial so that $s(0) = 0$ and $s(t_f) = l(\Gamma) = L$

where t_f is the total trajectory duration. In this way we got the trajectory by solving a *point-to-point motion* problem in t_f seconds from 0 to L.

In particular, we solved this problem by using a *quintic polynomial* with the well known three coefficients $6, -15, 10$ relative to a normalized trajectory which goes from 0 to 1 in 1 second with null velocity and acceleration both in $t_0 = 0$ and $t_f = 1$. So, we have:

$$s(t) = s_0 + (s_f - s_0)\left(6\left(\frac{t}{t_f}\right)^5 - 15\left(\frac{t}{t_f}\right)^4 + 10\left(\frac{t}{t_f}\right)^3\right)$$

$$\dot{s}(t) = (s_f - s_0)\frac{1}{t_f}\left(30\left(\frac{t}{t_f}\right)^4 - 60\left(\frac{t}{t_f}\right)^3 + 30\left(\frac{t}{t_f}\right)^2\right)$$

$$\ddot{s}(t) = (s_f - s_0)\frac{1}{t_f^2}\left(120\left(\frac{t}{t_f}\right)^3 - 180\left(\frac{t}{t_f}\right)^2 + 60\left(\frac{t}{t_f}\right)\right)$$

With these preliminary considerations, we considered the Lissajous figure described by the following parametric equations and depicted in figure 4.1 on page 36:

$$\begin{cases} x(t) = A_x \sin(\omega_x t + \phi_x), & \text{with } \omega_x = 1\, rad/s \text{ and } \phi_x = \frac{\pi}{2} \\ y(t) = A_y \sin(\omega_y t + \phi_y), & \text{with } \omega_y = 2\, rad/s \text{ and } \phi_y = 0 \end{cases} \quad , \quad t \in [0, 2\pi]$$

In particular, the parametric representation of this Lissajous figure in the frame $O' - x'y'z'$ where O' coincides with the center of the Lissajous figure, is:

$$\boldsymbol{p}'_e(s) = \begin{bmatrix} A_x \sin\left(\omega_x \frac{2\pi k}{L} s + \frac{\pi}{2}\right) \\ A_y \sin\left(\omega_y \frac{2\pi k}{L} s\right) \\ 0 \end{bmatrix}, \quad s \in [0, L]$$

Therefore $\boldsymbol{p}'_e(s) \in \mathbb{R}^3$ is the vector of operational space variables expressing the position of the manipulator's end-effector w.r.t. end-effector frame and L is the total path length (k times the length of the Lissajous figure; so k is the laps number during polishing task).

If $\boldsymbol{c}$ is the vector describing the origin of end-effector frame w.r.t. robot base frame $O - xyz$, and $\boldsymbol{R}$ the rotation matrix of end-effector frame w.r.t. robot base frame, the end-effector position w.r.t. robot base frame is:

$$\boldsymbol{p}_e(s) = \boldsymbol{c} + \boldsymbol{R}\boldsymbol{p}'_e(s)$$

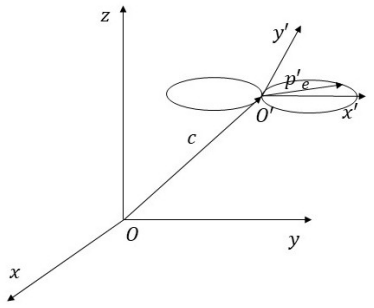


Figure 4.3: Parametric representation of the Lissajous figure in space

while the tangent vector is:

$$\boldsymbol{t} = \frac{d\boldsymbol{p}_e}{ds} = \boldsymbol{R} \begin{bmatrix} A_x\omega_x \frac{2\pi k}{L} \cos\left(\omega_x \frac{2\pi k}{L} s + \frac{\pi}{2}\right) \\ A_y\omega_y \frac{2\pi k}{L} \cos\left(\omega_y \frac{2\pi k}{L} s\right) \\ 0 \end{bmatrix}, \quad s \in [0, L]$$

Instead, the robot end-effector linear velocity w.r.t. robot base frame is:

$$\dot{\boldsymbol{p}}_e = \dot{s}\boldsymbol{t} = \dot{s}\boldsymbol{R} \begin{bmatrix} A_x\omega_x \frac{2\pi k}{L} \cos\left(\omega_x \frac{2\pi k}{L} s + \frac{\pi}{2}\right) \\ A_y\omega_y \frac{2\pi k}{L} \cos\left(\omega_y \frac{2\pi k}{L} s\right) \\ 0 \end{bmatrix}, \quad s \in [0, L]$$

Finally, we considered the orientation trajectory which is trivial since we wanted the robot not to rotate during polishing. So we have:

$$\begin{aligned} \boldsymbol{\phi}_e(t) = \boldsymbol{\phi}_e(0) = \boldsymbol{\phi}_0 \\ \dot{\boldsymbol{\phi}}_e = \mathbf{0} \end{aligned} \quad , \quad t \in [0, 2\pi]$$

Eventually, the output of the operational space trajectory planner is:

$$\boldsymbol{x}_{e_d}(t) = \begin{bmatrix} \boldsymbol{p}_e(s(t)) \\ \boldsymbol{\phi}_0 \end{bmatrix}, \quad \boldsymbol{x}_{e_d} \in \mathbb{R}^6$$

$$\dot{\boldsymbol{x}}_{e_d}(t) = \begin{bmatrix} \dot{\boldsymbol{p}}_e(s(t)) \\ \mathbf{0} \end{bmatrix}, \quad \dot{\boldsymbol{x}}_{e_d} \in \mathbb{R}^6$$

Moreover, by using a quintic polynomial as shown before, we guaranteed both end-effector velocity and acceleration to be zero in $t = 0$ and $t = t_f$.

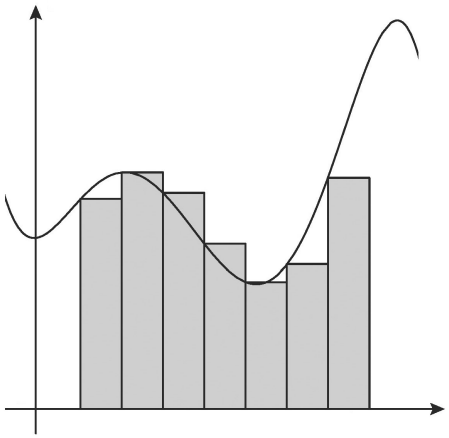


Figure 4.4: Forward Euler method

To ultimate this trajectory planning, we only had to compute the Lissajous figure length. By using the formula for calculating the length of a plane curve expressed through its parametric equations, we have:

$$L = \int_0^{2k\pi} \sqrt{x'(t)^2 + y'(t)^2}\, dt, \quad \text{where } k \text{ is the number of laps}$$

which is not simple to compute analytically. For this reason, we computed it approximately by observing that a possible numeric integral is:

$$\int_a^b f(t)\, dt \simeq [(1-\alpha)f(a) + \alpha f(b)](b-a), \quad \alpha \in [0,1]$$

which is the more exact the more the function $f(t)$ is slowly variable in the interval of integration. In particular we used the forward Euler method (*left Riemann sum*, see figure 4.4) by choosing $\alpha = 0$ and we divided the interval of integration in Nk small intervals (where N can be chosen arbitrarily) in order to have a more accurate estimation of the path length. Thus, we have:

$$\int_0^{2k\pi} f(t)\, dt \simeq \sum_{h=0}^{Nk-1} f(t_h)\frac{2\pi}{N}, \quad \text{with } f(t) = \sqrt{x'(t)^2 + y'(t)^2}$$

in which we divided the interval of integration $[0, 2k\pi]$ in Nk small intervals with a $2\pi/N$ step size so that $t_0 = 0$ and $t_{Nk-1} = 2k\pi - 2\pi/N$.

4.3 Planner Simulation

In order to verify that the output of the operational space trajectory planner was correct, we implemented the trajectory planner in Simulink by using the scheme shown in figure 4.5 on the following page.

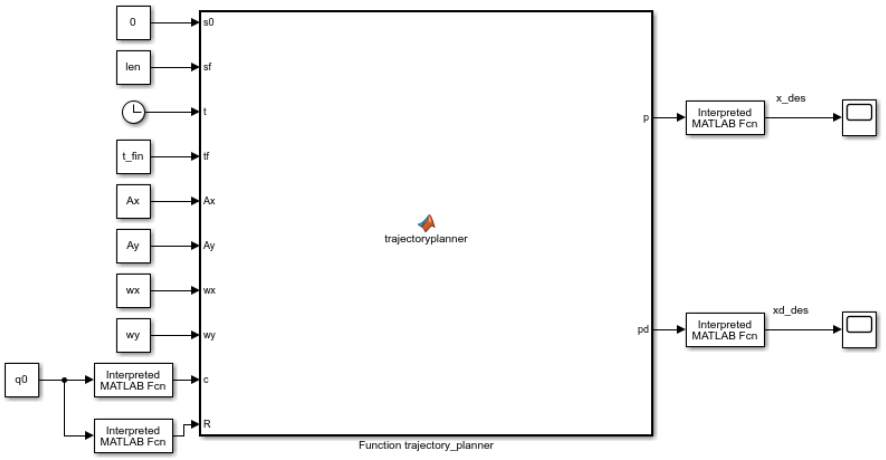


Figure 4.5: Trajectory planner Simulink scheme

The Matlab function implements the trajectory as explained before. In particular: the variable "len" is the total path length which can be computed as explained before, "t_fin", chosen equal to $60\,s$, is the total trajectory duration, "q0" is the initial manipulator configuration chosen arbitrarily (in this simulation the end-effector position was approximately equal to $[0.57, 0, 0.47]^T$) while "Ax" and "Ay", which represent the Lissajous figure amplitude in x and y directions respectively, were chosen equal to $15\,cm$ and $5\,cm$.

The trajectory planner output, relative to three laps, is shown in figure 4.6 on the next page. It is important to highlight that we multiplied both x and y components of the parametric representation of the Lissajous figure by a 2° order system step-response in order to avoid discontinuities in $t = 0$ (the end effector must start from its current position). So if

$$f(t) = 1 - \frac{te^{-t/\tau}}{\tau} - e^{-t/\tau}$$

is the 2° order system step-response with time constant $\tau = 0.5$ (so to have a settling time of $\simeq 2.5\,s$), we have:

$$\boldsymbol{p}'_e(s) = \begin{bmatrix} f(t)A_x \sin\left(\omega_x \frac{2\pi k}{L} s + \frac{\pi}{2}\right) \\ f(t)A_y \sin\left(\omega_y \frac{2\pi k}{L} s\right) \\ 0 \end{bmatrix}$$

which must be expressed in the base frame, while its time derivative expressed

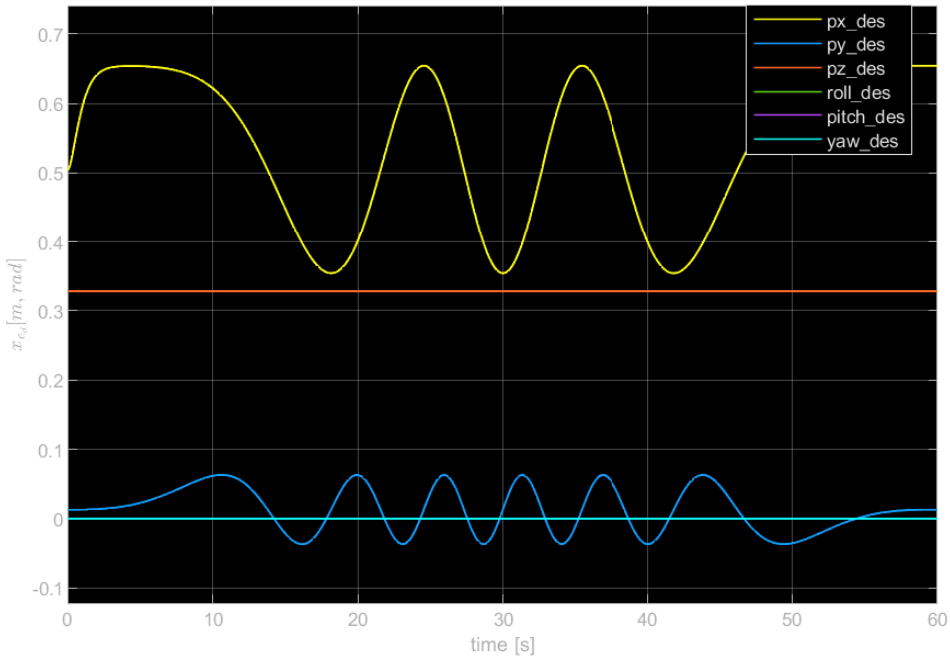


Figure 4.6: Trajectory planner pose output

in the robot base frame became:

$$\dot{\boldsymbol{p}}_e = \begin{bmatrix} \dot{f}(t)A_x \sin\left(\omega_x \frac{2\pi k}{L} s + \frac{\pi}{2}\right) + f(t)A_x\omega_x \frac{2\pi k}{L} \cos\left(\omega_x \frac{2\pi k}{L} s + \frac{\pi}{2}\right) \\ \dot{f}(t))A_y \sin\left(\omega_y \frac{2\pi k}{L} s\right) + f(t)A_y\omega_y \frac{2\pi k}{L} \cos\left(\omega_y \frac{2\pi k}{L} s\right) \\ 0 \end{bmatrix}$$

so that to have $\dot{\boldsymbol{p}}_e$ exactly equal to $\dfrac{d\boldsymbol{p}(s(t))}{dt}$ and where

$$\dot{f}(t) = \frac{df}{dt} = \frac{te^{-t/\tau}}{\tau^2}$$

Figure 4.7 on the following page shows the y component of the trajectory planner position output respect to the x component in order to have the Lissajous figure in the xy plane. The trajectory planner output corresponds to the desired Lissajous figure with the avoidance of discontinuities represented by the horizontal line which starts from the center of the figure (end effector starts from the center and smoothly describes the figure). Moreover, the three figures are perfectly overlapped.

Finally, figure 4.8 on the next page shows how the x component of the end-effector velocity has a peak at the beginning of the motion since, in order to avoid discontinuities, the manipulator has to go from the center to the right end of the Lissajous figure according to the $f(t)$ dynamics.

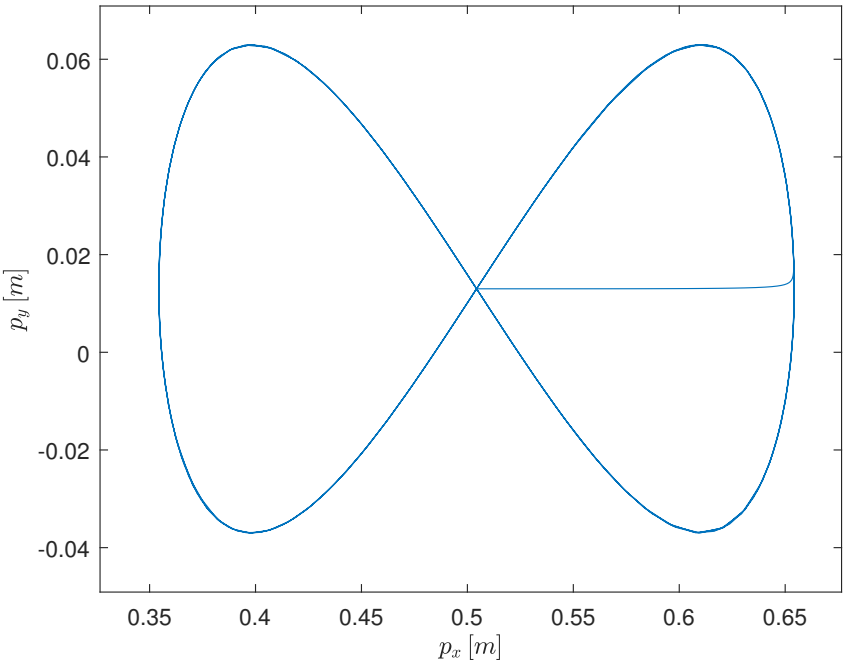


Figure 4.7: Lissajous figure from the trajectory planner

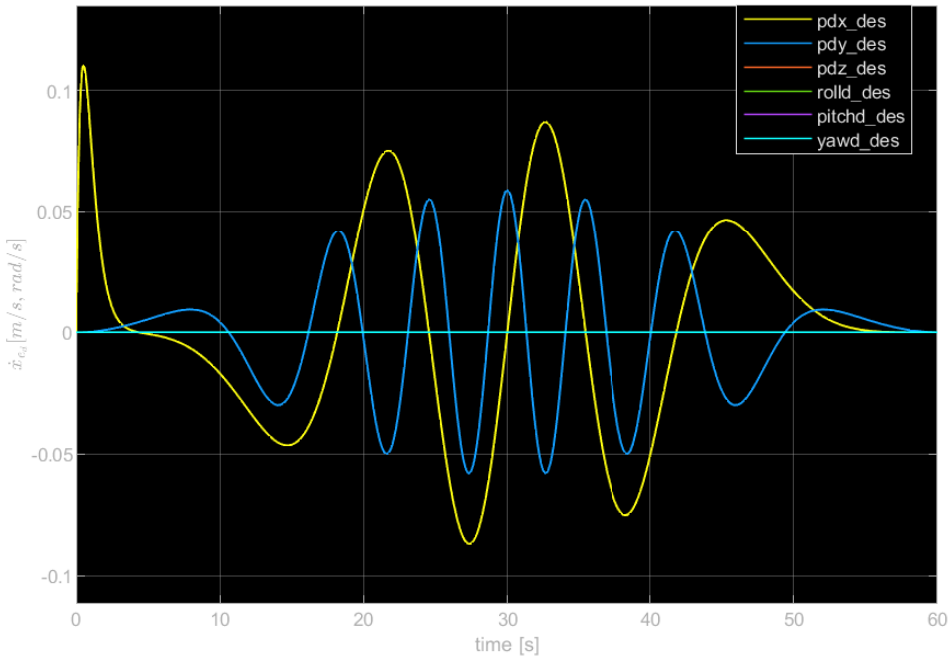


Figure 4.8: Trajectory planner velocity output

Chapter 5

Force/Compliance Control Algorithm

This chapter presents the torque control algorithm implemented on the robot in order to execute the polishing task. It is part of the category of force control algorithms in which the robot is no longer considered only to move the end effector in space but to physically interact with the environment in which it operates to perform a certain task, as it happens in advanced robotics. In particular, it is an *direct* force control in which the goal is to achieve a desired value for the contact force and not just to keep it limited (*indirect* force control).

5.1 Controller Design

The Force/Compliance control algorithm proposed in this work consists of an hybrid force control which provides the robot an intricate coordination of contact force and motion generation so that the manipulator can follow the path described by the Lissajous figure and resolve a force regulation problem simultaneously [21]. First we summarise the dynamic model of flexible joint robots and then we present the designed Cartesian compliance control and force control.

5.1.1 Flexible Joint Dynamics

The reduced flexible joint model (*Spong model* [24]) is described by the equations

$$\begin{aligned} \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) - \boldsymbol{K}_J(\boldsymbol{\theta} - \boldsymbol{q}) &= \boldsymbol{\tau}_{ext} \\ \boldsymbol{B}\ddot{\boldsymbol{\theta}} + \boldsymbol{K}_J(\boldsymbol{\theta} - \boldsymbol{q}) &= \boldsymbol{\tau}_m \end{aligned}$$

where $\boldsymbol{\theta} \in \mathbb{R}^7$ is the joint-side angular position upstream of the elastic transmission while $\boldsymbol{q} \in \mathbb{R}^7$ is the joint-side angular position downstream of

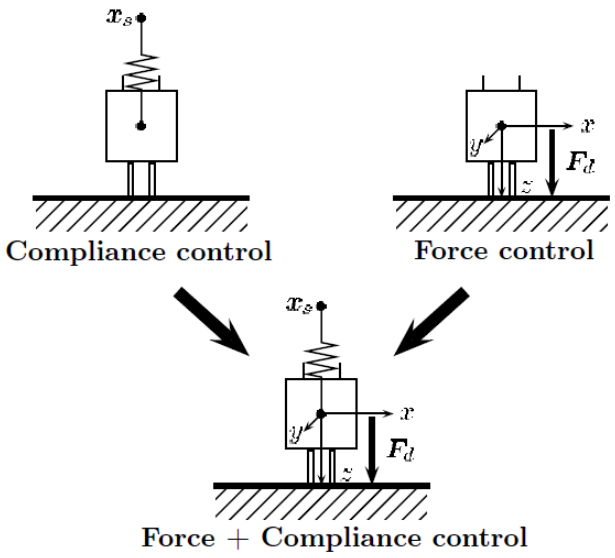


Figure 5.1: Conceptual schematic of the proposed controller

the elastic transmission.

These two equations, representing the link and motor side dynamics respectively, are coupled via the elastic joint torque $\boldsymbol{K}_J(\boldsymbol{\theta} - \boldsymbol{q})$ which is considered to have linear spring-like characteristics and which transfers the torque from motors to links.The matrices $\boldsymbol{K}_J \in \mathbb{R}^{7\times7}$ and $\boldsymbol{B} \in \mathbb{R}^{7\times7}$ are both constant diagonal positive definite matrices, expressing the lumped joint stiffness and motor inertia, respectively. Neither motor nor link side friction are considered, as their are actually compensated for on the real robot.

5.1.2 Flexible Joint Cartesian Compliance Control

We realized compliance control for the flexible joint case [21] such that the stability is preserved. This was achieved by making the position feedback to be a function of $\boldsymbol{\theta}$ only instead of both $\boldsymbol{\theta}$ and $\boldsymbol{q}$. For this, $\boldsymbol{q}$ is replaced with its static equivalent $\tilde{\boldsymbol{q}}(\boldsymbol{\theta}) = \boldsymbol{\zeta}^{-1}(\boldsymbol{\theta})$, which is numerically obtained by a contraction mapping with the implicit function $\boldsymbol{\zeta}(\boldsymbol{q}) = \boldsymbol{q}_e + \boldsymbol{K}_J^{-1}\boldsymbol{g}(\boldsymbol{q}_e)$, where $\boldsymbol{q}_e$ is the joint position at the equilibrium point. Under mild assumptions $\tilde{\boldsymbol{q}}(\boldsymbol{\theta})$ can be used as an estimate for $\boldsymbol{q}$.

The compliance law for the flexible joint robot can then be formulated as:

$$
\begin{aligned}
\boldsymbol{\tau}_{mc} &= \boldsymbol{J}_A^T(\tilde{\boldsymbol{q}})\Big(\boldsymbol{K}_p\tilde{\boldsymbol{x}} + \boldsymbol{D}_x\dot{\tilde{\boldsymbol{x}}}\Big) \\
\tilde{\boldsymbol{x}}(\boldsymbol{\theta}) &= \tilde{\boldsymbol{x}}\big(\tilde{\boldsymbol{q}}(\boldsymbol{\theta})\big) = \boldsymbol{x}_d - \boldsymbol{f}\big(\tilde{\boldsymbol{q}}(\boldsymbol{\theta})\big) = \boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{\theta}) \\
\dot{\tilde{\boldsymbol{x}}}(\boldsymbol{\theta}) &= \dot{\boldsymbol{x}}_d - \boldsymbol{J}_A(\tilde{\boldsymbol{q}})\dot{\boldsymbol{\theta}}
\end{aligned}
$$

For simplicity, we assumed $\dot{\boldsymbol{x}}(\boldsymbol{\theta}) = \boldsymbol{J}_A(\tilde{\boldsymbol{q}})\dot{\boldsymbol{\theta}} \approx \boldsymbol{J}_A(\boldsymbol{q})\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}$. The mapping $\boldsymbol{f}\colon \mathbb{R}^7 \to \mathbb{R}^6$ encodes the forward kinematics while $\boldsymbol{x}_d \in \mathbb{R}^6$ denotes the compliance set-point. The third equation relates joint space velocities to Cartesian space velocities by the task analytic Jacobian $\boldsymbol{J}_A$.

The matrices $\boldsymbol{K}_p \in \mathbb{R}^{6\times6}$ and $\boldsymbol{D}_x \in \mathbb{R}^{6\times6}$ are positive definite matrices for stiffness and damping, respectively. $\boldsymbol{D}_x$ can be chosen constant, often diagonal.

5.1.3 PI Controller: Force Regulation

In our controller design we started from the following Cartesian force tracking controller [21]

$$\boldsymbol{\tau}_{mf} = \boldsymbol{J}_A^T(\tilde{\boldsymbol{q}})\left[\boldsymbol{K}_{p_f}\Big(\boldsymbol{F}_d(t) - \boldsymbol{F}_{ext}(t)\Big) + \boldsymbol{K}_i\int_0^t \Big(\boldsymbol{F}_d(\sigma) - \boldsymbol{F}_{ext}(\sigma)\Big)\,d\sigma\right]$$

where $\boldsymbol{K}_{p_f} \in \mathbb{R}^{6\times6}$ and $\boldsymbol{K}_i \in \mathbb{R}^{6\times6}$ are diagonal positive definite matrices for proportional and integral control part respectively.

The desired wrench $\boldsymbol{F}_d(t) = \Big[\boldsymbol{f}_d^T(t), \boldsymbol{\tau}_d^T(t)\Big]^T \in \mathbb{R}^6$ is specified by the user or generated by a planner in order to apply a desired force/torque on the environment.

5.1.4 Unified Cartesian Force/Compliance Control Design

Combining the compliance and force controller for the flexible joint case via $\boldsymbol{\tau}_m = \boldsymbol{\tau}_{mc} + \boldsymbol{\tau}_{mf}$ we have the overall controller [21]

$$\boldsymbol{\tau}_m = \boldsymbol{J}_A^T(\boldsymbol{q})\left[\boldsymbol{K}_{p_f}\Big(\boldsymbol{F}_d(t)-\boldsymbol{F}_{ext}(t)\Big)+\boldsymbol{K}_i\int_0^t \Big(\boldsymbol{F}_d(\sigma)-\boldsymbol{F}_{ext}(\sigma)\Big)\,d\sigma+\boldsymbol{K}_p\tilde{\boldsymbol{x}}+\boldsymbol{D}_x\dot{\tilde{\boldsymbol{x}}}\right]$$

in which the compensation action of gravitational terms $\boldsymbol{g}(\boldsymbol{q})$ is implicit. In order to avoid the generation of *internal motions* in a redundant manipulator, if present, an additive damping term in joint space $-\boldsymbol{K}_D\dot{\boldsymbol{q}}$ can be considered.

In our task, since at steady-state conditions a normal constant force must be applied to the unknown surface while perpendicular motions follow the Lissajous figure, the desired wrench vector is $\boldsymbol{F}_d(t) = \Big[\boldsymbol{f}_d^T(t), \boldsymbol{0}^T\Big]^T$ where $\boldsymbol{f}_d^T(t) = \begin{bmatrix}0 & 0 & f_{z_d}(t)\end{bmatrix}$.

In particular, in order to avoid large overshoots and obtain a smooth behaviour, we chose $f_{z_d}(t) = f(t)f_{z_d}$ where $f(t)$ is the 2° order system step-response (as shown in section 4.3) and f_{z_d} is the desired normal constant force.

During this work, we didn't plan a trajectory in orientation since we wanted $\boldsymbol{\phi}_e(t) = \boldsymbol{\phi}_e(0) = \boldsymbol{\phi}_0$ for our task and we decided to represent the end-effector orientation with a *minimal representation* through Euler angles

XYZ. Therefore if end-effector orientation is $\boldsymbol{\phi} = [\varphi \quad \theta \quad \psi]^T$, the end-effector pose is:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\phi} \end{bmatrix}, \quad \boldsymbol{x} \in \mathbb{R}^6$$

Moreover, in order to avoid representation singularities ($\theta = \pm\pi/2$ for XYZ Euler angles), we decided to rotate the robot end-effector frame of $180\,deg$ around the y axis of the end-effector frame so as to have the possibility to align the end-effector frame with the robot base frame ($\boldsymbol{\phi} = \boldsymbol{0}$). So we defined:

$$T_d^{EE} = T_{EE}^F \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this way, by choosing properly the manipulator initial configuration $\boldsymbol{q}_0$, we could align the two frames and compute the analytic Jacobian, according to the XYZ angles, as follows

$$\boldsymbol{J}_A(\boldsymbol{q}) = \boldsymbol{T}_A^{-1}(\boldsymbol{\phi})\boldsymbol{J}(\boldsymbol{q})$$

with

$$\boldsymbol{T}_A^{-1}(\boldsymbol{\phi}) = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{T}^{-1}(\boldsymbol{\phi}) \end{bmatrix}$$

and

$$\boldsymbol{T}^{-1}(\boldsymbol{\phi}) = \frac{1}{c_\theta} \begin{bmatrix} 1 & s_\varphi s_\theta & -c_\varphi s_\theta \\ 0 & c_\varphi c_\theta & s_\varphi c_\theta \\ 0 & -s_\phi & c_\phi \end{bmatrix}$$

This means that $\boldsymbol{T}_A^{-1}(\boldsymbol{\phi})$ is always well defined during the trajectory since for XYZ Euler angles we have that $\boldsymbol{T}_A^{-1}(\boldsymbol{0}) = \boldsymbol{I}$.

In the control law shown above, $\boldsymbol{x}_d$ and $\dot{\boldsymbol{x}}_d$ are computed by the operational space trajectory planner (see section 4.2) while we chose $\boldsymbol{K}_p$ and $\boldsymbol{D}_x$ as

$$\boldsymbol{K}_p = \text{diag}\{1000, 1000, 500, 1000/20, 1000/20, 1000/20\}\,[N/m,\ Nm/rad]$$

and

$$\boldsymbol{D}_x = \text{diag}\{\sqrt{1000}, \sqrt{1000}, \sqrt{500}, \sqrt{1000/20}, \sqrt{1000/20}, \sqrt{1000/20}\}\,[Ns/m,\ Nms/rad]$$

It is important to highlight that we chose the z linear component both in $\boldsymbol{K}_p$ and $\boldsymbol{D}_x$ lower than the x and y components since along the z direction the compliance control can be considered as a disturbance for the force control.

As regards the PI controller, we designed the $\boldsymbol{K}_{p_f}$ and $\boldsymbol{K}_i$ matrices in MATLAB by using *sisotool* after creating in Simulink a linearised model of our open-loop system by using the *Model Linearizer* app. So we built

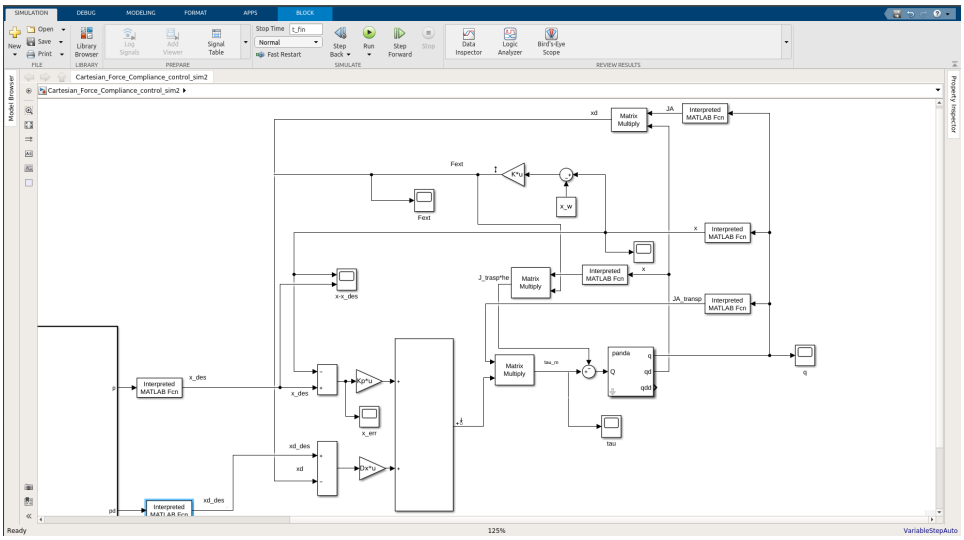


Figure 5.2: Simulink scheme for PI controller design

the rigid joints dynamic model of Panda [12] and we modelled the external wrench through an elastic model of the environment that exerts an elastic reaction force exclusively along the z direction following a one-way point contact between the robot and an elastic plane: $\boldsymbol{F}_{ext}(t) = \boldsymbol{K}\big(\boldsymbol{x} - \boldsymbol{x}_w\big)$ where $\boldsymbol{x}_w$ denotes the equilibrium pose between the elastic plane and the robot while $\boldsymbol{K} \in \mathbb{R}^6$ is the *environment stiffness matrix* which has just one non-zero component (z linear component) as well as $\boldsymbol{K}_{p_f}$ and $\boldsymbol{K}_i$.

In our Simulink scheme (see figure 5.2) we computed the external toques as $\boldsymbol{\tau}_{ext} = -\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{F}_{ext}$ and we considered the third component of the external wrench as the output of our linearised model and the entry point of the PI action as input in order to have a SISO to design the force controller. In particular, in order to have PI gains independent from the environment stiffness, we chose a unitary stiffness so as to adapt gains to the real environment stiffness by dividing $\boldsymbol{K}_{p_f}$ and $\boldsymbol{K}_i$ matrices by the environment stiffness along the z direction.

We designed the PI controller gains so as to obtain a good closed-loop system step-response in terms of overshoot and settling time. By choosing integral and proportional gains respectively as $k_{i_z} = 5000\, s^{-1}$ and $k_{p_{f_z}} = 2500$ we obtained the response shown in figure 5.3 on the next page, which seemed suitable for our task even though we knew very well that considering the complete non-linear robot model the response would have been quite different.

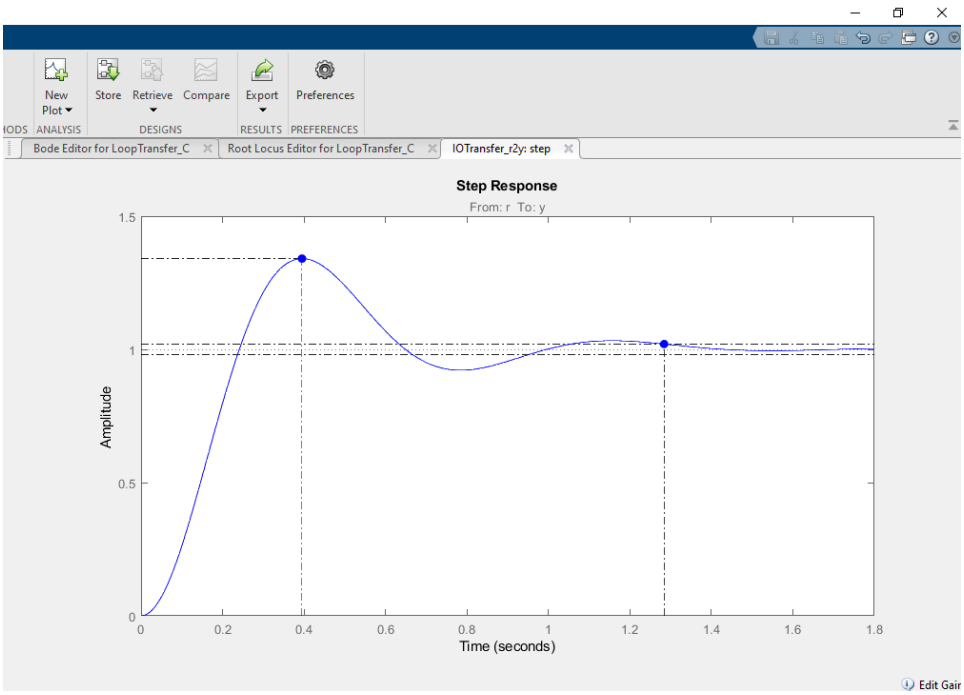


Figure 5.3: Closed-loop system step response

5.2 Controller Simulation

We tested in Simulink the force/compliance control algorithm before the implementation upon the real robot and realized the Simulink scheme shown in figure 5.4 on the following page in which we implemented the control law in a Matlab function (except for the integral action which is realized by an integrator block), as well as done for the trajectory planner, and considered a polishing task of $k = 4$ laps with a total duration of $80\,s$.

Above all we verified that the manipulator followed the Lissajous figure avoiding discontinuities in $t = 0$. As shown in figure 5.5 on the next page, we have a good tracking of the desired trajectory considering our gains in position and the fact that compliance control is not be formulated to solve tracking problems but rather regulation ones.

In figure 5.7 on page 51 we can see that there is a good overlapping between the desired Lissajous figure and that traced by the manipulator. We considered this result quite satisfactory considering that a high positional accuracy is not required, while compliance with respect to the environment is much more important due to the presence of humans in the workspace of the robot; moreover, a human operator would never be able to trace a perfect Lissajous figure many times during the polishing task.

Moreover, as shown in figure 5.6 on page 51, joint torques are quite smooth with respect to the Panda robot limits described in section 2.1.1.

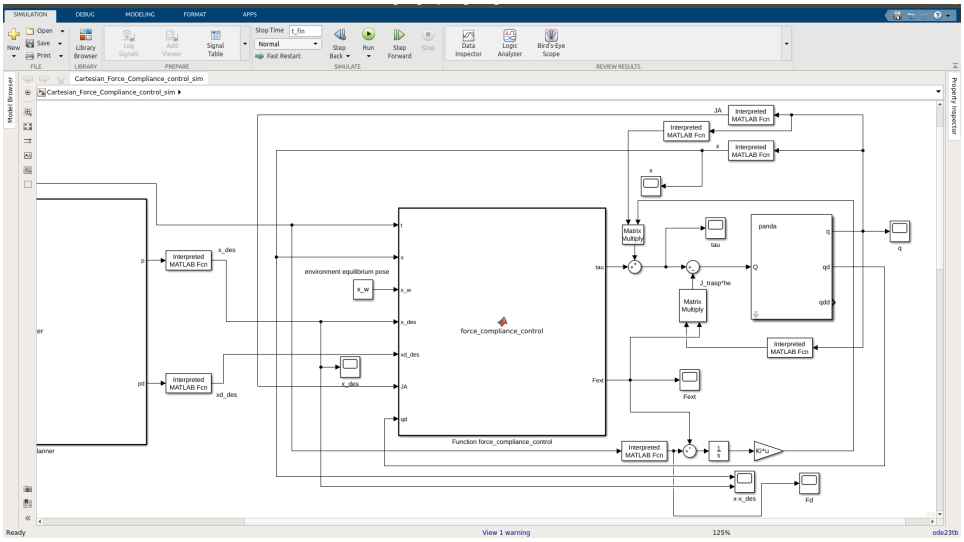


Figure 5.4: Control algorithm Simulink scheme

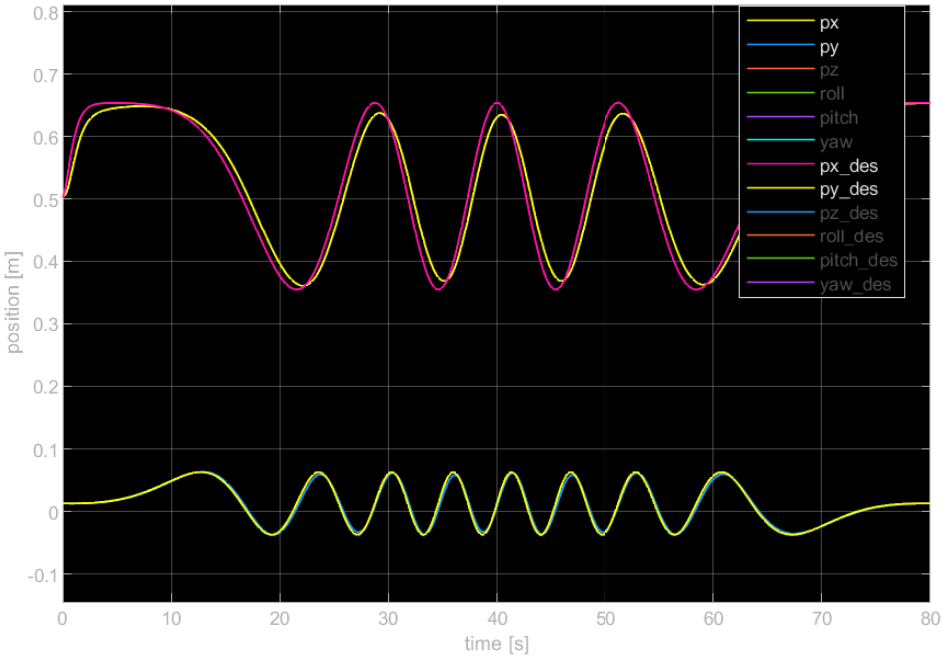


Figure 5.5: Tracking of the desired trajectory

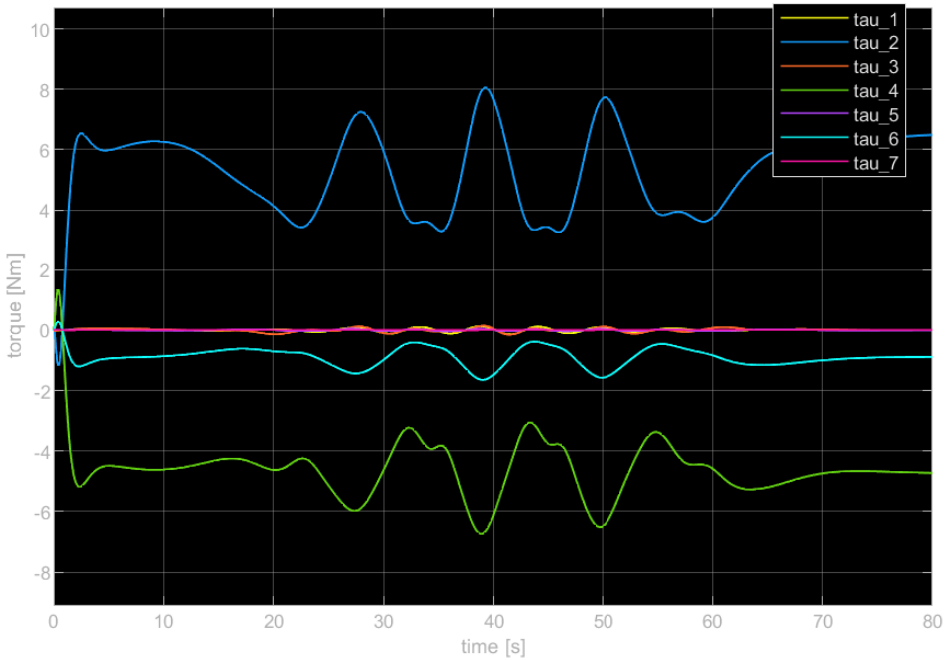


Figure 5.6: Joint torques

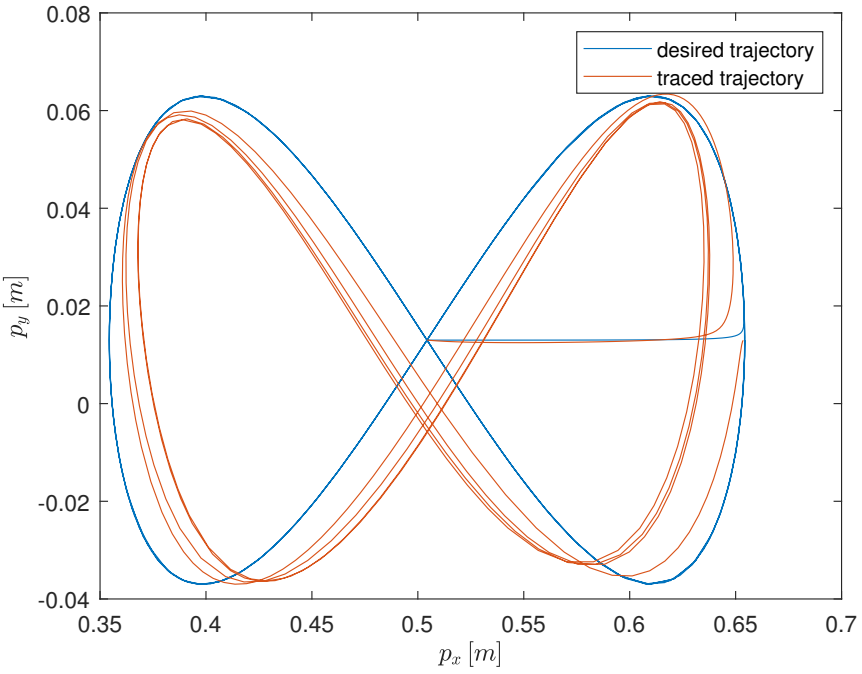


Figure 5.7: Comparison between desired and traced Lissajous figure

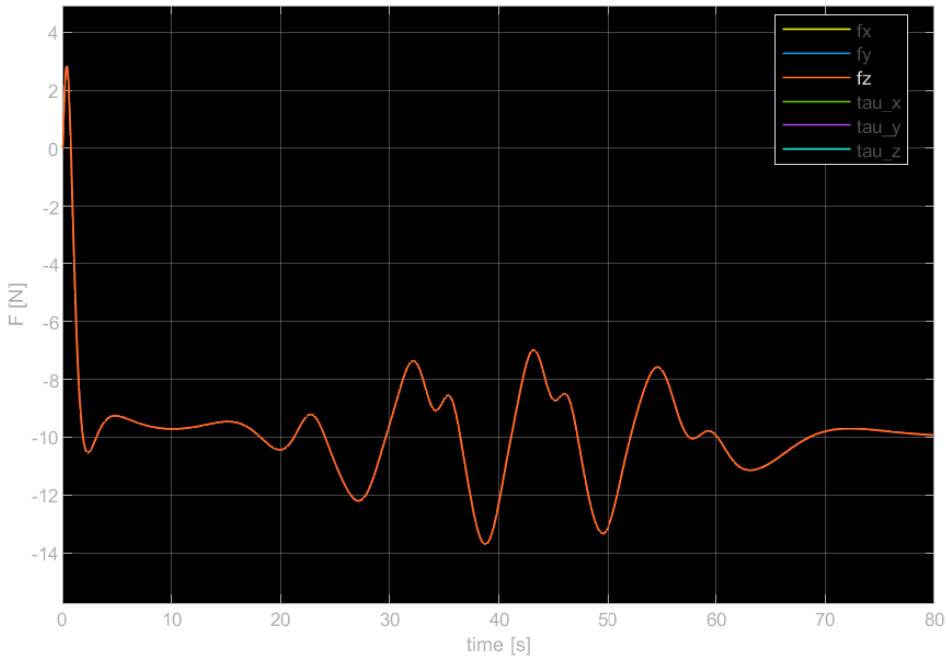


Figure 5.8: Force regulation

Finally in figure 5.8 we can see that the force applied by the end effector upon the environment reaches the desired value (here equal to $-10\,N$) after a transient of about $5\,s$ even though the effect of the external torques is evident as well as an initial overshoot due to the fact the we used a rigid robot model and not an elastic one.

5.3 Controller Implementation using libfranka

After testing the control algorithm in simulation, we proceeded by implementing it on the real robot. In particular, we used the external controller real-time interface (see figure 2.16 on page 25) in order to control the robot by sending directly the torque signals to the actuators (low-level control strategy).

We implemented the control algorithm by using libfranka in the same way we did it in MATLAB&Simulink. For the integral action in the force control we computed the external wrench error integral by using a numeric method in the same way we did for the Lissajous figure length in section 4.2.

The libfranka library provides a lot of functions which allow to get robot and gripper state as well as dynamic mode; for example, among others, we used *zeroJacobian* function to compute the geometric Jacobian in the base frame (there are not functions to compute the analytic Jacobian directly) and

O_T_EE function in order to obtain end-effector pose in the base frame.

We found particularly useful the *O_F_ext_hat_K* function which provides the estimated external wrench (force, torque) acting on the stiffness frame, expressed relative to the base frame (where the base frame coincides with the zero frame while the stiffness frame is set by default to coincide with the end-effector frame) since force control algorithms would request exteroceptive sensors in terms of hardware components. Nevertheless, we had to face the problem to remove the offset from the estimated external wrench since, in a general configuration, it is not null even though there are no contact between the robot and the environment (see figures 5.9 on the following page and 5.10 on the next page which show also an initial transient during the measurement) due to the bias affecting the joint torque sensors.

For this reason we implemented an additional control loop in which, after the robot goes to the initial configuration to execute the task, zero torques are sent to the robot (gravity&friction are compensated in the real-time controllers interface) for $4.5\,s$. After waiting the transient is passed, in the last $0.5\,s$ (500 samples) the robot computes the external wrench offset to be subtracted to the measurement during the polishing. Figures 5.11 on page 55 and 5.12 on page 55 show the estimate external wrench after offset removal when there are no contacts between the robot and the surrounding environment.

Finally, we also implemented subprograms in order to save in RAM buffers the variables of interest during the control loop and subprograms to write these buffers into ASCII files after the control loop is finished (to avoid to overrun the real-time deadline of the sampling time of $1\,ms$). Then, we used these files in MATLAB for computations or plotting (as done for the estimated external wrench).

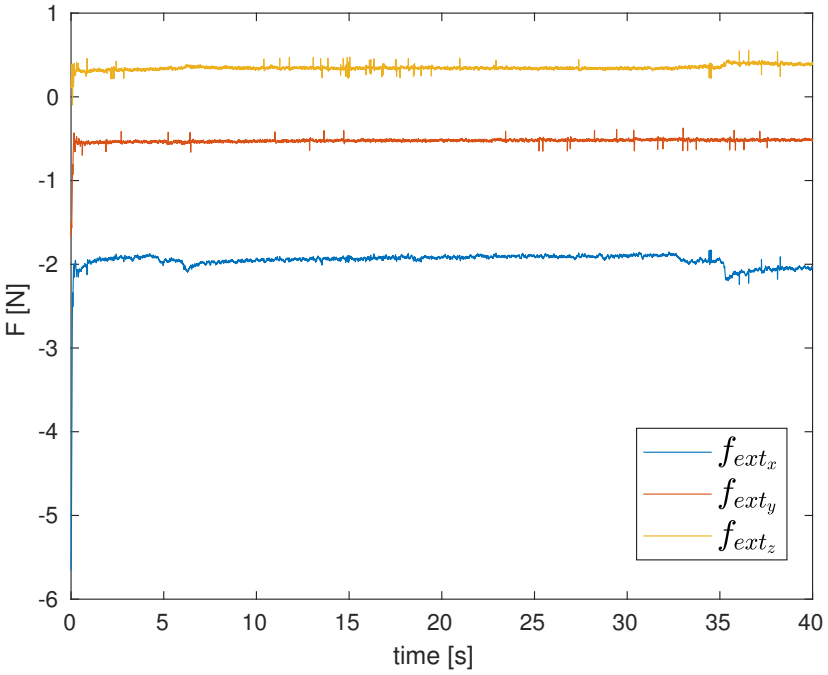


Figure 5.9: Estimated external forces by libfranka

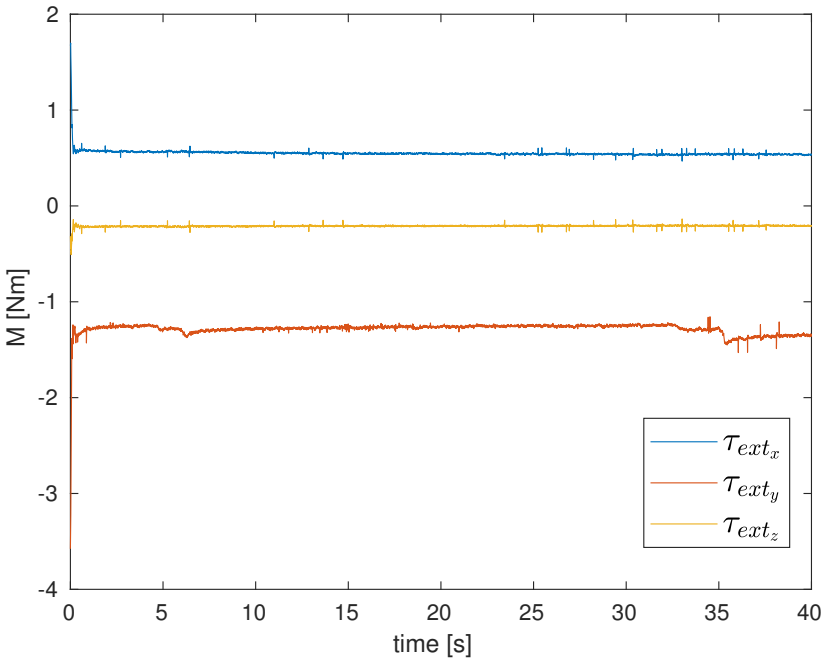


Figure 5.10: Estimated external moments by libfranka

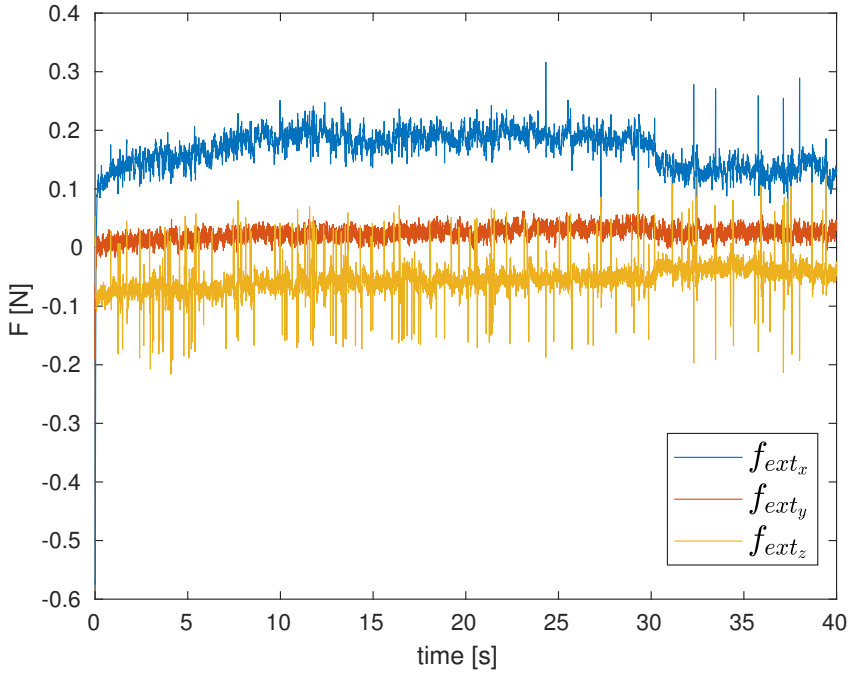


Figure 5.11: Estimated external forces without offset

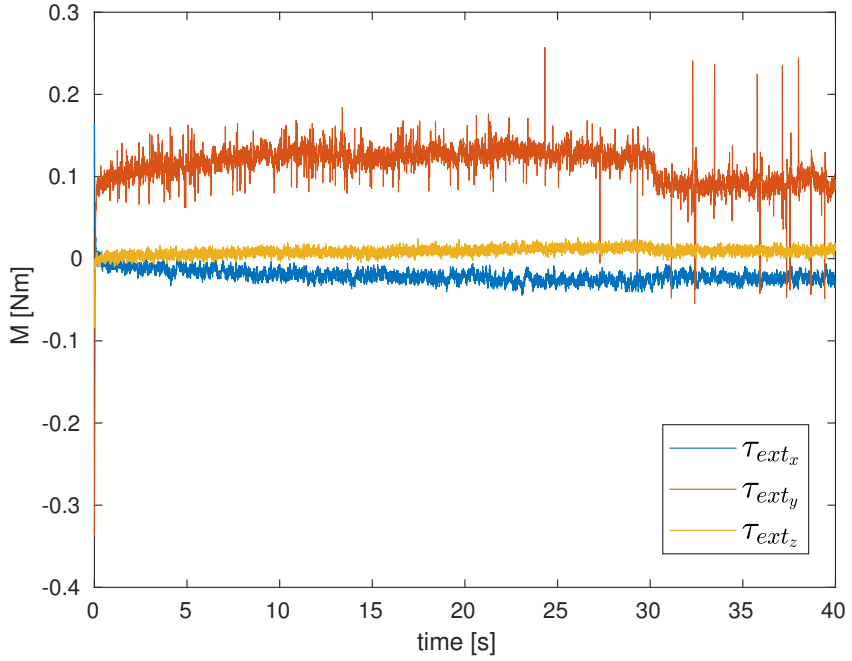


Figure 5.12: Estimated external torques without offset

Chapter 6

Task learning from Human Demonstration

This chapter deals with the learning of a suitable force model to generate the force reference for the force control loop during the polishing task by using Artificial Intelligence techniques. In particular, we first present the way through which we get data from human demonstrations and then how we implemented a signal generator based on the acquired data by using Gaussian Mixture Models (GMMs) and Gaussian Mixture Regressions (GMRs) [20].

6.1 Data Acquisition

The first step in learning from human demonstrations is that of data acquisition; in our work these data consist of the force vector applied by the human operator during an average polishing task. In order to get these data we used the Robotous force/torque sensor (see section 2.1.3) combined with ROS environment [10] (see section 2.2.4); in particular we attached the platform under Robotous sensor through double-sided tape and connected it to a Ubuntu laptop via USB as shown in figure 6.1 on the following page.

While the human operator executed the polishing task, a ROS node red the wrench applied on the force/torque sensor through USB port and published these data, in a *geometry_ msgs/WrenchStamped* message, in the *rft_ data* topic from which we recorded the z component force values using the *rosbag* package. So we first ran the *rosmaster* by typing the following command on the shell:

```
roscore
```

Then we ran the ROS node for data reading through the USB port in a new shell, from the source directory of the workspace (built beforehand), by tying the command:

```
rosrun rft_sensor read_rftdata
```

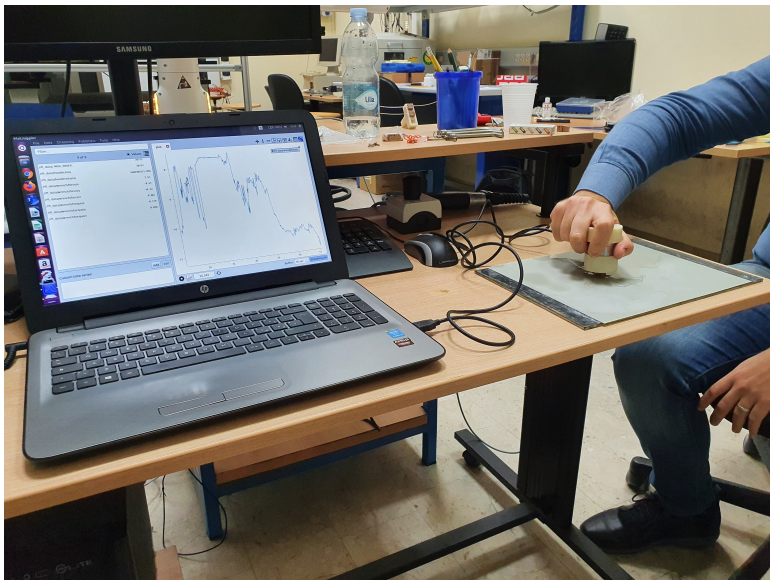


Figure 6.1: Data acquisition setup

where *read_ rftdata* is the ROS node name while *rft_ sensor* is the subdirectory containing another source directory in which we can find the ROS node. Finally we activated the recording from the *rftdata* topic by typing in another shell, simultaneously with the beginning of the polishing task, the command:

```
rosbag record rft_data
```

We used *plotjuggler* to display in realtime the data published on the topic by the first ROS node, as shown in figure 6.2 on the next page. The .bag file are saved in the current directory automatically after recording stopped with a name taken from the current date.

In this way we succeeded in recording four different .bag files about 2 minutes long (sufficient to retrieve the informations we needed) whose content can be plotted through the command:

```
rqt_bag filename.bag
```

from the directory containing the file.

Figure 6.3 on page 59 shows the acquisitions from which it is evident that the operator exerts an almost constant force during the polishing task and that this force tends to decrease between the different acquisitions (this is because these acquisitions were recorded in sequence so during the fourth one the operator was more tired compared to the first one). Even though these acquisitions contain some outliers (e.g., very large negative values in the last seconds of the third acquisition), these values did not influence the estimation since they are few; moreover the first acquisition contains some

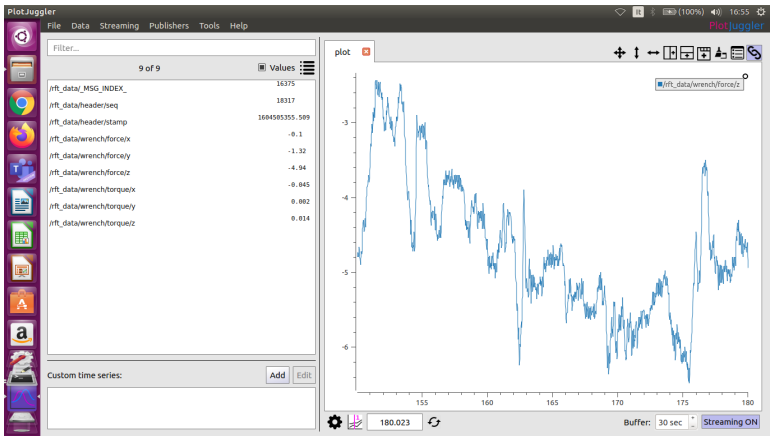


Figure 6.2: Plotjuggler output

return to zero force values due to the fact the operator stopped the motion in order to rest.

6.2 Gaussian Mixture Model (GMM)

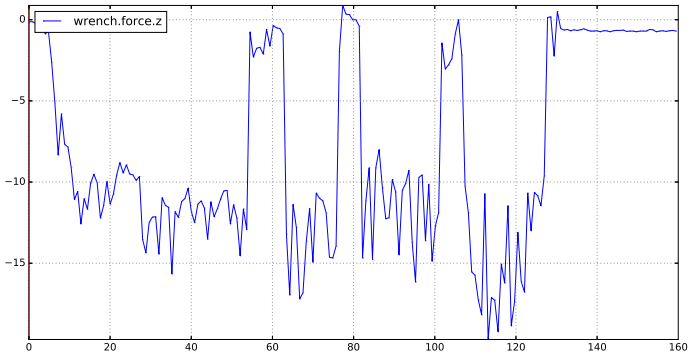
We tried to implement a signal generator based on the data acquired as described before by using a Gaussian Mixture Model. In particular, a GMM is a multivariate distribution that consists of multivariate Gaussian distribution components. Each component is defined by its mean and covariance and the mixture is defined by a vector of mixing proportions, where each mixing proportion represents the fraction of the population described by a corresponding component. In learning from examples, a GMM produces a force values density estimate over the input and output space based on the training set.

Formally, a GMM of K Gaussians is defined by the probability density function

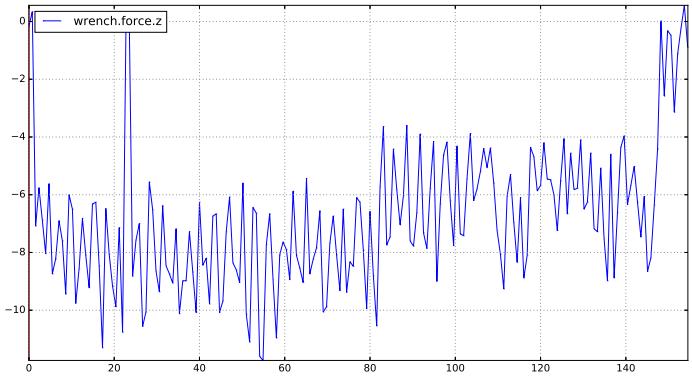
$$p(f_{z_j}) = \sum_{k=1}^{K} p(k)p(f_{z_j}|k)$$

where f_{z_j} is the data-point, $p(k)$ is the prior and $p(f_{z_j}|k)$ is the conditional probability density function. Time is encoded directly into the GMM, i.e., $f_{z_j} = (f_{z_{jt}}, f_{z_{js}})$, where $f_{z_{jt}}$ and $f_{z_{js}}$ correspond to time and spatial information, respectively. In this way smooth signal can be retrieved through regression, as described in the section 6.4. Moreover,

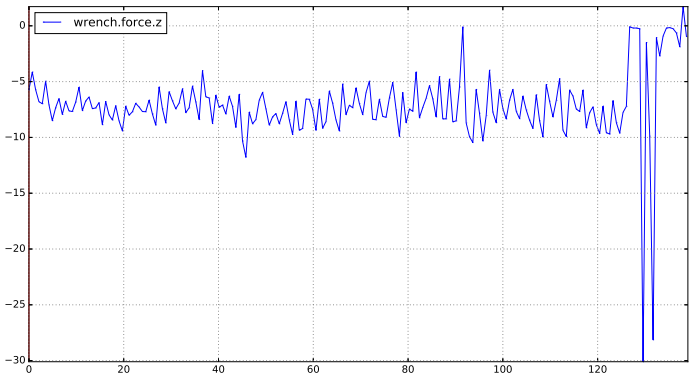
$$\begin{aligned} p(f_{z_j}|k) &= \mathcal{N}(f_{z_j}; \mu_k, \Sigma_k) \\ &= \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-\frac{1}{2}\left((f_{z_j} - \mu_k)^T \Sigma_k^{-1} (f_{z_j} - \mu_k)\right)} \end{aligned}$$



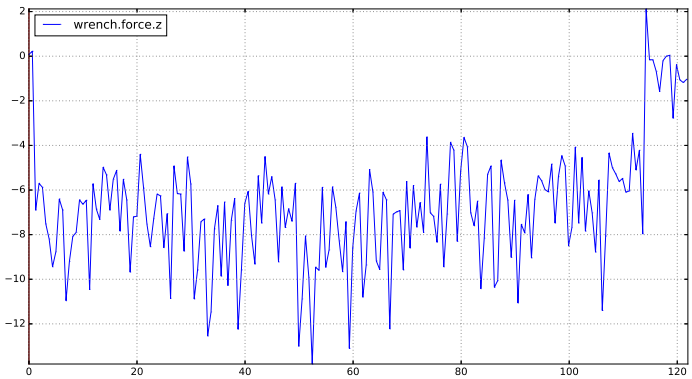
(a) *First data acquisition*



(b) *Second data acquisition*



(c) *Third data acquisition*



(d) *Fourth data acquisition*

Figure 6.3: Data acquisitions

where $p(k), \mu_k, \Sigma_k\}$ are parameters of the Normal Gaussian distribution components k defined as the priors, means and covariance respectively while D is data-points dimensionality (in our case $D = 2$ since we have only one space dimensionality and the additive time dimensionality).

Maximum Likelihood Estimation of the mixture parameters is performed iteratively using the *Expectation Maximization* (EM) algorithm [7]. EM is a local search algorithm that guarantees finding a locally optimal fit of Gaussians to the data through increasing the likelihood of the training set during optimization. This algorithm requires an initial estimation, which is done through k-means clustering. We used the programming by demonstration toolkit [1] in MATLAB for the implementation of GMMs.

6.3 Model Selection

A drawback of EM is that the optimal number of components K in a model may not be known beforehand. A common method consists in estimating multiple models with increasing number of components, and selecting an optimum based on some model selection criterion. We therefore needed to arrive at a trade-off between optimizing the model's likelihood (a measure of how well the model fits the data) and minimizing the number of parameters needed to encode the data. Different criteria have been proposed: Cross validation, Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) or Minimum Description Length (MDL) are commonly found in the literature. Cross validation has the disadvantage to require additional demonstrations to form a test set. We selected the Bayesian Information Criterion (BIC) that provided the most satisfying results with our dataset. Multiple GMMs are then estimated and the BIC score is used to select the optimal number of GMM components K:

$$S_{BIC} = -\mathcal{L} + \frac{n_p}{2}\log(N)$$

where $\mathcal{L} = \sum_{j=1}^{N} \log(f_{z_j})$ is the log-likelihood of the model using the demonstrations as testing set, np is the number of free parameters required for a mixture of K components, i.e., $np = (K-1) + K\big(D + \frac{1}{2}D(D+1)\big)$ for a Gaussian Mixture Model with full covariance matrix. N is the number of D-dimensional data-points. The first term of the equation measures how well the model fits the data, while the second term is a penalty factor which aims to minimize the number of parameters. In our experiments we compute a set of candidate GMMs with up to 10 components and we selected the model with the minimum score, $K = 2$ (see figure 6.4 on the next page).

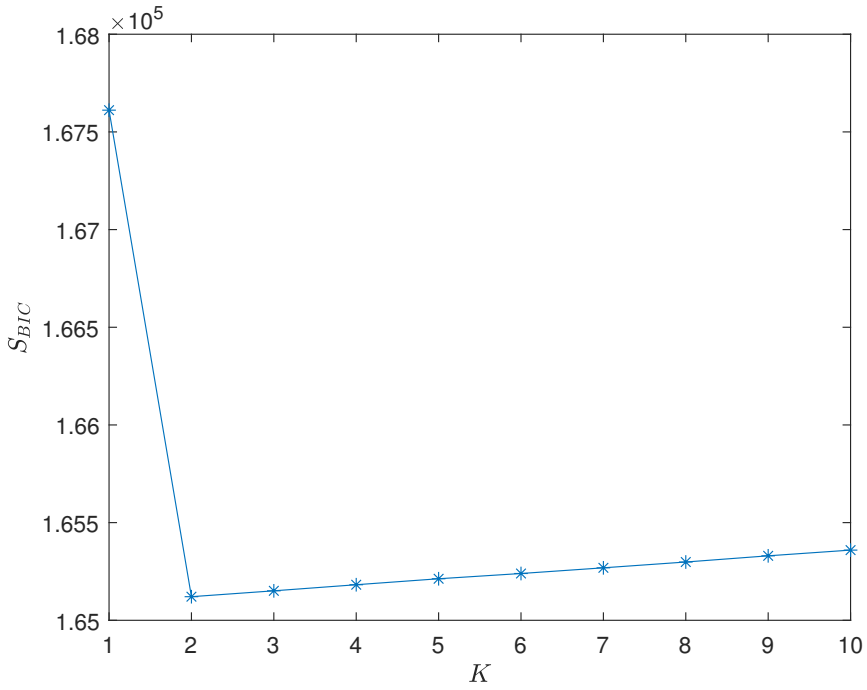


Figure 6.4: Estimation of the number of Gaussian components

6.4 Force-Value Generation by Gaussian Mixture Regression (GMR)

To reconstruct a general form for the signals we applied Gaussian Mixture Regression (GMR). Consecutive temporal values f_{z_t} were used as query points and the corresponding spatial values $\hat{f}_{z_s}$ were estimated through regression.

For each GMM, the temporal and spatial components (input and output parameters) are separated, i.e., the mean and covariance matrix of the Gaussian component k are defined by:

$$\mu_k = \{\mu_{t,k}, \mu_{s,k}\} \quad , \quad \Sigma_k = \begin{pmatrix} \Sigma_{t,k} & \Sigma_{ts,k} \\ \Sigma_{st,k} & \Sigma_{s,k} \end{pmatrix}$$

For each Gaussian component k, the conditional expectation of $f_{z_{s,k}}$ given f_{z_t}, and the estimated conditional covariance of $f_{z_{s,k}}$ given f_{z_t} are

$$\hat{f}_{z_{s,k}} = \mu_{s,k} + \Sigma_{st,k}(\Sigma_{t,k})^{-1}(f_{z_t} - \mu_{t,k})$$
$$\hat{\Sigma}_{s,k} = \Sigma_{s,k} - \Sigma_{st,k}(\Sigma_{t,k})^{-1}\Sigma_{ts,k}$$

where $\hat{f}_{z_{s,k}}$ and $\hat{\Sigma}_{s,k}$ are mixed according to the probability that the Gaussian

component $k \in \{1, \ldots, K\}$ is being responsible for f_{z_t}

$$\beta_k = \frac{p(f_{z_t}|k)}{\sum_{i=1}^{K} p(f_{z_t}|i)}$$

Therefore, for a mixture of K components, the conditional expectation of f_{z_s} given f_{z_t} and the conditional covariance of f_{z_s} given f_{z_t} are

$$\hat{f}_{z_s} = \sum_{k=1}^{K} \beta_k \hat{f}_{z_{s,k}} \quad , \quad \hat{\Sigma}_s = \sum_{k=1}^{K} \beta_k^2 \hat{\Sigma}_{s,k}$$

Thus, by evaluating $\{\hat{f}_{z_s}, \hat{\Sigma}_s\}$ at different time steps f_{z_t}, a generalized form of the force values $\hat{f}_z = \{\hat{f}_{z_t}, \hat{f}_{z_s}\}$ and associated covariance matrix were produced. We used the programming by demonstration toolkit [1] in MATLAB for the implementation of GMR.

Figure 6.5 on the following page shows the two Gaussians components of the GMM along time with their mean and covariance while figure 6.6 on the next page shows the GMR retrieval process result. We can see that the force value estimated from human demonstrations is almost constant and approximately equal to $-7\,N$; in this way we have found the desired normal force profile f_{z_d} to apply to the sandpaper during the polishing task.

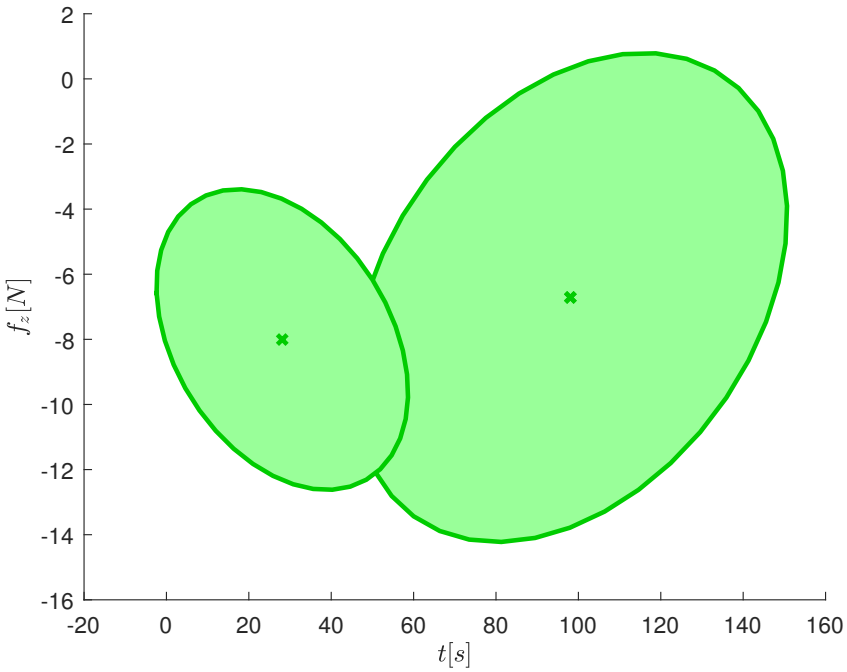


Figure 6.5: GMM representation with means and covariances

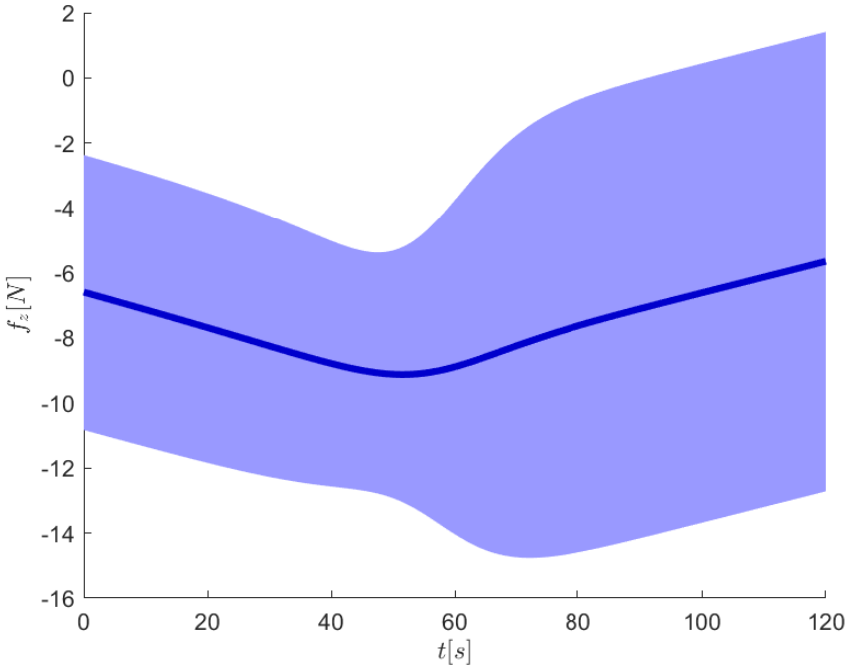


Figure 6.6: GMR retrieval process result

Chapter 7

Experimental results

This chapters shows the experimental results of the control algorithm described in the previous sections. In particular, in the experimental phase we decided to adopt a "step-by-step" strategy to prevent damages to the robot but also to understand more simply what part of the control algorithm could give problems.

For this reason we first tested the compliance control with and without the trajectory planner, then we tested the force regulation and finally we experimented the unified force/compliance control to execute the polishing task of the sensors.

7.1 Compliance Control Results

In order to test the compliance control, as a first step without the trajectory planner, we considered the control input as

$$\boldsymbol{\tau}_m = \boldsymbol{J}_A^T(\boldsymbol{q})\Big(\boldsymbol{K}_p\tilde{\boldsymbol{x}} - \boldsymbol{D}_x\dot{\boldsymbol{x}}\Big)$$
$$\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 - \boldsymbol{x}$$

With this control law we wanted the robot returned, according to the compliance given by $\boldsymbol{K}_p$, to its initial configuration when someone or something applies external forces or torques to the robot.

Figure 7.1 on the following page shows as the end-effector position returns to its initial value when we applied forces along x, y and z directions as well as figure 7.2 on the next page shows the same behaviour for the end-effector orientation described by the XYZ Euler angles φ, θ and ψ. For this reason we verified that the compliance control behaved as expected both in translation and rotation.

As a second step, we tried the compliance control in combination with

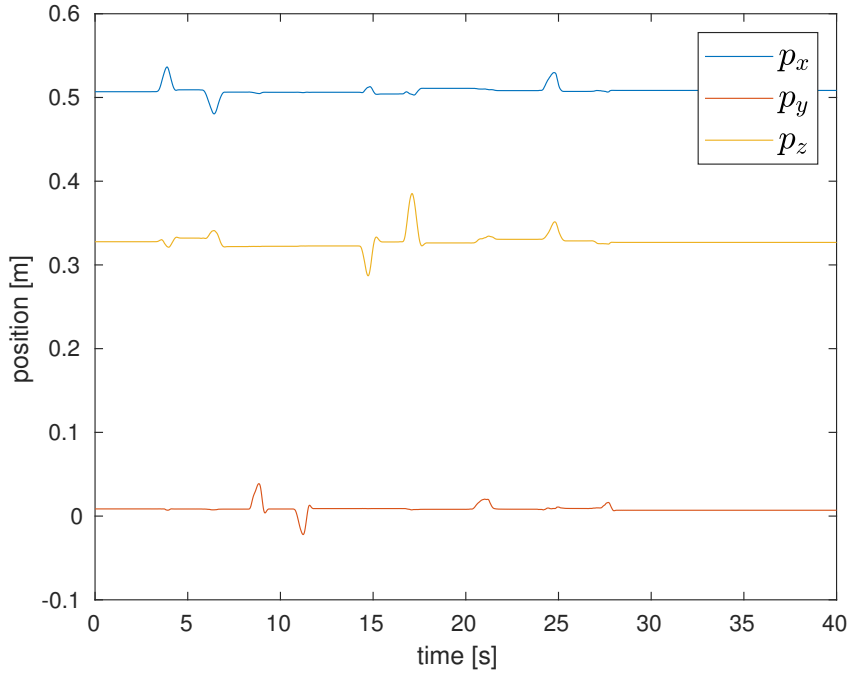


Figure 7.1: Compliance control in position without the trajectory planner

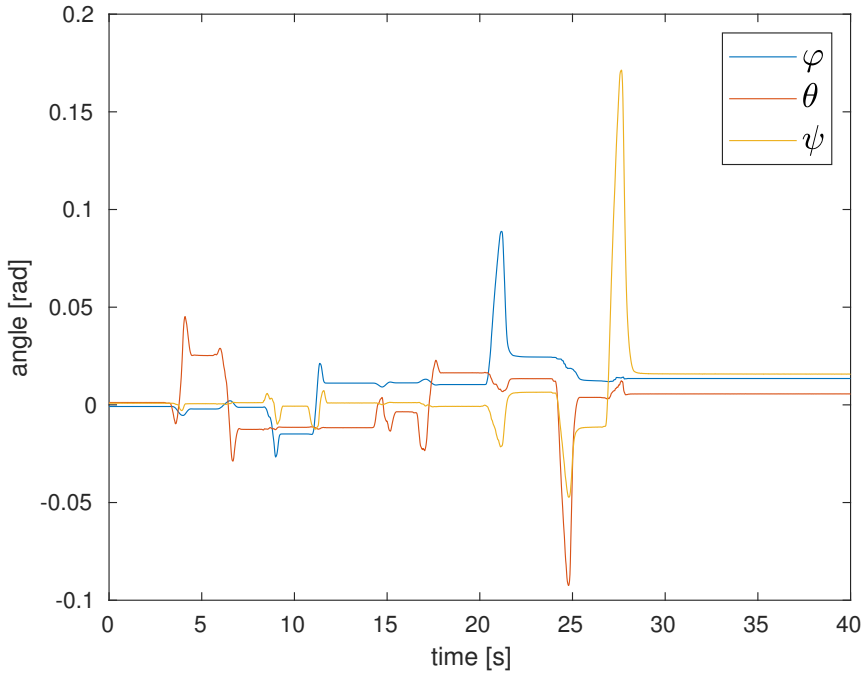


Figure 7.2: Compliance control in orientation without the trajectory planner

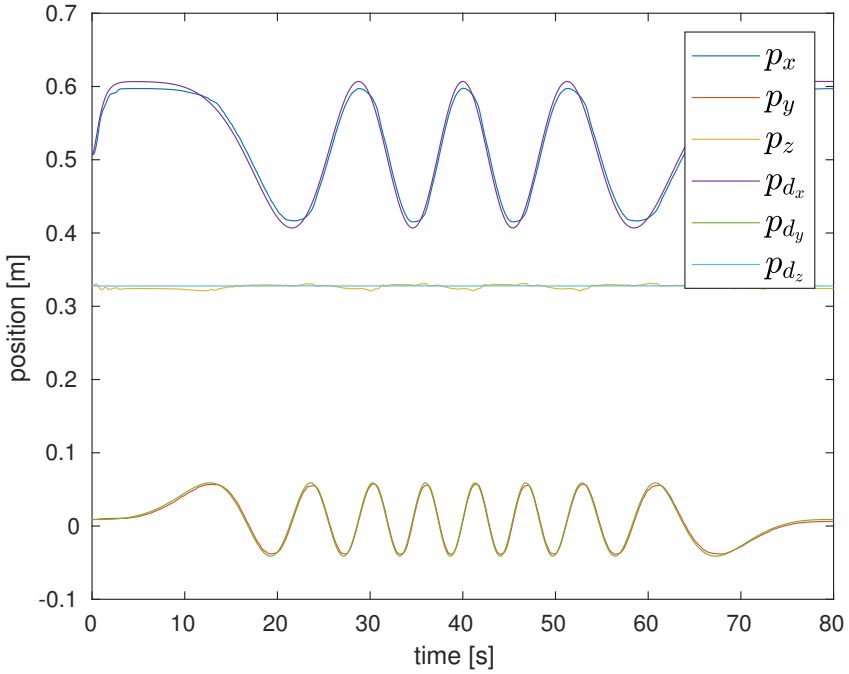


Figure 7.3: Compliance control with trajectory planner

the desired trajectory planner. So the control law was

$$\boldsymbol{\tau}_m = \boldsymbol{J}_A^T(\boldsymbol{q})\Big(\boldsymbol{K}_p\tilde{\boldsymbol{x}} + \boldsymbol{D}_x\dot{\tilde{\boldsymbol{x}}}\Big)$$
$$\tilde{\boldsymbol{x}} = \boldsymbol{x}_d - \boldsymbol{x}$$
$$\dot{\tilde{\boldsymbol{x}}} = \dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}$$

where $\boldsymbol{x}_d$ and $\dot{\boldsymbol{x}}_d$ are the output of the trajectory planning as described in section 4.2.

Figures 7.3 and 7.4 on the next page show how the compliance control works properly since the end effector follows the desired path as we expected after simulations (the initial avoidance of discontinuities allows the robot to move without displaying errors) while figures 7.5 on the following page and 7.6 on page 68 shows that the manipulator follows the desired path even though some disturbs occur during the trajectory.

7.2 Force Regulation Results

Once we were sure that compliance control works fine stand alone, we deactivated the trajectory planner and we tested the force control. So the

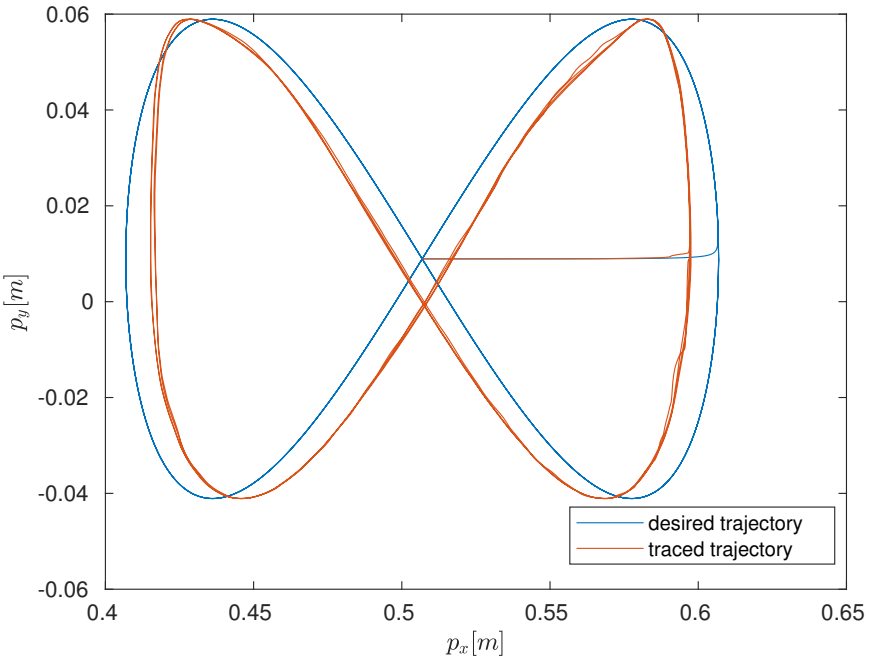


Figure 7.4: Desired and traced Lissajous figure

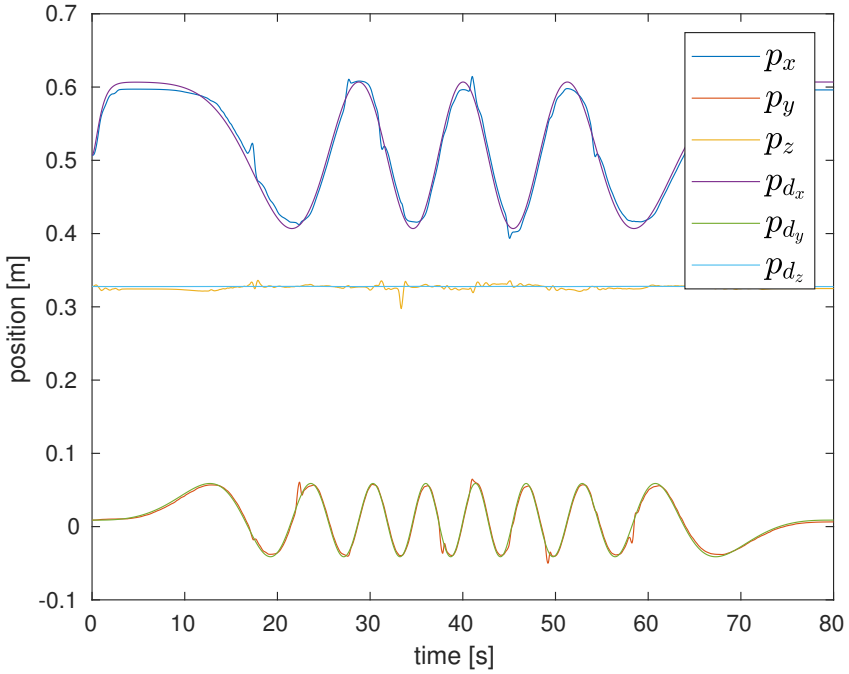


Figure 7.5: Compliance control with trajectory planner and disturbs

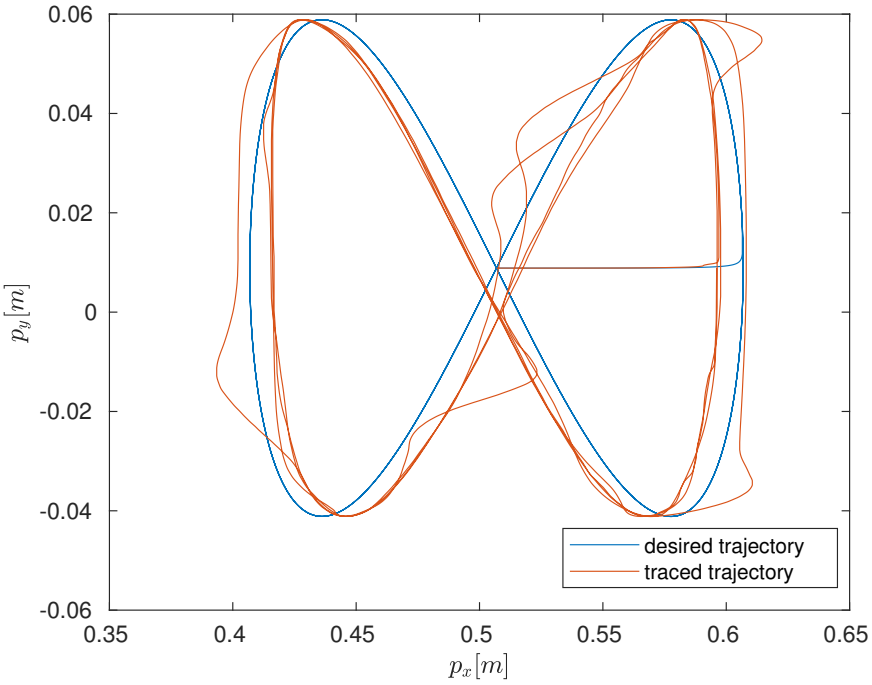


Figure 7.6: Desired and traced Lissajous figure with disturbs

considered control law was

$$\boldsymbol{\tau}_m = \boldsymbol{J}_A^T(\boldsymbol{q})\left[\boldsymbol{K}_{p_f}\Big(\boldsymbol{F}_d(t) - \boldsymbol{F}_{ext}(t)\Big) + \boldsymbol{K}_i \int_0^t \Big(\boldsymbol{F}_d(\sigma) - \boldsymbol{F}_{ext}(\sigma)\Big)\, d\sigma + \boldsymbol{K}_p\tilde{\boldsymbol{x}} - \boldsymbol{D}_x\dot{\boldsymbol{x}}\right]$$

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 - \boldsymbol{x}$$

As a first step, in order to find out if the $O\_F\_ext\_hat\_K$ function of libfranka library provided the force exerted or suffered by the robot, we chose a null desired force and tried the control law with the robot far from the environment. In particular, in this experiment we tried to exert a force to the end effector along the z direction while the control loop was running in the robot controller.

Figure 7.7 on the following page shows how the force measured by the robot is the force exerted to the end effector from the environment (for $O\_F\_ext\_hat\_K$ function end-effector z axis is downward as depicted in figure 2.5 on page 14 since we did not set for it the transformation matrix to the end-effector pose as described in section 5.1.4; however we could have done it by acting on *setK* libfranka function which sets the transformation from the end-effector frame to the stiffness frame) while figure 7.8 on the following page shows that end effector moved away along z in order to reach the desired null force.

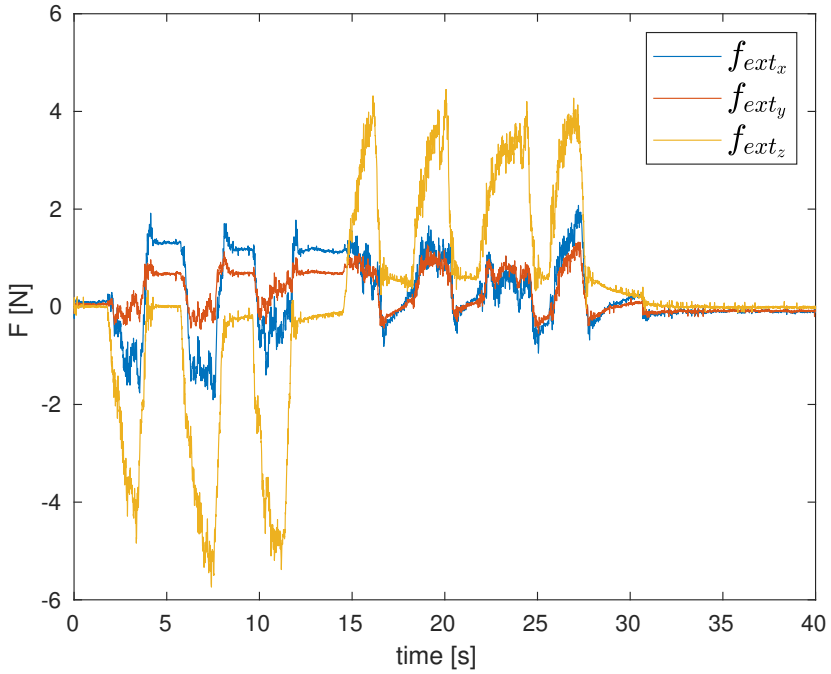


Figure 7.7: Force measured by libfranka function

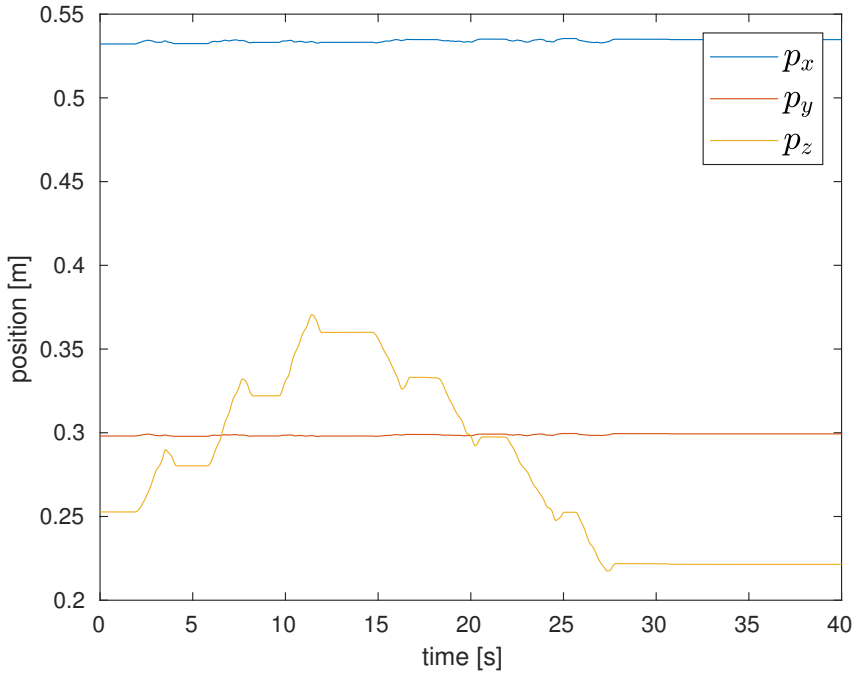


Figure 7.8: End-effector position during the experiment

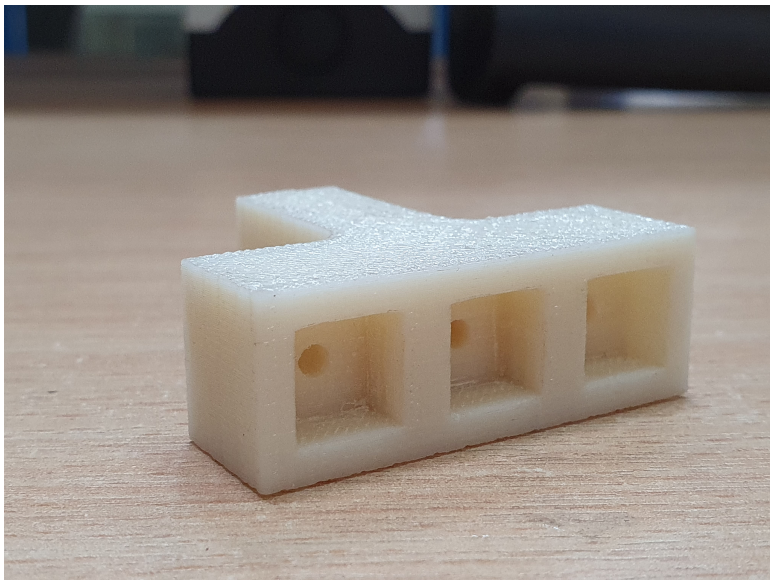


Figure 7.9: Holder with three cavities in which to place the sensors

Table 7.1: Steps size and corresponding force values

$\Delta_z\,(mm)$	$\lvert f_z \rvert\,(N)$
0.25	0.612
0.50	0.957
0.75	1.630
1	2.566
1.25	3.484
1.50	4.040

Before trying to give a non-zero value for the desired force, we had to estimate the environment stiffness in order to adapt the proportional and integral gains since we designed them in simulation, by considered a unitary stiffness (see section 5.1.4). We estimated the environment stiffness along the z direction by placing the robot end-effector, which grasped the sensors holder, in contact with the sandpaper used for the polishing task and applying small steps along z by commanding the robot through a Cartesian pose motion generator while simultaneously recording the forces generated due to the contact.

In particular, since the support under the sandpaper seemed very stiff, we gave $0.25\,mm$ at a time in order to prevent the generation of high contact forces that could have damaged the robot.

Table 7.1 shows the six steps considered along the z axis with the corresponding force values.

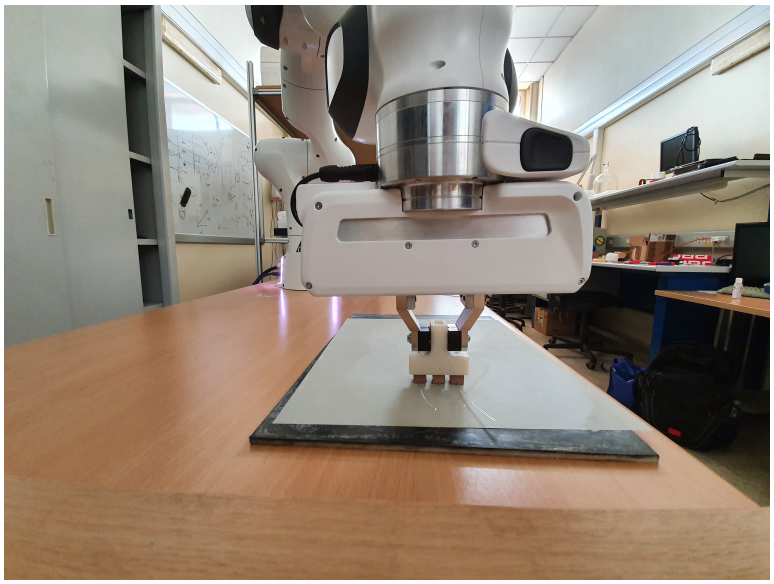


Figure 7.10: End effector in contact with the environment during the stiffness estimation

Figure 7.11 on the following page shows that the environment stiffness, which can be computed numerically with a pseudo-inversion, was of about $2608\,N/m$ but in our control we decided to over-estimate this stiffness in order to have a good steady-state error with the designed gains (since we found out that DC-gain of the system decreases with increasing stiffness).

After we completed the environment stiffness estimation process, we tested the force control with a non-zero value. In particular, we chose $f_{z_d} = -7\,N$ (as estimated in chapter 6 on page 56) in which the minus sign is due to the fact that if z axis is downward, the environment applies a force upward to the end effector (the reaction force applied by the end effector to the environment will be equal and opposite).

Figure 7.12 on the following page shows that, after a transient due to the fact that the robot did not start from contact, the desired-force value is reached as we wanted.

7.3 Sensors Polishing Results

After having tested both compliance and force control as stand alone modules, we experimented the unified force/compliance control to execute a real polishing task of $k = 4$ laps and a total duration of $80\,s$. In particular, we decided to set the z linear component in $\boldsymbol{K}_p$ to $50\,N/m$ (and the z linear component of $\boldsymbol{D}_x$ consequently) in order to have better performances.

Figure 7.13 on page 73 shows that there is a good tracking of the desired

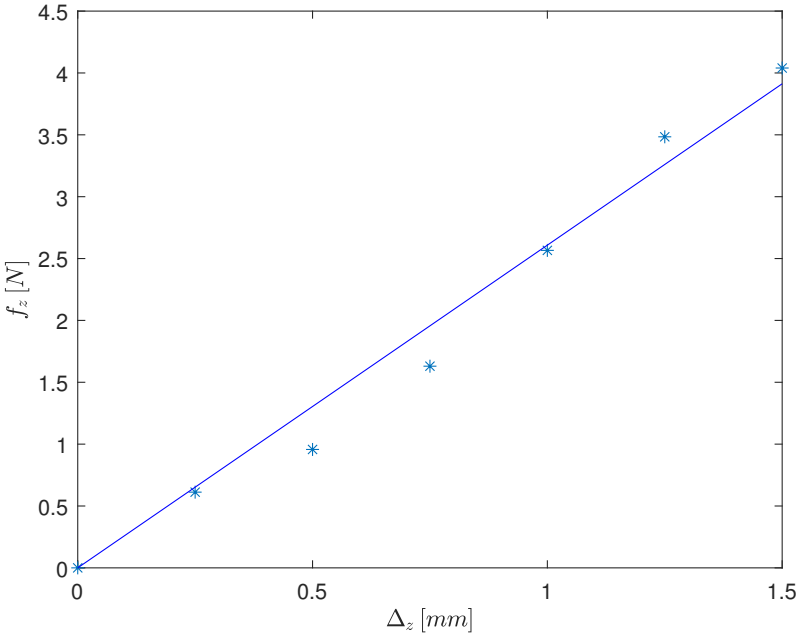


Figure 7.11: The stars represent the values reported in table 7.1 on page 70 while the solid line represents the values obtained with a least squares estimation

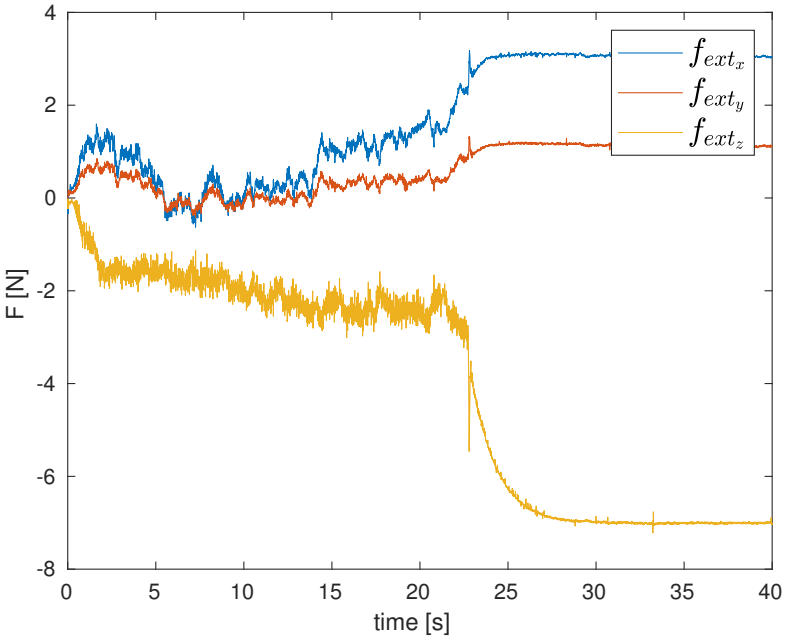


Figure 7.12: Force regulation to the desired value

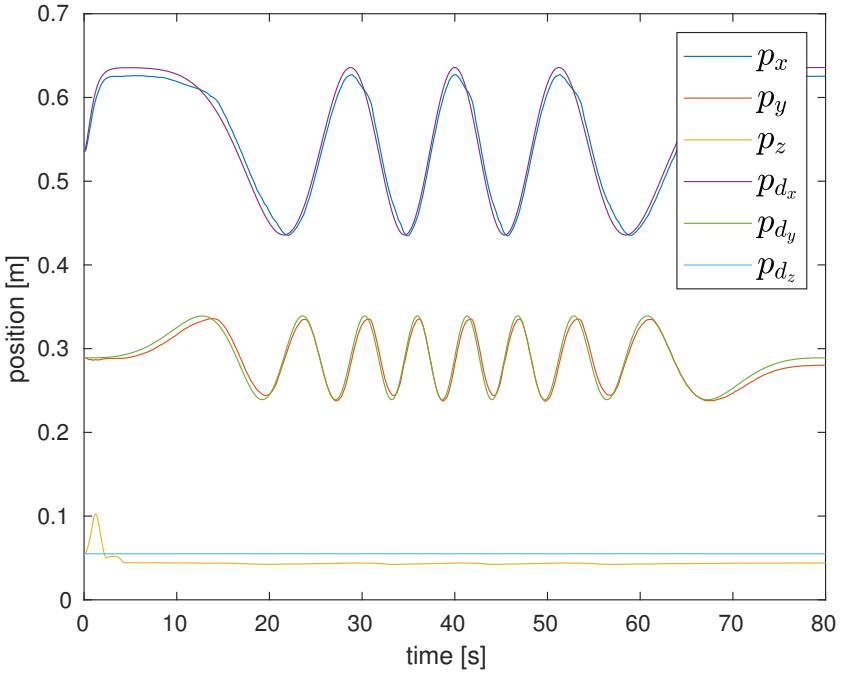


Figure 7.13: Trajectory tracking

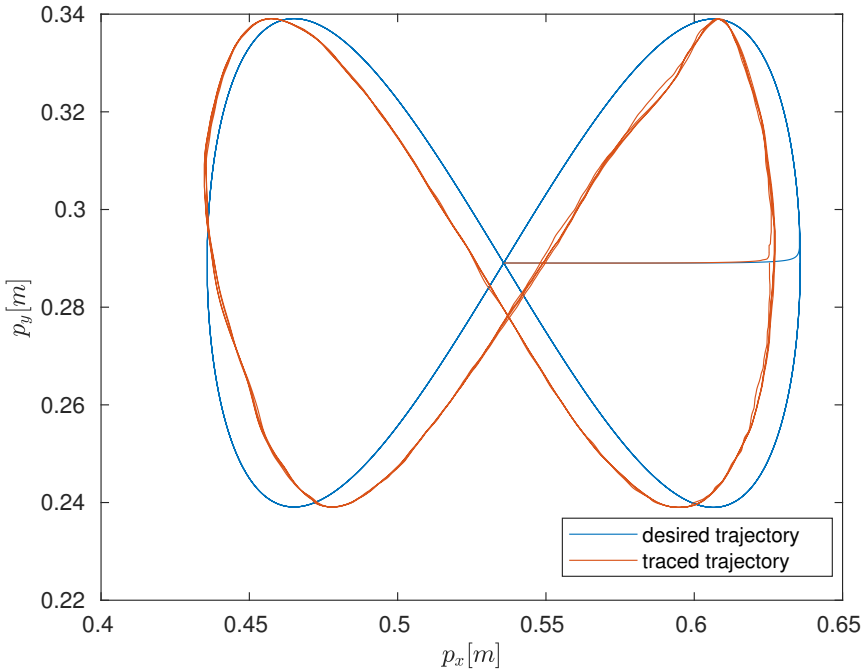


Figure 7.14: Comparison between Lissajous figures

trajectory while figure 7.14 on the preceding page shows the Lissajous figure described by the end effector during the polishing task. The matching is not perfect for the same reasons explained in section 5.2.

Figure 7.15 on the next page shows the joint torques during the polishing task which remain limited as during simulations tests. Finally, figure 7.16 on the following page shows the force regulation during the polishing task. It is important to underline that also x and y components are quite significant due to friction even though some water was dispersed over the sandpaper during the task.

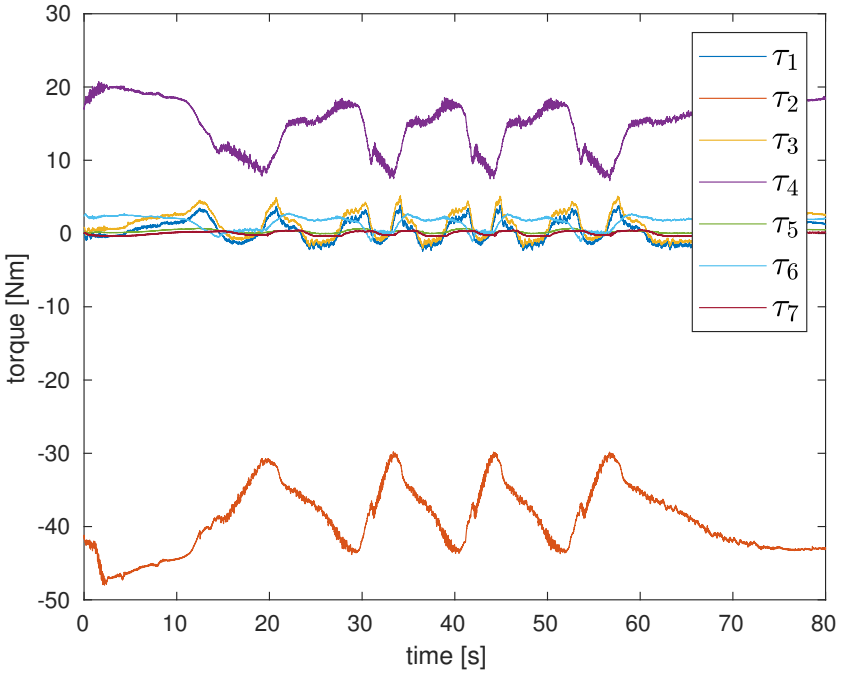


Figure 7.15: Joint torques during the polishing task

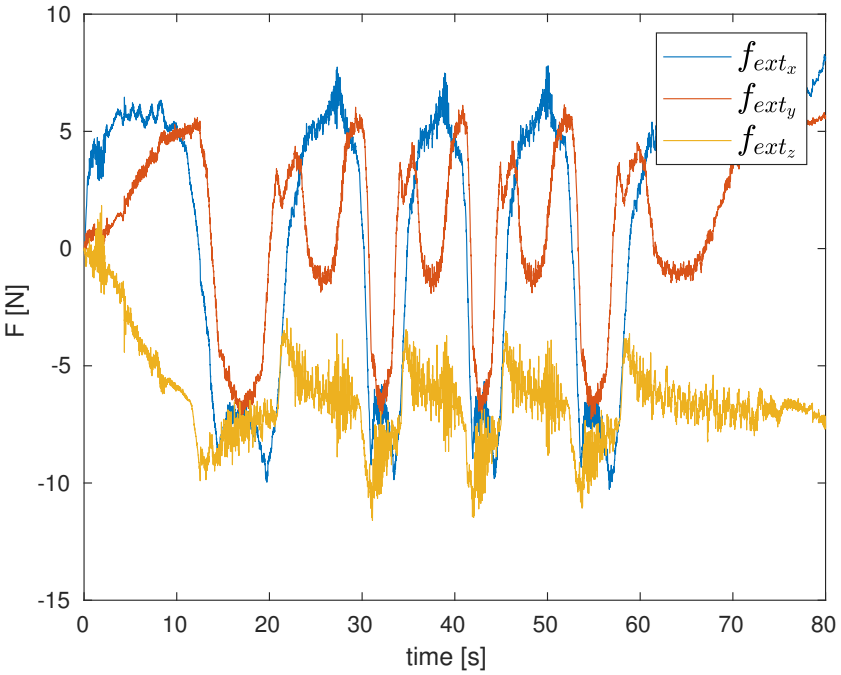


Figure 7.16: Force regulation during the polishing task

Chapter 8

Conclusions and Future developments

The aim of this work is to provide an automated approach for the polishing of POF sensors which can be used in medical field (e.g., for SARS-CoV-2 virus detection) as well as in other fields such as security or industrial field. In particular we proposed a novel method of polishing using a collaborative robot which allows to improve the ability of production (with the aim to achieve a large scale production and use of this sensors) by relieving operators from a tedious work while guaranteeing human safety during a synergistic labour between humans and robot in the manufacturing process of these sensors. This objective has been achieved through a torque-control technique which combines force regulation and trajectory tracking simultaneously; the experimental results prove the achievement of this specifications if compared with those obtained in simulation.

Many different future developments can be considered to complete and improve this work: first of all a user-friendly interface in the robot App Desk can be designed to make it possible starting the task even by an operator with no knowledge in robotics or computer science since in the current modality the control algorithm is started through the command shell. Moreover, since 4 laps are not enough to polish the sensors properly, a specific investigation can be carried out in order to find a correlation between constructional parameters which affect sensor performances (e.g., optimum polishing depth, see section 1.6) and the inputs of the automated task (i.e., total trajectory duration, the desired force profile and the number of described Lissajous loops) in order to improve the quality of the polishing task. In this sense, the use of a camera combined with the implementation of a neural network can be considered in order to allow the robot to periodically stop the movement and verify if the sensors have been polished properly or not in the same way as a human operator would do.

References

Bibliography

[1] S. Calinon, F. Guenter, and A. Billard. "On learning, representing, and reneralizing a task in a humanoid robot". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.2 (2007), pp. 286–298 (cit. on pp. 60, 62).

[2] N. Cennamo, G. D'Asgostino, F. Sequeira, F. Mattiello, G. Porto, A. Biasiolo, R. Nogueira, L. Bilro, and L. Zeni. "A simple and low-cost optical fiber intensity-based configuration for perfluorinated compounds in water solution". In: *Sensors* 18.9 (2018), p. 3009 (cit. on p. 6).

[3] N. Cennamo, D. Maniglio, R. Tatti, L. Zeni, and A. M. Bossi. "Deformable molecularly imprinted nanogels permit sensitivity-gain in plasmonic sensing". In: *Biosensors and Bioelectronics* 156 (2020), p. 112126 (cit. on p. 6).

[4] N. Cennamo, D. Massarotti, L. Conte, and L. Zeni. "Low cost sensors based on SPR in a plastic optical fiber for biosensor implementation". In: *Sensors* 11.12 (2011), pp. 11752–11760 (cit. on pp. 5, 6).

[5] N. Cennamo, M. Pesavento, L. Lunelli, L. Vanzetti, C. Pederzolli, L. Zeni, and L. Pasquardini. "An easy way to realize SPR aptasensor: A multimode plastic optical fiber platform for cancer biomarkers detection"". In: *Talanta* 140 (2015), pp. 88–95 (cit. on p. 6).

[6] R. Del Sole, M. R. Lazzoi, G. Mele, S. Mergola, A. Scardino, S. Scorrano, and G. Vasapollo. "Molecularly Imprinted Polymers: Present and Future Prospective". In: *International Journal of Molecular Sciences* 12.9 (2011), pp. 5908–5945 (cit. on p. 6).

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–38 (cit. on p. 60).

[8] F. Di Marzo, M. Sartelli, Cennamo R., Toccafondi G., F. Coccolini, G. La Torre, G. Tulli, M. Lombardi, and M. Cardi. "Recommendations for general surgery activities in a pandemic scenario (SARS-CoV-2)". In: *The British Journal of Surgey* (2020). URL: `https://doi.org/10.1002/bjs.11652` (cit. on p. 2).

[9] C. Drosten, S. Günther, W. Preiser, S. Van Der Werf, H.R. Brodt, S. Becker, H. Rabenau, M. Panning, L. Kolesnikova, R. A. Fouchier, A. Berger, A.M. Burguière, J. Cinatl, M. Eickmann, N. Escriou, K. Grywna, S. Kramme, J.C. Manuguerra, S. Müller, V. Rickerts, M. Stürmer, S. Vieth, H.D. Klenk, A. D. Osterhaus, H. Schmitz, and H. W. Doerr. "Identification of a novel coronavirus in patients with severe acute respiratory syndrome". In: *New England journal of medicine* 348.20 (2003), pp. 1967–1976 (cit. on p. 1).

[11] K. Gasior, T. Martynkien, and W. Urbanczyk. "Effect of constructional parameters on the performance of a surface plasmon resonance sensor based on a multimode polymer optical fiber". In: *Applied Optics* 53.35 (2014), pp. 8167–8174 (cit. on p. 8).

[12] C. Gaz, M. Cognetti, A. Oliva, P. Robuffo Giordano, and A. De Luca. "Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4147–4154 (cit. on p. 48).

[19] M. Moraz, D. Jacot, M. Papadimitriou-Olivgeris, L. Senn, G. Greub, K. Jaton, and O. Opota. "Clinical importance of reporting SARS-CoV-2 viral loads across the different stages of the COVID-19 pandemic". In: *medRxiv* (2020). URL: `https://doi.org/10.1101/2020.07.10.20149773` (cit. on p. 2).

[20] C. E. Reiley, E. Plaku, and G. D. Hager. "Motion generation of robotic surgical tasks: Learning from expert demonstrations". In: *Engineering in Medicine and Biology Society (EMBC)*. Buenos Aires, Argentina: IEEE, 2010 (cit. on p. 56).

[21] C. Schindlbeck and S. Haddadin. "Unified passivity-based Cartesian force/impedance control for rigid and flexible joint robots via task-energy tanks". In: *International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA: IEEE, 2015 (cit. on pp. 44–46).

[22] M. Shen, Y. Zhou, J. Ye, A. A. A. AL-maskri, Y. Kang, S. Zeng, and S. Cai. "Recent advances and perspectives of nucleic acid detection for coronavirus". In: *Journal of Pharmaceutical Analysis* 10.2 (2020), pp. 97–101 (cit. on p. 2).

[23] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, Planning and Control*. Ed. by Springer. 2010 (cit. on p. 36).

[24] M. W. Spong. "Modeling and control of elastic joint robots". In: *Journal of Dynamic Systems, Measurement, and Control* 109.4 (1987), pp. 310–318 (cit. on p. 44).

[25] L. Zeni, C. Perri, N. Cennamo, F. Arcadio, G. D'Agostino, M. Salmona, M. Beeg, and M. Gobbi. "A portable optical-fibre-based surface plasmon resonance biosensor for the detection of therapeutic antibodies in human serum". In: *Scientific Reports* 10.1 (2020), pp. 1–9 (cit. on p. 6).

Sitography

[10] Willow Garage, Open Robotics, and Stanford Artificial Intelligence Laboratory. *ROS*. 2007. URL: `https://www.ros.org/` (cit. on pp. 26, 56).

[13] Franka Emika GmbH. *Arm API documentation*. 2017. URL: `https://frankaemika.github.io/libfranka/classfranka_1_1Robot.html` (cit. on p. 25).

[14] Franka Emika GmbH. *FCI documentation*. 2017. URL: `https://frankaemika.github.io/docs/index.html` (cit. on p. 22).

[15] Franka Emika GmbH. *Hand API documentation*. 2017. URL: `https://frankaemika.github.io/libfranka/classfranka_1_1Gripper.html` (cit. on p. 25).

[16] Franka Emika GmbH. *libfranka documentation*. 2017. URL: `https://frankaemika.github.io/docs/libfranka.html` (cit. on p. 26).

[17] MathWorks Inc. *MATLAB*. 1984. URL: `https://it.mathworks.com/products/matlab.html` (cit. on p. 19).

[18] MathWorks Inc. *Simulink*. 2002. URL: `https://it.mathworks.com/products/simulink.html` (cit. on p. 19).