

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Haptic System for Teaching Manipulators

Afonso Raposo Pereira



Master in Electrical and Computers Engineering

Supervisor: Paulo Gomes da Costa

September 21, 2020

Resumo

Hoje em dia manipuladores robóticos são utilizados em abundância em diversas áreas como produção de bens materiais, operações médicas ou manuseamento de materiais perigosos. No entanto, para que o manipulador seja útil, é primeiro necessário programá-lo para que faça a tarefa pretendida. Este processo de "ensinar" o braço robótico torna-se uma tarefa mais acessível se for possível replicar manualmente os movimentos desejados. Para isto é possível tirar partido de controlo remoto permitindo controlar um manipulador à distância. No entanto, para que este controlo seja eficaz e acessível é importante que o utilizador obtenha um bom conhecimento do ambiente remoto em questão. Isto normalmente é feito com o uso de câmaras e microfones, que proporcionam capturas de imagem e som.

Este projeto de dissertação visa melhorar este tipo de controlo remoto para ensinamento de manipuladores robóticos adicionando um sistema de *feedback* háptico. Isto permite ao utilizador ter uma sensação tátil do ambiente remoto e assim um melhor controlo do braço robótico.

O sistema desenvolvido inclui dois manipuladores com três graus de liberdade, que foram modificados para englobar sensores de força, e duas interfaces gráficas que permitem um melhor controlo dos braços robóticos. É possível gravar sequências de posições captadas com a aplicação como também replicar essas sequências.

Cada manipulador é composto por três motores servo, três sensores de força, um eletroímã que serve de ferramenta e um microcontrolador. As componentes físicas do manipulador foram impressas com uma impressora 3D. O modelo original do braço robótico foi o "EEZYBotArm MK2", que sofreu as modificações necessárias para ser utilizado neste sistema.

Relativo a *software*, as interfaces gráficas foram desenvolvidas para computador através do programa *Lazarus* que utiliza a linguagem de programação *FreePascal*.

A comunicação utilizada entre o microcontrolador e a aplicação em computador é por porta série usando barramento USB. Por sua vez, a comunicação entre ambas aplicações é realizada através de mensagens TCP.

Abstract

Nowadays, robotic manipulators are vastly used in diverse areas such as goods production, medical surgeries and handling of dangerous materials. However, in order for the manipulator to be useful, it is first needed to program it to do the required task. This process of "teaching" the robotic arm becomes a more accessible procedure if it is possible to replicate the desired movements manually. To do this, it is possible to make use of remote control which provides the possibility to control a manipulator from a distance. Nonetheless, for this type of control to be effective and accessible it is important that the user acquires a good knowledge of the remote environment. This is usually done with the help of cameras and microphones which provide image and sound feedback.

This dissertation's project aims to enhance this type of remote control for teaching robotic manipulators with the addition of a haptic system. This provides the user with a tactile sensation of the remote environment and thus a better control of the robotic arm.

The system developed is composed of two robotic manipulators with three degrees of freedom, which were modified to contain force sensors, and two graphical interfaces that allow for a better control of the robotic arms. It is possible to record sequences of positions captured using the application as well as repeat those sequences.

Each manipulator is composed of three servo motors, three force sensors, one electromagnet that serves as the tool and a microcontroller. The physical components of the manipulator were 3D printed. The original model of the manipulator was the "EEZYBotArm MK2", which was modified with the necessary changes to be used in this system.

In terms of software, the graphical interfaces were developed for computer. These were created with the application Lazarus which uses FreePascal as the programming language.

The communication used between the microcontroller and computer application was via serial port using USB. On the other hand, the communication between both applications was made using TCP.

Acknowledgments

Em primeiro lugar, gostaria de agradecer ao meu orientador, Professor Paulo Gomes da Costa, pela ajuda e orientação que me forneceu durante estes últimos meses.

Quero deixar um enorme obrigado ao meu irmão Kikó por, desde sempre, acreditar em mim e no meu potencial. Por todas as vezes que discordou comigo e me fez pensar nas coisas mais objetivamente. Sem o apoio dele não seria quem sou hoje.

Aos meus pais, deixo o maior obrigado possível, por nunca duvidarem de mim e por sempre me incetivarem a querer ser melhor mas também a aproveitar estes anos que passaram.

Aos restantes membros da minha família, um muito obrigado pelos momentos, que foram sem dúvida muito importantes para mim.

Finalmente, aos meus amigos quero deixar um imenso obrigado por todas as brincadeiras, atividades e companhia. Quero agradecer em especial ao meu amigo-primo Rui Frazão, não só pelas fins de tarde a beber cerveja e comer batatas fritas, como também pelas discussões relacionadas com a tese e por partilhar o gosto pela ciência e tecnologia. Um obrigado especial à Ana Miguel e à Margarida Neves por todas as conversas, companhias e momentos que muito me ajudaram enquanto desenvolvia este projecto. E por fim, um enorme obrigado ao Granja, ao Riscas, à Bea e à Bela por estes últimos 5 anos de faculdade. Ajudaram imenso a transformar a FEUP na minha segunda casa.

Afonso Raposo Pereira

“Education never ends, Watson. It is a series of lessons, with the greatest for the last.”

Sir Arthur Conan Doyle, His Last Bow

Contents

Abbreviations	xv
1 Introduction	1
1.1 Context and Motivation	1
1.2 Objectives	2
1.3 Document Structure	3
2 Literature Review	5
2.1 Robotic Manipulators	5
2.1.1 Anthropomorphic Configuration	7
2.1.2 SCARA Configuration	7
2.1.3 Wrist and End-effector	8
2.2 Control	8
2.2.1 Forward Kinematics	8
2.2.2 Denavit-Hartenberg Convention	9
2.2.3 Inverse Kinematics	11
2.2.4 Jacobian	12
2.3 Teleoperation	13
2.3.1 Examples	14
2.4 Haptic Systems	16
2.4.1 Haptic Devices	16
3 Proposed Hardware	21
3.1 Robotic Manipulator	21
3.1.1 Body	21
3.1.2 End Effector	25
3.1.3 Complete Manipulator	25
3.1.4 Forward Kinematics	26
3.1.5 Inverse Kinematics	30
3.1.6 Jacobian	33
3.2 LX-16A Servo Motor	33
3.2.1 Command Packet	34
3.3 Load Cells	35
3.3.1 Calibration	36
3.4 Arduino Mega 2560	38
3.4.1 Visual Studio Code Editor	38
3.5 Conclusion	39

4	Proposed Software Architecture	41
4.1	System Architecture	41
4.1.1	Manipulator Subsystem	41
4.1.2	Manipulator to Computer Connection	44
4.1.3	Computer to Computer Connection	45
4.2	Communication Packets	45
4.2.1	Initial Communication Sequence	47
4.3	Force Control	48
4.3.1	End-Effector Weight Adjustment	50
4.4	Remote Control	52
4.5	Haptic Force Feedback	53
4.6	Graphical User Interface	54
4.6.1	Lazarus IDE	54
4.6.2	Local Application	54
4.6.3	Remote Application	58
4.6.4	Configuration Window	59
4.7	Recording and Sequencing	59
5	Results	61
5.1	Manual Control	61
5.2	Remote Control	63
5.3	Haptic Feedback	65
5.4	Recording and Sequence	66
5.5	Conclusion	68
6	Conclusion and Future Work	71
6.1	Conclusion	71
6.2	Future Work	72
A	Schematics	75
A.1	Connections Schematic	75
B	Channels	77
	References	79

List of Figures

1.1	System concept diagram.	2
2.1	Robotic Manipulator Example.	5
2.2	Joint Representation [2].	6
2.3	Anthropomorphic Configuration [2].	7
2.4	SCARA Configuration [2].	7
2.5	Spherical Wrist Configuration [2].	8
2.6	Kinematics Example [2].	9
2.7	Multiple Configuration for same Point [2].	11
2.8	Master-Slave control [5].	14
2.9	Canadarm on the Space Shuttle mission STS-2, Nov. 1981.	15
2.10	<i>da Vinci</i> Systems' Surgical robotics for minimally invasive surgery	15
2.11	Haptic System Schematic [8].	16
2.12	ER Haptic Controller Master [13].	17
2.13	Snaptic Paddle [14].	17
2.14	Lever Haptic System [9]	18
2.15	3D Systems' <i>Touch</i> device.	18
2.16	Turro <i>et al.</i> haptically augmented teleoperation technique [4].	19
2.17	Exoskeleton Arm Developed by [16].	19
2.18	Wearable Haptic Glove [17].	20
2.19	Haptx's VR Haptic Glove.	20
3.1	<i>EEZYbotArm Mk2</i> , created by <i>daGHIZmo</i>	21
3.2	Before and after model of the Base.	22
3.3	Model of the Base.	23
3.4	Exploded axonometric of the model of the Base.	23
3.5	Model of the Servo Holder for the Base.	23
3.6	Model of the Cell Holder for the Base.	23
3.7	Before and After of Link 2.	24
3.8	Before and After of Link 3.	24
3.9	Link 3 Motor Connector Before and After.	25
3.10	End Effector	25
3.11	Original model and Modified model.	26
3.12	Picture of Printed Robotic Arm.	26
3.13	Normal view of Robotic Arm.	27
3.14	Forward Kinematics' Profile View of Robotic Arm.	28
3.15	Forward Kinematics' Top View of Robotic Arm.	29
3.16	DH Reference Frames.	29

3.17	Notation used for Law of Cosines.	31
3.18	Inverse Kinematics Profile View of Robotic Arm.	31
3.19	LewanSoul's LX-16A.	34
3.20	TAL220 Load Cell Sensor.	35
3.21	TAL220b Load Cell Sensor.	35
3.22	Wheatstone Bridge Circuit.	36
3.23	HX711 ADC.	37
3.24	Custom Load Cell Calibrator.	37
3.25	Arduino Mega 2560.	38
3.26	Visual Studio Code Editor.	39
4.1	System Architecture.	42
4.2	Robotic Manipulator Architecture.	43
4.3	Bread Board Circuit Diagram.	43
4.4	Robotic Manipulator to Computer Architecture.	44
4.5	Computer to Computer Architecture.	45
4.6	Information Sequence Diagram.	47
4.7	Initial Sequence between microcontroller and PC.	48
4.8	Initial Sequence between local PC and remote PC.	48
4.9	Actuation calculation formula plotted.	49
4.10	Weight Adjustment Diagram for Link 3.	50
4.11	Local Link 2 Force Measure Values.	51
4.12	Remote Link 2 Force Measure Values.	52
4.13	Lazarus' Source Editor.	55
4.14	Local Application.	55
4.15	Local Application Window.	57
4.16	Local Application Window.	57
4.17	Remote Application Window.	58
4.18	Configuration Window.	59
4.19	Recorded Sequence File Example.	60
5.1	Joint 1 Manual Control Position vs Force Plot.	62
5.2	Joint 2 Manual Control Position vs Force Plot.	62
5.3	Joint 3 Manual Control Position vs Force Plot.	63
5.4	Joint 1 Remote Control Position vs Position Plot.	64
5.5	Joint 2 Remote Control Position vs Position Plot.	64
5.6	Joint 3 Remote Control Position vs Position Plot.	65
5.7	Joint 1 Haptic Feedback Plot.	66
5.8	Joint 2 Haptic Feedback Plot.	67
5.9	Joint 3 Haptic Feedback Plot.	67
5.10	Sequence File used for the test.	68
5.11	Joint 1 Sequencing Plot.	69
5.12	Joint 2 Sequencing Plot.	69
5.13	Joint 3 Sequencing Plot.	70

List of Tables

3.1	DH Parameters.	30
3.2	Motor Parameters.	34
3.3	Servo Command Packet Format.	34
3.4	Example Command Packet.	35
3.5	Load Cells' Scale Factors.	38
4.1	Channel "M" Data Packet Information.	46
4.2	Channel "M" Data Packet Example.	46
6.1	Robotic Manipulator Components List and Cost.	72
B.1	List of used Channels from Local Microcontroller to Local Computer.	77
B.2	List of used Channels from Local Computer to Local Microcontroller.	77
B.3	List of used Channels from Local Computer to Remote Computer.	78
B.4	List of used Channels from Remote Microcontroller to Remote Computer.	78
B.5	List of used Channels from Remote Computer to Remote Microcontroller.	78
B.6	List of used Channels from Remote Computer to Local Computer.	78

Abbreviations and Symbols

ADC	Analog-to-Digital Converter
CAD	Computer Aided Design
DOF	Degrees of Freedom
FDM	Fused Deposition Modeling
GUI	Graphical User Interface
IDE	Integrated Development Environment
IP	Internet Protocol
LED	Light Emitting Diode
NASA	National Aeronautics and Space Administration
PLA	Polylactic Acid
PWM	Pulse Width Modulation
SCARA	Selective Compliant Articulated Robot for Assembly
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SRMS	Shuttle Remote Manipulator System
TCP	Transmission Control Protocol
USB	Universal Serial Bus
VR	Virtual Reality

Chapter 1

Introduction

This first chapter provides an introduction to the theme of this dissertation explaining the context as well as the motivation for this project. Furthermore, some expected goals will be discussed and finally a section with the structure of the document.

1.1 Context and Motivation

With the Industrial Revolution, which took place in the early 19th century, a rise in production began and so, automation became inevitable. The first industrial robots appeared during the 1960s and were intended to replace workers in monotonous, and in many cases dangerous, tasks [1]. Not only this but industrial robots also allowed for a speed up in production which meant more profits.

The term "Robot" was first introduced to the world by the Czech writer Karel Čapek in the 1920s [1]. He wrote a play called Rossum's Universal Robots, or R.U.R., that premiered in Prague in the year 1921. The theme of the play was about man-made workers which were built to automate people's works and thus alleviate them from this burden. These machines were called Robots.

This term suffered an evolution throughout the years and nowadays, according to *ISO 8373*, an industrial robot is defined as "an automatically controlled, reprogrammable, multipurpose manipulator with three or more axes" [1].

Evidently, these robots require a large amount of time to develop. Not only in terms of building and deployment, but also in setting up to do the required task, or in other words "teaching" the robot.

In many industrial and laboratory scenarios, there comes the need to indicate the precise location and orientation of a tool being manipulated. This is needed when dealing with assembly or careful manipulation of objects, for instance. Traditionally, this is done by programming discrete spatial points and orientations on the robotic manipulator which requires special attention from trained technicians. This is usually feasible for simple motions, even though it might be time consuming. However, when dealing with more complex trajectories it takes much more time and effort to program and refine everything correctly.

Given this problem, there comes the need to create a system capable of decreasing not only the time it takes to teach a common robotic manipulator, but also the complexity that arises when doing so.

1.2 Objectives

The main objective of this dissertation's project is to develop a system that allows the teaching of a robotic manipulator in an easy and user-friendly manner. In order to do so, the system will operate in a remote controlled way. Regularly only a video stream and sound stream are used which is important to have a good understanding of the remote environment. However, a tactile sensation can become very useful in many circumstances and therefore the system should operate in real time and be able to give haptic feedback to whoever is using the device.

Because this project makes use of remote control, it is important that the communication made between both locations is done through the Internet over TCP/IP.

Having said this, the solution proposed in this dissertation consists of creating two identical robotic arms that will be integrated in a remote control system. One to be used as the local manipulator (or master manipulator) and the other one to be used as the remote manipulator (slave manipulator). Furthermore, it is also expected to provide the manipulators with the necessary sensors and actuators for the implementation of the haptic system and also develop that haptic system using force feedback.

The control scheme expected for the local manipulator is compliance control, which is when the user manually moves the manipulator to the required positions by applying forces on the arm itself. Like any remote system, when the local robotic manipulator changes poses, the remote manipulator should mimic that pose, as demonstrated in Fig. 1.1.

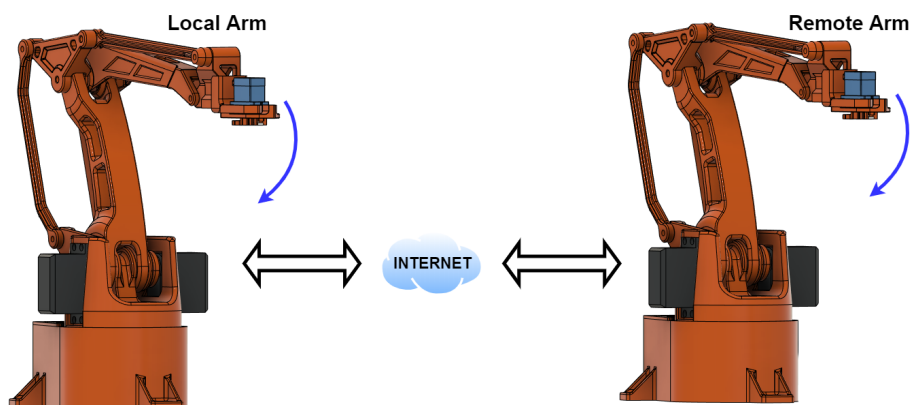


Figure 1.1: System concept diagram.

The type of haptic feedback chosen for this system was force feedback which will be made possible using the servo motors on the local manipulator. This will be useful in situations where the remote manipulator collides with something in its environment giving the user better sense.

Finally, a simple Graphical User Interface (GUI) was chosen to be implemented as the way to control and debug each robotic arm and thus give an easier experience to the user.

1.3 Document Structure

This dissertation contains a total of 7 chapters, including this first one which is the Introduction.

Chapter number 2, "Literature Review", provides some theoretical information about robotic manipulators along with a demonstration of several haptic systems from various authors and companies.

Chapter 3, "Proposed Hardware", presents the hardware used for this project. It consists of the manipulator, also explaining the modifications that were needed to be made, the motors used, the force sensors and the microcontroller.

Similarly, chapter 4, "Proposed Software Architecture", takes an overview of the software side of the project. It contains the proposed architectures along with an explanation of the communications between all the subsystems and, an explanation of the controls made and a thorough review of the graphical interfaces.

Chapter number 5, "Results", contains all the experiments made along with their analysis. This is where the validation of the system occurs.

To conclude the dissertation, chapter 6, "Conclusions and Future Work", contains the final conclusions of the project and the future work is established.

Chapter 2

Literature Review

The following chapter provides some important information and concepts that will prove useful for the well understanding of this dissertation. It is divided into two main sections: manipulators, in section 2.1, and haptic systems, in section 2.4.

2.1 Robotic Manipulators

As mentioned before, in section 1.1 of the Introduction, an industrial robot is defined as having three or more axes capable of movement. This can be accomplished with the use of actuators, for example, electrical motors, more commonly referred to as joints. A robotic manipulator is composed of several joints and links which are arranged in certain configurations depending on the task at hand. Normally one end of the manipulator is fixed and the other houses the tool, usually referred to as the end-effector. An example of a robotic manipulator can be seen in Fig. 2.1.



Figure 2.1: Robotic Manipulator Example.

The joints can be either rotational (revolute) or linear (prismatic). Revolute joints apply motion

using rotation whereas prismatic ones apply it via translation. A robotic manipulator has n degrees-of-freedom (DOF) when its configuration can be minimally indicated using n parameters, which is usually the number of joints the manipulator has. A manipulator capable of fully operating a tool in 3D space needs at least six DOF, three for translation (position) and the remaining three for rotation (orientation).

Fig. 2.2 displays the symbols that are going to be used in this chapter to represent each type of joint.

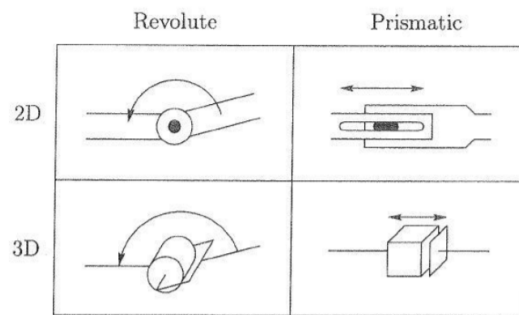


Figure 2.2: Joint Representation [2].

There are several criteria that classify robotic manipulators such as power source, application area, method of control or geometry [2].

Power source criteria refers to the type of actuators that drive the robotic manipulator. The most common ones are electrical, hydraulic and pneumatic [3] and each has its own strengths and weaknesses. Nowadays, AC-servo and DC-servo motors are being more and more used because of how cheap and accessible they are.

Application Area of a manipulator may be either assembly or non-assembly. Normally, the first ones are small and electrically powered. Non-assembly robots are used in areas such as welding, material handling, spray-painting and machine loading and unloading.

Method of Control of robotic manipulators falls into two types: servo and non-servo. Non-servo robots are not exactly programmable which almost does not classify them as robots, as defined before. This is because they rely on mechanical stoppers for movement which results in their movement being quite limited. Servo robots however, use closed-loop computer control which allows them much more freedom and multi-functionality because it is possible to determine their motion.

Geometry of the robotic manipulator refers to the types of joints used and in which configuration. The majority of geometries used are one of the following: anthropomorphic(RRR), spherical(RRP), SCARA(RRP), cylindrical(RPP), or Cartesian(PPP), where R stands for revolute joint and P for prismatic. The end-effector's reach depends on the geometry in use and the volume that the tool can operate in is called the workspace.

Some of these configurations will be demonstrated next.

2.1.1 Anthropomorphic Configuration

One common layout is the anthropomorphic, or articulated, manipulator which is composed of three revolute joints (RRR). This type is a very minimalist representation of the human arm where each joint compares to the torso, shoulder, and elbow, as demonstrated in Fig. 2.3.

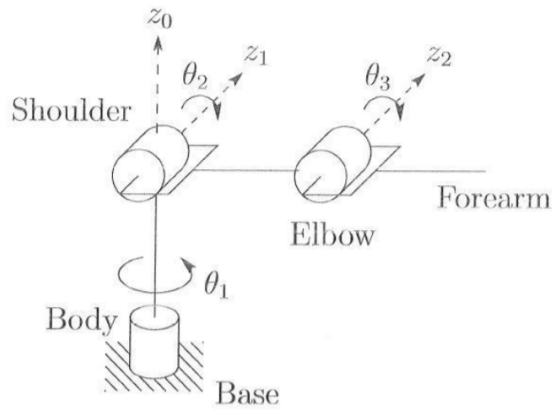


Figure 2.3: Anthropomorphic Configuration [2].

2.1.2 SCARA Configuration

As its name states, the Selective Compliant Articulated Robot for Assembly (SCARA) manipulator is refined for assembly operations, and therefore is a quite popular manipulator. Fig. 2.4 shows this configuration which is composed of 2 revolute joints and 1 prismatic joint (RRP).

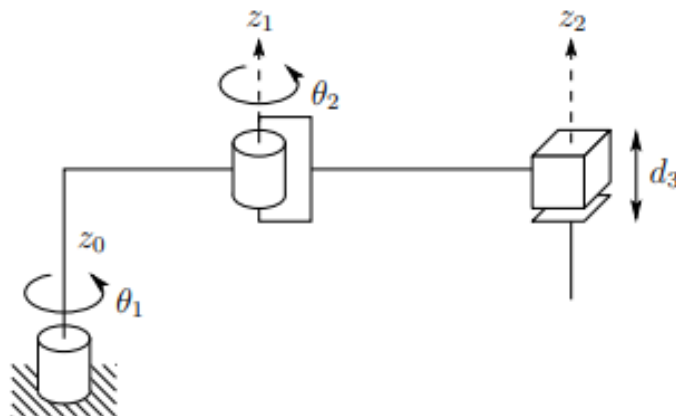


Figure 2.4: SCARA Configuration [2].

2.1.3 Wrist and End-effector

A common wrist design is the spherical wrist which is made of 3 revolute joints and allows for any orientation in 3D space. In this type of wrist, all 3 axes of rotation intersect the same point. This is a great way to approach the manipulator kinematics because it separates the translation from the rotation. Fig. 2.5 shows the configuration of said wrist.

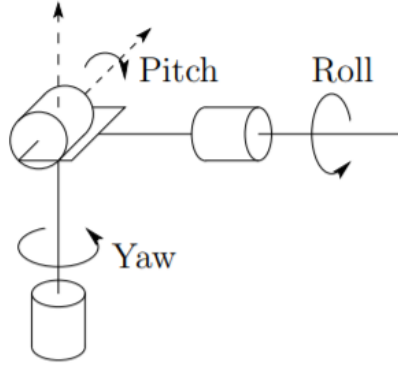


Figure 2.5: Spherical Wrist Configuration [2].

2.2 Control

The usefulness of manipulators is to automate certain tasks that might be dangerous or repetitive for an operator. This requires the machine to move to a particular point in its workspace. The way a manipulator can move from pose A (a pose is considered the position and orientation of the end-effector) to pose B is by changing its joint values, either angles in revolute joints or distance in prismatic ones.

2.2.1 Forward Kinematics

It is possible to determine the current pose of a manipulator if there is information about the current actuation of each joint along with the size of each link. This is called the forward kinematics of a manipulator.

The example present in Fig. 2.6 demonstrates a manipulator with 2 links, α_i , and 2 joints, θ_i . This information provides everything needed to calculate the point (x, y) in 2D space of the position of the end-effector.

We know from trigonometry that the projection of the position of the first joint (x_1, y_1) in the x-axis can be calculated from

$$x_1 = \alpha_1 \cos(\theta_1) \quad (2.1)$$

and similarly, for the projection in the y-axis

$$y_1 = \alpha_1 \sin(\theta_1) \quad (2.2)$$

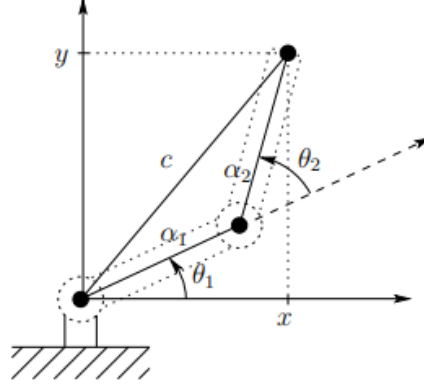


Figure 2.6: Kinematics Example [2].

Having this projection we can then calculate the projection of (x, y) . Note that the angle of actuation for the second joint is not relative to the world reference. Hence we have to add θ_1 to θ_2 in order to get the angle relative to the world and then we are able to calculate the final point of the end-effector

$$x = \alpha_1 \cos(\theta_1) + \alpha_2 \cos(\theta_1 + \theta_2) \quad (2.3)$$

$$y = \alpha_1 \sin(\theta_1) + \alpha_2 \sin(\theta_1 + \theta_2) \quad (2.4)$$

It is also easy to see from the example that the final angle of the tool relative to the world reference is $\theta = \theta_1 + \theta_2$.

This example is fairly simple and only uses two dimensions and therefore the formulas for the forward kinematics are straightforward. However, as the configurations of the robotic manipulators get more and more complicated, with more links and joints, these equations can rapidly increase in complexity.

2.2.2 Denavit-Hartenberg Convention

This convention is often used to determine the forward kinematics of a manipulator because it uses the minimum amount of parameters needed to do so. Furthermore, it is also an universal nomenclature which means anyone can understand it. Finally, it simplifies the computational necessities as much as possible.

In this convention, any homogeneous transformation is composed of a rotation along a z-axis, a translation in that same axis, a translation in the x-axis and finally a rotation around that x-axis.

This means there is only the need of 4 parameters to describe any transformation in 3D space, known as DH parameters:

- θ_i , rotation in z-axis;
- d_i , translation in z-axis;
- a_i , translation in x-axis;
- α_i , rotation in x-axis

Any transformation A from joint i to joint $i + 1$ is therefore

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i}$$

$$= \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

The key to this way of representing transformations is in the proper setup of the reference frames for each joint.

There are 3 rules when choosing the reference frames which are as follow:

1. The z-axis is the direction of the joint actuation axis;
2. The x-axis is parallel to the common normal between i and $i+1$;
3. The y-axis follows the right-hand rule.

After defining every reference frame and every DH parameter, it is possible to calculate the transformation matrix from the origin to the end-effector

$$T = A_1 A_2 \dots A_{n-1} A_n$$

$$= \begin{bmatrix} R_{3 \times 3} & P_{3 \times 1} \\ 0_{1 \times 3} & 1 \end{bmatrix} \quad (2.6)$$

where $R_{3 \times 3}$ is a rotation matrix with dimensions 3 by 3, $P_{3 \times 1}$ is the position matrix of the end-effector with dimensions 3 by 1, and $0_{1 \times 3}$ is a zero matrix with dimensions 1 by 3.

Nevertheless, the preferred way of controlling a robotic manipulator would be by specifying the position in space we wish the end-effector to go to, along with its rotation. This is where the study of Inverse Kinematics becomes useful.

2.2.3 Inverse Kinematics

The problem behind Inverse Kinematics is to calculate every joint's actuation given a pose we wish the end-effector to go to. In the previous example we could see that the Forward Kinematics formulas, 2.3 and 2.4, are non linear, meaning a solution to this problem is not always simple to find.

Furthermore, for one single position in space, there might be more than one possible configuration for the manipulator. An example of this situation can be seen in Fig. 2.7 where there are 2 possible configurations, and therefore 2 sets of possible angles for the joints.

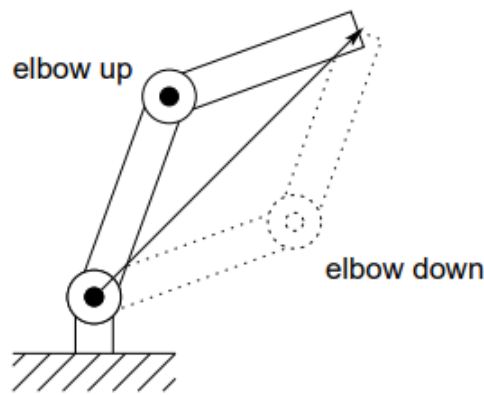


Figure 2.7: Multiple Configuration for same Point [2].

Looking back at the previous example of Fig. 2.6, the final equations for the inverse kinematics would be

$$\theta_1 = \tan^{-1} \left(\frac{y}{x} \right) - \tan^{-1} \left(\frac{\alpha_2 \sin(\theta_2)}{\alpha_1 + \alpha_2 \cos(\theta_2)} \right) \quad (2.7)$$

$$\theta_2 = \tan^{-1} \left(\frac{\pm \sqrt{1 - C^2}}{C} \right) \quad (2.8)$$

with $C = \frac{x^2 + y^2 - \alpha_1^2 - \alpha_2^2}{2\alpha_1\alpha_2}$.

The possibility to chose between positive or negative in θ_2 means one can chose between the two possible configurations in Fig. 2.7.

Notice that the equation for θ_1 depends on the angle of the second joint, which makes sense because the configuration, elbow up or elbow down, determines the angle of the first joint.

A more in-depth study of the inverse kinematics will be held when analysing the manipulator used.

2.2.4 Jacobian

Certain tasks require the manipulator to exert a certain force or move at a certain velocity. In order to do so, there comes the need to study the relationship between the velocities of the joints and the velocity/force of the end-effector. These relationships can be obtained via a matrix J , called the Jacobian of the manipulator.

Considering q a vector with n joint variables, and $x = f(q)$ a vector that defines the pose of the end-effector

$$q(t) = [q_1(t) \ q_2(t) \ \dots \ q_n(t)]^T$$

then, the Jacobian matrix J is defined as

$$\begin{aligned} \partial x &= \partial f(q) = J(q) \partial q \\ J(q) &= \frac{\partial f(q)}{\partial q} \\ &= \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \dots & \frac{\partial f_1}{\partial q_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_m}{\partial q_1} & \frac{\partial f_m}{\partial q_2} & \dots & \frac{\partial f_m}{\partial q_n} \end{bmatrix} \end{aligned} \quad (2.9)$$

The dimensions of this matrix, J , depend on the total number n of joints as well as the dimension m of the operative space. Considering the operative space to be 3D with 3 axis of rotation, then $J(q) \in \mathbf{R}^{6 \times n}$ and it is a mapping tool that relates the movement of each individual joint to the Cartesian velocity of the end-effector.

Considering a small period of time ∂t then

$$\frac{\partial x}{\partial t} = J(q) \frac{\partial q}{\partial t} \quad (2.10)$$

which is

$$\dot{x} = J(q) \dot{q} \quad (2.11)$$

Each element of the Jacobian matrix is a nonlinear function of q and so, they vary along with joint movement.

When the Jacobian is square (the manipulator has 6 joints) and non-singular, it's possible to calculate its inverse and therefore have a way to calculate the desired joint velocities in order to have a certain end-effector velocity

$$\dot{q} = J^{-1} \dot{x} \quad (2.12)$$

If the manipulator is over-actuated, meaning it has more than 6 joints, the pseudoinverse is used. However, if the configuration is composed of less than 6 joints, then this problem is only solvable if the control of the Cartesian velocities is possible.

When considering only revolute joints, the static relation between the vector forces and torques

at the end-effector, F , and the joint torques, T , is

$$T = J^T(q)F \quad (2.13)$$

where

$$F = [F_x \ F_y \ F_z \ n_x \ n_y \ n_z]^T$$

Using the previous example, Fig. 2.6, and the forward kinematics equations 2.3 and 2.4 we can define vector x like so

$$x = f(q) = \begin{bmatrix} \alpha_1 \cos(q_1) + \alpha_2 \cos(q_1 + q_2) \\ \alpha_1 \sin(q_1) + \alpha_2 \sin(q_1 + q_2) \\ 0 \\ 0 \\ 0 \\ q_1 + q_2 \end{bmatrix} \quad (2.14)$$

where $q_1 = \theta_1$ and $q_2 = \theta_2$. Note that the rotation only occurs in the z-axis.

The Jacobian can then be calculated and results in

$$J(q) = \begin{bmatrix} -\alpha_1 \sin(q_1) - \alpha_2 \sin(q_1 + q_2) & -\alpha_2 \sin(q_1 + q_2) \\ \alpha_1 \cos(q_1) + \alpha_2 \cos(q_1 + q_2) & \alpha_2 \cos(q_1 + q_2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \quad (2.15)$$

2.3 Teleoperation

Teleoperation began during the second world war because of the necessity to operate and handle radioactive materials [2]. This type of control of robotic manipulators allows the user to operate the robot from a safe distance. This method is also referred to as Remote control or Master-Slave control because there are 2 components in these systems: the control station and the remote environment. These components interchange information by some communication medium like the Internet, for example.

The operator station is where the operator controls the robotic arm with the use of another manipulator, often smaller and more lightweight, the master[4]. Similarly, the remote environment is where the manipulator that is supposed to interact with the environment and execute the tasks is located. This remote manipulator is the slave because it is supposed to do what the master, the operator, indicates. Fig. 2.8 demonstrates a schematic for this control.

Another use of this type of control is for teaching a robotic manipulator certain tasks. It is easy to understand that programming a manipulator by explicitly inputting each point in the

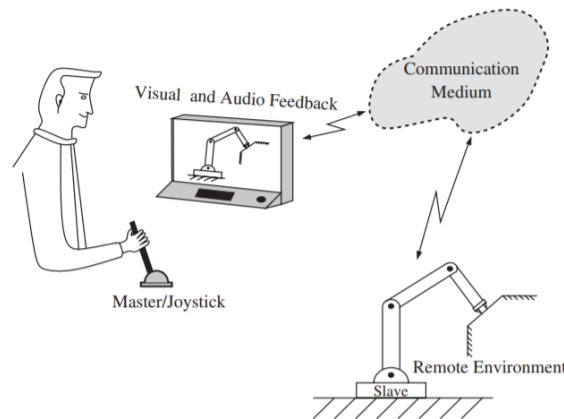


Figure 2.8: Master-Slave control [5].

correct order of movement is much harder and more tedious than replicating the task ourselves. Especially if dealing with a complex trajectory.

When working with teleoperation, having a good sense of the remote environment is quite important for proper operation of the robot. This is often done through a video stream and audio delivered to the operator allowing for feedback of the remote environment. This has a few problems, for example, in moving robots the camera's orientation might be different from that of the movement of the robot [6]. A good complement to teleoperation control systems is Haptic Technology, which will be discussed later in this chapter.

2.3.1 Examples

In November of 1981, National Aeronautics and Space Administration (NASA) launched the space shuttle mission STS-2. This space shuttle had a device called Canadarm, or Shuttle Remote Manipulator System (SRMS), attached to it. This device is a robotic arm, Fig. 2.9, and its usefulness ranged from carrying out cargo from the shuttle's bay into space, to breaking built up ice in a tricky vent. The SRMS is a 6 DOF robotic arm controlled via a joystick. Back then this was a major breakthrough for robotics and Joe Engle, the mission's commander, even mentioned "How intuitive it was to use the arm".

The *da Vinci* Surgical System² is a device used for robotic-assisted minimally invasive surgery. This system, developed by *Intuitive*, provides the surgeon performing the surgery with instruments that are guided via a console. This device has 3 main components, shown in Fig. 2.10, and there are 4 different models.

Senapati and Advincula in their paper [7] outline a safe and efficient surgical technique with the *da Vinci* surgical system. The procedure proposed is a robot-assisted laparoscopic myomec-tomy.

¹Source: <https://tinyurl.com/y8ypwetk>

²Source: <https://tinyurl.com/y8yfe49d>



Figure 2.9: Canadarm on the Space Shuttle mission STS-2¹, Nov. 1981.

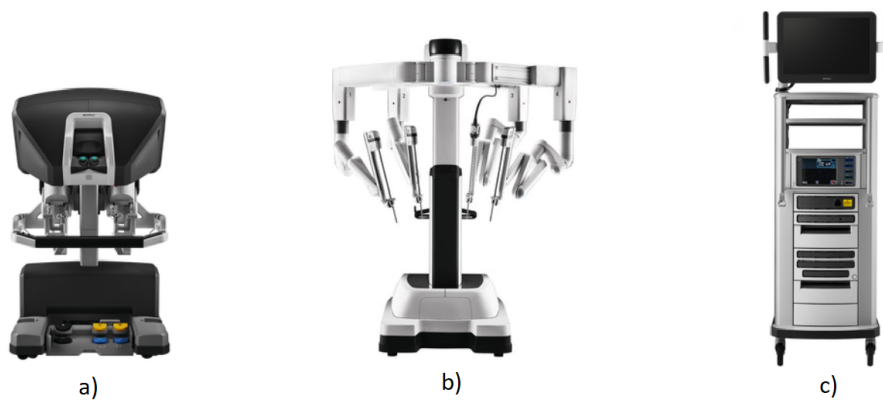


Figure 2.10: *da Vinci* Systems' Surgical robotics for minimally invasive surgery²: a) Surgeon Console; b) Patient Cart; c) Vision Cart.

2.4 Haptic Systems

The word "Haptic" derives from the Greek word *haptikos* which means "Of or relating to the sense of touch". Haptic Systems help users have a better experience when interacting with computers through the use of mechanisms that provide the sense of touch [8].

This technology has been in development for some time in areas such as manufacturing, transportation, space exploration and medical surgery [9]. A haptic system can provide the virtual sensation of touch to the operator through vibration [10], tactile sensation [11] and force feedback [6, 12, 13].

Haptic systems consist of two parts: the human and the machine [8]. Fig. 2.11 presents a schematic of a haptic system. The left-side of the figure relates to the human-being part of the system whereas the right-side refers to the machine part.

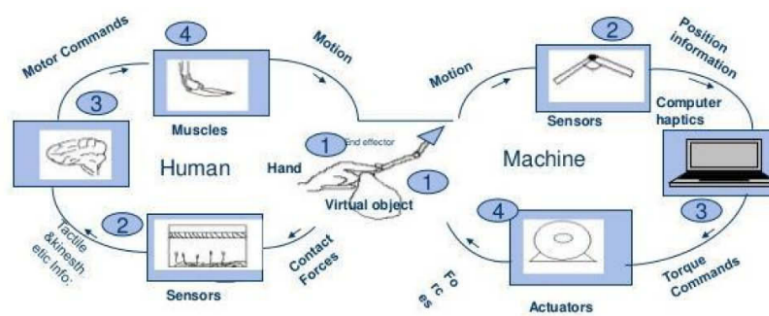


Figure 2.11: Haptic System Schematic [8].

Both parts of a haptic system are closely related. They both have sensors and actuators as well as a processing subsystem. For human beings these are the nerves receptors, the muscles and the brain, respectively. The nerves in our body sense the haptic feedback from the system which then communicate with the brain. Thereafter, the brain processes this information and sends signals to the muscles in order to create the motion required. Regarding the machine part, the sensors are, for example, encoders that provide the angles of the manipulator to the computer. These angles can be used to determine the desired end-effector position for a teleoperation control. The computer can then process the information to send to the actuators and so execute haptic feedback to the user.

2.4.1 Haptic Devices

A haptic device is referred to as a mechanical device that interfaces the user and the computer [8]. These are input-output devices because, as mentioned before, they take care of receiving the input from the user as well as outputting the desired haptic feedback.

Some types of haptic devices are joysticks with force feedback, haptic hand-controllers, exoskeleton arms and gloves.

2.4.1.1 Joysticks with Force Feedback

Oh *et al.* propose a 4-DOF haptic master using Electrorheological fluid for minimally invasive surgery system application [12]. This system is used for invasive surgery and therefore needs 4 DOF, 3 rotational and 1 translational, and is shown in Fig. 2.12. They also propose a control of repulsive force in a virtual environment in [13], where they use the 4-DOF haptic master developed previously in 2013 [12].

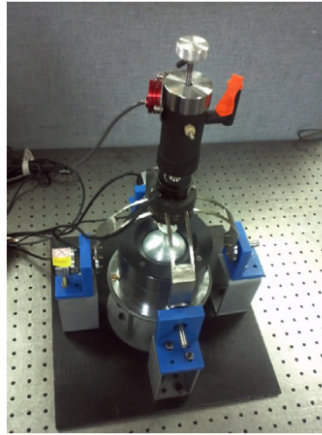


Figure 2.12: ER Haptic Controller Master [13].

Wong and Okamura in [14] propose a modification to the traditional 1 DOF Haptic Paddle design in order to create a 2 DOF device that operates like a traditional joystick. The Haptic Paddle is a haptic device used in educational settings. The problem with this is the small number of DOF available. By connecting 2 of these Haptic Paddles, they were able to create a device with 2 DOF which allows for a wider range of applications. Fig. 2.13 shows the simple Haptic Paddle on the left and the modified version on the right. This means the same can potentially be done for 3 DOF.

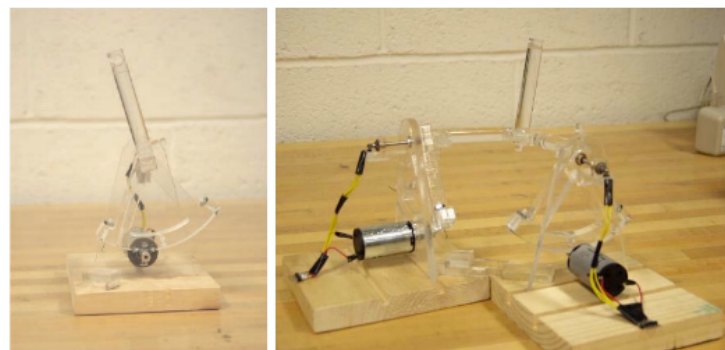


Figure 2.13: Snaptic Paddle [14].

Lee *et al.* propose a novel technique for predicting a repulsive force in a haptic interface [9].

They created a 1 DOF lever system in order to formulate a method for predicting this repulsive force, shown in Fig. 2.14. Afterwards, a steering-wheel was developed to apply this same technique.

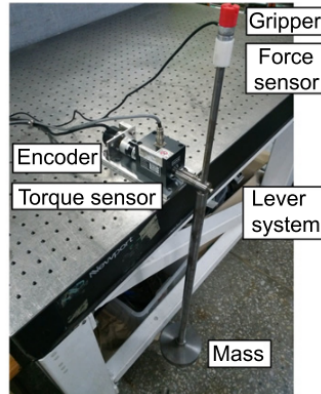


Figure 2.14: Lever Haptic System [9]

2.4.1.2 Haptic Hand-Controllers

This type of haptic controllers is very much used for medical purposes. In fact, some of them are even available for purchase by anyone. One example is the *Touch* haptic device developed by *3D Systems*, shown in Fig. 2.15.

In fact, K. Zareinia *et al.* made a more in depth study and comparison of several market available devices like the *Touch*, which used to be called Phantom OMNI, in [15].



Figure 2.15: 3D Systems' *Touch* device taken from their brochure ³.

Turro and coworkers proposed a way to haptically augment teleoperation back in 2001 [4]. In this paper, they proposed a force feedback to the master manipulator that depended on the offset error between the position of the end-effector of the slave device and the master's position. Because there would always be a small error, they used the cubic function for the computation of this feedback force in order to attenuate small errors and amplify large ones.

³Source: <https://tinyurl.com/y2lobbkbn>

Because they applied force feedback based on an error, they were able to change this value according to their needs. This enabled them to create virtual restrictions, which means they were able to create virtual trajectories. Moreover, in virtual environments with virtual objects, the master controller was able to receive repulsion forces based on the position of these virtual objects, as shown in Fig. 2.16.

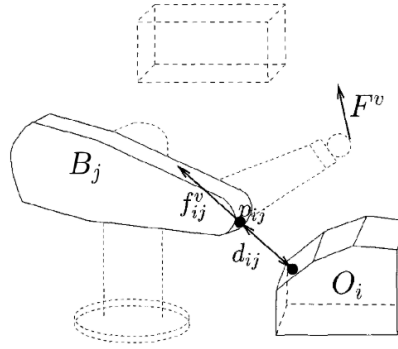


Figure 2.16: Turro *et al.* haptically augmented teleoperation technique [4].

2.4.1.3 Haptic Exoskeleton Arms

Haptic exoskeleton arms are devices built to be worn in one's arm. These devices allow for a somewhat more natural control of a robotic manipulator because it should replicate the motion someone produces with their own arm.

Carignan and coworkers developed the *Maryland-Georgetown-Army* Exoskeleton whose primary objective is for shoulder rehabilitation [16]. This exoskeleton arm, presented in Fig. 2.17, is made of 6 DOF and was integrated in a Virtual Reality (VR) environment for the task of wall painting. This experiment used different stiffness and viscosity for the virtual wall as a way of experiencing haptic feedback.



Figure 2.17: Exoskeleton Arm Developed by [16].

2.4.1.4 Haptic Gloves

Foottit *et al.* developed a haptic glove used as a game controller. This glove was composed of custom made flex sensors in each finger as well as small vibratory motors at their tips to create the haptic feedback. The glove, seen in Fig. 2.18, was custom knitted and integrated with the Unity game engine. It had the option of communicating wirelessly or by Universal Serial Bus (USB) cable with the computer program.



Figure 2.18: Wearable Haptic Glove [17].



Figure 2.19: Haptx's VR Haptic Glove³.

This startup has even collaborated with Nissan in order to develop a more realistic touch when it comes to vehicle design⁴.

³Source: <https://haptx.com/technology/>

⁴Source: <https://haptx.com/nissan-collaboration/>

Chapter 3

Proposed Hardware

This following chapter is entitled to demonstrate the physical tools used throughout the development of this dissertation's project.

Initially, a study of the robotic arm will be held. Starting with the changes made to the original model and followed by the theoretical study of the configuration at hand. Next, an overview of the servo motors used will take place followed by the force sensors and finally the microcontroller used.

3.1 Robotic Manipulator

3.1.1 Body

The body of the manipulator is a modified version of the *EEZYbotArm Mk2* from *daGHIZmo*¹, which is a Palletizer configuration that has three DOF, seen in Fig. 3.1.



Figure 3.1: *EEZYbotArm Mk2*, created by *daGHIZmo*¹.

¹Source: <https://www.thingiverse.com/thing:1454048>

The Computer Aided Design (CAD) files for this robot arm can be downloaded from the *Onshape*² website where the author has provided them for free. This is extremely useful for the changes that need to be made to the several parts because it allows for the use of a software capable of reading and modifying these files. These modifications were made using *Autodesk's Fusion 360* program.

The base of this model was original made to use servos like the MG955 or MG946R. However, because the motors used are different and don't have the same dimensions, the base of the body had to be slightly changed. Moreover, in order to measure the force applied to the base's axis of rotation, a load cell sensor had to be inserted somewhere and thus some further modifications were made. This was the case for the second and third links as well.

3.1.1.1 Base

The customization of the base consists of two different modifications from the original model. Fig. 3.2 shows the original version and the modified version for comparison purposes.

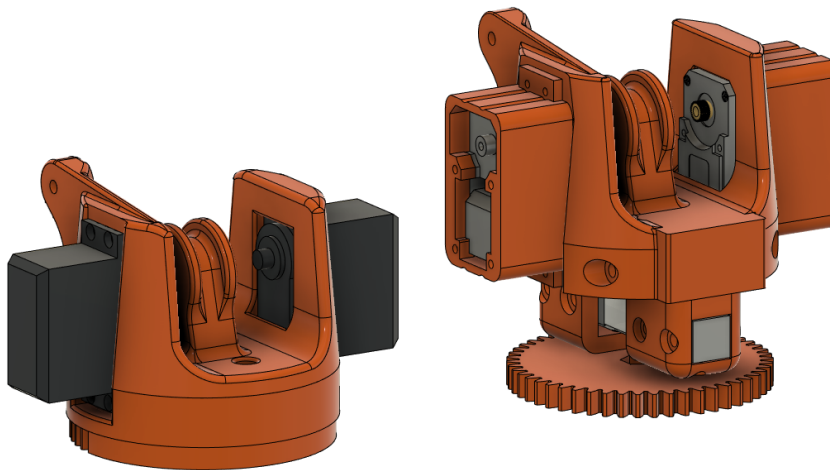


Figure 3.2: Before and after model of the Base.

First, the slots for the servos were changed and a holder to fix them in place was modeled. Second, a holder for the strain gauge cell was implemented. Fig. 3.3 demonstrates the complete modified version of the base and Fig. 3.4 demonstrates its exploded axonometric.

The servo holders are a simple casing that attaches to the backside of the servo using screws and then is attached to the base and also to the links 2 and 3. Fig. 3.5 shows the servo encased in its holder.

In order to implement these servo holders, there was the need to modify the base itself as well which was done to maintain the position of the axis of the old base to avoid further changes on the links.

²Source: <https://tinyurl.com/y3tfx4q3>

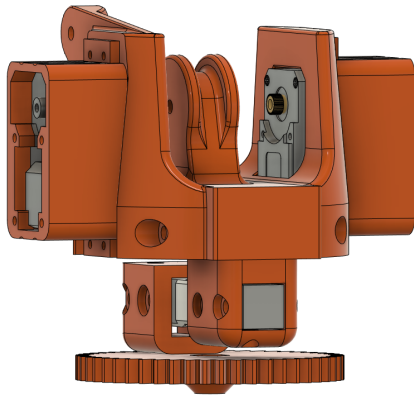


Figure 3.3: Model of the Base.

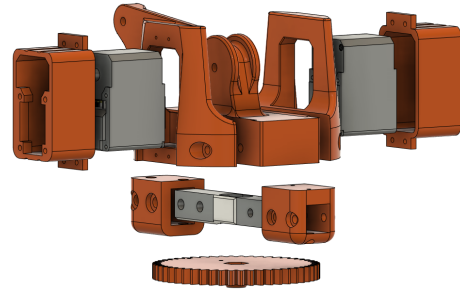


Figure 3.4: Exploded axonometric of the model of the Base.

The load cell sensor measures a force applied to it in one axis. In order for this component to work properly, both sides cannot be fixed to the same rigid body. This would prevent the sensor from flexing and thus measuring the force properly would be impossible.

The sensor used in this component was the TAL220 and Fig. 3.6 demonstrates the holder created for it. It is composed of two separate pieces that are fixed to either the base or the full gear. This way it is possible to read the force applied to the z-axis of rotation of the base.

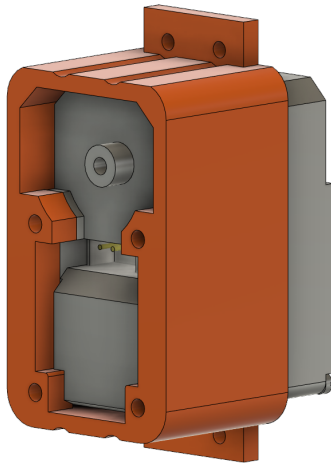


Figure 3.5: Model of the Servo Holder for the Base.

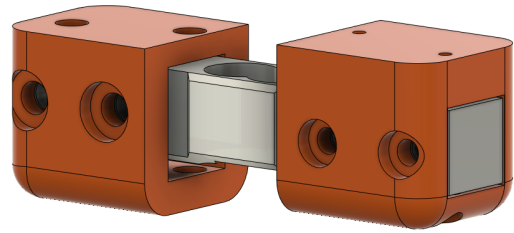


Figure 3.6: Model of the Cell Holder for the Base.

3.1.1.2 Link 2

The changes made on the second link were simple and straightforward. The main necessity was to insert the force sensor, and for it to work the model had to be split into two separate pieces. The sensor used was the same model as the one used previously in the base, the TAL220. Apart from that, the connection between the link and the motor had to be slightly changed because of the different motors used. Both the original and modified versions of this link can be seen in Fig. 3.7.

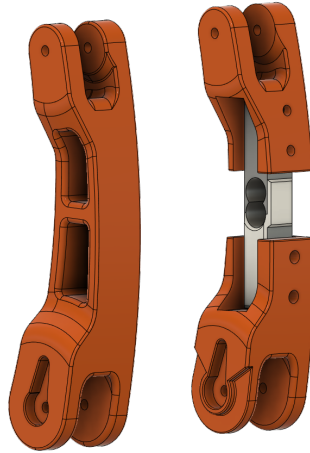


Figure 3.7: Before and After of Link 2.

3.1.1.3 Link 3

In terms of link 3, the only change needed was the insertion of the force sensor. Similarly to link 2, the model was split into two separate pieces in order to place the sensor in between them. Like the previous two modifications, the sensor used in this component was the TAL220. Fig. 3.8 shows the original model alongside the modified version.

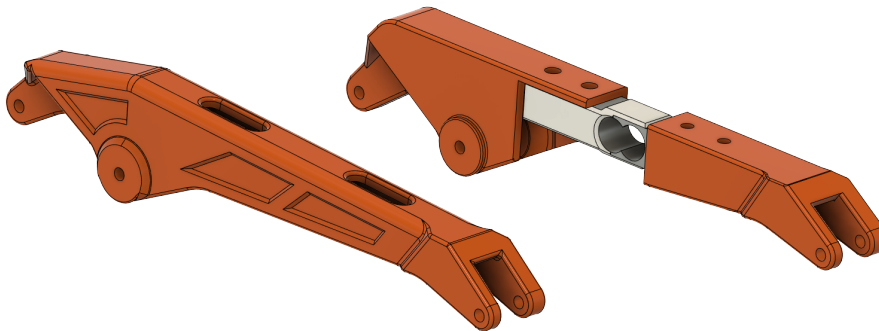


Figure 3.8: Before and After of Link 3.

Because the motor is not directly connected to the joint, there was no need to change this model any further. However, the piece that is connected to the motor's axis had to be modified in the same manner as link 2, as shown in Fig. 3.9.

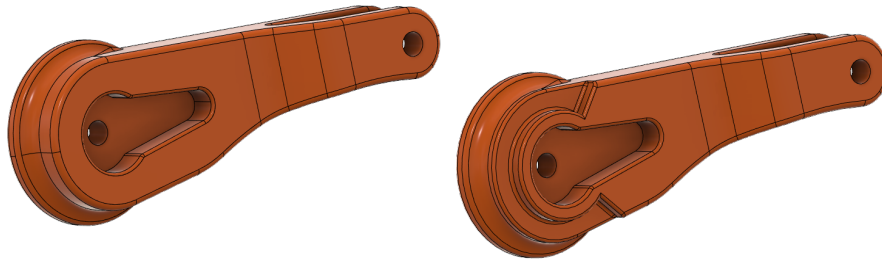


Figure 3.9: Link 3 Motor Connector Before and After.

3.1.2 End Effector

The end effector was changed from the original claw to an electromagnet. To do this a support that combines the electromagnet with the rest of the body was created. This was a simple piece that is demonstrated on Fig. 3.10.

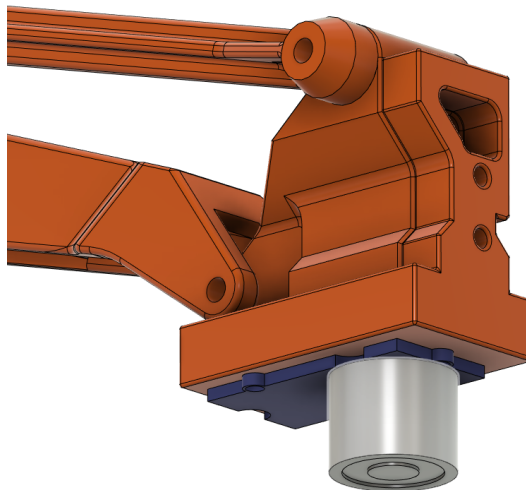


Figure 3.10: End Effector

This piece was developed with the intention of pointing the electromagnet downwards towards the floor instead of forwards in order to facilitate the action of picking up objects.

3.1.3 Complete Manipulator

Finally, after every modification made to the several more critical parts, it is possible to assemble the final version of the custom manipulator.

This model is presented in Fig. 3.11 alongside the original model for comparison purposes.

The manipulator was manufactured using 3D printing technology. This fabrication method is nowadays a very accessible way to create objects from a digital 3D model. The technique used was Fused Deposition Modeling (FDM) and the material used for printing each part was white

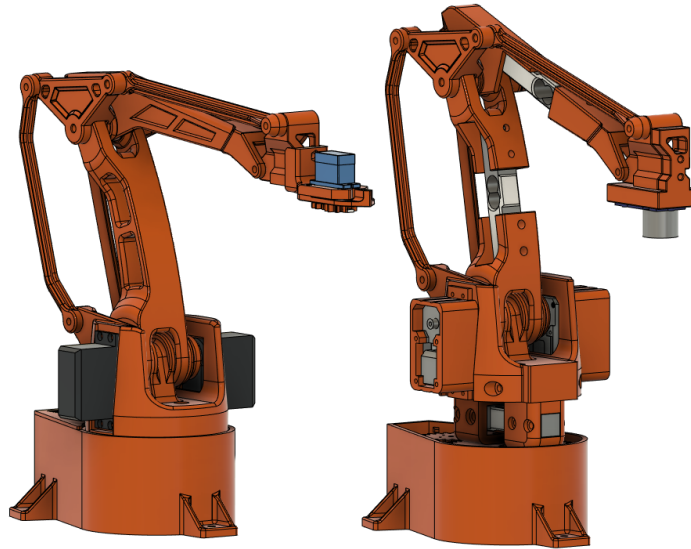


Figure 3.11: Original model and Modified model.

PolyLactic Acid (PLA). The models are presented in orange color as a way to contrast with the white background of the document.

The complete printed and assembled robotic arm can be seen in Fig. [3.12](#).

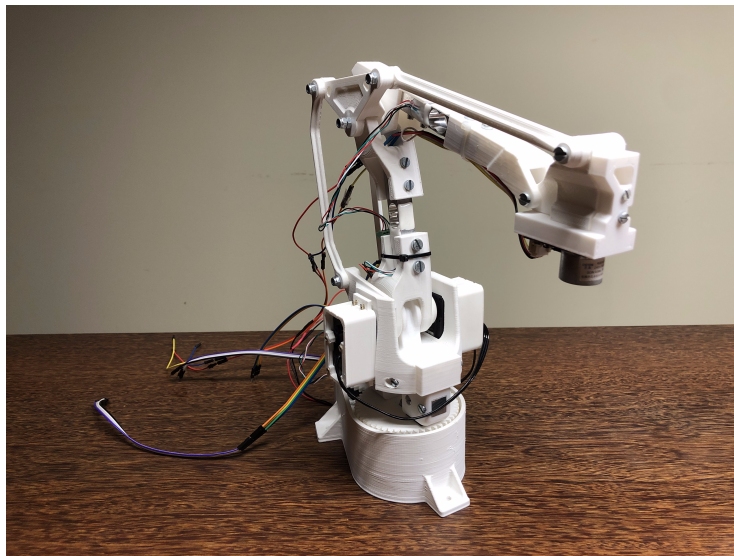


Figure 3.12: Picture of Printed Robotic Arm.

3.1.4 Forward Kinematics

As mentioned in chapter [2](#), the objective of the study of Forward Kinematics is to calculate the position and rotation of the end-effector of a robotic arm.

Fig. 3.13 demonstrates a pose of the robotic arm in which the goal is to determine the position P and the orientation of the arm.

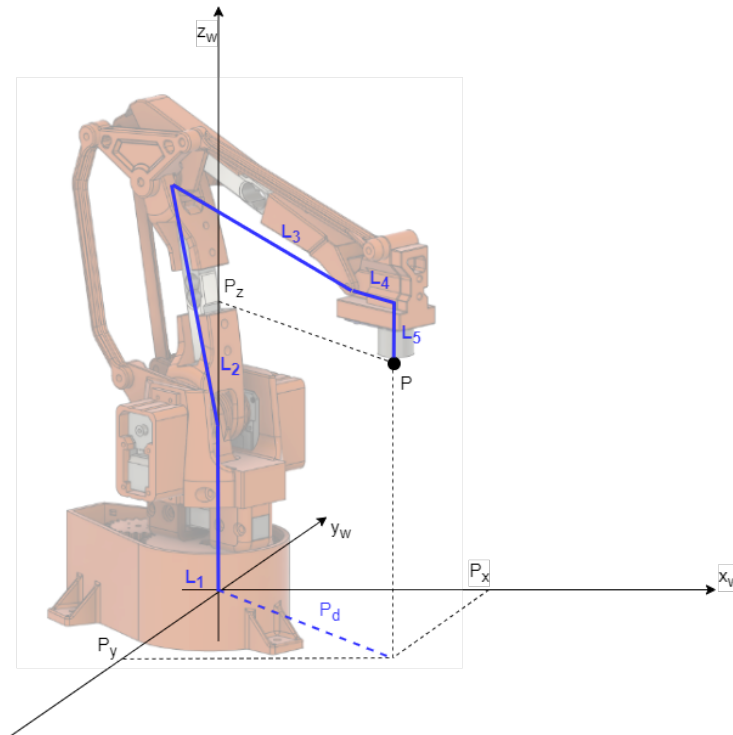


Figure 3.13: Normal view of Robotic Arm.

3.1.4.1 Geometrical Analysis

As a way to breakdown this problem, a study of the profile of the manipulator will be taken first which will then be expanded to 3D space.

Fig. 3.14 is a profile view of the manipulator and in it we can see the 3 main links that compose this robot arm as well as the second and third joints.

Given the configuration of this robotic manipulator, the angle of the third joint, θ_3 , relative to the world frame $ox_wy_wz_w$ does not vary with a change in the second joint, θ_2 . This happens because the motor that controls the third joint does not change it's position, and therefore doesn't rotate with actuation on the second joint.

Furthermore, and also because of the configuration, links 4 and 5 are always parallel and perpendicular to the ground plane, respectively. This means that we can simply add the offsets of link 4 and 5 to achieve the final position of the end-effector.

There is a slight angle offset between the angle of the second joint and the second link's direction. It is possible to see this offset, θ_{offset} , in Fig. 3.14. Joint number 2 has an angle of approximately 90 degrees, however Link 2 is not aligned with the z-axis. This offset is of around 11 degrees. This is due to how the model of link number 2 was modelled.

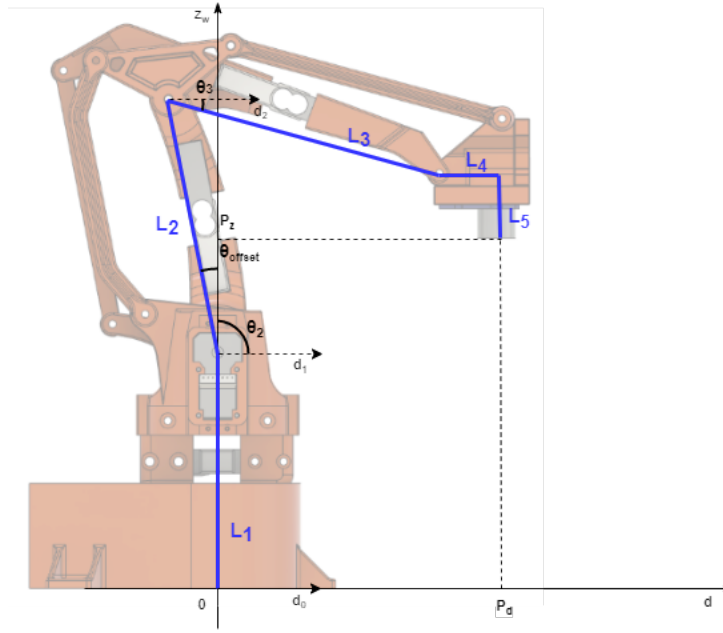


Figure 3.14: Forward Kinematics' Profile View of Robotic Arm.

Finally, the reference of the third joint is not the same as the reference of the second joint. Their axis of rotation are in fact opposite to one another and so a positive actuation in θ_3 is indeed a negative actuation in the reference of the second joint. This happens because the motors are placed in opposite directions in the base of the manipulator.

The position of the tool relative to the coordinate frame odz_w is

$$P_d = L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3) + L_4 \quad (3.1)$$

$$P_z = L_1 + L_2 \sin(\theta_2 + \theta_{offset}) - L_3 \sin(\theta_3) - L_5 \quad (3.2)$$

In this top down view of the robotic manipulator, Fig. 3.15, the blue line is the axis d of the previous profile Fig. 3.14.

Knowing the angle at which joint number 1, θ_1 , is and the position of the tool in the d axis, we can easily calculate its position in the coordinate frame $ox_w y_w z_w$ like so

$$P_x = \cos(\theta_1) P_d = \cos(\theta_1) (L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3) + L_4) \quad (3.3)$$

$$P_y = \sin(\theta_1) P_d = \sin(\theta_1) (L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3) + L_4) \quad (3.4)$$

$$P_z = L_1 + L_2 \sin(\theta_2 + \theta_{offset}) - L_3 \sin(\theta_3) - L_5 \quad (3.5)$$

Because of the configuration of the manipulator, the tool's orientation is always pointing downwards in the z -axis independently of any of the joint's angles.

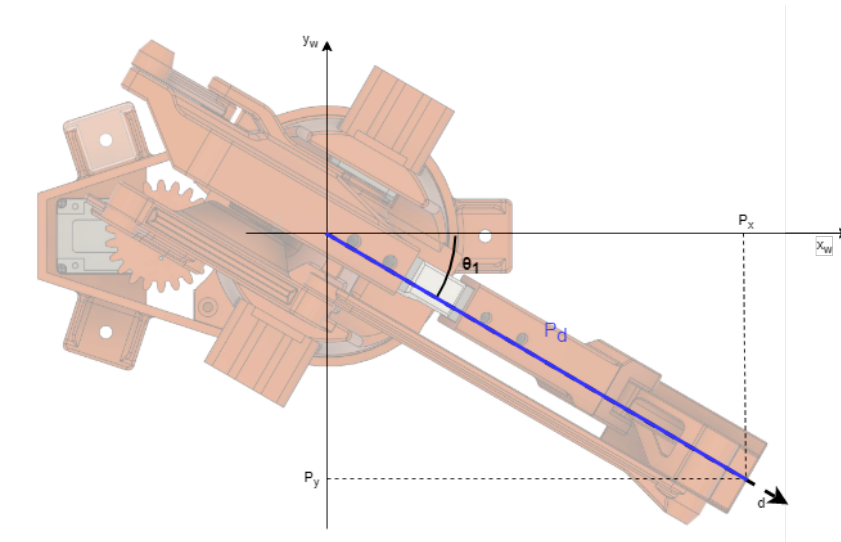


Figure 3.15: Forward Kinematics' Top View of Robotic Arm.

3.1.4.2 Denavit-Hartenberg Parameters

The choice of reference frames for the manipulator at hand is displayed in Fig. 3.16, where the blue vector represents the z-axis, the red vector represents the x-axis and finally the green vector represents the y-axis.

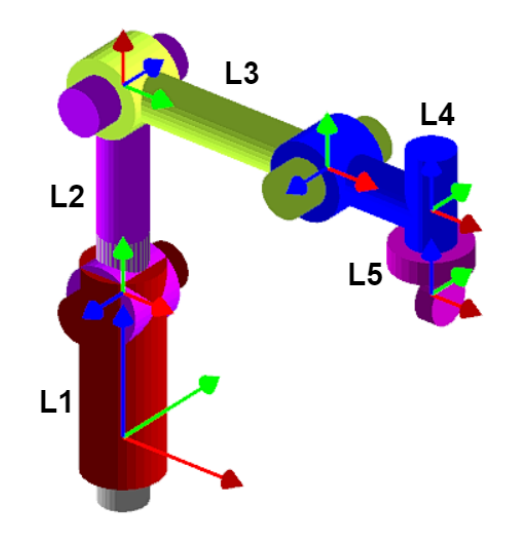


Figure 3.16: DH Reference Frames.

Once the reference frames were set, the study of the parameters followed which yielded the results demonstrated in Table 3.1, where the symbol * means the parameter is variable. The final two joints are not physical joints controlled by motors. They are present because of links 4 and 5 which are the offsets on x and y, respectively, for the end-effector.

Having all the parameters ready, calculating the transformation matrix is simply a matter of calculating every transformation matrix for each joint i , A_i , according to equation 2.5, followed by

equation 2.6.

Joints	θ_i	d_i	a_i	α_i
1	θ_1^*	L_1	0	$\frac{\pi}{2}$
2	$\theta_2^* + \theta_{offset}$	0	L_2	π
3	$\theta_3^* + \theta_{offset} + \theta_2^*$	0	L_3	$-\pi$
4	$\theta_4 = \theta_3^*$	0	L_4	$-\frac{\pi}{2}$
5	0	$-L_5$	0	0

Table 3.1: DH Parameters.

Using a custom made Matlab script which calculates the final transformation matrix, T, resulted

$$T = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & \cos(\theta_1)(L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3) + L_4) \\ \sin(\theta_1) & \cos(\theta_1) & 0 & \sin(\theta_1)(L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3) + L_4) \\ 0 & 0 & 1 & L_1 + L_2 \sin(\theta_2 + \theta_{offset}) - L_3 \sin(\theta_3) - L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

With this transformation matrix it's possible to acquire the translation matrix, P, from it:

$$P = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} \cos(\theta_1)(L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3) + L_4) \\ \sin(\theta_1)(L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3) + L_4) \\ L_1 + L_2 \sin(\theta_2 + \theta_{offset}) - L_3 \sin(\theta_3) - L_5 \end{bmatrix} \quad (3.7)$$

Because we know from the configuration of the robotic manipulator that the orientation of the end-effector is always towards the ground, there is no need to study the rotation matrix.

These results are coincident with those derived earlier when using the geometrical analysis.

3.1.5 Inverse Kinematics

3.1.5.1 Law of Cosines

This subsection is intended to display a trigonometrical law that will be important for the study of the Inverse Kinematics.

Given the notation used in Fig. 3.17, the Law of Cosines permits one to relate the lengths of the sides of a triangle to one of it's angles using the following equations

$$\begin{aligned} a^2 &= b^2 + c^2 - 2bc \cos(\alpha) \\ b^2 &= a^2 + c^2 - 2ac \cos(\beta) \\ c^2 &= a^2 + b^2 - 2ab \cos(\gamma) \end{aligned} \quad (3.8)$$

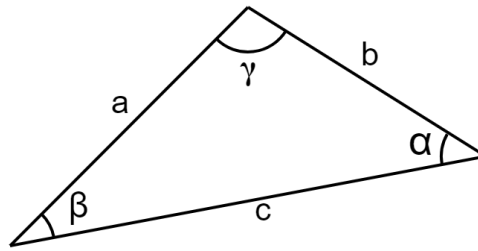


Figure 3.17: Notation used for Law of Cosines.

This law is a generalization of Pythagoras' Theorem, in the sense that if the angle being study is a right angle, 90 degrees, then the Law of Cosines becomes the Pythagoras Theorem.

3.1.5.2 Geometrical Analysis

Fig. 3.18 shows a profile view of the manipulator. This figure contains the nomenclature used throughout this section.

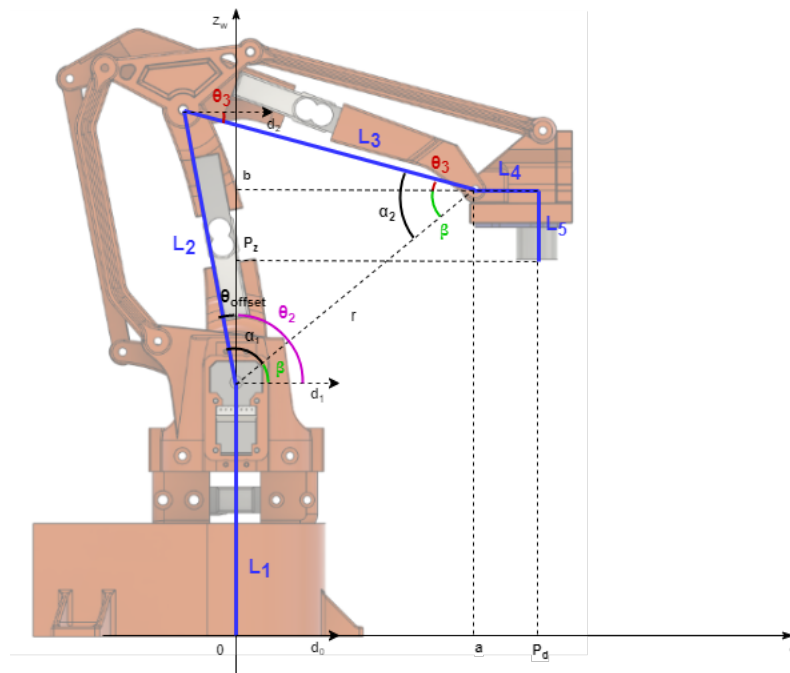


Figure 3.18: Inverse Kinematics Profile View of Robotic Arm.

In order to calculate the desired angles θ_2 and θ_3 it is first needed to calculate some auxiliary angles and distances. It's easy to see that a triangle is formed with the links 2 and 3 along with the side r . Using this triangle and the Law of Cosines mentioned before, it's possible to calculate the angles required. In order to do so, the distance r and angle β are required which can be calculated

as follows

$$r = \sqrt{(b - L_1)^2 + a^2} \quad (3.9)$$

$$\beta = \arctan\left(\frac{b - L_1}{a}\right) \quad (3.10)$$

where $a = P_d - L_4$ and $b = P_Z + L_5$.

Using the Law of Cosines we can calculate the values of the α variables and later calculate the desired angles for the second and third joints.

α_1 can be calculated like so

$$\begin{aligned} L_3^2 &= L_2^2 + r^2 - 2L_2r\cos(\alpha_1) \\ \cos(\alpha_1) &= \frac{L_3^2 - L_2^2 - r^2}{-2L_2r} \\ \alpha_1 &= \arccos\left(\frac{L_3^2 - L_2^2 - r^2}{-2L_2r}\right) \end{aligned} \quad (3.11)$$

and, similarly, α_2 can be calculated like so

$$\begin{aligned} L_2^2 &= L_3^2 + r^2 - 2L_3r\cos(\alpha_2) \\ \cos(\alpha_2) &= \frac{L_2^2 - L_3^2 - r^2}{-2L_3r} \\ \alpha_2 &= \arccos\left(\frac{L_2^2 - L_3^2 - r^2}{-2L_3r}\right) \end{aligned} \quad (3.12)$$

Finally, with the help of Fig. 3.18, it's easy to see that

$$\theta_2 = \alpha_1 + \beta - \theta_{offset} \quad (3.13)$$

$$\theta_3 = -(\alpha_2 - \beta) \quad (3.14)$$

The reason why θ_3 is negative, is because the angle in joint 3 is positive in the counterclockwise direction.

Last but not least, the calculation of θ_1 is done as follows

$$\theta_1 = \arctan\left(\frac{P_y}{P_x}\right) \quad (3.15)$$

Because the orientation of the tool is always pointing downwards to the floor plane, the study of Inverse Kinematics, in this case, is simply focused on the position of the end-effector rather than its orientation.

3.1.6 Jacobian

The Jacobian matrix of this manipulator's configuration can be calculated using the equations of the forward kinematics P_x (3.3), P_y (3.4) and P_z (3.5).

Considering that the pose of the manipulator, for a certain state q , is calculated using the following equation f

$$\mathbf{f}(q) = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.16)$$

where ϕ represents a rotation in the origin x axis, θ a rotation in the origin y axis and ψ a rotation in the origin z axis, it is possible to determine the Jacobian matrix using the following equation

$$J(q) = \frac{\partial \mathbf{f}(q)}{\partial q} = \begin{bmatrix} -s(\theta_1)Q & -L_2s(\theta_2 + \theta_{offset})c(\theta_1) & -L_3c(\theta_1)s(\theta_3) \\ c(\theta_1)Q & -L_2s(\theta_2 + \theta_{offset})s(\theta_1) & -L_3c(\theta_1)s(\theta_3) \\ 0 & L_2c(\theta_2 + \theta_{offset}) & L_3c(\theta_3) \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.17)$$

where $c(x)$ is the cosine of x , $s(x)$ is the sine of x and $Q = (L_4 + L_2 \cos(\theta_2 + \theta_{offset}) + L_3 \cos(\theta_3))$.

3.2 LX-16A Servo Motor

The motors used in this project are electric of type Servo. The model chosen was the LX-16A servo from LewanSoul, Fig. 3.19. This product is comprised of the motor, a servo drive, a serial bus communication interface and a potentiometer. It allows for 2 modes, continuous rotation or position control.

In position control, it is possible to operate between 0 and 240 degrees and in continuous rotation it is possible to adjust the speed, both clockwise and counter-clockwise. Each servo is setup to have an Identity number (ID). It is defaulted to 1 but can be changed by the user according to their needs. Some specifications for this model are shown in Table 3.2. Furthermore, the servo includes a Light Emitting Diode (LED) that is used to signal possible operation errors.

The control is made via serial communication through a single bus mode with a baud rate of 115200 bits per second. This communication uses a protocol based on custom command packets.

Net Weight	52g
Rotation Speed	0.16sec / 60degree (@7.4 Volt)
Servo Accuracy	0.24 degrees
Torque	17 Kg.cm (@7.4 Volt)
Control Method	Serial Command (Baud rate - 115200bps)
Parameter Feedback	Temperature, Voltage, Position

Table 3.2: Motor Parameters.

The format for these packets is presented in Table 3.3. Because of this way of communicating and controlling the motors, it is possible to connect several of them in series and control each one using their proper IDs in the command packet. This is extremely useful because only three pins (for the serial connection) are needed to control several motors.

Header	ID Number	Data Length	Command	Parameter	Checksum
0x55 0x55	ID	Length	Cmd	Param 1 ... Param N	Checksum

Table 3.3: Servo Command Packet Format.

3.2.1 Command Packet

A valid packet must include a Header which is two consecutive 0x55 indicating the beginning of a packet, followed by the ID Number of the servo(s) to which the message is supposed to be sent to. This ID is an 8 bit number ranging from 0 to 253 and the ID number 254 is the broadcast ID, which means every servo will process that packet.

Next is the Data Length, which is an 8 bit value indicating the length of the data that is to be sent, in bytes (including the data length byte). This value is equal to the size of the whole command packet minus three (two bytes for the header and one byte for the checksum).

The Command is a one byte number and it represents the instruction the servo is supposed to execute, such as requesting the position of the motor or changing its angle limits. There are two types of commands, read and write.



Figure 3.19: LewanSoul's LX-16A.

Write commands normally need parameters whereas read commands usually do not. When a read command is sent to the servo, it is followed by a command packet sent from the servo to the controller, which needs to be read and processed because it contains the information requested by the read command.

Finally the Checksum is calculated as follows:

$$Checksum = \sim (ID + Length + Cmd + Param1 + \dots + ParamN) \quad (3.18)$$

where $\sim$ means negation.

Considering the command *SERVO_ID_WRITE* which has a value of 13 (0x0D) and requires one parameter: the ID to be set (one byte), the length of the packet is four bytes (one for the ID, one for the Length, one for the Command and one for the Parameter). This command is used to change the current ID of a servo. If the ID to be set is 10 (0x0A), for example, and the servo's initial ID is 2 (0x02), then the checksum would be calculated like so:

$$Checksum = \sim (2 + 4 + 13 + 10) = \sim 0x1D = 0xE2 \quad (3.19)$$

Once every part of the command packet is calculated and acquired, it is possible to construct the command packet to be sent:

Header	ID Number	Data Length	Command	Parameters	Checksum
0x55 0x55	0x02	0x04	0x0D	0x0A	0xE2

Table 3.4: Example Command Packet.

3.3 Load Cells

A load cell is a device widely used to measure force and torque. These sensors can be very accurate and reliable if well designed and are used in numerous fields for weighing products like food, animals, vehicles and many others [18].

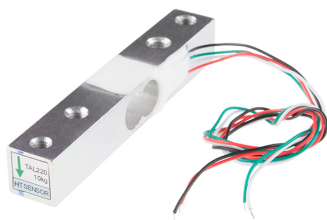


Figure 3.20: TAL220 Load Cell Sensor.

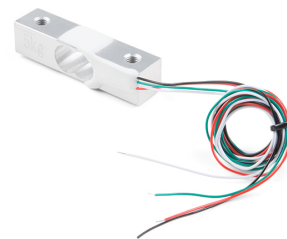


Figure 3.21: TAL220b Load Cell Sensor.

The way a load cell works is by utilizing the Poisson effect, which is when an object contracts or expands when subjected to mechanical stress, along with strain gauges. A strain gauge senses the mechanical changes in the surface of the load cell and its electrical resistance varies proportionally to the force applied [18].

Load cells like the TAL220 and TAL220b, Fig. 3.20 and Fig. 3.21 respectively, can only measure force in one direction, the same direction of the threaded holes.

These load cells use a Wheatstone bridge, like the one in Fig. 3.22, however all of the resistors are strain gauges and are therefore variable depending on the deformation of the material. When a force is applied on the load cell, the resistors vary and so does the voltage differential between V_{out+} and V_{out-} . This differential is then measured and can be translated into force because it is proportional to the amount of force being applied.

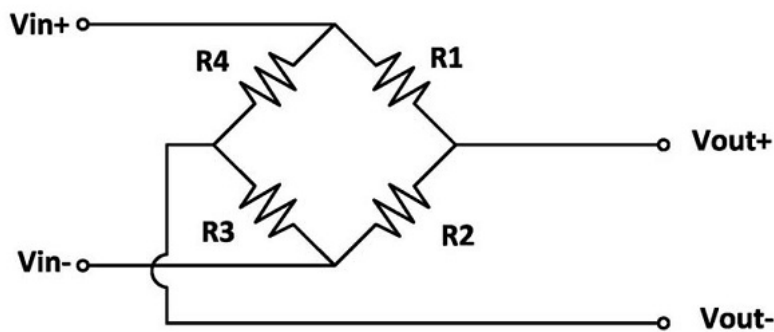


Figure 3.22: Wheatstone Bridge Circuit³.

This design requires some signal conditioning. Mainly because the output voltage of the Wheatstone bridge has a very small sensitivity and therefore the signal should be amplified in order to have a proper reading. In other words, the variation of voltage read in the output of the bridge is very little (in the order of mV) and so amplification is necessary.

A commonly used module for this task is the HX711⁴, Fig. 3.23, which is a 24-bit Analog-to-Digital Converter (ADC) with a refresh rate of 10 or 80Hz without the need of an external clock. This device communicates and is controlled by a microcontroller via Serial Peripheral Interface (SPI).

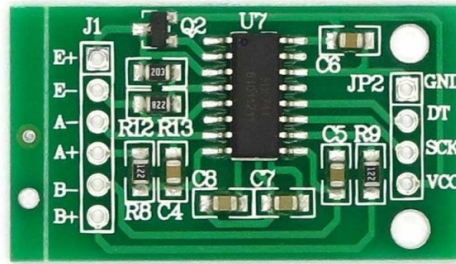
3.3.1 Calibration

When using the HX711 module to measure weights with the load cell, it is important to calibrate the sensor. This is usually done via software because not every cell is exactly the same and therefore some sort of calibration needs to take place in order to produce accurate measures.

The library used in the Arduino Mega was developed by *bodge* and can be found on his GitHub⁵ page. When using this library, the calibration process is done by taring the cell and

³Source: <https://www.utlcell.com/en/load-cell-electrical-circuit-/>

⁴Source: <https://tinyurl.com/yxpxlksz>

Figure 3.23: HX711 ADC⁴.

then placing a known weight on it. After this, the scale factor needs to be repeatedly changed until the measure obtained corresponds to the weight being measured.

In order to facilitate the calibration process, two pieces were modeled in *Fusion 360*, and then 3D printed. Fig. 3.24 demonstrates this setup where the weight being measured is of a container filled with water.



Figure 3.24: Custom Load Cell Calibrator.

The code from the library used converts a value read from the HX711 ADC into a weight measure using the following formula

$$Measure = \frac{Read - Offset}{ScaleFactor} \quad (3.20)$$

In this formula there are two constants that allow the user to adjust the resulting measure, the *Offset* and the *ScaleFactor*.

Knowing this formula is important because it makes the calibrating process much easier and faster since we can simply calculate the scale factor when measuring a known weight.

⁴Source: <https://github.com/bogde/HX711>

After doing this procedure for all load cells used in both the local and remote manipulators, the resulting scale factors are displayed in the following Table 3.5.

Manipulator	Link 1	Link 2	Link 3
Local	1081.81	1094.60	1112.46
Remote	1103.04	909.33	1131.34

Table 3.5: Load Cells' Scale Factors.

3.4 Arduino Mega 2560

The microcontroller used for this project was the Arduino Mega 2560⁶ which is a device developed by Arduino and based on the ATmega2560, Fig. 3.25. This board includes 54 digital input/output pins (15 of which have Pulse Width Modulation (PWM) possibility), 16 analog input pins, 4 hardware serial ports and a crystal oscillator of 16 MHz, among other specifications.

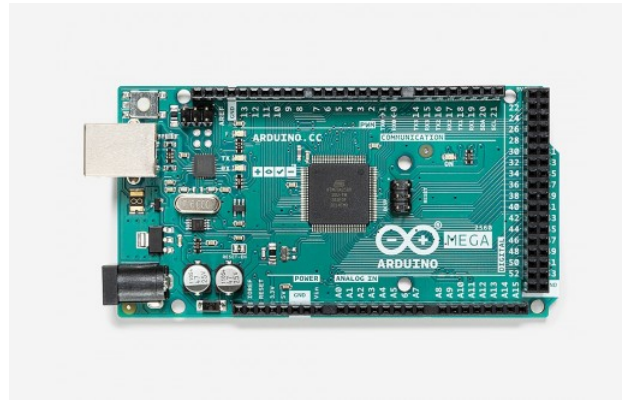


Figure 3.25: Arduino Mega 2560⁶.

The main reason for the choice of this microcontroller compared to others like the Arduino Uno⁷ was the possibility of having more than one serial communication. This is an important aspect because the servos used to actuate the manipulators require serial communication for their control (as was explained before) and the communication with the computer is done via serial communication as well. Hence there is the need of having more than one serial port available, which is not possible with the Arduino Uno.

3.4.1 Visual Studio Code Editor

The IDE used to develop the software for the microcontroller was *Microsoft's Visual Studio Code*⁸. This program is a free, open-source code editor with possibility of customization due to the extension packs available for download.

⁶Source: <https://store.arduino.cc/arduino-mega-2560-rev3>

⁷Website: <https://store.arduino.cc/arduino-uno-rev3>

⁸Website: <https://code.visualstudio.com/>

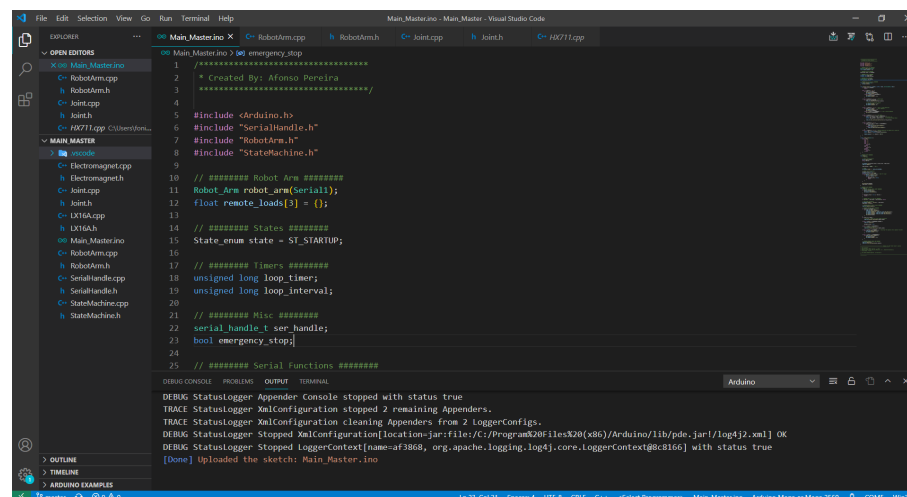


Figure 3.26: Visual Studio Code Editor.

This editor is equipped with an extension that allows their users to program and upload sketches to Arduino-like microcontrollers. This is great because it is also equipped with IntelliSense, unlike the Arduino IDE, which makes developing these programs faster and easier.

3.5 Conclusion

In this chapter a quick overview of the hardware tools used throughout the project was made, like the servo motors and load cells. It was also demonstrated all the changes that were required to make to the original model of the robotic arm, *EEZYbotArm Mk2*, in order to develop a manipulator more suitable to have haptic force feedback. Furthermore, a theoretical study of the kinematics of the modified model was done.

Chapter 4

Proposed Software Architecture

The aim of the chapter that follows is to demonstrate and explain the software side of the project developed.

The first section highlights the architecture used for the systems, subsystems and components. Next, a brief description of the communication protocols used takes place, followed by the control schemes.

The chapter finishes off with a thorough explanation of the GUIs created for both manipulators.

4.1 System Architecture

The system architecture used for this project is presented in Fig. 4.1. The system is composed of two main subsystems: the local subsystem and the remote subsystem.

Both subsystems are very similar to one another in terms of components with the exception of a camera sensor which is used in the remote subsystem and not in the local one. This camera is connected via USB and is used to provide visual feedback of the remote environment to the local system and thus give a better experience to the user operating the system.

Each subsystem along with the relations between them will be explained in the following subsections.

4.1.1 Manipulator Subsystem

The robotic manipulator subsystem, apart from the 3D printed parts which were explained in the previous chapter, consists of:

- One Arduino Mega 2560;
- One Bus Servo Controller;
- Three LX16-A Servo Motors;
- One Electromagnet;

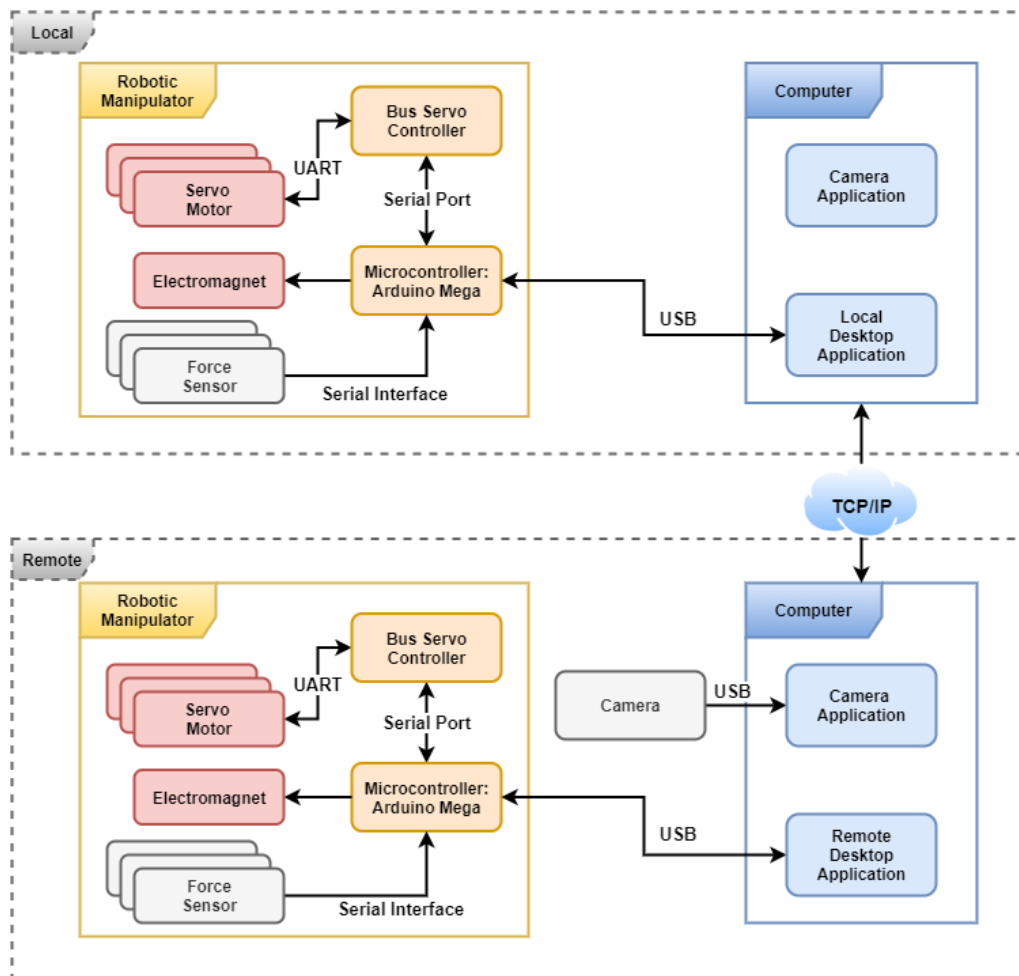


Figure 4.1: System Architecture.

- Three Force Sensors:
 - One Load Cell;
 - One HX711 ADC.

and each component relates to one another as shown in Fig. 4.2. Fig. 4.3 demonstrates a diagram with the connections made to the Arduino Mega and the full circuit diagram is included in Appendix A.1.

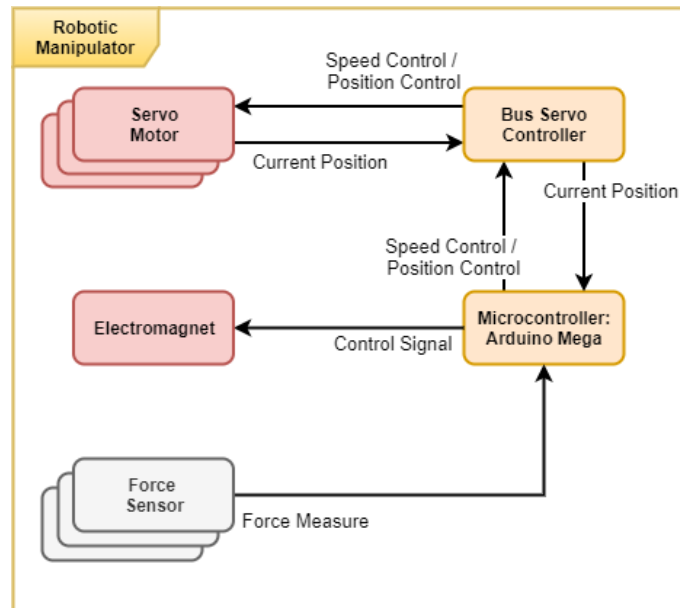


Figure 4.2: Robotic Manipulator Architecture.

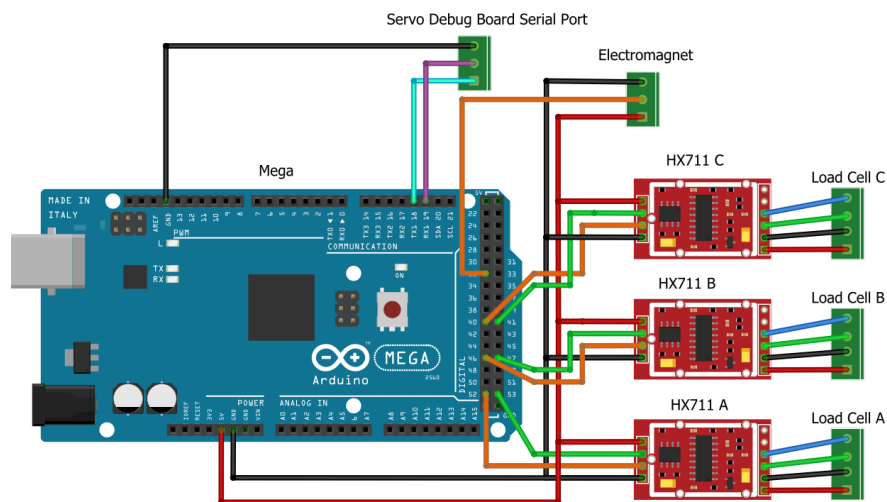


Figure 4.3: Bread Board Circuit Diagram.

The Arduino Mega is the brains of the manipulator. It is in charge of operating the actuators, the electromagnet and the servo motors, and of reading the sensors, the load cells. Furthermore,

it is also responsible for executing the control loop of the robotic arm and is used to communicate all this information with the computer.

The microcontroller reads force measures from the load cells with the help of the HX711 ADC at a frequency of 80Hz which could be a limitation. However, because the control loop is set to execute every 40 milliseconds, which is 25 Hz, this is not a problem since it reads only 1 measure for each cell every control loop.

The electromagnet requires a control signal to operate. This control signal is sent by the microcontroller and is a simple ON/OFF type control.

Finally to control the servo motors, a controller board is used which interfaces with the Arduino Mega via serial port. This is connected to Serial Port 1 of the microcontroller and the control is made using command packets as explained in the previous section 3.2.1. The information retrieved from each servo motor is their current position and the messages sent are related to their control, either speed or position.

4.1.2 Manipulator to Computer Connection

As mentioned before the microcontroller is responsible for the communication between the manipulator and the computer application. This is done using a USB cable that connects the Arduino Mega to a computer port and thus a serial communication is created.

The information passed from one subsystem to the other is practically the same for both the local and remote systems, as seen in Fig. 4.4.

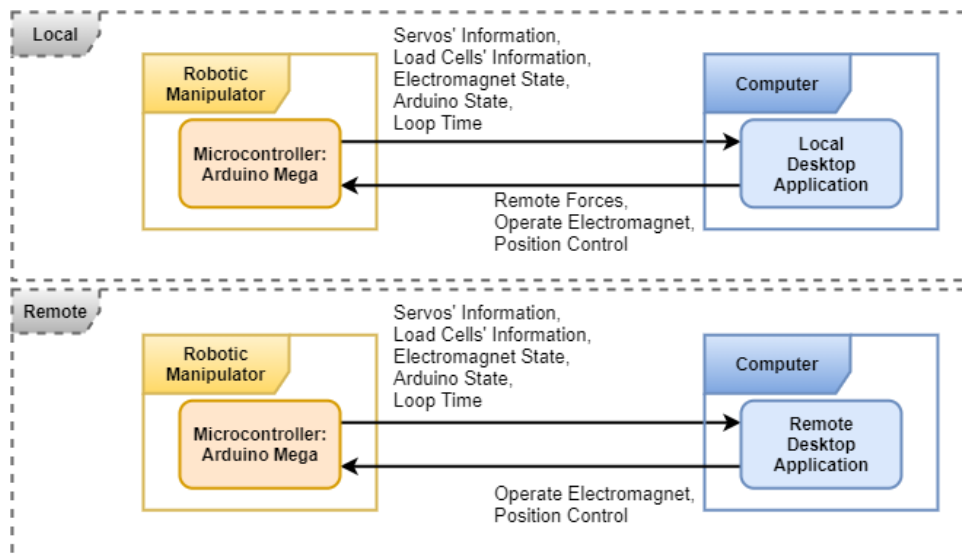


Figure 4.4: Robotic Manipulator to Computer Architecture.

In both systems the microcontroller sends information about the servos and the load cells along with the current state of the electromagnet (either active or not). Moreover, the state of the Arduino Mega and the time it took to execute the last control loop is also sent as a way to help debug the microcontroller.

Also in both systems, the desktop application sends commands to the microcontroller for position control and to operate the electromagnet. However, in the local system, the desktop application also sends the values of the remote load cells which is then used by the microcontroller to implement the haptic feedback.

4.1.3 Computer to Computer Connection

Finally, the relation between both computer subsystems is done via Transmission Control Protocol (TCP)/ Internet Protocol (IP), Fig. 4.5.

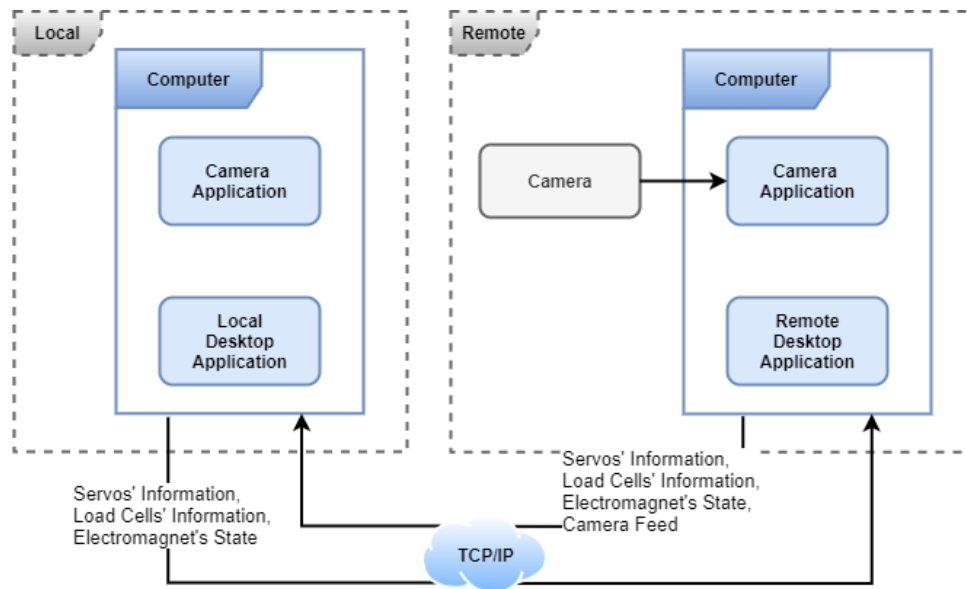


Figure 4.5: Computer to Computer Architecture.

The information passed over this medium from both subsystems consists of the information about the servos and the force sensors' measures along with the electromagnet's state. Apart from this information, the remote subsystem also sends data to the local subsystem about the camera feed it is capturing, which is then displayed by in an application.

4.2 Communication Packets

There are two types of communication in this project: the communication between the microcontroller and the computer program; and the communication between the computer handling the local robotic arm and the computer handling the remote robotic arm.

The communication between the microcontroller and the computer is done via serial communication with a baud rate of 115200 bps, 8 data bits, no parity and 1 stop bit. Whereas the communication between both computers is done using TCP over Internet.

The protocol used for the communication was using Channels, which was developed by Professor Paulo Costa. This library is available on his GitHub repository for Free Pascal.

In this method a data frame is composed of an alphabet character, which is called channel, followed by characters in hexadecimal notation which form one or more integer values, depending on the channel. Because these values are transmitted in hexadecimal form (which goes from 0 to F), the channel can only be a letter from "G" to "Z" in the alphabet, either capitalized or not. This allows for a grand total of 40 different channels to be defined.

This method of communication is somewhat inefficient because for each character sent, which is in fact an 8-bit value, only 4 bits of information, also referred to as 1 Nibble, are actually being transmitted. This is because in hexadecimal representation each character translates to 4 bits of information. However, because the beginning of any data packet is a character greater or equal than "G", it is easier to detect errors when receiving data packets.

In order for this method to fit the needs of the project, some modifications were made to this library. One of these changes was the implementation of variable packet sizes. The size of the packet is dependent on the channel being transmitted which must be specified for each channel. Both sides of the stream must have the same value for the length of a channel in order for the communication to work properly.

For example, when a data packet sent from the microcontroller to the computer has the channel "M", the expected amount of information in that packet is 88 bits or 22 Nibbles/Hexadecimal characters. This specific channel contains information about one Joint as shown in the table 4.1.

Channel "M"	Joint ID	Current Angle	Desired Angle	Actuation	Force Measure
Size	2 Nibbles	4 Nibbles	4 Nibbles	4 Nibbles	8 Nibbles

Table 4.1: Channel "M" Data Packet Information.

Note that the unit for the angles is centidegrees, the actuation represents the speed value, and the force measure is in milligrams-force.

For instance, if the current information of a Joint with an ID of 1 was the following:

- Current Angle = 12000 centidegrees;
- Desired Angle = 20000 centidegrees;
- Actuation = 0;
- Force Measure = 5000 milligrams-force;

then the message sent would be the one present in table 4.2.

	Joint ID	Current Angle	Desired Angle	Actuation	Force Measure
M	01	2EE0	4E20	0000	00001388

Table 4.2: Channel "M" Data Packet Example.

In order to properly parse information back and forth between the microcontroller and the computer as well as between both computers, some channels were defined. The complete list of

channels can be found in Appendix B which is split into several tables. Each table demonstrates the list of channels used along with it's size and a simple description of the information passed.

Fig. 4.6 demonstrates two different communication sequences which includes the channels sent from each subsystem to the others. Sequence a) occurs every control loop of the local microcontroller whereas sequence b) occurs every control loop of the remote microcontroller.

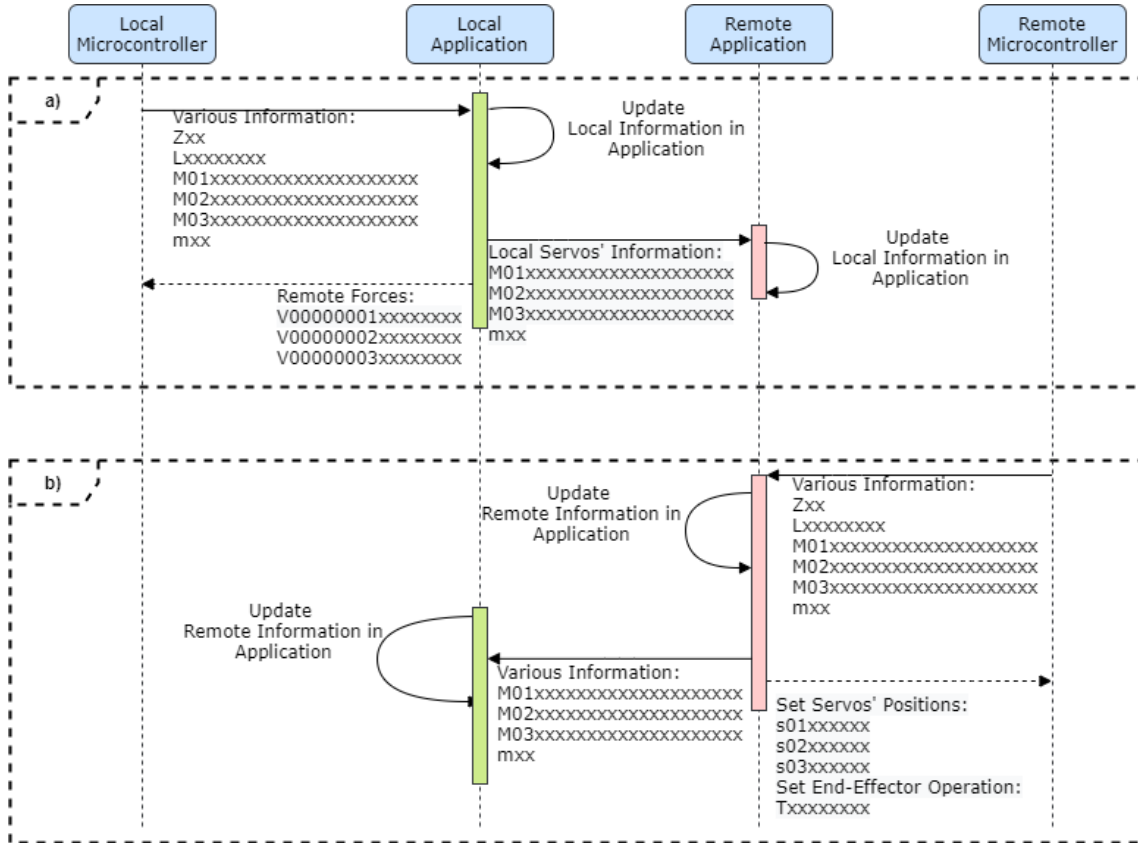


Figure 4.6: Information Sequence Diagram.

In sequence a) the microcontroller begins by sending information about the Arduino (channels Z and L) along with information on the joints and electromagnet (channels M and m). Once the local application receives this information it updates some internal variables and sends the information on the joints and electromagnet to the remote application. The remote application then updates it's own internal variables which are later used. Finally the local application sends information on the remote forces (channel V) from each remote load cell to the local microcontroller.

Sequence b) is quite similar to sequence a) except for the last step. In this sequence instead of sending the values of the remote forces, the remote application sets the joint's positions to the local positions (channel s) and does the same for the electromagnet (channel T).

4.2.1 Initial Communication Sequence

At startup, when the Arduino first establishes the connection with the application, an initial communication sequence takes place, presented in Fig. 4.7. Similarly, when both applications

(local and remote) first establish a connection, an initial sequence is also present, shown in Fig. 4.8.

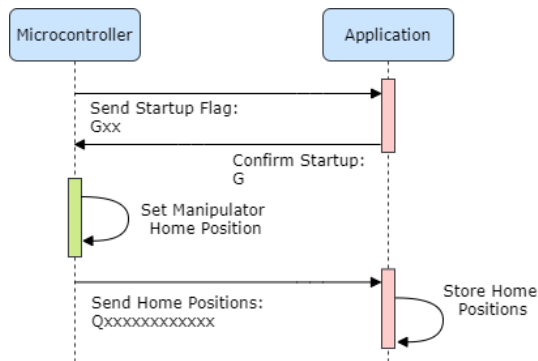


Figure 4.7: Initial Sequence between microcontroller and PC.

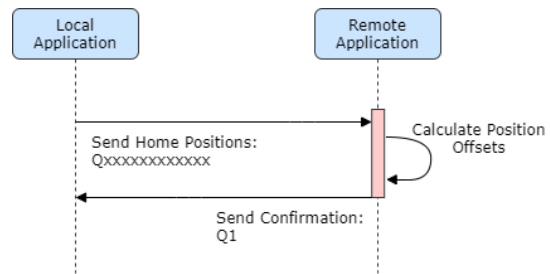


Figure 4.8: Initial Sequence between local PC and remote PC.

The initial sequence between the microcontroller and the computer, 4.7, begins with the Arduino sending a message with the channel "G" that is followed by the number of joints used in the robotic arm being controlled. This message is processed by the computer which then sends the confirmation of receiving that channel with a "G" as well.

After this the microcontroller sets the robotic arm in its home position and proceeds to send the message with the channel "Q" to the computer which includes the home position of each joint. This is important because when building each robotic arm, the angles of the motors didn't exactly match from one to the other. So, having the home positions of each robotic manipulator means it's possible to calculate the offsets from one to the other.

This is what happens when both computer applications initialize a connection, seen in Fig. 4.8. The local application begins by sending the home positions of the robotic arm via the channel "Q". Which the remote application receives and then calculates the offsets between both manipulators and ends by sending the confirmation with a "Q1" message.

4.3 Force Control

The control of the local robotic arm was done using the force measurements from the load cells. Since the manipulator was developed with three load cells, one for each joint, it is possible to measure the force applied to each cell and produce a force proportional to it on the joint. That way the user simply needs to physically move the end-effector to the position they desire.

When testing the measurement of forces using the load cells, it was noticeable that, from time to time, a faulty measurement was made. This measurement always resulted in a very high value, and in order to attenuate this noise, a median filter was implemented. The library used is available on GitHub and was created by the user *luisllamasbinaburo*¹.

¹Source: <https://github.com/luisllamasbinaburo/Arduino-MedianFilter>

Given the limitations of the hardware of the Arduino Mega, a filter with a size of 3 was implemented, which resulted in a good outcome and removed the occasional noisy measurements.

The formula used to relate the force measured in the load cell and the actuation, or speed, desired in the servo was the following

$$actuation = \tanh\left(\frac{ForceMeasured}{K}\right) MaxActuation \quad (4.1)$$

which can be seen plotted in Fig. 4.9.

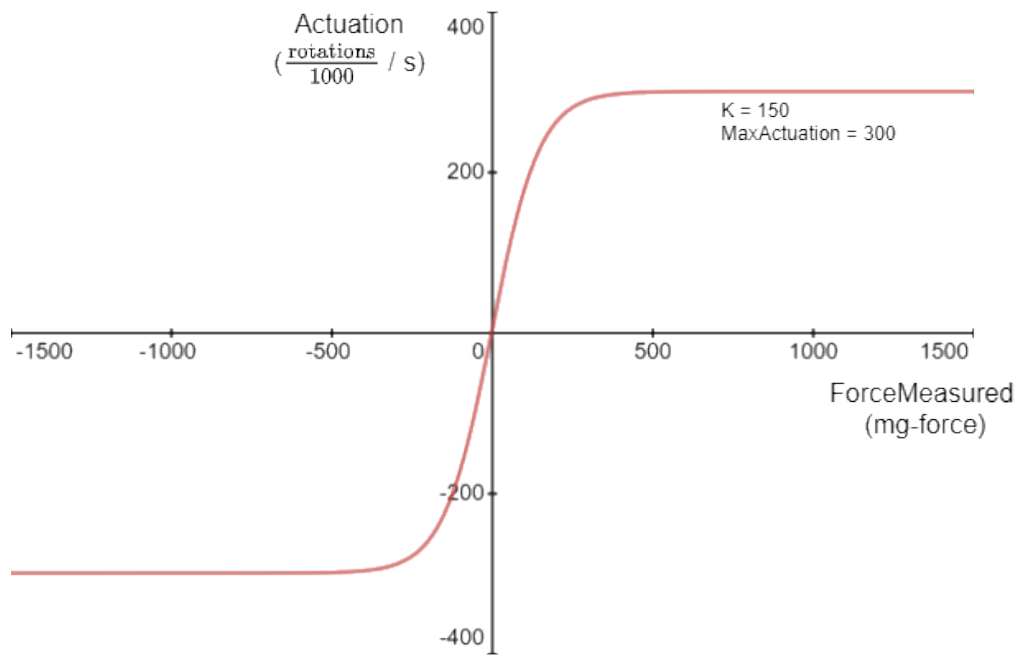


Figure 4.9: Actuation calculation formula plotted.

The reason the hyperbolic tangent function was used as opposed to a simple threshold, for example, was to get a smoother response of the movement of the robotic arm. The constant K allows for the adjustment of the function in order to acquire the best suitable response and control. Finally, the Max Actuation parameter simply provides the upper and lower limits for the function and can be used to control how fast that joint is allowed to move.

Once the value of the actuation is calculated, there comes the need to control the motor to move with that actuation value. This is possible to do using one of the many available commands called `SERVO_OR_MOTOR_MODE_WRITE`, which has 4 parameters: Servo mode, null value, lower 8 bits of rotation speed value and finally higher 8 bits of rotation speed value.

Essentially this command allows the user to define the control mode of the motor, either position control mode or motor control mode. For the purpose at hand, the control mode needed is the motor control, which uses the 16-bit value of rotation speed parameter to set the speed of the motor. This parameter has a range of -1000 to 1000 where 1000 corresponds to 1 full clockwise rotation per second.

4.3.1 End-Effector Weight Adjustment

With the maneuvering of the robotic arm, the load cells will have different loads depending on the angle each joint currently has. In link 3, for example, when the angle of joint 3 varies, so does the projection of the weight of the end-effector on the load cell. This is a problem because as mentioned before, the forces applied to the load cells is what drives the motors and moves the manipulator. In other words, in some cases the manipulator could move by itself only based on the current pose of the manipulator.

To resolve this issue, the measured force on each load cell was updated based on the angle the joint relative to that sensor currently has.

Because the load cell used for operating joint 1 isn't affected by the weight of the manipulator, no update was made to that measure. For the remaining two load cells, however, this update was necessary.

Two different approaches were taken to update the load cells for link 2 and link 3, which will be explained in the following sections.

4.3.1.1 Link 3 Load Cell

The approach taken to adjust the load cell measures was by projecting the weight of the end-effector on the load cell. This projection is required because the force sensors only measure forces that are perpendicular and in the correct direction, as mentioned before.

Fig. 4.10 demonstrates this projection on the load cell.

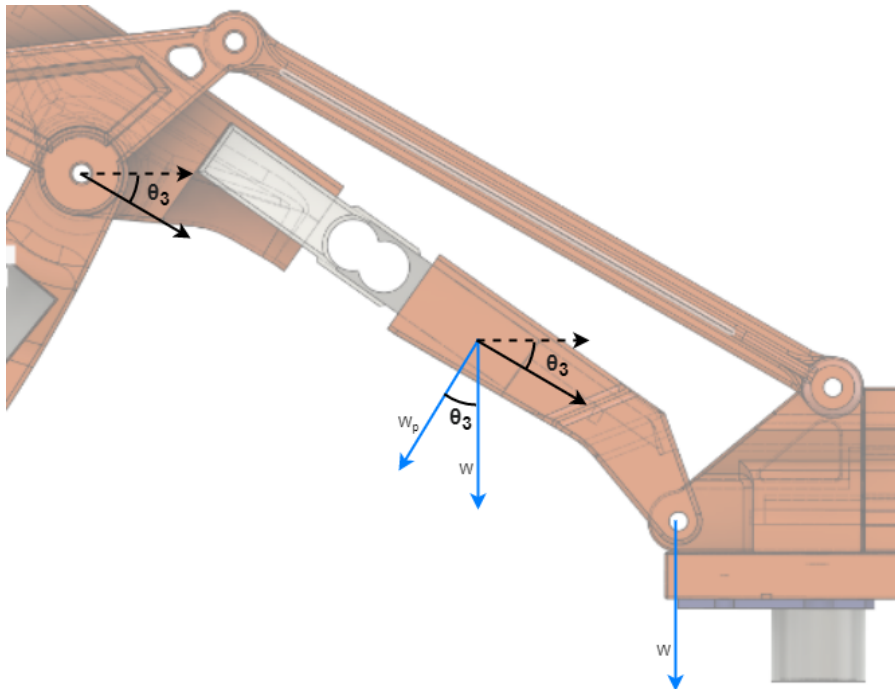


Figure 4.10: Weight Adjustment Diagram for Link 3.

Using trigonometry it is straightforward to see that the projected weight on the sensor can be calculated as follows

$$W_p = W \times \cos \theta_3 \quad (4.2)$$

where W is the weight of the end-effector and W_p the projected weight on the load cell.

This projection is calculated every time a measurement is taken, in other words every control loop, which is then used to calculate the updated measure.

4.3.1.2 Link 2 Load Cell

Regarding Link 2 the approach taken was practical instead of theoretical like with Link 3. For this load cell a measurement was taken, without any load and without actuating the motors, for 20 equally distant positions within the joint's possible range.

Having recorded all values, a graph of Force Measure vs Position was plotted to see if there was some sort of correlation that could be exploited. The graph for the local Link 2 can be seen in Fig. 4.11, and the same graph for the remote arm is displayed in Fig. 4.12. In these graphs the x axis is the position, in centidegrees, and the y axis is the force measured, in grams-force.

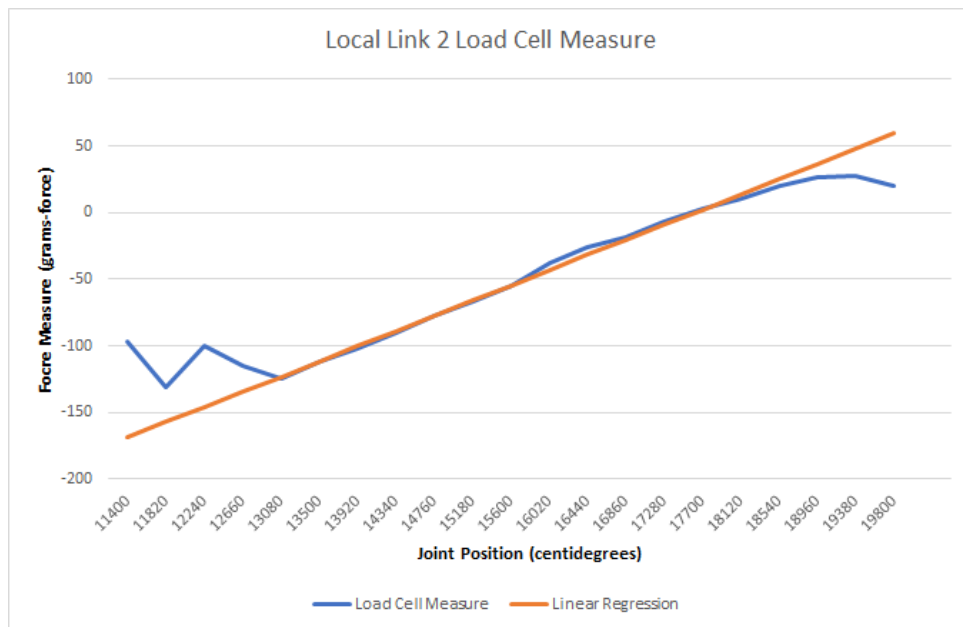


Figure 4.11: Local Link 2 Force Measure Values.

For the local weight adjustment, Fig. 4.11, there is a range where the points can be approximated by a linear function, from 13080 to 18540. After applying a linear regression to these points

(orange series in the figure), the following function was derived:

$$f(\theta) = \begin{cases} -115 & \text{if } \theta < 13080 \\ 0.02712193 \times \theta - 477.72622 & \text{if } 13080 \leq \theta \leq 18540 \\ 20 & \text{if } \theta > 18540 \end{cases} \quad (4.3)$$

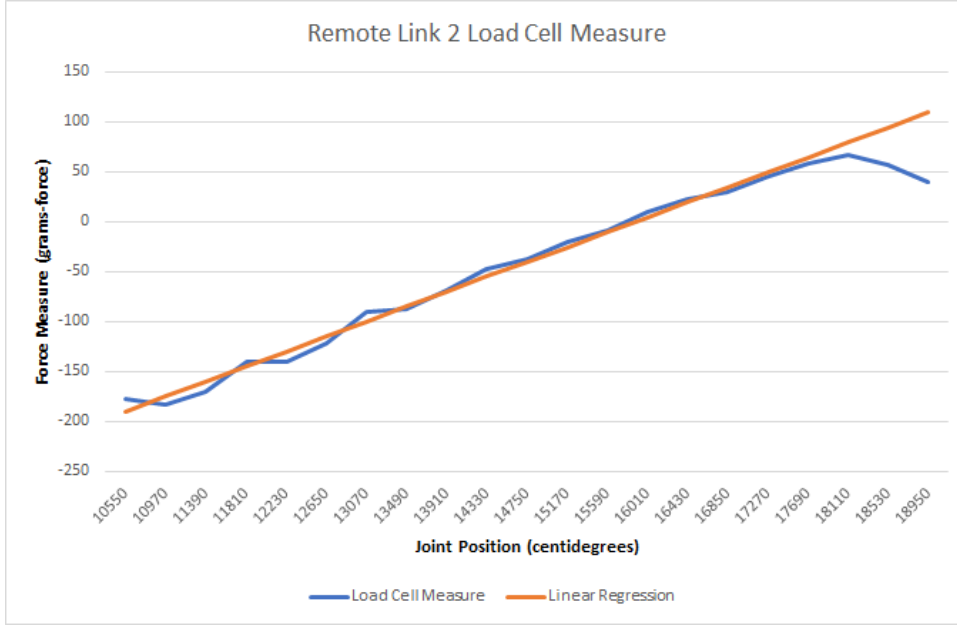


Figure 4.12: Remote Link 2 Force Measure Values.

Similarly to the local manipulator, the remote one too has a range of points where it is possible to approximate a linear function and thus achieve the following function:

$$f(\theta) = \begin{cases} 0.035564401 \times \theta - 564.8360116 & \text{if } \theta \leq 17690 \\ 50 & \text{if } \theta > 17690 \end{cases} \quad (4.4)$$

The values that result from these formulas are then subtracted to the measure taken each control loop to calculate the updated measure value.

4.4 Remote Control

To operate a remote manipulator, there comes the need to have some sort of control that moves that manipulator according to some input.

In this project, the input is the pose of the local manipulator, which is defined by the user, and the objective is to have the remote robotic arm replicate that position at any given moment. This control was done by knowing what angle each joint from the local manipulator currently has and setting those angles on the remote manipulator.

This is done using the command `SERVO_MOVE_TIME_WRITE` which has 4 parameters: lower 8 bits of angle value, higher 8 bits of angle value, lower 8 bits of time value and higher 8 bits of time value.

This command makes the servo motor rotate from its current angle to the angle parameter at constant speed within the time parameter. The angle parameter must have a range of 0 to 1000 which corresponds to a servo angle of 0 to 240 degrees and therefore the minimum angle the servo can move is 0.24 degrees. The time parameter is limited to between 0 and 30000 milliseconds. As soon as this command reaches the servo motor, the servo will begin rotating immediately.

4.5 Haptic Force Feedback

The system is equipped with haptic force feedback. This is achieved using the servo motors that control the joints along with the load cells present in the remote robotic manipulator.

The force values measured from the load cells of the remote arm are sent to the local arm's control software which then processes this information taking it into consideration when actuating the motors.

In every control loop of the local arm, an updated value for the load cell's measure is calculated if the values taken from the remote load cells have different signs from the local ones. This essentially means that there are forces being applied in both arms opposing one another and therefore some sort of haptic feedback must take place.

The formula used simply adds both values if their signs are different. However, if the sum of these values crosses from one sign to another, in other words reaches or passes zero, then the updated value is forced to zero. This is important because otherwise the local manipulator could be controlled by forces applied on the remote manipulator and this is not an expected feature.

The previous formula is used when the remote force on a joint is less than 500 grams-force. Otherwise, the updated load cell value is automatically set zero.

The following algorithm in pseudo-code demonstrates this procedure

```
if RemoteLoadMeasure * LocalLoadMeasure < 0 then
  if abs(RemoteLoadMeasure) > 500 then
    UpdatedCell ← 0
  else
    UpdatedCell ← UpdatedCell + RemoteLoadMeasure
    if Updatedcell * RemoteLoadMeasure > 0 then
      UpdatedCell ← 0
    end if
  end if
end if
```

where *RemoteLoadMeasure* is the measure acquired from the remote load cell, *LocalLoadMeasure* is the measure acquired from the local load cell and *UpdatedCell* is the final updated value used to actuate the servo.

This updated force value is then used in equation 4.3 to calculate the actuation for each joint of the local manipulator. These values are then applied to the motors using the same command used before, `SERVO_OR_MOTOR_MODE_WRITE`.

However, if the value of the actuation calculated is zero then the motor mode is set to position control and the current position is fixed using the command `SERVO_MOVE_TIME_WRITE`. This means that there is either a force being applied to the remote manipulator greater than the force being applied to the local manipulator or there is no force being applied to local manipulator.

4.6 Graphical User Interface

In order to easily control and debug both robotic manipulators, local and remote, a GUI was developed for each.

4.6.1 Lazarus IDE

The GUIs were developed using the application Lazarus² which is a cross-platform Integrated Development Environment (IDE) for Free Pascal. Because Free Pascal is a compiler that runs on Windows, Linux, MacOS, FreeBSD and others, there is no need to re-code anything to produce the same product in all these different operating systems.

This application proved to be very simple and easy to use thanks to the Source Editor, shown in Fig. 4.13, because it allows the user to manually insert and move around components.

4.6.2 Local Application

The local application is intended to be used by the user controlling the local robot arm. This application is expected to have information about both manipulators, some sort of debugging and further control options not available on the physical arm itself. Having said that, the main aspect of the application was divided into three different pages, a Control Page, a Debug Page and a Graph Page. In addition to these pages, there is also a Main Menu, a Toolbar, a Status Bar and some indicators as seen in Fig. 4.14.

On the top side of the application, the Main Menu and Toolbar are used mainly to configure the communication settings and to connect and disconnect from the microcontroller. There is also an option to clear the debug Memo which is found in the Debug Page.

On the bottom side of the application, there are indicators for the communication state with the microcontroller as well as its current state and last loop time. There are also 3 buttons used to connect and disconnect to the remote application and to Quit and close the application.

²Website: <https://www.lazarus-ide.org/index.php?page=about>

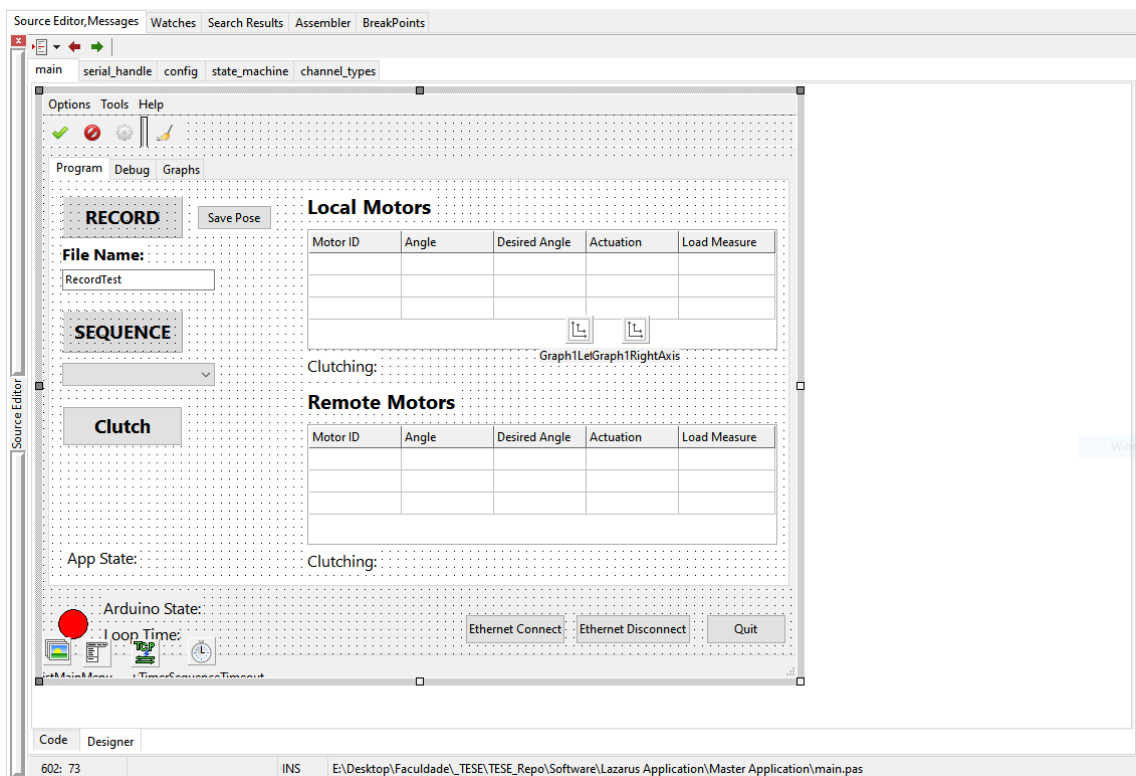


Figure 4.13: Lazarus' Source Editor.

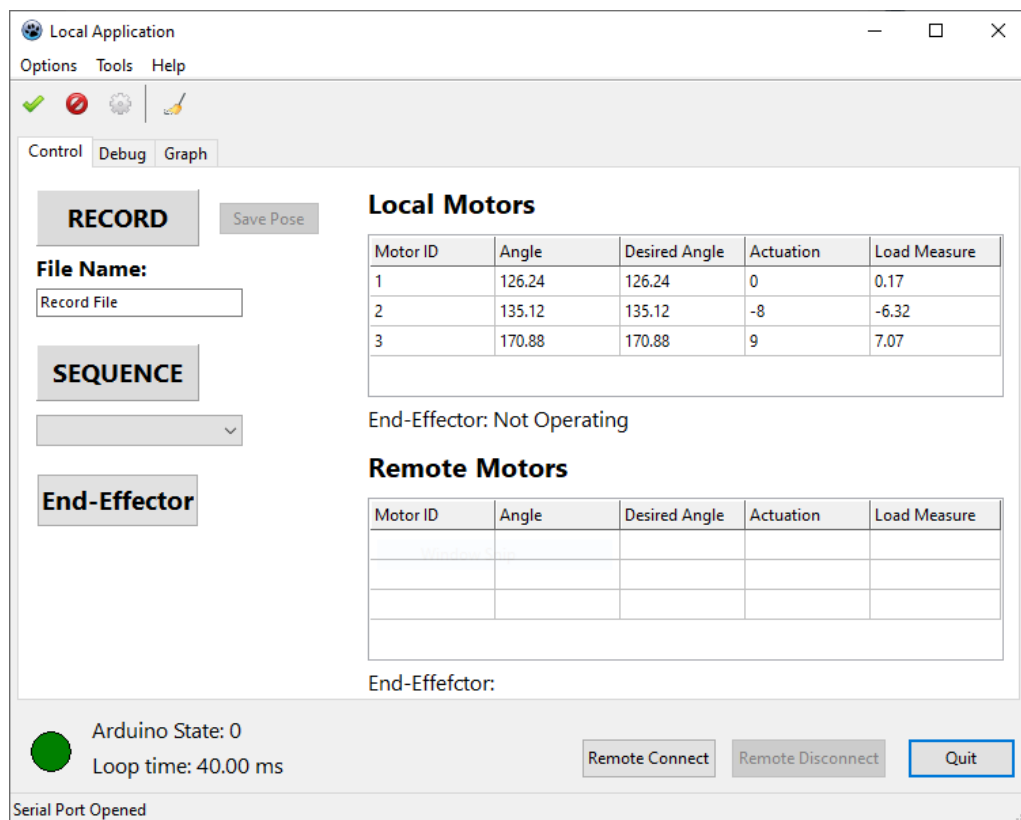


Figure 4.14: Local Application.

4.6.2.1 Control Page

The control page of the application is where the most important information is located along with the rest of the control options. Fig. 4.14 demonstrates this page and the components that it consists of.

In this page there are two grids on the right side with information about the current state of every joint of each robotic manipulator. The first grid has information about the local motors whereas the second one has information about the remote motors, as the headers indicate.

These grids display four values from each motor: the current angle in degrees, the desired angle in degrees, the current actuation on each joint and finally the force measured by the load cell relative to that motor's joint.

Furthermore, there is a label underneath each grid that informs whether the end-effector is currently operating or not.

On the left side, on the other hand, there are some controls available to the user. It is possible to record sequences, execute those sequences and operate the end-effector.

When the user intends to operate the end-effector, they simply need to click on the "End-Effector" button and the electromagnet will turn on in both manipulators. If the button is clicked again, then the electromagnet will be turned off. This button also changes colour depending on the state of the end-effector.

The procedures of recording and executing sequences will be explained later in this chapter.

4.6.2.2 Debug Page

The debug page, Fig. 4.15, consists of a Memo for debug messages, a Checklist of debug options, a label that indicates the state of the application and there is also the possibility to send a manual serial command.

The Memo present in this page is where every debug message is displayed on screen. Ranging from messages from the microcontroller to messages from the application itself. Using the debug options checklist on the right side, it is possible for the user to select which channels from the microcontroller should be displayed in the Memo.

Furthermore, there is also the possibility to display the raw input coming from the serial communication with the microcontroller using the checkbox in the bottom left corner.

Finally, the user can send a custom serial command to the microcontroller if it should be necessary at any given time.

4.6.2.3 Graph Page

The last page of the application, Fig. 4.16, is the Graph Page. This page is tasked with displaying a double y-axis Chart of Position vs Force vs Time and save data about every joint during a time period.

In this page the user has the option of choosing between the 3 possible joints using the Combo Box on the bottom left corner. After choosing the desired joint to be displayed in the Chart, the

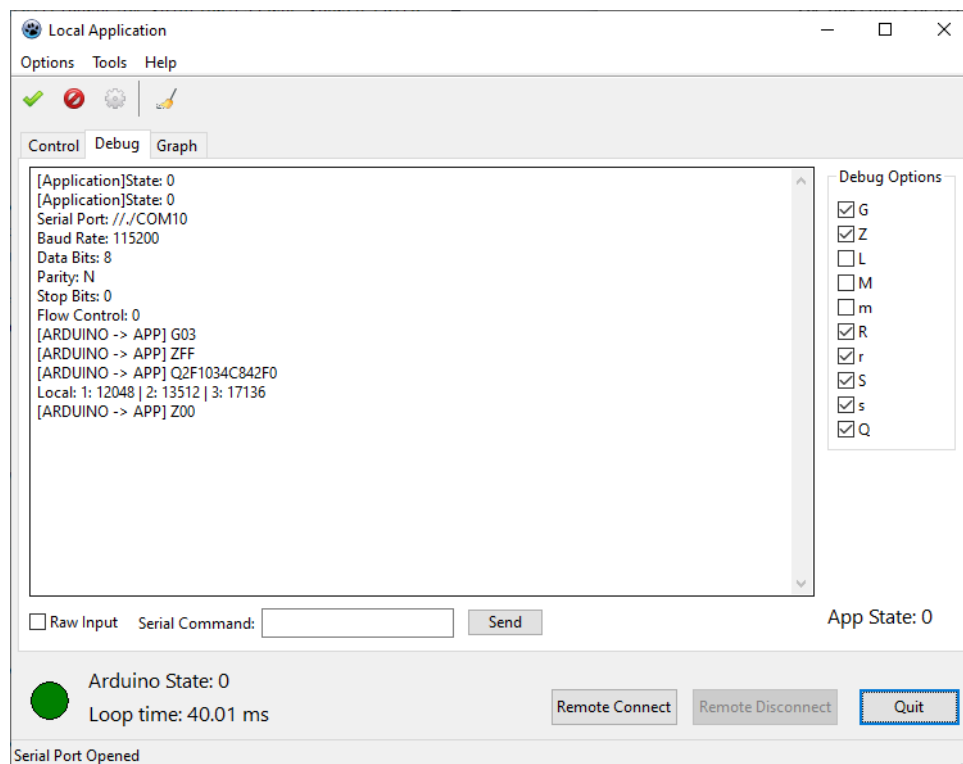


Figure 4.15: Local Application Window.



Figure 4.16: Local Application Window.

user simply needs to press the "Begin Plotting" button and the application will begin to plot the graph. In this plot will appear 4 series relating to the joint specified in the Combo Box: a series for the local position, one for the local force, one for the remote position and finally one for the remote force. This can prove useful as a form of debugging as it helps visualize the data in a time interval. It may also be of assistance when calculating the delay added by the communication platform.

The data saved during the time of sampling is done so in a Comma-Separated Values (CSV) type file. This type of file was chosen because it is simple to read and process using an external program like *Microsoft's Excel*³.

4.6.3 Remote Application

The application used to control the remote robotic arm is a simpler version of the local application, which was explained previously.

Instead of being composed of 3 pages like the local application it only has 2, the Control and the Debug pages. The major difference is in the Control page, Fig. 4.17, because the Debug page is exactly the same as the local application's Debug page.

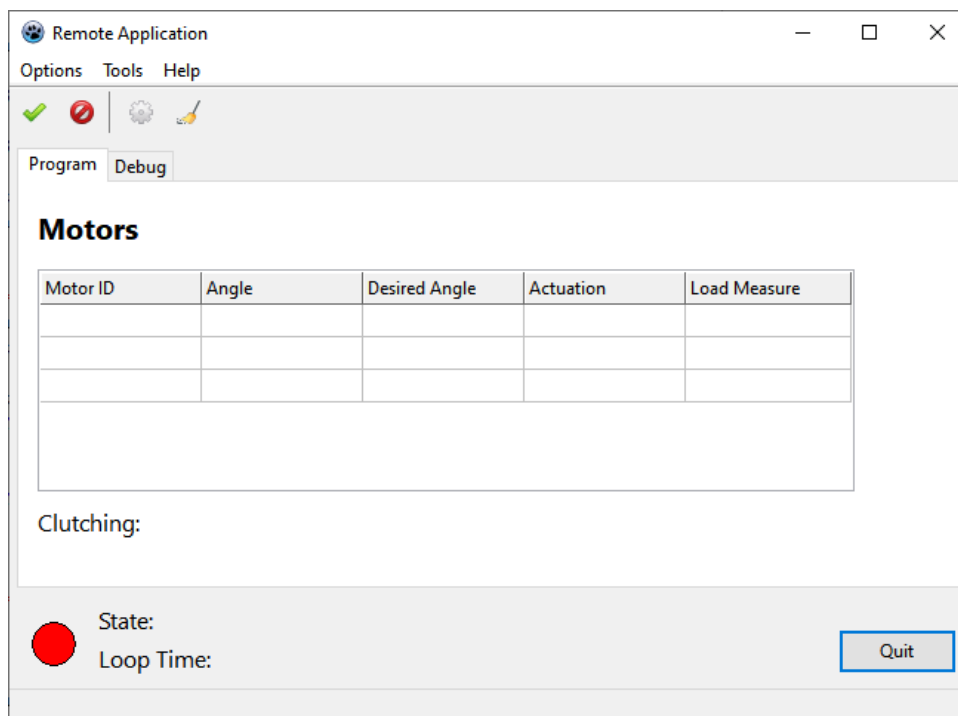


Figure 4.17: Remote Application Window.

This application is mainly used as a communication platform between both robotic manipulators and therefore does not need any control commands because they will be given by the user in

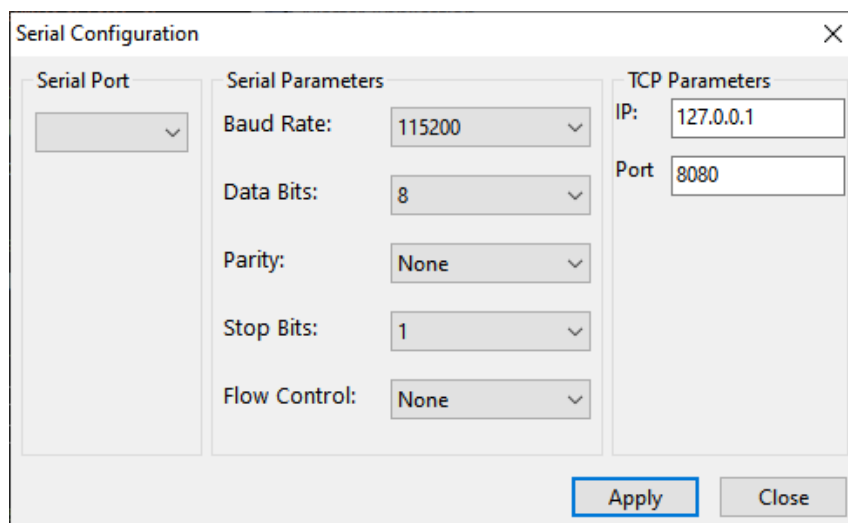
³Website: <https://www.microsoft.com/en-us/microsoft-365/excel?rtc=1>

the local application. Therefore, the Control page of this application is simply a display of information about the remote manipulator. It gives information about the state of each joint as well as whether the robotic arm's end-effector is operating or not.

The final difference is the fact that there are no buttons for establishing the connection to the local application. This is so because the remote application is setup as the server side of the connection and therefore only needs to listen for connections.

4.6.4 Configuration Window

In both applications there is a Configurations Window, Fig. 4.18, that pops up when the user either presses the Cog icon on the top bar or selects Tools->Configuration. This Configuration Window is used to setup the communications options for both Serial and TCP communications.



The screenshot shows a 'Serial Configuration' dialog box with a close button (X) in the top right corner. It is divided into three main sections: 'Serial Port', 'Serial Parameters', and 'TCP Parameters'. The 'Serial Port' section contains a dropdown menu. The 'Serial Parameters' section includes five settings: 'Baud Rate' (115200), 'Data Bits' (8), 'Parity' (None), 'Stop Bits' (1), and 'Flow Control' (None). The 'TCP Parameters' section includes 'IP' (127.0.0.1) and 'Port' (8080). At the bottom right, there are 'Apply' and 'Close' buttons.

Figure 4.18: Configuration Window.

The user can select which Serial Port the microcontroller is connected to as well as the parameters for that communication. Furthermore, it is possible to select the Port and the IP that the TCP connection will connect to (the remote application).

4.7 Recording and Sequencing

As mentioned before it is possible to record a sequence of positions and then replicate that sequence. This is done via the Control page, Fig. 4.14, with the buttons "RECORD" and "SEQUENCE".

When the user is ready to record a sequence they simply need to input the desired file name under "File Name:" and press the "RECORD" button. This will begin the recording procedure and so the smaller button "Save Pose" will be enabled.

After initializing this procedure, the user only needs to move the robotic arm to the desired pose and press the "Save Pose" button to record that position. This should be repeated until the

end of the desired sequence. To finalize the recording procedure the "RECORD" button should be pressed again which will save the file with all the positions recorded and will disable the "Save Pose" button once again.

The file with the sequence is saved in a directory called "recordings" which can be found in the directory of the application. The file is a simple text file that contains a header, indicating how many joints are needed for the sequence, followed by n lines of n poses and ends in a new line with the word "END", as seen in the example Fig. 4.19.

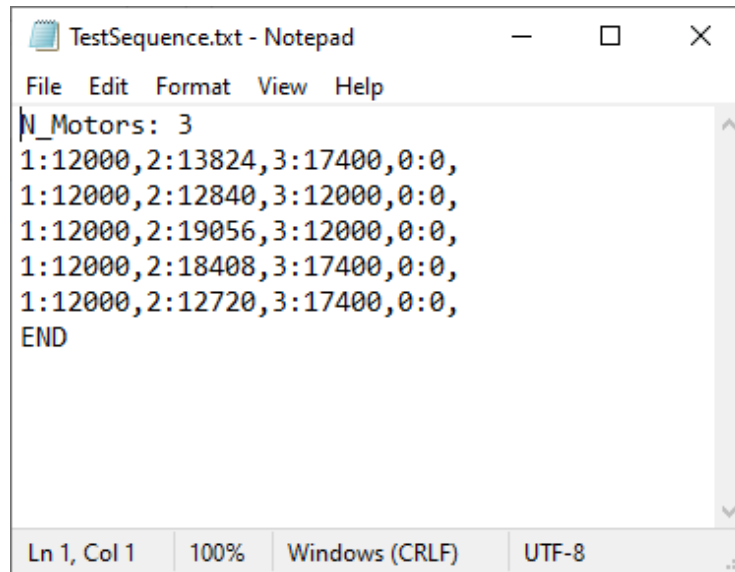


Figure 4.19: Recorded Sequence File Example.

Each pose line consists of a series of values separated by commas where each value is composed of the joint number and the current position of that joint, separated by a colon. If the joint number is 0, then the value relates to the end-effector and so a value of 1 means it is active and 0 means it is inactive.

This chapter had the aim of explaining the implementation of the software of this dissertation's project.

There was a description of the system architecture and all of its subsystems. It was also demonstrated how the communications between the various subsystems work along with the necessary types of controls used in the robot arms.

Finally a dissection of the GUIs was made followed by an explanation of the recording and sequencing procedures.

Chapter 5

Results

In the following chapter the results from several experiments will be displayed and analysed in order to validate the developed solution.

There were a total of 4 different experiments with the purpose of verifying and validating 4 different functionalities of the system: Manual Control, Remote Control, Haptic Feedback and Recording/Sequencing.

All of these experiments were done with both robotic manipulators connected to the same computer. The TCP/IP communication is used nonetheless but the delay between both manipulators is expected to be higher in a situation where both arms are connected to different machines.

5.1 Manual Control

The first experiment made with the complete system was the manual control of the local robotic arm. In this test, the user was simply asked to maneuver the manipulator, by applying forces to the end-effector, as desired to verify that the motion was working correctly and in a friendly manner. During the control of the arm, the positions of each joint along with the forces applied on each link were stored in a CSV file (using the Graphs Page of the GUI as mentioned in the previous chapter).

The figures [5.1](#), [5.2](#) and [5.3](#) demonstrate the changes in position and force over time for joints 1, 2 and 3, respectively.

Analyzing the first figure, [5.1](#), it is possible to see that once the force applied on the load cell reaches a certain value, around 75 grams-force or so, the angle of the joint begins to move in the positive direction. This happens for the first time at around 10000 ms after initializing the test.

Afterwards, once the force drops below approximately -75 grams-force the joint begins to rotate in the opposite direction until the force falls back to zero.

This graph shows that the manual control of the first joint does in fact work.

The same can be said for the other two joints because by analyzing the last two graphs, [5.2](#) and [5.3](#), the result is the same. When the force passes a certain value, the joint reacts and moves in the expected direction.

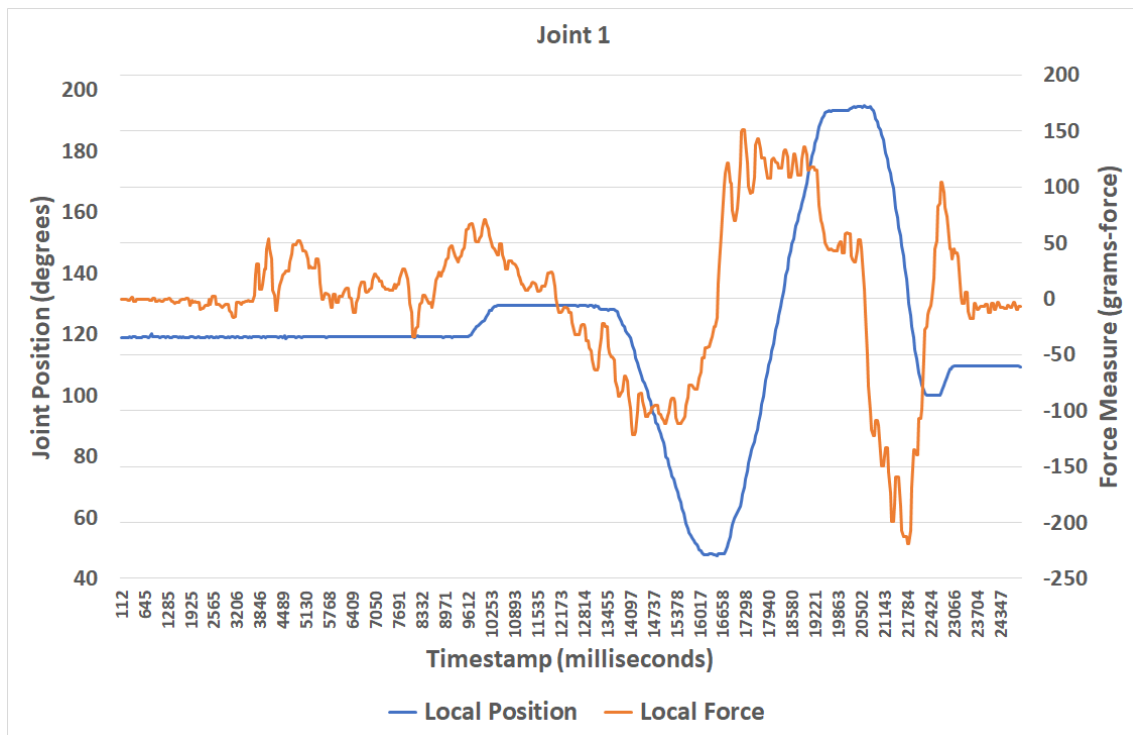


Figure 5.1: Joint 1 Manual Control Position vs Force Plot.

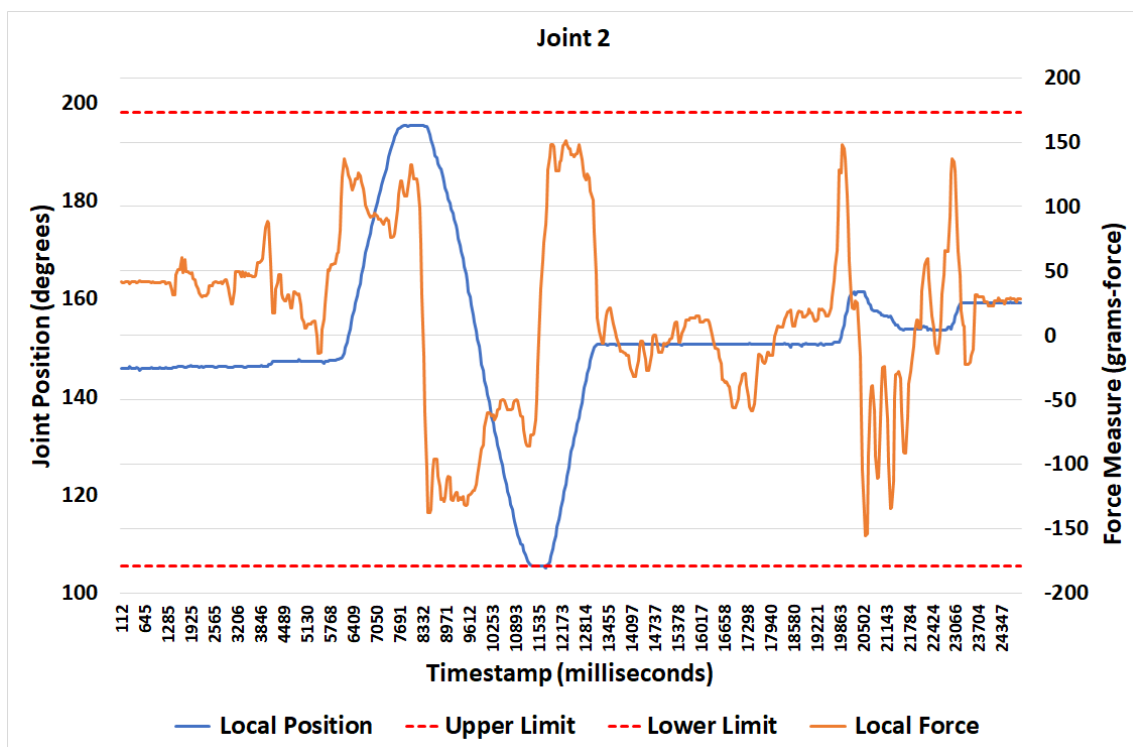


Figure 5.2: Joint 2 Manual Control Position vs Force Plot.

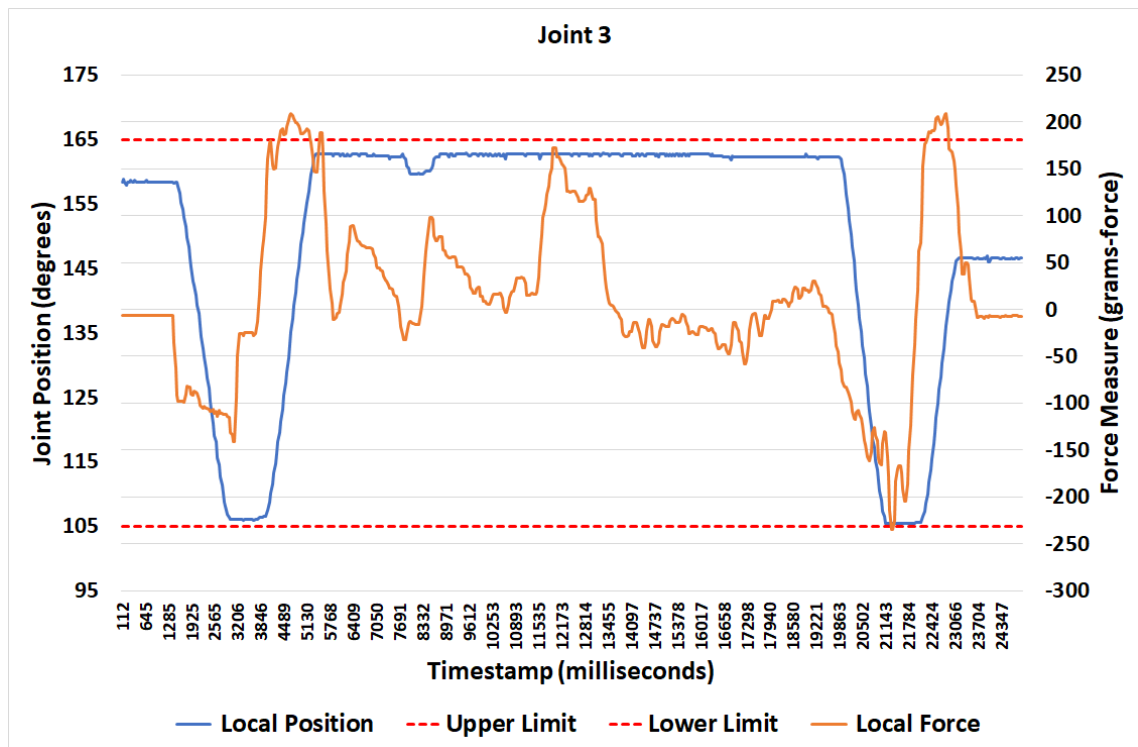


Figure 5.3: Joint 3 Manual Control Position vs Force Plot.

The only apparent problem happens in the third joint in the middle of the experiment. It is noticeable that there is a moment, approximately at 11000ms, when the force measured in the link is substantial, which should make the joint actuate in the positive direction. This does not happen and the reason why is that the upper limit for the motion of this joint is approximately 165 degrees. Hence the physical limit of the manipulator is met and the joint does not actuate further.

5.2 Remote Control

The test for the remote control was done at the same time as the manual control experiment, with the added TCP connection between both applications. This enabled the remote control of the remote robotic arm. To do this, the user simply needed to click the button to establish the communication as mentioned in the previous chapter.

It is easy to see that the positions recorded for each local joint are the same as in the previous experiment. However, to showcase that there is a remote control taking place, the following graphs will display 2 series, one for the local joint position and another for the remote joint position.

As done before, each figure that follows refers to one of the three available joints: figures 5.4, 5.5 and 5.6 refer to joints 1, 2 and 3, respectively. Despite being demonstrated in different plots, the experiment was done with all joints working simultaneously.

The objective of this remote control experiment is to have the remote robotic arm replicate the movements and positions that the local manipulator has. As we can see, in all three joints, the local

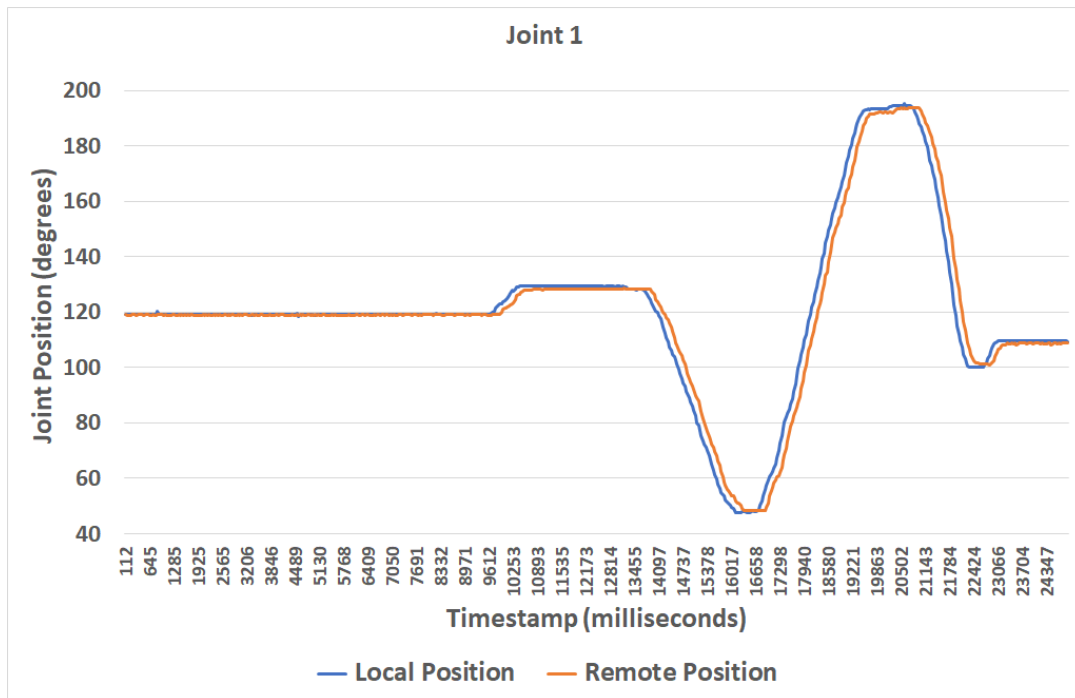


Figure 5.4: Joint 1 Remote Control Position vs Position Plot.

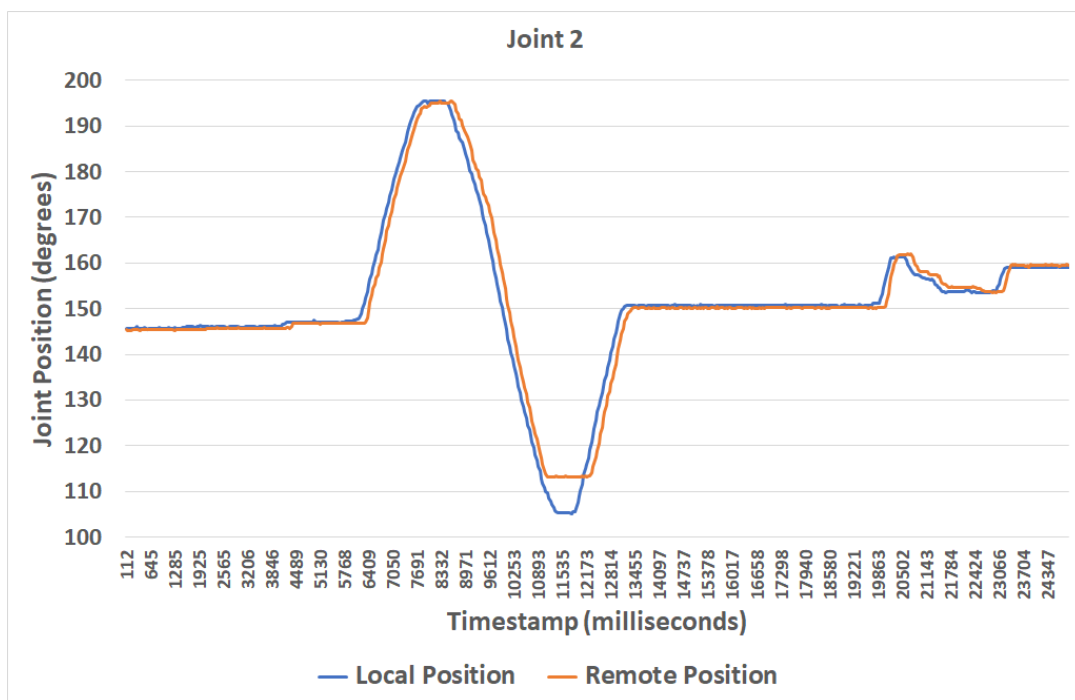


Figure 5.5: Joint 2 Remote Control Position vs Position Plot.

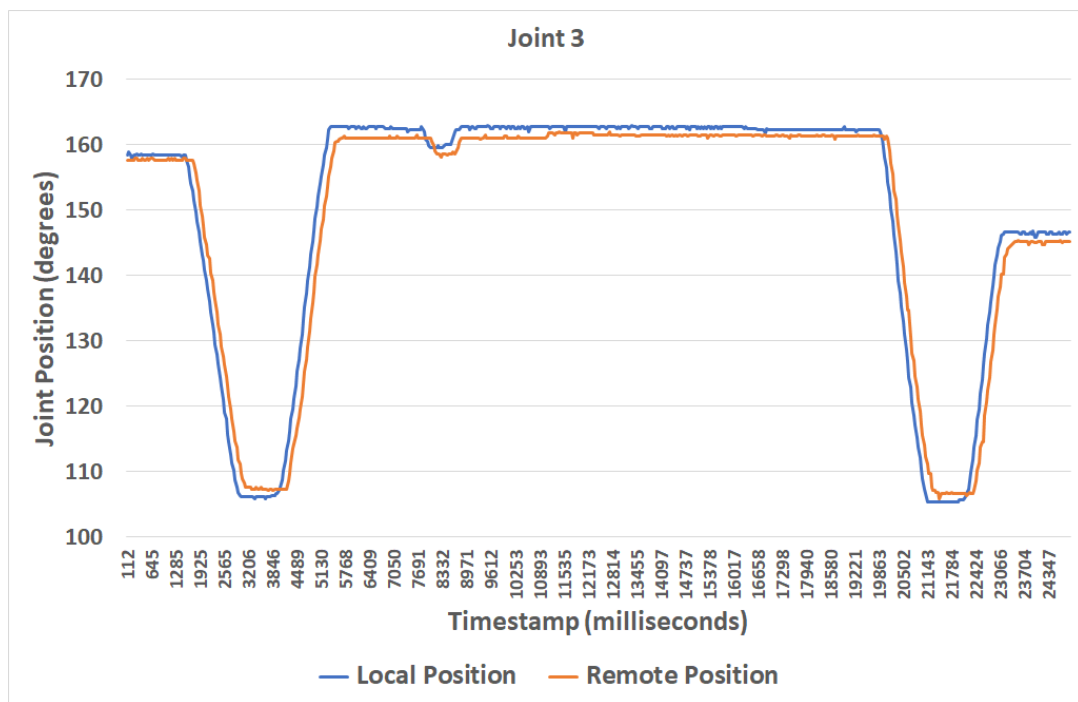


Figure 5.6: Joint 3 Remote Control Position vs Position Plot.

positions are indeed tracked by the remote manipulator because the remote plots are replicating the local ones. This has a bit of a delay, which is almost unnoticeable since a local communication is being used.

5.3 Haptic Feedback

The third test was regarding the haptic force feedback developed for the local manipulator. In this test, while operating the system in remote control, a force was applied on the remote robotic manipulator opposing its movement. This was done for all three joints in order to test each and every one of them.

As a way to demonstrate this functionality, three plots will be displayed and analysed (one for each joint). In these plots there will be three series displayed: one for the position of the joint in the local arm, another for the force being applied on the local load cell to move the robotic manipulator and finally one for the force being applied on the remote load cell relative to that joint.

The purpose of showing these three series is to demonstrate that when there is a force strong enough applied in the opposite direction of the movement, the joint does not continue moving even though the user keeps forcing the joint. This indicates that there is a constraint in that movement and therefore the user becomes more aware of the remote environment.

Figures 5.7, 5.8 and 5.9 refer to joints 1, 2 and 3, respectively.

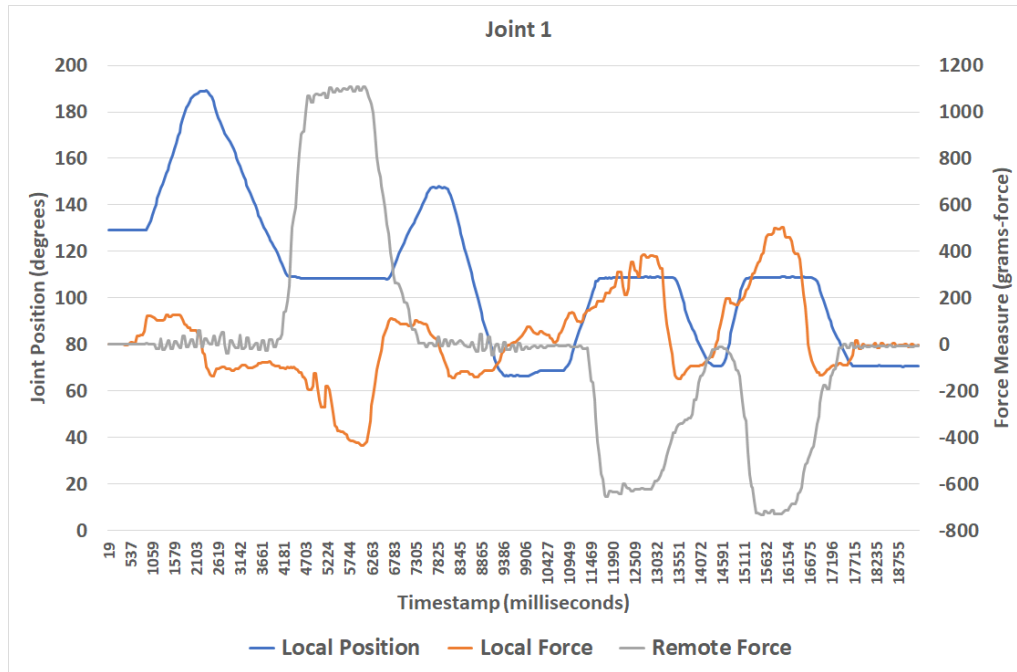


Figure 5.7: Joint 1 Haptic Feedback Plot.

Analyzing the first graph, Fig. 5.7, which refers to the joint of the base, the first occurrence of haptic feedback can be seen at around 4200 milliseconds. At this time the remote force begins to rise while the joint is moving in the negative direction. This can be seen by the blue line, the joint position, descending as well as the orange line, the force being applied on that joint, being negative. As it is possible to see, the blue line stabilizes at around 110 degrees even though the local force is strong enough for it to move in normal conditions. This reoccurs two more times at 11500 and 15100 milliseconds.

The same thing happens for the remaining two joints. For the second joint, Fig. 5.8, it happens at 2500, 8400 and 18600 milliseconds. Meanwhile, for the third joint, Fig. 5.9 this occurs twice at 3900 and 11000 milliseconds.

This demonstrates that the haptic feedback is in fact present on this system and works how it is supposed to.

5.4 Recording and Sequence

The final experiment done for this system was to test the recording and the sequencing aspects of the project.

First, a sequence was recorded in the local manipulator using the manual control. Fig. 5.10 shows the file that contains all the recorded points that compose the sequence used in this test.

Then, the user simply selected the desired sequence and pressed the "Sequence" button to begin the procedure.

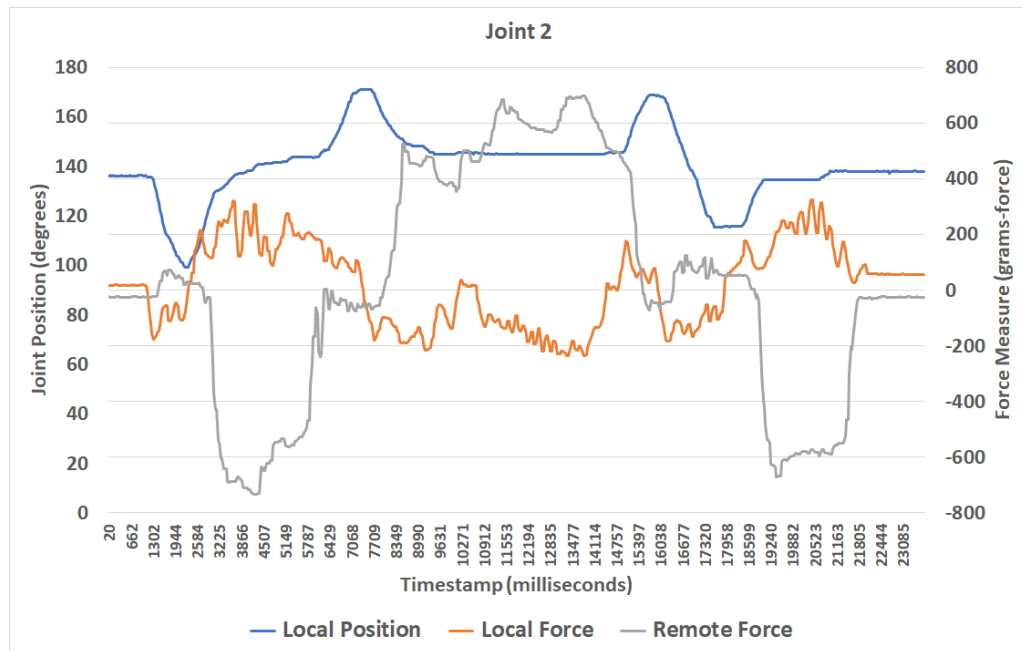


Figure 5.8: Joint 2 Haptic Feedback Plot.

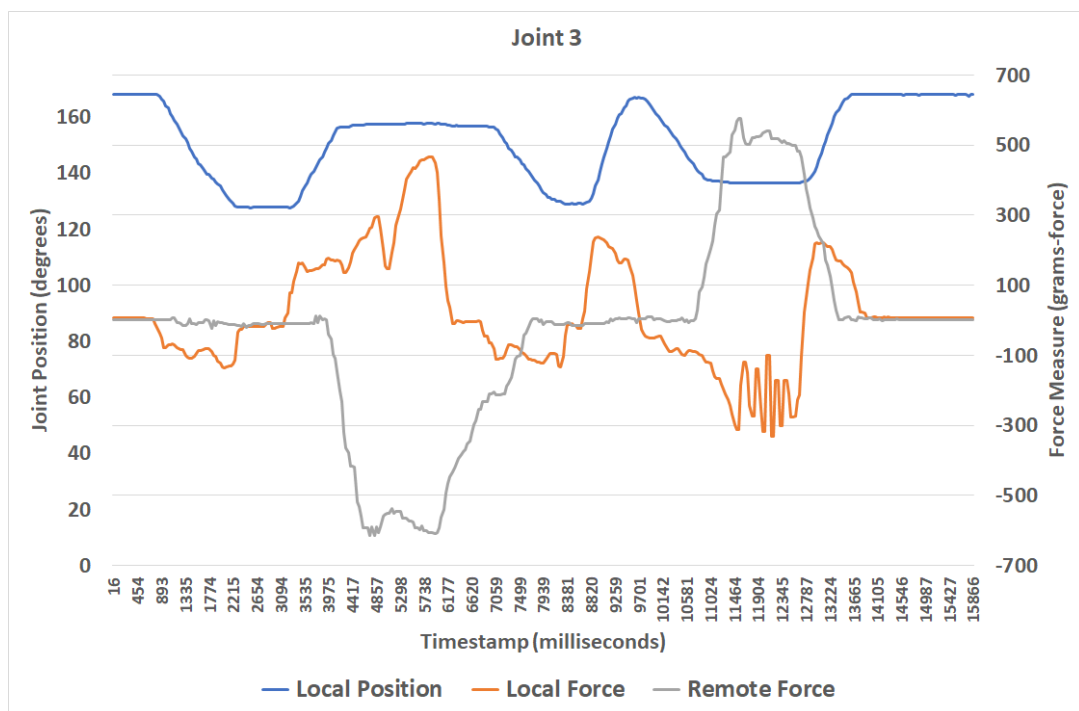
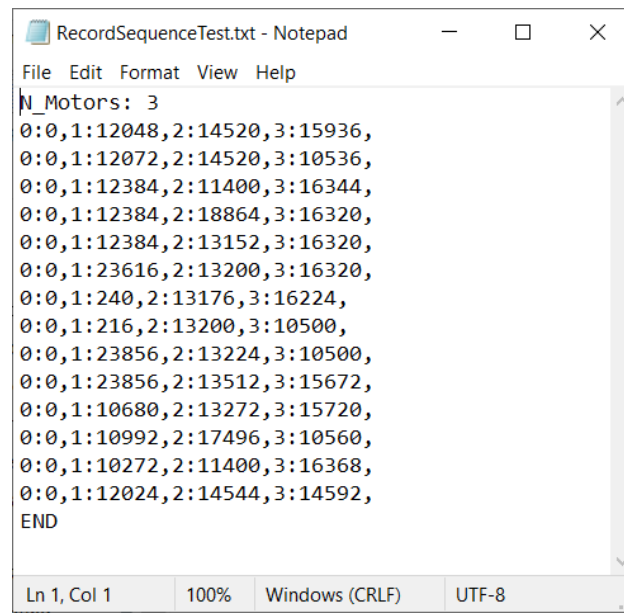


Figure 5.9: Joint 3 Haptic Feedback Plot.



```

RecordSequenceTest.txt - Notepad
File Edit Format View Help
N_Motors: 3
0:0,1:12048,2:14520,3:15936,
0:0,1:12072,2:14520,3:10536,
0:0,1:12384,2:11400,3:16344,
0:0,1:12384,2:18864,3:16320,
0:0,1:12384,2:13152,3:16320,
0:0,1:23616,2:13200,3:16320,
0:0,1:240,2:13176,3:16224,
0:0,1:216,2:13200,3:10500,
0:0,1:23856,2:13224,3:10500,
0:0,1:23856,2:13512,3:15672,
0:0,1:10680,2:13272,3:15720,
0:0,1:10992,2:17496,3:10560,
0:0,1:10272,2:11400,3:16368,
0:0,1:12024,2:14544,3:14592,
END
Ln 1, Col 1    100%    Windows (CRLF)    UTF-8

```

Figure 5.10: Sequence File used for the test.

To graphically demonstrate how the system behaves when requested to do a sequence, the following graphs will contain 3 series each: one for the desired position for that particular joint, another for the local current position of that joint and finally one for the remote current position of that joint. This helps to show how each joint (both local and remote) operates during a sequence.

Figures 5.11, 5.12 and 5.13 refer to joints 1, 2 and 3 respectively.

As it is possible to see from the figures, all joints, for both the local and remote manipulators, were capable of following their references. This, therefore, validates the procedure of recording and sequencing for this system.

5.5 Conclusion

In this chapter the objective was to provide a visual feedback of the results acquired from four different experiments. This was done so using graphs that display several information about each joint of the robotic manipulators.

With the tests, it was possible to validate four different aspects of the system:

1. the fact that the local manipulator can indeed be controlled and manipulated by the user using force;
2. the system was capable of handling remote control. In other words, the remote manipulator can be controlled remotely by the local manipulator;
3. the system is equipped with haptic feedback using force generated by torque from the motors;

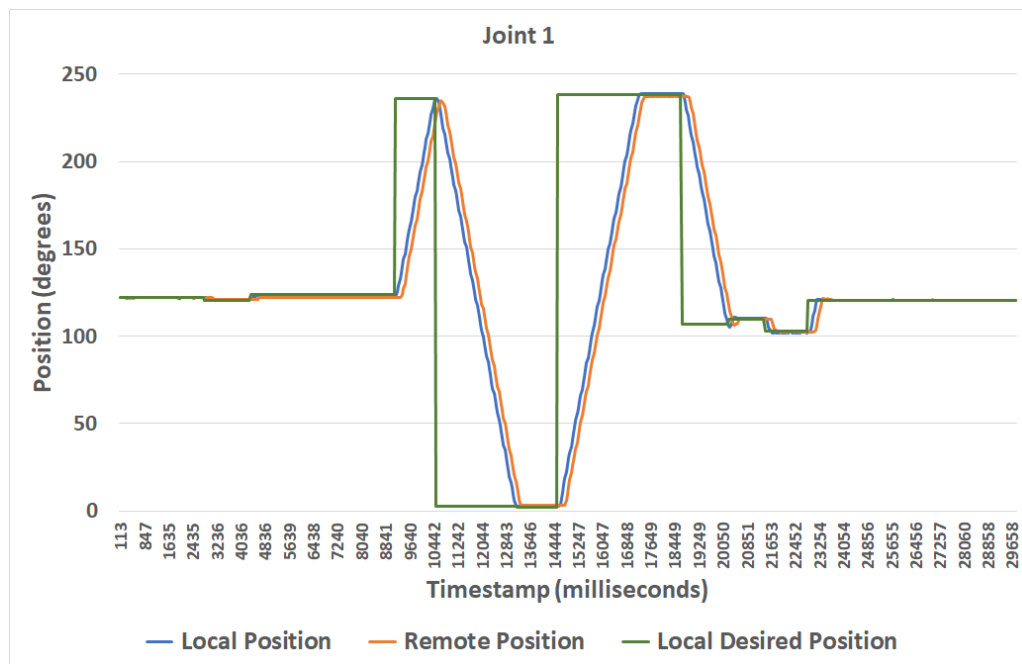


Figure 5.11: Joint 1 Sequencing Plot.

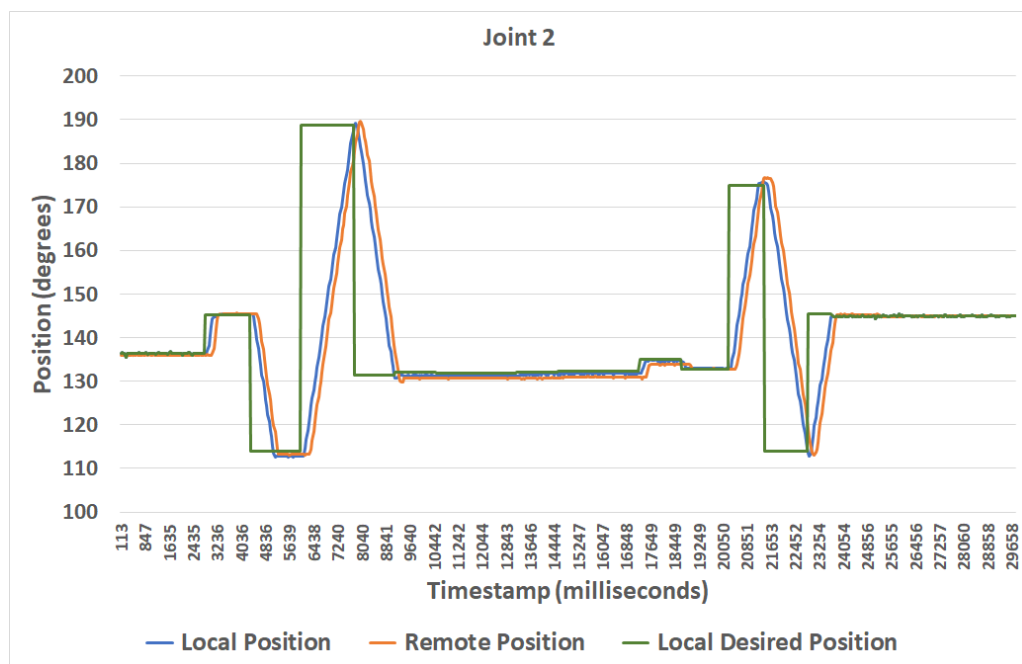


Figure 5.12: Joint 2 Sequencing Plot.

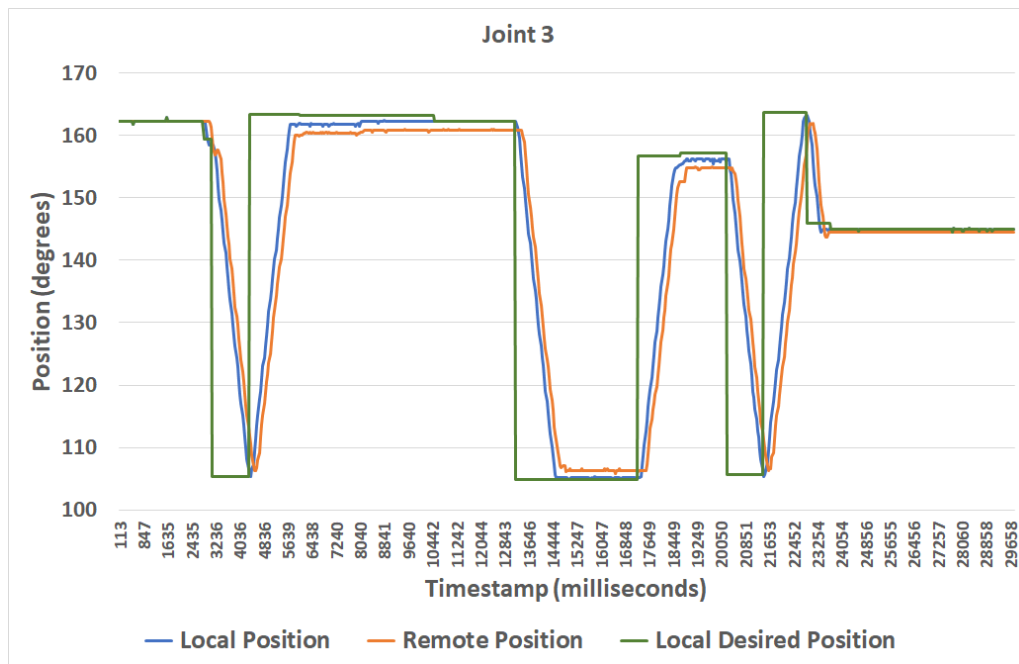


Figure 5.13: Joint 3 Sequencing Plot.

4. and finally, it is also possible to record and execute a sequence composed of several discrete positions.

Chapter 6

Conclusion and Future Work

This following chapter is the final chapter of this dissertation. Therefore, it is reserved for the final conclusions along with the proposed future work that should be done for this dissertation's project.

6.1 Conclusion

This master thesis' project had the objective of creating a system used for teaching remote robotic manipulators. Not only this but the device should be further equipped with haptic force feedback as a way to further help the user when maneuvering the system.

The first steps of the development of the project were based around building the manipulator. The base model used was a palletizer configuration by the name *EEZYbotArm MK2* which was provided by the user *daGHIZmo* on *Thingiverse*.

After acquiring the base model of the manipulator to be used, began the projections of the modifications, using *Fusion 360*, that needed to take place to convert the manipulator into it's desired form. This was an iterative procedure of modeling, printing, and validating the pieces projected.

Once every piece had been projected and verified, what followed was building both manipulators with the servo motors, load cells and electromagnets.

Having the manipulators ready for operation, the development of their control began. From the manual control of the local manipulator to the haptic force feedback in the local arm.

During all this previous development, the GUIs were being developed in parallel using the application Lazarus.

After finishing all the subsystems, manipulators, controls and GUIs, the system was verified with several tests and experiments designed to check all procedures.

In this final testing phase, the system went through several tweaks and changes needed to further improve the project.

Overall the product developed during this dissertation works well and as expected. The manual control of the local manipulator functions exactly how it was supposed to, making it very user-friendly. Furthermore, the haptic feedback when operating in remote control is working very well making the experience of this type of control much more interesting and intuitive to use.

The GUIs, in spite of not being very pretty, are organized to give a good experience to the user and help better control the manipulator.

The modified robotic manipulator is very inexpensive and could potentially be used for educational purposes. Table 6.1 contains a list of the components needed to build one robotic arm, excluding the hardware, like bolts, nuts and bearings, and the 3D printed parts.

Component	Quantity	Cost (September 2020)
LX16-A Servo Motors	3	$(15.16\text{€} \times 3) = 45.48\text{€}$
Hiwonder BusLinker Debug Board	1	10.82€
TAL220 Load Cell	3	$(8.66\text{€} \times 3) = 25.98\text{€}$
HX711 ADC	3	$(4.95\text{€} \times 3) = 14.85\text{€}$
Arduino Mega 2560	1	21.53€
Electromagnet	1	12.60€
DC/DC Step-Down Converter	1	3.90€
	Total	135.16€

Table 6.1: Robotic Manipulator Components List and Cost¹.

As seen from the previous table of prices, the total cost of the manipulator is around 135 euros without taking into account the hardware, bearings and 3D printed parts. Considering these last components cost around 50€ then the total cost of the robotic arm would be of close to 185€. This value is a low budget for the capabilities of this manipulator.

As a term of reference, there currently is a robotic manipulator for teaching called *DOBOT Magician Lite*², created by the company *DOBOT*, which is described as a "a multi-functional lightweight intelligent robotic arm". It has a price of 1,099.00 USD as of September 2020 and comes with many addons like multiple end tools, Bluetooth connection, graphic programming and many others.

6.2 Future Work

The product developed during this project can be improved in some fields, mainly the application and the remote control. The system does work and has achieved all the requirements set for it, however, the applications could be improved mainly in the visual aspect but also in some logic aspects.

A big improvement that should be considered is in the remote control mechanic. As of right now, the system uses follow angle only for the remote control and a good improvement would be

¹Note: All prices were acquired from either <https://www.botnroll.com/pt/>, <https://www.ptrobotics.com/> or <https://www.amazon.co.uk/> on September 10th of 2020.

²Website: <https://tinyurl.com/y5duxymy>

to change this to use both the angle and the current actuation of the local manipulator. This would provide a better and smoother remote control.

One quality of life improvement that could be prominent to the better and easier use of the robot arm is to include a button that controls the end-effector. This button should be placed somewhere near the end-effector to facilitate it's reach.

It is also pertinent that the system should be tested in different machines both in a local network as well as over the Internet as a way to verify the effects and influence of the delay between both subsystems.

Appendix B

Channels

Channel	Size (Nibbles)	Description
G	2	Good-to-go flag and number of motors used in current configuration.
Z	2	Current state of the microcontroller.
L	8	Loop time.
M	22	Information about a certain joint.
m	2	State of the end-effector.
R	2	Confirm state of recording.
r	6	Send joint position for recording.
S	2	Confirm state of sequencing flag.
s	2	Request new sequence position.
Q	12	Message with the home positions of the joints

Table B.1: List of used Channels from Local Microcontroller to Local Computer.

Channel	Size (Nibbles)	Description
R	8	Send recording state flag.
r	8	Send information about one joint for recording.
S	8	Send sequencing state flag.
s	8	Send next desired joint position. Also used in sequences to set position.
T	8	Operate end-effector's tool.
V	16	Force sensor from remote manipulator.

Table B.2: List of used Channels from Local Computer to Local Microcontroller.

Channel	Size (Nibbles)	Description
Q	8	Send recording state flag.
M	8	Send information about one joint for recording.
m	8	Send sequencing state flag.

Table B.3: List of used Channels from Local Computer to Remote Computer.

Channel	Size (Nibbles)	Description
G	2	Good-to-go flag and number of motors used in current configuration.
Z	2	Current state of the microcontroller.
L	8	Loop time.
M	22	Information about a certain joint.
m	2	State of the end-effector.
Q	12	Message with the home positions of the joints

Table B.4: List of used Channels from Remote Microcontroller to Remote Computer.

Channel	Size (Nibbles)	Description
s	8	Send next desired joint position. Also used in sequences to set position.
T	8	Operate end-effector's tool.

Table B.5: List of used Channels from Remote Computer to Remote Microcontroller.

Channel	Size (Nibbles)	Description
Q	8	Send recording state flag.
M	8	Send information about one joint for recording.
m	8	Send sequencing state flag.

Table B.6: List of used Channels from Remote Computer to Local Computer.

References

- [1] T.R. Kurfess and W.H.C. Bassetti. *Robotics and Automation Handbook*. Taylor & Francis, 2005.
- [2] Mark W. Spong, Seth. Hutchinson, and M. (Mathukumalli) Vidyasagar. *Robot modeling and control*. John Wiley & Sons, 2006.
- [3] Lorenzo Sciacivco and Bruno Siciliano. *Modelling and Control of Robot Manipulators*. Advanced Textbooks in Control and Signal Processing. Springer London, London, 2000.
- [4] Nicolas Turro, Oussama Khatib, and Eve Coste-Maniere. Haptically augmented teleoperation. In *Proc. 2001 ICRA. IEEE Int. Conf. Robot. Autom. (Cat. No.01CH37164)*, volume 1, pages 386–392. IEEE, 2001.
- [5] Peter F. Hokayem and Mark W. Spong. Bilateral teleoperation: An historical survey. *Automatica*, 42(12):2035–2057, dec 2006.
- [6] T.M. Lam, H.W. Boschloo, Max Mulder, and M.M. van Paassen. Artificial Force Field for Haptic Feedback in UAV Teleoperation. *IEEE Trans. Syst. Man, Cybern. - Part A Syst. Humans*, 39(6):1316–1330, nov 2009.
- [7] Sangeeta Senapati and Arnold P. Advincula. Surgical techniques: robot-assisted laparoscopic myomectomy with the da Vinci® surgical system. *J. Robot. Surg.*, 1(1):69–74, mar 2007.
- [8] M. Sreelakshmi and T.D. Subash. Haptic Technology: A comprehensive review on its applications and future prospects. *Mater. Today Proc.*, 4(2):4182–4187, 2017.
- [9] Sang-Rock Lee, Seung-Hyun Choi, Jong-Seok Oh, and Seung-Bok Choi. A novel approach for prediction of a repulsive force in a haptic manipulator: experimental verification with different trajectories. *Smart Mater. Struct.*, 24(2):025017, feb 2015.
- [10] Sadao Omata and Yoshikazu Terunuma. New tactile sensor like the human hand and its applications. *Sensors Actuators A Phys.*, 35(1):9–15, oct 1992.
- [11] Ramona Fagiani, Francesco Massi, Eric Chatelet, Yves Berthier, and Adnan Akay. Tactile perception by friction induced vibrations. *Tribol. Int.*, 44(10):1100–1110, sep 2011.
- [12] Jong-Seok Oh, Young-Min Han, Sang-Rock Lee, and Seung-Bok Choi. A 4-DOF haptic master using ER fluid for minimally invasive surgery system application. *Smart Mater. Struct.*, 22(4):045004, apr 2013.
- [13] Jong-Seok Oh, Seung-Hyun Choi, and Seung-Bok Choi. Control of repulsive force in a virtual environment using an electrorheological haptic master for a surgical robot application. *Smart Mater. Struct.*, 23(1):015010, jan 2014.

- [14] Christopher E. Wong and Allison M. Okamura. The snaptic paddle: A modular haptic device. In *Proc. - 1st Jt. Eurohaptics Conf. Symp. Haptic Interfaces Virtual Environ. Teleoperator Syst. World Haptics Conf. WHC 2005*, pages 537–538. Institute of Electrical and Electronics Engineers Inc., 2005.
- [15] Kourosh Zareinia, Yaser Maddahi, Canaan Ng, Nariman Sepehri, and Garnette R. Sutherland. Performance evaluation of haptic hand-controllers in a robot-assisted surgical system. *Int. J. Med. Robot. Comput. Assist. Surg.*, 11(4):486–501, dec 2015.
- [16] Craig Carignan, Jonathan Tang, and Stephen Roderick. Development of an exoskeleton haptic interface for virtual task training. In *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pages 3697–3702. IEEE, oct 2009.
- [17] Jacques Foottit, Dave Brown, Stefan Marks, and Andy M. Connor. A Wearable Haptic Game Controller. *Int. J. Game Theory Technol.*, 2(1):1–19, mar 2016.
- [18] Ivan Muller, Renato de Brito, Carlos Pereira, and Valner Brusamarello. Load cells in force sensing analysis – theory and a novel application. *IEEE Instrum. Meas. Mag.*, 13(1):15–19, feb 2010.